IBM

Application System/400™

SC21-8226-0

Machine Interface Functional Reference

IBM

Application System/400™

SC21-8226-0

Machine Interface Functional Reference

First Edition (August 1990)

The functions described in this publication apply to the IBM AS/400 machine interface.

Order publications through your IBM representative or the IBM branch serving your locality. Publications are not stocked at the address given below.

A form for readers' comments is provided at the back of this publication. If the form has been removed, you may address your comments to:

IBM Corporation, Department 245, 3605 North Highway 52 and 37th Street NW, Rochester, MN 55901-9986 USA.

Special Notices

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates.

Any reference to an IBM licensed program or other IBM product in this publication is not intended to state or imply that only IBM's program or other product may be used.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Commercial Relations, IBM Corporation, Purchase, NY 10577.

The following terms, denoted by an asterisk (*) in this publication, are trademarks of the IBM Corporation in the United States and/or other countries:

Application System/400 AS/400 IBM
400

This publication could contain technical inaccuracies or typographical errors.

The information herein is subject to change.

About This Manual

The information contained in *AS/400 Machine Interface Functional Reference* has not been submitted to any formal IBM test and is distributed on an 'as is' basis without any warranty either expressed or implied. This manual is written for release 3 of AS/400 Vertical Licensed Integrated Code (VLIC) and may not discuss all the functions available on your AS/400 system.

The *AS/400 Machine Interface Functional Reference* is a new manual.

The *AS/400 Machine Interface Functional Reference* defines the AS/400 Machine Interface to instructions, exceptions, and events.

This manual may refer to products that are announced but are not yet available.

Who Should Use This Manual

This manual is intended for knowledgeable system programmers having substantial experience on AS/400 computer systems.

What You Should Know

The reader should know one more high level languages, assembly languages of other computers, and understand instruction set architectures. The reader would do well to study capability-based computer architectures.

The reader should be familiar with AS/400 objects and their intended use.

How This Manual Is Organized

The *AS/400 Machine Interface Functional Reference* is organized into three parts:

1. Basic Function Instructions

   These instructions provide a basic set of functions commonly needed by most programs executing on the machine. Because of the basic nature of these instructions, they tend to experience less change in their operation in different machine implementations than the extended function instructions.

2. Extended Function Instructions

   These instructions provide an extended set of functions which can be used to control and monitor the operation of the machine. Because of the more complicated nature of these instructions, they are more exposed to changes in their operation in different machine implementations than the basic function instructions.

3. Instruction Support Interfaces

   This part of the document defines those portions of the Machine Interface which provide support for functions or data used pervasively on all instructions. It discusses the exceptions and program objects which can be operated on by instructions.

Contents

Basic Function Instructions

Extended Function Instructions

Instruction support interfaces

Basic Function Instructions

These instructions provide a basic set of functions commonly needed by most programs executing on the machine. Because of the basic nature of these instructions, they tend to experience less change in their operation in different machine implementations than the extended function instructions. Therefore, it is recommended that, where possible, programs be limited to using just these basic function instructions to minimize the impacts which can arise in moving to different machine implementations.

Chapter 1. Computation and Branching Instructions

This chapter describes all the instructions used for computation and branching. These instructions are arranged in alphabetic order. For an alphabetic summary of all the instructions, see Appendix A, "Instruction Summary."

1.1 Add Logical Character (ADDLC)

Op Code (Hex)	Operand 1	Operand 2	Operand 3
1023	Sum	Addend 1	Addend 2

Operand 1: Character variable scalar (fixed-length).

Operand 2: Character scalar (fixed-length).

Operand 3: Character scalar (fixed-length).

Optional Forms

Mnemonic	Op Code (Hex)	Form Type
ADDLCS	1123	Short
ADDLCI	1823	Indicator
ADDLCIS	1923	Indicator, Short
ADDLCB	1C23	Branch
ADDLCBS	1D23	Branch, Short

If the short instruction option is indicated in the op code, operand 1 is used as the first and second operational operands (receiver and first source operand). Operand 2 is used as the third operational operand (second source operand).

Extender: Branch or indicator options.

If the branch or indicator option is specified in the op code, the extender field must be present along with one to four branch targets (for branch options) or one to four indicator operands (for indicator options). The branch or indicator operands immediately follow the last operand listed above. See Chapter 1. "Introduction" for the encoding of the extender field and the allowed syntax of the branch and indicator operands.

Description: The unsigned binary value of the addend 1 operand is added to the unsigned binary value of the addend 2 operand and the result is placed in the sum operand.

Operands 1, 2, and 3 must be the same length; otherwise, the Create Program instruction signals an invalid length exception. The length can be a maximum of 256 bytes.

The addition operation is performed according to the rules of algebra. The result value is then placed (left-adjusted) in the receiver operand with truncating or padding taking place on the right. The pad value used in this instruction is a byte value of hex 00.

If operands overlap but do not share all of the same bytes, results of operations performed on these operands are not predictable. If overlapped operands share all of the same bytes, the results are predictable when direct addressing is used. If indirect addressing is used (that is, based operands, parameters, strings with variable lengths, and arrays with variable subscripts), the results are not always predictable.

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for this instruction.

Resultant Conditions: The logical sum of the character scalar operands is zero with no carry out of the leftmost bit position, not-zero with no carry, zero with carry, or not-zero with carry.

Exceptions

Exception		Operands 1	2	3	Other
06	Addressing				
	01 Spacing addressing violation	X	X	X	
	02 Boundary alignment	X	X	X	
	03 Range	X	X	X	
	06 Optimized addressability invalid	X	X	X	
08	Argument/parameter				
	01 Parameter reference violation	X	X	X	
10	Damage encountered				
	04 System object damage state	X	X	X	X
	44 Partial system object damage	X	X	X	X
1C	Machine-dependent exception				
	03 Machine storage limit exceeded				X
20	Machine support				
	02 Machine check				X
	03 Function check				X
22	Object access				
	01 Object not found	X	X	X	
	02 Object destroyed	X	X	X	
	03 Object suspended	X	X	X	
24	Pointer specification				
	01 Pointer does not exist	X	X	X	
	02 Pointer type invalid	X	X	X	
2A	Program creation				
	05 Invalid op-code extender field				X
	06 Invalid operand type	X	X	X	
	07 Invalid operand attribute	X	X	X	

Exception		Operands 1	2	3	Other
	08 Invalid operand value range	X	X	X	
	09 Invalid branch target operand				X
	0A Invalid operand length	X	X	X	
	0C Invalid operand odt reference	X	X	X	
	0D Reserved bits are not zero	X	X	X	X
2C	Program execution				
	04 Invalid branch target				X
2E	Resource control limit				
	01 user profile storage limit exceeded				X
36	Space management				
	01 space extension/truncation				X

1.2 Add Numeric (ADDN)

Op Code (Hex)	Operand 1	Operand 2	Operand 3
1043	Sum	Addend	Augend

Operand 1: Numeric variable scalar

Operand 2: Numeric scalar

Operand 3: Numeric scalar

Optional Forms

Mnemonic	Op Code (Hex)	Form Type
ADDNS	1143	Short
ADDNR	1243	Round
ADDNSR	1343	Short, Round
ADDNI	1843	Indicator
ADDNIS	1943	Indicator, Short
ADDNIR	1A43	Indicator, Round
ADDNISR	1B43	Indicator, Short, Round
ADDNB	1C43	Branch
ADDNBS	1D43	Branch, Short
ADDNBR	1E43	Branch, Round
ADDNBSR	1F43	Branch, Short, Round

Caution:

If operands overlap but do not share all of the same bytes, results of operations performed on these operands are not predictable. If overlapped operands share all of the same bytes, the results are predictable when direct addressing is used. If indirect addressing is used (that is, based operands, parameters, strings with variable lengths, and arrays with variable subscripts), the results are not always predictable.

The short form of the ADD NUMERIC instruction accepts two operands. The first operand is the Addend and Sum. The Addend is replaced by the Sum after the instruction completes. The second operand is the Augend.

Extender: Branch or indicator options.

If the branch or indicator option is specified in the op code, the extender field must be present along with one to four branch targets (for branch options) or one to four indicator operands (for indicator options). The branch or indicator operands immediately follow the last operand listed above. See Chapter 1. "Introduction" for the encoding of the extender field and the allowed syntax of the branch and indicator operands.

Description: The Sum is the result of adding the Addend and Augend.

Operands can have floating-point, packed or zoned decimal, signed or unsigned binary type.

Source operands are the Addend and Augend. The receiver operand is the Sum.

If operands are not of the same type, addends are converted according to the following rules:

1. If any one of the operands has floating point type, addends are converted to floating point type.
2. Otherwise, if any one of the operands has zoned or packed decimal type, addends are converted to packed decimal.
3. Otherwise, the binary operands are converted to a like type. Note: unsigned binary(2) scalars are logically treated as signed binary(4) scalars.

Addend and Augend are added according to their type. Floating point operands are added using floating point addition. Packed decimal addends are added using packed decimal addition. Unsigned binary addition is used with unsigned addends. Signed binary addends are added using two's complement binary addition.

Better performance can be obtained if all operands have the same type. Signed and unsigned binary additions execute faster than either packed decimal or floating point additions.

Decimal operands used in floating-point operations cannot contain more than 15 total digit positions.

For a decimal operation, alignment of the assumed decimal point takes place by padding with 0's on the right end of the addend with lesser precision.

Floating-point addition uses exponent comparison and significand addition. Alignment of the binary point is performed, if necessary, by shifting the significand of the value with the smaller exponent to the right. The exponent is increased by one for each binary digit shifted until the two exponents agree.

The operation uses the lengths and the precision of the source and receiver operands to calculate accurate results. Operations performed in decimal are limited to 31 decimal digits in the intermediate result.

The addition operation is performed according to the rules of algebra.

The result of the operation is copied into the sum operand. If this operand is not the same type as that used in performing the operation, the resultant value is converted to its type. If necessary, the resultant value is adjusted to the length of the sum, aligned at the assumed decimal point of the sum operand, or both before being copied. If nonzero digits are truncated on the left end of the resultant value, a size exception is signaled.

When the target of the instruction is signed or unsigned binary size exceptions can be suppressed.

For the optional round form of the instruction, specification of a floating-point receiver operand is invalid.

For fixed-point operations, if nonzero digits are truncated off the left end of the resultant value, a size exception is signaled.

For floating-point operations involving a fixed-point receiver field, if nonzero digits would be truncated off the left end of the resultant value, an invalid floating-point conversion exception is signaled.

For a floating-point sum, if the exponent of the resultant value is either too large or too small to be represented in the sum field, the floating-point overflow and floating-point underflow exceptions are signaled, respectively.

If a decimal to binary conversion causes a size exception to be signaled, the binary value contains the correct truncated result only if the decimal value contains 15 or fewer significant nonfractional digits.

Resultant Conditions: Positive, negative, or zero - The algebraic value of the numeric scalar sum operand is positive, negative, or zero. Unordered - The value assigned a floating-point sum operand is NaN.

Exceptions

Exception		Operands 1	2	3	Other
06	Addressing				
	01 Spacing addressing violation	X	X	X	
	02 Boundary alignment	X	X	X	
	03 Range	X	X	X	
	06 Optimized addressability invalid	X	X	X	
08	Argument/parameter				
	01 Parameter reference violation	X	X	X	
0C	Computation				
	02 Decimal data		X	X	
	03 Decimal point alignment		X	X	
	06 Floating-point overflow	X			
	07 Floating-point underflow	X			
	09 Floating-point invalid operand		X	X	X
	0A Size	X			
	0C Invalid floating-point conversion	X			
	0D Floating-point inexact result	X			
10	Damage encountered				
	04 System object damage state	X	X	X	X
	44 Partial system object damage	X	X	X	X
1C	Machine-dependent exception				
	03 Machine storage limit exceeded				X

Exception		Operands 1	2	3	Other
20	Machine support				
	02 Machine check				X
	03 Function check				X
22	Object access				
	01 Object not found	X	X	X	
	02 Object destroyed	X	X	X	
	03 Object suspended	X	X	X	
24	Pointer specification				
	01 Pointer does not exist	X	X	X	
	02 Pointer type invalid	X	X	X	
2A	Program creation				
	05 Invalid op-code extender field				X
	06 Invalid operand type	X	X	X	
	07 Invalid operand attribute	X	X	X	
	08 Invalid operand value range	X	X	X	
	09 Invalid branch target operand				X
	0C Invalid operand odt reference	X	X	X	
	0D Reserved bits are not zero	X	X	X	X
2C	Program execution				
	04 Invalid branch target				X
2E	Resource control limit				
	01 user profile storage limit exceeded				X
36	Space management				
	01 space extension/truncation				X

1.3 And (AND)

Op Code (Hex)	Operand 1	Operand 2	Operand 3
1093	Receiver	Source 1	Source 2

Operand 1: Character variable scalar or numeric variable scalar.

Operand 2: Character scalar or numeric scalar.

Operand 3: Character scalar or numeric scalar.

Optional Forms

Mnemonic	Op Code (Hex)	Form Type
ANDS	1193	Short
ANDI	1893	Indicator
ANDIS	1993	Indicator, Short
ANDB	1C93	Branch
ANDBS	1D93	Branch, Short

If the short instruction option is indicated in the op code, operand 1 is used as the first and second operational operands (receiver and first source operand). Operand 2 is used as the third operational operand (second source operand).

Extender: Branch options or indicator options.

If the branch or indicator option is specified in the op code, the extender field must be present along with one or two branch targets (for branch options) or one or two indicator operands (for indicator options). The branch or indicator operands immediately follow the last operand listed above. See Chapter 1. "Introduction" for the encoding of the extender field and the allowed syntax of the branch and indicator operands.

Description: The Boolean AND operation is performed on the string values in the source operands. The resulting string is placed in the receiver operand. The operands may be character or numeric scalars. They are both interpreted as bit strings. Substringing is supported for both character and numeric operands.

The length of the operation is equal to the length of the longer of the two source operands. The shorter of the two operands is logically padded on the right with hex 00 values. This assigns hex 00 values to the results for those bytes that correspond to the excess bytes of the longer operand.

The bit values of the result are determined as follows:

Source 1 Bit	Source 2 Bit	Result Bit
1	1	1
0	1	0
1	0	0
0	0	0

The result value is then placed (left-adjusted) in the receiver operand with truncating or padding taking place on the right. The pad value used in this instruction is a byte value of hex 00.

If operands overlap but do not share all of the same bytes, results of operations performed on these operands are not predictable. If overlapped operands share all of the same bytes, the results are predictable when direct addressing is used. If indirect addressing is used (that is, based operands, parameters, strings with variable lengths, and arrays with variable subscripts), the results are not always predictable.

Substring operand references that allow for a null substring reference (a length value of zero) may be specified for operands 1, 2, and 3. The effect of specifying a null substring reference for either or both of the source operands is that the result is all zero and instruction's resultant condition is zero. When a null substring reference is specified for the receiver, a result is not set and the instruction's resultant condition is Zero regardless of the values of the source operands.

When the receiver operand is a numeric variable scalar, it is possible that the result produced will not be a valid value for the numeric type. This can occur due to padding with hex 00, due to truncation, or due to the resultant bit string produced by the instruction. The instruction completes normally and signals no exceptions for these conditions.

Resultant Conditions: Zero - The bit value for the bits of the scalar receiver operand is either all zero or a null substring reference is specified for the receiver. Not zero - The bit value for the bits of the scalar receiver operand is not all zero.

Exceptions

Exception		Operands 1	2	3	Other
06	Addressing				
	01 Spacing addressing violation	X	X	X	
	02 Boundary alignment	X	X	X	
	03 Range	X	X	X	
	06 Optimized addressability invalid	X	X	X	
08	Argument/parameter				
	01 Parameter reference violation	X	X	X	
10	Damage encountered				
	04 System object damage state	X	X	X	X
	44 Partial system object damage	X	X	X	X
1C	Machine-dependent exception				
	03 Machine storage limit exceeded				X
20	Machine support				
	02 Machine check				X
	03 Function check				X

Exception		Operands 1	2	3	Other
22	Object access				
	01 Object not found	X	X	X	
	02 Object destroyed	X	X	X	
	03 Object suspended	X	X	X	
24	Pointer specification				
	01 Pointer does not exist	X	X	X	
	02 Pointer type invalid	X	X	X	
2A	Program creation				
	05 Invalid op-code extender field				X
	06 Invalid operand type	X	X	X	
	07 Invalid operand attribute	X	X	X	
	08 Invalid operand value range	X	X	X	
	09 Invalid branch target operand				X
	0A Invalid operand length	X	X	X	
	0C Invalid operand odt reference	X	X	X	
	0D Reserved bits are not zero	X	X	X	X
2C	Program execution				
	04 Invalid branch target				X
2E	Resource control limit				
	01 user profile storage limit exceeded				X
36	Space management				
	01 space extension/truncation				X

1.4 Branch (B)

Op Code (Hex)	Operand 1
1011	Branch Target

Operand 1: Instruction number, relative instruction number, branch point, instruction pointer, or instruction definition list element.

Description: Control is unconditionally transferred to the instruction indicated in the branch target operand. The instruction number indicated by the branch target operand must be within the instruction stream containing the branch instruction.

The branch target may be an element of an array of instruction pointers or an element of an instruction definition list. The specific element can be identified by using a compound subscript operand.

Exceptions

Exception		Operand 1	Other
06	Addressing		
	01 Spacing addressing violation	X	
	02 Boundary alignment violation	X	
	03 Range	X	
08	Argument/parameter		
	01 Parameter reference violation	X	
10	Damage encountered		
	04 System object damage state	X	X
	44 Partial system object damage	X	X
1C	Machine-dependent exception		
	03 Machine storage limit exceeded		X
20	Machine support		
	02 Machine check		X
	03 Function check		X
22	Object access		
	01 Object not found	X	
	02 Object destroyed	X	
	03 Object suspended	X	
24	Pointer specification		
	01 Pointer does not exist	X	
	02 Pointer type invalid	X	
2A	Program creation		
	06 Invalid operand type	X	

Exception		Operand 1	Other
	07 Invalid operand attribute	X	
	09 Invalid branch target operand	X	
	0C Invalid operand odt reference	X	
	0D Reserved bits are not zero	X	X
2C	Program execution		
	04 Invalid branch target	X	
2E	Resource control limit		
	01 user profile storage limit exceeded		X
36	Space management		
	01 space extension/truncation		X

1.5 Clear Bit in String (CLRBTS)

Op Code (Hex)	Operand 1	Operand 2
102E	Receiver	Offset

Operand 1: Character Variable Scalar or Numeric Variable Scalar.

Operand 2: Binary Scalar.

Description: Clears the bit of the receiver operand as indicated by the bit offset operand.

The selected bit from the receiver operand is set to a value of B'0'.

The receiver operand can be character or numeric. The leftmost bytes of the receiver operand are used in the operation. The receiver operand is interpreted as a bit string with the bits numbered left to right from 0 to the total number of bits in the string minus 1.

The receiver cannot be a variable substring.

The offset operand indicates which bit of the receiver operand is to be cleared, with a offset of zero indicating the leftmost bit of the leftmost byte of the receiver operand.

If a offset value less than zero or beyond the length of the string is specified a "scalar value invalid" exception is raised.

Exceptions

Exception		Operands 1	Operands 2	Other
06	Addressing			
	01 Spacing addressing violation	X	X	
	02 Boundary alignment violation	X	X	
	03 Range	X	X	
	06 Optimized addressability invalid	X	X	
08	Argument/parameter			
	01 Parameter reference violation	X	X	
10	Damage encountered			
	04 System object damage state			X
	44 Partial system object damage			X
1C	Machine-dependent exception			
	03 Machine storage limit exceeded			X
20	Machine support			
	02 Machine check			X
	03 Function check			X

Exception		Operands 1	2	Other
22	Object access			
	02 Object destroyed	X	X	
	03 Object suspended	X	X	
24	Pointer specification			
	01 Pointer does not exist	X	X	
	02 Pointer type invalid	X	X	
2A	Program creation			
	06 Invalid operand type	X	X	
	07 Invalid operand attribute	X	X	
	08 Invalid operand value range	X	X	
	0A Invalid operand length	X	X	
	0C Invalid operand odt reference	X	X	
	0D Reserved bits are not zero	X	X	X
2E	Resource control limit			
	01 user profile storage limit exceeded			X
32	Scalar specification			
	01 Scalar type invalid	X	X	
	03 Scalar value invalid		X	
36	Space management			
	01 space extension/truncation			X

1.6 Compare Bytes Left-Adjusted (CMPBLAB or CMPBLAI)

Op Code (Hex)	Extender	Operand 1	Operand 2	Operand 3 [4, 5]
1CC2	Branch options	Compare operand 1	Compare operand 2	Branch target
18C2	Indicator options			Indicator target

Operand 1: Numeric scalar or character scalar.

Operand 2: Numeric scalar or character scalar.

Operand 3 [*4, 5*]:

- *Branch Form*-Instruction number, relative instruction number, branch point, or instruction pointer.
- *Indicator Form*-Numeric variable scalar or character variable scalar.

Extender: Branch or indicator options.

Either the branch or indicator option is required by the instruction. The extender field is required along with from one to three branch targets (for branch option) or one to three indicator operands (for indicator option). The branch or indicator operands are required for operand 3 and optional for operands 4 and 5. See Chapter 1. "Introduction" for the bit encoding of the extender field and the allowed syntax of the branch and indicator operands.

Description: This instruction compares the logical string values of two left-adjusted compare operands. The logical string value of the first compare operand is compared with the logical string value of the second compare operand (no padding done). Based on the comparison, the resulting condition is used with the extender field to:

- Transfer control conditionally to the instruction indicated in one of the branch target operands (branch form).
- Assign a value to each of the indicator operands (indicator form).

The compare operands can be either character or numeric. Any numeric operands are interpreted as logical character strings.

The compare operands are compared byte by byte, from left to right with no numeric conversions performed. The length of the operation is equal to the length of the shorter of the two compare operands. The comparison begins with the leftmost byte of each of the compare operands and proceeds until all bytes of the shorter compare operand have been compared or until the first unequal pair of bytes is encountered.

Substring operand references that allow for a null substring reference (a length value of zero) may be specified for operands 1 and 2. The effect of specifying a null substring reference for either or both compare operands is that the instruction's resultant condition is equal.

Resultant Conditions: The scalar first compare operand has a higher, lower, or equal string value than the second compare operand.

Exceptions

Exception		Operands 1	2	3 [4, 5]	Other
06	Addressing				
	01 Spacing addressing violation	X	X	X	
	02 Boundary alignment	X	X	X	
	03 Range	X	X	X	
	06 Optimized addressability invalid	X	X	X	
08	Argument/parameter				
	01 Parameter reference violation	X	X	X	
10	Damage encountered				
	04 System object damage state	X	X	X	X
	44 Partial system object damage	X	X	X	X
1C	Machine-dependent exception				
	03 Machine storage limit exceeded				X
20	Machine support				
	02 Machine check				X
	03 Function check				X
22	Object access				
	01 Object not found	X	X	X	
	02 Object destroyed	X	X	X	
	03 Object suspended	X	X	X	
24	Pointer specification				
	01 Pointer does not exist	X	X	X	
	02 Pointer type invalid	X	X	X	
2A	Program creation				
	05 Invalid op-code extender field				X
	06 Invalid operand type	X	X	X	
	07 Invalid operand attribute	X	X	X	
	08 Invalid operand value range	X	X	X	
	09 Invalid branch target operand				X
	0A Invalid operand length	X	X	X	
	0C Invalid operand odt reference	X	X	X	
	0D Reserved bits are not zero	X	X	X	X
2C	Program execution				
	04 Branch target invalid				X

Exception		Operands 1	2	3 [4, 5]	Other
2E	Resource control limit				
	01 user profile storage limit exceeded				X
36	Space management				
	01 space extension/truncation				X

1.7 Compare Bytes Left-Adjusted with Pad (CMPBLAPB or CMPBLAPI)

Op Code (Hex)	Extender	Operand 1	Operand 2	Operand 3	Operand 4 [5, 6]
1CC3	Branch options	Compare operand 1	Compare operand 2	Pad	Branch target
18C3	Indicator options				Indicator target

Operand 1: Numeric scalar or character scalar.

Operand 2: Numeric scalar or character scalar.

Operand 3: Numeric scalar or character scalar.

Operand 4 [*5, 6*]:

- *Branch Form*-Instruction number, relative instruction number, branch point, or instruction pointer.
- *Indicator Form*-Numeric variable scalar or character variable scalar.

Extender: Branch or indicator options.

Either the branch or indicator option is required by the instruction. The extender field is required along with from one to three branch targets (for branch option) or one to three indicator operands (for indicator option). The branch or indicator operands are required for operand 4 and optional for operands 5 and 6. See Chapter 1. "Introduction" for the bit encoding of the extender field and the allowed syntax of the branch and indicator operands.

Description: This instruction compares the logical string values of two left-adjusted compare operands (padded if needed). The logical string value of the first compare operand is compared with the logical string value of the second compare operand. Based on the comparison, the resulting condition is used with the extender field to:

- Transfer control conditionally to the instruction indicated in one of the branch target operands (branch form).
- Assign a value to each of the indicator operands (indicator form).

The compare operands can be either character or numeric. Any numeric operands are interpreted as logical character strings.

The compare operands are compared byte by byte, from left to right with no numeric conversions being performed.

The length of the operation is equal to the length of the longer of the two compare operands. The shorter of the two compare operands is logically padded on the right with the 1-byte value indicated in the pad operand. If the pad operand is more than 1 byte in length, only its leftmost byte is used. The comparison begins with the leftmost byte of each of the compare operands and proceeds until all the bytes of the longer of the two compare operands have

been compared or until the first unequal pair of bytes is encountered. All excess bytes in the longer of the two compare operands are compared to the pad value.

Substring operand references that allow for a null substring reference (a length value of zero) may be specified for operands 1 and 2. The effect of specifying a null substring reference for one of the compare operands is that the other compare operand is compared with an equal length string of pad character values. When a null substring reference is specified for both compare operands, the resultant condition is equal.

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for operand 3.

Resultant Conditions: The scalar first compare operand has a higher, lower, or equal string value than the second compare operand.

Exceptions

Exception		Operands 1	2	3	4 [5, 6]	Other
06	Addressing					
	01 Spacing addressing violation	X	X	X	X	
	02 Boundary alignment	X	X	X	X	
	03 Range	X	X	X	X	
	06 Optimized addressability invalid	X	X	X	X	
08	Argument/parameter					
	01 Parameter reference violation	X	X	X	X	
10	Damage encountered					
	04 System object damage state	X	X	X	X	X
	44 Partial system object damage	X	X	X	X	X
1C	Machine-dependent exception					
	03 Machine storage limit exceeded					X
20	Machine support					
	02 Machine check					X
	03 Function check					X
22	Object access					
	01 Object not found	X	X	X	X	
	02 Object destroyed	X	X	X	X	
	03 Object suspended	X	X	X	X	
24	Pointer specification					
	01 Pointer does not exist	X	X	X	X	
	02 Pointer type invalid	X	X	X	X	
2A	Program creation					

Exception		Operands 1	2	3	4 [5, 6]	Other
	05 Invalid op-code extender field					X
	06 Invalid operand type	X	X	X	X	
	07 Invalid operand attribute	X	X	X	X	
	08 Invalid operand value range	X	X	X	X	
	09 Invalid branch target operand					X
	0A Invalid operand length	X	X			
	0C Invalid operand odt reference	X	X	X	X	
	0D Reserved bits are not zero	X	X	X	X	X
2C	Program execution					
	04 Branch target invalid					X
2E	Resource control limit					
	01 user profile storage limit exceeded					X
36	Space management					
	01 space extension/truncation					X

1.8 Compare Bytes Right-Adjusted (CMPBRAB or CMPBRAI)

Op Code (Hex)	Extender	Operand 1	Operand 2	Operand 3 [4, 5]
1CC6	Branch options	Compare operand 1	Compare operand 2	Branch target
18C6	Indicator options			Indicator target

Operand 1: Numeric scalar or character scalar.

Operand 2: Numeric scalar or character scalar.

Operand 3 [*4, 5*]:

- *Branch Form*-Instruction number, relative instruction number, branch point, or instruction pointer.
- *Indicator Form*-Numeric variable scalar or character variable scalar.

Extender: Branch or indicator options.

Either the branch or the indicator option is required by the instruction. The extender field is required along with from one to three branch targets (for branch option) or one to three indicator operands (for indicator option). The branch or indicator operands are required for operand 3 and optional for operands 4 and 5. See Chapter 1. "Introduction" for the bit encoding of the extender field and the allowed syntax of the branch and indicator operands.

Description: This instruction compares the logical string values of two right-adjusted compare operands. The logical string value of the first compare operand is compared with the logical string value of the second compare operand (no padding done). Based on the comparison, the resulting condition is used with the extender field to:

- Transfer control conditionally to the instruction indicated in one of the branch target operands (branch form).
- Assign a value to each of the indicator operands (indicator form).

The compare operands can be either string or numeric. Any numeric operands are interpreted as logical character strings.

The compare operands are compared byte by byte, from left to right with no numeric conversions performed. The length of the operation is equal to the length of the shorter of the two compare operands. The comparison begins with the leftmost byte of each of the compare operands and proceeds until all bytes of the shorter compare operand have been compared or until the first unequal pair of bytes is encountered.

Substring operand references that allow for a null substring reference (a length value of zero) may be specified for operands 1 and 2. The effect of specifying a null substring reference for either or both compare operands is that the instruction's resultant condition is equal.

Resultant Conditions: The scalar first compare operand has a higher, lower, or equal string value than the second compare operand.

Exceptions

Exception		Operands 1	2	3 [4, 5]	Other
06	Addressing				
	01 Spacing addressing violation	X	X	X	
	02 Boundary alignment violation	X	X	X	
	03 Range	X	X	X	
	06 Optimized addressability invalid	X	X	X	
08	Argument/parameter				
	01 Parameter reference violation	X	X	X	
10	Damage encountered				
	04 System object damage state	X	X	X	X
	44 Partial system object damage	X	X	X	X
1C	Machine-dependent exception				
	03 Machine storage limit exceeded				X
20	Machine support				
	02 Machine check				X
	03 Function check				X
22	Object access				
	01 Object not found	X	X	X	
	02 Object destroyed	X	X	X	
	03 Object suspended	X	X	X	
24	Pointer specification				
	01 Pointer does not exist	X	X	X	
	02 Pointer type invalid	X	X	X	
2A	Program creation				
	05 Invalid op-code extender field				X
	06 Invalid operand type	X	X	X	
	07 Invalid operand attribute	X	X	X	
	08 Invalid operand value range	X	X	X	
	09 Invalid branch target operand				X
	0A Invalid operand length	X	X		
	0C Invalid operand odt reference	X	X	X	
	0D Reserved bits are not zero	X	X	X	X
2C	Program execution				
	04 Branch target invalid			X	X

Exception		Operands 1	2	3 [4, 5]	Other
2E	Resource control limit				
	01 user profile storage limit exceeded				X
36	Space management				
	01 space extension/truncation				X

1.9 Compare Bytes Right-Adjusted with Pad (CMPBRAPB or CMPBRAPI)

Op Code (Hex)	Extender	Operand 1	Operand 2	Operand 3	Operand 4 [5, 6]
1CC7	Branch options	Compare operand 1	Compare operand 2	Pad	Branch target
18C7	Indicator options				Indicator target

Operand 1: Numeric scalar or character scalar.

Operand 2: Numeric scalar or character scalar.

Operand 3: Numeric scalar or character scalar.

Operand 4 [*5, 6*]:

- *Branch Form*-Instruction number, relative instruction number, branch point, or instruction pointer.
- *Indicator Form*-Numeric variable scalar or character variable scalar.

Extender: Branch or indicator options.

Either the branch or the indicator option is required by the instruction. The extender field is required along with from one to three branch targets (for branch option) or one to three indicator operands (for indicator option). The branch or indicator operands are required for operand 4 and optional for operands 5 and 6. See Chapter 1. "Introduction" for the bit encoding of the extender field and the allowed syntax of the branch and indicator operands.

Description: This instruction compares the logical string values of the right-adjusted compare operands (padded if needed). The logical string value of the first compare operand is compared with the logical string value of the second compare operand. Based on the comparison, the resulting condition is used with the extender field to:

- Transfer control conditionally to the instruction indicated in one of the branch target operands (branch form).
- Assign a value to each of the indicator operands (indicator form).

The compare operands can be either character or numeric. Any numeric operands are interpreted as logical character strings.

The compare operands are compared byte by byte, from left to right with no numeric conversions performed.

The length of the operation is equal to the length of the longer of the two compare operands. The shorter of the two compare operands is logically padded on the left with the 1-byte value indicated in the pad operand. If the pad operand is more than 1 byte in length, only its leftmost byte is used. The comparison begins with the leftmost byte of the longer of the compare operands.

Any excess bytes (on the left) in the longer compare operand are compared with the pad value. All other bytes are compared with the corresponding bytes in the other compare operand. The operation proceeds until all bytes in the longer operand are compared or until the first unequal pair of bytes is encountered.

Substring operand references that allow for a null substring reference (a length value of zero) may be specified for operands 1 and 2. The effect of specifying a null substring reference for one of the compare operands is that the other compare operand is compared with an equal length string of pad character values. When a null substring reference is specified for both compare operands, the instruction's resultant condition is equal.

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for operand 3.

Resultant Conditions: The scalar first compare operand has a higher, lower, or equal string value than the second compare operand.

Exceptions

Exception		Operands 1	2	3	4 [5, 6]	Other
06	Addressing					
	01 Spacing addressing violation	X	X	X	X	
	02 Boundary alignment	X	X	X	X	
	03 Range	X	X	X	X	
	06 Optimized addressability invalid	X	X	X	X	
08	Argument/parameter					
	01 Parameter reference violation	X	X	X	X	
10	Damage encountered					
	04 System object damage state	X	X	X	X	X
	44 Partial system object damage	X	X	X	X	X
1C	Machine-dependent exception					
	03 Machine storage limit exceeded					X
20	Machine support					
	02 Machine check					X
	03 Function check					X
22	Object access					
	01 Object not found	X	X	X	X	
	02 Object destroyed	X	X	X	X	
	03 Object suspended	X	X	X	X	
24	Pointer specification					
	01 Pointer does not exist	X	X	X	X	
	02 Pointer type invalid	X	X	X	X	

Exception		Operands 1	2	3	4 [5, 6]	Other
2A	Program creation					
	05 Invalid op-code extender field					X
	06 Invalid operand type	X	X	X	X	
	07 Invalid operand attribute	X	X	X	X	
	08 Invalid operand value range	X	X	X	X	
	09 Invalid branch target operand					X
	0A Invalid operand length	X	X			
	0C Invalid operand odt reference	X	X	X	X	
	0D Reserved bits are not zero	X	X	X	X	X
2C	Program execution					
	04 Branch target invalid			X	X	
2E	Resource control limit					
	01 user profile storage limit exceeded					X
36	Space management					
	01 space extension/truncation					X

1.10 Compare Numeric Value (CMPNVB or CMPNVI)

Op Code (Hex)	Extender	Operand 1	Operand 2	Operand 3 [4-6]
1C46	Branch options	Compare operand 1	Compare operand 2	Branch target
1846	Indicator options			Indicator target

Operand 1: Numeric scalar.

Operand 2: Numeric scalar.

Operand 3 [*4-6*]:

- *Branch Form*-Instruction number, relative instruction number, branch point, or instruction pointer.
- *Indicator Form*-Numeric variable scalar or character variable scalar.

Extender: Branch or indicator options.

Either the branch or indicator option is required by the instruction. The extender field is required along with from one to four branch targets (for branch option) or one to four indicator operands (for indicator option). The branch or indicator operands are required for operand 3 and optional for operands 4 and 5. See Chapter 1. "Introduction" for the bit encoding of the extender field and the allowed syntax of the branch and indicator operands.

Description: The numeric value of the first compare operand is compared with the signed or unsigned numeric value of the second compare operand. Based on the comparison, the resulting condition is used with the extender field to:

- Transfer control conditionally to the instruction indicated in one of the branch target operands (branch form).
- Assign a value to each of the indicator operands (indicator form).

For a decimal operation, alignment of the assumed decimal point takes place by padding with 0's on the right end of the compare operand with lesser precision.

Decimal operands used in floating-point operations cannot contain more than 15 total digit positions.

When both operands are signed numeric or both are unsigned numeric the length of the operation is equal to the length of the longer of the two compare operands.

When one operand is signed numeric and the other operand unsigned numeric the unsigned operand is converted to a signed value with more precision than its current size. The length of the operation is equal to the length of the longer of the two compare operands. A negative signed numeric value will always be less than a positive unsigned value.

Floating-point comparisons use exponent comparison and significand comparison. For a denormalized floating-point number, the comparison is performed as if the denormalized number had first been normalized.

For floating-point, two values compare unordered when at least one comparand is NaN. Every NaN compares unordered with everything including another NaN value.

Floating-point comparisons ignore the sign of zero. Positive zero always compares equal with negative zero.

A floating-point invalid operand exception is signaled when two floating-point values compare unordered and no branch or indicator option exists for any of the unordered, negation of unordered equal, or negation of equal resultant conditions.

When a comparison is made between a floating-point compare operand and a fixed-point decimal compare operand that contains fractional digit positions, a floating-point inexact result exception may be signaled because of the implicit conversion from decimal to floating-point.

Resultant Conditions: High, low, or equal-The first compare operand has a higher, lower, or equal numeric value than the second compare operand. Unordered-The first compare operand is unordered compared to the second compare operand.

Exceptions

Exception		Operands 1	2	3 [4-6]	Other
06	Addressing				
	01 Spacing addressing violation	X	X	X	
	02 Boundary alignment	X	X	X	
	03 Range	X	X	X	
	06 Optimized addressability invalid	X	X	X	
08	Argument/parameter				
	01 Parameter reference violation	X	X	X	
0C	Computation				
	02 Decimal data	X	X		
	03 Decimal point alignment	X	X		
	09 Floating-point invalid operand		X		X
	0D Floating-point inexact result				X
10	Damage encountered				
	04 System object damage state	X	X	X	X
	44 Partial system object damage	X	X	X	X
1C	Machine-dependent exception				
	03 Machine storage limit exceeded				X
20	Machine support				
	02 Machine check				X
	03 Function check				X

Exception		Operands 1	2	3 [4-6]	Other
22	Object access				
	01 Object not found	X	X	X	
	02 Object destroyed	X	X	X	
	03 Object suspended	X	X	X	
24	Pointer specification				
	01 Pointer does not exist	X	X	X	
	02 Pointer type invalid	X	X	X	
2A	Program creation				
	05 Invalid op-code extender field				X
	06 Invalid operand type	X	X	X	
	07 Invalid operand attribute	X	X	X	
	08 Invalid operand value range	X	X	X	
	09 Invalid branch target operand				X
	0C Invalid operand odt reference	X	X	X	
	0D Reserved bits are not zero	X	X	X	X
2C	Program execution				
	04 Branch target invalid			X	
2E	Resource control limit				
	01 user profile storage limit exceeded				X
36	Space management				
	01 space extension/truncation				X

1.11 Compute Array Index (CAI)

Op Code (Hex)	Operand 1	Operand 2	Operand 3	Operand 4
1044	Array index	Subscript A	Subscript B	Dimension

Operand 1: Binary(2) variable scalar.

Operand 2: Binary(2) scalar.

Operand 3: Binary(2) scalar.

Operand 4: Binary(2) constant scalar object or immediate operand.

Description: This instruction provides the ability to reduce multidimensional array subscript values into a single index value which can then be used in referencing the single-dimensional arrays of the system. This index value is computed by performing the following arithmetic operation on the indicated operands.

Array Index = Subscript A + ((Subscript B-1) X Dimension)

The numeric value of the subscript B operand is decreased by 1 and multiplied by the numeric value of the dimension operand. The result of this multiplication is added to the subscript A operand and the sum is placed in the array index operand.

All the operands must be binary with any implicit conversions occurring according to the rules of arithmetic operations. The usual rules of algebra are observed concerning the subtraction, addition, and multiplication of operands.

This instruction provides for mapping multidimensional arrays to single-dimensional arrays. The elements of an array with the dimensions (d1, d2, d3, ..., dn) can be defined as a single-dimensional array with d1*d2*d3*...*dn elements. To reference a specific element of the multidimensional array with subscripts (s1,s2,s3,...sn), it is necessary to convert the multiple subscripts to a single subscript for use in the single-dimensional AS/400 array. This single subscript can be computed using the following:

```
s1+((s2-1)*d1)+(s3-1)*d1*d2)+...+((sn-1)*d*d2*d3*...*dm)
```

where m = n-1

The CAI instruction is used to form a single index value from two subscript values. To reduce N subscript values into a single index value, N-1 uses of this instruction are necessary.

Assume that S1, S2, and S3 are three subscript values and that D1 is the size of one dimension, D2 is the size of the second dimension, and the D1D2 is the product of D1 and D2. The following two uses of this instruction reduce the three subscripts to a single subscript.

```
CAI INDEX, S1, S2, D1      Calculates s1+(s2-1)*d1
CAI INDEX, INDEX, S3, D1D2 Calculates s1+(s2-1)*d1+(s3-1)*d2*d1
```

Exceptions

Exception		Operands 1	2	3	4	Other
06	Addressing					
	01 Spacing addressing violation	X	X	X	X	
	02 Boundary alignment	X	X	X	X	
	03 Range	X	X	X	X	
	06 Optimized addressability invalid	X	X	X	X	
08	Argument/parameter					
	01 Parameter reference violation	X	X	X	X	
0C	Computation					
	0A size	X				
10	Damage encountered					
	04 System object damage state	X	X	X	X	X
	44 Partial system object damage	X	X	X	X	X
1C	Machine-dependent exception					
	03 Machine storage limit exceeded					X
20	Machine support					
	02 Machine check					X
	03 Function check					X
22	Object access					
	01 Object not found	X	X	X	X	
	02 Object destroyed	X	X	X	X	
	03 Object suspended	X	X	X	X	
24	Pointer specification					
	01 Pointer does not exist	X	X	X	X	
	02 Pointer type invalid	X	X	X	X	
2A	Program creation					
	06 Invalid operand type	X	X	X	X	
	07 Invalid operand attribute	X	X	X	X	
	08 Invalid operand value range	X	X	X	X	
	0C Invalid operand odt reference	X	X	X	X	
	0D Reserved bits are not zero	X	X	X	X	X
2E	Resource control limit					
	01 user profile storage limit exceeded					X
36	Space management					
	01 space extension/truncation					X

1.12 Compute Math Function Using One Input Value (CMF1)

Op Code (Hex)	Operand 1	Operand 2	Operand 3
100B	Receiver	Controls	Source

Operand 1: Numeric variable scalar.

Operand 2: Character(2) scalar.

Operand 3: Numeric scalar.

Optional Forms

Mnemonic	Op Code (Hex)	Form Type
CMF1I	180B	Indicator
CMF1B	1C0B	Branch

Extender: Branch options or indicator options.

If the branch or indicator option is specified in the op code, the extender field must be present along with one or two branch targets (for branch options) or one or two indicator operands (for indicator options). The branch or indicator operands immediately follow the last operand listed above. See Chapter 1. "Introduction" for the encoding of the extender field and the allowed syntax of the branch and indicator operands.

Description: The mathematical function, indicated by the controls operand, is performed on the source operand value and the result is placed in the receiver operand.

The calculation is always done in floating-point.

The result of the operation is copied into the receiver operand.

The controls operand must be a character scalar that specifies which mathematical function is to be performed. It must be at least 2 bytes in length and has the following format:

- Controls operand Char(2)
  - Hex 0001 = Sine
  - Hex 0002 = Arc sine
  - Hex 0003 = Cosine
  - Hex 0004 = Arc cosine
  - Hex 0005 = Tangent
  - Hex 0006 = Arc tangent
  - Hex 0007 = Cotangent
  - Hex 0010 = Exponential function
  - Hex 0011 = Logarithm based e (natural logarithm)
  - Hex 0012 = Sine hyperbolic
  - Hex 0013 = Cosine hyperbolic
  - Hex 0014 = Tangent hyperbolic
  - Hex 0015 = Arc tangent hyperbolic
  - Hex 0020 = Square root

— All other values are reserved

The controls operand mathematical functions are as follows:

- Hex 0001-Sine

  The sine of the numeric value of the source operand, whose value is considered to be in radians, is computed and placed in the receiver operand.

  The result is in the range:

  $-1 \leq SIN(x) \leq 1$

- Hex 0002-Arc sine

  The arc sine of the numeric value of the source operand is computed and the result (in radians) is placed in the receiver operand.

  The result is in the range:

  $-pi/2 \leq ASIN(x) \leq +pi/2$

- Hex 0003-Cosine

  The cosine of the numeric value of the source operand, whose value is considered to be in radians, is computed and placed in the receiver operand.

  The result is in the range:

  $-1 \leq COS(x) \leq 1$

- Hex 0004-Arc cosine

  The arc cosine of the numeric value of the source operand is computed and the result (in radians) is placed in the receiver operand.

  The result is in the range:

  $0 \leq ACOS(x) \leq pi$

- Hex 0005-Tangent

  The tangent of the source operand, whose value is considered to be in radians, is computed and the result is placed in the receiver operand.

  The result is in the range:

  $-infinity \leq TAN(x) \leq +infinity$

- Hex 0006-Arc tangent

  The arc tangent of the source operand is computed and the result (in radians) is placed in the receiver operand.

  The result is in the range:

  $-pi/2 \leq ATAN(x) \leq pi/2$

- Hex 0007-Cotangent

  The cotangent of the source operand, whose value is considered to be in radians, is computed and the result is placed in the receiver operand.

  The result is in the range:

  $-infinity \leq COT(x) \leq +infinity$

- Hex 0010-Exponential function

The computation e power (source operand) is performed and the result is placed in the receiver operand.

The result is in the range:

$0 \leq EXP(x) \leq +infinity$

- Hex 0011-Logarithm based e (natural logarithm)

  The natural logarithm of the source operand is computed and the result is placed in the receiver operand.

  The result is in the range:

  $-infinity \leq LN(x) \leq +infinity$

- Hex 0012-Sine hyperbolic

  The sine hyperbolic of the numeric value of the source operand is computed and the result (in radians) is placed in the receiver operand.

  The result is in the range:

  $-infinity \leq SINH(x) \leq +infinity$

- Hex 0013-Cosine hyperbolic

  The cosine hyperbolic of the numeric value of the source operand is computed and the result (in radians) is placed in the receiver operand.

  The result is in the range:

  $+1 \leq COSH(x) \leq +infinity$

- Hex 0014-Tangent hyperbolic

  The tangent hyperbolic of the numeric value of the source operand is computed and the result (in radians) is placed in the receiver operand.

  The result is in the range:

  $-1 \leq TANH(x) \leq +1$

- Hex 0015-Arc tangent hyperbolic

  The inverse of the tangent hyperbolic of the numeric value of the source operand is computed and the result (in radians) is placed in the receiver operand.

  The result is in the range:

  $-infinity \leq ATANH(x) \leq +infinity$

- Hex 0020-Square root

  The square root of the numeric value of the source operand is computed and placed in the receiver operand.

  The result is in the range:

  $0 \leq SQRT(x) \leq +infinity$

The following chart shows some special cases for certain arguments (X) of the different mathematical functions.

X Function	Masked NaN	Unmasked NaN	+infinity	-infinity	+0	-0	Maximum Value	Minimum Value	Other
Sine	g	A(e)	A(f)	A(f)	+0	-0	A(1,f)	A(1,f)	B(3)
Arc sine	g	A(e)	A(f)	A(f)	+0	-0	A(6,f)	A(6,f)	-
Cosine	g	A(e)	A(f)	A(f)	+1	+1	A(1,f)	A(1,f)	B(3)
Arc cosine	g	A(e)	A(f)	A(f)	+pi/2	+pi/2	A(6,f)	A(6,f)	-
Tangent	g	A(e)	A(f)	A(f)	+0	-0	A(1,f)	A(1,f)	B(3)
Arc tangent	g	A(e)	+pi/2	-pi/2	+0	-0	-	-	-
Cotangent	g	A(e)	A(f)	A(f)	+inf	-inf	A(1,f)	A(1,f)	B(3)
Exponent	g	A(e)	+inf	+0	+1	+1	C(4,a)	D(5,b)	-
Logarithm	g	A(e)	+inf	A(f)	-inf	-inf	-	-	A(2,f)
Sine hyperbolic	g	A(e)	+inf	-inf	+0	-0	-	-	-
Cosine hyperbolic	g	A(e)	+inf	+inf	+1	+1	-	-	-
Tangent hyperbolic	g	A(e)	+1	-1	+0	-0	-	-	-
Arc tangent hyperbolic	g	A(e)	A(f)	A(f)	+0	-0	A(6,f)	A(6,f)	-
Square root	g	A(e)	+inf	A(f)	+0	-0	-	-	A(2,f)

Figure 1-1. Special cases for arguments of CMF1 mathematical functions.

Capital letters in the chart indicate the exceptions, small letters indicate the returned results, and Arabic numerals indicate the limits of the arguments (X) as defined in the following lists:

A = Floating-point invalid operand (no result stored if unmasked; if masked, occurrence bit is set)

B = Floating-point inexact result (result is stored whether or not exception is masked)

C = Floating-point overflow (no result is stored if unmasked; if masked, occurrence bit is set)

D = Floating-point underflow (no result is stored if unmasked; occurrence bit is always set)

a = Result follows the rules that depend on round mode

b = Result is +0 or a denormalized value

c = Result is +infinity

d = Result is -infinity

e = Result is the masked form of the input NaN

f = Result is the system default masked NaN

g = Result is the input NaN

inf = Result is infinity

1 = | pi * 2**50 | = Hex 432921FB54442D18

2 = Argument is in the range: -inf < x < -0

3 = | pi * 2**26 | = Hex 41A921FB54442D18

4 = 1n(2**1023) Hex 40862E42FEFA39EF

5 = 1n(2**-1021.4555) = Hex C086200000000000

6 = Argument is in the range: $-1 \leq x \leq +1$

The following chart provides accuracy data for the mathematical functions that can be invoked by this instruction.

Function Name	Sample Selection			Accuracy Data			
				Relative Error (e)		Absolute Error (E)	
	A	Range of x	D	MAX(e)	SD(e)	MAX(E)	SD(E)
Arc cosine	9	0 < = x < = 3.14	U			8.26 * 10**-14	2.11 * 10**-15
Arc sine	10	-1.57 < = x < = 1.57	U	1.02 * 10**-13	2.66 * 10**-15		
Arc tangent	1	-pi/2 < x < pi/2	1			3.33 * 10**-16	9.57 * 10**-17
Arc tangent hyperbolic	14	-3 < = x < = 3	U			1.06 * 10**-14	1.79 * 10**-15
Cosine		(See Sine below)					
Cosine hyperbolic		(See Sine Hyperbolic) below)					
Cotangent	11	-10 < = x < = 100	U	4.83 * 10**-16	1.48 * 10**-16		
		.000001 < = x < = .001	U	4.36 * 10**-16	1.49 * 10**-16		
		4000 < = x < = 4000000	U	5.72 * 10**-16	1.46 * 10**-16		
Exponential	2	-100 < = x < = 300	U	5.70 * 10**-14	1.13 * 10**-14		
Natural	3	0.5 < = x < = 1.5	U			2.77 * 10**-16	8.01 * 10**-17
logarithm	4	-100 < = x < = 700	E	2.17 * 10**-16	7.37 * 10**-17		
Sine cosine	5	-10 < = x < = 100	U			2.22 * 10**-16	1.31 * 10**-16
		.000001 < = x < = .001	U			2.22 * 10**-16	1.56 * 10**-16
		4000 < = x < = 4000000	U			2.22 * 10**-16	1.28 * 10**-16
	6	-10 < = x < = 100	U			3.33 * 10**-16	8.39 * 10**-17
		.000001 < = x < = .001	U			4.33 * 10**-19	1.28 * 10**-19
		4000 < = x < = 4000000	U			3.33 * 10**-16	8.17 * 10**-17
Sine/cosine hyperbolic	12	-100 < = x < = 300	U	6.31 * 10**-16	1.97 * 10**-16		
Square root	7	-100 < = x < = 700	E	4.13 * 10**-16	1.27 * 10**-16		
Tangent	8	-10 < = x < = 100	U	4.59 * 10**-16	1.54 * 10**-16		
		.000001 < = x < = .001	U	4.42 * 10**-16	1.44 * 10**-16	3.25 * 10**-19	8.06 * 10**-20
		4000 < = x < = 4000000	U	4.77 * 10**-16	1.43 * 10**-16		
Tangent hyperbolic	13	-100 < = x < = 300	U	8.35 * 10**-16	3.87 * 10**-17	2.22 * 10**-16	3.17 * 10**-17

Figure 1-2 (Part 1 of 2). Accuracy data for CMF1 mathematical functions.

Algorithm Notes:

1. f(x) = x, and g(x) = ATAN(TAN(x)).
2. f(x) = e**x, and g(x) = e**(1n(e**x)).
3. f(x) = 1n(x), and g(x) = 1n(e**(1n(x))).
4. f(x) = x, and g(x) = 1n(e**x).
5. Sum of squares algorithm. f(x) = 1, and g(x) = SIN(x))**2 + (COS(x))**2.
6. Double angle algorithm. f(x) - SIN(2x), and g(x) = 2*(SIN(x)*COS(x)).
7. f(x) = e(**x, and g(x) = (SQR(e**x))**2.
8. f(x) = TAN(x), and g(x) = SIN(x) / COS(x).
9. f(x) = x, and g(x) = ACOS(COS(x)).
10. f(x) = x, and g(x) = ASIN(SIN(x)).
11. f(x) = COT(x), and g(x) = COS(x) / SIN(x).
12. f(x) = SINH(2x), and g(x) = 2*(SINH(x)*COSH(x)).
13. f(x) = TANH(x), and g(x) = SINH(x) / COSH(x).
14. f(x) = x, and g(x) = ATANH(TANH(x)).

Distribution Note: The sample input arguments were tangents of numbers, x, uniformly distributed between -pi/2 and +pi/2.

Figure 1-2 (Part 2 of 2). Accuracy data for CMF1 mathematical functions.

The vertical columns in the accuracy data chart have the following meanings:

- *Function Name*: This column identifies the principal mathematical functions evaluated with entries arranged in alphabetical order by function name.
- *Sample Selection*: This column identifies the selection of samples taken for a particular math function through the following subcolumns:
  - *A*: identifies the algorithm used against the argument, x, to gather the accuracy samples. The numbers in this column refer to notes describing the functions, f(x) and g(x), which were calculated to test for the anticipated relation where f(x) should equal g(x). An accuracy sample then, is an evaluation of the degree to which this relation held true. The algorithm used to sample the arctangent function, for example, defines g(x) to first calculate the tangent of x to provide an appropriate distribution of input arguments for the arctangent function. Since f(x) is defined simply as the value of x, the relation to be evaluated is then x=ARCTAN(TAN(x)). This type of algorithm, where a function and its inverse are used in tandem, is the usual type employed to provide the appropriate comparison values for the evaluation.
  - "Range of x": gives the range of x used to obtain the accuracy samples. The test values for x are uniformly distributed over this range. It should be noted that x is not always the direct input argument to the function being tested; it is sometimes desirable to distribute the input arguments in a nonuniform fashion to provide a more complete test of the function (see column D below). For each function, accuracy data is given for one or more segments within the valid range of x. In each case, the numbers given are the most meaningful to the function and range under consideration.
  - *D*: identifies the distribution of arguments input to the particular function being sampled. The letter E indicates an exponential distribution. The

letter U indicates a uniform distribution. A number refers to a note providing detailed information regarding the distribution.

- *Accuracy Data*: The maximum relative error and standard deviation of the relative error are generally useful and revealing statistics; however, they are useless for the range of a function where its value becomes zero. This is because the slightest error in the argument can cause an unpredictable fluctuation in the magnitude of the answer. When a small argument error would have this effect, the maximum absolute error and standard deviation of the absolute error are given for the range.
  - *Relative Error (e)*: The maximum relative error and standard deviation (root mean square) of the relative error are defined:

    MAX(e) = MAX(ABS((f(x) - g(x)) / f(x)))

    where: MAX selects the largest of its arguments and ABS takes the absolute value of its argument.

    SD(e) = SQR((1/N) SUMSQ((f(x) - g(x)) / f(x)))

    where: SQR takes the square root of its argument and SUMSQ takes the summation of the squares of its arguments over all of the test cases.
  - *Absolute Error (E)*: The maximum absolute error produced during the testing and the standard deviation (root mean square) of the absolute error are:

    MAX(E) = MAX(ABS(f(x) - g(x)))

    where: the operators are those defined above.

    SD(E) = SQR((1/N) SUMSQ(f(x) - g(x)))

    where: the operators are those defined above.

Limitations: The following are limits that apply to the functions performed by this instruction.

The source and receiver operands must both be specified as floating-point with the same length (4 bytes for short format or 8 bytes for long format).

Null substring references (a length value of zero) cannot be specified for this instruction.

Resultant Conditions: Positive, negative, or zero-The algebraic value of the receiver operand is positive, negative, or zero. Unordered-The value assigned to the floating-point result is NaN.

Exceptions

Exception		Operands 1	2	3	Other
06	Addressing				
	01 Spacing addressing violation	X	X	X	
	02 Boundary alignment violation	X	X	X	
	03 Range	X	X	X	
	06 Optimized addressability invalid	X	X	X	

Exception		Operands 1	2	3	Other
08	Argument/parameter				
	01 Parameter reference violation	X	X	X	
0C	Computation				
	06 Floating-point overflow	X			
	07 Floating-point underflow	X			
	09 Floating-point invalid operand			X	
	0D Floating-point inexact result	X			
10	Damage encountered				
	04 System object damage state				X
	44 Partial system object damage				X
1C	Machine-dependent exception				
	03 Machine storage limit exceeded				X
20	Machine support				
	02 Machine check				X
	03 Function check				X
22	Object access				
	01 Object not found	X	X	X	
	02 Object destroyed	X	X	X	
	03 Object suspended	X	X	X	
24	Pointer specification				
	01 Pointer does not exist	X	X	X	
	02 Pointer type invalid	X	X	X	
2A	Program creation				
	05 Invalid op-code extender field				X
	06 Invalid operand type	X	X	X	
	07 Invalid operand attribute	X	X	X	
	08 Invalid operand value range	X	X	X	
	09 Invalid branch target operand				X
	0C Invalid operand odt reference	X	X	X	
	0D Reserved bits are not zero	X	X	X	X
2C	Program execution				
	04 Invalid branch target				X
2E	Resource control limit				
	01 user profile storage limit exceeded				X
	02 Process storage limit exceeded				X
32	Scalar specification				
	01 Scalar type invalid	X	X	X	

Exception		Operands 1	2	3	Other
	03 Scalar value invalid		X		
36	Space management				
	01 space extension/truncation				X

1.13 Compute Math Function Using Two Input Values (CMF2)

Op Code (Hex)	Operand 1	Operand 2	Operand 3	Operand 4
100C	Receiver	Controls	Source 1	Source 2

Operand 1: Numeric variable scalar.

Operand 2: Character(2) scalar.

Operand 3: Numeric scalar.

Operand 4: Numeric scalar.

Optional Forms

Mnemonic	Op Code (Hex)	Form Type
CMF2I	180C	Indicator
CMF2B	1C0C	Branch

Extender: Branch options or indicator options.

If the branch or indicator option is specified in the op code, the extender field must be present along with one to four branch targets (for branch options) or one to four indicator operands (for indicator options). The branch or indicator operands immediately follow the last operand listed above. See Chapter 1. "Introduction" for the encoding of the extender field and the allowed syntax of the branch and indicator operands.

Description: The mathematical function, indicated by the controls operand, is performed on the source operand values and the result is placed in the receiver operand.

The calculation is always done in floating-point.

The controls operand must be a character scalar that specifies which mathematical function is to be performed. It must be at least 2 bytes in length and have the following format:

- Controls operand Char(2)
  - Hex 0001 = Power (x to the y)
  - All other values are reserved

The computation x power y, where x is the first source operand and y is the second source operand, is performed and the result is placed in the receiver operand.

The following chart shows some special cases for certain arguments of the power function (x**y). Within the chart, the capitalized letters X and Y refer to the absolute value of the arguments x and y; that is, X = |x| and Y = |y|.

y / x	-inf y= 2n+1	y<0, y=2n	y<0 real	y<0	-1	-1/2	+0 or -0	+1/2	+1	y>0 y= 2n+1	y>0 y=2n	y>0 real	+inf	M-NaN	UnM-NaN
+inf	+0	+0	+0	+0	+0	+1	+inf	+inf	+inf	+inf	+inf	+inf	b	A(c)	
x>1	+0	$\frac{+1}{x^{**}Y}$	$\frac{+1}{x^{**}Y}$	$\frac{+1}{x^{**}Y}$	$\frac{+1}{x}$	$\frac{+1}{SQRT(x)}$	+1	SQRT(x)	x	x**y	x**y	x**y	+inf	b	A(c)
x=+1	+1	+1	+1	+1	+1	+1	+1	+1	+1	+1	+1	+1	+1	b	A(c)
0<x<1	+inf	$\frac{+1}{x^{**}Y}$	$\frac{+1}{x^{**}Y}$	$\frac{+1}{x^{**}Y}$	$\frac{+1}{x}$	$\frac{+1}{SQRT(x)}$	+1	SQRT(x)	x	x**y	x**y	x**y	+0	b	A(c)
x=+0	E(f)	E(f)	E(f)	E(f)	E(f)	E(f)	+1	+0	+0	+0	+0	+0	+0	b	A(c)
x=-0	E(f)	E(g)	E(f)	E(f)	E(g)	E(g)	+1	-0	-0	-0	+0	+0	+0	b	A(c)
0>x>-1	A(a)	$\frac{-1}{X^{**}Y}$	$\frac{+1}{X^{**}Y}$	A(a)	$\frac{-1}{X}$	A(a)	+1	A(a)	x	-X**y	X**y	A(a)	A(a)	b	A(c)
x=-1	A(a)	-1	+1	A(a)	-1	A(a)	+1	A(a)	-1	-1	+1	A(a)	A(a)	b	A(c)
x<-1	A(a)	$\frac{-1}{X^{**}Y}$	$\frac{+1}{X^{**}Y}$	A(a)	$\frac{-1}{X}$	A(a)	+1	A(a)	x	-X**y	X**y	A(a)	A(a)	b	A(c)
x=-inf	A(a)	-0	+0	A(a)	-0	A(a)	+1	A(a)	-inf	-inf	+inf	A(a)	A(a)	b	A(c)
Masked NaN	b	b	b	b	b	b	b	b	b	b	b	b	b	d	A(e)
Un-masked NaN	A(c)	A(c)	A(c)	A(c)	A(c)	A(c)	A(c)	A(c)	A(c)	A(c)	A(c)	A(c)	A(c)	A(e)	A(e)

*Figure 1-3. Special cases of the power function (x**y)*

Capital letters in the chart indicate the exceptions and small letters indicate the returned results as defined in the following list:

A Floating-point invalid operand

E Divide by zero

a Result is the system default masked NaN

b Result is the same NaN

c Result is the same NaN masked

d Result is the larger NaN

e Result is the larger NAN masked

f Result is +infinity

g Result is -infinity

The following chart provides accuracy data for the mathematical function that can be invoked by this instruction.

Function Name	Sample Selection		Accuracy Data	
	x	y	MAX(e)	SD(e)
Power	1/3	-345 < = y < = 330	4.99 * 10**-16	1.90 * 10**-16
	.75	-1320 < = y < = 1320	2.96 * 10**-16	2.39 * 10**-16
	.9	-3605 < = y < = 3605	1.23 * 10**-16	1.02 * 10**-16
	10	-165 < = y < = 165	7.10 * 10**-16	3.18 * 10**-16
	712	-57 < = y < = 57	1.75 * 10**-15	7.24 * 10**-16

Figure 1-4. Accuracy data for CMF2 mathematical functions.

The vertical columns in the accuracy data chart have the following meanings:

- *Function Name*: This column identifies the mathematical function.
- *Sample Selection*: This column identifies the selection of samples taken for the power function. The algorithm used against the arguments, x and y, to gather the accuracy samples was a test for the anticipated relation where f(x) should equal g(x,y):

  where:

  ```
  f (x)= x
  g(x,y)= (x**y)**(1/y)
  ```

  An accuracy sample then, is an evaluation of the degree to which this relation held true.

  The range of argument values for x and y were selected such that x was held constant at a particular value and y was uniformly varied throughout a range of values which avoided overflowing or underflowing the result field. The particular values selected are indicated in the subcolumns entitled x and y.
- *Accuracy Data*: The maximum relative error and standard deviation (root mean square) of the relative error are generally useful and revealing statistics. These statistics for the relative error, (e), are provided in the following subcolumns:

  MAX(e) = MAX(ABS((f(x) - g(x)) / f(x)))

  where: MAX selects the largest of its arguments and ABS takes the absolute value of its argument.

  SD(e) = SQR((1/N) SUMSQ((f(x) - g(x)) / f(x)))

  where: SQR takes the square root of its argument and SUMSQ takes the summation of the squares of its arguments over all of the test cases.

Limitations: The following are limits that apply to the functions performed by this instruction.

The source and receiver operands must both be specified as floating-point with the same length (4 bytes for short format or 8 bytes for long format).

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for this instruction.

Resultant Conditions: Positive, negative, or zero-The algebraic value of the receiver operand is positive, negative, or zero. Unordered-The value assigned to the floating-point result is NaN.

Exceptions

Exception		Operands 1	2	3	4	Other
06	Addressing					
	01 space addressing violation	X	X	X	X	
	02 boundary alignment violation	X	X	X	X	
	03 range	X	X	X	X	
	06 optimized addressability invalid	X	X	X	X	
08	Argument/parameter					
	01 parameter reference violation	X	X	X	X	
0C	Computation					
	06 floating-point overflow	X				
	07 floating-point underflow	X				
	09 floating-point invalid operand			X	X	
	0C invalid floating-point conversion	X				
	0D floating-point inexact result	X				
	0E floating-point zero divide	X				
10	Damage encountered					
	04 System object damage state					X
	44 partial system object damage					X
1C	Machine-dependent exception					
	03 machine storage limit exceeded					X
20	Machine support					
	02 machine check					X
	03 function check					X
22	Object access					
	01 object not found	X	X	X	X	
	02 object destroyed	X	X	X	X	
	03 object suspended	X	X	X	X	
24	Pointer specification					
	01 pointer does not exist	X	X	X	X	
	02 pointer type invalid	X	X	X	X	
2A	Program creation					
	05 invalid op code extender field					X

Exception		Operands 1	2	3	4	Other
	06 invalid operand type	X	X	X	X	
	07 invalid operand attribute	X	X	X	X	
	08 invalid operand value range	X	X	X	X	
	09 invalid branch target operand					X
	0C invalid operand odt reference	X	X	X	X	
	0D reserved bits are not zero	X	X	X	X	X
2C	Program execution					
	04 invalid branch target					X
2E	Resource control limit					
	01 user profile storage limit exceeded					X
	02 process storage limit exceeded					X
32	Scalar specification					
	01 scalar type invalid	X	X	X	X	
	03 scalar value invalid		X			
36	Space management					
	01 space extension/truncation					X

1.14 Concatenate (CAT)

Op Code (Hex)	Operand 1	Operand 2	Operand 3
10F3	Receiver	Source 1	Source 2

Operand 1: Character variable scalar.

Operand 2: Character scalar.

Operand 3: Character scalar.

Description: The character string value of the second source operand is joined to the right end of the character string value of the first source operand. The resulting string value is placed (left-adjusted) in the receiver operand.

The length of the operation is equal to the length of the receiver operand with the resulting string truncated or is logically padded on the right end accordingly. The pad value for this instruction is hex 40.

Substring operand references that allow for a null substring reference (a length value of zero) may be specified for operands 1, 2, and 3. The effect of specifying a null substring reference for one source operand is that the other source operand is used as the result of the concatenation. The effect of specifying a null substring reference for both source operands is that the bytes of the receiver are each set with a value of hex 40. The effect of specifying a null substring reference for the receiver is that a result is not set regardless of the value of the source operands.

Exceptions

Exception		Operands 1	2	3	Other
06	Addressing				
	01 space addressing violation	X	X	X	
	02 boundary alignment	X	X	X	
	03 range	X	X	X	
	06 optimized addressability invalid	X	X	X	
08	Argument/parameter				
	01 parameter reference violation				X
10	Damage encountered				
	04 system object damage state	X	X	X	X
	44 partial system object damage	X	X	X	X
1C	Machine-dependent exception				
	03 machine storage limit exceeded				X
20	Machine support				
	02 machine check				X
	03 function check				X

Exception		Operands 1	2	3	Other
22	Object access				
	01 object not found	X	X	X	
	02 object destroyed	X	X	X	
	03 object suspended	X	X	X	
24	Pointer specification				
	01 pointer does not exist	X	X	X	
	02 pointer type invalid	X	X	X	
2A	Program creation				
	06 invalid operand type	X	X	X	
	07 invalid operand attribute	X	X	X	
	08 invalid operand value range	X	X	X	
	0A invalid operand length	X	X	X	
	0C invalid operand odt reference	X	X	X	
	0D reserved bits are not zero	X	X	X	X
2E	Resource control limit				
	01 user profile storage limit exceeded				X
36	Space management				
	01 space extension/truncation				X

1.15 Convert BSC to Character (CVTBC)

Op Code (Hex)	Operand 1	Operand 2	Operand 3
10AF	Receiver	Controls	Source

Operand 1: Character variable scalar.

Operand 2: Character(3) variable scalar (fixed-length).

Operand 3: Character scalar.

Optional Forms

Mnemonic	Op Code (Hex)	Form Type
CVTBCI	18AF	Indicator
CVTBCB	1CAF	Branch

Extender: Branch options or indicator options.

If the branch or indicator option is specified in the op code, the extender field must be present along with one to three branch targets (for branch options) or one to three indicator operands (for indicator options). The branch or indicator operands immediately follow the last operand listed above. See Chapter 1. "Introduction" for the encoding of the extender field and the allowed syntax of the branch and indicator operands.

Description: This instruction converts a string value from the BSC (binary synchronous communications) compressed format to a character string. The operation converts the source (operand 3) from the BSC compressed format to character under control of the controls (operand 2) and places the result into the receiver (operand 1).

The source and receiver operands must both be character strings.

The controls operand must be a character scalar that specifies additional information to be used to control the conversion operation. It must be at least 3 bytes in length and have the following format:

- Controls operand — Char(3)
  - — Source offset — Bin(2)
  - — Record separator — Char(1)

The source offset specifies the offset where bytes are to be accessed from the source operand. If the offset is equal to or greater than the length specified for the source operand (it identifies a byte beyond the end of the source operand), a template value invalid exception is signaled. As output from the instruction, the source offset is set to specify the offset that indicates how much of the source is processed when the instruction ends.

The record separator, if specified with a value other than hex 01, contains the value used to separate converted records in the source operand. A value of hex

01 specifies that record separators do not occur in the converted records in the source.

Only the first 3 bytes of the controls operand are used. Any excess bytes are ignored.

The operation begins by accessing the bytes of the source operand located at the offset specified in the source offset. This is assumed to be the start of a record. The bytes of the record in the source operand are converted into the receiver record according to the following algorithm.

The strings to be built in the receiver are contained in the source as blank compression entries and strings of consecutive nonblank characters.

The format of the blank compression entries occurring in the source are as follows:

- Blank compression entry — Char(2)
  - Interchange group separator — Char(1)
  - Count of compressed blanks — Char(1)

The interchange group separator has a fixed value of hex 1D.

The compressed blanks count provides for describing up to 63 compressed blanks. The count of the number of blanks (up to 63) to be decompressed is formed by subtracting hex 40 from the value of the count field. The count field can vary from a value of hex 41 to hex 7F. If the count field contains a value outside of this range, a conversion exception is signaled.

Strings of blanks described by blank compression entries in the source are repeated in the receiver the number of times specified by the blank compression count.

Nonblank strings in the source are copied into the receiver intact with no alteration.

If the receiver record is filled with converted data without encountering the end of the source operand, the instruction ends with a resultant condition of *completed record*. This can occur in two ways. If a record separator was not specified, the instruction ends when enough bytes have been converted from the source to fill the receiver. If a record separator was specified, the instruction ends when a source byte is encountered with that value prior to or just after filling the receiver record. The offset value for the source locates the byte following the last source record (including the record separator) for which conversion was completed. When the record separator value is encountered, any remaining bytes in the receiver are padded with blanks.

If the end of the source operand is encountered (whether or not in conjunction with a record separator or the filling of the receiver), the instruction ends with a resultant condition of *source exhausted*. The offset value for the source locates the byte following the last byte of the source operand. The remaining bytes in the receiver after the converted record are padded with blanks.

If the converted form of a record cannot be completely contained in the receiver, the instruction ends with a resultant condition of *truncated record*. The offset value for the source locates the byte following the last source byte for which conversion was performed, unless a blank compression entry was being processed. In this case, the source offset is set to locate the byte after the blank compression entry. If the source does not contain record separators, this condition can only occur for the case in which a blank compression entry was being converted when the receiver record became full.

Any form of overlap between the operands on this instruction yields unpredictable results in the receiver operand.

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for this instruction.

Resultant Conditions: Completed record-The receiver record has been completely filled with converted data from a source record. Source exhausted-All of the bytes in the source operand have been converted into the receiver operand. Truncated record-The receiver record cannot contain all of the converted data from the source record.

Exceptions

Exception		Operands 1	2	3	Other
06	Addressing				
	01 space addressing violation	X	X	X	
	02 boundary alignment violation	X	X	X	
	03 range	X	X	X	
	06 optimized addressability invalid	X	X	X	
08	Argument/parameter				
	01 parameter reference violation	X	X	X	
0C	Computation				
	01 conversion			X	
10	Damage encountered				
	04 System object damage state				X
	44 partial system object damage				X
1C	Machine-dependent exception				
	03 machine storage limit exceeded				X
20	Machine support				
	02 machine check				X
	03 function check				X
22	Object access				
	01 object not found	X	X	X	
	02 object destroyed	X	X	X	
	03 object suspended	X	X	X	

Exception		Operands 1	2	3	Other
24	Pointer specification				
	01 pointer does not exist	X	X	X	
	02 pointer type invalid	X	X	X	
2A	Program creation				
	05 invalid op code extender field				X
	06 invalid operand type	X	X	X	
	07 invalid operand attribute	X	X	X	
	08 invalid operand value range	X	X	X	
	09 invalid branch target operand				X
	0A invalid operand length		X		
	0C invalid operand odt reference	X	X	X	
	0D reserved bits are not zero	X	X	X	X
2C	Program execution				
	04 invalid branch target				X
2E	Resource control limit				
	01 user profile storage limit exceeded				X
32	Scalar specification				
	01 scalar type invalid	X	X	X	
36	Space management				
	01 space extension/truncation				X
38	Template specification				
	01 template value invalid		X		

1.16 Convert Character to BSC (CVTCB)

Op Code (Hex)	Operand 1	Operand 2	Operand 3
108F	Receiver	Controls	Source

Operand 1: Character variable scalar.

Operand 2: Character(3) variable scalar (fixed-length).

Operand 3: Character scalar.

Optional Forms

Mnemonic	Op Code (Hex)	Form Type
CVTCBI	188F	Indicator
CVTCBB	1C8F	Branch

Extender: Branch options or indicator options.

If the branch or indicator option is specified in the op code, the extender field must be present along with one or two branch targets (for branch options) or one or two indicator operands (for indicator options). The branch or indicator operations immediately follow the last operand listed above. See Chapter 1. "Introduction" for the encoding of the extender field and the allowed syntax of the branch and indicator operands.

Description: This instruction converts a string value from character to BSC (binary synchronous communications) compressed format. The operation converts the source (operand 3) from character to the BSC compressed format under control of the controls (operand 2) and places the result into the receiver (operand 1).

The source and receiver operands must both be character strings.

The controls operand must be a character scalar that specifies additional information to be used to control the conversion operation. It must be at least 3 bytes in length and have the following format:

- Controls operand — Char(3)
  - — Receiver offset — Bin(2)
  - — Record separator — Char(1)

The receiver offset specifies the offset where bytes are to be placed into the receiver operand. If the offset is equal to or greater than the length specified for the receiver operand (it identifies a byte beyond the end of the receiver), a template value invalid exception is signaled. As output from the instruction, the source offset is set to specify the offset that indicates how much of the receiver has been filled when the instruction ends.

The record separator, if specified with a value other than hex 01, contains the value used to separate converted records in the receiver operand. A value of

hex 01 specifies that record separators are not to be placed into the receiver to separate converted records.

Only the first 3 bytes of the controls operand are used. Any excess bytes are ignored.

The source operand is assumed to be one record. The bytes of the record in the source operand are converted into the receiver operand at the location specified in the receiver offset according to the following algorithm.

The bytes of the source record are interrogated to identify the strings of consecutive blank (hex 40) characters and the strings of consecutive nonblank characters which occur in the source record. Only three or more blank characters are treated as a blank string for purposes of conversion into the receiver.

As the blank and nonblank strings are encountered in the source they are packaged into the receiver.

Blank strings are reflected in the receiver as one or more blank compression entries. The format of the blank compression entries built into the receiver are as follows:

- Blank compression entry — Char(2)
  - Interchange group separator — Char(1)
  - Count of compressed blanks — Char(1)

The interchange group separator has a fixed value of hex 1D.

The compressed blanks count provides for compressing up to 63 blanks. The value of the count field is formed by adding hex 40 to the actual number of blanks (up to 63) to be compressed. The count field can vary from a value of hex 43 to hex 7F.

Nonblank strings are copied into the receiver intact with no alteration or additional control information.

When the end of the source record is encountered the record separator value if specified is placed into the receiver and the instruction ends with a resultant condition of *source exhausted*. The offset value for the receiver locates the byte following the converted record in the receiver. The value of the remaining bytes in the receiver after the converted record is unpredictable.

If the converted form of a record cannot be completely contained in the receiver (including the record separator if specified), the instruction ends with a resultant condition of *receiver overrun*. The offset value for the receiver remains unchanged. The remaining bytes in the receiver, starting with the byte located by the receiver offset, are unpredictable.

Any form of overlap between the operands on this instruction yields unpredictable results in the receiver operand.

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for this instruction.

Resultant Conditions: Source exhausted-All of the bytes in the source operand have been converted into the receiver operand. Receiver overrun-An overrun condition in the receiver operand was detected before all of the bytes in the source operand were processed.

Exceptions

Exception		Operands 1	2	3	Other
06	Addressing				
	01 space addressing violation	X	X	X	
	02 boundary alignment violation	X	X	X	
	03 range	X	X	X	
	06 optimized addressability invalid	X	X	X	
08	Argument/parameter				
	01 parameter reference violation	X	X	X	
10	Damage encountered				
	04 System object damage state				X
	44 partial system object damage				X
1C	Machine-dependent exception				
	03 machine storage limit exceeded				X
20	Machine support				
	02 machine check				X
	03 function check				X
22	Object access				
	01 object not found	X	X	X	
	02 object destroyed	X	X	X	
	03 object suspended	X	X	X	
24	Pointer specification				
	01 pointer does not exist	X	X	X	
	02 pointer type invalid	X	X	X	
2A	Program creation				
	05 invalid op code extender field				X
	06 invalid operand type	X	X	X	
	07 invalid operand attribute	X	X	X	
	08 invalid operand value range	X	X	X	
	09 invalid branch target operand				X
	0A invalid operand length		X		
	0C invalid operand odt reference	X	X	X	
	0D reserved bits are not zero	X	X	X	X
2C	Program execution				
	04 invalid branch target				X

Exception		Operands 1	2	3	Other
2E	Resource control limit				
	01 user profile storage limit exceeded				X
32	Scalar specification				
	01 scalar type invalid	X	X	X	
36	Space management				
	01 space extension/truncation				X
38	Template specification				
	01 template value invalid		X		

1.17 Convert Character to Hex (CVTCH)

Op Code (Hex)	Operand 1	Operand 2
1082	Receiver	Source

Operand 1: Character variable scalar.

Operand 2: Character variable scalar.

Description: Each character (8-bit value) of the string value in the source operand is converted to a hex digit (4-bit value) and placed in the receiver operand. The source operand characters must relate to valid hex digits or a conversion exception is signaled.

Characters **Hex Digits**

Hex F0-hex F9 = Hex 0-hex 9

Hex C1-hex C6 = Hex A-hex F

The operation begins with the two operands left-adjusted and proceeds left to right until all the hex digits of the receiver operand have been filled. If the source operand is too small, it is logically padded on the right with zero characters (hex F0). If the source operand is too large, a length conformance or an invalid operand length exception is signaled.

Substring operand references that allow for a null substring reference (a length value of zero) may be specified for operands 1 and 2. The effect of specifying a null substring reference for the source is that the bytes of the receiver are each set with a value of hex 00. The effect of specifying a null substring reference for the receiver is that no result is set.

Exceptions

Exception		Operands 1	2	Other
06	Addressing			
	01 space addressing violation	X	X	
	02 boundary alignment	X	X	
	03 range	X	X	
	06 optimized addressability invalid	X	X	
08	Argument/parameter			
	01 parameter reference violation	X	X	
0C	Computation			
	01 conversion		X	
	C8 length conformance	X		
10	Damage encountered			
	04 system object damage	X	X	X
	44 partial system object damage	X	X	X
1C	Machine-dependent exception			

Exception		Operands 1	2	Other
	03 machine storage limit exceeded			X
20	Machine support			
	02 machine check			X
	03 function check			X
22	Object access			
	01 object not found	X	X	
	02 object destroyed	X	X	
	03 object suspended	X	X	
24	Pointer specification			
	01 pointer does not exist	X	X	
	02 pointer type invalid	X	X	
2A	Program creation			
	06 invalid operand type	X	X	
	07 invalid operand attribute	X	X	
	08 invalid operand value range	X	X	
	0A invalid operand length	X		
	0C invalid operand odt reference	X	X	
	0D reserved bits are not zero	X	X	X
2E	Resource control limit			
	01 user profile storage limit exceeded			X
36	Space management			
	01 space extension/truncation			X

1.18 Convert Character to MRJE (CVTCM)

Op Code (Hex)	Operand 1	Operand 2	Operand 3
108B	Receiver	Controls	Source

Operand 1: Character variable scalar.

Operand 2: Character(13) variable scalar (fixed-length).

Operand 3: Character scalar.

Optional Forms

Mnemonic	Op Code (Hex)	Form Type
CVTCMI	188B	Indicator
CVTCMB	1C8B	Branch

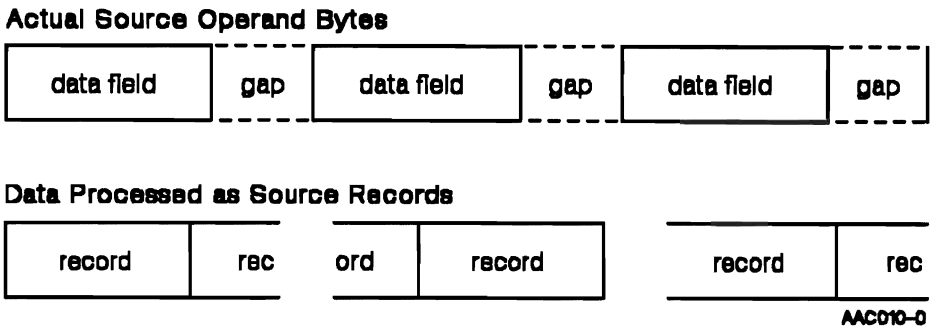
Extender: Branch options or indicator options.

If the branch or indicator option is specified in the op code, the extender field must be present along with one or two branch targets (for branch options) or one or two indicator operands (for indicator options). The branch or indicator operands immediately follow the last operand listed above. See Chapter 1. "Introduction" for the encoding of the extender field and the allowed syntax of the branch and indicator operands.

Description: This instruction converts a string of characters to MRJE (MULTI-LEAVING remote job entry) compressed format. The operation converts the source (operand 3) from character to the MRJE compressed format under control of the controls (operand 2) and places the results in the receiver (operand 1).

The source and receiver operands must both be character strings. The source operand cannot be specified as either a signed or unsigned immediate value.

The source operand can be described through the controls operand as being composed of one or more fixed length data fields, which may be separated by fixed length gaps of characters to be ignored during the conversion operation. Additionally, the controls operand specifies the amount of data to be processed from the source to produce a converted record in the receiver. This may be a different value than the length of the data fields in the source. The following diagram shows this structure for the source operand.



The controls operand must be a character scalar that specifies additional information to be used to control the conversion operation. It must be at least 13 bytes in length and have the following format:

• Controls operand	Char(13)
— Offset into the receiver operand	Bin(2)
— Offset into the source operand	Bin(2)
— Algorithm modifier	Char(1)
— Source record length	Char(1)
— Data field length	Bin(2)
— Offset to next gap in source operand	Bin(2)
— Gap length	Bin(2)
— Record control block (RCB) value	Char(1)

As input to the instruction, the source and receiver offset fields specify the offsets where bytes of the source and receiver operands are to be processed. If an offset is equal to or greater than the length specified for the operand it corresponds to (it identifies a byte beyond the end of the operand), a template value invalid exception is signaled. As output from the instruction, the source and receiver offset fields specify offsets that indicate how much of the operation is complete when the instruction ends.

The algorithm modifier has the following valid values:

- Hex 00 = Perform full compression.
- Hex 01 = Perform only truncation of trailing blanks.

The source record length value specifies the amount of data from the source to be processed. If a source record length of 0 is specified, a template value invalid exception is signaled.

The data field length value specifies the length of the data fields in the source. Data fields occurring in the source may be separated by gaps of characters, which are to be ignored during the conversion operation. Specification of a data field length of 0 indicates that the source operand is one data field. In this case, the gap length and gap offset values have no meaning and are ignored.

The gap offset value specifies the offset to the next gap in the source. This value is both input to and output from the instruction. This is relative to the current byte to be processed in the source as located by the source offset value. No

validation is done for this offset. It is assumed to be valid relative to the source operand. The gap offset value is ignored if the data field length is specified with a value of 0.

The gap length value specifies the amount of data occurring between data fields in the source operand which is to be ignored during the conversion operation. The gap length value is ignored if the data field length is specified with a value of 0.

The record control block (RCB) field specifies the RCB value that is to precede the converted form of each record in the receiver. It can have any value.

Only the first 13 bytes of the controls operand are used. Any excess bytes are ignored.

The operation begins by accessing the bytes of the source operand at the location specified by the source offset. This is assumed to be the start of a source record. Only the bytes of the data fields in the source are accessed for conversion purposes. Gaps between data fields are ignored, causing the access of data field bytes to occur as if the data fields were contiguous with one another. Bytes accessed from the source for the source record length are considered a source record for the conversion operation. They are converted into the receiver operand at the location specified by the receiver offset according to the following algorithm.

The RCB value is placed into the first byte of the receiver record.

An SRCB (sub record control byte) value of hex 80 is placed into the second byte of the receiver record.

If the algorithm modifier specifies full compression (a value of hex 00) then:

The bytes of the source record are interrogated to locate the blank character strings (2 or more consecutive blanks), identical character strings (3 or more consecutive identical characters), and nonidentical character strings occurring in the source. A blank character string occurring at the end of the record is treated as a special case (see following information on trailing blanks).

If the algorithm modifier specifies blank truncation (a value of hex 01) then:

The bytes of the source record are interrogated to determine if a blank character string exists at the end of the source record. If one exists, it is treated as a string of trailing blanks. All characters prior to it in the record are treated as one string of nonidentical characters.

The strings encountered (blank, identical, or nonidentical) are reflected in the receiver by building one or more SCBs (string control bytes) in the receiver to describe them.

The format of the SCBs built into the receiver is:

- SCB format is `o k l jjjjj`

The bit meanings are:

Bit	Value	Meaning
o	0	End of record; the EOR SCB is hex 00.
	1	All other SCBs.
k	0	The string is compressed.
	1	The string is not compressed.
1		For k = 0:
	0	Blanks (hex 40s) have been deleted.
	1	Nonblank characters have been deleted. The next character in the data stream is the specimen character.
		For k = 1:
		This bit is part of the length field for length of uncompressed data.
jjjjj		Number of characters that have been deleted if k = 0. The value can be 2-31.
1jjjjj		Number of characters to the next SCB (no compression) if k = 1. The value can be 1-63. The uncompressed (nonidentical bytes) follow the SCB in the data stream.

When the end of a source record is encountered, an EOR (end of record) SCB (hex 00) is built into the receiver. Trailing blanks in a record including a record of all blanks are represented in the receiver by an EOR character if either full compression or trailing blank truncation is specified.

If the end of the source operand is not encountered, the operation then continues by reapplying the above algorithm to the next record in the source operand.

If the end of the source operand is encountered (whether or not in conjunction with a record boundary), the instruction ends with a resultant condition of *source exhausted*. The offset value for the source locates the byte following the last source record for which conversion was completed. The gap offset value indicates the offset to the next gap relative to the source offset value set for this condition. The gap offset value has no meaning and is not set when the data field length is 0. The offset value for the receiver locates the byte following the last fully converted record in the receiver. The value of the remaining bytes in the receiver after the last converted record is unpredictable.

If the converted form of a record cannot be completely contained in the receiver, the instruction ends with a resultant condition of receiver overrun. The offset value for the source locates the byte following the last source record for which conversion was completed. The gap offset value indicates the offset to the next gap relative to the source offset value set for this condition. The gap offset value has no meaning and is not set when the data field length is 0. The offset value for the receiver locates the byte following the last fully converted record in the receiver. The value of the remaining bytes in the receiver after the last converted record is unpredictable.

Any form of overlap between the operands of this instruction yields unpredictable results in the receiver operand.

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for this instruction.

Resultant Conditions: Source exhausted-All complete records in the source operand have been converted into the receiver operand. Receiver overrun-An overrun condition in the receiver operand was detected prior to processing all of the bytes in the source operand.

If source exhausted and receiver overrun occur at the same time, the source exhausted condition is recognized first. When source exhausted is the resultant condition, the receiver may also be full. In this case, the offset into the receiver may contain a value equal to the length specified for the receiver, and this condition will cause an exception on the next invocation of the instruction. The processing performed for the source exhausted condition provides for this case when the instruction is invoked multiple times with the same controls operand template. When the receiver overrun condition is the resultant condition, the source always contains data that can be converted.

Exceptions

Exception		Operands 1	2	3	Other
06	Addressing				
	01 space addressing violation	X	X	X	
	02 boundary alignment violation	X	X	X	
	03 range	X	X	X	
	06 optimized addressability invalid	X	X	X	
08	Argument/parameter				
	01 parameter reference violation	X	X	X	
10	Damage encountered				
	04 System object damage state				X
	44 partial system object damage				X
1C	Machine-dependent exception				
	03 machine storage limit exceeded				X
20	Machine support				
	02 machine check				X
	03 function check				X
22	Object access				
	01 object not found	X	X	X	
	02 object destroyed	X	X	X	
	03 object suspended	X	X	X	
24	Pointer specification				
	01 pointer does not exist	X	X	X	
	02 pointer type invalid	X	X	X	
2A	Program creation				
	05 invalid op code extender field				X
	06 invalid operand type	X	X	X	

Exception		Operands 1	2	3	Other
	07 invalid operand attribute	X	X	X	
	08 invalid operand value range	X	X	X	
	09 invalid branch target operand	X	X	X	
	0A invalid operand length		X		
	0C invalid operand odt reference	X	X	X	
	0D reserved bits are not zero	X	X	X	X
2C	Program execution				
	04 invalid branch target				X
2E	Resource control limit				
	01 user profile storage limit exceeded				X
32	Scalar specification				
	01 scalar type invalid	X	X	X	
36	Space management				
	01 space extension/truncation				X
38	Template specification				
	01 template value invalid			X	

1.19 Convert Character to Numeric (CVTCN)

Op Code (Hex)	Operand 1	Operand 2	Operand 3
1083	Receiver	Source	Attributes

Operand 1: Numeric variable scalar or data-pointer-defined numeric scalar.

Operand 2: Character scalar or data-pointer-defined character scalar.

Operand 3: Character(7) scalar or data-pointer-defined character scalar.

Description: The character scalar specified by operand 2 is treated as though it were a numeric scalar with the attributes specified by operand 3. The character string source operand is converted to the numeric forms of the receiver operand and moved to the receiver operand. The value of operand 2, when viewed in this manner, is converted to the type, length, and precision of the numeric receiver, operand 1, following the rules for the Copy Numeric Value instruction.

The length of operand 2 must be large enough to contain the numeric value described by operand 3. If it is not large enough, a scalar value invalid exception is signaled. If it is larger than needed, its leftmost bytes are used as the value, and the rightmost bytes are ignored.

Normal rules of arithmetic conversion apply except for the following. If operand 2 is interpreted as a zoned decimal value, a value of hex 40 in the rightmost byte referenced in the conversion is treated as a positive sign and a zero digit.

If a decimal to binary conversion causes a size exception to be signaled, the binary value contains the correct truncated result only if the decimal value contains 15 or fewer significant nonfractional digits.

The format of the attribute operand specified by operand 3 is as follows:

- Scalar attributes — Char(7)
  - Scalar type — Char(1)

    Hex 00 = Signed binary
    Hex 01 = Floating-point
    Hex 02 = Zoned decimal
    Hex 03 = Packed decimal
    Hex 0A = Unsigned binary
  - Scalar length — Bin(2)

    If binary:

    Length (L) (where L = 2 or 4) — Bits 0-15

    If floating-point:

    Length (L) (where L = 4 or 8) — Bits 0-15

    If zoned decimal or packed decimal:

    Fractional digits (F) — Bits 0-7

    Total digits (T) — Bits 8-15
    (where $1 \leq T \leq 31$ and $0 \leq F \leq T$)

— Reserved (binary 0) Bin(4)

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for this instruction.

Exceptions

Exception		Operands 1	2	3	Other
06	Addressing				
	01 space addressing violation	X	X	X	
	02 boundary alignment	X	X	X	
	03 range	X	X	X	
	04 external data object not found	X	X	X	
	06 optimized addressability invalid	X	X	X	
08	Argument/parameter				
	01 parameter reference violation	X	X	X	
0C	Computation				
	02 decimal data		X	X	
	06 floating-point overflow	X			
	07 floating-point underflow	X			
	09 floating-point invalid operand		X		
	0A size	X			
	0C floating-point conversion	X			
	0D floating-point inexact result	X			
10	Damage encountered				
	04 system object damage state	X	X	X	X
	44 partial system object damage	X	X	X	X
1C	Machine-dependent exception				
	03 machine storage limit exceeded				X
20	Machine support				
	02 machine check				X
	03 function check				X
22	Object access				
	01 object not found	X	X	X	
	02 object destroyed	X	X	X	
	03 object suspended	X	X	X	
24	Pointer specification				
	01 pointer does not exist	X	X	X	
	02 pointer type invalid	X	X	X	
2A	Program creation				
	06 invalid operand type	X	X	X	

Exception		Operands 1	2	3	Other
	07 invalid operand attribute	X	X	X	
	08 invalid operand value range	X	X	X	
	0A invalid operand length		X	X	
	0C invalid operand odt reference	X	X	X	
	0D reserved bits are not zero	X	X	X	X
2E	Resource control limit				
	01 user profile storage limit exceeded				X
32	Scalar specification				
	01 scalar type invalid	X	X	X	
	02 scalar attribute invalid			X	
	03 scalar value invalid			X	
36	Space management				
	01 space extension/truncation				X

1.20 Convert Character to SNA (CVTCS)

Op Code (Hex)	Operand 1	Operand 2	Operand 3
10CB	Receiver	Controls	Source

Operand 1: Character variable scalar.

Operand 2: Character(15) variable scalar (fixed length).

Operand 3: Character scalar.

Optional Forms

Mnemonic	Op Code (Hex)	Form Type
CVTCSI	18CB	Indicator
CVTCSB	1CCB	Branch

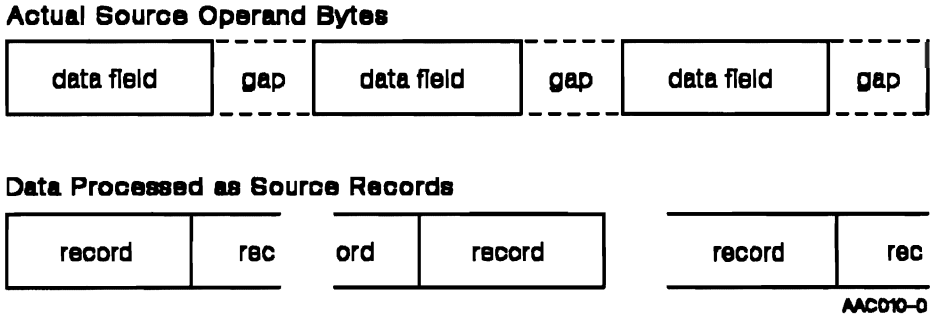
Extender: Branch options or indicator options.

If the branch or indicator option is specified in the op code, the extender field must be present along with one or two branch targets (for branch options) or one or two indicator operands (for indicator options). The branch or indicator operands immediately follow the last operand listed above. See Chapter 1. "Introduction" for the encoding of the extender field and the allowed syntax of the branch and indicator operands.

Description: This instruction converts the source (operand 3) from character to SNA (systems network architecture) format under control of the controls (operand 2) and places the result into the receiver (operand 1).

The source and receiver operands must both be character strings. The source operand may not be specified as an immediate operand.

The source operand can be described by the controls operand as being one or more fixed-length data fields that may be separated by fixed-length gaps of characters to be ignored during the conversion operation. Additionally, the controls operand specifies the amount of data to be processed from the source to produce a converted record in the receiver. This may be a different value than the length of the data fields in the source. The following diagram shows this structure for the source operand.



The controls operand must be a character scalar that specifies additional information to be used to control the conversion operation. The operand must be at least 15 bytes in length and has the following format:

- Controls operand — Char(15)
  - — Offset into the receiver operand — Bin(2)
  - — Offset into the source operand — Bin(2)
  - — Algorithm modifier — Char(1)
  - — Source record length — Char(1)
  - — Data field length — Bin(2)
  - — Gap offset — Bin(2)
  - — Gap length — Bin(2)
  - — Record separator character — Char(1)
  - — Prime compression character — Char(1)
  - — Unconverted source record bytes — Char(1)

When the source and receiver operands are input to the instruction, they specify the offsets where the bytes of the source and receiver operands are to be processed. If an offset is equal to or greater than the length specified for the operand, the offset identifies a byte beyond the end of the operand and a template value invalid exception is signaled. When the source and the receiver are output from the instruction, they specify offsets that indicate how much of the operation is complete when the instruction ends.

The algorithm modifier specifies the optional functions to be performed. Any combination of functions can be specified as indicated by the bit meanings in the following chart. At least one of the functions must be specified. If all of the algorithm modifier bits are zero, a template value invalid exception is signaled. The algorithm modifier bit meanings are:

Bits	Meaning
0	0 = Do not perform compression. 1 = Perform compression.
1-2	00 = Do not use record separators and no blank truncation. Do not perform data transparency conversion. 01 = Reserved. 10 = Use record separators and perform blank truncation. Do not perform data transparency conversion. 11 = Use record separators and perform blank truncation. Perform data transparency conversion.
3	0 = Do not perform record spanning. 1 = Perform record spanning. (allowed only when bit 1 = 1)
4-7	(Reserved)

The source record length value specifies the amount of data from the source to be processed to produce a converted record in the receiver. Specification of a source record length of zero results in a template value invalid exception.

The data field length value specifies the length of the data fields in the source. Data fields occurring in the source may be separated by gaps of characters that are to be ignored during the conversion operation. Specification of a data field

length of zero indicates that the source operand is one data field. In this case, the gap length and gap offset values have no meaning and are ignored.

The gap offset value specifies the offset to the next gap in the source. This value is both input to and output from the instruction. This is relative to the current byte to be processed in the source as located by the source offset value. No validation is done for this offset. It is assumed to be valid relative to the source operand. The gap offset value is ignored if the data field length is specified with a value of zero.

The gap length value specifies the amount of data that is to be ignored between data fields in the source operand during the conversion operation. The gap length value is ignored if the data field length is zero.

The record separator character value specifies the character that precedes the converted form of each record in the receiver. It also serves as a delimiter when the previous record is truncating trailing blanks. The Convert SNA to Character instruction recognizes any value that is less than hex 40. The record separator value is ignored if record separators are not used as specified in the algorithm modifier.

The prime compression character value specifies the character to be used as the prime compression character when performing compression of the source data to SNA format. It may have any value. The prime compression character value is ignored if the compression function is not specified in the algorithm modifier.

The unconverted source record bytes value specifies the number of bytes remaining in the current source record that are yet to be converted.

When the record spanning function is specified in the algorithm modifier, the unconverted source record bytes value is both input to and output from the instruction. On input, a value of hex 00 means it is the start of a new record and the initial conversion step is yet to be performed. That is, a record separator character has not yet been placed in the receiver. On input, a nonzero value less than or equal to the record length specifies the number of bytes remaining in the current source record that are yet to be converted into the receiver. This value is assumed to be the valid count of unconverted source record bytes relative to the current byte to be processed in the source as located by the source offset value. As such, it is used to determine the location of the next record boundary in the source operand. This value must be less than or equal to the source record length value; otherwise, a template value invalid exception is signaled. On output this field is set with a value as defined above that describes the number of bytes of the current source record that have not yet been converted.

When the record spanning function is not specified in the algorithm modifier, the unconverted source record bytes value is ignored.

Only the first 15 bytes of the controls operand are used. Any excess bytes are ignored.

The description of the conversion process is presented as a series of separately performed steps that may be selected in allowable combinations to accomplish the conversion function. It is presented this way to allow for describing these

functions separately. However, in the actual execution of the instruction, these functions may be performed in conjunction with one another or separately depending upon which technique is determined to provide the best implementation.

The operation is performed either on a record-by-record basis, record processing, or on a nonrecord basis, string processing. This is determined by the functions selected in the algorithm modifier. Specifying the use record separators and do blank truncation function indicates record processing is to be performed. If this is not specified, in which case compression must be specified, it indicates that string processing is to be performed.

The operation begins by accessing the bytes of the source operand at the location specified by the source offset.

When record processing is specified, the source offset may locate the start of a full or partial record.

When the record spanning function has not been specified in the algorithm modifier, the source offset is assumed to locate the start of a record.

When the record spanning function has been specified in the algorithm modifier, the source offset is assumed to locate a point at which processing of a possible partially converted record is to be resumed. In this case the unconverted source record bytes value contains the length of the remaining portion of the source record to be converted. The conversion process in this case is started by completing the conversion of the current source record before processing the next full source record.

When string processing is specified, the source offset locates the start of the source string to be converted.

Only the bytes of the data fields in the source are accessed for conversion purposes. Gaps between data fields are ignored causing the access of data field bytes to occur as if the data fields were contiguous. A string of bytes accessed from the source for a length equal to the source record length is considered to be a record for the conversion operation.

When during the conversion process the end of the source operation is encountered, the instruction ends with a resultant condition of source exhausted.

When record processing is specified in the algorithm modifier, this check is performed at the start of conversion for each record. If the source operand does not contain a full record, the source exhausted condition is recognized. The instruction is terminated with status in the controls operand describing the last completely converted record. For source exhausted, partial conversion of a source record is not performed.

When string processing is specified in the algorithm modifier, then compression must be specified and the compression function described below defines the detection of source exhausted.

If the converted form of the source cannot be completely contained in the receiver, the instruction ends with a resultant condition of receiver overrun. See the description of this condition in the conversion process described below to

determine the status of the controls operand values and the converted bytes in the receiver for each case.

When string processing is specified, the bytes accessed from the source are converted on a string basis into the receiver operand at the location specified by the receiver offset. In this case, the compression function must be specified and the conversion process proceeds with the compression function defined below.

When record processing is specified, the bytes accessed from the source are converted one record at a time into the receiver operand at the location specified by the receiver offset performing the functions specified in the algorithm modifier in the sequence defined by the following algorithm.

The first function performed is trailing blank truncation.

A truncated record is built by logically appending the record data to the record separator value specified in the controls operand and removing all blank characters after the last nonblank character in the record. If a record has no trailing blanks, then no actual truncation takes place. A null record, a record consisting entirely of blanks, will be converted as just the record separator character with no other data following it. The truncated record then consists of the record separator character followed by the truncated record data, the full record data, or no data from the record.

If either the data transparency conversion or the compression function is specified in the algorithm modifier, the conversion process continues for this record with the next specified function.

If not, the conversion process for this record is completed by placing the truncated record into the receiver. If the truncated record cannot be completely contained in the receiver, the instruction ends with a resultant condition of receiver overrun. When the record spanning function is specified in the algorithm modifier, as much of the truncated record as will fit is placed into the receiver and the controls operand is updated to describe how much of the source record was successfully converted into the receiver. When the record spanning function is not specified in the algorithm modifier, the controls operand is updated to describe only the last fully converted record in the receiver and the value of the remaining bytes in the receiver is unpredictable.

The second function performed is data transparency conversion.

Data transparency conversion is performed if the function is specified in the algorithm modifier. This provides for making the data in a record transparent to the Convert SNA to Character instruction in the area of its scanning for record separator values. Transparent data is built by preceding the data with 2 bytes of transparency control information. The first byte has a fixed value of hex 35 and is referred to as the TRN (transparency) control character. The second byte is a 1-byte hexadecimal count, a value ranging from 1 to 255 decimal, of the number of bytes of data that follow and is referred to as the TRN count. This contains the length of the data and does not include the TRN control information length.

Transparency conversion can be specified only in conjunction with record processing and, as such, is performed on the truncated form of the source record. The transparent record is built by preceding the data that follows the record separator in the truncated record with the TRN control information. The TRN count

in this case contains the length of just the truncated data for the record and does not include the record separator character. For the special case of a null record, no TRN control information is placed after the record separator character because there is no record data to be made transparent.

If the compression function is specified in the algorithm modifier, the conversion process continues for this record with the compression function.

If not, the conversion process for this record is completed by placing the transparent record into the receiver. If the transparent record cannot be completely contained in the receiver, the instruction ends with a resultant condition of receiver overrun.

When the record spanning function is specified in the algorithm modifier, as much of the transparent record as will fit is placed into the receiver and the controls operand is updated to describe how much of the source record was successfully converted into the receiver. The TRN count is also adjusted to describe the length of the data successfully converted into the receiver; thus, the transparent data for the record is not spanned out of the receiver. The remaining bytes of the transparent record, if any, will be processed as a partial source record on the next invocation of the instruction and will be preceded by the appropriate TRN control information. For the special case where only 1 to 3 bytes are available at the end of the receiver, (not enough room for the record separator, the transparency control, and a byte of data) then just the record separator is placed in the receiver for the record being converted. This can cause up to 2 bytes of unused space at the end of the receiver. The value of these unused bytes is unpredictable.

When the record spanning function is not specified in the algorithm modifier, the controls operand is updated to describe only the last fully converted record in the receiver and the value of the remaining bytes in the receiver is unpredictable.

The third function performed is compression.

Compression is performed if the function is specified in the algorithm modifier. This provides for reducing the size of strings of duplicate characters in the source data. The source data to be compressed may have assumed a partially converted form at this point as a result of processing for functions specified in the algorithm modifier. Compressed data is built by concatenating one or more compression strings together to describe the bytes that make up the converted form of the source data prior to the compression step. The bytes of the converted source data are interrogated to locate the prime compression character strings (two or more consecutive prime compression characters), duplicate character strings (three or more duplicate nonprime characters) and nonduplicate character strings occurring in the source.

The character strings encountered (prime, duplicate and nonduplicate) are reflected in the compressed data by building one or more compression strings to describe them. Compression strings are comprised of an SCB (string control byte) possibly followed by one or more bytes of data related to the character string to be described.

The format of an SCB and the description of the data that may follow it are:

- SCB Char(1)
  - Control Bits 0-1

    00 = n nonduplicate characters are between this SCB and the next one; where n is the value of the count field (1-63).
    01 = Reserved
    10 = This SCB represents n deleted prime compression characters; where n is the value of the count field (2-63). The next byte is the next SCB.
    11 = This SCB represents n deleted duplicate characters; where n is the value of the count field (3-63). The next byte contains a specimen of the deleted characters. The byte following the specimen character contains the next SCB.

  - Count Bits 2-7

    This contains the number of characters that have been deleted for a prime or duplicate string, or the number of characters to the next SCB for a nonduplicate string. A count value of zero cannot be produced.

When record processing is specified, the compression is performed as follows.

The compression function is performed on just the converted form of the current source record including the record separator character. The converted form of the source record prior to the compression step may be a truncated record or a transparent record as described above, depending upon the functions selected in the algorithm modifier. The record separator and TRN control information is always converted as a nonduplicate compression entry to provide for length adjustment of the TRN count, if necessary.

The conversion process for this record is completed by placing the compressed record into the receiver. If the compressed record cannot be completely contained in the receiver, the instruction ends with a resultant condition of receiver overrun.

When the record spanning function is specified in the algorithm modifier, as much of the compressed record as will fit is placed into the receiver and the controls operand is updated to describe how much of the source record was successfully converted into the receiver. The last compression entry placed into the receiver may be adjusted if necessary to a length that provides for filling out the receiver. This length adjustment applies only to compression entries for nonduplicate strings. Compression entries for duplicate strings are placed in the receiver only if they fit with no adjustment. For the special case where data transparency conversion is specified, the transparent data being described is not spanned out of the receiver. This is provided for by performing length adjustment on the TRN count of a transparent record, which may be included in the compressed data so that it describes only the source data that was successfully converted into the receiver. For the special case where only 2 to 5 bytes are available at the end of the receiver, not enough room for the compression entry for a nonduplicate string containing the record separator and the TRN control, and up to a 2-byte compression entry for some of the transparent data, the nonduplicate compression entry is adjusted to describe only the record separator. By doing this, no more than 3 bytes of unused space will remain in the receiver. The value of these unused bytes is unpredictable. Unconverted source record bytes, if any, will be processed as a partial source record on the next invocation

of the instruction and will be preceded by the appropriate TRN control information when performing transparency conversion.

When the record spanning function is not specified in the algorithm modifier, the controls operand is updated to describe only the last fully converted record in the receiver. The value of the remaining bytes in the receiver is unpredictable.

When string processing is specified, the compression is performed as follows.

The compression function is performed on the data for the entire source operand on a compression string basis. In this case, the fields in the controls operand related to record processing are ignored.

The conversion process for the source operand is completed by placing the compressed data into the receiver.

When the compressed data cannot be completely contained in the receiver, the instruction ends with a resultant condition of receiver overrun. As much of the compressed data as will fit is placed into the receiver and the controls operand is updated to describe how much of the source data was successfully converted into the receiver. The last compression entry placed into the receiver may be adjusted if necessary to a length that provides for filling out the receiver. This length adjustment applies only to compression entries for nonduplicate strings. Compression entries for duplicate strings are placed in the receiver only if they fit with no adjustment. By doing this, no more than 1 byte of unused space will remain in the receiver.

When the compressed data can be completely contained in the receiver, the instruction ends with a resultant condition of source exhausted. The compressed data is placed into the receiver and the controls operand is updated to indicate that all of the source data was successfully converted into the receiver.

At this point, either conversion of a source record has been completed or conversion has been interrupted due to detection of the source exhausted or receiver overrun conditions. For record processing, if neither of the above conditions has been detected either during conversion of or at completion of conversion for the current record, the conversion process continues on the next source record with the blank truncation step described above.

At completion of the instruction, the offset value for the receiver locates the byte following the last converted byte in the receiver. The value of the remaining bytes in the receiver after the last converted byte are unpredictable. The offset value for the source locates the byte following the last source byte for which conversion was completed. When the record spanning function is specified in the algorithm modifier, the unconverted source record bytes value specifies the length of the remaining source record bytes yet to be converted. When the record spanning function is not specified in the algorithm modifier, the unconverted source record bytes value has no meaning and is not set. The gap offset value indicates the offset to the next gap relative to the source offset value set for this condition. The gap offset value has no meaning and is not set when the data field length is zero.

Limitations: The following are limits that apply to the functions performed by this instruction.

Any form of overlap between the operands on this instruction yields unpredictable results in the receiver operand.

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for this instruction.

Resultant Conditions: Source exhausted - All bytes in the source operand have been converted into the receiver operand. Receiver overrun - An overrun condition in the receiver operand was detected before all of the bytes in the source operand were processed.

Programming Notes:

If the source operand does not end on a record boundary, in which case the last record is spanned out of the source, this instruction performs conversion only up to the start of that partial record. In this case, the user of the instruction must move this partial record to combine it with the rest of the record in the source operand to provide for its being processed correctly upon the next invocation of the instruction. If full records are provided, the instruction performs its conversions out to the end of the source operand and no special processing is required.

For the special case of a tie between the source exhausted and receiver overrun conditions, the source exhausted condition is recognized first. That is, when source exhausted is the resultant condition, the receiver may also be full. In this case, the offset into the receiver operand may contain a value equal to the length specified for the receiver, which would cause an exception to be detected on the next invocation of the instruction. The processing performed for the source exhausted condition should provide for this case if the instruction is to be invoked multiple times with the same controls operand template. When the receiver overrun condition is the resultant condition, the source will always contain data that can be converted.

Exceptions

Exception		Operands 1	2	3	Other
06	Addressing				
	01 Spacing addressing violation	X	X	X	
	02 Boundary alignment violation	X	X	X	
	03 Range	X	X	X	
	06 Optimized addressability invalid	X	X	X	
08	Argument/parameter				
	01 Parameter reference violation	X	X	X	
10	Damage encountered				
	04 System object damage state				X
	44 Partial system object damage				X
1C	Machine-dependent exception				
	03 Machine storage limit exceeded				X

Exception		Operands 1	2	3	Other
20	Machine support				
	02 Machine check				X
	03 Function check				X
22	Object access				
	01 Object not found	X	X	X	
	02 Object destroyed	X	X	X	
	03 Object suspended	X	X	X	
24	Pointer specification				
	01 Pointer does not exist	X	X	X	
	02 Pointer type invalid	X	X	X	
2A	Program creation				
	05 Invalid op-code extender field				X
	06 Invalid operand type	X	X	X	
	07 Invalid operand attribute	X	X	X	
	08 Invalid operand value range	X	X	X	
	09 Invalid branch target operand				X
	0A Invalid operand length		X		
	0C Invalid operand odt reference	X	X	X	
	0D Reserved bits are not zero	X	X	X	X
2C	Program execution				
	04 Invalid branch target				X
2E	Resource control limit				
	01 user profile storage limit exceeded				X
32	Scalar specification				
	01 Scalar type invalid	X	X	X	
36	Space management				
	01 space extension/truncation				X
38	Template specification				
	01 Template value invalid		X		

1.21 Convert Decimal Form to Floating-Point (CVTDFFP)

Op Code (Hex)	Operand 1	Operand 2	Operand 3
107F	Receiver	Decimal exponent	Decimal significand

Operand 1: Floating-point variable scalar.

Operand 2: Packed scalar or zoned scalar.

Operand 3: Packed scalar or zoned scalar.

Description: This instruction converts the decimal form of a floating-point value specified by a decimal exponent and a decimal significand to binary floating-point format, and places the result in the receiver operand. The decimal exponent (operand 2) and decimal significand (operand 3) are considered to specify a decimal form of a floating-point number. The value of this number is considered to be as follows:

```
Value = S * (10**E)
```

where:

S = The value of the decimal significand operand.
E = The value of the decimal exponent operand.
* Denotes multiplication.
** Denotes exponentiation.

The decimal exponent must be specified as a decimal integer value; no fractional digit positions may be specified in its definition. The decimal exponent is a signed integer value specifying a power of 10 which gives the floating-point value its magnitude. A decimal exponent value too large or too small to be represented in the receiver will result in the detection of the appropriate floating-point overflow or floating-point underflow exception.

The decimal significand must be specified as a decimal value with a single integer digit position and optional fractional digit positions. The decimal significand is a signed decimal value specifying decimal digits which give the floating-point value its precision. The significant digits of the decimal significand are considered to start with the leftmost nonzero decimal digit and continue to the right to the end of the decimal significand value. Significant digits beyond 7 for a short float receiver, and beyond 15 for a long float receiver exceed the precision provided for in the binary floating-point receiver. These excess digits do participate in the conversion to provide for uniqueness of the conversion as well as for proper rounding.

The decimal form floating-point value specified by the decimal exponent and decimal significand operands is converted to a binary floating-point number and rounded to the precision of the result field as follows:

Source values which, in magnitude M, are in the range where (10**31-1) * 10**-31 <= M <= (10**31-1) *10**+31 are converted subject to the normal rounding error defined for the floating-point rounding modes.

Source values which, in magnitude M, are in the range where (10**31-1) * 10**-31 > M > (10**31-1) *10** +31 are converted such that the rounding error incurred on the conversion may exceed that defined above. For round to nearest, this error will not exceed by more than .47 units in the least significant digit position of the result in relation to the error that would be incurred for normal rounding. For the other floating-point rounding modes, this error will not exceed 1.47 units in the least significant digit position of the result.

The converted and rounded value is then assigned to the floating-point receiver.

Exceptions

Exception		Operands 1	2	3	Other
06	Addressing				
	01 Spacing addressing violation	X	X	X	
	02 Boundary alignment violation	X	X	X	
	03 Range	X	X	X	
	06 Optimized addressability invalid	X	X	X	
08	Argument/parameter				
	01 Parameter reference violation	X	X	X	
0C	Computation				
	02 Decimal data		X	X	
	06 Floating-point overflow	X			
	07 Floating-point underflow	X			
	0D Floating-point inexact result	X			
10	Damage encountered				
	04 System object damage state				X
	44 Partial system object damage				X
1C	Machine-dependent exception				
	03 Machine storage limit exceeded				X
20	Machine support				
	02 Machine check				X
	02 Function check				X
22	Object access				
	01 Object not found	X	X	X	
	02 Object destroyed	X	X	X	
	03 Object suspended	X	X	X	
24	Pointer specification				
	01 Pointer does not exist	X	X	X	
	02 Pointer type invalid	X	X	X	
2A	Program creation				
	05 Invalid op-code extender field				X

Exception		Operands 1	2	3	Other
	06 Invalid operand type	X	X	X	
	07 Invalid operand attribute	X	X	X	
	08 Invalid operand value range	X	X	X	
	0A Invalid operand length	X	X	X	
	0C Invalid operand odt reference	X	X	X	
	0D Reserved bits are not zero	X	X	X	X
2E	Resource control limit				
	01 user profile storage limit exceeded				X
32	Scalar specification				
	01 Scalar type invalid	X	X	X	
36	Space management				
	01 space extension/truncation				X

1.22 Convert External Form to Numeric Value (CVTEFN)

Op Code (Hex)	Operand 1	Operand 2	Operand 3
1087	Receiver	Source	Mask

Operand 1: Numeric variable scalar or data-pointer-defined numeric scalar.

Operand 2: Character scalar or data-pointer-defined character scalar.

Operand 3: Character(3) scalar, null, or data-pointer-defined character(3) scalar.

Description: This instruction scans a character string for a valid decimal number in display format, removes the display character, and places the results in the receiver operand. The operation begins by scanning the character string value in the source operand to make sure it is a valid decimal number in display format.

The character string defined by operand 2 consists of the following optional entries:

- Currency symbol - This value is optional and, if present, must precede any sign and digit values. The valid symbol is determined by operand 3. The currency symbol may be preceded in the field by blank (hex 40) characters.
- Sign symbol - This value is optional and, if present, may precede any digit values (a leading sign) or may follow the digit values (a trailing sign). Valid signs are positive (hex 4E) and negative (hex 60). The sign symbol, if it is a leading sign, may be preceded by blank characters. If the sign symbol is a trailing sign, it must be the rightmost character in the field. Only one sign symbol is allowed.
- Decimal digits - Up to 31 decimal digits may be specified. Valid decimal digits are in the range of hex F0 through hex F9 (0-9). The first decimal digit may be preceded by blank characters (hex 40), but hex 40 values located to the right of the leftmost decimal digit are invalid.

The decimal digits may be divided into two parts by the decimal point symbol: an integer part and a fractional part. Digits to the left of the decimal point are interpreted as integer values. Digits to the right are interpreted as a fractional values. If no decimal point symbol is included, the value is interpreted as an integer value. The valid decimal point symbol is determined by operand 3. If the decimal point symbol precedes the leftmost decimal digit, the digit value is interpreted as a fractional value, and the leftmost decimal digit must be adjacent to the decimal point symbol. If the decimal point follows the rightmost decimal digit, the digit value is interpreted as an integer value, and the rightmost decimal digit must be adjacent to the decimal point.

Decimal digits in the integer portion may optionally have comma symbols separating groups of three digits. The leftmost group may contain one, two, or three decimal digits, and each succeeding group must be preceded by the comma symbol and contain three digits. The comma symbol must be adjacent to a decimal digit on either side. The valid comma symbol is determined by operand 3.

Decimal digits in the fractional portion may not be separated by commas and must be adjacent to one another.

Examples of external formats follow. The following symbols are used.

$ currency symbol
. decimal point
, comma
D digit (hex F0-F9)
blank (hex 40)
+ positive sign
- negative sign

Format	Comments
$+DDDD.DD	Currency symbol, leading sign, no comma separators
DD,DDD-	Comma symbol, no fraction, trailing sign
-.DDD	No integer, leading sign
$DDD,DDD-	No fraction, comma symbol, trailing sign
$ + DD.DD	Embedded blanks before digits

Operand 3 must be a 3-byte character scalar. Byte 1 of the string indicates the byte value that is to be used for the currency symbol. Byte 2 of the string indicates the byte value to be used for the comma symbol. Byte 3 of the string indicates the byte value to be used for the decimal point symbol. If operand 3 is null, the currency symbol (hex 5B), comma (hex 6B), and decimal point (hex 4B) are used.

If the syntax rules are violated, a conversion exception is signaled. If not, a zoned decimal value is formed from the digits of the display format character string. This number is placed in the receiver operand following the rules of a normal arithmetic conversion.

If a decimal to binary conversion causes a size exception to be signaled, the binary value contains the correct truncated result only if the decimal value contains 15 or fewer significant nonfractional digits.

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for this instruction.

Exceptions

Exception		Operands 1	2	3	Other
06	Addressing				
	01 Spacing addressing violation	X	X	X	
	02 Boundary alignment	X	X	X	
	03 Range	X	X	X	
	06 Optimized addressability invalid	X	X	X	
08	Argument/parameter				
	01 Parameter reference violation	X	X	X	
0C	Computation				

Exception		Operands 1	2	3	Other
	01 Conversion		X		
	0A Size	X			
10	Damage encountered				
	04 System object damage	X	X	X	X
	44 Partial system object damage	X	X	X	X
1C	Machine-dependent exception				
	03 Machine storage limit exceeded				X
20	Machine support				
	02 Machine check				X
	03 Function check				X
22	Object access				
	01 Object not found	X	X	X	
	02 Object destroyed	X	X	X	
	03 Object suspended	X	X	X	
24	Pointer specification				
	01 Pointer does not exist	X	X	X	
	02 Pointer type invalid	X	X	X	
2A	Program creation				
	06 Invalid operand type	X	X	X	
	07 Invalid operand attribute	X	X	X	
	08 Invalid operand value range	X	X	X	
	0A Invalid operand length		X	X	
	0C Invalid operand odt reference	X	X	X	
	0D Reserved bits are not zero	X	X	X	X
2E	Resource control limit				
	01 user profile storage limit exceeded				X
32	Scalar specification				
	01 Scalar type invalid	X	X	X	
	02 Scalar attribute invalid			X	
36	Space management				
	01 space extension/truncation				X

1.23 Convert Floating-Point to Decimal Form (CVTFPDF)

Op Code (Hex)	Operand 1	Operand 2	Operand 3
10BF	Decimal exponent	Decimal significand	Source

Operand 1: Packed variable scalar or zoned variable scalar.

Operand 2: Packed variable scalar or zoned variable scalar.

Operand 3: Floating-point scalar.

Optional Form

Mnemonic	Op Code (Hex)	Form Type
CVTFPDFR	12BF	Round

Description: This instruction converts a binary floating-point value to a decimal form of a floating-point value specified by a decimal exponent and a decimal significand, and places the result in the decimal exponent and decimal significand operands.

The value of this number is considered to be as follows:

```
Value = S * (10**E)
```

where:

S = The value of the decimal significand operand.

E = The value of the decimal exponent operand.

* Denotes multiplication.

** Denotes exponentiation.

The decimal exponent must be specified as a decimal integer value. No fractional digit positions are allowed. It must be specified with at least five digit positions. The decimal exponent provides for containing a signed integer value specifying a power of 10 which gives the floating-point value its magnitude.

The decimal significand must be specified as a decimal value with a single integer digit position and optional fractional digit positions. The decimal significand provides for containing a signed decimal value specifying decimal digit is which give the floating-point value its precision. The decimal significand is formed as a normalized value, that is, the leftmost digit position is nonzero for a nonzero source value.

When the source contains a representation of a normalized binary floating-point number with decimal significand digits beyond the leftmost 7 digits for a short floating-point source or beyond the leftmost 15 digits for a long floating-point source, the precision allowed for the binary floating-point source is exceeded.

When the source contains a representation of a denormalized binary floating-point number, it may provide less precision than the precision of a normalized binary floating-point number, depending on the particular source value. Decimal

significand digits exceeding the precision of the source are set as a result of the conversion to provide for uniqueness of conversion and are correct, except for rounding errors. These digits are only as precise as the floating-point calculations that produced the source value. The floating-point inexact result exception provides a means of detecting loss of precision in floating-point calculations.

The binary floating-point source is converted to a decimal form floating-point value and rounded to the precision of the decimal significand operand as follows:

- The decimal significand is formed as a normalized value and the decimal exponent is set accordingly.
- For the nonround form of the instruction, the value to be assigned to the decimal significand is adjusted to the precision of the decimal significand, if necessary, according to the current float rounding mode in effect for the process. For the optional round form of the instruction, the decimal round algorithm is used for the precision adjustment of the decimal significand. The decimal round algorithm overrides the current floating-point rounding mode that is in effect for the process.
- Source values which, in magnitude M, are in the range where (10**31-1) * 10**-31 < = M < = (10**31-1) * 10**+31 are converted subject to the normal rounding error defined for the floating-point rounding modes and the optional round form of the instruction.
- Source values which, in magnitude M, are in the range where (10**31-1) * 10**-31 > M > (10**31-1) * 10**+31 are converted such that the rounding error incurred on the conversion may exceed that defined above. For round to nearest and the optional round form of the instruction, this error will not exceed by more than .47 units in the least significant digit position of the result, the error that would be incurred for a correctly rounded result. For the other floating-point rounding modes, this error will not exceed 1.47 units in the least significant digit position of the result.
- If necessary, the decimal exponent value is adjusted to compensate for rounding.
- The converted and rounded value is then assigned to the decimal exponent and decimal significand operands.

A size exception cannot occur on the assignment of the decimal exponent or the decimal significand values.

Limitations: The following are limits that apply to the functions performed by this instruction.

The result of the operation is unpredictable for any type of overlap between the decimal exponent and decimal significand operands.

Exceptions

Exception		Operands 1	2	3	Other
06	Addressing				
	01 Spacing addressing violation	X	X	X	
	02 Boundary alignment violation	X	X	X	
	03 Range	X	X	X	

Exception		Operands 1	2	3	Other
	06 Optimized addressability invalid	X	X	X	
08	Argument/parameter				
	01 Parameter reference violation	X	X	X	
0C	Computation				
	0C Invalid floating-point conversion	X	X		
	0D Floating-point inexact result		X		
10	Damage encountered				
	04 System object damage state				X
	44 Partial system object damage				X
1C	Machine-dependent exception				
	03 Machine storage limit exceeded				X
20	Machine support				
	02 Machine check				X
	03 Function check				X
22	Object access				
	01 Object not found	X	X	X	
	02 Object destroyed	X	X	X	
	03 Object suspended	X	X	X	
24	Pointer specification				
	01 Pointer does not exist	X	X	X	
	02 Pointer type invalid	X	X	X	
2A	Program creation				
	05 Invalid op-code extender field				X
	06 Invalid operand type	X	X	X	
	07 Invalid operand attribute	X	X	X	
	08 Invalid operand value range	X	X	X	
	0A Invalid operand length	X	X	X	
	0C Invalid operand odt reference	X	X	X	
	0D Reserved bits are not zero	X	X	X	X
2E	Resource control limit				
	01 user profile storage limit exceeded				X
32	Scalar specification				
	04 Scalar type invalid	X	X	X	
36	Space management				
	01 space extension/truncation				X

1.24 Convert Hex to Character (CVTHC)

Op Code (Hex)	Operand 1	Operand 2
1086	Receiver	Source

Operand 1: Character variable scalar.

Operand 2: Character variable scalar.

Description: Each hex digit (4-bit value) of the string value in the source operand is converted to a character (8-bit value) and placed in the receiver operand.

Hex Digits Characters

Hex 0-9 = Hex F0-F9

Hex A-F = Hex C1-C6

The operation begins with the two operands left-adjusted and proceeds left to right until all the characters of the receiver operand have been filled. If the source operand contains fewer hex digits than needed to fill the receiver, the excess characters are assigned a value of hex F0. If the source operand is too large, a length conformance or an invalid operand length exception is signaled.

Substring operand references that allow for a null substring reference (a length value of zero) may be specified for operands 1 and 2. The effect of specifying a null substring reference for the source is that the bytes of the receiver are each set with a value of hex F0. The effect of specifying a null substring reference for the receiver is that no result is set.

Exceptions

Exception		Operands 1	Operands 2	Other
06	Addressing			
	01 Spacing addressing violation	X	X	
	02 Boundary alignment	X	X	
	03 Range	X	X	
	06 Optimized addressability invalid	X	X	
08	Argument/parameter			
	01 Parameter reference violation	X	X	
0C	Computation			
	08 Length conformance		X	
10	Damage encountered			
	04 System object damage state	X	X	X
	44 Partial system object damage	X	X	X
1C	Machine-dependent exception			
	03 Machine storage limit exceeded			X

Exception		Operands 1	2	Other
20	Machine support			
	02 Machine check			X
	02 Function check			X
22	Object access			
	01 Object not found	X	X	
	02 Object destroyed	X	X	
	03 Object suspended	X	X	
24	Pointer specification			
	01 Pointer does not exist	X	X	
	02 Pointer type invalid	X	X	
2A	Program creation			
	06 Invalid operand type	X	X	
	07 Invalid operand attribute	X	X	
	08 Invalid operand value range	X	X	
	0A Invalid operand length	X	X	
	0A Invalid operand odt reference	X	X	
	0D Reserved bits are not zero	X	X	X
2E	Resource control limit			
	01 user profile storage limit exceeded			X
36	Space management			
	01 space extension/truncation			X

1.25 Convert MRJE to Character (CVTMC)

Op Code (Hex)	Operand 1	Operand 2	Operand 3
10AB	Receiver	Controls	Source

Operand 1: Character variable scalar.

Operand 2: Character(6) variable scalar (fixed-length).

Operand 3: Character scalar.

Optional Forms

Mnemonic	Op Code (Hex)	Form Type
CVTMCI	18AB	Indicator
CVTMCB	1CAB	Branch

Extender: Branch options or indicator options.

If the branch or indicator option is specified in the op code, the extender field must be present along with one or two branch targets (for branch options) or one or two indicator operands (for indicator options). The branch or indicator operands immediately follow the last operand listed above. See Chapter 1. "Introduction" for the encoding of the extender field and the allowed syntax of the branch and indicator operands.

Description: This instruction converts a character string from the MRJE (MULTI-LEAVING remote job entry) compressed format to character format. The operation converts the source (operand 3) from the MRJE compressed format to character format under control of the controls (operand 2) and places the results in the receiver (operand 1).

The source and receiver operands must both be character strings. The source operand cannot be specified as either a signed or unsigned immediate value.

The controls operand must be a character scalar that specifies additional information to be used to control the conversion operation. It must be at least 6 bytes in length and have the following format:

- Controls operand — Char(6)
  - Offset into the receiver operand — Bin(2)
  - Offset into the source operand — Bin(2)
  - Algorithm modifier — Char(1)
  - Receiver record length — Char(1)

As input to the instruction, the source and receiver offset fields specify the offsets where bytes of the source and receiver operands are to be processed. If an offset is equal to or greater than the length specified for the operand it corresponds to (it identifies a byte beyond the end of the operand), a template value invalid exception is signaled. As output from the instruction, the source and receiver offset fields specify offsets that indicate how much of the operation is complete when the instruction ends.

The algorithm modifier has the following valid values:

- Hex 00 = Do not move SRCBs (sub record control bytes) from the source into the receiver.
- Hex 01 = Move SRCBs from the source into the receiver.

The receiver record length value specifies the record length to be used to convert source records into the receiver operand. This length applies to only the string portion of the receiver record and does not include the optional SRCB field. If a receiver record length of 0 is specified, a template value invalid exception is signaled.

Only the first 6 bytes of the controls operand are used. Any excess bytes are ignored.

The operation begins by accessing the bytes of the source operand at the location specified by the source offset. This is assumed to be the start of a record. The bytes of the records in the source operand are converted into the receiver operand at the location specified by the receiver offset according to the following algorithm.

The first byte of the source record is considered to be an RCB (record control byte) that is to be ignored during conversion.

The second byte of the source record is considered to be an SRCB. If an algorithm modifier of value hex 00 was specified, the SRCB is ignored. If an algorithm modifier of value hex 01 was specified, the SRCB is copied into the receiver.

The strings to be built in the receiver record are described in the source after the SRCB by one or more SCBs (string control bytes).

The format of the SCBs in the source are as follows:

```
o k 1 jjjjj
```

The bit meanings are:

Bit	Value	Meaning
o	0	End of record;the EOR SCB is hex 00.
	1	All other SCBs.
k	0	The string is compressed.
	1	The string is not compressed.
1		For k = 0:
	0	Blanks (hex 40s) have been deleted.
	1	Nonblank characters have been deleted. The next character in the data stream is the specimen character.
		For k = 1:

Bit	Value	Meaning
		This bit is part of the length field for length of uncompressed data.
jjjjj		Number of characters that have been deleted if k = 0. The value can be 1-31.
1jjjjj		Number of characters to the next SCB (no compression) if k=1. The value can be 1-63.
		The uncompressed (nonidentical bytes) follow the SCB in the data stream.

A length of 0 encountered in an SCB results in the signaling of a conversion exception.

Strings of blanks or nonblank identical characters described in the source record are repeated in the receiver the number of times indicated by the SCB count value.

Strings of nonidentical characters described in the source record are moved into the receiver for the length indicated by the SCB count value.

When an EOR (end of record) SCB (hex 00) is encountered in the source, the receiver is padded with blanks out to the end of the current record.

If the converted form of a source record is larger than the receiver record length, the instruction is terminated by signaling a length conformance exception.

If the end of the source operand is not encountered, the operation then continues by reapplying the above algorithm to the next record in the source operand.

If the end of the source operand is encountered (whether or not in conjunction with a record boundary, EOR SCB in the source), the instruction ends with a resultant condition of *source exhausted*. The offset value for the receiver locates the byte following the last fully converted record in the receiver. The offset value for the source locates the byte following the last source record for which conversion is complete. The value of the remaining bytes in the receiver after the last converted record are unpredictable.

If the converted form of a record cannot be completely contained in the receiver, the instruction ends with a resultant condition of *receiver overrun*. The offset value for the receiver locates the byte following the last fully converted record in the receiver. The offset value for the source locates the byte following the last source record for which conversion is complete. The value of the remaining bytes in the receiver after the last converted record is unpredictable.

If the source exhausted and the receiver overrun conditions occur at the same time, the source exhausted condition is recognized first. In this case, the offset into the receiver operand may contain a value equal to the length specified for the receiver which causes an exception to be signaled on the next invocation of the instruction. The processing performed for the source exhausted condition provides for this case if the instruction is invoked multiple times with the same

controls operand template. When the receiver overrun condition is the resultant condition, the source always contains data that can be converted.

Limitations: The following are limits that apply to the functions performed by this instruction.

Any form of overlap between the operands on this instruction yields unpredictable results in the receiver operand.

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for this instruction.

Resultant Conditions: Source exhausted - All full records in the source operand have been converted into the receiver operand. Receiver overrun - An overrun condition in the receiver operand was detected prior to processing all of the bytes in the source operand.

Exceptions

Exception		Operands 1	2	3	Other
06	Addressing				
	01 Spacing addressing violation	X	X	X	
	02 Boundary alignment violation	X	X	X	
	03 Range	X	X	X	
	06 Optimized addressability invalid	X	X	X	
08	Argument/parameter				
	01 Parameter reference violation	X	X	X	
0C	Computation				
	01 Conversion			X	
	08 Length conformance	X			
10	Damage encountered				
	04 System object damage state				X
	44 Partial system object damage				X
1C	Machine-dependent exception				
	03 Machine storage limit exceeded				X
20	Machine support				
	02 Machine check				X
	03 Function check				X
22	Object access				
	01 Object not found	X	X	X	
	02 Object destroyed	X	X	X	
	03 Object suspended	X	X	X	
24	Pointer specification				
	01 Pointer does not exist	X	X	X	

Exception		Operands 1	2	3	Other
	02 Pointer type invalid	X	X	X	
2A	Program creation				
	05 Invalid op-code extender field				X
	06 Invalid operand type	X	X	X	
	07 Invalid operand attribute	X	X	X	
	08 Invalid operand value range	X	X	X	
	09 Invalid branch target operand				X
	0A Invalid operand length		X		
	0C Invalid operand odt reference	X	X	X	
	0D Reserved bits are not zero	X	X	X	X
2C	Program execution				
	04 Invalid branch target				X
2E	Resource control limit				
	01 user profile storage limit exceeded				X
32	Scalar specification				
	01 Scalar type invalid	X	X	X	
36	Space management				
	01 space extension/truncation				X
38	Template specification				
	01 Template value invalid		X		

1.26 Convert Numeric to Character (CVTNC)

Op Code (Hex)	Operand 1	Operand 2	Operand 3
10A3	Receiver	Source	Attributes

Operand 1: Character variable scalar or data-pointer-defined character scalar.

Operand 2: Numeric scalar or data-pointer-defined numeric scalar.

Operand 3: Character(7) scalar or data-pointer-defined character(7) scalar.

Description: The source numeric value (operand 2) is converted and copied to the receiver character string (operand 1). The receiver operand is treated as though it had the attributes supplied by operand 3. Operand 1, when viewed in this manner, receives the numeric value of operand 2 following the rules of the Copy Numeric Value instruction.

The format of operand 3 is as follows:

- Scalar attributes — Char(7)
  - Scalar type — Char(1)

    Hex 00 = Signed binary
    Hex 01 = Floating-point
    Hex 02 = Zoned decimal
    Hex 03 = Packed decimal
    Hex 0A = Unsigned binary
  - Scalar length — Bin(2)

    If binary:

    Length (L) (where L = 2 or 4) — Bits 0-15

    If floating-point:

    Length (where L = 4 or 8) — Bits 0-15

    If zoned decimal or packed decimal:

    Fractional digits (F) — Bits 0-7

    Total digits (T) (where $1 \leq T \leq 31$ and $0 \leq F \leq T$) — Bits 8-15
  - Reserved (binary 0) — Bin(4)

The byte length of operand 1 must be large enough to contain the numeric value described by operand 3. If it is not large enough, a scalar value invalid exception is signaled. If it is larger than needed, the numeric value is placed in the leftmost bytes and the unneeded rightmost bytes are unchanged by the instruction.

If a decimal to binary conversion causes a size exception to be signaled, the binary value contains the correct truncated result only if the decimal value contains 15 or fewer significant nonfractional digits.

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for this instruction.

Exceptions

Exception		Operands 1	2	3	Other
06	Addressing				
	01 Spacing addressing violation	X	X	X	
	02 Boundary alignment	X	X	X	
	03 Range	X	X	X	
	04 External data object not found	X	X	X	
	06 Optimized addressability invalid	X	X	X	
08	Argument/parameter				
	01 Parameter reference violation	X	X	X	
0C	Computation				
	02 Decimal data		X		
	06 Floating-point overflow	X			
	07 Floating-point underflow	X			
	09 Floating-point invalid operand		X		
	0A Size	X			
	0C Invalid floating-point conversion	X			
	0D Floating-point inexact result	X			
10	Damage encountered				
	04 System object damage state	X	X	X	X
	44 Partial system object damage	X	X	X	X
1C	Machine-dependent exception				
	03 Machine storage limit exceeded				X
20	Machine support				
	02 Machine check				X
	03 Function check				X
22	Object access				
	01 Object not found	X	X	X	
	02 Object destroyed	X	X	X	
	03 Object suspended	X	X	X	
24	Pointer specification				
	01 Pointer does not exist	X	X	X	
	02 Pointer type invalid	X	X	X	
2A	Program creation				
	06 Invalid operand type	X	X	X	
	07 Invalid operand attribute	X	X	X	

Exception		Operands 1	2	3	Other
	08 Invalid operand value range	X	X	X	
	0A Invalid operand length	X		X	
	0C Invalid operand odt reference	X	X	X	
	0D Reserved bits are not zero	X	X	X	X
2E	Resource control limit				
	01 user profile storage limit exceeded				X
32	Scalar specification				
	01 Scalar type invalid	X	X	X	
	02 Scalar attribute invalid				
	03 Scalar value invalid			X	
36	Space management				
	01 space extension/truncation				X

1.27 Convert SNA to Character (CVTSC)

Op Code (Hex)	Operand 1	Operand 2	Operand 3
10DB	Receiver	Controls	Source

Operand 1: Character variable scalar.

Operand 2: Character(14) variable scalar (fixed length).

Operand 3: Character scalar.

Optional Forms

Mnemonic	Op Code (Hex)	Form Type
CVTSCI	18DB	Indicator
CVTSCB	1CDB	Branch

Extender: Branch options or indicator options.

If the branch or indicator option is specified in the op code, the extender field must be present along with one to three branch targets (for branch options) or one to three indicator operands (for indicator options). The branch or indicator operands immediately follow the last operand listed above. See Chapter 1. "Introduction" for the encoding of the extender field and the allowed syntax of the branch and indicator operands.

Description: This instruction converts a string value from SNA (systems network architecture) format to character. The operation converts the source (operand 3) from SNA format to character under control of the controls (operand 2) and places the result into the receiver (operand 1).

The source and receiver operands must both be character strings. The source operand may not be specified as an immediate operand.

The controls operand must be a character scalar that specifies additional information to be used to control the conversion operation. It must be at least 14 bytes in length and have the following format:

- Controls operand base template — Char(14)
  - Receiver offset — Bin(2)
  - Source offset — Bin(2)
  - Algorithm modifier — Char(1)
  - Receiver record length — Char(1)
  - Record separator — Char(1)
  - Prime compression — Char(1)
  - Unconverted receiver record bytes — Char(1)
  - Conversion status — Char(2)
  - Unconverted transparency string bytes — Char(1)

  - Offset into template to translate table — Bin(2)
- Controls operand optional template extension — Char(64)
  - Record separator translate table — Char(64)

Upon input to the instruction, the source and receiver offset fields specify the offsets where bytes of the source and receiver operands are to be processed. If an offset is equal to or greater than the length specified for the operand it corresponds to (it identifies a byte beyond the end of the operand), a template value invalid exception is signaled. As output from the instruction they are set to specify offsets that indicate how much of the operation is complete when the instruction ends.

The algorithm modifier specifies the optional functions to be performed. Any combination of functions not precluded by the bit definitions below is valid except that at least one of the functions must be specified. All algorithm modifier bits cannot be zero. Specification of an invalid algorithm modifier value results in a template value invalid exception. The meaning of the bits in the algorithm modifier is the following:

Bits	Meaning
0	0 = Do not perform decompression. Interpret a source character value of hex 00 as null. 1 = Perform decompression. Interpret a source character value of hex 00 as a record separator.
1-2	00 = No record separators in source, no blank padding. Do not perform data transparency conversion. 01 = Reserved. 10 = Record separators in source, perform blank padding. Do not perform data transparency conversion. 11 = Record separators in source, perform blank padding. Perform data transparency conversion.
3-4	00 = Do not put record separators into receiver. 01 = Move record separators from source to receiver (allowed only when bit 1 = 1) 10 = Translate record separators from source to receiver (allowed only when bit 1 = 1) 11 = Move record separator from controls to receiver.
5-7	Reserved

The receiver record length value specifies the record length to be used to convert source records into the receiver operand. This length applies only to the data portion of the receiver record and does not include the optional record separator. Specification of a receiver record length of zero results in a template value invalid exception. The receiver record length value is ignored if no record separator processing is requested in the algorithm modifier.

The record separator value specifies the character that is to precede the converted form of each record in the receiver. The record separator character specified in the controls operand is used only for the case where the move record separator from controls to receiver function is specified in the algorithm modifier, or where a missing record separator in the source is detected.

The prime compression value specifies the character to be used as the prime compression character when performing decompression of the SNA format source data to character. It may have any value. The prime compression value

is ignored if the decompression function is not specified in the algorithm modifier.

The unconverted receiver record bytes value specifies the number of bytes remaining in the current receiver record that are yet to be set with converted bytes from the source.

When record separator processing is specified in the algorithm modifier, this value is both input to and output from the instruction. On input, a value of hex 00 means it is the start of processing for a new record, and the initial conversion step is yet to be performed. This indicates that for the case where a function for putting record separators into the receiver is specified in the algorithm modifier, a record separator character has yet to be placed in the receiver. On input, a nonzero value less than or equal to the record length specifies the number of bytes remaining in the current receiver record that are yet to be set with converted bytes from the source. This value is assumed to be the valid count of unconverted receiver record bytes relative to the current byte to be processed in the receiver as located by the receiver offset value. As such, it is used to determine the location of the next record boundary in the receiver operand. This value must be less than or equal to the receiver record length value; otherwise, a template value invalid exception is signaled. On output, this field is set with a value as defined above which describes the number of bytes of the current receiver record not yet containing converted data.

When record separator processing is not specified in the algorithm modifier, this value is ignored.

The conversion status value specifies status information for the operation to be performed. The meaning of the bits in the conversion status is the following:

Bits	Meaning
0	0 = No transparency string active.
	1 = Transparency string active. Unconverted transparency string bytes value contains the remaining string length.
1-15	Reserved

This field is both input to and output from the instruction. It provides for checkpointing the conversion status over successive executions of the instruction.

If the conversion status indicates transparency string active, but the algorithm modifier does not specify perform data transparency conversion, a template value invalid exception is signaled.

The unconverted transparency string bytes value specifies the number of bytes remaining to be converted for a partially processed transparency string in the source.

When perform data transparency conversion is specified in the algorithm modifier, the unconverted transparency string bytes value can be both input to and output from the instruction.

On input, when the no transparency string active status is specified in the conversion status, this value is ignored.

On input, when transparency string active status is specified in the conversion status, this value contains a count for the remaining bytes to be converted for a transparency string in the source. A value of hex 00 means the count field for a transparency string is the first byte of data to be processed from the source operand. A value of hex 01 through hex FF specifies the count of the remaining bytes to be converted for a transparency string. This value is assumed to be the valid count of unconverted transparency string bytes relative to the current byte to be processed in the source as located by the source offset value.

On output, this value is set if necessary along with the transparency string active status to describe a partially converted transparency string. A value of hex 00 will be set if the count field is the next byte to be processed for a transparency string. A value of hex 01 through hex FF specifying the number of remaining bytes to be converted for a transparency string, will be set if the count field has already been processed.

When do not perform data transparency conversion is specified in the algorithm modifier, the unconverted transparency string bytes value is ignored.

The offset into template to translate table value specifies the offset from the beginning of the template to the record separator translate table. This value is ignored unless the translate record separators from source to receiver function is specified in the algorithm modifier.

The record separator translate table value specifies the translate table to be used in translating record separators specified in the source to the record separator value to be placed into the receiver. It is assumed to be 64 bytes in length, providing for translation of record separator values of from hex 00 to hex 3F. This translate table is used only when the translate record separators from source to receiver function is specified in the algorithm modifier. See the record separator conversion function under the conversion process described below for more detail on the usage of the translate table.

Only the first 14 bytes of the controls operand base template and the optional 64-byte extension area specified for the record separator translate table are used. Any excess bytes are ignored.

The description of the conversion process is presented as a series of separately performed steps, which may be selected in allowable combinations to accomplish the conversion function. It is presented this way to allow for describing these functions separately. However, in the actual execution of the instruction, these functions may be performed in conjunction with one another or separately, depending upon which technique is determined to provide the best implementation.

The operation is performed either on a record-by-record basis, record processing, or on a nonrecord basis, string processing. This is determined by the functions selected in the algorithm modifier. Specifying the record separators in source, perform blank padding or move record separator from controls to receiver indicates record processing is to be performed. If neither of these functions is specified, in which case decompression must be specified, it indicates that string processing is to be performed.

The operation begins by accessing the bytes of the source operand at the location specified by the source offset.

When record processing is specified, the source offset may locate a point at which processing of a partially converted record is to be resumed or processing for a full record is to be started. The unconverted receiver record bytes value indicates whether conversion processing is to be started with a partial or a full record. Additionally, the transparency string active indicator in the conversion status field indicates whether conversion of a transparency string is active for the case of resumption of processing for a partially converted record. The conversion process is started by completing the conversion of a partial source record if necessary before processing the first full source record.

When string processing is specified, the source offset is assumed to locate the start of a compression entry.

When during the conversion process the end of the receiver operand is encountered, the instruction ends with a resultant condition of receiver overrun.

When record processing is specified in the algorithm modifier, this check is performed at the start of conversion for each record. A source exhausted condition would be detected before a receiver overrun condition if there is no source data to convert. If the receiver operand does not have room for a full record, the receiver overrun condition is recognized. The instruction is terminated with status in the controls operand describing the last completely converted record. For receiver overrun, partial conversion of a source record is not performed.

When string processing is specified in the algorithm modifier, then decompression must be specified and the decompression function described below defines the detection of receiver overrun.

When during the conversion process the end of the source operand is encountered, the instruction ends with a resultant condition of source exhausted. See the description of this condition in the conversion process described below to determine the status of the controls operand values and the converted bytes in the receiver for each case.

When string processing is specified, the bytes accessed from the source are converted on a string basis into the receiver operand at the location specified by the receiver offset. In this case, the decompression function must be specified and the conversion process is accomplished with just the decompression function defined below.

When record processing is specified the bytes accessed from the source are converted one record at a time into the receiver operand at the location specified by the receiver offset performing the functions specified in the algorithm modifier in the sequence defined by the following algorithm.

Record separator conversion is performed as requested in the algorithm modifier during the initial record separator processing performed as each record is being converted. This provides for controlling the setting of the record separator value in the receiver.

When the record separators in source option is specified, the following algorithm is used to locate them. A record separator is recognized in the source when a character value less than hex 40 is encountered. When do not perform decompression is specified, a source character value of hex 00 is recognized as a null value rather than as a record separator. In this case, the processing of the

current record continues with the next source byte and the receiver is not updated. When perform data transparency conversion is specified, a character value of hex 35 is recognized as the start of a transparency string rather than as a record separator.

If the do not put record separators into the receiver function is specified, the record separator, if any, from the source record being processed is removed from the converted form of the source record and will not be placed in the receiver.

If the move record separators from the source to the receiver function is specified, the record separator from the source record being processed is left as is in the converted form of the source record and will be placed in the receiver.

If the translate record separators from the source to the receiver function is specified, the record separator from the source record being processed is translated using the specified translate table, replaced with its translated value in the converted form of the source record and, will be placed in the receiver. The translation is performed as in the translate instruction with the record separator value serving as the source byte to be translated. It is used as an index into the specified translate table to select the byte in the translate table that contains the value to which the record separator is to be set. If the selected translate table byte is equal to hex FF, it is recognized as an escape code. The instruction ends with a resultant condition of escape code encountered, and the controls operand is set to describe the conversion status as of the processing completed just prior to the conversion step for the record separator. If the selected translate table byte is not equal to hex FF, the record separator in the converted form of the record is set to its value.

If the move record separator from controls to receiver function is specified, the controls record separator value is used in the converted form of the source record and will be placed into the receiver.

When the record separators in source do blank padding function is requested, an assumed record separator will be used if a record separator is missing in the source data. In this case, the controls record separator character is used as the record separator to precede the converted record if record separators are to be placed in the receiver. The conversion process continues, bypassing the record separator conversion step that would normally be performed. The condition of a missing record separator is detected when during initial processing for a full record, the first byte of data is not a record separator character.

Decompression is performed if the function is specified in the algorithm modifier. This provides for converting strings of duplicate characters in compressed format in the source back to their full size in the receiver. Decompression of the source data is accomplished by concatenating together character strings described by the compression strings occurring in the source. The source offset value is assumed to locate the start of a compression string. Processing of a partial decompressed record is performed if necessary.

The character strings to be built into the receiver are described in the source by one or more compression strings. Compression strings are comprised of an SCB (string control byte) possibly followed by one or more bytes of data related to the character string to be built into the receiver.

The format of an SCB and the description of the data that may follow it is as follows:

- SCB Char(1)
  - Control Bits 0-1

    00 = n nonduplicate characters are between this SCB and the next one; where n is the value of the count field (1-63).
    01 = Reserved.
    10 = This SCB represents n deleted prime compression characters; where n is the value of the count field (1-63). The next byte is the next SCB.
    11 = This SCB represents n deleted duplicate characters; where n is the value of the count field (1-63). The next byte contains a specimen of the deleted characters. The byte following the specimen character contains the next SCB.

  - Count Bits 2-7

    This contains the number of characters that have been deleted for a prime or duplicate string, or the number of characters to the next SCB for a nonduplicate string. A count value of zero is invalid and results in the signaling of a conversion exception.

Strings of prime compression characters or duplicate characters described in the source are repeated in the decompressed character string the number of times indicated by the SCB count value.

Strings of nonduplicate characters described in the source record are formed into a decompressed character string for the length indicated by the SCB count value.

If the end of the source is encountered prior to the end of a compression string, a conversion exception is signaled.

When record processing is specified, decompression is performed one record at a time. In this case, a conversion exception is signaled if a compression string describes a character string that would span a record boundary in the receiver. If the source contains record separators, the case of a missing record separator in the source is detected as defined under the initial description of the conversion process. Record separator conversion, as requested in the algorithm modifier, is performed as the initial step in the building of the decompressed record. A record separator to be placed into the receiver is in addition to the data to be converted into receiver for the length specified in the receiver record length field. The decompression of compression strings from the source continues until a record separator character for the next record is recognized when the source contains record separators, or until the decompressed data required to fill the receiver record has been processed or the end of the source is encountered whether record separators are in the source or not. Transparency strings encountered in the decompressed character string are not scanned for a record separator value. If the end of the source is encountered, the data decompressed to that point appended to the optional record separator for this record forms a partial decompressed record. Otherwise, the decompressed character strings appended to the optional record separator for this record form the decompressed record. The conversion process then continues for this record with the next specified function.

When string processing is specified, decompression is performed on a compression string basis with no record oriented processing implied. The conversion process for each compression string from the source is completed by placing the decompressed character string into the receiver. The conversion process continues decompressing compression strings from the source until the end of the source or the receiver is encountered. When the end of the source operand is encountered, the instruction ends with a resultant condition of source exhausted. When a character string cannot be completely contained in the receiver, the instruction ends with a resultant condition of receiver overrun. For either of the above ending conditions, the controls operand is updated to describe the status of the conversion operation as of the last completely converted compression entry. Partial conversion of a compression entry is not performed.

Data transparency conversion is performed if perform data transparency conversion is specified in the algorithm modifier. This provides for correctly identifying record separators in the source even if the data for a record contains value that could be interpreted as record separator values. Processing of active transparency strings is performed if necessary.

A nontransparent record is built by appending the nontransparent and transparent data converted from the record to the record separator for the record. The nontransparent record may be produced from either a partial record from the source or a full record from the source. This is accomplished by first accessing the record separator for a full record. The case of a missing record separator in the source is detected as defined under the initial description of the conversion process. Record separator conversion as requested in the algorithm modifier is performed if it has not already been performed by a prior step; the rest of the source record is scanned for values of less than hex 40.

A value greater than or equal to hex 40 is considered nontransparent data and is concatenated onto the record being built as is.

A value equal to hex 35 identifies the start of a transparency string. A transparency string is comprised of 2 bytes of transparency control information followed by the data to be made transparent to scanning for record separators. The first byte has a fixed value of hex 35 and is referred to as the TRN (transparency) control character. The second byte is a 1-byte hexadecimal count, a value remaining from 1 to 255 decimal, of the number of bytes of data that follow and is referred to as the TRN count. A TRN count of zero is invalid and causes a conversion exception. This contains the length of the transparent data and does not include the TRN control information length. The transparent data is concatenated to the nontransparent record being built and is not scanned for record separator characters.

A value equal to hex 00 is recognized as the record separator for the next record only when perform decompression is specified in the algorithm modifier. In this case, the nontransparent record is complete. When do not perform decompression is specified in the algorithm modifier, a value equal to hex 00 is ignored and is not included as part of the nontransparent data built for the current record.

A value less than hex 40 but not equal to hex 35 is considered to be the record separator for the next record, and the forming of the nontransparent record is complete.

The building of the nontransparent record is completed when the length of the data converted into the receiver equals the receiver record length if the record separator for the next record is not encountered prior to that point.

If the end of the source is encountered prior to completion of building the nontransparent record, the nontransparent record built up to this point is placed in the receiver and the instruction ends with a resultant condition of source exhausted. The controls operand is updated to describe the status for the partially converted record. This includes describing a partially converted transparency string, if necessary, by setting the active transparency string status and the unconverted transparency string bytes value.

If the building of the nontransparent record is completed prior to encountering the end of the source, the conversion process continues with the blank padding function described below.

Blank padding is performed if the function is specified in the algorithm modifier. This provides for expanding out to the size specified by the receiver record length the source records for which trailing blanks have been truncated. The padded record may be produced from either a partial record from the source or a full record from the source.

The record separator for this record is accessed. The case of a missing record separator in the source is detected as defined under the initial description of the conversion process. Record separator conversion, as requested in the algorithm modifier, is performed if it has not already been performed by a prior step.

The nontruncated data, if any, for the record is appended to the optional record separator for the record. The nontruncated data is determined by scanning the source record for the record separator for the next record. This scan is concluded after processing enough data to completely fill the receiver record or upon encountering the record separator for the next record. The data processed prior to concluding the scan is considered the nontruncated data for the record.

The blanks, if any, required to pad the record out to the nontruncated data for the record, concluding the forming of the padded record.

If the end of the source is encountered during the forming of the padded record, the data processed up to that point, appended to the optional record separator for the record, is placed into the receiver and the instruction ends with a resultant condition of source exhausted. The controls operand is updated to describe the status of the partially converted record.

If the forming of the padded record is concluded prior to encountering the end of the source, the conversion of the record is completed by placing the converted form of the record into the receiver.

At this point, either conversion of a source record has been completed or conversion has been interrupted due to detection of the source exhausted or receiver overrun condition. For record processing, if neither of the above conditions has been detected either during conversion of or at completion of conversion for the current record, the conversion process continues on the next source record with the decompression function described above.

At completion of the instruction, the offset value for the receiver locates the byte following the last converted byte in the receiver. The value of the remaining bytes in the receiver after the last converted byte are unpredictable. The offset value for the source locates the byte following the last source byte for which conversion was completed. When record processing is specified, the unconverted receiver record bytes value specifies the length of the receiver record bytes not yet containing converted data. When perform data transparency conversion is specified in the algorithm modifier, the conversion status indicates whether conversion of a transparency string was active and the unconverted transparency string bytes value specifies the length of the remaining bytes to be processed for an active transparency string.

This instruction does not provide support for compression entries in the source describing data that would span records in the receiver. SNA data from some systems may violate this restriction and as such be incompatible with the instruction. A provision can be made to avoid this incompatibility by performing the conversion of the SNA data through two invocations of this instruction. The first invocation would specify decompression with no record separator processing. The second invocation would specify record separator processing with no decompression. This technique provides for separating the decompression step from record separator processing; thus, the incompatibility is avoided.

This instruction can end with the escape code encountered condition. In this case, it is expected that the user of the instruction will want to do some special processing for the record separator causing the condition. In order to resume execution of the instruction, the user will have to set the appropriate value for the record separator into the receiver and update the controls operand offset values correctly to provide for restarting processing at the right points in the receiver and source operands.

For the special case of a tie between the source exhausted and receiver overrun conditions, the source exhausted condition is recognized first. That is, when source exhausted is the resultant condition, the receiver may also be full. In this case, the offset into the receiver operand may contain a value equal to the length specified for the receiver, which would cause an exception to be detected on the next invocation of the instruction. The processing performed for the source exhausted condition should provide for this case if the instruction is to be invoked multiple times with the same controls operand template. When the receiver overrun condition is the resultant condition, the source will always contain data that can be converted.

This instruction will, in certain cases, ignore what would normally have been interpreted as a record separator value of hex 00. This applies (hex 00 is ignored) for the special case when do not perform decompression and record separators in source are specified in the algorithm modifier. Note that this does not apply when perform decompression is specified, or when do not perform decompression and no record separators in source and move record separator from controls to receiver are specified in the algorithm modifier.

Limitations: The following are limits that apply to the functions performed by this instruction.

Any form of overlap between the operands on this instruction yields unpredictable results in the receiver operand.

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for this instruction.

Resultant Conditions: Source exhausted-The end of the source operand is encountered and no more bytes from the source can be converted. Receiver overrun-An overrun condition in the receiver operand is detected before all of the bytes in the source operand have been processed. Escape code encountered-A record separator character is encountered in the source operand that is to be treated as an escape code.

Exceptions

Exception		Operands 1	2	3	Other
06	Addressing				
	01 Spacing addressing violation	X	X	X	
	02 Boundary alignment violation	X	X	X	
	03 Range	X	X	X	
	06 Optimized addressability invalid	X	X	X	
08	Argument/parameter				
	01 Parameter reference violation	X	X	X	
0C	Computation				
	01 Conversion			X	
10	Damage encountered				
	04 System object damage state				X
	44 Partial system object damage				X
1C	Machine-dependent exception				
	03 Machine storage limit exceeded				X
20	Machine support				
	02 Machine check				X
	03 Function check				X
22	Object access				
	01 Object not found	X	X	X	
	02 Object destroyed	X	X	X	
	03 Object suspended	X	X	X	
24	Pointer specification				
	01 Pointer does not exist	X	X	X	
	02 Pointer type invalid	X	X	X	
2A	Program creation				
	05 Invalid op-code extender field				X
	06 Invalid operand type	X	X	X	
	07 Invalid operand attribute	X	X	X	
	08 Invalid operand value range	X	X	X	

Exception		Operands 1	2	3	Other
	09 Invalid branch target operand				X
	0A Invalid operand length		X		
	0C Invalid operand odt reference	X	X	X	
	0D Reserved bits are not zero	X	X	X	X
2C	Program execution				
	04 Invalid branch target				X
2E	Resource control limit				
	01 user profile storage limit exceeded				X
32	Scalar specification				
	01 Scalar type invalid	X	X	X	
36	Space management				
	01 space extension/truncation				X
38	Template specification				
	01 Template value invalid		X		

1.28 Copy Bits Arithmetic (CPYBTA)

Op Code (Hex)	Operand 1	Operand 2	Operand 3	Operand 4
102C	Receiver	Source	Offset	Length

Operand 1: Character Variable Scalar or Numeric Variable Scalar.

Operand 2: Character Variable Scalar or Numeric Variable Scalar.

Operand 3: Signed or Unsigned Binary Immediate.

Operand 4: Signed or Unsigned Binary Immediate.

Description: Copies the signed bit string source operand starting at the specified offset for a specified length right adjusted to the receiver and pads on the left with the sign of the bit string source.

The selected bits from the source operand are treated as an signed bit string and copied to the receiver value.

The source operand can be character or numeric. The leftmost bytes of the source operand are used in the operation. The source operand is interpreted as a bit string with the bits numbered left to right from 0 to the total number of bits in the string minus 1.

The offset operand indicates which bit of the source operand is to be copied, with a offset of zero indicating the leftmost bit of the leftmost byte of the source operand.

The length operand indicates the number of bits that are to be copied.

If the sum of the offset plus the length exceed the length of the source an "invalid operand length" exception will be raised.

Limitations: Neither the receiver nor the source operand can be a variable length substring.

The length of the receiver cannot exceed four bytes.

The offset must have a non-negative value.

The length operand must be an immediate value between 1 and 32.

Exceptions

Exception		Operands 1	2	3	4	Other
06	Addressing					
	01 Spacing addressing violation	X	X			
	02 Boundary alignment violation	X	X			
	03 Range	X	X			
	06 Optimized addressability invalid	X	X			

Exception		Operands 1	2	3	4	Other
08	Argument/parameter					
	01 Parameter reference violation	X	X			
10	Damage encountered					
	04 System object damage state					X
	44 Partial system object damage					X
1C	Machine-dependent exception					
	03 Machine storage limit exceeded					X
20	Machine support					
	02 Machine check					X
	03 Function check					X
22	Object access					
	02 Object destroyed	X	X			
	03 Object suspended	X	X			
24	Pointer specification					
	01 Pointer does not exist	X	X			
	02 Pointer type invalid	X	X			
2A	Program creation					
	06 Invalid operand type	X	X	X	X	
	07 Invalid operand attribute	X	X	X	X	
	08 Invalid operand value range	X	X	X	X	
	0A Invalid operand length	X	X			
	0C Invalid operand odt reference	X	X			
	0D Reserved bits are not zero	X	X			
2E	Resource control limit					
	01 user profile storage limit exceeded					X
36	Space management					
	01 space extension/truncation					X

1.29 Copy Bits Logical (CPYBTL)

Op Code (Hex)	Operand 1	Operand 2	Operand 3	Operand 4
101C	Receiver	Source	Offset	Length

Operand 1: Character Variable Scalar or Numeric Variable Scalar.

Operand 2: Character Variable Scalar or Numeric Variable Scalar.

Operand 3: Signed or Unsigned Binary Immediate.

Operand 4: Signed or Unsigned Binary Immediate.

Description: Copies the unsigned bit string source operand starting at the specified offset for a specified length to the receiver.

If the receiver is shorter than the length, the left most bits are removed to make the source bit string conform to the length of the receiver. No exceptions are generated when truncation occurs.

The selected bits from the source operand are treated as an unsigned bit string and copied right adjusted to the receiver and padded on the left with binary zeros.

The source operand can be character or numeric. The leftmost bytes of the source operand are used in the operation. The source operand is interpreted as a bit string with the bits numbered left to right from 0 to the total number of bits in the string minus 1.

The offset operand indicates which bit of the source operand is to be copied, with a offset of zero indicating the leftmost bit of the leftmost byte of the source operand.

The length operand indicates the number of bits that are to be copied.

If the sum of the offset plus the length exceed the length of the source an "invalid operand length" exception will be raised.

Limitations: Neither the receiver nor the source operand can be a variable length substring.

The length of the receiver cannot exceed four bytes.

The offset must have a non-negative value.

The length operand must be an immediate value between 1 and 32.

Exceptions

Exception		Operands 1	2	3	4	Other
06	Addressing					
	01 Spacing addressing violation	X	X			

Exception		Operands 1	2	3	4	Other
	02 Boundary alignment violation	X	X			
	03 Range	X	X			
	06 Optimized addressability invalid	X	X			
08	Argument/parameter					
	01 Parameter reference violation	X	X	X	X	
10	Damage encountered					
	04 System object damage state					X
	44 Partial system object damage					X
1C	Machine-dependent exception					
	03 Machine storage limit exceeded					X
20	Machine support					
	02 Machine check					X
	03 Function check					X
22	Object access					
	02 Object destroyed	X	X			
	03 Object suspended	X	X			
24	Pointer specification					
	01 Pointer does not exist	X	X			
	02 Pointer type invalid	X	X			
2A	Program creation					
	06 Invalid operand type	X	X	X	X	
	07 Invalid operand attribute	X	X	X	X	
	08 Invalid operand value range	X	X	X	X	
	0A Invalid operand length	X	X			
	0C Invalid operand odt reference	X	X			
	0D Reserved bits are not zero	X	X			
2E	Resource control limit					
	01 user profile storage limit exceeded					X
36	Space management					
	01 space extension/truncation					X

1.30 Copy Bits with Left Logical Shift (CPYBTLLS)

Op Code (Hex)	Operand 1	Operand 2	Operand 3
102F	Receiver	Source	Shift control

Operand 1: Character variable scalar or numeric variable scalar.

Operand 2: Character scalar or numeric scalar.

Operand 3: Character(2) scalar or unsigned binary(2) scalar.

Description: This instruction copies the bit string value of the source operand to the bit string defined by the receiver operand with a left logical shift of the source bit string value under control of the shift control operand.

The operation results in copying the shifted bit string value of the source to the bit string of the receiver while padding the receiver with bit values of 0 and truncating bit values of the source as is appropriate for the specific operation.

No indication is given of truncation of bit values from the shifted source value. This is true whether the values truncated are 0 or 1.

The operation is performed such that the bit string of the source is considered to be extended on the left and right by an unlimited number of bit string positions of value 0. Additionally, a receiver bit string view (window) with the attributes of the receiver is considered to overlay this conceptual bit string value of the source starting at the leftmost bit position of the original source value. A left logical shift of the conceptual bit string value of the source is then performed relative to the receiver bit string view according to the shift criteria specified in the shift control operand. After the shift, the bit string value then contained within the receiver bit string view is copied to the receiver.

The source and the receiver can be either character or numeric. Any numeric operands are interpreted as logical character strings. Due to the operation being treated as a character string operation, the source operand may not be specified as a signed immediate operand. Additionally, for a source operand specified as an unsigned immediate value, only a 1-byte immediate value may be specified.

The shift control operand may be specified as an immediate operand, as a character(2) scalar, or as an unsigned binary(2) scalar. It provides an unsigned binary value indicating the number of bit positions for which the left logical shift of the source bit string value is to be performed. A zero value specifies no shift.

Operands 1 and 2 may be specified as variable length substring compound operands.

Operand 3 may not be specified as a variable length substring compound operand.

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for this instruction.

Exceptions

Exception		Operands 1	2	3	Other
06	Addressing				
	01 Spacing addressing violation	X	X	X	
	02 Boundary alignment violation	X	X	X	
	03 Range	X	X	X	
	06 Optimized addressability invalid	X	X	X	
08	Argument/parameter				
	01 Parameter reference violation	X	X	X	
10	Damage encountered				
	04 System object damage state				X
	44 Partial system object damage				X
1C	Machine-dependent exception				
	03 Machine storage limit exceeded				X
20	Machine support				
	02 Machine check				X
	03 Function check				X
22	Object access				
	02 Object destroyed	X	X	X	
	03 Object suspended	X	X	X	
24	Pointer specification				
	01 Pointer does not exist	X	X	X	
	02 Pointer type invalid	X	X	X	
2A	Program creation				
	06 Invalid operand type	X	X	X	
	07 Invalid operand attribute	X	X	X	
	08 Invalid operand value range	X	X	X	
	0A Invalid operand length	X	X	X	
	0C Invalid operand odt reference	X	X	X	
	0D Reserved bits are not zero	X	X	X	X
2E	Resource control limit				
	01 user profile storage limit exceeded				X
32	Scalar specification				
	01 Scalar type invalid	X	X	X	
36	Space management				
	01 space extension/truncation				X

1.31 Copy Bits with Right Arithmetic Shift (CPYBTRAS)

Op Code (Hex)	Operand 1	Operand 2	Operand 3
101B	Receiver	Source	Shift Control

Operand 1: Character variable or numeric variable scalar.

Operand 2: Character scalar or numeric scalar.

Operand 3: Character(2) scalar or unsigned binary(2) scalar.

Description: The instruction copies the bit string value of the source operand to the bit string defined by the receiver operand with a right arithmetic shift of the source bit string value under control of the shift control operand.

The operation results in copying the shifted bit string value of the source to the bit string of the receiver while padding the receiver with bit values of 0 or 1 depending on the high order bit value of the source, and truncating bit values of the source as is appropriate for the specific operation.

No indication is given of truncation of bit values from the shifted source value. This is true whether the values truncated are 0 or 1.

The operation is performed such that the bit string of the source is considered a signed numeric binary value, with the value of the sign bit of the source conceptually extended on the left an unlimited number of bit string positions. A right arithmetic shift of the conceptual bit string value of the source is then performed according to the shift criteria specified in the shift control operand. No indication is given of truncation of bit values from the shifted conceptual source value. This is true whether the values truncated are 0 or 1. After the shift, the conceptual bit string value is then copied to the receiver, right aligned.

Viewing the bit string value of the source and the bit string value copied to the receiver as signed numeric, the sign of the value copied to the receiver will be the same as the sign of the source.

A right shift of one bit position is equivalent to dividing the signed numeric bit string value of the source by 2 with rounding downward, and assigning a signed numeric bit string equivalent to that result to the receiver. For example, if the signed numeric view of the source bit string is +9, shifting one bit position right yields +4. However if the signed numeric view of the source bit string is -9, shifting one bit position right yields -5.

If all the significant bits of the conceptual source bit string are shifted out of the field, the resulting conceptual bit string value will be all zero bits for positive numbers, and all one bits for negative numbers.

The source and the receiver can be either character or numeric. Any numeric operands are interpreted as logical character strings. Due to the operation being treated as a character string operation, the source operand may not be specified as a signed immediate operand. Additionally, for a source operand specified as an unsigned immediate value, only a 1-byte immediate value may be specified.

The shift control operand may be specified as an immediate operand, as a character(2) scalar, or as a unsigned binary(2) scalar. It provides an unsigned binary value indicating the number of bit positions for which the right logical shift of the source bit string value is to be performed. A zero value specifies no shift.

Limitations: The following are limits that apply to the functions performed by this instruction.

Operands 1 and 2 may be specified as variable length substring compound operands.

Operand 3 may not be specified as a variable length substring compound operand.

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for this instruction.

Exceptions

Exception		Operands 1	2	3	Other
06	Addressing				
	01 Spacing addressing violation	X	X	X	
	02 Boundary alignment violation	X	X	X	
	03 Range	X	X	X	
	06 Optimized addressability invalid	X	X	X	
08	Argument/parameter				
	01 Parameter reference violation	X	X	X	
10	Damage encountered				
	04 System object damage state				X
	44 Partial system object damage				X
1C	Machine-dependent exception				
	03 Machine storage limit exceeded				X
20	Machine support				
	02 Machine check				X
	03 Function check				X
22	Object access				
	02 Object destroyed	X	X	X	
	03 Object suspended	X	X	X	
24	Pointer specification				
	01 Pointer does not exist	X	X	X	
	02 Pointer type invalid	X	X	X	
2A	Program creation				
	06 Invalid operand type	X	X	X	
	07 Invalid operand attribute	X	X	X	

Exception		Operands 1	2	3	Other
	08 Invalid operand value range	X	X	X	
	0A Invalid operand length	X	X	X	
	0C Invalid operand odt reference	X	X	X	
	0D Reserved bits are not zero	X	X	X	X
2E	Resource control limit				
	01 user profile storage limit exceeded				X
32	Scalar specification				
	01 Scalar type invalid	X	X	X	
36	Space management				
	01 space extension/truncation				X

1.32 Copy Bits with Right Logical Shift (CPYBTRLS)

Op Code (Hex)	Operand 1	Operand 2	Operand 3
103F	Receiver	Source	Shift control

Operand 1: Character variable scalar or numeric variable scalar.

Operand 2: Character scalar or numeric scalar.

Operand 3: Character(2) scalar or unsigned binary(2) scalar.

Description: This instruction copies the bit string value of the source operand to the bit string defined by the receiver operand with a right logical shift of the source bit string value under control of the shift control operand.

The operation results in copying the shifted bit string value of the source to the bit string of the receiver while padding the receiver with bit values of 0 and truncating bit values of the source as is appropriate for the specific operation.

No indication is given of truncation of bit values from the shifted source value. This is true whether the values truncated are 0 or 1.

The operation is performed such that the bit string of the source is considered to be extended on the left and right by an unlimited number of bit string positions of value 0. Additionally, a receiver bit string view (window) with the attributes of the receiver is considered to overlay this conceptual bit string value of the source starting at the leftmost bit position of the original source value. A right logical shift of the conceptual bit string value of the source is then performed relative to the receiver bit string view according to the shift criteria specified in the shift control operand. After the shift, the bit string value then contained within the receiver bit string view is copied to the receiver.

The source and the receiver can be either character or numeric. Any numeric operands are interpreted as logical character strings. Due to the operation being treated as a character string operation, the source operand may not be specified as a signed immediate operand. Additionally, for a source operand specified as an unsigned immediate value, only a 1-byte immediate value may be specified.

The shift control operand may be specified as an immediate operand, as a character(2) scalar, or as a unsigned binary(2) scalar. It provides an unsigned binary value indicating the number of bit positions for which the right logical shift of the source bit string value is to be performed. A zero value specifies no shift.

Limitations: The following are limits that apply to the functions performed by this instruction.

Operands 1 and 2 may be specified as variable length substring compound operands.

Operand 3 may not be specified as a variable length substring compound operand.

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for this instruction.

Exceptions

Exception		Operands 1	2	3	Other
06	Addressing				
	01 Spacing addressing violation	X	X	X	
	02 Boundary alignment violation	X	X	X	
	03 Range	X	X	X	
	06 Optimized addressability invalid	X	X	X	
08	Argument/parameter				
	01 Parameter reference violation	X	X	X	
10	Damage encountered				
	04 System object damage state				X
	44 Partial system object damage				X
1C	Machine-dependent exception				
	03 Machine storage limit exceeded				X
20	Machine support				
	02 Machine check				X
	03 Function check				X
22	Object access				
	02 Object destroyed	X	X	X	
	03 Object suspended	X	X	X	
24	Pointer specification				
	01 Pointer does not exist	X	X	X	
	02 Pointer type invalid	X	X	X	
2A	Program creation				
	06 Invalid operand type	X	X	X	
	07 Invalid operand attribute	X	X	X	
	08 Invalid operand value range	X	X	X	
	0A Invalid operand length	X	X	X	
	0C Invalid operand odt reference	X	X	X	
	0D Reserved bits are not zero	X	X	X	X
2E	Resource control limit				
	01 user profile storage limit exceeded				X
32	Scalar specification				
	01 Scalar type invalid	X	X	X	
36	Space management				

Exception	Operands 1	2	3	Other
01 space extension/truncation				X

1.33 Copy Bytes Left-Adjusted (CPYBLA)

Op Code (Hex)	Operand 1	Operand 2
10B2	Receiver	Source

Operand 1: Character variable scalar, numeric variable scalar, data-pointer-defined character scalar, or data-pointer-defined numeric scalar.

Operand 2: Character scalar, numeric scalar, data-pointer-defined character scalar, or data-pointer-defined numeric scalar.

Description: The logical string value of the source operand is copied to the logical string value of the receiver operand (no padding done). The operands can be either character or numeric. Any numeric operands are interpreted as logical character strings.

The length of the operation is equal to the length of the shorter of the two operands. The copying begins with the two operands left-adjusted and proceeds until the shorter operand has been copied.

Substring operand references that allow for a null substring reference (a length value of zero) may be specified for operands 1 and 2. The effect of specifying a null substring reference for either operand is that no result is set.

If either operand is a character variable scalar, it may have a length as great as 16776191 bytes.

Exceptions

Exception		Operands 1	Operands 2	Other
06	Addressing			
	01 Spacing addressing violation	X	X	
	02 Boundary alignment	X	X	
	03 Range	X	X	
	04 External data object not found	X	X	
	06 Optimized addressability invalid	X	X	
08	Argument/parameter			
	01 Parameter reference violation	X	X	
10	Damage encountered			
	04 System object damage state	X	X	X
	44 Partial system object damage	X	X	X
1C	Machine-dependent exception			
	03 Machine storage limit exceeded			X
20	Machine support			
	02 Machine check			X
	03 Function check			X

Exception		Operands 1	2	Other
22	Object access			
	01 Object not found	X	X	
	02 Object destroyed	X	X	
	03 Object suspended	X	X	
24	Pointer specification			
	01 Pointer does not exist	X	X	
	02 Pointer type invalid	X	X	
2A	Program creation			
	06 Invalid operand type	X	X	
	07 Invalid operand attribute	X	X	
	08 Invalid operand value range	X	X	
	0A Invalid operand length	X	X	
	0C Invalid operand odt reference	X	X	
	0D Reserved bits are not zero	X	X	X
2E	Resource control limit			
	01 user profile storage limit exceeded			X
32	Scalar specification			
	01 Scalar type invalid	X	X	
36	Space management			
	01 space extension/truncation			X

1.34 Copy Bytes Left-Adjusted with Pad (CPYBLAP)

Op Code (Hex)	Operand 1	Operand 2	Operand 3
10B3	Receiver	Source	Pad

Operand 1: Character variable scalar or numeric variable scalar, data-pointer-defined character scalar, or data-pointer-defined numeric scalar.

Operand 2: Character scalar, numeric scalar, data-pointer-defined character scalar, or data-pointer-defined numeric scalar.

Operand 3: Character scalar or numeric scalar.

Description: The logical string value of the source operand is copied to the logical string value of the receiver operand (padded if needed).

The operands can be either character or numeric. Any numeric operands are interpreted as logical character strings.

The length of the operation is equal to the length of the receiver operand. If the source operand is shorter than the receiver operand, the source operand is copied to the leftmost bytes of the receiver operand, and each excess byte of the receiver operand is assigned the single byte value in the pad operand. If the pad operand is more than 1 byte in length, only its leftmost byte is used. If the source operand is longer than the receiver operand, the leftmost bytes of the source operand (equal in length to the receiver operand) are copied to the receiver operand.

Substring operand references that allow for a null substring reference (a length value of zero) may be specified for operands 1 and 2. The effect of specifying a null substring reference for the source is that the bytes of the receiver are each set with the single byte value of the pad operand. The effect of specifying a null substring reference for the receiver is that no result is set.

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for operand 3.

If either of the first two operands is a character variable scalar, it may have a length as great as 16776191.

Exceptions

Exception		Operands 1	2	3	Other
06	Addressing				
	01 Spacing addressing violation	X	X	X	
	02 Boundary alignment	X	X	X	
	03 Range	X	X	X	
	04 External data object not found	X	X		
	06 Optimized addressability invalid	X	X	X	
08	Argument/parameter				

Exception		Operands 1	2	3	Other
	01 Parameter reference violation	X	X	X	
10	Damage encountered				
	04 System object damage state	X	X	X	X
	44 Partial system object damage	X	X	X	X
1C	Machine-dependent exception				
	03 Machine storage limit exceeded				X
20	Machine support				
	02 Machine check				X
	03 Function check				X
22	Object access				
	01 Object not found	X	X	X	
	02 Object destroyed	X	X	X	
	03 Object suspended	X	X	X	
24	Pointer specification				
	01 Pointer does not exist	X	X	X	
	02 Pointer type invalid	X	X	X	
2A	Program creation				
	06 Invalid operand type	X	X	X	
	07 Invalid operand attribute	X	X	X	
	08 Invalid operand value range	X	X	X	
	0A Invalid operand length	X	X		
	0C Invalid operand odt reference	X	X	X	X
	0D Reserved bits are not zero	X	X	X	X
2E	Resource control limit				
	01 user profile storage limit exceeded				X
32	Scalar specification				
	01 Scalar type invalid	X	X		
36	Space management				
	01 space extension/truncation				X

1.35 Copy Bytes Overlap Left-Adjusted (CPYBOLA)

Op Code (Hex)	Operand 1	Operand 2
10BA	Receiver	Source

Operand 1: Character variable scalar or numeric variable scalar.

Operand 2: Character variable scalar or numeric variable scalar.

Description: The logical string value of the source operand is copied to the logical string value of the receiver operand (no padding done). The operands can be either character or numeric. Any numeric operands are interpreted as logical character strings.

The length of the operation is equal to the length of the shorter of the two operands. The copying begins with the two operands left-adjusted and proceeds until the shorter operand has been copied. The excess bytes in the longer operand are not included in the operation.

Predictable results occur even if two operands overlap because the source operand is, in effect, first copied to an intermediate result.

Substring operand references that allow for a null substring reference (a length value of zero) may be specified for operands 1 and 2. The effect of specifying a null substring reference for either operand is that no result is set.

Exceptions

Exception		Operands 1	Operands 2	Other
06	Addressing			
	01 Spacing addressing violation	X	X	
	02 Boundary alignment	X	X	
	03 Range	X	X	
	06 Optimized addressability invalid	X	X	
08	Argument/parameter			
	01 Parameter reference violation	X	X	
10	Damage encountered			
	04 System object damage state	X	X	X
	44 Partial system object damage	X	X	X
1C	Machine-dependent exception			
	03 Machine storage limit exceeded			X
20	Machine support			
	02 Machine check			X
	03 Function check			X
22	Object access			
	01 Object not found	X	X	

Exception		Operands 1	2	Other
	02 Object destroyed	X	X	
	03 Object suspended	X	X	
24	Pointer specification			
	01 Pointer does not exist	X	X	
	02 Pointer type invalid	X	X	
2A	Program creation			
	06 Invalid operand type	X	X	
	07 Invalid operand attribute	X	X	
	08 Invalid operand value range	X	X	
	0A Invalid operand length	X	X	
	0C Invalid operand odt reference	X	X	
	0D Reserved bits are not zero	X	X	X
2E	Resource control limit			
	01 user profile storage limit exceeded			X
32	Scalar specification			
	01 Scalar type invalid	X	X	
36	Space management			
	01 space extension/truncation			X

1.36 Copy Bytes Overlap Left-Adjusted with Pad (CPYBOLAP)

Op Code (Hex)	Operand 1	Operand 2	Operand 3
10BB	Receiver	Source	Pad

Operand 1: Character variable scalar or numeric variable scalar.

Operand 2: Character variable scalar or numeric variable scalar.

Operand 3: Character scalar or numeric scalar.

Description: The logical string value of the source operand is copied to the logical string value of the receiver operand.

The operands can be either character or numeric. Any numeric operands are interpreted as logical character strings.

The length of the operation is equal to the length of the receiver operand. If the source operand is shorter than the receiver operand, the source operand is copied to the leftmost bytes of the receiver operand and each excess byte of the receiver operand is assigned the single byte value in the pad operand. If the pad operand is more than 1 byte in length, only its leftmost byte is used. If the source operand is longer than the receiver operand, the leftmost bytes of the source operand (equal in length to the receiver operand) are copied to the receiver operand.

Predictable results occur even if two operands overlap because the source operand is, in effect, first copied to an intermediate result.

Substring operand references that allow for a null substring reference (a length value of zero) may be specified for operands 1 and 2. The effect of specifying a null substring reference for the source is that the bytes of the receiver are each set with the single byte value of the pad operand. The effect of specifying a null substring reference for the receiver is that no result is set.

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for operand 3.

Exceptions

Exception		Operands 1	2	3	Other
06	Addressing				
	01 Spacing addressing violation	X	X	X	
	02 Boundary alignment	X	X	X	
	03 Range	X	X	X	
	06 Optimized addressability invalid	X	X	X	
08	Argument/parameter				
	01 Parameter reference violation	X	X	X	
10	Damage encountered				

Exception		Operands 1	2	3	Other
	04 System object damage state	X	X	X	X
	44 Partial system object damage	X	X	X	X
1C	Machine-dependent exception				
	03 Machine storage limit exceeded			X	X
20	Machine support				
	02 Machine check				X
	03 Function check				X
22	Object access				
	01 Object not found	X	X	X	
	02 Object destroyed	X	X	X	
	03 Object suspended	X	X	X	
24	Pointer specification				
	01 Pointer does not exist	X	X	X	
	02 Pointer type invalid	X	X	X	
2A	Program creation				
	06 Invalid operand type	X	X	X	
	07 Invalid operand attribute	X	X	X	
	08 Invalid operand value range	X	X	X	
	0A Invalid operand length	X	X		
	0C Invalid operand odt reference	X	X	X	
	0D Reserved bits are not zero	X	X	X	X
2E	Resource control limit				
	01 user profile storage limit exceeded				X
36	Space management				
	01 space extension/truncation				X

1.37 Copy Bytes Repeatedly (CPYBREP)

Op Code (Hex)	Operand 1	Operand 2
10BE	Receiver	Source

Operand 1: Numeric variable scalar or character variable scalar (fixed-length).

Operand 2: Numeric scalar or character scalar (fixed length).

Description: The logical string value of the source operand is repeatedly copied to the receiver operand until the receiver is filled. The operands can be either character or numeric. Any numeric operands are interpreted as logical character strings.

The operation begins with the two operands left-adjusted and continues until the receiver operand is completely filled. If the source operand is shorter than the receiver, it is repeatedly copied from left to right (all or in part) until the receiver operand is completely filled. If the source operand is longer than the receive operand, the leftmost bytes of the source operand (equal in length to the receiver operand) are copied to the receiver operand.

Substring operand references that allow for a null substring reference (a length value of zero) may be specified for operands 1 and 2. The effect of specifying a null substring reference for either operand is that no result is set.

If either operand is a character variable scalar, it may have a length as great as 16776191.

Exceptions

Exception		Operands 1	Operands 2	Other
06	Addressing			
	01 Spacing addressing violation	X	X	
	02 Boundary alignment	X	X	
	03 Range	X	X	
	06 Optimized addressability invalid	X	X	
08	Argument/parameter			
	01 Parameter reference violation	X	X	
10	Damage encountered			
	04 System object damage state	X	X	X
	44 Partial system object damage	X	X	X
1C	Machine-dependent exception			
	03 Machine storage limit exceeded			X
20	Machine support			
	02 Machine check			X
	03 Function check			X

Exception		Operands 1	2	Other
22	Object access			
	01 Object not found	X	X	
	02 Object destroyed	X	X	
	03 Object suspended	X	X	
24	Pointer specification			
	01 Pointer does not exist	X	X	
	02 Pointer type invalid	X	X	
2A	Program creation			
	06 Invalid operand type	X	X	
	07 Invalid operand attribute	X	X	
	08 Invalid operand value range	X	X	
	0A Invalid operand length	X	X	
	0C Invalid operand odt reference	X	X	
	0D Reserved bits are not zero	X	X	X
2E	Resource control limit			
	01 user profile storage limit exceeded			X
36	Space management			
	01 space extension/truncation			X

1.38 Copy Bytes Right-Adjusted (CPYBRA)

Op Code (Hex)	Operand 1	Operand 2
10B6	Receiver	Source

Operand 1: Character variable scalar, numeric variable scalar, data-pointer-defined character scalar, or data-pointer-defined numeric scalar.

Operand 2: Character scalar, numeric scalar, data-pointer-defined character scalar, or data-pointer-defined numeric scalar.

Description: The logical string value of the source operand is copied to the logical string value of the receiver operand (no padding done). The operands can be either character or numeric. Any numeric operands are interpreted as logical character strings.

The length of the operation is equal to the length of the shorter of the two operands. The rightmost bytes (equal to the length of the shorter of the two operands) of the source operand are copied to the rightmost bytes of the receiver operand. The excess bytes in the longer operand are not included in the operation.

Substring operand references that allow for a null substring reference (a length value of zero) may be specified for operands 1 and 2. The effect of specifying a null substring reference for either operand is that no result is set.

Exceptions

Exception		Operands 1	Operands 2	Other
06	Addressing			
	01 Spacing addressing violation	X	X	
	02 Boundary alignment	X	X	
	03 Range	X	X	
	04 External data object not found	X	X	
	06 Optimized addressability invalid	X	X	
08	Argument/parameter			
	01 Parameter reference violation	X	X	
10	Damage encountered			
	04 System object damage state	X	X	X
	44 Partial system object damage	X	X	X
1C	Machine-dependent exception			
	03 Machine storage limit exceeded			X
20	Machine support			
	02 Machine check			X
	03 Function check			X
22	Object access			

Exception		Operands 1	Operands 2	Other
	01 Object not found	X	X	
	02 Object destroyed	X	X	
	03 Object suspended	X	X	
24	Pointer specification			
	01 Pointer does not exist	X	X	
	02 Pointer type invalid	X	X	
2A	Program creation			
	06 Invalid operand type	X	X	
	07 Invalid operand attribute	X	X	
	08 Invalid operand value range	X	X	
	0A Invalid operand length	X	X	
	0C Invalid operand odt reference	X	X	
	0D Reserved bits are not zero	X	X	X
2E	Resource control limit			
	01 user profile storage limit exceeded			X
32	Scalar specification			
	01 Scalar type invalid	X	X	
36	Space management			
	01 space extension/truncation			X

1.39 Copy Bytes Right-Adjusted with Pad (CPYBRAP)

Op Code (Hex)	Operand 1	Operand 2	Operand 3
10B7	Receiver	Source	Pad

Operand 1: Character variable scalar, numeric variable scalar, data-pointer-defined character scalar, or data-pointer-defined numeric scalar.

Operand 2: Character scalar, numeric scalar, data-pointer-defined character scalar, or data-pointer-defined numeric scalar.

Operand 3: Character scalar or numeric scalar.

Description: The logical string value of the source operand is copied to the logical string value of the receiver operand (padded if needed). The operands can be either character or numeric. Any numeric operands are interpreted as logical character strings.

The length of the operation is equal to the length of the receiver operand. If the source operand is shorter than the receiver operand, the source operand is copied to the rightmost bytes of receiver operand, and each excess byte is assigned the single byte value in the pad operand. If the pad operand is more than 1 byte in length, only its leftmost byte is used. If the source operand is longer than the receiver operand, the rightmost bytes of the source operand (equal in length to the receiver operand) are copied to the receiver operand.

Substring operand references that allow for a null substring reference (a length value of zero) may be specified for operands 1 and 2. The effect of specifying a null substring reference for the source is that the bytes of the receiver are each set with the single byte value of the pad operand. The effect of specifying a null substring reference for the receiver is that no result is set.

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for operand 3.

Exceptions

Exception		Operands 1	2	3	Other
06	Addressing				
	01 Spacing addressing violation	X	X	X	
	02 Boundary alignment	X	X	X	
	03 Range	X	X	X	
	04 External data object not found	X	X		
	06 Optimized addressability invalid	X	X	X	
08	Argument/parameter				
	01 Parameter reference violation	X	X	X	
10	Damage encountered				
	04 System object damage state	X	X	X	X
	44 Partial system object damage	X	X	X	X

Exception		Operands 1	2	3	Other
1C	Machine-dependent exception				
	03 Machine storage limit exceeded				X
20	Machine support				
	02 Machine check				X
	03 Function check				X
22	Object access				
	01 Object not found	X	X	X	
	02 Object destroyed	X	X	X	
	03 Object suspended	X	X	X	
24	Pointer specification				
	01 Pointer does not exist	X	X	X	
	02 Pointer type invalid	X	X	X	
2A	Program creation				
	06 Invalid operand type	X	X	X	
	07 Invalid operand attribute	X	X	X	
	08 Invalid operand value range	X	X	X	
	0A Invalid operand length	X	X	X	
	0C Invalid operand odt reference	X	X	X	
	0D Reserved bits are not zero	X	X	X	X
2E	Resource control limit				
	01 user profile storage limit exceeded				X
32	Scalar specification				
	01 Scalar type invalid	X	X		
36	Space management				
	01 space extension/truncation				X

1.40 Copy Bytes to Bits Arithmetic (CPYBBTA)

Op Code (Hex)	Operand 1	Operand 2	Operand 3	Operand 4
104C	Receiver	Offset	Length	Source

Operand 1: Character variable scalar or numeric variable scalar.

Operand 2: Signed binary immediate or unsigned binary immediate.

Operand 3: Signed binary immediate or unsigned binary immediate.

Operand 4: Character variable scalar or numeric variable scalar.

Description: This instruction copies a byte string from the source operand to a bit string in the receiver operand.

The source operand is interpreted as a signed binary value and may be sign extended or truncated on the left to fit into the bit string in the receiver operand. No indication is given when truncation occurs.

The location of the bit string in the receiver operand is specified by the offset operand. The value of the offset operand specifies the bit offset from the start of the receiver operand to the start of the bit string. Thus, an offset operand value of 0 specifies that the bit string starts at the leftmost bit position of the receiver operand.

The length of the bit string in the receiver operand is specified by the length operand. The value of the length operand specifies the length of the bit string in bits.

Limitations: The following are limits that apply to the functions performed by this instruction.

Null operands may not be specified. Variable length substring operands may not be specified. Substring operand references that allow for a null substring reference (a length value of zero) may not be specified.

If the source operand and the bit string in the receiver operand overlap, the results are unpredictable.

A source operand longer than 4 bytes may not be specified.

If the offset operand is signed binary immediate, a negative value may not be specified.

A length operand with a value less than 1 or greater than 32 may not be specified.

The bit string specified by the offset operand and the length operand may not extend outside the receiver operand.

Exceptions

Exception		Operands 1	2	3	4	Other
06	Addressing					
	01 Spacing addressing violation	X			X	
	02 Boundary alignment violation	X			X	
	03 Range	X			X	
	06 Optimized addressability invalid	X			X	
08	Argument/parameter					
	01 Parameter reference violation	X			X	
10	Damage encountered					
	04 System object damage state					X
	44 Partial system object damage					X
1C	Machine-dependent exception					
	03 Machine storage limit exceeded					X
20	Machine support					
	02 Machine check					X
	03 Function check					X
22	Object access					
	02 Object destroyed	X			X	
	03 Object suspended	X			X	
24	Pointer specification					
	01 Pointer does not exist	X			X	
	02 Pointer type invalid	X			X	
2A	Program creation					
	06 Invalid operand type	X	X	X	X	
	07 Invalid operand attribute	X	X	X	X	
	08 Invalid operand value range	X	X	X	X	
	0A Invalid operand length	X			X	
	0C Invalid operand odt reference	X			X	
	0D Reserved bits are not zero	X	X	X	X	X
2E	Resource control limit					
	01 user profile storage limit exceeded					X
36	Space management					
	01 space extension/truncation					X

1.41 Copy Bytes to Bits Logical (CPYBBTL)

Op Code (Hex)	Operand 1	Operand 2	Operand 3	Operand 4
103C	Receiver	Offset	Length	Source

Operand 1: Character variable scalar or numeric variable scalar.

Operand 2: Signed binary immediate or unsigned binary immediate.

Operand 3: Signed binary immediate or unsigned binary immediate.

Operand 4: Character variable scalar or numeric variable scalar.

Description: This instruction copies a byte string from the source operand to a bit string in the receiver operand.

The source operand is interpreted as an unsigned binary value and may be padded on the left with 0's or truncated on the left to fit into the bit string in the receiver operand. No indication is given when truncation occurs.

The location of the bit string in the receiver operand is specified by the offset operand. The value of the offset operand specifies the bit offset from the start of the receiver operand to the start of the bit string. Thus, an offset operand value of 0 specifies that the bit string starts at the leftmost bit position of the receiver operand.

The length of the bit string in the receiver operand is specified by the length operand. The value of the length operand specifies the length of the bit string in bits.

Limitations: The following are limits that apply to the functions performed by this instruction.

Null operands may not be specified. Variable length substring operands may not be specified. Substring operand references that allow for a null substring reference (a length value of zero) may not be specified.

If the source operand and the bit string in the receiver operand overlap, the results are unpredictable.

A source operand longer than 4 bytes may not be specified.

If the offset operand is signed binary immediate, a negative value may not be specified.

A length operand with a value less than 1 or greater than 32 may not be specified.

The bit string specified by the offset operand and the length operand may not extend outside the receiver operand.

Exceptions

Exception		Operands 1	2	3	4	Other
06	Addressing					
	01 Spacing addressing violation	X			X	
	02 Boundary alignment violation	X			X	
	03 Range	X			X	
	06 Optimized addressability invalid	X			X	
08	Argument/parameter					
	01 Parameter reference violation	X			X	
10	Damage encountered					
	04 System object damage state					X
	44 Partial system object damage					X
1C	Machine-dependent exception					
	03 Machine storage limit exceeded					X
20	Machine support					
	02 Machine check					X
	03 Function check					X
22	Object access					
	02 Object destroyed	X			X	
	03 Object suspended	X			X	
24	Pointer specification					
	01 Pointer does not exist	X			X	
	02 Pointer type invalid	X			X	
2A	Program creation					
	06 Invalid operand type	X	X	X	X	
	07 Invalid operand attribute	X	X	X	X	
	08 Invalid operand value range	X	X	X	X	
	0A Invalid operand length	X			X	
	0C Invalid operand odt reference	X			X	
	0D Reserved bits are not zero	X	X	X	X	X
2E	Resource control limit					
	01 user profile storage limit exceeded					X
36	Space management					
	01 space extension/truncation					X

1.42 Copy Extended Characters Left-Adjusted With Pad (CPYECLAP)

Op Code (Hex)	Operand 1	Operand 2	Operand 3
1053	Receiver	Source	Pad

Operand 1: Data-pointer-defined character scalar.

Operand 2: Data-pointer-defined character scalar.

Operand 3: Character(3) scalar or null.

Description: The extended character string value of the source operand is copied to the receiver operand.

The operation is performed at the length of the receiver operand. If the source operand is shorter than the receiver, the source operand is copied to the left-most bytes of the receiver and the excess bytes of the receiver are assigned the appropriate value from the pad operand.

The pad operand, operand 3, is three bytes in length and has the following format:

- Pad operand — Char(3)
  - Single byte pad value — Char(1)
  - Double byte pad value — Char(2)

If the pad operand is more than three bytes in length, only its leftmost three bytes are used. Specifying a null pad operand results in default pad values of X'40', for single byte, and X'4040', for double byte, being used. The single byte pad value and the first byte of the double byte pad value cannot be either a shift out control character (SO ='0E'X) value or a shift in control character (SI='0F'X) value. Specification of such an invalid value results in the signaling of the scalar value invalid exception.

Operands 1 and 2 must be specified as Data Pointers which define either a simple (single byte) character data field or one of the extended (double byte) character data fields.

Support for usage of a Data Pointer defining an extended character scalar value is limited to this instruction. Usage of such a data pointer defined value on any other instruction is not supported and results in the signaling of the scalar type invalid exception.

For more information on support for extended character data fields, refer to the Set Data Pointer Attributes, Materialize Pointer, and Create Cursor instructions.

Four data types are supported for data pointer definition of extended (double byte) character fields, OPEN, EITHER, ONLYNS and ONLYS. Except for ONLYNS, the double byte character data must be surrounded by a shift out control character (SO ='0E'X) and a shift in control character (SI='0F'X).

- The ONLYNS field only contains double byte data with no SO, SI delimiters surrounding it.

- The ONLYS field can only contain double byte character data within a SO..SI pair.
- The EITHER field can consist of double byte character or single byte character data but only one type at a time. If double byte character data is present it must be surrounded by an SO .. SI pair.
- The OPEN field can consist of a mixture of double byte character and single byte character data. If double byte character data is present it must be surrounded by an SO .. SI pair.

Specifying an extended character value which violates the above restrictions results in the signaling of the invalid extended character data exception.

The valid copy operations which can be specified on this instruction are the following:

		Op 1			
		Onlyns	Onlys	Open	Either
0	Onlyns	yes	yes	yes	yes
p	Onlys	yes	yes	yes	yes
	Open	no	no	yes	no
2	Either	no	no	yes	yes

Figure 1-5. Valid copy operations for CPYECLAP

Specifying a copy operation other than the valid operations defined above results in the signaling of the invalid extended character operation exception.

When the copy operation is for a source of type ONLYNS (no SO/SI delimiters) being copied to a receiver which is not ONLYNS, SO and SI delimiters are implicitly added around the source value as part of the copy operation.

When the source value is longer than can be contained in the receiver, truncation is necessary and the following truncation rules apply:

1. Truncation is on the right (like simple character copy operations).
2. When the string to be truncated is a single byte character string, or an extended character string when the receiver is ONLYNS, bytes beyond those that fit into the receiver are truncated with no further processing needed.
3. When the string to be truncated is an extended character string and the receiver is not ONLYNS, the bytes that fall at the end of the receiver are truncated as follows:
   a. When the last byte that would fit in the receiver is the first byte of an extended character, that byte is truncated and replaced with an SI character.
   b. When the last byte that would fit in the receiver is the second byte of an extended character, both bytes of that extended character are truncated and replaced with a SI character followed by a single byte pad value. This type of truncation can only occur when converting to an OPEN field.

When the source value is shorter than that which can be contained in the receiver, padding is necessary. One of three types of padding is performed:

1. Double byte (DB) - the source value is padded on the right with double byte pad values out to the length of the receiver.
2. Double byte concatenated with a SI value (DB||SI) - the source double byte value is padded on the right with double byte pad values out to the second to last byte of the receiver and an SI delimiter is placed in the last byte of the receiver.
3. Single byte (SB) - the source value is padded on the right with single byte pad values out to the length of the receiver.

The type of padding performed is determined by the type of operands involved in the operation:

1. If the receiver is ONLYNS, DB padding is performed.
2. If the receiver is ONLYS, DB||SI padding will be performed.
3. If the receiver is EITHER and the source contained a double byte value, DB||SI padding is performed.
4. If the receiver is EITHER and the source contained a single byte value, SB padding is performed.
5. If the receiver is OPEN, SB padding is performed.

The above padding rules cover all the operand combinations which are allowed on the instruction. A complete understanding of the operand combinations allowed (prior diagram), and the values which can be contained in the different operand types is necessary to appreciate that these rules do cover all the valid combinations.

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for operand 3.

Exceptions

Exception		Operands 1	2	3	Other
06	Addressing				
	01 Spacing addressing violation	X	X	X	
	02 Boundary alignment	X	X	X	
	03 Range	X	X	X	
	04 External data object not found	X	X		
	06 Optimized addressability invalid	X	X	X	
08	Argument/parameter				
	01 Parameter reference violation	X	X	X	
0C	Computation				
	12 Invalid extended character data		X		
	13 Invalid extended character operation				X
10	Damage encountered				
	04 System object damage state	X	X	X	X
	44 Partial system object damage	X	X	X	X

Exception		Operands 1	2	3	Other
1C	Machine-dependent exception				
	03 Machine storage limit exceeded				X
20	Machine support				
	02 Machine check				X
	03 Function check				X
22	Object access				
	01 Object not found	X	X	X	
	02 Object destroyed	X	X	X	
	03 Object suspended	X	X	X	
24	Pointer specification				
	01 Pointer does not exist	X	X	X	
	02 Pointer type invalid	X	X	X	
2A	Program creation				
	06 Invalid operand type	X	X	X	
	07 Invalid operand attribute	X	X	X	
	08 Invalid operand value range	X	X	X	
	0A Invalid operand length	X	X		
	0C Invalid operand odt reference	X	X	X	X
	0D Reserved bits are not zero	X	X	X	X
2E	Resource control limit				
	01 user profile storage limit exceeded				X
32	Scalar specification				
	01 Scalar type invalid	X	X		
	01 Scalar value invalid			X	
36	Space management				
	01 space extension/truncation				X

1.43 Copy Hex Digit Numeric to Numeric (CPYHEXNN)

Op Code (Hex)	Operand 1	Operand 2
1092	Receiver	Source

Operand 1: Numeric variable scalar or character variable scalar (fixed-length).

Operand 2: Numeric scalar or character scalar (fixed-length).

Description: The numeric hex digit value (rightmost 4 bits) of the leftmost byte referred to by the source operand is copied to the numeric hex digit value (rightmost 4 bits) of the leftmost byte referred to by the receiver operand. The operands can be either character strings or numeric. Any numeric operands are interpreted as logical character strings.

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for this instruction.

Exceptions

Exception		Operands 1	Operands 2	Other
06	Addressing			
	01 Spacing addressing violation	X	X	
	02 Boundary alignment	X	X	
	03 Range	X	X	
	06 Optimized addressability invalid	X	X	
08	Argument/parameter			
	01 Parameter reference violation	X	X	
10	Damage encountered			
	04 System object damage state	X	X	X
	44 Partial system object damage	X	X	X
1C	Machine-dependent exception			
	03 Machine storage limit exceeded			X
20	Machine support			
	02 Machine check			X
	03 Function check			X
22	Object access			
	01 Object not found	X	X	
	02 Object destroyed	X	X	
	03 Object suspended	X	X	
24	Pointer specification			
	01 Pointer does not exist	X	X	
	02 Pointer type invalid	X	X	

Exception		Operands 1	2	Other
2A	Program creation			
	06 Invalid operand type	X	X	
	07 Invalid operand attribute	X	X	
	08 Invalid operand value range	X	X	
	0A Invalid operand length	X	X	
	0C Invalid operand odt reference	X	X	
	0D Reserved bits are not zero	X	X	X
2E	Resource control limit			
	01 user profile storage limit exceeded			X
36	Space management			
	01 space extension/truncation			X

1.44 Copy Hex Digit Numeric to Zone (CPYHEXNZ)

Op Code (Hex)	Operand 1	Operand 2
1096	Receiver	Source

Operand 1: Numeric variable scalar or character variable scalar (fixed-length).

Operand 2: Numeric scalar or character scalar (fixed-length).

Description: The numeric hex digit value (rightmost 4 bits) of the leftmost byte referred to by the source operand is copied to the numeric hex digit value (rightmost 4 bits) of the leftmost byte referred to by the receiver operand. The operands can be either character strings or numeric. Any numeric operands are interpreted as logical character strings.

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for this instruction.

Exceptions

Exception		Operands 1	Operands 2	Other
06	Addressing			
	01 Spacing addressing violation	X	X	
	02 Boundary alignment	X	X	
	03 Range	X	X	
	06 Optimized addressability invalid	X	X	
08	Argument/parameter			
	01 Parameter reference violation	X	X	
10	Damage encountered			
	04 System object damage state	X	X	X
	44 Partial system object damage	X	X	X
1C	Machine-dependent exception			
	03 Machine storage limit exceeded			X
20	Machine support			
	02 Machine check			X
	03 Function check			X
22	Object access			
	01 Object not found	X	X	
	02 Object destroyed	X	X	
	03 Object suspended	X	X	
24	Pointer specification			
	01 Pointer does not exist	X	X	
	02 Pointer type invalid	X	X	

Exception		Operands 1	2	Other
2A	Program creation			
	06 Invalid operand type	X	X	
	07 Invalid operand attribute	X	X	
	08 Invalid operand value range	X	X	
	0A Invalid operand length	X	X	
	0C Invalid operand odt reference	X	X	
	0D Reserved bits are not zero	X	X	X
2E	Resource control limit			
	01 user profile storage limit exceeded			X
36	Space management			
	01 space extension/truncation			X

1.45 Copy Hex Digit Zone To Numeric (CPYHEXZN)

Op Code (Hex)	Operand 1	Operand 2
109A	Receiver	Source

Operand 1: Numeric variable scalar or character variable scalar (fixed-length).

Operand 2: Numeric scalar or character scalar (fixed-length).

Description: The zone hex digit value (leftmost 4 bits) of the leftmost byte referred to by the source operand is copied to the numeric hex digit value (rightmost 4 bits) of the leftmost byte referred to by the receiver operand.

The operands can be either character strings or numeric. Any numeric operands are interpreted as logical character strings.

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for this instruction.

Exceptions

Exception		Operands 1	Operands 2	Other
06	Addressing			
	01 Spacing addressing violation	X	X	
	02 Boundary alignment	X	X	
	03 Range	X	X	
	06 Optimized addressability invalid	X	X	
08	Argument/parameter			
	01 Parameter reference violation	X	X	
10	Damage encountered			
	04 System object damage state	X	X	X
	44 Partial system object damage	X	X	X
1C	Machine-dependent exception			
	03 Machine storage limit exceeded			X
20	Machine support			
	02 Machine check			X
	03 Function check			X
22	Object access			
	01 Object not found	X	X	
	02 Object destroyed	X	X	
	03 Object suspended	X	X	
24	Pointer specification			
	01 Pointer does not exist	X	X	
	02 Pointer type invalid	X	X	

Exception		Operands 1	2	Other
2A	Program creation			
	06 Invalid operand type	X	X	
	07 Invalid operand attribute	X	X	
	08 Invalid operand value range	X	X	
	0A Invalid operand length	X	X	
	0C Invalid operand odt reference	X	X	
	0D Reserved bits are not zero	X	X	X
2E	Resource control limit			
	01 user profile storage limit exceeded			X
36	Space management			
	01 space extension/truncation			X

1.46 Copy Hex Digit Zone To Zone (CPYHEXZZ)

Op Code (Hex)	Operand 1	Operand 2
109E	Receiver	Source

Operand 1: Numeric variable scalar or character variable scalar (fixed-length).

Operand 2: Numeric scalar or character scalar (fixed-length).

Description: The zone hex digit value (leftmost 4 bits) of the leftmost byte referred to by the source operand is copied to the zone hex digit value (leftmost 4 bits) of the leftmost byte referred to by the receiver operand.

The operands can be either character strings or numeric. Any numeric operands are interpreted as logical character strings.

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for this instruction.

Exceptions

Exception		Operands 1	Operands 2	Other
06	Addressing			
	01 Spacing addressing violation	X	X	
	02 Boundary alignment	X	X	
	03 Range	X	X	
	06 Optimized addressability invalid	X	X	
08	Argument/parameter			
	01 Parameter reference violation	X	X	
10	Damage encountered			
	04 System object damage state	X	X	X
	44 Partial system object damage	X	X	X
1C	Machine-dependent exception			
	03 Machine storage limit exceeded			X
20	Machine support			
	02 Machine check			X
	03 Function check			X
22	Object access			
	01 Object not found	X	X	
	02 Object destroyed	X	X	
	03 Object suspended	X	X	
24	Pointer specification			
	01 Pointer does not exist	X	X	
	02 Pointer type invalid	X	X	

Exception		Operands 1	2	Other
2A	Program creation			
	06 Invalid operand type	X	X	
	07 Invalid operand attribute	X	X	
	08 Invalid operand value range	X	X	
	0A Invalid operand length	X	X	
	0C Invalid operand odt reference	X	X	
	0D Reserved bits are not zero	X	X	X
2E	Resource control limit			
	01 user profile storage limit exceeded			X
36	Space management			
	01 space extension/truncation			X

1.47 Copy Numeric Value (CPYNV)

Op Code (Hex)	Operand 1	Operand 2
1042	Receiver	Source

Operand 1: Numeric variable scalar or data-pointer-defined numeric scalar.

Operand 2: Numeric scalar or data pointer-defined-numeric scalar.

Optional Forms

Mnemonic	Op Code (Hex)	Form Type
CPYNVR	1242	Round
CPYNVI	1842	Indicator
CPYNVIR	1A42	Indicator, Round
CPYNVB	1C42	Branch
CPYNVBR	1E42	Branch, Round

Extender: Branch or indicator options.

If the branch or indicator option is specified in the op code, the extender field must be present along with one to four branch targets (for branch options) or one to four indicator operands (for indicator options). The branch or indicator operands immediately follow the last operand listed above. See Chapter 1. "Introduction" for the encoding of the extender field and the allowed syntax of the branch and indicator operands.

Description: The numeric value of the source operand is copied to the numeric receiver operand.

Both operands must be numeric. If necessary, the source operand is converted to the same type as the receiver operand before being copied to the receiver operand. The source value is adjusted to the length of the receiver operand, aligned at the assumed decimal point of the receiver operand, or both before being copied to it. If significant digits are truncated on the left end of the source value, a size exception is signaled. When the receiver is binary this size exception may be suppressed by using the suppress binary size exception program attribute on the CRTPG instruction.

If a decimal to binary conversion causes a size exception to be signaled or if the size exception is suppressed the binary value contains the correct truncated result only if the decimal value contains 15 or fewer significant nonfractional digits.

Conversions between floating-point integers and integer formats (binary or decimal with no fractional digits) is exact, except when an exception is signaled.

An invalid floating-point conversion exception is signaled when an attempt is made to convert from floating-point to binary or decimal and the result would represent infinity or NaN, or nonzero digits would be truncated from the left end of the resultant value.

For the optional round form of the instruction, a floating-point receiver operand is invalid.

For a fixed-point operation, if significant digits are truncated from the left end of the source value, a size exception is signaled. When the receiver is binary this size exception may be suppressed by using the suppress binary size exception program attribute on the CRTPG instruction.

For a floating-point receiver, if the exponent of the resultant value is too large or too small to be represented in the receiver field, the floating-point overflow and floating-point underflow exceptions are signaled, respectively.

Resultant Conditions: Positive, negative, or zero-The algebraic value of the numeric scalar receiver operand is either positive, negative, or zero. Unordered-The value assigned a floating-point receiver operand is NaN.

Exceptions

Exception		Operands 1	Operands 2	Other
06	Addressing			
	01 Spacing addressing violation	X	X	
	02 Boundary alignment	X	X	
	03 Range	X	X	
	04 External data object not found	X	X	
	06 Optimized addressability invalid	X	X	
08	Argument/parameter			
	01 Parameter reference violation	X	X	
0C	Computation			
	02 Decimal data		X	
	06 Floating-point overflow	X		
	07 Floating-point underlow	X		
	09 Floating-point invalid operand	X		X
	0A Size	X		
	0C Invalid floatin-point conversion	X		
	0A Floating-point inexact result	X		
10	Damage encountered			
	04 System object damage state	X	X	X
	44 Partial system object damage	X	X	X
1C	Machine-dependent exception			
	03 Machine storage limit exceeded			X
20	Machine support			
	02 Machine check			X
	03 Function check			X
22	Object access			

Exception		Operands 1	2	Other
	01 Object not found	X	X	
	02 Object destroyed	X	X	X
	03 Object suspended	X	X	X
24	Pointer specification			
	01 Pointer does not exist	X	X	X
	02 Pointer type invalid	X	X	X
2A	Program creation			
	05 Invalid op code extender field			X
	06 Invalid operand type	X	X	
	07 Invalid operand attribute	X	X	
	08 Invalid operand value range	X	X	
	09 Invalid branch target operand			X
	0C Invalid operand odt reference	X	X	X
	0D Reserved bits are not zero	X	X	X
2C	Program execution			
	04 Invalid branch target			X
2E	Resource control limit			
	01 user profile storage limit exceeded			X
32	Scalar specification			
	01 Scalar type invalid		X	X
36	Space management			
	01 space extension/truncation			X

1.48 Divide (DIV)

Op Code (Hex)	Operand 1	Operand 2	Operand 3
104F	Quotient	Dividend	Divisor

Operand 1: Numeric variable scalar.

Operand 2: Numeric scalar.

Operand 3: Numeric scalar.

Optional Forms

Mnemonic	Op Code (Hex)	Form Type
DIVS	114F	Short
DIVR	124F	Round
DIVSR	134F	Short, Round
DIVI	184F	Indicator
DIVIS	194F	Indicator, Short
DIVIR	1A4F	Indicator, Round
DIVISR	1B4F	Indicator, Short, Round
DIVB	1C4F	Branch
DIVBS	1D4F	Branch, Short
DIVBR	1E4F	Branch, Round
DIVBSR	1F4F	Branch, Short, Round

If the short instruction option is indicated in the op code, operand 1 is used as the first and second operational operands (receiver and first source operand). Operand 2 is used as the third operational operand (second source operand).

Extender: Branch or indicator options.

If the branch or indicator option is specified in the op code, the extender field must be present along with one to four branch targets (for branch options) or one to four indicator operands (for indicator options). The branch or indicator operands will immediately follow the last operand listed above. See Chapter 1. "Introduction" for the encoding of the extender field and the allowed syntax of the branch and indicator operands.

Description: The Quotient is the result of dividing the Dividend by the Divisor.

Operands can have floating-point, packed or zoned decimal, signed or unsigned binary type.

Source operands are the Dividend and Divisor. The receiver operand is the Quotient.

If operands are not of the same type, source operands are converted according to the following rules:

1. If any one of the operands has floating point type, source operands are converted to floating point type.

2. Otherwise, if any one of the operands has zoned or packed decimal type, source operands are converted to packed decimal.
3. Otherwise, the binary operands are converted to a like type. Note: unsigned binary(2) scalars are logically treated as signed binary(4) scalars.

Source operands are divided according to their type. Floating point operands are divided using floating point division. Packed decimal operands are divided using packed decimal division. Unsigned binary division is used with unsigned source operands. Signed binary operands are divided using two's complement binary division.

Better performance can be obtained if all operands have the same type. Signed and unsigned binary division execute faster than either packed decimal or floating point division.

Decimal operands used in floating-point operations cannot contain more than 15 total digit positions.

If the divisor has a numeric value of zero, a zero divide or floating-point zero divide exception is signaled respectively for fixed-point versus floating-point operations. If the dividend has a value of zero, the result of the division is a zero quotient value.

If the divisor has a numeric value of 0, a zero divide exception is signaled. If the dividend has a value of 0, the result of the division is a zero value quotient.

For a decimal operation, the precision of the result of the divide operation is determined by the number of fractional digit positions specified for the quotient. In other words, the divide operation will be performed so as to calculate a resultant quotient of the same precision as that specified for the quotient operand. If necessary, internal alignment of the assumed decimal point for the dividend and divisor operands is performed to ensure the correct precision for the resultant quotient value. These internal alignments are not subject to detection of the decimal point alignment exception. An internal quotient value will be calculated for any combination of decimal attributes which may be specified for the instruction's operands. However, the assignment of the result to the quotient operand is subject to detection of the size exception thereby limiting the assignment to, at most, the rightmost 31 digits of the calculated result.

Floating-point division uses exponent subtraction and significand division.

If the dividend operand is shorter than the divisor operand, it is logically adjusted to the length of the divisor operand.

For fixed-point computations and for the significand division of a floating-point computation, the division operation is performed according to the rules of algebra. Unsigned binary is treated as a positive number for the algebra.

For a floating-point computation, the operation is performed as if to infinite precision.

The result of the operation is copied into the quotient operand. If this operand is not the same type as that used in performing the operation, the resultant value is converted to its type. If necessary, the resultant value is adjusted to the

length of the quotient operand, aligned at the assumed decimal point of the quotient operand, or both before being copied to it.

If significant digits are truncated on the left end of the resultant value, a size exception is signaled.

If a decimal to binary conversion causes a size exception to be signaled, the binary value contains the correct truncated result only if the decimal value contains 15 or fewer significant nonfractional digits.

For the optional round form of the instruction, specification of a floating-point receiver operand is invalid.

For fixed-point operations in programs that request to be notified of size exceptions, if nonzero digits are truncated from the left end of the resultant value, a size exception is signaled.

For floating-point operations that involve a fixed-point receiver field, if nonzero digits would be truncated from the left end of the resultant value, an invalid floating-point conversion exception is signaled.

For a floating-point quotient operand, if the exponent of the resultant value is either too large or too small to be represented in the quotient field, the floating-point overflow and floating-point underflow exceptions are signaled, respectively.

Resultant Conditions: Positive, negative, or zero-The algebraic value of the numeric scalar quotient is positive, negative, or zero. Unordered-The value assigned a floating-point quotient operand is NaN.

Exceptions

Exception		Operands 1	2	3	Other
06	Addressing				
	01 Spacing addressing violation	X	X	X	
	02 Boundary alignment	X	X	X	
	03 Range	X	X	X	
	06 Optimized addressability invalid	X	X	X	
08	Argument/parameter				
	01 Parameter reference violation	X	X	X	
0C	Computation				
	02 Decimal data		X	X	
	06 Floating-point overflow	X			
	07 Floating-point underlow	X			
	09 Floating-point invalid operand		X	X	X
	0A Size	X			
	0B Zero divide		X		
	0C Invalid floatin-point conversion	X			
	0D Floating-point inexact result	X			

Exception		Operands 1	2	3	Other
	0E Floating-point divide by zero			X	
10	Damage encountered				
	04 System object damage state	X	X	X	X
	44 Partial system object damage	X	X	X	X
1C	Machine-dependent exception				
	03 Machine storage limit exceeded				X
20	Machine support				
	02 Machine check				X
	03 Function check				X
22	Object access				
	01 Object not found	X	X	X	
	02 Object destroyed	X	X	X	
	03 Object suspended	X	X	X	
24	Pointer specification				
	01 Pointer does not exist	X	X	X	
	02 Pointer type invalid	X	X	X	
2A	Program creation				
	05 Invalid op code extender field				X
	06 Invalid operand type	X	X	X	
	07 Invalid operand attribute	X	X	X	
	08 Invalid operand value range	X	X	X	
	09 Invalid branch target operand				X
	0C Invalid operand odt reference	X	X	X	
	0D Reserved bits are not zero	X	X	X	X
2C	Program execution				
	04 Invalid branch target				X
2E	Resource control limit				
	01 user profile storage limit exceeded				X
36	Space management				
	01 space extension/truncation				X

1.49 Divide with Remainder (DIVREM)

Op Code (Hex)	Operand 1	Operand 2	Operand 3	Operand 4
1074	Quotient	Dividend	Divisor	Remainder

Operand 1: Numeric variable scalar.

Operand 2: Numeric scalar.

Operand 3: Numeric scalar.

Operand 4: Numeric variable scalar.

Optional Forms

(The optional forms apply to the quotient only.)

Mnemonic	Op Code (Hex)	Form Type
DIVREMS	1174	Short
DIVREMR	1274	Round
DIVREMSR	1374	Short, Round
DIVREMI	1874	Indicator
DIVREMIS	1974	Indicator, Short
DIVREMIR	1A74	Indicator, Round
DIVREMISR	1B74	Indicator, Short, Round
DIVREMB	1C74	Branch
DIVREMBS	1D74	Branch, Short
DIVREMBR	1E74	Branch, Round
DIVREMBSR	1F74	Branch, Short, Round

If the short instruction option is indicated in the op code, operand 1 is used as the first and second operational operands (receiver and first source operand). Operand 2 is used as the third operational operand (second source operand).

Extender: Branch or indicator options.

If the branch or indicator option is specified in the op code, the extender field must be present along with one to three branch targets (for branch options) or one to three indicator operands (for indicator options). The branch or indicator operands immediately follow the last operand listed above. See Chapter 1. "Introduction" for the encoding of the extender field and the allowed syntax of the branch and indicator operands.

Description: The Quotient is the result of dividing the Dividend by the Divisor. The Remainder is the Dividend minus the product of the Divisor and Quotient.

Operands can have packed or zoned decimal, signed or unsigned binary type.

Source operands are the Dividend and Divisor. The receiver operands are the Quotient and Remainder.

If operands are not of the same type, source operands are converted according to the following rules:

1. If any one of the operands has zoned or packed decimal type, source operands are converted to packed decimal.
2. Otherwise, the binary operands are converted to a like type. Note: unsigned binary(2) scalars are logically treated as signed binary(4) scalars.

Source operands are divided according to their type. Packed decimal operands are divided using packed decimal division. Unsigned binary division is used with unsigned source operands. Signed binary operands are divided using two's complement binary division.

Better performance can be obtained if all operands have the same type. Signed and unsigned binary division execute faster than packed decimal division.

Floating-point is not supported for this instruction.

If the divisor operand has a numeric value of 0, a zero divide exception is signaled. If the dividend operand has a value of 0, the result of the division is a zero value quotient and remainder.

For a decimal operation, the precision of the result of the divide operation is determined by the number of fractional digit positions specified for the quotient. In other words, the divide operation will be performed so as to calculate a resultant quotient of the same precision as that specified for the quotient operand. If necessary, internal alignment of the assumed decimal point for the dividend and divisor operands is performed to ensure the correct precision for the resultant quotient value. These internal alignments are not subject to detection of the decimal point alignment exception. An internal quotient value will be calculated for any combination of decimal attributes which may be specified for the instruction's operands. However, the assignment of the result to the quotient operand is subject to detection of the size exception thereby limiting the assignment to, at most, the rightmost 31 digits of the calculated result.

If the dividend operand is shorter than the divisor operand, it is logically adjusted to the length of the divisor operand.

The division operation is performed according to the rules of algebra. Unsigned binary is treated as a positive number for the algebra. The quotient result of the operation is copied into the quotient operand. If this operand is not the same type as that used in performing the operation, the resultant value is converted to its type. If necessary, the resultant value is adjusted to the length of the quotient operand, aligned at the assumed decimal point of the quotient operand, or both before being copied to it. If significant digits are truncated on the left end of the resultant value, a size exception is signaled.

After the quotient numeric value has been determined, the numeric value of the remainder operand is calculated as follows:

```
Remainder = Dividend - (Quotient*Divisor)
```

If the optional round form of this instruction is being used, the rounding applies to the quotient but not the remainder. The quotient value used to calculate the remainder is the resultant value of the division. The resultant value of the calculation is copied into the remainder operand. The sign of the remainder is the same as that of the dividend operand unless the remainder has a value of 0, in which case its sign is positive. If the remainder operand is not the same type as

that used in performing the operation, the resultant value is converted to its type. If necessary, the resultant value is adjusted to the length of the remainder operand, aligned at the assumed decimal point of the remainder operand, or both before being copied to it. If significant digits are truncated off the left end of the resultant value, a size exception is signaled.

If a decimal to binary conversion causes a size exception to be signaled (in programs that request size exceptions to be signalled), the binary value contains the correct truncated result only if the decimal value contains 15 or fewer significant nonfractional digits.

Resultant Conditions: The algebraic value of the numeric scalar quotient is positive, negative, or 0.

Exceptions

Exception		Operands 1	2	3	4	Other
06	Addressing					
	01 Spacing addressing violation	X	X	X	X	
	02 Boundary alignment	X	X	X	X	
	03 Range	X	X	X	X	
	06 Optimized addressability invalid	X	X	X	X	
08	Argument/parameter					
	01 Parameter reference violation	X	X	X	X	
0C	Computation					
	02 Decimal data		X	X		
	0A Size	X			X	
	0B Zero divide			X		
10	Damage encountered					
	04 System object damage state	X	X	X	X	X
	44 Partial system object damage	X	X	X	X	X
1C	Machine-dependent exception					
	03 Machine storage limit exceeded					X
20	Machine support					
	02 Machine check					X
	03 Function check					X
22	Object access					
	01 Object not found	X	X	X	X	
	02 Object destroyed	X	X	X	X	
	03 Object suspended	X	X	X	X	
24	Pointer specification					
	01 Pointer does not exist	X	X	X	X	
	02 Pointer type invalid	X	X	X	X	

Exception		Operands 1	2	3	4	Other
2A	Program creation					
	05 Invalid op code extender field					X
	06 Invalid operand type	X	X	X	X	
	07 Invalid operand attribute	X	X	X	X	
	08 Invalid operand value range	X	X	X	X	
	09 Invalid branch target operand					X
	0C Invalid operand odt reference	X	X	X	X	
	0D Reserved bits are not zero	X	X	X	X	X
2C	Program execution					
	04 Invalid branch target					X
2E	Resource control limit					
	01 user profile storage limit exceeded					X
36	Space management					
	01 space extension/truncation					X

1.50 Edit (EDIT)

Op Code (Hex)	Operand 1	Operand 2	Operand 3
10E3	Receiver	Source	Edit Mask

Operand 1: Character variable scalar or data-pointer-defined character scalar.

Operand 2: Numeric scalar or data-pointer-defined numeric scalar.

Operand 3: Character variable scalar or data-pointer-defined character scalar.

Description: The value of a numeric scalar is transformed from its internal form to character form suitable for display at a source/sink device. The following general editing functions can be performed during transforming of the source operand to the receiver operand:

- Unconditional insertion of a source value digit with a zone as a function of the source value's algebraic sign
- Unconditional insertion of a mask operand character string
- Conditional insertion of one of two possible mask operand character strings as a function of the source value's algebraic sign
- Conditional insertion of a source value digit or a mask operand replacement character as a function of source value leading zero suppression
- Conditional insertion of either a mask operand character string or a series of replacement characters as a function of source value leading zero suppression
- Conditional floating insertion of one of two possible mask operand character strings as a function of both the algebraic sign of the source value and leading zero suppression

The operation is performed by transforming the source (operand 2) under control of the edit mask (operand 3) and placing the result in the receiver (operand 1).

The mask operand (operand 3) is limited to no more than 256 bytes.

Mask Syntax: The source field is converted to packed decimal format. The edit mask contains both control character and data character strings. Both the edit mask and the source fields are processed left to right, and the edited result is placed in the result field from left to right. If the number of digits in the source field is even, the four high-order bits of the source field are ignored and not checked for validity. All other source digits as well as the sign are checked for validity, and a decimal data exception is signaled when one is invalid. Overlapping of any of these fields gives unpredictable results.

Nine fixed value control characters can be in the edit mask, hex AA through hex AD and hex AF through hex B3. Four of these control characters specify strings of characters to be inserted into the result field under certain conditions; and the other five indicate that a digit from the source field should be checked and the appropriate action taken.

One variable value control character can be in the edit mask. This control character indicates the end of a string of characters. The value of the end-of-string character can vary with each execution of the instruction and is determined by the value of the first character in the edit mask. If the first character of the edit mask is a value less than hex 40, then that value is used as the end-of-string character. If the first character of the edit mask is a value equal to or greater than hex 40, then hex AE is used as the end-of-string character.

A significance indicator is set to the off state at the start of the execution of this instruction. It remains in this state until a nonzero source digit is encountered in the source field or until one of the four unconditional digits (hex AA through hex AD) or an unconditional string (hex B3) is encountered in the edit mask.

When significance is detected, the selected floating string is overlaid into the result field immediately before (to the left of) the first significant result character.

When the significance indicator is set to the on state, the first significant result character has been reached. The state of the significance indicator determines whether the fill character or a digit from the source field is to be inserted into the result field for conditional digits and characters in conditional strings specified in the edit mask field. The fill character is a hex 40 until it is replaced by the first character following the floating string specification control character (hex B1).

When the significance indicator is in the off state:

- A conditional digit control character in the edit mask causes the fill character to be moved to the result field.
- A character in a conditional string in the edit mask causes the fill character to be moved to the result field.

When the significance indicator is in the on state:

- A conditional digit control character in the edit mask causes a source digit to be moved to the result field.
- A character in a conditional string in the edit mask is moved to the result field.

The following control characters are found in the edit mask field.

End-of-String Character: One of these control characters (a value less than hex 40 or hex AE) indicates the end of a character string and must be present even if the string is null.

Static Field Character:

Hex AF This control character indicates the start of a static field. A static field is used to indicate that one of two mask character strings immediately following this character is to be inserted into the result field, depending upon the algebraic sign of the source field. If the sign is positive, the first string is to be inserted into the result field; if the sign is negative, the second string is to be inserted.

Static field format:

Hex AF positive string. . .less than hex 40 or hex AE negative string. . .hex AE

Floating String Specification Field Character:

Hex B1 This control character indicates the start of a floating string specification field. The first character of the field is used as the fill character; following the fill character are two strings delimited by the end-of-string control character. If the algebraic sign of the source field is positive, the first string is to be overlaid into the result field; if the sign is negative, the second string is to be overlaid.

The string selected to be overlaid into the result field, called a floating string, appears immediately to the left of the first significant result character. If significance is never set, neither string is placed in the result field.

Conditional source digit positions (hex B2 control characters) must be provided in the edit mask immediately following the hex B1 field to accommodate the longer of the two floating strings; otherwise, a length conformance exception is signaled. For each of these B2 strings, the fill character is inserted into the result field, and source digits are not consumed. This ensures that the floating string never overlays bytes preceding the receiver operand.

Floating string specification field format:

Hex B1 fill character positive string. . . end-of-string character negative
string. . .end-of-string character

Hex B2. . .

Conditional String Character:

Hex B0 This control character indicates the start of a conditional string, which consists of any characters delimited by the end-of-string control character. Depending on the state of the significance indicator, this string or fill characters replacing it is inserted into the result field. If the significance indicator is off, a fill character for every character in the conditional string is placed in the result field. If the indicator is on, the characters in the conditional string are placed in the result field.

Conditional string format:

Hex B0 conditional string. . .end-of-string character

Unconditional String Character:

Hex B3 This control character turns on the significance indicator and indicates the start of an unconditional string that consists of any characters delimited by the end-of-string control character. This string is unconditionally inserted into the result field regardless of the state of the significance indicator. If the indicator is off when a B3 control character is encountered, the appropriate floating string is overlaid into the result field before (to the left of) the B3 unconditional string (or to the left of where the unconditional string would have been if it were not null).

Unconditional string format:

Hex B3 unconditional string. . .end-of-string character

Control Characters That Correspond to Digits in the Source Field:

Hex B2 This control character specifies that either the corresponding source field digit or the floating string (hex B1) fill character is inserted into the result field, depending on the state of the significance indicator. If the significance indicator is off, the fill character is placed in the result field; if the indicator is on, the source digit is placed. When a source digit is moved to the result field, the zone supplied is hex F. When significance (that is, a nonzero source digit) is detected, the floating string is overlaid to the left of the first significant character.

Control characters hex AA, hex AB, hex AC, and hex AD turn on the significance indicator. If the indicator is off when one of these control characters is encountered, the appropriate floating string is overlaid into the result field before (to the left of) the result digit.

Hex AA This control character specifies that the corresponding source field digit is unconditionally placed in the 4 low-order bits of the result field with the zone set to a hex F.

Hex AB This control character specifies that the corresponding source field digit is unconditionally placed in the result field. If the sign of the source field is positive, the zoned portion of the digit is set to hex F (the preferred positive sign); if the sign is negative, the zone portion is set to hex D (the preferred negative sign).

Hex AC This control character specifies that the corresponding source field digit is unconditionally placed in the result field. If the algebraic sign of the source field is positive, the zone portion of the result is set to hex F (the preferred positive sign); otherwise, the source sign field is moved to the result zone field.

Hex AD This control character specifies that the corresponding source field digit is unconditionally placed in the result field. If the algebraic sign of the source field is negative, the zone is set to hex D (the preferred negative sign); otherwise, the source field sign is moved to the zone position of the result byte.

The following table provides an overview of the results obtained with the valid edit conditions and sequences.

Table 1-1 (Page 1 of 3). Valid Edit Conditions and Results

Mask Character	Previous Significance Indicator	Source Digit	Source Sign	Result Character(s)	Resulting Significance Indicator
AF	Off/On	Any	Positive	Positive string inserted	No Change
	Off/On	Any	Negative	Negative string inserted	No Change
AA	Off	0-9	Positive	Positive floating string overlaid; hex F, source digit	On
	Off	0-9	Negative	Negative floating string overlaid; hex F, source digit	On
	On	0-9	Any	Hex F, source digit	On

Table 1-1 (Page 2 of 3). Valid Edit Conditions and Results

Mask Character	Previous Significance Indicator	Source Digit	Source Sign	Result Character(s)	Resulting Significance Indicator
AB	Off	0-9	Positive	Positive floating string overlaid; hex F, source digit	On
	Off	0-9	Negative	Negative floating string overlaid; hex D, source digit	On
	On	0-9	Positive	Hex F, source digit	On
	On	0-9	Negative	Hex D, source digit	On
AC	Off	0-9	Positive	Positive floating string overlaid; hex F, source digit	
	Off	0-9	Negative	Negative floating string overlaid; source sign and digit	On
	On	0-9	Positive	Hex F, source digit	On
	On	0-9	Negative	Source sign and digit	On
AD	Off	0-9	Positive	Positive floating string overlaid; source sign and digit	On
	Off	0-9	Negative	Negative floating string overlaid; hex D, source digit	On
	On	0-9	Positive	Source sign and digit	On
	On	0-9	Negative	Hex D, source digit	On
B0	Off	Any	Any	Insert fill character for each B0 string character	Off
	On	Any	Any	Insert B0 character string	On
B1 (including necessary B2s)	Off	Any	Any	Insert the fill character for each B2 character that corresponds to a character in the longer of the two floating strings	No Change
B2 (not for a B1 field)	Off	0	Any	Insert fill character	Off
	Off	1-9	Positive	Overlay positive floating string and insert hex F, source digit	On
	Off	1-9	Negative	Overlay negative floating string and insert hex F, source digit	On

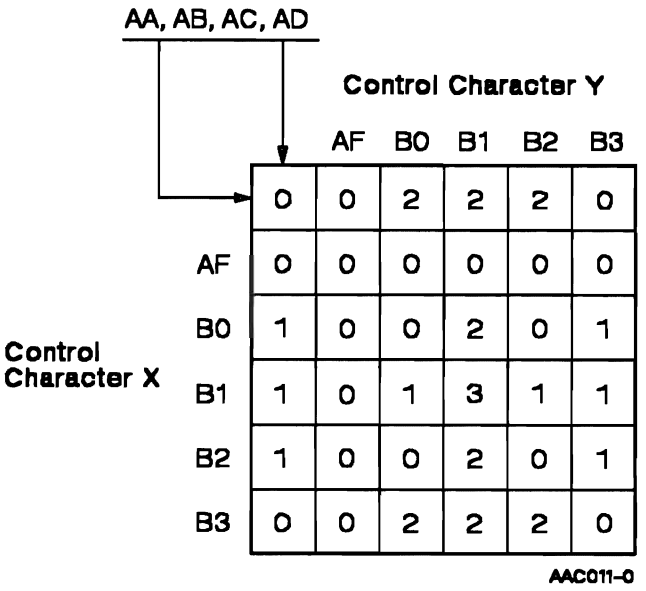
Table 1-1 (Page 3 of 3). Valid Edit Conditions and Results

Mask Character	Previous Significance Indicator	Source Digit	Source Sign	Result Character(s)	Resulting Significance Indicator
	On	0-9	Any	Hex F, source digit	
B3	Off	Any	Positive	Overlay positive floating string and insert B3 character string	On
	Off	Any	Negative	Overlay negative floating string and insert B3 character string	On
	On	Any	Any	Insert B3 character string	On

Note:

1. Any character is a valid fill character, including the end-of-string character.
2. Hex AF, hex B1, hex B0, and hex B3 strings must be terminated by the end-of-string character even if they are null strings.
3. If a hex B1 field has not been encountered (specified) when the significance indicator is turned on, the floating string is considered to be a null string and is therefore not used to overlay into the result field.
4. If the positive and negative strings of a static field are of unequal length, additional static fields are necessary to ensure that the sum of the lengths of the positive strings equal the sum of the lengths of the negative strings; otherwise, a length conformance exception is signaled because the receiver length does not correspond to the length implied by the edit mask and source field sign.

The following figure indicates the valid ordering of control characters in an edit mask field.



Explanation:

Condition Definition

0 Both X and Y can appear in the edit mask field in either order.
1 Y cannot precede X.
2 X cannot precede Y.
3 Both control characters (two B1's) cannot appear in an edit mask field.

Violation of any of the above rules will result in an edit mask syntax exception.

Figure 1-6. Edit Mask Field Control Characters

The following steps are performed when the editing is done:

- Convert Source Value to Packed Decimal
  - The numeric value in the source operand is converted to a packed decimal intermediate value before the editing is done. If the source operand is binary, then the attributes of the intermediate packed field before the edit are calculated as follows:

    Binary(2) = packed (5,0) or
    Binary(4) = packed (10,0)

- Edit
  - The editing of the source digits and mask insertion characters into the receiver operand is done from left to right.
- Insert Floating String into Receiver Field
  - If a floating string is to be inserted into the receiver field, this is done after the other editing.

Edit Digit Count Exception: An edit digit count exception is signaled when:

- The end of the source field is reached and there are more control characters that correspond to digits in the edit mask field.

- The end of the edit mask field is reached and there are more digit positions in the source field.

Edit Mask Syntax Exception: An edit mask syntax exception is signaled when an invalid edit mask control character is encountered or when a sequence rule is violated.

Length Conformance Exception: A length conformance exception is signaled when:

- The end of the edit mask field is reached and there are more character positions in the result field.
- The end of the result field is reached and more positions remain in the edit mask field.
- The number of B2s following a B1 field cannot accommodate the longer of the two floating strings.

Limitations: The following are limits that apply to the functions performed by this instruction.

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for this instruction.

Exceptions

Exception		Operands 1	2	3	Other
06	Addressing				
	01 Spacing addressing violation	X	X	X	
	02 Boundary alignment	X	X	X	
	03 Range	X	X	X	
	04 External data object not found	X	X	X	
	06 Optimized addressability invalid	X	X	X	
08	Argument/parameter				
	01 Parameter reference violation	X	X	X	
0C	Computation				
	02 Decimal data		X		
	04 Edit digit count		X		
	05 Edit mask syntax			X	
	08 Length conformance	X			
10	Damage encountered				
	04 System object damage state	X	X	X	X
	44 Partial system object damage	X	X	X	X
1C	Machine-dependent exception				
	03 Machine storage limit exceeded				X
20	Machine support				
	02 Machine check				X

Exception		Operands 1	2	3	Other
	03 Function check				X
22	Object access				
	01 Object not found	X	X	X	
	02 Object destroyed	X	X	X	
	03 Object suspended	X	X	X	
24	Pointer specification				
	01 Pointer does not exist	X	X	X	
	02 Pointer type invalid	X	X	X	
2A	Program creation				
	06 Invalid operand type	X	X	X	
	07 Invalid operand attribute	X	X	X	
	08 Invalid operand value range	X	X	X	
	0A Invalid operand length	X		X	
	0C Invalid operand odt reference	X	X	X	
	0D Reserved bits are not zero	X	X	X	X
2E	Resource control limit				
	01 user profile storage limit exceeded				X
32	Scalar specification				
	01 Scalar type invalid	X	X	X	
	02 Scalar attributes invalid			X	
36	Space management				
	01 space extension/truncation				X

1.51 Exchange Bytes (EXCHBY)

Op Code (Hex)	Operand 1	Operand 2
10CE	Source 1	Source 2

Operand 1: Character variable scalar (fixed-length) or numeric variable scalar.

Operand 2: Character variable scalar (fixed-length) or numeric variable scalar.

Description: The logical character string values of the two source operands are exchanged. The value of the second source operand is placed in the first source operand and the value of the first source operand is placed in the second operand.

The operands can be either character or numeric. Any numeric operands are interpreted as logical character strings. Both operands must have the same length.

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for this instruction.

Exceptions

Exception		Operands 1	Operands 2	Other
06	Addressing			
	01 Spacing addressing violation	X	X	
	02 Boundary alignment	X	X	
	03 Range	X	X	
	06 Optimized addressability invalid	X	X	
08	Argument/parameter			
	01 Parameter reference violation	X	X	
10	Damage encountered			
	04 System object damage state	X	X	X
	44 Partial system object damage	X	X	X
1C	Machine-dependent exception			
	03 Machine storage limit exceeded			X
20	Machine support			
	02 Machine check			X
	03 Function check			X
22	Object access			
	01 Object not found	X	X	
	02 Object destroyed	X	X	
	03 Object suspended	X	X	

Exception		Operands 1	2	Other
24	Pointer specification			
	01 Pointer does not exist	X	X	
	02 Pointer type invalid	X	X	
2A	Program creation			
	06 Invalid operand type	X	X	
	07 Invalid operand attribute	X	X	
	08 Invalid operand value range	X	X	
	0A Invalid operand length	X	X	
	0C Invalid operand odt reference	X	X	
	0D Reserved bits are not zero	X	X	X
2E	Resource control limit			
	01 user profile storage limit exceeded			X
36	Space management			
	01 space extension/truncation			X

1.52 Exclusive Or (XOR)

Op Code (Hex)	Operand 1	Operand 2	Operand 3
109B	Receiver	Source 1	Source 2

Operand 1: Character variable scalar or numeric variable scalar.

Operand 2: Character scalar or numeric scalar.

Operand 3: Character scalar or numeric scalar.

Optional Forms

Mnemonic	Op Code (Hex)	Form Type
XORS	119B	Short
XORI	189B	Indicator
XORIS	199B	Indicator, Short
XORB	1C9B	Branch
XORBS	1D9B	Branch, Short

If the short instruction option is indicated in the op code, operand 1 is used as the first and second operational operands (receiver and first source operand). Operand 2 is used as the third operational operand (second source operand).

Extender: Branch or indicator options.

If the branch or indicator option is specified in the op code, the extender field must be present along with one or two branch targets (for branch options) or one or two indicator operands (for indicator options). The branch or indicator operands immediately follow the last operand listed above. See Chapter 1. "Introduction" for the encoding of the extender field and the allowed syntax of the branch and indicator operands.

Description: The Boolean EXCLUSIVE OR operation is performed on the string values in the source operands. The resulting string is placed in the receiver operand.

The operands may be character or numeric scalars. They are both interpreted as bit strings. Substringing is supported for both character and numeric operands.

The length of the operation is equal to the length of the longer of the two source operands. The shorter of the two operands is padded on the right. The operation begins with the two source operands left-adjusted and continues bit by bit until they are completed.

The bit values of the result are determined as follows:

Source 1 Bit	Source 2 Bit	Result Bit
1	1	0
0	0	0

Source 1 Bit	Source 2 Bit	Result Bit
1	0	1
0	1	1

The result value is then placed (left-adjusted) in the receiver operand with truncating or padding taking place on the right.

The pad value used in this instruction is a hex 00.

Substring operand references that allow for a null substring reference (a length value of zero) may be specified for operands 1, 2, and 3. The effect of specifying a null substring reference for one source operand is that the other source operand is EXCLUSIVE ORed with an equal length string of all hex 00s. When a null substring reference is specified for both source operands, the result is all zero and the instruction's resultant condition is zero. When a null substring reference is specified for the receiver, a result is not set and the instruction's resultant condition is zero regardless of the values of the source operands.

If operands overlap but do not share all of the same bytes, results of operations performed on these operands are not predictable. If overlapped operands share all of the same bytes, the results are predictable when direct addressing is used. If indirect addressing is used (that is, based operands, parameters, strings with variable lengths, and arrays with variable subscripts), the results are not always predictable.

When the receiver operand is a numeric variable scalar, it is possible that the result produced will not be a valid value for the numeric type. This can occur due to padding with hex 00, due to truncation, or due to the resultant bit string produced by the instruction. The instruction completes normally and signals no exceptions for these conditions.

Resultant Conditions: Zero-The bit value for the bits of the scalar receiver operand is either all zero or a null substring reference is specified for the receiver. Not zero-The bit value for the bits of the scalar receiver operand is not all zero.

Exceptions

Exception		Operands 1	2	3	Other
06	Addressing				
	01 space addressing violation	X	X	X	
	02 boundary alignment	X	X	X	
	03 range	X	X	X	
	06 optimized addressability invalid	X	X	X	
08	Argument/parameter				
	01 parameter reference violation	X	X	X	
10	Damage encountered				
	04 system object damage state	X	X	X	X
	44 partial system object damage	X	X	X	X

Exception		Operands 1	2	3	Other
1C	Machine-dependent exception				
	03 machine storage limit exceeded				X
20	Machine support				
	02 machine check				X
	03 function check				X
22	Object access				
	01 object not found	X	X	X	
	02 object destroyed	X	X	X	
	03 object suspended	X	X	X	
24	Pointer specification				
	01 pointer does not exist	X	X	X	
	02 pointer type invalid	X	X	X	
2A	Program creation				
	05 invalid op code extender field				X
	06 invalid operand type	X	X	X	
	07 invalid operand attribute	X	X	X	
	08 invalid operand value range	X	X	X	
	09 invalid branch target operand				X
	0A invalid operand length	X	X	X	
	0C invalid operand odt reference	X	X	X	
	0D reserved bits are not zero	X	X	X	X
2C	Program execution				
	04 invalid branch target				X
2E	Resource control limit				
	01 user profile storage limit exceeded				X
36	Space management				
	01 space extension/truncation				X

1.53 Extended Character Scan (ECSCAN)

Op Code (Hex)	Operand 1	Operand 2	Operand 3	Operand 4
10D4	Receiver	Base	Compare operand	Mode operand

Operand 1: Binary variable scalar or binary array.

Operand 2: Character variable scalar.

Operand 3: Character scalar.

Operand 4: Character(1) scalar.

Optional Forms

Mnemonic	Op Code (Hex)	Form Type
ESCANI	18D4	Indicator
ESCANB	1CD4	Branch

Extender: Branch or indicator options.

Either the branch option or indicator option is required by the instruction. The extender field is required along with from one to three branch targets (for branch option) or one to three indicator operands (for indicator option). The branch or indicator operands are required for operand 3 and optional for operands 4 and 5. See Chapter 1. "Introduction" for the bit encoding of the extender field and the allowed syntax of the branch and indicator operands.

Description: This instruction scans the string value of the base operand for occurrences of the string value of the compare operand and indicates the relative locations of these occurrences in the receiver operand. The character string value of the base operand is scanned for occurrences of the character string value of the compare operand under control of the mode operand and mode control characters embedded in the base string.

The base and compare operands must both be character strings. The length of the compare operand must not be greater than that of the base string. The base and compare operand are interpreted as containing a mixture of 1-byte (simple) and 2-byte (extended) character codes. The mode, simple or extended, with which the string is to be interpreted, is controlled initially by the mode operand and thereafter by mode control characters embedded in the strings. The mode control characters are as follows:

- Hex 0E = Shift out of simple character mode to extended mode.
- Hex 0F = Shift into simple character mode from extended mode. This is recognized only if it occurs in the first byte position of an extended character code.

The format of the mode operand is as follows:

- Mode operand — Char(1)
  - Operand 2 initial mode indicator — Bit 0

0 = Operand starts in simple character mode.
1 = Operand starts in extended character mode.

- Operand 3 initial mode indicator — Bit 1

  0 = Operand starts in simple character mode.
  1 = Operand starts in extended character mode.

- Reserved (binary 0) — Bits 2-7

The operation begins at the left end of the base string and continues character by character, left to right. When the base string is interpreted in simple character mode, the operation moves through the base string 1 byte at a time. When the base string is interpreted in extended character mode, the operation moves through the base string 2 bytes at a time.

The compare operand value is the entire byte string specified for the compare operand. The mode operand determines the initial mode of the compare operand. The first character of the compare operand value is assumed to be a valid character for the initial mode of the compare operand and not a mode control character. Mode control characters in the compare operand value participate in comparisons performed during the scan function except that a mode control character as the first character of the compare operand causes unpredictable results.

The base string is scanned until the mode of the characters being processed is the same as the initial mode of the compare operand value. The operation continues comparing the characters of the base string with those of the compare operand value. The starting character of the characters being compared in the base string is always a valid character for the initial mode of the compare operand value. A mode control character encountered in the base string that changed the base string mode to match the initial mode of the compare operand value does not participate in the comparison. The length of the comparison is equal to the length of the compare operand value and the comparison is performed the same as performed by the Compare Bytes Left Adjusted instruction.

If a set of bytes that matches the compare operand value is found, the binary value for the relative location of the leftmost base string character of the set of bytes is placed in the receiver operand.

If the receiver operand is a scalar, only the first occurrence of the compare operand is noted. If the receiver operand is an array, as many occurrences as there are elements in the array are noted.

If a mode change is encountered in the base string, the base string is again scanned until the mode of the characters being processed is the same as the initial mode of the compare operand value, and then the comparisons are resumed.

The operation continues until no more occurrences of the compare operand value can be noted in the receiver operand or until the number of bytes remaining to be scanned in the base string is less than the length of the compare operand value. When the second condition occurs, the receiver value is set to zero. If the receiver operand is an array, all its remaining elements are also set to zero.

If the escape code encountered result condition is specified (through a branch or indicator option), verifications are performed on the base string as it is scanned. Each byte of the base string is checked for a value less than hex 40. When a value less than hex 40 is encountered, it is then determined if it is a valid mode control character.

If a byte value of less than hex 40 is not a valid mode control character, it is considered to be an escape code. The binary value for the relative location of the character (simple or extended) being interrogated is placed in the receiver operand, and the appropriate action (indicator or branch) is performed according to the specification of the escape code encountered result condition. If the receiver operand is an array, the next array element after any elements set with locations or prior occurrences of the compare operand, is set with the location of the character containing the escape code and all the remaining array elements are set to zero.

If the escape encountered result condition is not specified, verifications of the character codes in the base string are not performed.

Resultant Conditions: Positive or zero-The numeric value(s) of the receiver operand is either positive or zero. In the case where the receiver operand is an array, the resultant condition is zero if all elements are zero. Escape code encountered-An escape character code value was encountered during the scanning of the base string. Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for this instruction.

Exceptions

Exception		Operands 1	2	3	4	Other
06	Addressing					
	01 space addressing violation	X	X	X	X	
	02 boundary alignment violation	X	X	X	X	
	03 range	X	X	X	X	
	06 optimized addressability invalid	X	X	X	X	
08	Argument/parameter					
	01 parameter reference violation	X	X	X	X	
0C	Computation					
	08 length conformance		X	X	X	
10	Damage encountered					
	04 System object damage state					X
	44 partial system object damage					X
1C	Machine-dependent exception					
	03 machine storage limit exceeded					X
20	Machine support					
	02 machine check					X
	03 function check					X

Exception		Operands 1	2	3	4	Other
22	Object access					
	01 object not found	X	X	X	X	
	02 object destroyed	X	X	X	X	
	03 object suspended	X	X	X	X	
24	Pointer specification					
	01 pointer does not exist	X	X	X	X	
	02 pointer type invalid	X	X	X	X	
2A	Program creation					
	05 invalid op code extender field					X
	06 invalid operand type	X	X	X	X	
	07 invalid operand attribute	X	X	X	X	
	08 invalid operand value range	X	X	X	X	
	09 invalid branch target operand					X
	0A invalid operand length		X	X		
	0C invalid operand odt reference	X	X	X	X	
	0D reserved bits are not zero	X	X	X	X	X
2C	Program execution					
	04 invalid branch target					X
2E	Resource control limit					
	01 user profile storage limit exceeded					X
32	Scalar specification					
	01 scalar type invalid	X	X	X	X	
	03 scalar value invalid				X	
36	Space management					
	01 space extension/truncation					X

1.54 Extract Exponent (EXTREXP)

Op Code (Hex)	Operand 1	Operand 2
1072	Receiver	Source

Operand 1: Binary variable scalar.

Operand 2: Floating-point scalar.

Optional Forms

Mnemonic	Op Code (Hex)	Form Type
EXTREXPI	1872	Indicator
EXTREXPB	1C72	Branch

Extender: Branch options or indicator options.

If the branch or indicator option is specified in the op code, the extender field must be present along with one to four branch targets (for branch options) or one to four indicator operands (for indicator options). The branch or indicator operations immediately follow the last operand listed above. See Chapter 1. "Introduction" for the encoding of the extender field and the allowed syntax of the branch and indicator operands.

Description: This instruction extracts the exponent portion of a floating-point scalar source operand and places it into the receiver operand as a binary variable scalar.

The operands must be the numeric types indicated because no conversions are performed.

The source floating-point field is interrogated to determine the binary floating-point value represented and either a signed exponent, for number values, or a special identifier, for infinity and NaN values, is placed in the binary variable scalar receiver operand.

The value to be assigned to the receiver, is dependent upon the floating-point value represented in the source operand as described below. It is a signed binary integer value and a numeric assignment of the value is made to the receiver.

When the source represents a normalized number, the biased exponent contained in the exponent field of the source is converted to the corresponding signed exponent by subtracting the bias of 127 for short or 1023 for long to determine the value to be returned. The resulting value ranges from -126 to +127 for short format, -1022 to +1023 for long format. When the receiver is unsigned binary a negative exponant will result in a size exception unless size exceptions are suppressed by using the suppress binary size exception program attribute on the CRTPG instruction.

When the source represents a denormalized number, the value to be returned is determined by adjusting the signed exponent of the denormalized number. The signed exponent of a denormalized number is a fixed value of -126 for the short

format and -1022 for the long format. It is adjusted to the value the signed exponent would be if the source value was adjusted to a normalized number. The resulting value ranges from -127 to -149 for short format, -1023 to -1074 for long format.

When the source represents a value of zero, the value returned is zero.

When the source represents infinity, the value returned is +32767.

When the source represents a not-a-number, the value returned is -32768 for a signed binary receiver. For an unsigned binary(2) a value of 32768 is returned, and for a unsigned binary(4) a value of 4294934528 is returned.

Resultant Conditions: Normalized-The source operand value represents a normalized binary floating-point number. The signed exponent is stored in the receiver. Denormalized-The source operand value represents a denormalized binary floating-point number. An adjusted signed exponent is stored in the receiver. Infinity-The source operand value represents infinity. The receiver is set with a value of +32767. NaN-The source operand value represents a not-a-number. The receiver is set with a value of -32768 when signed binary, with a value of 32768 when unsigned binary(2), and with a value of 4294934528 when unsigned binary(4).

Exceptions

Exception		Operands 1	Operands 2	Other
06	Addressing			
	01 space addressing violation	X	X	
	02 boundary alignment violation	X	X	
	03 range	X	X	
	06 optimized addressability invalid	X	X	
08	Argument/parameter			
	01 parameter reference violation	X	X	
10	Damage encountered			
	04 System object damage state			X
	44 partial system object damage			X
1C	Machine-dependent exception			
	03 machine storage limit exceeded			X
20	Machine support			
	02 machine check			X
	03 function check			X
22	Object access			
	01 object not found	X	X	
	02 object destroyed	X	X	
	03 object suspended	X	X	
24	Pointer specification			

Exception		Operands 1	2	Other
	01 pointer does not exist	X	X	
	02 pointer type invalid	X	X	
2A	Program creation			
	05 invalid op code extender field			X
	06 invalid operand type	X	X	
	07 invalid operand attribute	X	X	
	08 invalid operand value range	X	X	
	09 invalid branch target operand			X
	0C invalid operand odt reference	X	X	X
	0D reserved bits are not zero	X	X	X
2C	Program execution			
	04 invalid branch target			X
2E	Resource control limit			
	01 user profile storage limit exceeded			X
32	Scalar specification			
	01 scalar type invalid	X	X	
36	Space management			
	01 space extension/truncation			X

1.55 Extract Magnitude (EXTRMAG)

Op Code (Hex)	Operand 1	Operand 2
1052	Receiver	Source

Operand 1: Numeric variable scalar.

Operand 2: Numeric scalar.

Optional Forms

Mnemonic	Op Code (Hex)	Form Type
EXTRMAGS	1152	Short
EXTRMAGI	1852	Indicator
EXTRMAGIS	1952	Indicator, Short
EXTRMAGB	1C52	Branch
EXTRMAGBS	1D52	Branch, Short

If the short instruction option is indicated in the op code, operand 1 is used as the first and second operational operands (receiver and first source operand). Operand 2 is used as the third operational operand (second source operand).

Extender: Branch or indicator options.

If the branch or indicator option is specified in the op code, the extender field must be present along with one to four branch targets (for branch options) or one to four indicator operands (for indicator options). The branch or indicator operands immediately follow the last operand listed above. See Chapter 1. "Introduction" for the encoding of the extender field and the allowed syntax of the branch and indicator operands.

Description: The numeric value of the source operand is converted to its absolute value and placed in the numeric variable scalar receiver operand.

The absolute value is formed from the source operand as follows:

- Signed binary
  - Extract the numeric value and form twos complement if the source operand is negative.
- Unsigned signed binary
  - Extract the numeric value.
- Packed/Zoned
  - Extract the numeric value and force the source operand's sign to positive.
- Floating-point
  - Extract the numeric value and force the significand sign to positive.

The result of the operation is copied into the receiver operand according to the rules of the Copy Numeric Value instruction. If this operand is not the same type

as that used in performing the operation, the resultant value is converted to its type. If necessary, the resultant value is adjusted to the length of the receiver operand, or aligned at the assumed decimal point of the receiver operand, or both before being copied to it. If significant digits are truncated on the left end of the resultant value, a size exception is signaled. An attempt to extract the magnitude of a maximum negative binary value to a binary scalar of the same size also results in a size exception.

When the source floating-point operand represents not-a-number, the sign field of the source is not forced to positive and this value is not altered in the receiver.

If a decimal to binary conversion causes a size exception to be signaled, the binary value contains the correct truncated result only if the decimal value contains 15 or fewer significant nonfractional digits.

For a fixed-point operation, if significant digits are truncated from the left end of the resultant value, a size exception is signaled. An attempt to extract the absolute value of a maximum negative binary value into a binary scalar of the same size also results in a size exception.

For floating-point operations that involve a fixed-point receiver field, if nonzero digits would be truncated from the left end of the resultant value, an invalid floating-point conversion exception is signaled.

For a floating-point receiver operand, if the exponent of the resultant value is either too large or too small to be represented in the receiver field, the floating-point overflow or the floating-point underflow exception is signaled.

Resultant Conditions: Positive or zero-The algebraic value of the receiver operand is either positive or zero. Unordered-The value assigned a floating-point receiver operand is NaN.

Exceptions

Exception		Operands 1	Operands 2	Other
06	Addressing			
	01 space addressing violation	X	X	
	02 boundary alignment	X	X	
	03 range	X	X	
	06 optimized addressability invalid	X	X	
08	Argument/parameter			
	01 parameter reference violation	X	X	
0C	Computation			
	02 decimal data		X	
	06 floating-point overflow	X		
	07 floating-point underflow	X		
	09 floating-point invalid operand		X	
	0A size	X		

Exception		Operands 1	2	Other
	0D floating-point inexact result	X		
10	Damage encountered			
	04 system object damage	X	X	X
	44 partial system object damage	X	X	X
1C	Machine-dependent exception			
	03 machine storage limit exceeded			X
20	Machine support			
	02 machine check			X
	03 function check			X
22	Object access			
	01 object not found	X	X	
	02 object destroyed	X	X	
	03 object suspended	X	X	
24	Pointer specification			
	01 pointer does not exist	X	X	
	02 pointer type invalid	X	X	
2A	Program creation			
	05 invalid op code extender field			X
	06 invalid operand type	X	X	
	07 invalid operand attribute	X	X	
	08 invalid operand value range	X	X	
	09 invalid branch target operand			X
	0C invalid operand odt reference	X	X	
	0D reserved bits are not zero	X	X	X
2C	Program execution			
	04 invalid branch target			X
2E	Resource control limit			
	01 user profile storage limit exceeded			X
36	Space management			
	01 space extension/truncation			X

1.56 Multiply (MULT)

Op Code (Hex)	Operand 1	Operand 2	Operand 3
104B	Product	Multiplicand	Multiplier

Operand 1: Numeric variable scalar.

Operand 2: Numeric scalar.

Operand 3: Numeric scalar.

Optional Forms

Mnemonic	Op Code (Hex)	Form Type
MULTS	114B	Short
MULTR	124B	Round
MULTSR	134B	Short, Round
MULTI	184B	Indicator
MULTIS	194B	Indicator, Short
MULTIR	1A4B	Indicator, Round
MULTISR	1B4B	Indicator, Short, Round
MULTB	1C4B	Branch
MULTBS	1D4B	Branch, Short
MULTBR	1E4B	Branch, Round
MULTBSR	1F4B	Branch, Short, Round

If the short instruction option is indicated in the op code, operand 1 is used as the first and second operational operands (receiver and first source operand). Operand 2 is used as the third operational operand (second source operand).

Extender: Branch or indicator options.

If the branch or indicator option is specified in the op code, the extender field must be present along with one to four branch targets (for branch options) or one to four indicator operands (for indicator options). The branch or indicator operands immediately follow the last operand listed above. See Chapter 1. "Introduction" for the encoding of the extender field and the allowed syntax of the branch and indicator operands.

Description: The Product is the result of multiplying the Multiplicand and the Multiplier.

Operands can have floating-point, packed or zoned decimal, signed or unsigned binary type.

Source operands are the Multiplicand and Multiplier. The receiver operand is the Product.

If operands are not of the same type, source operands are converted according to the following rules:

1. If any one of the operands has floating point type, source operands are converted to floating point type.
2. Otherwise, if any one of the operands has zoned or packed decimal type, source operands are converted to packed decimal.
3. Otherwise, the binary operands are converted to a like type. Note: unsigned binary(2) scalars are logically treated as signed binary(4) scalars.

Source operands are multiplied according to their type. Floating point operands are multiplied using floating point multiplication. Packed decimal operands are multiplied using packed decimal multiplication. Unsigned binary multiplication is used with unsigned source operands. Signed binary operands are multiplied using two's complement binary multiplication.

Better performance can be obtained if all operands have the same type. Signed and unsigned binary multiplication execute faster than either packed decimal or floating point multiplication.

Decimal operands used in floating-point operations cannot contain more than 15 total digit positions.

If the multiplicand operand or the multiplier operand has a value of 0, the result of the multiplication is a zero product.

For a decimal operation, no alignment of the assumed decimal point is performed for the multiplier and multiplicand operands.

The operation occurs using the specified lengths of the multiplicand and multiplier operands with no logical zero padding on the left necessary.

Floating-point multiplication uses exponent addition and significand multiplication.

For nonfloating-point computations and for significand multiplication for floating-point operations, the multiplication operation is performed according to the rules of algebra. Unsigned binary operands are treated as positive numbers for the algebra.

The result of the operation is copied into the product operand. If this operand is not the same type as that used in performing the operation, the resultant value is converted to its type. If necessary, the resultant value is adjusted to the length of the product operand, aligned at the assumed decimal point of the product operand, or both before being copied to it.

For the optional round form of the instruction, specification of a floating-point receiver operand is invalid.

For fixed-point operations in programs that request to be notified of size exceptions, if nonzero digits are truncated from the left end of the resultant value, a size exception is signaled.

For floating-point operations involving a fixed-point receiver field (if nonzero digits would be truncated from the left end of the resultant value), an invalid floating-point conversion exception is signaled.

For a floating-point product operand, if the exponent of the resultant value is either too large or too small to be represented in the product field, the floating-point overflow or the floating-point underflow exception is signaled.

If operands overlap but do not share all of the same bytes, results of operations performed on these operands are not predictable. If overlapped operands share all of the same bytes, the results are predictable when direct addressing is used. If indirect addressing is used (that is, based operands, parameters, strings with variable lengths, and arrays with variable subscripts), the results are not always predictable.

If a decimal to binary conversion causes a size exception to be signaled, the binary value contains the correct truncated result only if the decimal value contains 15 or fewer significant nonfractional digits.

Resultant Conditions: Positive, negative, or zero-The algebraic value of the numeric scalar product is positive, negative, or zero. Unordered-The value assigned a floating-point product operand is NaN.

Exceptions

Exception		Operands 1	2	3 [4, 5]	Other
06	Addressing				
	01 space addressing violation	X	X	X	
	02 boundary alignment	X	X	X	
	03 range	X	X	X	
	06 optimized addressability invalid	X	X	X	
08	Argument/parameter				
	01 parameter reference violation	X	X	X	
0C	Computation				
	02 decimal data		X	X	
	06 floating-point overflow	X			
	07 floating-point underflow	X			
	09 floating-point invalid operand		X	X	X
	0A size	X			
	0C invalid floating-point conversion	X			
	0D floating-point inexact result	X			
10	Damage encountered				
	04 system object damage state	X	X	X	X
	44 partial system object damage	X	X	X	X
1C	Machine-dependent exception				
	03 machine storage limit exceeded				X

Exception		Operands 1	2	3 [4, 5]	Other
20	Machine support				
	02 machine check				X
	03 function check				X
22	Object access				
	01 object not found	X	X	X	
	02 object destroyed	X	X	X	
	03 object suspended	X	X	X	
24	Pointer specification				
	01 pointer does not exist	X	X	X	
	02 pointer type invalid	X	X	X	
2A	Program creation				
	05 invalid op code extender field				X
	06 invalid operand type	X	X	X	
	07 invalid operand attribute	X	X	X	
	08 invalid operand value range	X	X	X	
	09 invalid branch target operand				X
	0C invalid operand odt reference	X	X	X	
	0D reserved bits are not zero	X	X	X	X
2C	Program execution				
	04 invalid branch target				X
2E	Resource control limit				
	01 user profile storage limit exceeded				X
36	Space management				
	01 space extension/truncation				X

1.57 Negate (NEG)

Op Code (Hex)	Operand 1	Operand 2
1056	Receiver	Source

Operand 1: Numeric variable scalar.

Operand 2: Numeric scalar.

Optional Forms

Mnemonic	Op Code (Hex)	Form Type
NEGS	1156	Short
NEGI	1856	Indicator
NEGIS	1956	Indicator, Short
NEGB	1C56	Branch
NEGBS	1D56	Branch, Short

If the short instruction option is indicated in the op code, operand 1 is used as the first and second operational operands (receiver and first source operand). Operand 2 is used as the third operational operand (second source operand).

Extender: Branch or indicator options.

If the branch or indicator option is specified in the op code, the extender field must be present along with one to four branch targets (for branch options) or one to four indicator operands (for indicator options). The branch or indicator operands immediately follow the last operand listed above. See Chapter 1. "Introduction" for the encoding of the extender field and the allowed syntax of the branch and indicator operands.

Description: The numeric value in the source operand is changed as if it had been multiplied by a negative one (-1). The result is placed in the receiver operand.

The sign changing of the source operand value (positive to negative and negative to positive) is performed as follows:

- Binary
  - Extract the numeric value and form the twos complement of it.
- Packed/Zoned
  - Extract the numeric value and force its sign to positive if it is negative or to negative if it is positive.
- Floating-point
  - Extract the numeric value and force the significand sign to positive if it is negative or to negative if it is positive.

The result of the operation is copied into the receiver operand. If this operand is not the same type as that used in performing the operation, the resultant value is converted to its type. If necessary, the resultant value is adjusted to the

length of the receiver operand, aligned at the assumed decimal point of the receiver operand, or both before being copied to it. If significant digits are truncated on the left end of the resultant value, a size exception is signaled. An attempt to negate a maximum negative signed binary value to a signed binary scalar of the same size also results in a size exception. When the receiver is binary the size exception may be suppressed by using the suppress binary size exception attribute on the CRTPG instruction. If a packed or zoned 0 is negated, the result is always positive 0.

When the source floating-point operand represents not-a-number, the sign field of the source is not forced to positive and this value is not altered in the receiver.

For a fixed-point operation, if significant digits are truncated from the left end of the resultant value, a size exception is signaled. An attempt to negate a maximum negative binary value into a binary scalar of the same size also results in a size exception.

For floating-point operations that involve a fixed-point receiver field, if nonzero digits would be truncated from the left end of the resultant value, an invalid floating-point conversion exception is signaled.

For a floating-point receiver operand, if the exponent of the resultant value is either too large or too small to be represented in the receiver field, the floating-point overflow and the floating-point underflow exceptions are signaled.

If a decimal to binary conversion causes a size exception to be signaled or if the size exception was suppressed, the binary value contains the correct truncated result only if the decimal value contains 15 or fewer significant nonfractional digits.

Resultant Conditions: Positive, negative, or zero-The algebraic value of the receiver operand is either positive, negative, or zero. Unordered-The value assigned a floating-point receiver operand is NaN.

Exceptions

Exception		Operands 1	Operands 2	Other
06	Addressing			
	01 space addressing violation	X	X	
	02 boundary alignment	X	X	
	03 range	X	X	
	06 optimized addressability invalid	X	X	
08	Argument/parameter			
	01 parameter reference violation	X	X	
0C	Computation			
	02 decimal data		X	
	06 floating-point overflow	X		
	07 floating-point underflow	X		
	09 floating-point invalid operand		X	X

Exception		Operands 1	2	Other
	0A size	X		
	0C invalid floating-point conversion	X		
	0D floating-point inexact result	X		
10	Damage encountered			
	04 system object damage	X	X	X
	44 partial system object damage	X	X	X
1C	Machine-dependent exception			
	03 machine storage limit exceeded			X
20	Machine support			
	02 machine check			X
	03 function check			X
22	Object access			
	01 object not found	X	X	
	02 object destroyed	X	X	
	03 object suspended	X	X	
24	Pointer specification			
	01 pointer does not exist	X	X	
	02 pointer type invalid	X	X	
2A	Program creation			
	05 invalid op code extender field			X
	06 invalid operand type	X	X	
	07 invalid operand attribute	X	X	
	08 invalid operand value range	X	X	
	09 invalid branch target operand			X
	0C invalid operand odt reference	X	X	
	0D reserved bits are not zero	X	X	X
2C	Program execution			
	04 invalid branch target			X
2E	Resource control limit			
	01 user profile storage limit exceeded			X
36	Space management			
	01 space extension/truncation			X

1.58 Not (NOT)

Op Code (Hex)	Operand 1	Operand 2
108A	Receiver	Source

Operand 1: Character variable scalar or numeric variable scalar.

Operand 2: Character variable scalar or numeric variable

Optional Forms

Mnemonic	Op Code (Hex)	Form Type
NOTS	118A	Short
NOTI	188A	Indicator
NOTIS	198A	Indicator, Short
NOTB	1C8A	Branch
NOTBS	1D8A	Branch, Short

If the short instruction option is indicated in the op code, operand 1 is used as the first and second operational operands (receiver and first source operand). Operand 2 is used as the third operational operand (second source operand).

Extender: Branch or indicator options.

If the branch or indicator option is specified in the op code, the extender field must be present along with one or two branch targets (for branch options) or one or two indicator operands (for indicator options). The branch or indicator operands immediately follow the last operand listed above. See Chapter 1. "Introduction" for the encoding of the extender field and the allowed syntax of the branch and indicator operands.

Description: The Boolean NOT operation is performed on the string value in the source operand. The resulting string is placed in the receiver operand.

The operands may be character or numeric scalars. They are both interpreted as bit strings. Substringing is supported for both character and numeric operands.

The length of the operation is equal to the length of the source operand.

The bit values of the result are determined as follows:

Source Bit	Result Bit
1	0
0	1

The result value is then placed (left-adjusted) in the receiver operand with truncating or padding taking place on the right. The pad value used in this instruction is a hex 00 byte.

Substring operand references that allow for a null substring reference (a length value of zero) may be specified for operands 1 and 2. The effect of specifying a null substring reference for the source operand is that the result is all zero and the instruction's resultant condition is zero. When a null substring reference is specified for the receiver, a result is not set and the instruction's resultant condition is zero regardless of the value of the source operand.

When the receiver operand is a numeric variable scalar, it is possible that the result produced will not be a valid value for the numeric type. This can occur due to padding with hex 00, due to truncation, or due to the resultant bit string produced by the instruction. The instruction completes normally and signals no exceptions for these conditions.

Resultant Conditions: Zero-The bit value for the bits of the scalar receiver operand is either all zero or a null substring reference is specified for the receiver. Not zero-The bit value for the bits of the scalar receiver operand is not all zero.

Exceptions

Exception		Operands 1	Operands 2	Other
06	Addressing			
	01 space addressing violation	X	X	
	02 boundary alignment	X	X	
	03 range	X	X	
	06 optimized addressability invalid	X	X	
08	Argument/parameter			
	01 parameter reference violation	X	X	
10	Damage encountered			
	04 system object damage state	X	X	X
	44 partial system object damage	X	X	X
1C	Machine-dependent exception			
	03 machine storage limit exceeded			X
20	Machine support			
	02 machine check			X
	03 function check			X
22	Object access			
	01 object not found	X	X	
	02 object destroyed	X	X	
	03 object suspended	X	X	
24	Pointer specification			
	01 pointer does not exist	X	X	
	02 pointer type invalid	X	X	
2A	Program creation			

Exception		Operands 1	2	Other
	05 invalid op code extender field			X
	06 invalid operand type	X	X	
	07 invalid operand attribute	X	X	
	08 invalid operand value range	X	X	
	09 invalid branch target operand			X
	0A invalid operand length	X	X	
	0C invalid operand odt reference	X	X	
	0D reserved bits are not zero	X	X	X
2C	Program execution			
	04 invalid branch target			X
2E	Resource control limit			
	01 user profile storage limit exceeded			X
36	Space management			
	01 space extension/truncation			X

1.59 Or (OR)

Op Code (Hex)	Operand 1	Operand 2	Operand 3
1097	Receiver	Source 1	Source 2

Operand 1: Character variable scalar or numeric variable scalar.

Operand 2: Character scalar or numeric scalar.

Operand 3: Character scalar or numeric scalar.

Optional Forms

Mnemonic	Op Code (Hex)	Form Type
ORS	1197	Short
ORI	1897	Indicator
ORIS	1997	Indicator, Short
ORB	1C97	Branch
ORBS	1D97	Branch, Short

If the short instruction option is indicated in the op code, operand 1 is used as the first and second operational operands (receiver and first source operand). Operand 2 is used as the third operational operand (second source operand).

Extender: Branch or Indicator options.

If the branch or indicator option is specified in the op code, the extender field must be present along with one or two branch targets (for branch options) or one or two indicator operands (for indicator options). The branch or indicator operands immediately follow the last operand listed above. See Chapter 1. "Introduction" for the encoding of the extender field and the allowed syntax of the branch and indicator operands.

Description: The Boolean OR operation is performed on the string values in the source operands. The resulting string is placed in the receiver operand.

The operands may be character or numeric scalars. They are both interpreted as bit strings. Substringing is supported for both character and numeric operands.

The length of the operation is equal to the length of the longer of the two source operands. The shorter of the two operands is logically padded on the right with hex 00. The excess bytes in the longer operand are assigned to the results.

The bit values of the result are determined as follows:

Source 1 Bit	Source 2 Bit	Result Bit
1	1	1
0	1	1
1	0	1

Source 1 Bit	Source 2 Bit	Result Bit
0	0	0

The result value is then placed (left-adjusted) in the receiver operand with truncating or padding taking place on the right. The pad value used in this instruction is a hex 00.

Substring operand references that allow for a null substring reference (a length value of zero) may be specified for operands 1, 2, and 3. The effect of specifying a null substring reference for one source operand is that the other source operand is ORed with an equal length string of all hex 00s. This causes the value of the other operand to be assigned to the result. When a null substring reference is specified for both source operands, the result is all zero and the instruction's resultant condition is zero. When a null substring reference is specified for the receiver, a result is not set and the instruction's resultant condition is zero regardless of the values of the source operands.

If operands overlap but do not share all of the same bytes, results of operations performed on these operands are not predictable. If overlapped operands share all of the same bytes, the results are predictable when direct addressing is used. If indirect addressing is used (that is, based operands, parameters, strings with variable lengths, and arrays with variable subscripts), the results are not always predictable.

When the receiver operand is a numeric variable scalar, it is possible that the result produced will not be a valid value for the numeric type. This can occur due to padding with hex 00, due to truncation, or due to the resultant bit string produced by the instruction. The instruction completes normally and signals no exceptions for these conditions.

Resultant Conditions: Zero-The bit value for the bits of the scalar receiver operand is either all zero or a null substring reference is specified for the receiver. Not zero-The bit value for the bits of the scalar receiver operand is not all zero.

Exceptions

Exception		Operands 1	2	3	Other
06	Addressing				
	01 space addressing violation	X	X	X	
	02 boundary alignment	X	X	X	
	03 range	X	X	X	
	06 optimized addressability invalid	X	X	X	
08	Argument/parameter				
	01 parameter reference violation	X	X	X	
10	Damage encountered				
	04 system object damage state	X	X	X	X
	44 partial system object damage	X	X	X	X
1C	Machine-dependent exception				

Exception		Operands 1	2	3	Other
	03 machine storage limit exceeded				X
20	Machine support				
	02 machine check				X
	03 function check				X
22	Object access				
	01 object not found	X	X	X	
	02 object destroyed	X	X	X	
	03 object suspended	X	X	X	
24	Pointer specification				
	01 pointer does not exist	X	X	X	
	02 pointer type invalid	X	X	X	
2A	Program creation				
	05 invalid op code extender field				X
	06 invalid operand type	X	X	X	
	07 invalid operand attribute	X	X	X	
	08 invalid operand value range	X	X	X	
	09 invalid branch target operand				X
	0A invalid operand length	X	X	X	
	0C invalid operand odt reference	X	X	X	
	0D reserved bits are not zero	X	X	X	X
2C	Program execution				
	04 invalid branch target				X
2E	Resource control limit				
	01 user profile storage limit exceeded				X
36	Space management				
	01 space extension/truncation				X

1.60 Remainder (REM)

Op Code (Hex)	Operand 1	Operand 2	Operand 3
1073	Remainder	Dividend	Divisor

Operand 1: Numeric variable scalar.

Operand 2: Numeric scalar.

Operand 3: Numeric scalar.

Optional Forms

Mnemonic	Op Code (Hex)	Form Type
REMS	1173	Short
REMI	1873	Indicator
REMIS	1973	Indicator, Short
REMB	1C73	Branch
REMBS	1D73	Branch, Short

If the short instruction option is indicated in the op code, operand 1 is used as the first and second operational operands (receiver and first source operand). Operand 2 is used as the third operational operand (second source operand).

Extender: Branch or indicator options.

If the branch or indicator option is specified in the op code, the extender field must be present along with one to three branch targets (for branch options) or one to three indicator operands (for indicator options). The branch or indicator operands immediately follow the last operand listed above. See Chapter 1. "Introduction" for the encoding of the extender field and the allowed syntax of the branch and indicator operands.

Description: The Remainder is the result of dividing the Dividend by the Divisor and placing the remainder in operand 1.

Operands can have packed or zoned decimal, signed or unsigned binary type.

Source operands are the Dividend and Divisor. The receiver operand is the Remainder.

If operands are not of the same type, source operands are converted according to the following rules:

1. If any one of the operands has zoned or packed decimal type, source operands are converted to packed decimal.
2. Otherwise, the binary operands are converted to a like type. Note: unsigned binary(2) scalars are logically treated as signed binary(4) scalars.

Source operands are divided according to their type. Packed decimal operands are divided using packed decimal division. Unsigned binary division is used with

unsigned source operands. Signed binary operands are divided using two's complement binary division.

Better performance can be obtained if all operands have the same type. Signed and unsigned binary division execute faster than packed decimal division.

Floating-point is not supported for this instruction.

If the divisor has a numeric value of 0, a zero divide exception is signaled. If the dividend has a value of 0, the result of the division is a zero value remainder.

For a decimal operation, the internal quotient value produced by the divide operation is always calculated with a precision of zero fractional digit positions. If necessary, internal alignment of the assumed decimal point for the dividend and divisor operands is performed to insure the correct precision for the resultant quotient value. These internal alignments are not subject to detection of the decimal point alignment exception. An internal quotient and the corresponding remainder value will be calculated for any combination of decimal attributes which may be specified for the instruction's operands. However, as described below, the assignment of the remainder value is limited to that portion of the remainder value which fits in the remainder operand.

If the dividend is shorter than the divisor, it is logically adjusted to the length of the divisor.

The division operation is performed according to the rules of algebra. Unsigned binary is treated as a positive number for the algebra. Before the remainder is calculated, an intermediate quotient is calculated. The attributes of this quotient are derived from the attributes of the dividend and divisor operands as follows:

Dividend	**Divisor**	**Intermediate Quotient**
IM,SIM or SBIN(2)	IM,SIM or SBIN(2)	SBIN(2)
IM,SIM or SBIN(2)	SBIN(4)	SBIN(4)
IM,SIM,SBIN(2) or UBIN(2)	DECIMAL(P2,Q2)	DECIMAL(5+Q2,0)
IM,SIM,SBIN(2) or SBIN(4)	UBIN(2) or UBIN(4)	UBIN(4)
UBIN(2) or UBIN(4)	IM,SIM,SBIN(2) or SBIN(4)	UBIN(4)
UBIN(2) or UBIN(4)	UBIN(2) or UBIN(4)	UBIN(4)
SBIN(4)	IM,SIM or SBIN(2)	SBIN(4)
SBIN(4) or UBIN(4)	DECIMAL(P2,Q2)	DECIMAL(10+Q2,0)
DECIMAL(P1,Q1)	IM,SIM,SBIN(2) or UBIN(2)	DECIMAL(P1,0)
DECIMAL(P1,Q1)	SBIN(4) or UBIN(4)	DECIMAL(P1,0)
DECIMAL(P1,Q1)	DECIMAL(P2,Q2)	DECIMAL(P1-Q1+Q,0) Where Q = Larger of Q1 or Q2

IM = IMMEDIATE
SIM = SIGNED IMMEDIATE
SBIN = SIGNED BINARY
UBIN = UNSIGNED BINARY

DECIMAL = PACKED OR ZONED

After the intermediate quotient numeric value has been determined, the numeric value of the remainder operand is calculated as follows:

Remainder = Dividend - (Quotient*Divisor)

When signed arithmetic is used, the sign of the remainder is the same as that of the dividend unless the remainder has a value of 0. When the remainder has a value of 0, the sign of the remainder is positive.

The resultant value of the calculation is copied into the remainder operand. If this operand is not the same type as that used in performing the operation, the resultant value is converted to its type. If necessary, the resultant value is adjusted to the length of the remainder operand, aligned at the assumed decimal point of the remainder operand, or both before being copied to it.

If significant digits are truncated on the left end of the resultant value, a size exception is signaled for those programs that request to be notified of size exceptions.

If a decimal to binary conversion causes a size exception to be signaled in programs that request to be notified of size exceptions, the binary value contains the correct truncated result only if the decimal value contains 15 or fewer significant nonfractional digits.

Resultant Conditions: The algebraic value of the numeric scalar remainder is positive, negative, or 0.

Exceptions

Exception		Operands 1	2	3	Other
06	Addressing				
	01 space addressing violation	X	X	X	
	02 boundary alignment	X	X	X	
	03 range	X	X	X	
	06 optimized addressability invalid	X	X	X	
08	Argument/parameter				
	01 parameter reference violation	X	X	X	
0C	Computation				
	02 decimal data		X	X	
	0A size	X			
	0B zero divide			X	
10	Damage encountered				
	04 system object damage state	X	X	X	X
	44 partial system object damage	X	X	X	X
1C	Machine-dependent exception				

Exception		Operands 1	2	3	Other
	03 machine storage limit exceeded				X
20	Machine support				
	02 machine check				X
	03 function check				X
22	Object access				
	01 object not found	X	X	X	
	02 object destroyed	X	X	X	
	03 object suspended	X	X	X	
24	Pointer specification				
	01 pointer does not exist	X	X	X	
	02 pointer type invalid	X	X	X	
2A	Program creation				
	05 invalid op code extender field				X
	06 invalid operand type	X	X	X	
	07 invalid operand attribute	X	X	X	
	08 invalid operand value range	X	X	X	
	09 invalid branch target				X
	0C invalid operand odt reference	X	X	X	
	0D reserved bits are not zero	X	X	X	X
2C	Program execution				
	04 invalid branch target				X
2E	Resource control limit				
	01 user profile storage limit exceeded			X	
36	Space management				
	01 space extension/truncation			X	

1.61 Scale (SCALE)

Op Code (Hex)	Operand 1	Operand 2	Operand 3
1063	Receiver	Source	Scale factor

Operand 1: Numeric variable scalar.

Operand 2: Numeric scalar.

Operand 3: Binary(2) scalar.

Optional Forms

Mnemonic	Op Code (Hex)	Form Type
SCALES	1163	Short
SCALEI	1863	Indicator
SCALEIS	1963	Indicator, Short
SCALEB	1C63	Branch
SCALEBS	1D63	Branch, Short

If the short instruction option is indicated in the op code, operand 1 is used as the first and second operational operands (receiver and first source operand). Operand 2 is used as the third operational operand (second source operand).

Extender: Branch or indicator options.

If the branch or indicator option is specified in the op code, the extender field must be present along with one to four branch targets (for branch options) or one to four indicator operands (for indicator options). The branch or indicator operands immediately follow the last operand listed above. See Chapter 1. "Introduction" for the encoding of the extender field and the allowed syntax of the branch and indicator operands.

Description: The scale instruction performs numeric scaling of the source operand based on the scale factor and places the results in the receiver operand. The numeric operation is as follows:

```
Operand 1 = Operand 2 *(B**N)
```

where:

N is the binary integer value of the scale operand. It can be positive, negative, or 0. If N is 0, then the operation simply copies the source operand value into the receiver operand.

B is the arithmetic base for the type of numeric value in the source operand.

Base Type B

Binary 2

Packed/Zoned 10

Floating-point 2

The scale operation is a shift of N binary, packed, or zoned digits. The shift is to the left if N is positive, to the right if N is negative. For a floating-point source

operand, the scale operation is performed as if the source operand is multiplied by a floating-point value of 2**N.

If the source and receiver operands have different attributes, the scaling operation is done in an intermediate field with the same attributes as the source operand. If a fixed-point scaling operation causes nonzero digits to be truncated on the left end of the intermediate field, a size exception is signaled. For a floating-point scaling operation, the floating-point overflow and the floating-point underflow exceptions can be signaled during the calculation of the intermediate result.

The resultant value of the calculation is copied into the receiver operand. If this operand is not the same type as that used in performing the operation, the resultant value is converted to its type. If necessary, the resultant value is adjusted to the length of the receiver operand, aligned at the assumed decimal point of the receiver operand, or both before being copied to it. For fixed-point operations, if nonzero digits are truncated off the left end of the resultant value, a size exception is signaled.

For floating-point operations involving fixed-point receiver fields, if nonzero digits would be truncated from the left end of the resultant value, an invalid floating-point conversion exception is signaled.

For floating-point receiver fields, if the exponent of the resultant value is either too large or too small to be represented in the receiver field, the floating-point overflow or floating-point underflow exception is signaled.

A scalar value invalid exception is signaled if the value of N is beyond the range of the particular type of the source operand as specified in the following table.

Source Operand Type	Maximum Value of N
Signed Binary(2)	$-15 \leq N \leq 15$
Unsigned Binary(2)	$-16 \leq N \leq 16$
Signed Binary(4)	$-31 \leq N \leq 31$
Unsigned Binary(4)	$-32 \leq N \leq 32$
Decimal(P,Q)	$-31 \leq N \leq 31$

For a scale operation in floating-point, no limitations are placed on the values allowed for N other than the implicit limits imposed due to the floating-point overflow and underflow exceptions.

Limitations: The following are limits that apply to the functions performed by this instruction.

If a decimal to binary conversion causes a size exception to be signaled, the binary value contains the correct truncated result only if the decimal value contains 15 or fewer significant nonfractional digits.

Resultant Condition: Positive, negative, or zero-The algebraic value of the receiver operand is positive, negative, or zero. Unordered-The value assigned a floating-point receiver operand is NaN.

Exceptions

Exception		Operands 1	2	3	Other
06	Addressing				
	01 space addressing violation	X	X	X	
	02 boundary alignment	X	X	X	
	03 range	X	X	X	
	06 optimized addressability invalid	X	X	X	
08	Argument/parameter				
	01 parameter reference violation	X	X	X	
0C	Computation				
	02 decimal data		X		
	06 floating-point overflow	X			
	07 floating-point underflow	X			
	09 floating-point invalid operand		X		X
	0A size	X			
	0C invalid floating-point conversion	X			
	0D floating-point inexact result	X			
10	Damage encountered				
	04 system object damage state	X	X	X	X
	44 partial system object damage	X	X	X	X
1C	Machine-dependent exception				
	03 machine storage limit exceeded				X
20	Machine support				
	02 machine check				X
	03 function check				X
22	Object access				
	01 object not found	X	X	X	
	02 object destroyed	X	X	X	
	03 object suspended	X	X	X	
24	Pointer specification				
	01 pointer does not exist	X	X	X	
	02 pointer type invalid	X	X	X	
2A	Program creation				
	05 invalid op code extender field				X
	06 invalid operand type	X	X	X	
	07 invalid operand attribute	X	X	X	
	08 invalid operand value range	X	X	X	
	09 invalid branch target				X

Exception		Operands 1	2	3	Other
	0C invalid operand odt reference	X	X	X	
	0D reserved bits are not zero	X	X	X	X
2C	Program execution				
	04 invalid branch target				X
2E	Resource control limit				
	01 user profile storage limit exceeded				X
32	Scalar specification				
	03 scalar value invalid			X	
36	Space management				
	01 space extension/truncation				X

1.62 Scan (SCAN)

Op Code (Hex)	Operand 1	Operand 2	Operand 3
10D3	Receiver	Base	Compare operand

Operand 1: Binary variable scalar or binary array.

Operand 2: Character variable scalar.

Operand 3: Character scalar.

Optional Forms

Mnemonic	Op Code (Hex)	Form Type
SCANI	18D3	Indicator
SCANB	1CD3	Branch

Extender: Branch or indicator options.

If the branch or indicator option is specified in the op code, the extender field must be present along with one or two branch targets (for branch options) or one or two indicator targets (for indicator options). The branch or indicator targets immediately follow the last operand listed above. See Chapter 1. "Introduction" for the encoding of the extender field and the allowed syntax of the branch and indicator operands.

Description: The character string value of the base operand is scanned for occurrences of the character string value of the compare operand.

The base and compare operands must both be character strings. The length of the compare operand must not be greater than that of the base string.

The operation begins at the left end of the base string and continues character by character, from left to right, comparing the characters of the base string with those of the compare operand. The length of the comparisons are equal to the length of the compare operand value and function as if they were being compared in the Compare Bytes Left-Adjusted instruction.

If a set of bytes that match the compare operand is found, the binary value for the ordinal position of its leftmost base string character is placed in the receiver operand.

If the receiver operand is a scalar, only the first occurrence of the compare operand is noted. If it is an array, as many occurrences as there are elements in the array are noted.

The operation continues until no more occurrences of the compare operand can be noted in the receiver operand or until the number of characters (bytes) remaining to be scanned in the base string is less than the length of the compare operand.

When the second condition occurs, the receiver value is set to 0. If the receiver operand is an array, all its remaining elements are also set to 0.

The base operand and the compare operand can be variable length substring compound operands.

Substring operand references that allow for a null substring reference (a length value of zero) may be specified for operands 2 and 3. The effect of specifying a null substring reference for the compare operand or both operands is that the receiver is set to zero (no match found) and the instruction's resultant condition is null compare operand. Specifying a null substring reference for just the base operand is not allowed due to the requirement that the length of the compare operand must not be greater than that of the base string.

Resultant Conditions: Zero or positive-The numeric value(s) of the receiver operand is either zero or positive. When the receiver operand is an array, the resultant condition is zero if all elements are zero. One of these two conditions will result when the compare operand is not a null substring reference. Null compare operand-The compare operand is a null substring reference; therefore, the receiver has been set to zero which indicates that no occurrences were found.

Exceptions

Exception		Operands 1	2	3	Other
06	Addressing				
	01 space addressing violation	X	X	X	
	02 boundary alignment	X	X	X	
	03 range	X	X	X	
	06 optimized addressability invalid	X	X	X	
08	Argument/parameter				
	01 parameter reference violation	X	X	X	
0C	Computation				
	08 length conformance		X	X	
10	Damage encountered				
	04 system object damage state	X	X	X	X
	44 partial system object damage	X	X	X	X
1C	Machine-dependent exception				
	03 machine storage limit exceeded				X
20	Machine support				
	02 machine check				X
	03 function check				X
22	Object access				
	01 object not found	X	X	X	
	02 object destroyed	X	X	X	

Exception		Operands 1	2	3	Other
	03 object suspended	X	X	X	
24	Pointer specification				
	01 pointer does not exist	X	X	X	
	02 pointer type invalid	X	X	X	
2A	Program creation				
	05 invalid op code extender field				X
	06 invalid operand type	X	X	X	
	07 invalid operand attribute	X	X	X	
	08 invalid operand value range	X	X	X	
	09 invalid branch target operand				X
	0A invalid operand length		X	X	
	0C invalid operand odt reference	X	X	X	
	0D reserved bits are not zero	X	X	X	X
2C	Program execution				
	04 invalid branch target				X
2E	Resource control limit				
	01 user profile storage limit exceeded				X
36	Space management				
	01 space extension/truncation				X

1.63 Scan with Control (SCANWC)

Op Code (Hex)	Operand 1	Operand 2	Operand 3	Operand 4
10E4	Base locator	Controls	Options	Escape target or null

Operand 1: Space pointer.

Operand 2: Character(8) variable scalar (fixed length).

Operand 3: Character(4) scalar (fixed length).

Operand 4: Instruction number, relative instruction number, branch point, instruction pointer, instruction definition list element, or null.

Optional Forms

Mnemonic	Op Code (Hex)	Form Type
SCANWC	10E4	Short
SCANWCI	18E4	Indicator
SCANWCB	1C84	Branch

Extender: Branch or indicator options.

If the branch or indicator option is specified in the op code, the extender field must be present along with one to four branch targets (for branch options) or one to four indicator operands (for indicator options). The branch or indicator operands immediately follow the last operand listed above. See Chapter 1. "Introduction" for the encoding of the extender field and the allowed syntax of the branch and indicator operands.

Description: The base string to be scanned is specified by the base locator and controls operands. The base locator addresses first character of the base string. The controls specifies the length of the base string in the base length field.

The scan operation begins at the left end of the base string and continues character by character, left-to-right. The scan operation can be performed on a base string which contains all simple (1-byte) or all extended (2-byte) character codes or a mixture of the two. When the base string is being interpreted in simple character mode, the operation moves through the base string one byte at a time. When the base string is being interpreted in extended character mode, the operation moves through the base string 2 bytes at a time. The character string value of the base operand is scanned for occurrences of a character value satisfying the criteria specified in the control and options operands.

The scan is completed by updating the base locator and controls operands with scan status when a character value being scanned for is found, the end of the base string is encountered, or an escape code is encountered when the escape target operand is specified. The base locator is set with addressability to the character (simple or extended) which caused the instruction to complete execution. The controls operand is set with information which identifies the mode (simple or extended) of the base string character addressed by the base locator and which provides for resumption of the scan operation with minimal overhead.

The controls and options operands specify the modes to be used in interpreting characters during the scan operation. Characters can be interpreted in one of two character modes: simple (1-byte) and extended (2-byte). Additionally, the base string can be scanned in one of two scan modes, mixed (base string may contain a mixture of both character modes) and nonmixed (base string contains one mode of characters).

When the mixed scan mode is specified in the options operand, the base string is interpreted as containing a mixture of simple and extended character codes. The mode, simple or extended, with which the string is to be interpreted, is controlled initially by the base mode indicator in the controls operand and thereafter by mode control characters imbedded in the base string. The mode control characters are as follows:

- Hex 0E = Shift out (SO) of simple character mode to extended mode.
- Hex 0F = Shift in (SI) to simple character mode from extended mode. This is only recognized if it occurs in the first byte position of an extended character code.

When the nonmixed scan mode is specified in the options operand, the base string is interpreted using only the character mode specified by the base mode indicator in the controls operand. Character mode shifting can not occur because no mode control characters are recognized when scanning in nonmixed mode.

The base locator operand is a space pointer which is both input to and output from the instruction. On input, it locates the first character of the base string to be processed. On output, it locates the character of the base string which caused the instruction to complete.

The controls operand must be a character scalar which specifies additional information to be used to control the scan operation. It must be at least 8 bytes long and have the following format:

- Controls operand — Char(8)
  - Control indicators — Char(1)
  - Reserved — Char(1)
  - Comparison characters — Char(2)
  - Reserved — Char(1)
  - Base end — Char(3)
    - Instruction work area — Char(1)
    - Base length — Char(2)

Only the first 8 bytes of the controls operand are used. Any excess bytes are ignored. Reserved fields must contain binary zeros. The control indicators field has the following format:

- Control indicators — Char(1)
  - Base mode — Bit 0

    0 = Simple
    1 = Extended
  - Comparison character mode — Bit 1

0 = Simple
1 = Extended

- Reserved (must be 0) Bit 2-6
- Scan state Bit 7

  0 = Resume scan
  1 = Start scan

The base mode is both input to and output from the instruction. In either case, it specifies the mode of the character in the base string currently addressed by the base locator.

The comparison character mode is not changed by the instruction. It specifies the mode of the comparison character contained in the controls operand.

The scan state is both input to and output from the instruction. As input, it indicates whether the scan operation for the base string is being started or resumed. If it is being started, the instruction assumes that the base length value in the base end field of the controls operand specifies the length of the base string, and the instruction work area value is ignored. If it is being resumed, the instruction assumes the base end field has been set by a prior start scan execution of the instruction with an internal machine value identifying the end of the base string.

For a start scan execution of the instruction, the scan state indicator is reset to indicate resume scan to provide for subsequent resumption of the scan operation. Additionally, for a start scan execution of the instruction, the base end field is set with an internally optimized value which identifies the end of the base string being scanned. This value then overlays the values which were in the instruction work area and base length fields on input to the instruction. Predictable operation of the instruction on a resume scan execution depends upon this base end field being left intact with the value set by the start scan execution.

For a resume scan execution of the instruction, the scan state and base end fields are unchanged.

The comparison character is input to the instruction. It specifies a character code to be used in the comparisons performed during the scanning of the base string. The comparison character mode in the control indicators specifies the mode (simple or extended) of the comparison character. If it is a simple character, the first byte of the comparison character field is ignored and the comparison character is assumed to be specified in the second byte. If it is an extended character, the comparison character is specified as a 2-byte value in the comparison character field.

The base end field is both input to and output from the instruction. It contains data which identifies the end of the base string. Initially, for a start scan execution of the instruction, it contains the length of the base string in the base length field. Additionally, the base end field is used to retain information over multiple instruction executions which provides for minimizing the overhead required to resume the scan operation for a particular base string. This information is set on the initial start scan execution of the instruction and is used during subsequent resume scan executions of the instruction to determine the end of the base string to be scanned. If the end of the base string being scanned must be altered during iterative usage of this instruction, a start scan execution of the

instruction must be performed to provide for correctly resetting the internally optimized value to be stored in the base end from the values specified in the base locator operand and base length field.

For the special case of a start scan execution where a length value of zero (no characters to scan) is specified in the base length field, the instruction results in a not found resultant condition. In this case, the base locator is not verified and the scan state indicator, the base end field, and the base locator are not changed. The options operand must be a character scalar which specifies the options to be used to control the scan operation. It must be at least 4 bytes in length and has the following format:

- Options operand — Char(4)
  - Options indicators — Char(1)
  - Reserved — Char(3)

The options operand must be specified as a constant character scalar.

Only the first 4 bytes of the options operand are used. Any excess bytes are ignored. Reserved fields must contain binary zeros. The option indicators field has the following format:

- Option indicators — Char(1)
  - Reserved — Bit 0
  - Scan mode — Bit 1

    0 = Mixed
    1 = Nonmixed
  - Reserved — Bits 2-3
  - Comparison relation — Bits 4-6
    - Equal, (=) relation — Bit 4
    - Less than, (<) relation — Bit 5
    - Greater than, (>) relation — Bit 6

      0 = No match on relation
      1 = Match on relation
  - Reserved — Bit 7

The scan mode specifies whether the base string contains a mixture of character modes, or contains all one mode of characters; that is, whether or not mode control characters should be recognized in the base string. Mixed specifies that there is a mixture of character modes and, therefore, mode control characters should be recognized. Nonmixed specifies that there is not a mixture of character modes and, therefore, mode control characters should not be recognized. Note that the base mode indicator in the controls operand specifies the character mode of the base string character addressed by the base locator.

The comparison relation specifies the relation or relations of the comparison character to characters of the base string which will satisfy the scan operation and cause completion of the instruction with one of the height, low, or equal resultant conditions. Multiple relations may be specified in conjunction with one another. Specifying all relations insures a match against any character in the base string which is of the same mode as the comparison character. Specifying

no relation insures a not found resultant condition, in the absence of an escape due to verification, regardless of the values of the characters in the base string which match the mode of the comparison character.

An example of comparison scanning is a scan of simple mode characters for a value less than hex 40. This could be done by specifying a comparison character of hex 40 and a comparison relation of greater than in conjunction with a branch option for the resultant condition of high. This could also be done by specifying a comparison character of hex 3F and comparison relations of equal and greater than in conjunction with branch options for equal and high. The target of the branch options in either case would be the instructions to process the character less than hex 40 in value.

The escape target operand controls the verification of bytes of the base string for values less than hex 40. Verification, if requested, is always performed in conjunction with whatever comparison processing has been requested. That is, verification is performed even if no comparison relation is specified. This operand is discussed in more detail in the following material.

During the scan operation, the characters of the base string which are not of the same mode as the comparison character are skipped over until the mode of the characters being processed is the same as the mode of the comparison character. The operation then proceeds by comparing the comparison character with each of the characters of the base string. These comparisons behave as if the characters were being compared in the Compare Bytes Left Adjusted instruction.

If a base string character satisfying the criteria specified in the controls and options operands is found, the base locator is set to address the first byte of it, the base mode indicator is set to indicate the mode of the base string as of that character, and the instruction is completed with the appropriate resultant condition based on the relation (high, low, or equal) of the comparison character to the base string character.

If a matching base string character is not found prior to encountering a mode change, the characters of the base string are again skipped over until the mode of the characters being processed is the same as the mode of the comparison character before comparisons are resumed.

If a matching base string character is not found prior to encountering the end of the base string, the base location is set to address the first byte of the character encountered at the end of the base string, the base mode indicator is set to indicate the mode of the base string as of that character, and the instruction is completed with the not found resultant condition. A mode control string results in the changing of the base string mode, but the base locator is left addressing the mode control character.

If the escape target operand is specified (operand 4 is not null), verifications are performed on the characters of the base string prior to their being skipped or compared with the comparison character. Each byte of the base string is checked for a value less than hex 40. Additionally, for a mixed scan mode, when such a value is encountered, it is then determined if it is a valid mode control character.

- Hex 0E (SO) when the base string is being interpreted in simple character mode.

- Hex 0F (SI) in the left byte of the character code when the base string is being interpreted in extended character mode.

If a byte value of less than hex 40 is not a valid mode control character, it is considered to be an escape code. The base locator is set to address the first byte of the base string character (simple or extended) which contains the escape code, the base mode indicator is set to indicate the mode of the base string as of that character, and a branch is taken to the target specified by the escape target operand. When the escape target branch is performed, the value of any optional indicator operands is meaningless.

If the escape target operand is not specified (operand 4 is null), verifications of the character codes in the base string are not performed. However, for a mixed scan mode, mode control values are always processed as described previously under the discussion of the mixed scan mode.

Substring operand references which allow for a null substring reference (a length value of zero) may not be specified for this instruction.

Variable length substring compound operands may not be specified for operands two and three.

If possible, use a Space Pointer Machine Object for the base locator, operand 1. Appreciably less overhead is incurred in accessing and storing the value of the base locator if this is done.

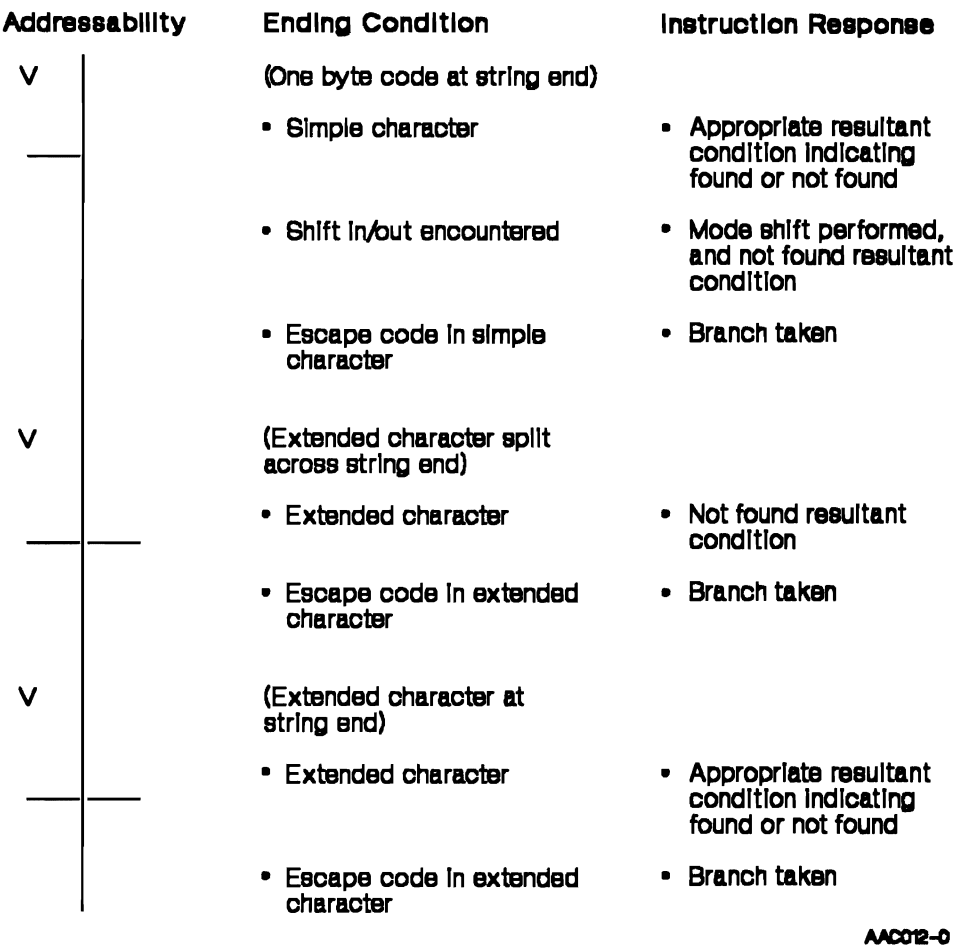
If possible, specify through its ODT definition, the controls operand on an 8-byte multiple (doubleword) boundary relative to the start of the space containing it. Appreciably less overhead is incurred in accessing and storing the value of the controls if this is done.

For the case where a base string is to be just scanned for byte values less than hex 40, two techniques can be used.

- A direct simple mode scan for a value less than hex 40 without usage of the escape target verification feature.
- A scan for any character with usage of the escape target verification feature.

  The direct scan approach, the former, is the more efficient.

The following diagram defines the various conditions which can be encountered at the end of the base string and what the base locator addressability is in each case. The solid vertical line represents the end of the base string. The dashes represent the bytes before and after the base string end. The V is positioned over the byte addressed by the base locator in each case. These are the conditions which can be encountered when the base locator input to the instruction addresses a byte prior to the base string end. When the base length field specifies a value of zero for a start scan execution of the instruction, or the input base locator addresses a point beyond the end of the instruction, no processing is performed and the instruction is immediately completed with the not found resultant condition.



An analysis of the diagram shows that normally, after appropriate processing for the particular found, not found, or escape condition, the scan can be restarted at the byte of data which would follow the base string end in the data stream being scanned. Any mode shift required by an ending mode control character will have been performed.

However, one ending condition may require subsequent resumption of the scan at the character encountered at the end of the base string. This is the case where the instruction completes with the not found resultant condition and the base string ends with an extended character split across string end. That is, the base mode indicator specifies extended mode, the base locator addresses the last byte of the base string, and that byte value is not a shift out, hex 0E character. In this case, complete verification of the extended character and relation comparison could not be performed. If this extended character is to be processed, it must be done through another execution of the Scan instruction where both bytes of the character can be input to the instruction within the confines of the base string.

Resultant Conditions

- High, Low, Equal: A character value was found in the base string which satisfies the criteria specified in the controls and options operands in that the comparison character is of higher, lower, or equal string value to the base string character.
- Not found: A character value was not found in the base string which satisfied the criteria specified in the controls and options operands.

Exceptions

Exception		Operands 1	2	3	4	Other
06	Addressing					
	01 space addressing violation	X	X	X	X	
	02 boundary alignment	X	X	X	X	
	03 range	X	X	X	X	
	06 optimized addressability invalid	X	X	X	X	
08	Argument/parameter					
	01 parameter reference violation	X	X	X	X	
0C	Computation					
	08 length conformance		X	X		
10	Damage encountered					
	04 System object damage state					X
	44 partial system object damage					X
1C	Machine-dependent exception					
	03 machine storage limit exceeded					X
20	Machine support					
	02 machine check					X
	03 function check					X
22	Object access					
	01 object not found	X	X	X	X	
	02 object destroyed	X	X	X	X	
	03 object suspended	X	X	X	X	
24	Pointer specification					
	01 pointer does not exist	X	X	X	X	
	02 pointer type invalid	X	X	X	X	
2A	Program creation					
	05 invalid op code extender field					X
	06 invalid operand type	X	X	X	X	
	07 invalid operand attribute	X	X	X	X	
	08 invalid operand value range	X	X	X	X	
	09 invalid branch target operand				X	X

Exception		Operands 1	2	3	4	Other
	0A invalid operand length		X	X		
	0C invalid operand odt reference	X	X	X	X	
	0D reserved bits are not zero	X	X	X	X	X
2C	Program execution					
	04 invalid branch target					X
2E	Resource control limit					
	01 user profile storage limit exceeded					X
32	Scalar specification					
	01 scalar type invalid	X	X	X	X	
	03 scalar value invalid		X	X		
36	Space management					
	01 space extension/truncation					X

1.64 Search (SEARCH)

Op Code (Hex)	Operand 1	Operand 2	Operand 3	Operand 4
1084	Receiver	Array	Find	Location

Operand 1: Binary variable scalar or binary variable array.

Operand 2: Character array or numeric array.

Operand 3: Character variable scalar or numeric variable scalar.

Operand 4: Binary scalar.

Optional Forms

Mnemonic	Op Code (Hex)	Form Type
SEARCHI	1884	Indicator
SEARCHB	1C84	Branch

Extender: Branch or indicator options.

If the branch or indicator option is specified in the op code, the extender field must be present along with one or two branch targets (for branch options) or one or two indicator targets (for indicator options). The branch or indicator targets immediately follow the last operand listed above. See Chapter 1. "Introduction" for the encoding of the extender field and the allowed syntax of the branch and indicator operands.

Description: The portions of the array operand indicated by the location operand are searched for occurrences of the value indicated in the find operand.

The operation begins with the first element of the array operand and continues element by element, comparing those characters of each element (beginning with the character indicated in the location operand) with the characters of the find operand. The location operand contains an integer value representing the relative location of the first character in each element to be used to begin the compare.

The integer value of the location operand must range from 1 to L, where L is the length of the array operand elements; otherwise, a scalar value invalid exception is signaled. A value of 1 indicates the leftmost character of each element.

The array and find operands can be either character or numeric. Any numeric operands are interpreted as logical character strings. The compares between these operands are performed at the length of the find operand and function as if they were being compared in the Compare Bytes Left-Adjusted instruction.

The length of the find operand must not be so large that it exceeds the length of the array operand elements when used with the location operand value. The array element length used is the length of the array scalar elements and not the length of the entire array element, which can be larger in noncontiguous arrays.

As each occurrence of the find value is encountered, the integer value of the index for this array element is placed in the receiver operand. If the receiver operand is a scalar, only the first element containing the find value is noted. If the receiver operand is an array, as many occurrences as there are elements within the receiver array are noted.

If the value of the index for an array element containing an occurrence of the find value is too large to be contained in the receiver, a size exception is signaled.

The operation continues until no more occurrences of elements containing the find value can be noted in the receiver operand or until the array operand has been completely searched. When the second condition occurs, the receiver value is set to LB-1, where LB is the value of the lower bound index of the array. If LB is the most negative 32-bit integer, then LB-1 is the most positive 32-bit integer; otherwise, LB-1 is 1 less than LB. If the receiver operand is an array, all its remaining elements are also set to LB-1. The find operand can be a variable length substring compound operand.

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for this instruction.

Resultant Conditions: The numeric value(s) of the receiver operand is either LB-1 or in the range LB through UB, where UB is the value of the upper bound index of the array. When the receiver is LB-1, the resultant condition is zero. When the receiver is in the range LB through UB, the resultant condition is positive. When the receiver is an array, the resultant condition is zero if all elements are LB-1; otherwise, it is positive. The resultant condition is unpredictable when the No Binary Size Exception program template option is used.

Exceptions

Exception		Operands 1	2	3	4	Other
06	Addressing					
	01 space addressing violation	X	X	X	X	
	02 boundary alignment	X	X	X	X	
	03 range	X	X	X	X	
	06 optimized addressability invalid	X	X	X	X	
08	Argument/parameter					
	01 parameter reference violation	X	X	X	X	
0C	Computation					
	08 length conformance		X	X		
	0A size	X				
10	Damage encountered					
	04 system object damage state	X	X	X	X	X
	44 partial system object damage	X	X	X	X	X
1C	Machine-dependent exception					
	03 machine storage limit exceeded					X

Exception		Operands 1	2	3	4	Other
20	Machine support					
	02 machine check					X
	03 function check					X
22	Object access					
	01 object not found	X	X	X	X	
	02 object destroyed	X	X	X	X	
	03 object suspended	X	X	X	X	
24	Pointer specification					
	01 pointer does not exist	X	X	X	X	
	02 pointer type invalid	X	X	X	X	
2A	Program creation					
	05 invalid op code extender field					X
	06 invalid operand type	X	X	X	X	
	07 invalid operand attribute	X	X	X	X	
	08 invalid operand value range	X	X	X	X	
	09 invalid branch target operand					X
	0A invalid operand length			X		
	0C invalid operand odt reference	X	X	X	X	
	0D reserved bits are not zero	X	X	X	X	X
2C	Program execution					
	04 invalid branch target					X
2E	Resource control limit					
	01 user profile storage limit exceeded					X
32	Scalar specification					
	01 scalar type invalid	X	X	X	X	
	03 scalar value invalid				X	
36	Space management					
	01 space extension/truncation					X

1.65 Set Bit in String (SETBTS)

Op Code (Hex)	Operand 1	Operand 2
101E	Source	Offset

Operand 1: Character Variable Scalar or Numeric Variable Scalar.

Operand 2: Binary Scalar.

Description: Sets the bit of the receiver operand as indicated by the bit offset operand.

The selected bit from the receiver operand is set to a value of B'1'.

The receiver operand can be a character or numeric variable. The leftmost bytes of the receiver operand are used in the operation. The receiver operand is interpreted as a bit string with the bits numbered left to right from 0 to the total number of bits in the string minus 1.

The receiver cannot be a variable substring.

The offset operand indicates which bit of the receiver operand is to be set, with a offset of zero indicating the leftmost bit of the leftmost byte of the receiver operand. This value may be specified as a constant or any valid binary scalar variable.

If a offset value less than zero or beyond the length of the receiver is specified a "scalar value invalid" exception is raised.

Exceptions

Exception		Operands 1	Operands 2	Other
06	Addressing			
	01 Spacing addressing violation	X	X	
	02 Boundary alignment violation	X	X	
	03 Range	X	X	
	06 Optimized addressability invalid	X	X	
08	Argument/parameter			
	01 Parameter reference violation	X	X	
10	Damage encountered			
	04 System object damage state			X
	44 Partial system object damage			X
1C	Machine-dependent exception			
	03 Machine storage limit exceeded			X
20	Machine support			
	02 Machine check			X
	03 Function check			X

Exception		Operands 1	2	Other
22	Object access			
	02 Object destroyed	X	X	
	03 Object suspended	X	X	
24	Pointer specification			
	01 Pointer does not exist	X	X	
	02 Pointer type invalid	X	X	
2A	Program creation			
	06 Invalid operand type	X	X	
	07 Invalid operand attribute	X	X	
	08 Invalid operand value range	X	X	
	0A Invalid operand length	X	X	
	0C Invalid operand odt reference	X	X	
	0D Reserved bits are not zero	X	X	X
2E	Resource control limit			
	01 user profile storage limit exceeded			X
32	Scalar specification			
	01 Scalar type invalid	X	X	
	03 Scalar value invalid		X	
36	Space management			
	01 space extension/truncation			X

1.66 Set Instruction Pointer (SETIP)

Op Code (Hex)	Operand 1	Operand 2
1022	Receiver	Branch target

Operand 1: Instruction pointer.

Operand 2: Instruction number, relative instruction number, or branch point.

Description: The value of the branch target (operand 2) is used to set the value of the instruction pointer specified by operand 1. The instruction number indicated by the branch target must provide the address of an instruction within the program containing the Set Instruction Pointer instruction.

Exceptions

Exception		Operands 1	2	Other
06	Addressing			
	01 space addressing violation	X		
	02 boundary alignment	X		
	03 range	X		
	06 optimized addressability invalid	X		
08	Argument/parameter			
	01 parameter reference violation	X		
10	Damage encountered			
	04 system object damage state	X	X	X
	44 partial system object damage	X	X	X
1C	Machine-dependent exception			
	03 machine storage limit exceeded			X
20	Machine support			
	02 machine check			X
	03 function check			X
22	Object access			
	01 object not found	X		
	02 object destroyed	X		
	03 object suspended	X		
24	Pointer specification			
	01 pointer does not exist	X		
	02 pointer type invalid	X		
2A	Program creation			
	06 invalid operand type	X	X	
	07 invalid operand attribute	X		

Exception		Operands 1	2	Other
	08 invalid operand value range	X		
	09 invalid branch target operand		X	
	0C invalid operand odt reference	X	X	
	0D reserved bits are not zero	X	X	X
2C	Program execution			
	04 branch target invalid		X	
2E	Resource control limit			
	01 user profile storage limit exceeded			X
36	Space management			
	01 space extension/truncation			X

1.67 Store and Set Computational Attributes (SSCA)

Op Code (Hex)	Operand 1	Operand 2	Operand 3
107B	Receiver	Source	Controls

Operand 1: Character(5) variable scalar (fixed length).

Operand 2: Character(5) scalar or null (fixed length).

Operand 3: Character(5) scalar or null (fixed length).

Description: This instruction stores and optionally sets the attributes for controlling computational operations for the process this instruction is executed in.

The receiver is assigned the values that each of the computational attributes had at the start of execution of the instruction. It has the same format and bit assignment as the source.

The source specifies new values for the computational attributes for the process. The particular computational attributes that are selected for modification are determined by the controls operand. The source operand has the following format:

- Floating-point exception masks — Char(2)

  0 = Disabled (exception is masked)
  1 = Enabled (exception is unmasked)

  - Reserved (binary 0) — Bits 0-9
  - Floating-point overflow — Bit 10
  - Floating-point underflow — Bit 11
  - Floating-point zero divide — Bit 12
  - Floating-point inexact result — Bit 13
  - Floating-point invalid operand — Bit 14
  - Reserved (binary 0) — Bit 15

- Floating-point exception occurrence flags — Char(2)

  0 = Exception has not occurred
  1 = Exception has occurred

  - Reserved (binary 0) — Bits 0-9
  - Floating-point overflow — Bit 10
  - Floating-point underflow — Bit 11
  - Floating-point zero divide — Bit 12
  - Floating-point inexact result — Bit 13
  - Floating-point invalid operand — Bit 14
  - Reserved (binary 0) — Bit 15

- Modes — Char(1)

- Reserved — Bit 0
- Floating-point rounding mode — Bits 1-2

  00 = Round toward positive infinity
  01 = Round toward negative infinity
  10 = Round toward zero
  11 = Round to nearest (default)

- Reserved — Bits 3-7

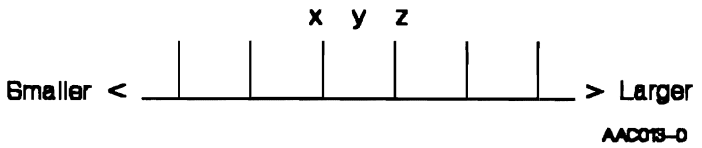
The controls operand is used to select those attributes that are to be set from the bit values of the source operand. The format of the controls is the same as that for the source. A value of one for a bit in controls indicates that the corresponding computational attribute for the process is to be set from the value of that bit of the source. A value of zero for a bit in controls indicates that the corresponding computational attribute for the process is not to be changed, and will retain the value it had prior to this instruction. For an attribute controlled by a multiple-bit field, such as the rounding modes, all of the bits in the field must be ones or all must be zeros. A mixture of ones and zeros in such a field results in a scalar value invalid exception.

If the source and controls operands are both null, the instruction will just return the current computational attributes. If the source is specified, the computational attributes of the process are modified under control of the controls operand. If the source operand is specified and the controls operand is null, the instruction will change all of the computational attributes to the values specified in the source. If the source operand is null and the controls operand is specified, an invalid operand type exception is signaled.

With the floating-point exception masks field, it is possible to unmask/mask the exception processing and handling for each of the five floating-point exceptions. If an exception that is unmasked occurs, then the corresponding exception occurrence bit is set, and the exception is signaled. If an exception that is masked occurs, the exception is not signaled, but the exception occurrence flag is still set to indicate the occurrence of the exception.

The floating-point exception occurrence flag for each exception may be set or cleared by this instruction from the source operand under control of the controls operand.

Unless specified otherwise by a particular instruction, or precluded due to implicit conversions, all floating-point operations are performed as if correct to infinite precision, and then rounded to fit in a destinations format while potentially signaling an exception that the result is inexact. To allow control of the floating-point rounding operations performed within a process, four floating-point rounding modes are supported. Assume y is the infinitely precise number that is to be rounded, bracketed most closely by x and z, where x is the largest representable value less than y and z is the smallest representable value greater than y. Note that x or z may be infinity. The following diagram shows this relationship of x, y, and z on a scale of numerically progressing values where the vertical bars denote values representable in a floating-point format.



Given the above, if y is not exactly representable in the receiving field format, the rounding modes change y as follows:

Round to nearest with round to even in case of a tie is the default rounding mode in effect upon the initiation of a process. For this rounding mode, y is rounded to the closer of x or z. If they are equally close, the even one (the one whose least significant bit is a zero) is chosen. For the purposes of this mode of rounding, infinity is treated as if it was even. Except for the case of y being rounded to a value of infinity, the rounded result will differ from the infinitely precise result by at most half of the least significant digit position of the chosen value. This rounding mode differs slightly from the decimal round algorithm performed for the optional round form of an instruction. This rounding mode would round a value of 0.5 to 0, where the decimal round algorithm would round that value to 1.

Round toward positive infinity indicates directed rounding upward is to occur. For this mode, y is rounded to z.

Round toward negative infinity indicates directed rounding downward is to occur. For this mode, y is rounded to x.

Round toward zero indicates truncation is to occur. For this mode, y is rounded to the smaller (in magnitude) of x or z.

Arithmetic operations upon infinity are exact. Negative infinity is less than every finite value, which is less than positive infinity.

The computational attributes are set with a default value upon process initiation. The default attributes are as follows:

- The floating-point inexact result exception is masked. The other floating-point exceptions are unmasked.
- All occurrence bits have a zero value.
- Round to the nearest rounding mode.

These attributes can be modified by a program executing this instruction. The new attributes are then in effect for the program executing this instruction and for programs invoked subsequent to it unless changed through another execution of this instruction. External exception handlers and invocation exit routines are invoked with the same attributes as were last in effect for the program invocation they are related to. Event handlers do not really relate to another invocation in the process. As such, they are invoked with the attributes that were in effect at the point the process was interrupted to handle the event.

Upon return to the invocation of a program from subsequent program invocations, the computational attributes, other than exception occurrence attributes, are restored to those that were in effect when the program gave up control. The exception occurrence attributes are left intact reflecting the occur-

rence of any floating-point exceptions during the execution of subsequent invocations.

Internal exception handlers execute under the invocation of the program containing them. As such, the above discussion of how computational attributes are restored upon returning from an external exception handler does not apply. The execution of an internal exception handler occurs in a manner similar to the execution of an internal subroutine invoked through the Call Internal instruction. If the internal exception handler modifies the attributes, the modification remains in effect for that invocation when the exception handler completes the exception.

Limitations: The following are limits that apply to the functions performed by this instruction.

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for this instruction.

Exceptions

Exception		Operands 1	2	3	Other
06	Addressing				
	01 space addressing violation	X	X	X	
	02 boundary alignment	X	X	X	
	03 range	X	X	X	
	06 optimized addressability invalid	X	X	X	
08	Argument/parameter				
	01 parameter reference violation	X	X	X	
10	Damage encountered				
	44 partial system object damage				X
1C	Machine-dependent exception				
	03 machine storage limit exceeded				X
20	Machine support				
	02 machine check				X
	03 function check				X
22	Object access				
	01 object not found	X	X	X	
	02 object destroyed	X	X	X	
	03 object suspended	X	X	X	
24	Pointer specification				
	01 pointer does not exist	X	X	X	
	02 pointer type invalid	X	X	X	
2A	Program creation				
	06 invalid operand type	X	X	X	
	07 invalid operand attribute	X	X	X	

Exception		Operands 1	2	3	Other
	08 invalid operand value range	X	X	X	
	0C invalid operand odt reference	X	X	X	
	0D reserved bits are not zero	X	X	X	X
2E	Resource control limit				
	01 user profile storage limit exceeded				X
32	Scalar specification				
	01 scalar type invalid	X	X	X	
	03 scalar value invalid		X	X	
36	Space management				
	01 space extension/truncation				X

1.68 Subtract Logical Character (SUBLC)

Op Code (Hex)	Operand 1	Operand 2	Operand 3
1027	Difference	Minuend	Subtrahend

Operand 1: Character variable scalar (fixed-length).

Operand 2: Character scalar (fixed-length).

Operand 3: Character scalar (fixed-length).

Optional Forms

Mnemonic	Op Code (Hex)	Form Type
SUBLCS	1127	Short
SUBLCI	1827	Indicator
SUBLCIS	1927	Indicator, Short
SUBLCB	1C27	Branch
SUBLCBS	1D27	Branch, Short

If the short instruction option is indicated in the op code, operand 1 is used as the first and second operational operands (receiver and first source operand). Operand 2 is used as the third operational operand (second source operand).

Extender: Branch or indicator options.

If the branch or indicator option is specified in the op code, the extender field must be present along with one to three branch targets (for branch options) or one to three indicator operands (for indicator options). The branch or indicator operands immediately follow the last operand listed above. See Chapter 1. "Introduction" for the encoding of the extender field and the allowed syntax of the branch and indicator operands.

Description: The unsigned binary value of the subtrahend operand is subtracted from the unsigned binary value of the minuend operand, and the result is placed in the difference operand.

Operands 1, 2, and 3 must be the same length; otherwise, the Create Program instruction signals an invalid length exception.

The subtraction operation is performed as though the ones complement of the second operand and a low-order 1-bit were added to the first operand.

The result value is then placed (left-adjusted) into the receiver operand with truncating or padding taking place on the right. The pad value used in this instruction is a byte value of hex 00.

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for this instruction.

If operands overlap but do not share all of the same bytes, results of operations performed on these operands are not predictable. If overlapped operands share

all of the same bytes, the results are predictable when direct addressing is used. If indirect addressing is used (that is, based operands, parameters, strings with variable lengths, and arrays with variable subscripts), the results are not always predictable.

Resultant Conditions: The logical difference of the character scalar operands is zero with carry out of the high-order bit position, not-zero with carry, or not-zero with no carry.

Exceptions

Exception		Operands 1	2	3	Other
06	Addressing				
	01 Space addressing violation	X	X	X	
	02 Boundary alignment	X	X	X	
	03 Range	X	X	X	
	06 Optimized addressability invalid	X	X	X	
08	Argument/Parameter				
	01 Parameter reference violation	X	X	X	
10	Damage Encountered				
	04 System object damage state	X	X	X	X
	44 Partial system object damage	X	X	X	X
1C	Machine-Dependent Exception				
	03 Machine storage limit exceeded				X
20	Machine Support				
	02 Machine check				X
	03 Function check				X
22	Object Access				
	01 Object not found	X	X	X	
	02 Object destroyed	X	X	X	
	03 Object suspended	X	X	X	
24	Pointer Specification				
	01 Pointer does not exist	X	X	X	
	02 Pointer type invalid	X	X	X	
2A	Program Creation				
	05 Invalid op code extender field				X
	06 Invalid operand type	X	X	X	
	07 Invalid operand attribute	X	X	X	
	08 Invalid operand value range	X	X	X	
	09 Invalid branch target operand				X
	0A Invalid operand length	X	X	X	
	0C Invalid operand ODT reference	X	X	X	

Exception		Operands 1	2	3	Other
	0D Reserved bits are not zero	X	X	X	X
2C	Program Execution				
	04 Invalid branch target				X
2E	Resource Control Limit				
	01 User Profile storage limit exceeded				X
32	Scalar Specification				
	01 Scalar type invalid	X	X	X	
	02 Scalar attributes invalid	X	X	X	
36	Space Management				
	01 Space Extension/Truncation				X

1.69 Subtract Numeric (SUBN)

Op Code (Hex)	Operand 1	Operand 2	Operand 3
1047	Difference	Minuend	Subtrahend

Operand 1: Numeric variable scalar.

Operand 2: Numeric scalar.

Operand 3: Numeric scalar.

Optional Forms

Mnemonic	Op Code (Hex)	Form Type
SUBNS	1147	Short
SUBNR	1247	Round
SUBNSR	1347	Short, Round
SUBNB	1C47	Branch
SUBNBS	1D47	Branch, Short
SUBNBR	1E47	Branch, Round
SUBNBSR	1F47	Branch, Short, Round
SUBNI	1847	Indicator
SUBNIS	1947	Indicator, Short
SUBNIR	1A47	Indicator, Round
SUBNISR	1B47	Indicator, Short, Round.

The short form of the SUBTRACT NUMERIC instruction accepts two operands. The first operand is the Minuend before execution and the Difference after execution. The Minuend is replaced by the Difference after the instruction completes. The second operand is the Subtrahend.

Extender: Branch or indicator options.

If the branch or indicator option is specified in the op code, the extender field must be present along with one to four branch targets (for branch options) or one to four indicator operands (for indicator options). The branch or indicator operands immediately follow the last operand listed above. See Chapter 1. "Introduction" for the encoding of the extender field and the allowed syntax of the branch and indicator operands.

Description: The Difference is the result of subtracting the Subtrahend from the Minuend.

Operands can have floating-point, packed or zoned decimal, signed or unsigned binary type.

Source operands are the Minuend and Subtrahend. The receiver operand is the Difference.

If operands have different types, source operands, Minuend and Subtrahend, are converted according to the following rules:

1. If any one of the operands has floating point type, source operands are converted to floating point type.
2. Otherwise, if any one of the operands has zoned or packed decimal type, source operands are converted to packed decimal.
3. Otherwise, the binary operands are converted to a like type. Note: unsigned binary(2) scalars are logically treated as signed binary(4) scalars.

Minuend and Subtrahend are subtracted according to their type. Floating point operands are subtracted using floating point subtraction. Packed decimal operands are subtracted using packed decimal subtraction. Unsigned binary subtraction is used with unsigned binary operands. Signed binary operands are subtracted using two's complement binary subtraction.

Better performance can be obtained if all operands have the same type. Signed and unsigned binary subtractions execute faster than either packed decimal or floating point subtractions.

Decimal operands used in floating-point operations cannot contain more than 15 total digit positions.

For a decimal operation, alignment of the assumed decimal point takes place by padding with 0's on the right end of the source operand with lesser precision.

Floating-point subtraction uses exponent comparison and significand subtraction. Alignment of the binary point is performed, if necessary, by shifting the significand of the value with the smaller exponent to the right. The exponent is increased by one for each binary digit shifted until the two exponents agree.

The operation uses the length and the precision of the source and receiver operands to calculate accurate results. Operations performed in decimal are limited to 31 decimal digits in the intermediate result.

The subtract operation is performed according to the rules of algebra.

The result of the operation is copied into the difference operand. If this operand is not the same type as that used in performing the operation, the resultant value is converted to its type. If necessary, the resultant value is adjusted to the length of the difference operand, aligned at the assumed decimal point of the difference operand, or both before being copied to it. For fixed-point operation, if significant digits are truncated on the left end of the resultant value, a size exception is signaled.

For the optional round form of the instruction, specification of a floating-point receiver operand is invalid.

For floating-point operations involving a fixed-point receiver field, if nonzero digits would be truncated off the left end of the resultant value, an invalid floating-point conversion exception is signaled.

For a floating-point difference operand, if the exponent of the resultant value is either too large or too small to be represented in the difference field, the floating-point overflow or the floating-point underflow exception is signaled.

If a decimal to binary conversion causes a size exception to be signaled, the binary value contains the correct truncated result only if the decimal value contains 15 or fewer significant nonfractional digits.

Size exceptions can be inhibited.

Limitations: The following are limits that apply to the functions performed by this instruction.

If operands overlap but do not share all of the same bytes, results of operations performed on these operands are not predictable. If overlapped operands share all of the same bytes, the results are predictable when direct addressing is used. If indirect addressing is used (that is, based operands, parameters, strings with variable lengths, and arrays with variable subscripts), the results are not always predictable.

Resultant Conditions: Positive, negative, or zero-The algebraic value of the numeric scalar difference is positive, negative, or zero. Unordered-The value assigned a floating-point difference operand is NaN.

Exceptions

Exception		Operands 1	2	3 [4, 5]	Other
06	Addressing				
	01 Space addressing violation	X	X	X	
	02 Boundary alignment	X	X	X	
	03 Range	X	X	X	
	06 Optimized addressability invalid	X	X	X	
08	Argument/Parameter				
	01 Parameter reference violation	X	X	X	
0C	Computation				
	02 Decimal data		X	X	
	03 Decimal point alignment		X	X	
	06 Floating-point overflow	X			
	07 Floating-point underflow	X			
	09 Floating-point invalid operand		X	X	X
	0A Size	X			
	0C Invalid floating-point conversion	X			
	0D Floating-point inexact result	X			
10	Damage Encountered				
	04 System object damage state	X	X	X	X
	44 Partial system object damage	X	X	X	X
1C	Machine-Dependent Exception				

Exception		Operands 1	2	3 [4, 5]	Other
	03 Machine storage limit exceeded				X
20	Machine Support				
	02 Machine check				X
	03 Function check				X
22	Object Access				
	01 Object not found	X	X	X	
	02 Object destroyed	X	X	X	
	03 Object suspended	X	X	X	
24	Pointer Specification				
	01 Pointer does not exist	X	X	X	
	02 Pointer type invalid	X	X	X	
2A	Program Creation				
	05 Invalid op code extender field				X
	06 Invalid operand type	X	X	X	
	07 Invalid operand attribute	X	X	X	
	08 Invalid operand value range	X	X	X	
	09 Invalid branch target operand				X
	0C Invalid operand ODT reference	X	X	X	
	0D Reserved bits are not zero	X	X	X	X
2C	Program Execution				
	04 Invalid branch target				X
2E	Resource Control Limit				
	01 User Profile storage limit exceeded				X
36	Space Management				
	01 Space Extension/Truncation				X

1.70 Test and Replace Characters (TSTRPLC)

Op Code (Hex)	Operand 1	Operand 2
10A2	Receiver	Replacement

Operand 1: Character variable scalar.

Operand 2: Character scalar.

Description: The character string value represented by operand 1 is tested byte by byte from left to right. Any byte to the left of the leftmost byte which has a value in the range of hex F1 to hex F9 is assigned a byte value equal to the left-most byte of operand 2. Both operands must be character strings. Only the first character of the replacement string is used in the operation.

The operation stops when the first nonzero zoned decimal digit is found or when all characters of the receiver operand have been replaced.

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for this instruction.

Exceptions

Exception		Operands 1	Operands 2	Other
06	Addressing			
	01 Space addressing violation	X	X	
	02 Boundary alignment	X	X	
	03 Range	X	X	
	06 Optimized addressability invalid	X	X	
08	Argument/Parameter			
	01 Parameter reference violation	X	X	
10	Damage Encountered			
	04 System object damage state	X	X	X
	44 Partial system object damage	X	X	X
1C	Machine-Dependent Exception			
	03 Machine storage limit exceeded			X
20	Machine Support			
	02 Machine check			X
	03 Function check			X
22	Object Access			
	01 Object not found	X	X	
	02 Object destroyed	X	X	
	03 Object suspended	X	X	
24	Pointer Specification			
	01 Pointer does not exist	X	X	

Exception		Operands 1	2	Other
	02 Pointer type invalid	X	X	
2A	Program Creation			
	06 Invalid operand type	X	X	
	07 Invalid operand attribute	X	X	
	08 Invalid operand value range	X	X	
	0A Invalid operand length	X	X	
	0C Invalid operand ODT reference	X	X	
	0D Reserved bits are not zero	X	X	X
2E	Resource Control Limit			
	01 User Profile storage limit exceeded			X
36	Space Management			
	01 Space Extension/Truncation			X

1.71 Test Bit in String (TSTBTSB or TSTBTSI)

Op Code (Hex)	Extender	Operand 1	Operand 2	Operand 3 [4]
1C0E	Branch options	Source	Offset	Branch target
180E	Indicator options			Indicator target

Operand 1: Character scalar or numeric scalar.

Operand 2: Binary Scalar.

Operand 3 [*4*]:

- *Branch Form*-Instruction Number or Relative Instruction Number or Branch Point or Instruction Pointer or Instruction Definition Element.
- *Indicator Form*-Numeric Variable Scalar or Character Variable Scalar.

  X'F1' - If the result of the test matches the corresponding indicator option.

  X'F0' - If the result of the test does not match the corresponding indicator option.

Extender: Branch or indicator options.

Either the branch option or the indicator option is required by the instruction. The extender field is required along with one or two branch targets (for the branch option) or one or two indicator operands (for indicator option). See Chapter 1. "Introduction" for the bit encoding of the extender field and the allowed syntax of the branch and indicator operands.

Description: Tests the bit of the source operand as indicated by the offset operand to determine if the bit is set or not set.

Based on the test, the resulting condition is used with the extender field to either

- Transfer control conditionally to the instruction indicated in one of the branch target operands (branch form).
- Assign a value to each of the indicator operands (indicator form).

The source operand can be character or numeric. The leftmost bytes of the source operand are used in the operation. The source operand is interpreted as a bit string with the bits numbered left to right from 0 to the total number of bits in the string minus one.

The source operand cannot be a variable substring.

The offset operand indicates which bit of the source operand is to be tested, with a offset of zero indicating the leftmost bit of the leftmost byte of the source operand.

If an offset value less than zero or beyond the length of the string is specified a "scalar value invalid" exception is raised.

Resultant Conditions: Zero, One: The selected bit of the bit string source operand is either zero or one.

Exceptions

Exception		Operands 1	2	3 [4]	Other
06	Addressing				
	01 Spacing addressing violation	X	X	X	
	02 Boundary alignment violation	X	X	X	
	03 Range	X	X	X	
	06 Optimized addressability invalid	X	X	X	
08	Argument/parameter				
	01 Parameter reference violation	X	X	X	
10	Damage encountered				
	04 System object damage state				X
	44 Partial system object damage				X
1C	Machine-dependent exception				
	03 Machine storage limit exceeded				X
20	Machine support				
	02 Machine check				X
	03 Function check				X
22	Object access				
	02 Object destroyed	X	X	X	
	03 Object suspended	X	X	X	
24	Pointer specification				
	01 Pointer does not exist	X	X	X	
	02 Pointer type invalid	X	X	X	
2A	Program creation				
	06 Invalid operand type	X	X	X	
	07 Invalid operand attribute	X	X	X	
	08 Invalid operand value range	X	X	X	
	0A Invalid operand length	X	X	X	
	0C Invalid operand odt reference	X	X	X	
	0D Reserved bits are not zero	X	X	X	X
2E	Resource control limit				
	01 user profile storage limit exceeded				X
32	Scalar specification				
	01 Scalar type invalid	X	X	X	
	03 Scalar value invalid		X		

Exception		Operands 1	2	3 [4]	Other
36	Space management				
	01 space extension/truncation				X

1.72 Test Bits Under Mask (TSTBUMB or TSTBUMI)

Op Code (Hex)	Extender	Operand 1	Operand 2	Operand 3 [4, 5]
1C2A	Branch Options	Source	Mask	Branch target
182A	Indicator Options			Indicator target

Operand 1: Character variable scalar or numeric variable scalar.

Operand 2: Character scalar or numeric scalar.

Operand 3 [4, 5]

- *Branch Form*-Instruction number, relative instruction number, branch point, or instruction pointer.
- *Indicator Form*-Numeric variable scalar or character variable scalar.

Extender: Branch or indicator options.

Either the branch option or the indicator option is required by the instruction. The extender field is required along with from one to three branch targets (for branch option) or one to three indicator operands (for indicator option). The branch or indicator operands are required for operand 3 and optional for operands 4 and 5. See Chapter 1. "Introduction" for the bit encoding of the extender field and the allowed syntax of the branch and indicator operands.

Description: Selected bits from the leftmost byte of the source operand are tested to determine their bit values.

Based on the test, the resulting condition is used with the extender field to:

- Transfer control conditionally to the instruction indicated in one of the branch target operands (branch form).
- Assign a value to each of the indicator operands (indicator form).

The source and the mask operands can be character or numeric. The leftmost byte of each of the operands is used in the operands. The operands are interpreted as bit strings. The testing is performed bit by bit with only those bits indicated by the mask operand being tested. A 1-bit in the mask operand specifies that the corresponding bit in the source value is to be tested. A 0-bit in the mask operand specifies that the corresponding bit in the source value is to be ignored.

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for this instruction.

Resultant Conditions: The selected bits of the bit string source operand are all zeros, all ones, or mixed ones and zeros. A mask operand of all zeros causes a zero resultant condition.

Exceptions

Exception		Operands 1	2	3 [4, 5]	Other
06	Addressing				
	01 space addressing violation	X	X	X	
	02 boundary alignment	X	X	X	
	03 range	X	X	X	
	06 optimized addressability invalid	X	X	X	
08	Argument/parameter				
	01 parameter reference violation	X	X	X	
10	Damage encountered				
	04 system object damage state	X	X	X	X
	44 partial system object damage	X	X	X	X
1C	Machine-dependent exception				
	03 machine storage limit exceeded				X
20	Machine support				
	02 machine check				X
	03 function check				X
22	Object access				
	01 object not found	X	X	X	
	02 object destroyed	X	X	X	
	03 object suspended	X	X	X	
24	Pointer specification				
	01 pointer does not exist	X	X	X	
	02 pointer type invalid	X	X	X	
2A	Program creation				
	05 invalid op code extender field				X
	06 invalid operand type	X	X	X	
	07 invalid operand attribute	X	X	X	
	08 invalid operand value range	X	X	X	
	09 invalid branch target				X
	0A invalid operand length	X	X		
	0C invalid operand odt reference	X	X	X	
	0D reserved bits are not zero	X	X	X	X
2C	Program execution				
	04 branch target invalid			X	
2E	Resource control limit				
	01 user profile storage limit exceeded				X
36	Space management				

Exception	Operands 1	2	3 [4, 5]	Other
01 space extension/truncation				X

1.73 Translate (XLATE)

Op Code (Hex)	Operand 1	Operand 2	Operand 3	Operand 4
1094	Receiver	Source	Position	Replacement

Operand 1: Character variable scalar.

Operand 2: Character scalar.

Operand 3: Character scalar or null.

Operand 4: Character scalar.

Description: Selected characters in the string value of the source operand are translated into a different encoding and placed in the receiver operand. The characters selected for translation and the character values they are translated to are indicated by entries in the position and replacement strings. All the operands must be character strings. The source and receiver values must be of the same length. The position and replacement operands can differ in length. If operand 3 is null, a 256-character string is used, ranging in value from hex 00 to hex FF (EBCDIC collating sequence).

The operation begins with all the operands left-adjusted and proceeds character by character, from left to right until the character string value of the receiver operand is completed.

Each character of the source operand value is compared with the individual characters in the position operand. If a character of equal value does not exist in the position string, the source character is placed unchanged in the receiver operand. If a character of equal value is found in the position string, the corresponding character in the same relative location within the replacement string is placed in the receiver operand as the source character translated value. If the replacement string is shorter than the position string and a match of a source to position string character occurs for which there is no corresponding replacement character, the source character is placed unchanged in the receiver operand. If the replacement string is longer than the position string, the rightmost excess characters of the replacement string are not used in the translation operation because they have no corresponding position string characters. If a character in the position string is duplicated, the first occurrence (leftmost) is used.

If operands overlap but do not share all of the same bytes, results of operations performed on these operands are not predictable. If overlapped operands share all of the same bytes, the results are predictable when direct addressing is used. If indirect addressing is used (that is, based operands, parameters, strings with variable lengths, and arrays with variable subscripts), the results are not always predictable.

The receiver, source, position, and replacement operands can be variable length substring compound operands.

Substring operand references that allow for a null substring reference (a length value of zero) may be specified for all of the operands on this instruction. The effect of specifying a null substring reference for either the position or replacement operands is that the source operand is copied to the receiver with no

change in value. The effect of specifying a null substring reference for both the receiver and the source operands (they must have the same length) is that no result is set.

Exceptions

Exception		Operands 1	2	3	4	Other
06	Addressing					
	01 Space addressing violation	X	X	X	X	
	02 Boundary alignment	X	X	X	X	
	03 Range	X	X	X	X	
	06 Optimized addressability invalid	X	X	X	X	
08	Argument/Parameter					
	01 Parameter reference violation	X	X	X	X	
0C	Computation					
	08 Length conformance	X	X			
10	Damage Encountered					
	04 System object damage state	X	X	X	X	X
	44 Partial system object damage	X	X	X	X	X
1C	Machine-Dependent Exception					
	03 Machine storage limit exceeded					X
20	Machine Support					
	02 Machine check					X
	03 Function check					X
22	Object Access					
	01 Object not found	X	X	X	X	
	02 Object destroyed	X	X	X	X	
	03 Object suspended	X	X	X	X	
24	Pointer Specification					
	01 Pointer does not exist	X	X	X	X	
	02 Pointer type invalid	X	X	X	X	
2A	Program Creation					
	06 Invalid operand type	X	X	X	X	
	07 Invalid operand attribute	X	X	X	X	
	08 Invalid operand value range	X	X	X	X	
	0A Invalid operand length	X	X	X	X	
	0C Invalid operand ODT reference	X	X	X	X	
	0D Reserved bits are not zero	X	X	X	X	X
2E	Resource Control Limit					
	01 User Profile storage limit exceeded					X

Exception		Operands 1	2	3	4	Other
36	Space Management					
	01 Space Extension/Truncation					X

1.74 Translate with Table (XLATEWT)

Op Code (Hex)	Operand 1	Operand 2	Operand 3
109F	Receiver	Source	Table

Operand 1: Character variable scalar.

Operand 2: Character scalar.

Operand 3: Character scalar.

Description: The source characters are translated under control of the translate table and placed in the receiver. The operation begins with the leftmost character of operand 2 and proceeds character-by-character, left-to-right.

Characters are translated as follows:

- The source character is used as an offset and added to the location of operand 3.
- The character contained in the offset location is the translated character. This character is copied to the receiver in the same relative position as the original character in the source string.

If operand 3 is less than 256 bytes long, the character in the source may specify an offset beyond the end of operand 3. If operand 2 is longer than operand 1, then only the leftmost portion of operand 2, equal to the length of operand 1, is translated. If operand 2 is shorter than operand 1, then only the leftmost portion of operand 1, equal to the length of operand 2, is changed. The remaining portion of operand 1 is unchanged.

If operand 1 overlaps with operand 2 and/or 3, the overlapped operands are updated for every character translated. The operation proceeds from left to right, one character at a time. The following example shows the results of an overlapped operands translate operation. Operands 1, 2, and 3 have the same coincident character string with a value of hex 050403020103.

Hex 050403020103-Initial value

Hex 030403020103-After the 1st character is translated

Hex 030103020103-After the 2nd character is translated

Hex 030102020103-After the 3rd character is translated

Hex 030102020103-After the 4th character is translated

Hex 030102020103-After the 5th character is translated

Hex 030102020102-After the 6th character, the final result

Note that the instruction does not use the length specified for the table operand to constrain access of the bytes addressed by the table operand.

If operand 3 is less than 256 characters long, and a source character specifies an offset beyond the end of operand 3, the result characters are obtained from

byte locations in the space following operand 3. If that portion of the space is not currently allocated, a space addressing exception is signaled. If operand 3 is a constant with a length less than 256, source characters specifying offsets greater than or equal to the length of the constant are translated into unpredictable characters.

All of the operands support variable length substring compound scalars.

Substring operand references that allow for a null substring reference (a length value of zero) may be specified for all of the operands on this instruction. Specifying a null substring reference for the table operand does not affect the operation of the instruction. In this case, the bytes addressed by the table operand are still accessed as described above. This is due to the definition of the function of this instruction which does not use the length specified for the table operand to constrain access of the bytes addressed by the table operand. The effect of specifying a null substring reference for either or both of the receiver and the source operands is that no result is set.

Exceptions

Exception		Operands 1	2	3	Other
06	Addressing				
	01 Space addressing violation	X	X	X	
	02 Boundary alignment	X	X	X	
	03 Range	X	X	X	
	06 Optimized addressability invalid	X	X	X	
08	Argument/Parameter				
	01 Parameter reference violation	X	X	X	
10	Damage Encountered				
	44 Partial system object damage				X
1C	Machine-Dependent Exception				
	03 Machine storage limit exceeded				X
20	Machine Support				
	02 Machine check				X
	03 Function check				X
22	Object Access				
	02 Object destroyed	X	X	X	
	03 Object suspended	X	X	X	
24	Pointer Specification				
	01 Pointer does not exist	X	X	X	
	02 Pointer type invalid	X	X	X	
2A	Program Creation				
	06 Invalid operand type	X	X	X	
	07 Invalid operand attribute	X	X	X	

Exception		Operands 1	2	3	Other
	08 Invalid operand value range	X	X	X	
	0A Invalid operand length	X	X		
	0C Invalid operand ODT reference	X	X	X	
	0D Reserved bits are not zero	X	X	X	X
2E	Resource Control Limit				
	01 User Profile storage limit exceeded				X
32	Scalar Specification				
	01 Scalar type invalid	X	X	X	
36	Space Management				
	01 Space Extension/Truncation				X

1.75 Trim Length (TRIML)

Op Code (Hex)	Operand 1	Operand 2	Operand 3
10A7	Receiver length	Source string	Trim character

Operand 1: Numeric variable scalar.

Operand 2: Character scalar.

Operand 3: Character(1) scalar.

Description: The operation determines the resultant length of operand 2 after the character specified by operand 3 has been trimmed from the end of operand 2. The resulting length is stored in operand 1. Operand 2 is trimmed from the end as follows: if the rightmost (last) character of operand 2 is equal to the character specified by operand 3, the length of the trimmed operand 2 string is reduced by 1. This operation continues until the rightmost character is no longer equal to operand 3 or the trimmed length is zero. If operand 3 is longer than one character, only the first (leftmost) character is used as the trim character.

Operands 2 and 3 are not changed by this instruction. Operand 2 or 3 may be variable length substring compound scalars.

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for this instruction.

Exceptions

Exception		Operands 1	2	3	Other
06	Addressing				
	01 Space addressing violation	X	X	X	
	02 Boundary alignment	X	X	X	
	03 Range	X	X	X	
	06 Optimized addressability invalid	X	X	X	
08	Argument/Parameter				
	01 Parameter reference violation	X	X	X	
0C	Computation				
	0A Size	X			
10	Damage Encountered				
	44 Partial system object damage				X
1C	Machine-Dependent Exception				
	03 Machine storage limit exceeded				X
20	Machine Support				
	02 Machine check				X
	03 Function check				X

Exception		Operands 1	2	3	Other
22	Object Access				
	01 Object not found	X	X	X	
	02 Object destroyed	X	X	X	
	03 Object suspended	X	X	X	
24	Pointer Specification				
	01 Pointer does not exist	X	X	X	
	02 Pointer type invalid	X	X	X	
2A	Program Creation				
	06 Invalid operand type	X	X	X	
	07 Invalid operand attribute	X	X	X	
	08 Invalid operand value range	X	X	X	
	09 Invalid branch target operand		X	X	
	0A Invalid operand length	X	X	X	
	0C Invalid operand ODT reference	X	X	X	
	0D Reserved bits are not zero	X	X	X	X
2E	Resource Control Limit				
	01 User Profile storage limit exceeded				X
32	Scalar Specification				
	01 Scalar type invalid	X	X	X	
36	Space Management				
	01 Space Extension/Truncation				X

1.76 Verify (VERIFY)

Op Code (Hex)	Operand 1	Operand 2	Operand 3
10D7	Receiver	Source	Class

Operand 1: Binary variable scalar or binary array.

Operand 2: Character scalar.

Operand 3: Character scalar.

Optional Forms

Mnemonic	Op Code (Hex)	Form Type
VERIFYI	18D7	Indicator
VERIFYB	1CD7	Branch

Extender: Branch or indicator options.

If the branch or indicator option is specified in the op code, the extender field must be present along with one or two branch targets (for branch options) or one or two indicator operands (for indicator options). The branch or indicator operands immediately follow the last operand listed above. See Chapter 1. "Introduction" for the encoding of the extender field and the allowed syntax of the branch and indicator operands.

Description: Each character of the source operand character string value is checked to verify that it is among the valid characters indicated in the class operand.

The operation begins at the left end of the source string and continues character by character, from left to right. Each character of the source value is compared with the characters of the class operand. If a match for the source character exists in the class string, the next source character is verified. If a match for the source character does not exist in the class string, the binary value for the relative location of the character within the source string is placed in the receiver operand.

If the receiver operand is a scalar, only the first occurrence of an invalid character is noted. If the receiver operand is an array, as many occurrences as there are elements in the array are noted.

The operation continues until no more occurrences of invalid characters can be noted or until the end of the source string is encountered. When the second condition occurs, the current receiver value is set to 0. If the receiver operand is an array, all its remaining entries are set to 0's.

The source and class operands can be variable length substring compound operands.

Substring operand references that allow for a null substring reference (a length value of zero) may be specified for operands 2 and 3. The effect of specifying a null substring reference for the class operand when a nonnull string reference is specified for the source is that all of the characters of the source are considered

invalid. In this case, the receiver is accordingly set with the offset(s) to the bytes of the source, and the instruction's resultant condition is positive. The effect of specifying a null substring reference for the source operand (no characters to verify) is that the receiver is set to zero and the instruction's resultant condition is zero regardless of what is specified for the class operand.

Resultant Conditions: The numeric value(s) of the receiver is either 0 or positive. When the receiver operand is an array, the resultant condition is 0 if all elements are 0.

Exceptions

Exception		Operands 1	2	3	Other
06	Addressing				
	01 space addressing violation	X	X	X	
	02 boundary alignment	X	X	X	
	03 range	X	X	X	
	06 optimized addressability invalid	X	X	X	
08	Argument/parameter				
	01 parameter reference violation	X	X	X	
10	Damage encountered				
	04 system object damage	X	X	X	X
	44 partial system object damage	X	X	X	X
1C	Machine-dependent exception				
	03 machine storage limit exceeded				X
20	Machine support				
	02 machine check				X
	03 function check				X
22	Object access				
	01 object not found	X	X	X	
	02 object destroyed	X	X	X	
	03 object suspended	X	X	X	
24	Pointer specification				
	01 pointer does not exist	X	X	X	
	02 pointer type invalid	X	X	X	
2A	Program creation				
	05 invalid op code extender field				X
	06 invalid operand type	X	X	X	
	07 invalid operand attribute	X	X	X	
	08 invalid operand value range	X	X	X	
	09 invalid branch target operand				X
	0A invalid operand length				X

Exception		Operands 1	2	3	Other
	0C invalid operand odt reference	X	X	X	
	0D reserved bits are not zero	X	X	X	X
2C	Program execution				
	04 branch target invalid				X
2E	Resource control limit				
	01 user profile storage limit exceeded				X
36	Space management				
	01 space extension/truncation				X

Chapter 2. Pointer/Name Resolution Addressing Instructions

This chapter describes the instructions used for pointer and name resolution functions. These instructions are in alphabetic order. See Appendix A, "Instruction Summary," for an alphabetic summary of all the instructions.

2.1 Compare Pointer for Object Addressability (CMPPTRAB or CMPPTRAI)

Op Code (Hex)	Extender	Operand 1	Operand 2	Operand 3 [4]
1CD2	Branch options	Compare operand 1	Compare operand 2	Branch target
18D2	Indicator options			Indicator target

Operand 1: Data pointer, space pointer, system pointer, or instruction pointer.

Operand 2: Data pointer, space pointer, system pointer, or instruction pointer.

Operand 3 [4]:

- *Branch Form* — Instruction number, relative instruction number, branch point, or instruction pointer.
- *Indicator Form* — Numeric variable scalar or character variable scalar.

Extender: Branch or indicator options.

Either the branch option or the indicator option is required by the instruction.

The extender field is required along with one or two branch targets (for branch option) or one or two indicator operands (for indicator option). The branch or indicator operands are required for operand 3 and optional for operand 4. See Chapter 1. "Introduction" for the bit encoding of the extender field and the allowed syntax of the branch and indicator operands.

Description: The object addressed by operand 1 is compared with the object addressed by operand 2 to determine if both operands are addressing the same object. Based on the comparison, the resulting condition is used with the extender to transfer control (branch form) or to assign a value to each of the indicator operands (indicator form).

If operand 1 is a data pointer, a space pointer, or a system pointer, operand 2 may be any pointer type except for instruction pointer in any combination. An equal condition occurs if the pointers are addressing the same object. For space pointers and data pointers, only the space they are addressing is considered in the comparison. That is, the space offset portion of the pointer is ignored.

For system pointer compare operands, an equal condition occurs if the system pointer is compared with a space pointer or data pointer that addresses the

space that is associated with the object that is addressed by the system pointer. For example, a space pointer that addresses a byte in a space associated with a system object compares equal with a system pointer that addresses the system object.

For instruction pointer comparisons, both operands must be instruction pointers; otherwise, an invalid pointer type exception is signaled. An equal condition occurs when both instruction pointers are addressing the same instruction in the same program. A not equal condition occurs if the instruction pointers are not addressing the same instruction in the same program.

A pointer does not exist exception is signaled if a pointer does not exist in either of the operands.

Resultant Conditions: Equal, not equal.

Authorization Required

- Retrieve
  - Contexts referenced for address resolution

Lock Enforcement

- Materialize
  - Contexts referenced for address resolution

Exceptions

Exception		Operands 1	2	3	4	Other
06	Addressing					
	01 Space addressing violation	X	X	X	X	
	02 Boundary alignment	X	X	X	X	
	03 Range	X	X	X	X	
	06 Optimized addressability invalid	X	X	X	X	
08	Argument/parameter					
	01 Parameter reference violation	X	X	X	X	
0A	Authorization					
	01 Unauthorized for operation	X	X			
10	Damage encountered					
	04 System object damage state	X	X	X	X	X
	05 authority verification terminated due to damaged object					X
	44 Partial system object damage	X	X	X	X	X
1A	Lock state					
	01 Invalid lock state	X	X			
1C	Machine-dependent exception					
	03 Machine storage limit exceeded					X

Exception		Operands 1	2	3	4	Other
20	Machine support					
	02 Machine check					X
	03 Function check					X
22	Object access					
	01 Object not found	X	X	X	X	
	02 Object destroyed	X	X	X	X	
	03 Object suspended	X	X	X	X	
	07 authority verification terminated due to destroyed object					X
24	Pointer specification					
	01 Pointer does not exist	X	X	X	X	
	02 Pointer type invalid	X	X	X	X	
2A	Program creation					
	05 Invalid op code extender field					X
	06 Invalid operand type	X	X	X	X	
	07 Invalid operand attribute	X	X	X	X	
	08 Invalid operand value range	X	X	X	X	
	09 Invalid branch target operand			X	X	
	0A Invalid operand length			X	X	
	0C Invalid operand odt reference	X	X	X	X	
	0D Reserved bits are not zero	X	X	X	X	X
2E	Resource control limit					
	01 user profile storage limit exceeded					X
36	Space management					
	01 space extension/truncation					X

2.2 Compare Pointer Type (CMPPTRTB or CMPPTRTI)

Op Code (Hex)	Extender	Operand 1	Operand 2	Operand 3 [4]
1CE2	Branch options	Compare operand 1	Compare operand 2	Branch target
18E2	Indicator options			Indicator target

Operand 1: Data pointer, space pointer, system pointer, or instruction pointer.

Operand 2: Character(1) scalar or null.

Operand 3 [4]:

- *Branch Form* — Instruction number, relative instruction number, branch point, or instruction pointer.
- *Indicator Form* — Numeric variable scalar or character variable scalar.

Extender: Branch or indicator options.

Either the branch option or the indicator option is required by the instruction.

The extender field is required along with one or two branch targets (for branch option) or one or two indicator operands (for indicator option). The branch or indicator operands are required for operand 3 and optional for operand 4. See Chapter 1. "Introduction" for the bit encoding of the extender field and the allowed syntax of the branch and indicator operands.

Description: The instruction compares the pointer type currently in operand 1 with the character scalar identified by operand 2. Based on the comparison, the resulting condition is used with the extender to transfer control (branch form) or to assign a value to each of the indicator operands (indicator form).

Operand 1 can specify a space pointer machine object only when operand 2 is null.

If operand 2 is null or if operand 2 specifies a comparison value of hex 00, an equal condition occurs if a pointer does not exist in the storage area identified by operand 1.

Following are the allowable values for operand 2:

Hex 00 — A pointer does not exist at this location

Hex 01 — System pointer

Hex 02 — Space pointer

Hex 03 — Data pointer

Hex 04 — Instruction pointer

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for this instruction.

Resultant Conditions: Equal, not equal.

Authorization Required

- Retrieve
  - Contexts referenced for address resolution

Lock Enforcement

- Materialize
  - Contexts referenced for address resolution

Exceptions

Exception		Operands 1	2	3	4	Other
06	Addressing					
	01 Space addressing violation	X	X	X	X	
	02 Boundary alignment	X	X	X	X	
	03 Range	X	X	X	X	
	06 Optimized addressability invalid	X	X	X	X	
08	Argument/parameter					
	01 Parameter reference violation	X	X	X	X	
0A	Authorization					
	01 Unauthorized for operation	X				
10	Damage encountered					
	04 System object damage state	X	X	X	X	X
	05 authority verification terminated due to damaged object					X
	44 Partial system object damage	X	X	X	X	X
1A	Lock state					
	01 Invalid lock state	X				
1C	Machine-dependent exception					
	03 Machine storage limit exceeded					X
20	Machine support					
	02 Machine check					X
	03 Function check					X
22	Object access					
	01 Object not found	X	X	X	X	
	02 Object destroyed	X	X	X	X	
	03 Object suspended	X	X	X	X	
	07 authority verification terminated due to destroyed object					X

Exception		Operands 1	2	3	4	Other
24	Pointer specification					
	01 Pointer does not exist	X	X	X	X	
	02 Pointer type invalid	X	X	X	X	
2A	Program creation					
	05 Invalid op code extender operand					X
	06 Invalid operand type	X	X	X	X	
	07 Invalid operand attribute	X	X	X	X	
	08 Invalid operand value range	X	X	X	X	
	09 Invalid branch target operand			X	X	
	0A Invalid operand length		X	X	X	
	0C Invalid operand odt reference	X	X	X	X	
	0D Reserved bits are not zero	X	X	X	X	X
2E	Resource control limit					
	01 user profile storage limit exceeded					X
32	Scalar specification					
	03 Scalar value invalid			X		
36	Space management					
	01 space extension/truncation					X

2.3 Copy Bytes with Pointers (CPYBWP)

Op Code (Hex)	Operand 1	Operand 2
0132	Receiver	Source

Operand 1: Character variable scalar, space pointer, data pointer, system pointer, or instruction pointer.

Operand 2: Character variable scalar, space pointer data object, data pointer, system pointer, instruction pointer, or null.

Description: If either operand is a character variable scalar, it can have a length as great as 16776191 bytes.

This instruction copies either the pointer value or the byte string specified for the source operand into the receiver operand depending upon whether or not a space pointer machine object is specified as one of the operands.

Operations involving space pointer machine objects perform a pointer value copy operation for only space pointer values or the pointer does not exist state. Due to this, a space pointer machine object may only be specified as an operand in conjunction with another pointer or a null second operand. The pointer does not exist state is copied from the source to the receiver pointer without signaling the pointer does not exist exception. Source pointer data objects must either be not set or contain a space pointer value when being copied into a receiver space pointer machine object. Receiver pointer data objects will be set with either the system default pointer does not exist value or the space pointer value from a source space pointer machine object.

Normal pointer alignment checking is performed on a pointer data object specified as an operand in conjunction with a space pointer machine object.

Operations not involving space pointer machine objects, those involving just data objects as operands, perform a byte string copy of the data for the specified operands.

The value of the byte string specified by operand 2 is copied to the byte string specified by operand 1 (no padding done).

The byte string identified by operand 2 can contain the storage forms of both scalars and pointers. Normal pointer alignment checking is not done.

When the OVRPGATR instruction is not used to override CPYBWP, the only alignment requirement is that the space addressability alignment of the two operands must be to the same position relative to a 16-byte multiple boundary. A boundary alignment exception is signaled if the alignment is incorrect. The pointer attributes of any complete pointers in the source are preserved if they can be completely copied into the receiver. Partial pointer storage forms are copied into the receiver as scalar data. Scalars in the source are copied to the receiver as scalars.

When the OVRPGATR instruction is used to override this instruction the alignment requirement is removed. If the space addressability alignment of the two

operands is the same relative to 16-byte multiple boundary then this instruction will work the same as stated above. If the space addressability alignment is different then this instruction will work like a CPYBLA and the pointer attributes of any complete pointers in the source are not preserved in the receiver.

If a pointer data object operand contains a data pointer value upon execution of the instruction, the pointer storage form is copied rather than than the scalar described by the data pointer value. The character variable scalar reference allowed on either operand cannot be described through a data pointer value.

The length of the operation is equal to the length of the shorter of the two operands. The copying begins with the two operands left-adjusted and proceeds until completion of the shorter operand.

Operand 1 can specify a space pointer machine object only when operand 2 is null.

If operand 2 is null, operand 1 must define a pointer reference; otherwise, an exception is signaled. When operand 2 is null, the byte string identified by operand 1 is set to the system default pointer does not exist value.

Exceptions

Exception		Operands 1	Operands 2	Other
06	Addressing			
	01 Space addressing violation	X	X	
	02 Boundary alignment	X	X	
	03 Range	X	X	
	06 Optimized addressability invalid	X	X	
08	Argument/parameter			
	01 Parameter reference violation	X	X	
10	Damage encountered			
	04 System object damage state	X	X	X
	44 Partial system object damage	X	X	X
1C	Machine-dependent exception			
	03 machine storage limit exceeded			X
20	Machine support			
	02 Machine check			X
	03 Function check			X
22	Object access			
	01 Object not found	X	X	
	02 Object destroyed	X	X	
	03 Object suspended	X	X	
24	Pointer specification			
	01 Pointer does not exist	X	X	

Exception		Operands 1	2	Other
	02 Pointer type invalid	X	X	
2A	Program creation			
	06 Invalid operand type	X	X	
	07 Invalid operand attribute		X	
	08 Invalid operand value range	X	X	
	0A Invalid operand length		X	
	0C Invalid operand odt reference	X	X	
	0D Reserved bits are not zero	X	X	X
2E	Resource control limit			
	01 user profile storage limit exceeded			X
36	Space management			
	01 space extension/truncation			X

2.4 Resolve Data Pointer (RSLVDP)

Op Code (Hex)	Operand 1	Operand 2	Operand 3
0163	Pointer for addressability to data object	Data object identification	Program

Operand 1: Data pointer.

Operand 2: Character(32) scalar (fixed-length) or null.

Operand 3: System pointer or null.

Description: A data pointer with addressability to and the attributes of an external scalar data element is returned in the storage area identified by operand 1. The following describes the instruction's function when operand 2 is null:

- If operand 1 does not contain a data pointer, an exception is signaled.
- If the data pointer specified by operand 1 is not resolved and has an initial value declaration, the instruction resolves the data pointer to the external scalar that the initial value describes. The initial value defines the external scalar to be located and, optionally, defines the program in which it is to be located. If the program name is specified in the initial value, only that program's activation entry is searched for the external scalar. If no program is specified, programs associated with the activation entries in the process static storage area are searched in reverse order of the activation entries, and operand 3 is ignored.
- If the data pointer is currently resolved and defines an existing scalar, the instruction causes no operation, and no exception is signaled.

The following describes the instruction's function when operand 2 is not null:

- A data pointer that is resolved to the external scalar identified by operand 2 is returned in operand 1. Operand 2 is a 32-byte value that provides the name of the external scalar to be located.
- Operand 3 specifies a system pointer that identifies the program whose activation is to be searched for the external scalar definition. If operand 3 is null, the instruction searches all activations in the process, starting with the most recent activation and continuing to the oldest. The activation under which the instruction is issued also participates in the search. If operand 3 is not null, the instruction searches the activation of the program addressed by the system pointer.

If the external scalar is not located, the object not found exception is signaled. If an unresolved system pointer is encountered when the program searches the activation entries, the pointer not resolved exception is signaled. If the PSSA chain being modified bit is on when this instruction is executed, a stack control invalid exception is signaled.

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for this instruction.

Authorization Required

- Retrieve
  - Contexts referenced for address resolution

Lock Enforcement

- Materialize
  - Contexts referenced for address resolution

Exceptions

Exception		Operands 1	2	3	Other
06	Addressing				
	01 Space addressing violation	X	X	X	
	02 Boundary alignment	X	X	X	
	03 Range	X	X	X	
	04 External data object not found	X			
	06 Optimized addressability invalid	X	X	X	
08	Argument/parameter				
	01 Parameter reference violation	X	X	X	
0A	Authorization				
	01 Unauthorized for operation	X		X	
10	Damage encountered				
	04 System object damage state	X	X	X	X
	05 authority verification terminated due to damaged object				X
	44 Partial system object damage	X	X	X	X
1A	Lock state				
	01 Invalid lock state			X	
1C	Machine-dependent exception				
	03 Machine storage limit exceeded				X
20	Machine support				
	02 Machine check				X
	03 Function check				X
22	Object access				
	01 Object not found	X	X	X	
	02 Object destroyed	X	X	X	
	03 Object suspended	X	X	X	
	04 Pointer not resolved				X
	07 authority verification terminated due to destroyed object				X
24	Pointer specification				

Exception		Operands 1	2	3	Other
	01 Pointer does not exist	X	X	X	
	02 Pointer type invalid	X	X	X	
	04 Pointer not resolved				X
2A	Program creation				
	06 Invalid operand type	X	X	X	
	07 Invalid operand attribute	X	X	X	
	08 Invalid operand value range	X	X	X	
	0A Invalid operand odt reference		X		
	0C Invalid operand odt reference	X	X	X	
	0D Reserved bits are not zero	X	X	X	X
2C	Program execution				
	03 Stack control invalid				X
2E	Resource control limit				
	01 user profile storage limit exceeded				X
32	Scalar specification				
	01 Scalar type invalid	X	X	X	
	02 Scalar attributes invalid		X		
	03 Scalar value invalid		X		
36	Space management				
	01 space extension/truncation				X

2.5 Resolve System Pointer (RSLVSP)

Op Code (Hex)	Operand 1	Operand 2	Operand 3	Operand 4
0164	Pointer for address-ability to object	Object identifica-tion and required authori-zation	Context through which objects is to be located	Authority to be set[1]

Operand 1: System pointer.

Operand 2: Character(34) scalar (fixed-length) or null.

Operand 3: System pointer or null.

Operand 4: Character(2) scalar (fixed-length) or null.

Description: This instruction locates an object identified by a symbolic address and stores the object's addressability and authority[1] in a system pointer. A resolved system pointer is returned in operand 1 with addressability to a system object and the requested authority currently available to the process for the object.

Note: The ownership flag is never set in the system pointer.

Operand 2 specifies the symbolic identification of the object to be located. Operand 3 identifies the context to be searched in order to locate the object. Operand 4 identifies the authority states to be set in the pointer. First, the instruction locates an object based on operands 2 and 3. Then, the instruction sets the appropriate authority states in the system pointer.

The following describes the instruction's function when operand 2 is null:

- If operand 1 does not contain a system pointer, an exception is signaled.
- If the system pointer specified by operand 1 is not resolved but has an initial value declaration, the instruction resolves the system pointer to the object that the initial value describes. The initial value defines the following:
  - Object to be located (by type, subtype, and name)
  - Context to be searched to locate the object (optional)
  - Minimum authority required for the object

  If a context is specified, only that context is referenced to locate the object, and operand 3 is ignored. If no context is specified, the context(s) located by the process name resolution list is used to locate the object, and operand 3 is ignored. If the object is of a type that can only be addressed through the machine context, then only the machine context is searched, and the context (if any) identified in the initial value or identified in operand 3 is ignored.

---

1 Programs executing in user-domain may not assign authority in the resulting system pointer. The value in operand 4 is ignored and no exception is raised.

If the minimum required authority in the initial value is not set (binary 0), the instruction resolves the operand 1 system pointer to the first object encountered with the designated type code, subtype code, and object name without regard to the authorization available to the process for the object. If one or more authorization (or ownership) states are required (signified by binary 1's), the context(s) is searched until an object is encountered with the designated type, subtype, and name and for which the process currently has all required authorization states.

- If the system pointer specified by operand 1 is currently resolved to address an existing object, the instruction does not modify the addressability contained in the pointer and causes only the authority attribute in the pointer to be modified based on operand 4.

If operand 2 is not null, the operand 1 system pointer is resolved to the object identified by operand 2 in the context(s) specified by operand 3. The format of operand 2 is as follows:

• Object specification	Char(32)
— Type code	Char(1)
— Subtype code	Char(1)
— Object name	Char(30)
• Required authorization (1 = required)	Char(2)
— Object control	Bit 0
— Object management	Bit 1
— Authorized pointer	Bit 2
— Space authority	Bit 3
— Retrieve	Bit 4
— Insert	Bit 5
— Delete	Bit 6
— Update	Bit 7
— Ownership	Bit 8
— Excluded	Bit 9
— Authority List Management	Bit 10
— Reserved (binary zero)	Bit 11-15

The allowed type codes are as follows:

Hex 01 = Access group
Hex 02 = Program
Hex 04 = Context
Hex 06 = Byte string space
Hex 07 = Journal space
Hex 08 = User profile
Hex 09 = Journal port
Hex 0A = Queue
Hex 0B = Data space
Hex 0C = Data space index

Hex 0D = Cursor
Hex 0E = Index
Hex 0F = Commit block
Hex 10 = Logical unit description
Hex 11 = Network description
Hex 12 = Controller description
Hex 13 = Dump space
Hex 14 = Class of Service Description
Hex 15 = Mode Description
Hex 19 = Space
Hex 1A = Process control space
Hex 1B = Authorization List
Hex 1C = Dictionary

All other codes are reserved. If other codes are specified, they cause a scalar value invalid exception to be signaled.

Operand 3 identifies the context in which to locate the object identified by operand 2. If operand 3 is null, then the contexts identified in the process name resolution list are searched in the order in which they appear in the list. If operand 3 is not null, the system pointer specified must address a context, and only this context is used to locate the object. If the object is of a type that can only be addressed through the machine context, then only the machine context is searched, and operand 3 and the process name resolution list are ignored.

If the required authorization field in operand 2 is not set (binary 0's), the instruction resolves the operand 1 system pointer to the first object encountered with the designated type code, subtype code, and object name without regard to the authorization currently available to the process. If one or more authorization (or ownership) states are required (signified by binary 1's), the context is searched until an object is encountered with the designated type, subtype, name, and the user profiles governing the instruction's execution that have all the required authorization states.

Once addressability has been set in the pointer, operand 4 is used to determine which, if any, of the object authority states is to be set into the pointer. Only the object authority states correlating with bits 0 through 7, that is, object control through update, can be set into the pointer. This restriction applies whether the authority mask controlling which authorities to set in the pointer comes from operand 4, operand 2, or the initial value for the system pointer.

If operand 4 is null, the object authority states required to locate the object are set in the pointer. This required object authority is as specified in operand 2 or in the initial value for operand 1 if operand 2 is null. If the process does not currently have authorized pointer authority for the object, no authority is stored in the system pointer, and no exception is signaled.

If operands 2 and 4 are null and operand 1 is a resolved system pointer, the authority states in the pointer are not modified.

If operand 4 is not null, it specifies the object authority states to be set in the resolved system pointer. The format of operand 4 is as follows:

- Requested authorization (1 = set authority) Char(2)
  - Object control Bit 0

- Object management — Bit 1
- Authorized pointer — Bit 2
- Space authority — Bit 2
- Retrieve — Bit 4
- Insert — Bit 5
- Delete — Bit 6
- Update — Bit 7
- Reserved (binary 0) — Bits 8-15

The authority states set in the resolved system pointer are based on the following:

- The authority already stored in the pointer can be increased only when the process has authorized pointer authority to the referenced object. If this authority is not available and the pointer was resolved by this instruction, the authority in the operand 1 system pointer is set to the not set state, and no exception is signaled. If operand 2 is null, if operand 1 is a resolved system pointer containing authority, and if authorized pointer authority is not available to the process, additional authorities cannot be stored in the pointer.
- If the process does not currently have all the authority states requested in operand 4, only the requested and available states are set in the pointer, and no exception is signaled.
- Note that the authority stored in the operand 1 system pointer is a source of authority applies to this instruction when operand 2 is null and operand 1 is a resolved system pointer with authority stored in it.

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for this instruction.

Authorization Required

- Retrieve
  - Contexts referenced for address resolution (including operand 3)

Lock Enforcement

- Materialization
  - Contexts referenced for address resolution (including operand 3)

Exceptions

Exception		Operands 1	2	3	4	Other
06	Addressing					
	01 Space addressing violation	X	X	X	X	
	02 Boundary alignment	X	X	X	X	
	03 Range	X	X	X	X	
	06 Optimized addressability invalid	X	X	X	X	
08	Argument/parameter					
	01 Parameter reference violation	X	X	X	X	

Exception		Operands 1	2	3	4	Other
0A	Authorization					
	01 Unauthorized for operation	X		X		
10	Damage encountered					
	02 Machine context damage state					X
	04 System object damage state	X	X	X	X	X
	05 authority verification terminated due to damaged object					X
	44 Partial system object damage	X	X	X	X	X
1A	Lock state					
	01 Invalid lock state	X		X		
20	Machine support					
	02 Machine check					X
	03 Function check					X
22	Object access					
	01 Object not found	X	X	X	X	
	02 Object destroyed	X	X	X	X	
	03 Object suspended	X	X	X	X	
	07 authority verification terminated due to destroyed object					X
24	Pointer specification					
	01 Pointer does not exist	X	X	X	X	
	02 Pointer type invalid	X	X	X	X	
	04 Pointer not resolved					X
2A	Program creation					
	06 Invalid operand type	X	X	X	X	
	07 Invalid operand attribute	X	X	X	X	
	08 Invalid operand value range	X	X	X	X	
	0A Invalid operand length		X		X	
	0C Invalid operand odt reference	X	X	X	X	
	0D Reserved bits are not zero	X	X	X	X	X
2E	Resource control limit					
	01 user profile storage limit exceeded					X
32	Scalar specification					
	02 Scalar attributes invalid		X		X	
	03 Scalar value invalid		X		X	
36	Space management					
	01 space extension/truncation					X

Chapter 3. Space Object Addressing Instructions

This chapter describes the instructions used for space object addressing. These instructions are in alphabetic order. For an alphabetic summary of all the instructions, see Appendix A, "Instruction Summary"

3.1 Add Space Pointer (ADDSPP)

Op Code (Hex)	Operand 1	Operand 2	Operand 3
0083	Receiver Pointer	Source Pointer	Increment

Operand 1: Space pointer.

Operand 2: Space pointer.

Operand 3: Binary scalar.

Description: This instruction adds a signed or unsigned binary value to the offset of a space pointer. The value of the binary scalar represented by operand 3 is added to the space address contained in the space pointer specified by operand 2, and the result is stored in the space pointer identified by operand 1. Operand 3 can have a positive or negative value. The space object that the pointer is addressing is not changed by the instruction.

Operand 2 must contain a space pointer when the execution of the instruction is initiated; otherwise, an invalid pointer type exception is signaled. When the addressability in a space pointer is modified, the instruction signals a space addressing exception only when the space address to be stored in the pointer has a negative offset value or when the offset addresses beyond the largest space allocatable in the object. This maximum offset value is dependent on the size and packaging of the object containing the space and is independent of the actual size of the space allocated. If the exception is signaled by this instruction for this reason, the pointer is not modified by the instruction. Attempts to use a pointer whose offset value lies between the currently allocated extent of the space and the maximum allocatable extent of the space cause the space addressing exception to be signaled.

The object destroyed exception, optimized addressability invalid exception, parameter reference violation exception, and pointer does not exist exception are not signaled when operand 1 and operand 2 are space pointer machine objects. This occurs when operand 2 contains an internal machine value that indicates one of these error conditions exists. If the corresponding exception is not signaled, operand 1 is set with an internal machine value that preserves the exception condition that existed for operand 2. The appropriate exception condition will be signaled for either pointer when a subsequent attempt is made to reference the space data that the pointer addresses.

Exceptions

Exception		Operands 1	2	3[4-6]	Other
06	Addressing				
	01 space addressing violation	X	X	X	
	02 boundary alignment	X	X	X	
	03 range	X	X	X	
	06 optimized addressability invalid	X	X	X	
08	Argument/parameter				
	01 parameter reference violation	X	X	X	
10	Damage encountered				
	04 system object damage state	X	X	X	X
	44 partial system object damage	X	X	X	X
1C	Machine-dependent exception				
	03 machine storage limit exceeded				X
20	Machine support				
	02 machine check				X
	03 function check				X
22	Object access				
	01 object not found	X	X	X	
	02 object destroyed	X	X	X	
	03 object suspended	X	X	X	
24	Pointer specification				
	01 pointer does not exist	X	X	X	
	02 pointer type invalid	X	X	X	
2A	Program creation				
	06 invalid operand type	X	X	X	
	07 invalid operand attribute	X	X	X	
	08 invalid operand value range	X	X	X	
	0C invalid operand odt reference	X	X	X	
	0D reserved bits are not zero	X	X	X	X
2E	Resource control limit				
	01 user profile storage limit exceeded				X
36	Space management				
	01 space extension/truncation				X

3.2 Compare Pointer for Space Addressability (CMPPSPADB or CMPPSPADI)

Op Code (Hex)	Extender	Operand 1	Operand 2	Operand 3 [4-6]
1CE6	Branch Operations	Compare Operand 1	Compare Operand 2	Branch target
18E6	Indicator options			Indicator target

Operand 1: Space pointer or data pointer.

Operand 2: Numeric variable scalar, character variable scalar, numeric variable array, character variable array, space pointer, or data pointer.

Operand 3 [*4-6*]:

- *Branch Form*-Instruction number, relative instruction number, branch point, or instruction pointer.
- *Indicator Form*-Numeric variable scalar or character variable scalar.

Extender: Branch or indicator options.

Either the branch option or the indicator option is required by the instruction.

The extender field is required along with from one to four branch targets (for branch option) or one to four indicator operands (for indicator option). The branch or indicator operands are required for operand 3 and optional for operands 4-6. See Chapter 1. *Introduction* for the bit encoding of the extender field and the allowed syntax of the branch and indicator operands.

Description: The space addressability contained in the pointer specified by operand 1 is compared with the space addressability defined by operand 2.

The value of the operand 1 pointer is compared based on the following:

- If operand 2 is a scalar data object (element or array), the space addressability of that data object is compared with the space addressability contained in the operand 1 pointer.
- If operand 2 is a pointer, it must be a space pointer or data pointer, and the space addressability contained in the pointer is compared with the space addressability contained in the operand 1 pointer.

Based on the results of the comparison, the resulting condition is used with the extender to transfer control (branch form) or to assign a value to each of the indicator operands (indicator form). If the operands are not in the same space, the resultant condition is unequal. If the operands are in the same space and the offset into the space of operand 1 is larger or smaller than the offset of operand 2, the resultant condition is high or low, respectively. An equal condition occurs only if the operands are in the same space at the same offset. Therefore, the resultant conditions (high, low, equal, and unequal) are mutually exclusive. Consequently, if you specify that an action be taken upon the nonexistence of a condition, this results in the action being taken upon the occurrence

of any of the other three possible conditions. For example, a branch not high would result in the branch being taken on a low, equal, or unequal condition.

The object destroyed exception, optimized addressability invalid exception, parameter reference violation exception, and pointer does not exist exception are not signaled when operand 1 or operand 2 is a space pointer machine object or when operand 2 is a scalar based on a space pointer machine object. This occurs when the space pointer machine object contains an internal machine value that indicates one of these error conditions exists. If the corresponding exception is not signaled, the resulting condition of the comparison operation is not defined other than that it will be one of the four valid resultant conditions for this instruction.

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for this instruction.

Resultant Conditions: High, low, equal, unequal.

Exceptions

Exception		Operands 1	2	3 [4-6]	Other
06	Addressing				
	01 space addressing violation	X	X	X	
	02 boundary alignment	X	X		
	03 range	X	X		
	04 external data object not found	X	X		
	06 optimized addressability invalid	X	X		
08	Argument/parameter				
	01 parameter reference violation	X	X		
10	Damage encountered				
	04 system object damage state	X	X	X	X
	44 partial system object damage	X	X	X	X
1C	Machine-dependent exception				
	03 machine storage limit exceeded				X
20	Machine support				
	02 machine check				X
	03 function check				X
22	Object access				
	01 object not found	X	X	X	
	02 object destroyed	X	X	X	
	03 object suspended	X	X	X	
24	Pointer specification				
	01 pointer does not exist	X	X	X	
	02 pointer type invalid	X	X	X	

Exception		Operands 1	2	3 [4-6]	Other
2A	Program creation				
	05 invalid op code extender field				X
	06 invalid operand type	X	X	X	
	07 invalid operand attribute	X	X	X	
	08 invalid operand value range	X	X	X	
	09 invalid branch target operand			X	
	0C invalid operand odt reference	X	X	X	X
	0D reserved bits are not zero	X	X	X	X
2E	Resource control limit				
	01 user profile storage limit exceeded				X
36	Space management				
	01 space extension/truncation				X

3.3 Compare Space Addressability (CMPSPADB or CMPSPADI)

Op Code (Hex)	Extender	Operand 1	Operand 2	Operand 3 [4-6]
1CF2	Branch options	Compare operand 1	Compare operand 2	Branch target
18F2	Indicator options			Indicator target

Operand 1: Numeric variable scalar, character variable scalar, numeric variable array, character variable array, pointer, or pointer array.

Operand 2: Numeric variable scalar, character variable scalar, numeric variable array, character variable array, or pointer data object array.

Operand 3 [*4-6*]:

- *Branch Form*-Instruction number, relative instruction number, branch point, or instruction pointer.
- *Indicator Form*-Numeric variable scalar or character variable scalar.

Extender: Branch or indicator options.

Either the branch option or the indicator option is required by the instruction.

The extender field is required along with from one to four branch targets (for branch option) or one to four indicator operands (for indicator option). The branch or indicator operands are required for operand 3 and optional for operands 4-6. See Chapter 1. "Introduction" for the bit encoding of the extender field and the allowed syntax of the branch and indicator operands.

Description: The space addressability of the object specified by operand 1 is compared with the space addressability of the object specified by operand 2.

Based on the results of the comparison, the resulting condition is used with the extender to transfer control (branch form) or to assign a value to each of the indicator operands (indicator form). If the operands are not in the same space, the resultant condition is unequal. If the operands are in the same space and the offset of operand 1 is larger or smaller than the offset of operand 2, the resultant condition is high or low, respectively. Equal occurs only if the operands are in the same space at the same offset. Therefore, the resultant conditions (high, low, equal, and unequal) are mutually exclusive. Consequently, if you specify that an action be taken upon the nonexistence of a condition, this results in the action being taken upon the occurrence of any of the other three possible conditions. For example, a branch not high would result in the branch being taken on a low, equal, or unequal condition.

If a pointer data object operand contains a data pointer value upon execution of the instruction, the addressability is compared to the pointer data object rather than to the scalar described by the data pointer value. The variable scalar references allowed on operands 1 and 2 cannot be described through a data pointer value.

The object destroyed exception, optimized addressability invalid exception, parameter reference violation exception, and pointer does not exist exception are not signaled when operand 1 or operand 2 is based on a space pointer machine object. This occurs when the space pointer machine object contains an internal machine value that indicates one of these error conditions exists. If the corresponding exception is not signaled, the resulting condition of the comparison operation is not defined other than that it will be one of the four valid resultant conditions for this instruction.

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for this instruction.

Resultant Conditions: High, low, equal, unequal.

Exceptions

Exception		Operands 1	2	3 [4-6]	Other
06	Addressing				
	01 space addressing violation	X	X	X	
	02 boundary alignment	X	X	X	
	03 range	X	X	X	
	06 optimized addressability invalid	X	X	X	
08	Argument/parameter				
	01 parameter reference violation	X	X	X	
10	Damage encountered				
	04 system object damage state	X	X	X	X
	44 partial system object damage	X	X	X	X
1C	Machine-dependent exception				
	03 machine storage limit exceeded				X
20	Machine support				
	02 machine check				X
	03 function check				X
22	Object access				
	01 object not found	X	X	X	
	02 object destroyed	X	X	X	
	03 object suspended	X	X	X	
24	Pointer specification				
	01 pointer does not exist	X	X	X	
	02 pointer type invalid	X	X	X	
2A	Program creation				
	05 invalid op code extender field				X
	06 invalid operand type	X	X	X	
	07 invalid operand attribute	X	X	X	

Exception		Operands 1	2	3 [4-6]	Other
	08 invalid operand value range	X	X	X	
	09 invalid branch target operand			X	
	0C invalid operand odt reference	X	X	X	X
	0D reserved bits are not zero	X	X	X	X
2E	Resource control limit				
	01 user profile storage limit exceeded				X
36	Space management				
	01 space extension/truncation				X

3.4 Set Data Pointer (SETDP)

Op Code (Hex)	Operand 1	Operand 2
0096	Receiver	Source 1

Operand 1: Data pointer.

Operand 2: Numeric variable scalar, character variable scalar, numeric variable array, or character variable array.

Description: A data pointer is created and returned in the storage area specified by operand 1 and has the attributes and space addressability of the object specified by operand 2. Addressability is set to the low-order (leftmost) byte of the object specified by operand 2. The attributes given to the data pointer include scalar type and scalar length.

If operand 2 is a substring compound operand, the length attribute is set equal to the length of the substring. If operand 2 is a subscript compound operand, the attributes and addressability of the single array element specified are assigned to the data pointer. If operand 2 is an array, the attributes and addressability of the first element of the array are assigned to the data pointer. A data pointer can only be set to describe an element of a data array, not a data array in its entirety.

When the addressability in the data pointer is modified, the instruction signals the space addressing exception when one of the following conditions occurs:

- When the space address to be stored in the pointer would have a negative offset value.
- When the offset would address an area beyond the largest space allocatable in the object. This maximum offset value is dependent on the size and packaging of the object containing the space and is independent of the actual size of the space allocated.

If the exception is signaled by this instruction for one of these reasons, the pointer is not modified by the instruction.

Attempts to use a pointer whose offset value lies between the currently allocated extent of the space and the maximum allocatable extent cause the space addressing exception to be signaled.

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for this instruction.

Exceptions

Exception		Operands 1	Operands 2	Other
06	Addressing			
	01 space addressing violation	X	X	
	02 boundary alignment	X	X	
	03 range	X	X	
	06 optimized addressability invalid	X	X	

Exception		Operands 1	2	Other
08	Argument/parameter			
	01 parameter reference violation	X	X	
10	Damage encountered			
	04 system object damage state	X	X	X
	44 partial system object damage	X	X	X
1C	Machine-dependent exception			
	03 machine storage limit exceeded			X
20	Machine support			
	02 machine check			X
	03 function check			X
22	Object access			
	01 object not found		X	
	02 object destroyed	X	X	
	03 object suspended	X	X	
24	Pointer specification			
	01 pointer does not exist	X	X	
	02 pointer type invalid	X	X	
2A	Program creation			
	06 invalid operand type	X	X	
	08 invalid operand value range		X	
	0C invalid operand odt reference	X	X	
	0D reserved bits are not zero	X	X	X
2E	Resource control limit			
	01 user profile storage limit exceeded			X
36	Space management			
	01 space extension/truncation			X

3.5 Set Data Pointer Addressability (SETDPADR)

Op Code (Hex)	Operand 1	Operand 2
0046	Receiver	Source

Operand 1: Data pointer.

Operand 2: Numeric variable scalar, character variable scalar, numeric variable array, or character variable array.

Description: The space addressability of the object specified for operand 2 is assigned to the data pointer specified by operand 1. If operand 1 contains a resolved data pointer at the initiation of the instruction's execution, the data pointer's scalar attribute component is not changed by the instruction. If operand 1 contains an initialized but unresolved data pointer at the initiation of the instruction's execution, the data pointer is resolved in order to establish the scalar attribute component of the pointer. If operand 1 contains other than a resolved data pointer at the initiation of the instruction's execution, the instruction creates and returns a data pointer in operand 1 with the addressability of the object specified for operand 2, and the instruction establishes the attributes as a character(1) scalar.

When the addressability is set into a data pointer, the space addressing exception is signaled by the instruction only when the space address to be stored in the pointer has a negative offset value or if the offset addresses beyond the largest space allocatable in the object. This maximum offset value is dependent on the size and packaging of the object containing the space and is independent of the actual size of the space allocated. If the exception is signaled for this reason, the pointer is not modified by the instruction. Attempts to use a pointer whose offset value lies between the currently allocated extent of the space and the maximum allocatable extent of the space cause the space addressing exception to be signaled.

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for this instruction.

Exceptions

Exception		Operands 1	Operands 2	Other
06	Addressing			
	01 space addressing violation	X	X	
	02 boundary alignment	X	X	
	03 range	X	X	
	04 external data object not found	X		
	06 optimized addressability invalid	X	X	
08	Argument/parameter			
	01 parameter reference violation	X	X	
10	Damage encountered			
	04 system object damage state	X	X	X
	44 partial system object damage	X	X	X

Exception		Operands 1	Operands 2	Other
1C	Machine-dependent exception			
	03 machine storage limit exceeded			X
20	Machine support			
	02 machine check			X
	03 function check			X
22	Object access			
	01 object not found	X	X	
	02 object destroyed	X	X	
	03 object suspended	X	X	
24	Pointer specification			
	01 pointer does not exist	X	X	
	02 pointer type invalid	X	X	
2A	Program creation			
	06 invalid operand type	X	X	
	08 invalid operand value range	X	X	
	0C invalid operand odt reference	X	X	
	0D reserved bits are not zero	X	X	X
2E	Resource control limit			
	01 user profile storage limit exceeded			X
36	Space management			
	01 space extension/truncation			X

3.6 Set Data Pointer Attributes (SETDPAT)

Op Code (Hex)	Operand 1	Operand 2
004A	Receiver	Attributes

Operand 1: Data pointer.

Operand 2: Character(7) scalar (fixed-length).

Description: The value of the character scalar specified by operand 2 is interpreted as an encoded representation of an attribute set that is assigned to the attribute portion of the data pointer specified by operand 1. The addressability portion of the data pointer is not modified. If operand 1 contains an initialized but unresolved data pointer at the initiation of the instruction's execution, the data pointer is resolved in order to establish the addressability in the pointer. The attributes specified by the instruction are then assigned to the data pointer. If operand 1 does not contain a data pointer at the initiation of the instruction's execution, an exception is signaled.

The format of the attribute set is as follows:

- Data pointer attributes — Char(7)
  - Scalar type — Char(1)

    Hex 00 = Signed binary
    Hex 01 = Floating-point
    Hex 02 = Zoned decimal
    Hex 03 = Packed decimal
    Hex 04 = Character
    Hex 06 = Onlyns
    Hex 07 = Onlys
    Hex 08 = Either
    Hex 09 = Open
    Hex 0A = Unsigned binary

  - Scalar length — Bin(2)

    If binary or character:
    Length (only 2 or 4 for binary)
    If floating-point:
    Length (only 4 or 8 for floating-point)
    If zoned decimal or packed decimal:
    Fractional digits (F) — Bits 0-7
    Total digits (T) — Bits 8-15
    (where $1 \leq T \leq 31$, $0 \leq F \leq T$)

    If character:
    Length (L, where $1 \leq L \leq 32767$)
    If Onlyns:
    Length = L (where $2 \leq L \leq 32,766$) and, L is the number of bytes, L is even.
    If Onlys or Either:
    Length = L (where $4 \leq L \leq 32,766$) and, L is the number of bytes, L is even, L includes any SO and SI characters.
    If Open:

Length = L (where 4 ≤ L ≤ 32,766) and, L is the number of bytes, L includes any SO and SI characters.

– Reserved (binary 0) Bin(4)

Support for usage of a Data Pointer describing an Onlyns, Onlys, Either, or Open scalar value is limited to the Copy Extended Characters Left Adjusted With Pad instruction. Usage of such a data pointer defined value on any other instruction is not supported and results in the signaling of the scalar type invalid exception.

This support for the Onlyns, Onlys, Either, and Open scalar values is essentially a primitive supplement to more comprehensive support provided by Data Base Management. For more information on the meaning and usage of these scalar values refer to the Create Cursor instruction.

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for this instruction.

Exceptions

Exception		Operands 1	2	Other
06	Addressing			
	01 space addressing violation	X	X	
	02 boundary alignment	X	X	
	03 range	X	X	
	04 external data object not found	X		
	06 optimized addressability invalid	X	X	
08	Argument/parameter			
	01 parameter reference violation	X	X	
10	Damage encountered			
	04 system object damage state	X	X	X
	44 partial system object damage	X	X	X
1C	Machine-dependent exception			
	03 machine storage limit exceeded			X
20	Machine support			
	02 machine check			X
	03 function check			X
22	Object access			
	01 object not found	X	X	
	02 object destroyed	X	X	
	03 object suspended	X	X	
24	Pointer specification			
	01 pointer does not exist	X	X	
	02 pointer type invalid	X	X	

Exception		Operands 1	2	Other
2A	Program creation			
	06 invalid operand type	X	X	
	07 invalid operand attribute	X	X	
	08 invalid operand value range	X	X	
	0A invalid operand length		X	
	0C invalid operand odt reference	X	X	
	0D reserved bits are not zero	X	X	X
2E	Resource control limit			
	01 user profile storage limit exceeded			X
32	Scalar specifications			
	02 scaler attributes invalid		X	
	03 scalar value invalid		X	
36	Space management			
	01 space extension/truncation			X

3.7 Set Space Pointer (SETSPP)

Op Code (Hex)	Operand 1	Operand 2
0082	Receiver	Source

Operand 1: Space pointer.

Operand 2: Numeric variable scalar, character variable scalar, numeric variable array, character variable array, or pointer data object.

Description: A space pointer is returned in operand 1 and is set to address the lowest order (leftmost) byte of the byte string identified by operand 2.

When the addressability is set in a space pointer, the instruction signals the space addressing exception when the offset addresses beyond the largest space allocatable in the object or when the space address to be stored in the pointer has a nonpositive offset value. This offset value is dependent on the size and packaging of the object containing the space and is independent of the actual size of the space allocated. If the exception is signaled for this reason, the pointer is not modified by the instruction. Attempts to use a pointer whose offset value lies between the currently allocated extent of the space and the maximum allocatable extent of the space cause the space addressing exception to be signaled.

If a pointer data object specified for operand 2 contains a data pointer value upon execution of the instruction, the addressability is set to the pointer storage form rather than to the scalar described by the data pointer value. The variable scalar references allowed on operand 2 cannot be described through a data pointer value.

The object destroyed exception, the optimized addressability invalid exception, the parameter reference violation exception, and the pointer does not exist exception are not signaled when operand 1 is a space pointer machine object and operand 2 is based on a space pointer machine object. This occurs when the basing space pointer machine object for operand 2 contains an internal machine value that indicates one of these error conditions exists. If the corresponding exception is not signaled, operand 1 is set with an internal machine value that preserves the exception condition which existed for operand 2. The appropriate exception condition is signaled for either pointer upon a subsequent attempt to reference the space data the pointer addresses.

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for this instruction.

Exceptions

Exception		Operands 1	Operands 2	Other
06	Addressing			
	01 space addressing violation	X	X	
	02 boundary alignment	X	X	
	03 range	X	X	
	06 optimized addressability invalid	X	X	

Exception		Operands 1	2	Other
08	Argument/parameter			
	01 parameter reference violation	X	X	
10	Damage encountered			
	04 system object damage state	X	X	X
	44 partial system object damage	X	X	X
1C	Machine-dependent exception			
	03 machine storage limit exceeded			X
20	Machine support			
	02 machine check			X
	03 function check			X
22	Object access			
	01 object not found	X		
	02 object destroyed	X	X	
	03 object suspended	X	X	
24	Pointer specification			
	01 pointer does not exist	X	X	
	02 pointer type invalid	X	X	
2A	Program creation			
	06 invalid operand type	X	X	
	07 invalid operand attribute	X	X	
	08 invalid operand value range	X	X	
	0A invalid operand length	X	X	
	0C invalid operand odt reference	X	X	
	0D reserved bits are not zero	X	X	X
2E	Resource control limit			
	01 user profile storage limit exceeded			X
pit = 5.32	Scalar specification			
	01 scalar type invalid	X	X	
36	Space management			
	01 space extension/truncation			X

3.8 Set Space Pointer with Displacement (SETSPPD)

Op Code (Hex)	Operand 1	Operand 2	Operand 3
0093	Receiver	Source	Displacement

Operand 1: Space pointer.

Operand 2: Numeric variable scalar, character variable scalar, numeric variable array, character variable array, or pointer data object.

Operand 3: Binary scalar.

Description: A space pointer is returned in operand 1 and is set to the space addressability of the lowest (leftmost) byte of the object specified for operand 2 as modified algebraically by an integer displacement specified by operand 3. Operand 3 can have a positive or negative value.

When the addressability is set in a space pointer, the instruction signals the space addressing exception when the space address to be stored in the pointer has a negative offset value or when the offset addresses beyond the largest space allocatable in the object. This maximum offset value is dependent on the size and packaging of the object containing the space and is independent of the actual size of the space allocated. If the exception is signaled for this reason, the pointer is not modified by the instruction. Attempts to use a pointer whose offset value lies between the currently allocated extent of the space and the maximum allocatable extent of the space cause the space addressing exception to be signaled.

If a pointer data object specified for operand 2 contains a data pointer value upon execution of the instruction, the addressability is set to the pointer storage form rather than to the scalar described by the data pointer value. The variable scalar references allowed on operand 2 cannot be described through a data pointer value.

The object destroyed exception, the optimized addressability invalid exception, the parameter reference violation exception, and the pointer does not exist exception are not signaled when operand 1 is a space pointer machine object and operand 2 is based on a space pointer machine object. This occurs when the basing space pointer machine object for operand 2 contains an internal machine value that indicates one of these error conditions exists. If the corresponding exception is not signaled, operand 1 is set with an internal machine value that preserves the exception condition which existed for operand 2. The appropriate exception condition is signaled for either pointer upon a subsequent attempt is made to reference the space data the pointer addresses.

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for this instruction.

Exceptions

Exception		Operands 1	2	Other	
06	Addressing				
	01 space addressing violation	X	X	X	
	02 boundary alignment	X	X	X	
	03 range	X	X	X	
	06 optimized addressability invalid	X	X	X	
08	Argument/parameter				
	01 parameter reference violation	X	X	X	
10	Damage encountered				
	04 system object damage state	X	X	X	X
	44 partial system object damage	X	X	X	X
1C	Machine-dependent exception				
	03 machine storage limit exceeded				X
20	Machine support				
	02 machine check				X
	03 function check				X
22	Object access				
	01 object not found	X	X	X	
	02 object destroyed	X	X	X	
	03 object suspended	X	X	X	
24	Pointer specification				
	01 pointer does not exist	X	X	X	
	02 pointer type invalid	X	X	X	
2A	Program creation				
	06 invalid operand type	X	X	X	
	07 invalid operand attribute	X	X	X	
	08 invalid operand value range	X	X	X	
	0C invalid operand odt reference	X	X	X	
	0D reserved bits are not zero	X	X	X	X
2E	Resource control limit				
	01 user profile storage limit exceeded			X	
36	Space management				
	01 space extension/truncation			X	

3.9 Set Space Pointer from Pointer (SETSPPFP)

Op Code (Hex)	Operand 1	Operand 2
0022	Receiver	Source Pointer

Operand 1: Space pointer.

Operand 2: Data pointer, system pointer, or space pointer.

Description: A space pointer is returned in operand 1 with the addressability to a space object from the pointer specified by operand 2.

The meaning of the pointers allowed for operand 2 is as follows:

Pointer	Meaning
Data pointer or space pointer	The space pointer returned in operand 1 is set to address of the leftmost byte of the byte string addressed by the source pointer for operand 2.
System pointer	The space pointer returned in operand 1 is set to address the first byte of the space contained in the system object addressed by the system pointer for operand 2. The space object addressed is either the created system space or an associated space for the system object addressed by the system pointer. If the operand 2 system pointer addresses a system object with no associated space, the invalid space reference exception is signaled.

The object destroyed exception, optimized addressability invalid exception, parameter reference violation exception, and pointer does not exist exception are not signaled when operand 1 and operand 2 are space pointer machine objects. This occurs when operand 2 contains an internal machine value that indicates one of these error conditions exists. If the corresponding exception is not signaled, operand 1 is set with an internal machine value that preserves the exception condition that existed for operand 2. The appropriate exception condition will be signaled for either pointer when a subsequent attempt is made to reference the space data that the pointer addresses.

Authorization Required

- Space authority
  - Operand 2 (if a system pointer)
- Retrieve
  - Contexts referenced for address resolution

Lock Enforcement

- Materialize
  - Contexts referenced for address resolution

Exceptions

Exception		Operands 1	2	Other
06	Addressing			
	01 space addressing violation	X	X	
	02 boundary alignment	X	X	
	03 range	X	X	
	04 external data object not found		X	
	05 invalid space reference		X	
	06 optimized addressability invalid	X	X	
08	Argument/parameter			
	01 parameter reference violation			X
0A	Authorization			
	01 unauthorized for operation		X	
10	Damage encountered			
	04 system object damage state	X	X	X
	05 authority verification terminated due to damaged object			X
	44 partial system object damage	X	X	X
1A	Lock state			
	01 invalid lock state		X	
1C	Machine-dependent exception			
	03 machine storage limit exceeded			X
20	Machine support			
	02 machine check			X
	03 function check			X
22	Object access			
	01 object not found	X	X	
	02 object destroyed	X	X	
	03 object suspended	X	X	
	07 authority verification terminated due to destroyed object			X
24	Pointer specification			
	01 pointer does not exist	X	X	
	02 pointer type invalid	X	X	
	03 pointer addressing invalid object		X	
2A	Program creation			
	06 invalid operand type	X	X	
	07 invalid operand attribute	X	X	
	08 invalid operand value range	X	X	

Exception		Operands 1	2	Other
	0C invalid operand odt reference	X	X	
	0D reserved bits are not zero	X	X	X
2E	Resource control limit			
	01 user profile storage limit exceeded			X
36	Space management			
	01 space extension/truncation			X

3.10 Set Space Pointer Offset (SETSPPO)

Op Code (Hex)	Operand 1	Operand 2
0092	Receiver	Source 1

Operand 1: Space pointer.

Operand 2: Binary scalar.

Description: The value of the binary scalar specified by operand 2 is assigned to the offset portion of the space pointer identified by operand 1. The space pointer continues to address the same space object.

Operand 1 must contain a space pointer at the initiation of the instruction's execution; otherwise, an invalid pointer type exception is signaled.

When the addressability in the space pointer is modified, the instruction signals a space addressing exception when one of the following conditions occurs:

- When the space address to be stored in the pointer has a negative offset value.
- When the offset addresses beyond the largest space allocatable in the object. This maximum offset value is dependent on the size and packaging of the object containing the space and is independent of the actual size of the space allocated.

If the exception is signaled by this instruction for this reason, the pointer is not modified by the instruction.

Attempts to use a pointer whose offset value lies between the currently allocated extent of the space and the maximum allocatable extent cause the space addressing exception to be signaled.

Exceptions

Exception		Operands 1	Operands 2	Other
06	Addressing			
	01 space addressing violation	X	X	
	02 boundary alignment	X	X	
	03 range	X	X	
	06 optimized addressability invalid	X	X	
08	Argument/parameter			
	01 parameter reference violation	X	X	
10	Damage encountered			
	04 system object damage state	X	X	X
	44 partial system object damage	X	X	X
1C	Machine-dependent exception			
	03 machine storage limit exceeded			X
20	Machine support			

Exception		Operands 1	Operands 2	Other
	02 machine check			X
	03 function check			X
22	Object access			
	01 object not found	X	X	
	02 object destroyed	X	X	
	03 object suspended	X	X	
24	Pointer specification			
	01 pointer does not exist	X	X	
	02 pointer type invalid	X	X	
2A	Program creation			
	06 invalid operand type	X	X	
	07 invalid operand attribute	X	X	
	08 invalid operand value range	X	X	
	0C invalid operand odt reference	X	X	
	0D reserved bits are not zero	X	X	X
2E	Resource control limit			
	01 user profile storage limit exceeded			X
32	Scalar specification			
	01 scalar type invalid	X	X	
36	Space management			
	01 space extension/truncation			X

3.11 Set System Pointer from Pointer (SETSPFP)

Op Code (Hex)	Operand 1	Operand 2
0032	Receiver	Source pointer

Operand 1: System pointer.

Operand 2: System pointer, space pointer, data pointer, or instruction pointer.

Description: This instruction returns a system pointer to the system object address by the supplied pointer.

If operand 2 is a system pointer, then a system pointer addressing the same object is returned in operand 1 containing the same authority as the input pointer.

If operand 2 is a space pointer or a data pointer, then a system pointer addressing the system object that contains the associated space addressed by operand 2 is returned in operand 1.

If operand 2 is an instruction pointer, then a system pointer addressing the program system object that contains the instruction addressed by operand 2 is returned in operand 1.

If operand 2 is an unresolved system pointer or data pointer, the pointer is resolved first.

Authorization Required

- Retrieve
  - Contexts referenced for address resolution

Lock Enforcement

- Materialization
  - Contexts referenced for address resolution

Exceptions

Exception		Operands 1	Operands 2	Other
06	Addressing			
	01 space addressing violation	X	X	
	02 boundary alignment	X	X	
	03 range	X	X	
	04 external data object not found	X	X	
	06 optimized addressability invalid	X	X	
08	Argument/parameter			
	01 parameter reference violation			X
0A	Authorization			
	01 unauthorized for operation		X	

Exception		Operands 1	2	Other
10	Damage encountered			
	02 machine context damage			X
	04 system object damage state	X	X	X
	05 authority verification terminated due to damaged object			X
	44 partial system object damage	X	X	X
1A	Lock state			
	01 invalid lock state		X	
1C	Machine-dependent exception			
	03 machine storage limit exceeded			X
20	Machine support			
	02 machine check			X
	03 function check			X
22	Object access			
	01 object not found		X	
	02 object destroyed	X	X	
	03 object suspended	X	X	
	07 authority verification terminated due to destroyed object			X
24	Pointer specification			
	01 pointer does not exist	X	X	
	02 pointer type invalid	X	X	
2A	Program creation			
	06 invalid operand type	X	X	
	07 invalid operand attribute	X	X	
	08 invalid operand value range	X	X	
	0C invalid operand odt reference	X	X	
	0D reserved bits are not zero	X	X	X
2E	Resource control limit			
	01 user profile storage limit exceeded			X
32	Scalar specification			
	01 scalar type invalid	X	X	
36	Space management			
	01 space extension/truncation			X

3.12 Store Space Pointer Offset (STSPPO)

Op Code (Hex)	Operand 1	Operand 2
00A2	Receiver	Source

Operand 1: Binary variable scalar.

Operand 2: Space pointer.

Description: The offset value of the space pointer referenced by operand 2 is stored in the binary variable scalar defined by operand 1.

If operand 2 does not contain a space pointer at the initiation of the instruction's execution, an invalid pointer type exception is signaled. If binary size exceptions are to be signalled either because the program creation attribute indicated to do so or because a translator directive indicated to do so, they will be signalled under the following conditions. If the offset value is greater than 32 767 and operand 1 is a signed binary (2) scalar, a size exception is signaled. If the offset value is greater than 65 535 and operand 1 is an unsigned binary (2) scalar, a size exception is signaled.

Exceptions

Exception		Operands 1	Operands 2	Other
06	Addressing			
	01 space addressing violation	X	X	
	02 boundary alignment	X	X	
	03 range	X	X	
	06 optimized addressability invalid	X	X	
08	Argument/parameter			
	01 parameter reference violation	X	X	
0C	Computations			
	0A size	X		
10	Damage encountered			
	04 system object damage state	X	X	X
	44 partial system object damage	X	X	X
1C	Machine-dependent exception			
	03 machine storage limit exceeded			X
20	Machine support			
	02 machine check			X
	03 function check			X
22	Object access			
	01 object not found	X	X	
	02 object destroyed	X	X	
	03 object suspended	X	X	

Exception		Operands 1	2	Other
24	Pointer specification			
	01 pointer does not exist	X	X	
	02 pointer type invalid	X	X	
2A	Program creation			
	06 invalid operand type	X	X	
	07 invalid operand attribute	X		
	08 invalid operand value range	X	X	
	0C invalid operand odt reference	X	X	
	0D reserved bits are not zero	X	X	X
2E	Resource control limit			
	01 user profile storage limit exceeded			X
36	Space management			
	01 space extension/truncation			X

3.13 Subtract Space Pointer Offset (SUBSPP)

Op Code (Hex)	Operand 1	Operand 2	Operand 3
0087	Receiver pointer	Source pointer	Decrement

Operand 1: Space pointer.

Operand 2: Space pointer.

Operand 3: Binary scalar.

Description: The value of the binary scalar specified by operand 3 is subtracted from the space address contained in the space pointer specified by operand 2; the result is stored in the space pointer identified by operand 1. Operand 3 can have a positive or negative value. The space object that the pointer is addressing is not changed by the instruction. If operand 2 does not contain a space pointer at the initiation of the instruction's execution, an invalid pointer type exception is signaled.

When the addressability in the space pointer is modified, the instruction signals a space addressing exception when one of the following conditions occurs:

- When the space address to be stored in the pointer has a negative offset value.
- When the offset addresses beyond the largest space allocatable in the object. This maximum offset value is dependent on the size and packaging of the object containing the space and is independent of the actual size of the space allocated.

If the exception is signaled by this instruction for this reason, the pointer is not modified by the instruction.

Attempts to use a pointer whose offset value lies between the currently allocated extent of the space and the maximum allocatable extent cause the space addressing exception to be signaled.

The object destroyed exception, optimized addressability invalid exception, parameter reference violation exception, and pointer does not exist exception are not signaled when operand 1 and operand 2 are space pointer machine objects. This occurs when operand 2 contains an internal machine value that indicates one of these error conditions exists. If the corresponding exception is not signaled, operand 1 is set with an internal machine value that preserves the exception condition that existed for operand 2. The appropriate exception condition will be signaled for either pointer when a subsequent attempt is made to reference the space data that the pointer addresses.

Exceptions

Exception		Operands 1	2	Other
06	Addressing			
	01 space addressing violation	X	X	X
	02 boundary alignment	X	X	X

Exception		Operands 1	2	Other	
	03 range	X	X	X	
	06 optimized addressability invalid	X	X	X	
08	Argument/parameter				
	01 parameter reference violation	X	X	X	
10	Damage encountered				
	04 system object damage state	X	X	X	X
	44 partial system object damage	X	X	X	X
1C	Machine-dependent exception				
	03 machine storage limit exceeded				X
20	Machine support				
	02 machine check				X
	03 function check				X
22	Object access				
	01 object not found	X	X	X	
	02 object destroyed	X	X	X	
	03 object suspended	X	X	X	
24	Pointer specification				
	01 pointer does not exist	X	X	X	
	02 pointer type invalid	X	X	X	
2A	Program creation				
	06 invalid operand type	X	X	X	
	07 invalid operand attribute	X		X	
	08 invalid operand value range	X	X	X	
	0C invalid operand odt reference	X	X	X	
	0D reserved bits are not zero	X	X	X	X
2E	Resource control limit				
	01 user profile storage limit exceeded			X	
32	Scalar specification				
	01 scalar type invalid	X	X	X	
	03 scalar value invalid		X	X	
36	Space management				
	01 space extension/truncation			X	

Chapter 4. Space Management Instructions

This chapter describes the instructions used for space management. These instructions are in alphabetic order. For an alphabetic summary of all the instructions, see Appendix A, "Instruction Summary."

4.1 Materialize Space Attributes (MATS)

Op Code (Hex)	Operand 1	Operand 2
0036	Receiver	Space object

Operand 1: Space pointer.

Operand 2: System pointer.

Description: The current attributes of the space object specified by operand 2 are materialized into the receiver specified by operand 1.

The first 4 bytes that are materialized identify the total number of bytes that may be used by the instruction. This value is supplied as input to the instruction and is not modified by the instruction. A value of less than 8 causes a materialization length exception.

The second 4 bytes that are materialized identify the total number of bytes available to be materialized. The instruction materializes as many bytes as can be contained in the area specified as the receiver. If the byte area identified by the receiver is greater than that required to contain the information requested, then the excess bytes are unchanged. No exceptions (other than the materialization length exception described previously) are signaled in the event that the receiver contains insufficient area for the materialization.

The template identified by operand 1 must be 16-byte aligned in the space. The format of the materialization is as follows:

- Materialization size specification — Char(8)
  - Number of bytes provided for materialization — Bin (4)
  - Number of bytes available for materialization (always 96 for this instruction) — Bin (4)
- Object identification — Char(32)
  - Object type — Char(1)
  - Object subtype — Char(1)
  - Object name — Char(30)
- Object creation options — Char(4)
  - Existence attributes — Bit 0

    0 = Temporary
    1 = Permanent
  - Space attribute — Bit 1

    0 = Fixed-length
    1 = Variable-length
  - Context — Bit 2

    0 = Addressability not in context
    1 = Addressability in context
  - Access group — Bit 3

0 = Not member of access group
1 = Member of access group

- Reserved (binary 0) Bits 4-12
- Initialize space Bit 13

  0 = Initialize
  1 = Do not initialize

- Automatically extend space Bit 14

  0 = No
  1 = Yes

- Reserved (binary 0) Bits 15-31

- Reserved (binary 0) Char(4)
- Size of space Bin(4)
- Initial value of space Char(1)
- Performance class Char(4)
  - Space alignment Bit 0

    0 = The space associated with the object is allocated to allow proper alignment of pointers at 16-byte alignments within the space. If no space is specified for the object, this value must be specified for the performance class.
    1 = The space associated with the object is allocated to allow proper alignment of pointers at 16-byte alignments within the space as well as to allow proper alignment of input/output buffers at 512-byte alignments within the space.

  - Reserved (binary 0) Bits 1-4
  - Main storage pool selection Bit 5

    0 = Process default main storage pool is used for object.
    1 = Machine default main storage pool is used for object.

  - Transient storage pool selection Bit 6

    0 = Default main storage pool (process default or machine default as specified for main storage pool selection) is used for object.
    1 = Transient storage pool is used for object.

  - Block transfer on implicit access state modification Bit 7

    0 = Transfer the minimum storage transfer size for this object. This value is 1 storage unit.
    1 = Transfer the machine default storage transfer size. This value is 8 storage units.

  - Unit number Bits 8-15
  - Reserved (binary 0) Bits 16-31
- Reserved (binary 0) Char(7)
- Context System pointer
- Access group System pointer

This instruction cannot be used to materialize the public authority specified creation option, the initial owner specified creation option, or the template extension which can be specified on space creation. The Materialize Authority instruction can be used to materialize the current public authority for the space. The Materialize System Object instruction can be used to materialize the current owner of the space.

Authorization Required

- Operational or space authority
  - Operand 2
- Retrieve
  - Contexts referenced for address resolution

Lock Enforcement

- Materialize
  - Operand 2
  - Contexts referenced for address resolution

Exceptions

Exception		Operands 1	Operands 2	Other
06	Addressing			
	01 space addressing violation	X	X	
	02 boundary alignment	X	X	
	03 range	X	X	
	06 optimized addressability invalid	X	X	
08	Argument/parameter			
	01 parameter reference violation	X	X	
0A	Authorization			
	01 unauthorized for operation		X	
10	Damage encountered			
	04 system object damage state	X	X	X
	05 authority verification terminated due to damaged object			X
	44 partial system object damage	X	X	X
1A	Lock state			
	01 invalid lock state		X	
1C	Machine-dependent exception			
	03 machine storage limit exceeded			X
20	Machine support			
	02 machine check			X
	03 function check			X

Exception		Operands 1	2	Other
22	Object access			
	01 object not found	X	X	
	02 object destroyed	X	X	
	03 object suspended	X	X	
	07 authority verification terminated due to destroyed object			X
24	Pointer specification			
	01 pointer does not exist	X	X	
	02 pointer type invalid	X	X	
	03 pointer addressing invalid object		X	
2A	Program creation			
	06 invalid operand type	X	X	
	07 invalid operand attribute	X	X	
	08 invalid operand value range	X	X	
	0A invalid operand length	X		
	0C invalid operand odt reference	X	X	
	0D reserved bits are not zero	X	X	X
2E	Resource control limit			
	01 user profile storage limit exceeded			X
36	Space management			
	01 space extension/truncation			X
38	Template specification			
	03 materialization length exception	X		

4.2 Modify Space Attributes (MODS)

Op Code (Hex)	Operand 1	Operand 2
0062	System object.	Space modification template.

Operand 1: System pointer.

Operand 2: Binary scalar or character(28) scalar.

Description: The attributes of the space associated with the system object specified for operand 1 are modified with the attribute values specified in operand 2. Operand 1 may address any system object.

The operand 2 space modification template is specified with one of two formats. The abbreviated format, operand 2 specified as a binary scalar, only provides for modifying the size of space attribute. The full format, operand 2 specified as a character scalar, provides for modifying the full set of space attributes.

When operand 2 is a binary value, it specifies the size in bytes to which the space size is to be modified. The current allocation of the space is extended or truncated accordingly to match as closely as possible the specified size. The modified space size will be of at least the size specified. The actual size allocated is dependent upon the algorithm used within the specific implementation of the machine. If the space is of fixed size, or if the value of operand 2 is negative, or if the operand 2 size is larger than that for the largest space that can be associated with the object, the space extension/truncation exception is signaled. When operand 2 is a character scalar, it specifies a selection of space attribute values to be used to modify the attributes of the space. It must be at least 28 bytes long and have the following format:

- Modification selection — Char(4)
  - Modify space length attribute — Bit 0

    0 = No
    1 = Yes
  - Modify size of space — Bit 1

    0 = No
    1 = Yes
  - Modify initial value of space — Bit 2

    0 = No
    1 = Yes
  - Modify performance class — Bit 3

    0 = No
    1 = Yes
  - Modify initialize space attribute — Bit 4

    0 = No
    1 = Yes
  - Reinitialize space — Bit 5

0 = No
1 = Yes

- Modify automatically extend space attribute — Bit 6

  0 = No
  1 = Yes

- Reserved (binary 0) — Bits 7-31

- Indicator attributes — Char(4)
  - Reserved (binary 0) — Bit 0
  - Space length — Bit 1

    0 = Fixed length
    1 = Variable length

  - Initialize space — Bit 2

    0 = Initialize
    1 = Do not initialize

  - Automatically extend space — Bit 3

    0 = No
    1 = Yes

  - Reserved binary 0) — Bits 4-31
- Reserved (binary 0) — Char(4)
- Size of space — Bin(4)
- Initial value of space — Char(1)
- Performance class — Char(4)
- Reserved (binary 0) — Char(7)

The modification selection indicator fields select the modifications to be performed on the space.

The modify space length attribute modification selection field controls whether or not the space length attribute is to be modified. When yes is specified, the value of the space length indicator is used to modify the space to be specified fixed or variable length attribute. When no is specified, the space length indicator attribute value is ignored and the space length attribute is not modified.

The modify size of space modification selection field controls whether or not the allocation size of the space is to be modified. When yes is specified, the current allocation of the space is extended or truncated accordingly to match as closely as possible the specified size. The modified size will be at least the size specified. The actual size allocated is dependent upon the algorithm used within the specific implementation of the machine. When no is specified, the current allocation of the space is not modified and the size of space field is ignored.

Modification of the size of space attribute for a space of fixed length can only be performed in conjunction with modification of the space length attribute. In this case, the space length attribute may be modified to the same fixed length attribute or to the variable length attribute. An attempt to modify the size of space

attribute for a space of fixed length without modification of the space length attribute results in the signaling of the space extension/truncation exception. Modification of the size of space attribute for a space of variable length can always be performed separately from a modification of the space length attribute.

When the size of space attribute is to be modified, if the value of the size of space field is negative or specifies a size larger than that for the largest space that can be associated with the object, the space extension/truncation exception is signaled. The modify initial value of space modification selection field controls whether or not the initial value of space attribute is to be modified. When yes is specified, the value of the initial value of space field is used to modify the corresponding attribute of this space. This byte value will be used to initialize any new space allocations for this space due to an extension to the size of space attribute on the current execution of this instruction as well as any subsequent modifications. When no is specified, the initial value of space field is ignored and the initial value of space attribute is not modified.

The modify performance class modification selection field controls whether or not the performance class attribute of the specified system object is to be modified with the values relating to space objects. When yes is specified, the value of the performance class field is used to modify the corresponding attribute of the specified system object. When no is specified, the performance class attribute of the specified system object is not modified.

The modify initialize space attribute modification selection field controls whether or not the initialize space attribute is to be modified. When yes is specified, the value of the initialize space indicator attribute is used to modify that attribute of the specified space to the specified value. When no is specified, the initialize space indicator attribute value is ignored and the initialize space attribute is not modified.

Changing the value of the initialize space attribute only affects whether or not future extensions of the space will be initialized or not. That is, it is the state of this attribute at the time of allocation of the storage for a space that determines whether that newly allocated storage area will be initialized to the initial value specified for the space. Modifications of this attribute subsequent to the allocation of storage to a space have no effect on the value of that previously allocated storage area.

The reinitialize space modification selection field controls whether the storage allocated to the space is to be reinitialized in its entirety. When no is specified, the space is not reinitialized. When yes is specified, the space is reinitialized. This reinitialization is performed after all other attribute modifications which may also have been specified on the instruction have been made. Thus changes to the size of the space, the initial value of the space, etc will be put into effect and be considered the current attributes of the space for purposes of the reinitialization. The byte value used for the reinitialization is either the current initial value for the space if the initialize space attribute for the space currently specifies yes, or a value of X'00' if the initialize space attribute currently specifies no.

Note that specifying yes for the reinitialize space modification selection field for a space with current attributes of fixed length size zero results in no operation, because such a space has no allocated storage to reinitialize. Also, note that reinitialization of a space will have the side effect of resetting partial damage for a space object containing the space if the space object had previously been

marked as having partial damage. This only applies to space objects; i.e. reinitialization of an associated space does not have the side effect of resetting partial damage for the MI object containing it.

The modify automatically extend space attribute modification selection field controls whether or not the automatically extend space attribute is to be modified. When yes is specified, the value of the automatically extend space indicator attribute is used to modify that attribute of the specified space to the specified value. When no is specified, the automatically extend space indicator attribute value is ignored and the automatically extend space attribute is not modified. The automatically extend space attribute can only be specified as yes when the space length attribute for the space is already variable length, or when the space length attribute is being modified to variable length. Invalid specification of the automatically extend space attribute results in the signaling of the invalid space modification exception.

Modification to or from the state of a space being fixed length of size zero can not be performed for the following objects:

Class of Service Description

Controller description

Cursor

Data space

Logical unit description

Mode Description

Network description

Space

If such a modification is attempted for these objects, the invalid space modification exception is signaled.

Specifying yes for the modify performance class modification selection field is only allowed when the space to be modified is a fixed length space of size zero. This modification may be specified in conjunction with other modifications. Only bit 0 of the performance class field is used to modify the performance class attribute of the specified system object. A bit value of zero requests that the start of the space storage provide 16-byte multiple (pointer) machine address alignment. A bit value of one requests that the start of the space storage provide 512-byte multiple (buffer) machine address alignment. Bits 1 through 31 are ignored. Specifying yes for the modify performance class modification selection field when the space to be modified is not a fixed length space of size zero results in the signaling of the invalid space modification exception.

A fixed length space of size zero is defined by the machine to have no internal storage allocation. Due to this, a modification to or from this state is, in essence, the same as a destroy or create for the space associated with the specified system object. The effect of modifying to this state is similar to destroying the associated space in that address references to the space through previously set pointers will result in signaling of the object destroyed exception. Additionally, an attempt to set a space pointer to the space associated with the specified system object through the Set Space Pointer from Pointer instruction will result in the signaling of the invalid space reference exception. To the con-

trary, modifying the space attributes from this state is similar to creating an associated space in that the Set Space Pointer from Pointer instruction can be used to set a space pointer to the start of a storage within the associated space and the allocated space storage can be used to contain space data.

The extension and truncation of a space is always by an implementation-defined multiple of 256 bytes. This means that if, for example, the implementation defined multiple is 2 (or 512 bytes), any modification of the space size will be in increments of 512 bytes.

Authorization Required

- Object management
  - Operand 1
- Retrieve
  - Contexts referenced for address resolution

Lock Enforcement

- Materialize
  - Contexts referenced for address resolution
- Object control
  - Operand 1 (when operand 2 is binary)
- Modify
  - Operand 1 (when operand 2 is character)

Exceptions

Exception		Operands 1	Operands 2	Other
06	Addressing			
	01 space addressing violation	X	X	
	02 boundary alignment	X	X	
	03 range	X	X	
	06 optimized addressability invalid	X	X	
08	Argument/parameter			
	01 parameter reference violation	X	X	
0A	Authorization			
	01 unauthorized for operation	X		
10	Damage encountered			
	04 system object damage state	X	X	X
	05 authority verification terminated due to damaged object			X
	44 partial system object damage	X	X	X
1A	Lock state			
	01 invalid lock state	X		

Exception		Operands 1	2	Other
1C	Machine-dependent exception			
	03 machine storage limit exceeded			X
	04 object storage limit exceeded	X		
20	Machine support			
	02 machine check			X
	03 function check			X
22	Object access			
	01 object not found	X	X	
	02 object destroyed	X	X	
	03 object suspended	X	X	
	07 authority verification terminated due to destroyed object			X
24	Pointer specification			
	01 pointer does not exist	X	X	
	02 pointer type invalid	X	X	
	03 pointer addressing invalid object	X		
2A	Program creation			
	06 invalid operand type	X	X	
	07 invalid operand attribute	X	X	
	08 invalid operand value range	X	X	
	0A invalid operand length		X	
	0C invalid operand odt reference	X	X	
	0D reserved bits are not zero	X	X	X
2E	Resource control limit			
	01 user profile storage limit exceeded	X		
36	Space management			
	01 space extension/truncation	X	X	
	02 invalid space modification	X	X	

Chapter 5. Program Management Instructions

This chapter describes all instructions used for program management. These instructions are in alphabetic order. For an alphabetic summary of all the instructions, see Appendix A, "Instruction Summary"

5.1 Materialize Program (MATPG)

Op Code (Hex)	Operand 1	Operand 2
0232	Attribute receiver	Program

Operand 1: Space pointer.

Operand 2: System pointer.

Description: The program identified by operand 2 is materialized into the template identified by operand 1.

Operand 2 is a system pointer that identifies the program to be materialized. The values in the materialization relate to the current attributes of the materialized program.

The template identified by operand 1 must be 16-byte aligned.

The first 4 bytes of the materialization template identify the total number of bytes in the template. This value is supplied as input to the instruction and is not modified. A value of less than 8 causes the materialization length exception to be signaled.

The second 4 bytes of the materialization template are modified by the instruction to contain a value identifying the template size required to provide for the total number of bytes available to be materialized. The instruction materializes as many bytes as can be contained in the area specified by the receiver. If the byte area identified by the receiver is greater than that required to contain the information requested, then the excess bytes are unchanged. No exceptions (other than the materialization length exception) are signaled in the event that the receiver contains insufficient area for the materialization.

The following attributes apply to the materialization of a program:

- The existence attribute indicates whether the program is temporary or permanent.
- The observation attribute entry specifies the template components of the programs that currently can be materialized.
- If the program has an associated space, then the space attribute is set to indicate either fixed- or variable-length; the initial value for the space is returned in the initial value of space entry, and the size of space entry is set to the current size value of the space. If the program has no associated space, the size of space entry is set to a zero value, and the space attribute and initial value of space entry values are meaningless.
- If the program is addressed by a context, then the context addressability attribute is set to indicate this, and a system pointer to the addressing context is returned in the context entry. If the program is not addressed by a context, then the context addressability attribute is set to indicate this, and binary 0's are returned in the context entry.

- If the program is a member of an access group, then the access group attribute is set to indicate this, and a system pointer to the access group is returned in the access group entry. If the program is not a member of an access group, then the access group attribute is set to indicate this, and binary 0's are returned in the access group entry.
- The performance class entry is set to reflect the performance class information associated with the program.
- The user exit attribute defines if the referenced program is allowed to be used as a user exit program.

The program data cannot be materialized if observability has been removed. If the program was created with an observation attribute that cannot be materialized, the program data (instruction stream, ODV, OES, user data, and object mapping table components) cannot be materialized by this instruction. If the program data cannot be materialized, 0's are placed in the fields of the program template that describe the size and offsets to the program data components. The only information that can be materialized is that part of the program template up to and including the offset to the OMT (object mapping template) entry.

The offset to the OMT component entry specifies the location of the OMT component in the materialized program template. The OMT consists of a variable-length vector of 6-byte entries. The number of entries is identical to the number of ODV entries because there is one OMT entry for each ODV entry. The OMT entries correspond one for one with the ODV entries; each OMT entry gives a location mapping for the object defined by its associated ODV entry.

The following describes the formats for an OMT entry:

- OMT entry Char(6)
  - Addressability type Char(1)

    Hex 00= Base addressability is from the start of the static storage

    Hex 01= Base addressability is from the start of the automatic storage area

    Hex 02= Base addressability is from the start of the storage area addressed by a space pointer

    Hex 03= Base addressability is from the start of the storage area of a parameter

    Hex 04= Base addressability is from the start of the storage area addressed by the space pointer found in the process communication object attribute of the process executing the program

    Hex FF= Base addressability not provided. The object is contained in machine storage areas to which addressability cannot be given, or a parameter has addressability to an object that is in the storage of another program
  - Offset from base Char(3)

    For types hex 00, hex 01, hex 02, hex 03, and hex 04, this is a 3-byte logical binary value representing the offset to the object from the base addressability. For type hex FF, the value is binary 0.
  - Base addressability Char(2)

For types hex 02 and hex 03, this is a 2-byte binary field containing the number of the OMT entry for the space pointer or a parameter that provides base addressability for this object. For types hex 00, hex 01, hex 04, and hex FF, the value is binary 0.

Authorization Required

- Retrieve
  - Operand 2
  - Contexts referenced for address resolution

Lock Enforcement

- Materialize
  - Operand 2
  - Contexts referenced for address resolution

Exceptions

Exception		Operands 1	Operands 2	Other
06	Addressing			
	01 space addressing violation	X	X	
	02 boundary alignment	X	X	
	03 range	X	X	
	06 optimized addressability invalid	X	X	
08	Argument/parameter			
	01 parameter reference violation	X	X	
0A	Authorization			
	01 unauthorized for operation		X	
10	Damage encountered			
	04 system object damage state	X	X	X
	05 authority verification terminated due to damaged object			X
	44 partial system object damage	X	X	X
1A	Lock state			
	01 invalid lock state		X	
1C	Machine-dependent exception			
	03 machine storage limit exceeded			X
20	Machine support			
	02 machine check			X
	03 function check			X
22	Object access			
	01 object not found	X	X	
	02 object destroyed	X	X	

Exception		Operands 1	2	Other
	03 object suspended	X	X	
	07 authority verification terminated due to destroyed object			X
24	Pointer specification			
	01 pointer does not exist	X	X	
	02 pointer type invalid	X	X	
	03 pointer addressing invalid object		X	
2A	Program creation			
	06 invalid operand type	X	X	
	07 invalid operand attribute	X	X	
	08 invalid operand value range	X	X	
	0A invalid operand length	X		
	0C invalid operand odt reference	X		
	0D reserved bits are not zero	X	X	X
2E	Resource control limit			
	01 user profile storage limit exceeded			X
36	Space management			
	01 space extension/truncation			X
38	Template specification			
	03 materialization length exception	X		

Chapter 6. Program Execution Instructions

This chapter describes the instructions used for program execution control. These instructions are in alphabetic order. For an alphabetic summary of all the instructions, see Appendix A, "Instruction Summary."

6.1 Activate Program (ACTPG)

Op Code (Hex)	Operand 1	Operand 2
0212	Program or program activation entry	Program

Operand 1: Space pointer, system pointer, or data object.

Operand 2: System pointer.

Warning: The following information is subject to change from release to release. Use it with caution and be prepared to adjust for changes with each new release.

Description: This instruction allocates and initializes storage for static objects that are declared for a specified program within the executing process. The program identified by operand 2 is activated in the executing process. The program is activated by allocating an area in the PSSA (process static storage area) to contain the program static storage. This static storage is then available each time the program is invoked within the process. The pointer object specified by operand 1 receives a space pointer addressing the activation of the referenced program. The activation consists of storage for the program's static objects as well as a system pointer to the associated program, a space pointer to the next activation entry (if one exists) in the PSSA, a space pointer to the preceding activation entry in the PSSA, and attributes specifying the status of the activation. The PSSA is located by a space pointer specified when the process was initiated. The location identified by the space pointer is considered to be the beginning of the PSSA and must be 16-byte aligned.

The user must properly initialize the PSSA base entry before the first program is activated in the process.

A space pointer locating the PSSA can be materialized using the Materialize Process instruction.

If the chain being modified bit is on and an attempt is made to activate or deactivate a program with static storage, a stack control invalid exception is signaled.

The program is activated by allocating an area in the PSSA space sufficient to contain the activation entry. The area used for allocating the first activation in a space is located by the next available storage location pointer in the PSSA base

entry; otherwise, this pointer locates the first free byte after all activation entries in the space. This pointer must address a 16-byte aligned area in the space, or a boundary alignment exception is signaled. The pointer may be set to address beyond the currently allocated storage in the space, which is implicitly extended, and no exception is signaled. If the space is not currently large enough to contain the entry and if it is extendable, it is implicitly extended by the machine. The owner's authority to the space is included with the authority of the extending process when checking for object management authority when the space is extended. If the space is of a fixed size or cannot be extended to contain the entry, a space extension truncation exception is signaled.

The new activation entry is initialized as follows:

- The previous activation entry pointer is copied from the most recent activation entry in the PSSA base entry.
- The next activation entry pointer field is unchanged by the instruction (the last activation is process pointer in the PSSA base entry specifies the last activation on the chain).
- The associated program pointer is copied from the operand 2 system pointer.
- The activation number is set to a value one greater than the activation number entry in the previous activation.
- The activation is marked as active (the activation status is set to binary 1).
- The invocation count is set to 0.
- The activation mark is obtained by incrementing the mark counter field in the machine by one and copying the resulting value.
- The length field is set to the number of bytes of storage occupied by the PSSA header and the static data following it.
- The reserved fields are set to binary 0.

A space pointer addressing the new activation entry is stored in the last activation entry pointer of the PSSA base entry, and the next available storage location in the PSSA base entry is set to address the next available 16-byte aligned area beyond the new activation entry.

If the referenced program's activation already exists within the process PSSA chain when the Activate Program instruction is executed, the program's static storage is reused if the activation was active, and may or may not be reused if the activation was inactive. In either case, the storage is reinitialized, the activation is set to the active state, and the operand 1 space pointer is set to the reinitialized activation. No chain pointers are modified, and the activation entry remains at the same relative location in the chain of PSSA entries.

When a new activation is allocated or an existing inactive allocation is reactivated, the mark counter in the machine is incremented by 1 and the resulting value is copied to the active mark field of the activation. If an attempt is made to activate an already active activation, the activation mark and mark counter values are not updated.

When a new activation is allocated, space occupied by other activations in the inactive state may be used for the new activation. The current PSSA space is

the space located by the next available location pointer within the PSSA base entry.

PSSA entries that have all the following conditions are removed from the PSSA chain:

- Inactive
- Reside in the current PSSA space
- Have an invocation count of 0
- Have no active activations or activations with a nonzero invocation count at a higher address in the current PSSA space
- Appear as the last entries in the linked PSSA chain

The new activation is placed at the lowest address within the current PSSA space that is higher than both the address of any activation in the chain which is in the current PSSA space and the address of any unallocated space between previously existing noncontiguous activations. If no previous activations remain in the current PSSA space (after being removed under the above conditions), the new activation is placed at the lowest address (in the current PSSA space) of the removed activations. If no previous activations existed in the current PSSA space, the next available location pointer in the PSSA base entry specifies the location where the new activation is to be allocated.

If the program addressed by the operand 2 system pointer addresses a program that requires no static storage, no activation entry is allocated, and the operand 2 system pointer is copied to the operand 1 pointer.

A space pointer machine object may not be specified for operand 1.

Authorization Required

- Operational
  - Program referenced by operand 2
- Retrieve
  - Contexts referenced for address resolution

Lock Enforcement

- Materialize
  - Contexts referenced for address resolution

Exceptions

Exception		Operands 1	Operands 2	Other
06	Addressing			
	01 space addressing violation	X	X	
	02 boundary alignment	X	X	
	03 range	X	X	
	06 optimized addressability invalid	X	X	
0A	Authorization			
	01 unauthorized for operation		X	

Exception		Operands 1	2	Other
10	Damage encountered			
	04 system object damage state	X	X	X
	05 authority verification terminated due to damaged object			X
	44 partial system object damage	X	X	X
1A	Lock state			
	01 invalid lock state		X	
1C	Machine-dependent exception			
	03 machine storage limit exceeded			X
20	Machine support			
	02 machine check			X
	03 function check			X
22	Object access			
	01 object not found	X	X	
	02 object destroyed	X	X	
	03 object suspended	X	X	
	07 authority verification terminated due to destroyed object			X
24	Pointer specification			
	01 pointer does not exist		X	
	02 pointer type invalid	X	X	
	03 pointer addressing invalid object		X	
2A	Program creation			
	06 invalid operand type	X	X	
	07 invalid operand attribute	X	X	
	08 invalid operand value range	X	X	
	0C invalid operand odt reference	X	X	
	0D reserved bits are not zero	X	X	X
2C	Program execution			
	03 stack control invalid			X
2E	Resource control limit			
	01 user profile storage limit exceeded			X
36	Space management			
	01 space extension/truncation			X

6.2 Call External (CALLX)

Op Code (Hex)	Operand 1	Operand 2	Operand 3
0283	Program to be called or Call Template	Argument list	Return list

Operand 1: System pointer or Space Pointer Data Object.

Operand 2: Operand list or null.

Operand 3: Instruction definition list or null.

Warning: The following information is subject to change from release to release. Use it with caution and be prepared to adjust for changes with each new release.

Description: The instruction preserves the calling invocation and causes control to be passed to the external entry point of the program specified by operand 1.

Operand 1 may be specified as a system pointer which directly addresses the program that is to receive control or as a space pointer to a call template which identifies the program to receive control. Specifying a template allows for additional controls over how the specified program is to be invoked. The format of the call template is the following:

- Call options — Char(4)
  - Suppress adopted user profiles — Bit 0

    0 = no
    1 = yes
  - Reserved (binary zero) — Bit 1-30
  - Force program state to user state for call — Bit 31

    0 = no
    1 = yes
- Reserved (binary zero) — Char(12)
- Program to be called — System Pointer

The suppress adopted user profiles call option specifies whether or not the program adopted and propagated user profiles which may be serving as sources of authority to the process are to be suppressed from supplying authority to the new invocation. Specifying yes causes the propagation of adopted user profiles to be stopped as of the calling invocation, thereby, not allowing the called invocation to benefit from their authority. Specifying no allows the normal propagation of adopted and propagated user profiles to occur. The called program may adopt its owning user profile, if necessary, to supplement the authority available to its invocation.

The force program state to user state option specifies whether or not the call needs to be done in the current program state or as though the calling program

were running in the user state without the calling program changing to run in the user state.

The instruction ensures that the program is properly activated in the process, if required. The following conditions are allowed:

- If the referenced program requires no static storage, the program is invoked, and no activation is created.
- If operand 1 is a system pointer to a program that requires static storage, the program is implicitly activated. The chain of activation entries located by the PSSA (process static storage area) is searched for an entry for the referenced program. If an entry is located that is not active, it is set to the active state, and the static storage is reinitialized based on the program definition. If no activated entry exists for the program, a new entry is allocated and initialized. See the Activate Program instruction for a definition of this function. The activation mark value for a newly created activation will be the same as the invocation mark value described later.

After any needed static storage has been allocated or located, automatic storage is allocated and initialized for the newly invoked program. The automatic storage is obtained from the PASA (process automatic storage area).

The update PASA stack program attribute specified on program creation indicates whether or not the program requires that the PASA stack information contained in the PASA base entry and invocation entries must be updated. Refer to the Create Program instruction for the detail on how to specify this program attribute. Upon invocation of a program that requires that the stack be updated, it is possible that prior invocations may exist that did not require the stack update. These invocations would not have their associated stack information updated to reflect the current chain of invocations active in the PASA. If necessary, the PASA stack information in the PASA base entry and all prior invocation entries is updated with the current status prior to continuing with the invocation of a program requiring update of the PASA stack.

The PASA is located by a space pointer specified when the process is initiated. The location identified by the space pointer is considered to be the beginning of the PASA and must be 16-byte aligned. The PASA base entry must be initialized by the user before the process is initiated. The current invocation entry in process and next available storage location and mark counter values are accessed as input to the machine only during the initiation of the process. Thereafter, the machine maintains these values internally. The PASA base entry fields are optionally updated on each program invocation depending upon whether or not the program being invoked has specified the update PASA stack program attribute.

A space pointer locating the PASA can be materialized by using the Materialize Process instruction.

A space pointer locating the PASA invocation entry for the currently executing program can be materialized using the Materialize Invocation Entry instruction.

The program is invoked by allocating an area in the PASA space sufficient to contain the invocation entry. The area used for allocation is located by the next available storage location pointer in the PASA base entry for the invocation of the initial program in the process. For all other invocations of programs within

the process, the area used for the allocation is located by an internal machine value that is maintained with the space address of the next available storage location. This pointer must address a 16-byte aligned area in the space, or a boundary alignment exception is signaled. If the space is not currently large enough to contain the entry and if it is extendable, it is implicitly extended by the machine. The owner's authority to the space is included with the authority of the process when checking for object management authority when the space is extended. If the space is of a fixed size or cannot be extended enough to contain the entry, a space extension/truncation exception is signaled.

For programs created with the update PASA stack attribute specifying that they require the PASA stack update, the new invocation entry is updated as follows:

- The previous invocation entry pointer is set from the current invocation entry in the process address value in the machine.
- The next invocation entry is not modified.
- The associated program pointer is copied from the operand 1 system pointer.
- The invocation number is incremented by 1 beyond that in the calling invocation. The first invocation in the current process state has an invocation number of 1.
- The invocation type value is set to hex 01 to indicate how the program was invoked.
- The value of the mark counter in the machine is incremented by one and the new value is copied to the invocation mark field. The new value is also copied to the activation mark field of the program's activation if the activation was initialized by this instruction.
- The user area field is set to binary 0.
- The program's automatic storage is initialized as defined in the program definition.
- The invocation count, if any, in the associated activation is incremented by 1.

For programs created with the update PASA stack attribute specifying that they do not require the PASA stack update, the new invocation entry is updated as follows:

- The value of the mark counter in the machine is incremented by one. The new value is also copied to the activation mark field of the program's activation if the activation was initialized by this instruction.
- PASA stack information necessary to provide for subsequent program invocations or updating of stack information for this invocation is stored in the machine. This includes values associated with this invocation for the previous invocation entry address, next available storage location, program pointer, invocation number, invocation type, and mark counter.
- The program's automatic storage is initialized as defined in the program definition.

For programs created with the update PASA stack attribute that specifies that they require the PASA stack update, a space pointer addressing the new invocation entry is stored in the next invocation entry pointer of the invoking invocation.

For programs created with the update PASA stack attribute that specifies that they require the PASA stack update, a space pointer addressing the new invocation entry is stored in the current invocation entry pointer of the PASA base entry and the next available storage location in the PASA base entry is set to address the next available 16-byte aligned area beyond the new invocation entry.

A program with no automatic data has a PASA entry created for it. The created PASA entry consists of only a stack control entry.

The user defines the invocation attribute entry. This entry is not used after the program is initialized.

Following the allocation and initialization of the invocation entry, control is passed to the invoked program.

Operand 2 specifies an operand list that identifies the arguments to be passed to the invocation entry to be called. If operand 2 is null, no arguments are passed by the instruction. A parameter list length exception is signaled if the number of arguments passed does not correspond to the number required by the parameter list of the target program.

Operand 3 specifies an IDL (instruction definition list) that identifies the instruction number(s) of alternate return points within the calling invocation. A Return External instruction in an invocation immediately subordinate to the calling invocation can indirectly reference a specific entry in the IDL to cause a return of control to the instruction associated with the referenced IDL entry. If operand 3 is null, then the calling invocation has no alternate return points associated with the call.

Authorization Required

- Operational
  - Program referenced by operand 1
- Retrieve
  - Contexts referenced for address resolution

Lock Enforcement

- Materialize
  - Contexts referenced for address resolution

Exceptions

Exception		Operands 1	2	3	Other
06	Addressing				
	01 space addressing violation	X			
	02 boundary alignment	X			
	03 range	X			
	06 optimized addressability invalid	X			
08	Argument/parameter				
	01 parameter reference violation	X			
	02 parameter list length violation		X		

Exception		Operands 1	2	3	Other
0A	Authorization				
	01 unauthorized for operation	X			
10	Damage encountered				
	04 system object damage state	X	X	X	X
	05 authority verification terminated due to damaged object				X
	44 partial system object damage	X	X	X	X
1A	Lock state				
	01 invalid lock state	X			
1C	Machine-dependent exception				
	03 machine storage limit exceeded				X
20	Machine support				
	02 machine check				X
	03 function check				X
22	Object access				
	01 object not found	X			
	02 object destroyed	X			
	03 object suspended	X			
	07 authority verification terminated due to destroyed object				X
24	Pointer specification				
	01 pointer does not exist	X			
	02 pointer type invalid	X			
	03 pointer addressing invalid object	X			
2A	Program creation				
	06 invalid operand type	X	X	X	
	07 invalid operand attribute	X			
	08 invalid operand value range	X			
	0C invalid operand odt reference	X	X	X	
	0D reserved bits are not zero	X	X	X	X
2C	Program execution				
	03 stack control invalid				X
2E	Resource control limit				
	01 user profile storage limit exceeded				X
36	Space management				
	01 space extension/truncation				X
38	Template specification				

Exception	Operands 1	2	3	Other
01 template value invalid	X			

6.3 Call Internal (CALLI)

Op Code (Hex)	Operand 1	Operand 2	Operand 3
0293	Internal entry point	Argument list	Return target

Operand 1: Internal entry point.

Operand 2: Operand list or null.

Operand 3: Instruction pointer.

Warning: The following information is subject to change from release to release. Use it with caution and be prepared to adjust for changes with each new release.

Description: The internal entry point specified by operand 1 is located in the same invocation in which the Call Internal instruction is executed. A subinvocation is defined, and execution control is transferred to the first instruction associated with the internal entry point. The instruction does not cause a new invocation to be established. Therefore, there is no allocation of objects, and instructions in the subinvocation have access to all invocation objects.

Operand 2 specifies an operand list that identifies the arguments to be passed to the subinvocation. If operand 2 is null, no arguments are passed. After an argument has been passed on a Call Internal instruction, the corresponding parameter may be referenced. This causes an indirect reference to the storage area located by the argument. This mapping exists until the parameter is assigned a new mapping based on a subsequent Call Internal instruction. A reference to an internal parameter before its being assigned an argument mapping causes a parameter reference violation exception to be signaled.

Operand 3 specifies an instruction pointer that identifies the pointer into which the machine places addressability to the instruction immediately following the Call Internal instruction. A branch instruction in the called subinvocation can directly reference this instruction pointer to cause control to be passed back to the instruction immediately following the Call Internal instruction.

Exceptions

Exception		Operands 1	2	3	Other
06	Addressing				
	01 space addressing violation			X	
	02 boundary alignment			X	
	03 range			X	
	06 optimized addressability invalid			X	
08	Argument/parameter				
	01 parameter reference violation		X		
10	Damage encountered				

Exception		Operands 1	2	3	Other
	04 system object damage state	X	X	X	X
	44 partial system object damage	X	X	X	X
1C	Machine-dependent exception				
	03 machine storage limit exceeded				X
20	Machine support				
	02 machine check				X
	03 function check				X
24	Pointer specification				
	01 pointer does not exist			X	
	02 pointer type invalid			X	
2A	Program creation				
	06 invalid operand type	X	X	X	
	09 invalid branch target			X	
	0B invalid number of operands		X		
	0C invalid operand odt reference	X	X	X	
	0D reserved bits are not zero	X	X	X	X
2E	Resource control limit				
	01 user profile storage limit exceeded				X
36	Space management				
	01 space extension/truncation				X

6.4 Clear Invocation Exit (CLRIEXIT)

Op Code (Hex)
0250

Description: The instruction removes the invocation exit program for the requesting invocation. No exception is signaled if an invocation exit program is not specified for the current invocation. Also, an implicit clear of the invocation exit occurs when the invocation exit program is given control, or the program which set the invocation exit completes execution.

Exceptions

Exception		Other
10	Damage encountered	
	04 System object damage state	X
	44 Partial system object damage	X
1C	Machine-dependent exception	
	03 Machine storage limit exceeded	X
20	Machine support	
	02 Machine check	X
	03 Function check	X
2A	Program creation	
	0D Reserved bits are not zero	X
2E	Resource control limit	
	01 user profile storage limit exceeded	X
36	Space management	
	01 space extension/truncation	X

6.5 De-Activate Program (DEACTPG)

Op Code (Hex)	Operand 1
0225	Program

Operand 1: System pointer or null.

Description: The instruction locates the activation entry addressed through operand 1 and marks it as inactive if the appropriate conditions are satisfied.

If operand 1 is null, the program issuing the instruction is to be de-activated. An activation in use by invocation exception is signaled if the activation entry's invocation count is not equal to 1.

If operand 1 is a system pointer to a program, then that program's activation entry is de-activated if its invocation count is 0. Otherwise, an activation in use by invocation exception is signaled.

In the previous two cases, if the program has no static storage or no activation, no operation is performed and no exception is signaled.

The activation is de-activated when the activation status is set to not currently active (0). When the activation is not active and its invocation count is 0, the storage occupied by the activation is subject to reuse for allocating other activations.

Authorization Required

- Retrieve
  - Contexts referenced for address resolution

Lock Enforcement

- Materialize
  - Contexts referenced for address resolution

Exceptions

Exception		Operand 1	Other
06	Addressing		
	01 space addressing violation	X	
	02 boundary alignment	X	
	03 range	X	
	06 optimized addressability invalid	X	
08	Argument/parameter		
	01 parameter reference violation	X	
0A	Authorization		
	01 unauthorized for operation	X	
10	Damage encountered		
	04 system object damage state	X	X

Exception		Operand 1	Operand Other
	05 authority verification terminated due to damaged object		X
	44 partial system object damage	X	X
1A	Lock state		
	01 invalid lock state	X	
20	Machine support		
	02 machine check		X
	03 function check		X
22	Object access		
	01 object not found	X	
	02 object destroyed	X	
	03 object suspended	X	
	07 authority verification terminated due to destroyed object		X
24	Pointer specification		
	01 pointer does not exist	X	
	02 pointer type invalid	X	
	03 pointer addressing invalid object	X	
2A	Program creation		
	06 invalid operand type	X	
	07 invalid operand attribute	X	
	0A invalid operand value range	X	
	0C invalid operand odt reference	X	
	0D reserved bits are not zero	X	X
2C	Program execution		
	03 stack control invalid		X
	05 activation in use by invocation	X	
2E	Resource control limit		
	01 user profile storage limit exceeded		X
32	Scalar specification		
	01 scalar type invalid	X	
36	Space management		
	01 space extension/truncation		X

6.6 End (END)

Op Code (Hex)
0260

No operands are specified.

Description: The instruction delimits the end of a program's instruction stream. When this instruction is encountered in execution, it causes a return to the preceding invocation (if present) or causes termination of the process phase if the instruction is executed in the highest-level invocation for a process. The End instruction delineates the end of the instruction stream. When it is encountered in execution, the instruction functions as a Return External instruction with a null operand. Refer to the Return External instruction for a description of that instruction.

Exceptions

Exception		Other
10	Damage encountered	
	04 system object damage state	X
	44 partial system object damage	X
1C	Machine-dependent exception	
	03 machine storage limit exceeded	X
20	Machine support	
	02 machine check	X
	03 function check	X
2A	Program creation	
	0D reserved bits are not zero	X
2E	Resource control limit	
	01 user profile storage limit exceeded	X
36	Space management	
	01 space extension/truncation	X

6.7 Modify Automatic Storage Allocation (MODASA)

Op Code (Hex)	Operand 1	Operand 2
02F2	Storage allo-cation	Modifica-tion size

Operand 1: Space pointer data object or null.

Operand 2: Signed binary scalar.

Description: The size of automatic storage assigned to the invocation of the currently executing program is extended or truncated by the size specified by operand 2. A positive value indicates that the storage allocation is to be extended; a negative value indicates that the storage allocation is to be truncated. The instruction also returns addressability of the allocated or deallocated storage area in the space pointer identified by operand 1. When allocating additional space, the space pointer locates the first byte of the allocated area. If space is deallocated, the space pointer locates the first byte of the deallocated area. If operand 1 is null, the storage is allocated or deallocated but no addressability is returned. The space pointer identified by operand 1 always addresses storage that is on a 16-byte boundary.

This instruction modifies the next available storage location address value in the machine. Additionally, if the program executing this instruction specified the program requires PASA stack update program attribute, the instruction modifies the next available storage location pointer in the PASA (process automatic storage area) base entry.

The owner's authority to the space is included with the authority of the process when a space is extended and when checked for object management authority.

If the space is extended, the new bytes contain the initial value for the space; otherwise, no initialization is done to the allocated area.

A space extension/truncation exception is signaled if the space containing the PASA cannot be extended. A scalar value invalid exception is signaled if truncation causes the next available storage location pointer in the PASA to point to a location that precedes the beginning of the data of the automatic storage entry for the executing invocation.

The storage allocated with this instruction is not initialized to any value. If implicit space extension occurs, however, the extended portion is initialized to the default value specified for the space when it was created.

A space pointer machine object cannot be specified for operand 1.

Exceptions

Exception		Operands 1	Operands 2	Other
06	Addressing			
	01 space addressing violation	X	X	

Exception		Operands 1	2	Other
	02 boundary alignment	X	X	
	03 range	X	X	
	06 optimized addressability invalid	X	X	
08	Argument/parameter			
	01 parameter reference violation	X	X	
10	Damage encountered			
	04 system object damage state	X	X	X
	44 partial system object damage	X	X	X
1C	Machine-dependent exception			
	03 machine storage limit exceeded			X
	04 object storage limit exceeded			X
20	Machine support			
	02 machine check			X
	03 function check			X
22	Object access			
	01 object not found	X	X	
	02 object destroyed	X	X	
	03 object suspended	X	X	
24	Pointer specification			
	01 pointer does not exist	X	X	
	02 pointer type invalid	X	X	
2A	Program creation			
	06 invalid operand type	X	X	
	07 invalid operand attribute	X	X	
	08 invalid operand value range	X	X	
	0A invalid operand length	X	X	
	0C invalid operand odt reference	X	X	
	0D reserved bits are not zero	X	X	X
2C	Program execution			
	03 stack control invalid			X
2E	Resource control limit			
	01 user profile storage limit exceeded			X
32	Scalar specification			
	01 scalar type invalid	X	X	
	02 scalar attributes invalid		X	
	03 scalar value invalid		X	

Exception		Operands 1	2	Other
36	Space management			
	01 space extension/truncation			X

6.8 Return External (RTX)

Op Code (Hex)	Operand 1
02A1	Return point

Operand 1: Signed binary (2) scalar or null.

Description: The instruction terminates execution of the invocation in which the instruction is specified. All automatic program objects in the invocation are destroyed by removing the returning program's automatic storage from the PASA (process automatic storage area) by the updating of the PASA chaining pointers.

A Return External instruction can be specified within an invocation's subinvocation, and no exception is signaled.

If a higher invocation exists in the invocation hierarchy, the instruction causes execution to resume in the preceding invocation in the process' invocation hierarchy at an instruction location indirectly specified by operand 1. If operand 1 is binary 0 or null, the next instruction following the Call External instruction from which control was relinquished in the preceding invocation in the hierarchy is given execution control. If the value of operand 1 is not 0, the value represents an index into the IDL (instruction definition list) specified as the return list operand in the Call External instruction, and the value causes control to be passed to the instruction referenced by the corresponding IDL entry. The first IDL entry is referenced by a value of one. If operand 1 is not 0 and no return list was specified in the Call External instruction, or if the value of operand 1 exceeds the number of entries in the IDL, or if the value is negative, a return point invalid exception is signaled.

If the prior invocation is for a program created with the update PASA stack attribute specifying that it requires the PASA stack update, the instruction sets the current invocation entry in the PASA base entry to address the immediately preceding invocation, and it also sets addressability to the returning invocation into the next available storage location entry in the PASA header.

If the prior invocation is for a program created with the update PASA stack attribute specifying that it does not require the PASA stack update, the instruction only updates internal machine values related to the invocation stack.

If a higher invocation does not exist, the Return External instruction causes termination of the current process state. If operand 1 is not 0 and is not null, the return point invalid exception is signaled. Refer to the Terminate Process instruction for the functions performed in process termination.

If the returning invocation has received control to process an event, then control is returned to the point where the event handler was invoked. In this case, if operand 1 is not 0 and is not null, then a return point invalid exception is signaled.

If the returning invocation has received control from the machine to process an exception, the return instruction invalid exception is signaled.

If the returning invocation has an activation, the invocation count in the activation is decremented by 1.

If the returning invocation currently has an invocation exit set, the invocation exit is not given control and is implicitly cleared.

Exceptions

Exception		Operand 1	Other
06	Addressing		
	01 space addressing violation	X	
	02 boundary alignment	X	
	03 range	X	
	06 optimized addressability invalid	X	
08	Argument/parameter		
	01 parameter reference violation		X
10	Damage encountered		
	04 system object damage state	X	X
	44 partial system object damage	X	X
1C	Machine-dependent exception		
	03 machine storage limit exceeded	X	
20	Machine support		
	02 machine check		X
	03 function check		X
22	Object access		
	01 object not found	X	
	02 object destroyed	X	X
	03 object suspended	X	
24	Pointer specification		
	01 pointer does not exist	X	X
	02 pointer type invalid	X	X
2A	Program creation		
	06 invalid operand type	X	
	07 invalid operand attribute	X	
	08 invalid operand value range	X	
	0A invalid operand length	X	
	0C invalid operand odt reference	X	
	0D reserved bits are not zero	X	X
2C	Program execution		
	01 return instruction invalid		X
	02 return point invalid	X	

Exception		Operand 1	Other
2E	Resource control limit		
	01 user profile storage limit exceeded		X
36	Space management		
	01 space extension/truncation		X

6.9 Set Argument List Length (SETALLEN)

Op Code (Hex)	Operand 1	Operand 2
0242	Argu-ment list	Length

Operand 1: Operand list.

Operand 2: Binary scalar.

Description This instruction specifies the number of arguments to be passed on a succeeding Call External or Transfer Control instruction. The current length of the variable-length operand list (used as an argument list) specified by operand 1 is modified to the value indicated in the binary scalar specified by operand 2. This length value specifies the number of arguments (starting from the first) to be passed from the list when the operand list is referenced on a Call External or Transfer Control instruction.

Only variable-length operand lists with the argument list attribute may be modified by the instruction.

The value in operand 2 may range from 0 (meaning no arguments are to be passed) to the maximum size specified in the ODT definition of the operand list (meaning all defined arguments are to be passed).

The length of the argument list remains in effect for the duration of the current invocation or until a Set Argument List Length instruction is issued against this operand list.

Exceptions

Exception		Operands 1	2	Other
06	Addressing			
	01 space addressing violation		X	
	02 boundary alignment		X	
	03 range		X	
	06 optimized addressability invalid		X	
08	Argument/parameter			
	01 parameter reference violation		X	
	03 argument list length modification violation	X		
10	Damage encountered			
	04 system object damage state	X	X	X
	44 partial system object damage	X	X	X
1C	Machine-dependent exception			
	03 machine storage limit exceeded	X		
20	Machine support			

Exception		Operands 1	2	Other
	02 machine check			X
	03 function check			X
22	Object access			
	01 object not found		X	
	02 object destroyed		X	
	03 object suspended		X	
24	Pointer specification			
	01 pointer does not exist		X	
	02 pointer type invalid		X	
2A	Program creation			
	06 invalid operand type	X	X	
	07 invalid operand attribute	X	X	
	08 invalid operand value range		X	
	0A invalid operand length		X	
	0C invalid operand odt reference	X	X	
	0D reserved bits are not zero	X	X	X
2E	Resource control limit			
	01 user profile storage limit exceeded			X
32	Scalar specification			
	03 scalar value invalid	X		
36	Space management			
	01 space extension/truncation			X

6.10 Set Invocation Exit (SETIEXIT)

Op Code (Hex)	Operand 1	Operand 2
0252	Invocation exit program	Argument list

Operand 1: System pointer.

Operand 2: Operand list or null.

Description: This instruction allows the external entry point of the program specified by operand 1 to be given control when the requesting invocation is destroyed due to normal exception handling actions, or due to any process termination. Normal exception handling actions are considered to be those actions performed by the Return From Exception or the Signal Exception instructions.

Operand 1 is a system pointer addressing the program that is to receive control. The operand 1 system pointer must be in either the static or automatic storage of the program invoking this instruction.

Operand 2 specifies an operand list that identifies the arguments to be passed to the invocation exit program being called. If operand 2 is null, no arguments are passed to the invocation.

No operand verification takes place when this instruction is executed. Nor are copies made of the operands, so changes made to the operand values after execution of this instruction will be used during later operand verification. Operand verification occurs on the original form of the operands when the invocation exit program is invoked. At that time operational authorization to the invocation exit program and retrieve authorization to any contexts referenced for materialization take place. Also, materialization lock enforcement occurs to contexts referenced for materialization.

If an invocation exit program currently exists for the requesting invocation, it is replaced, and no exception is signaled. The invocation exit set by this instruction is implicitly cleared when the invocation exit program is given control, or the program which set the invocation exit completes execution.

If any invocations are to be destroyed due to normal exception handling actions, then those invocation exit programs associated with the invocations to be destroyed are given control before execution proceeds to the signaled exception handler.

An invocation exit bypassed due to a RTNEXCP or a SIGEXCP instruction event is signaled when both of the following conditions occur:

- Exception management is destroying an invocation stack due to a Signal Exception instruction, a Return From Exception instruction, or process termination.
- An invocation exit program is to be destroyed due to a second Signal Exception or a second Return From Exception instruction.

The invocation exit program that is being destroyed is terminated, and its associated invocation execution is terminated. Termination of invocations due to a previous Signal Exception instruction, a Return From Exception instruction, or a process termination is then resumed.

If a process phase is terminated and the process was not in termination phase, then the invocations are terminated. Invocation exit programs set for the terminated invocations are allowed to run. If an invocation to be terminated is an invocation exit program, then the following occurs:

- An invocation exit bypassed due to process termination event is signaled.
- If an invocation exit has been set for this invocation exit, it is allowed to run.
- The invocation exit is terminated and the associated invocation is terminated (the invocation exit is not reinvoked).

Invocation exit programs for the remaining invocations to be terminated are then allowed to run.

Exceptions

Exception		Operands 1	2	3	4	Other
06	Addressing					
	01 space addressing violation	X				
	02 boundary alignment	X				
	03 range	X				
	06 optimized addressability invalid	X				
08	Argument/parameter					
	01 parameter reference violation	X				
10	Damage encountered					
	04 system object damage state					X
	44 partial system object damage					X
1C	Machine-dependent exception					
	03 machine storage limit exceeded					X
20	Machine support					
	02 machine check					X
	03 function check					X
2A	Program creation					
	06 invalid operand type	X	X			
	07 invalid operand attribute	X				
	08 invalid operand value range	X				
	0C invalid operand odt reference	X	X			
	0D reserved bits are not zero	X	X	X	X	X
2E	Resource control limit					
	01 user profile storage limit exceeded					X

Exception		Operands 1	2	3	4	Other
32	Scalar specification					
	01 scalar type invalid	X				
36	Space management					
	01 space extension/truncation					X

6.11 Store Parameter List Length (STPLLEN)

Op Code (Hex)	Operand 1
0241	Length

Operand 1: Binary variable scalar.

Description: A value is returned in operand 1 that represents the number of parameters associated with the invocation's external entry point for which arguments have been passed on the preceding Call External or Transfer Control instruction.

The value can range from 0 (no parameters were received) to the maximum size possible for the parameter list associated with the external entry point.

Exceptions

Exception		Operand 1	Other
06	Addressing		
	01 space addressing violation	X	
	02 boundary alignment	X	
	03 range	X	
	06 optimized addressability invalid	X	
08	Argument/parameter		
	01 parameter reference violation	X	
10	Damage encountered		
	04 system object damage state	X	X
	44 partial system object damage	X	X
1C	Machine-dependent exception		
	03 machine storage limit exceeded	X	
20	Machine support		
	02 machine check		X
	03 function check		X
22	Object access		
	01 object not found	X	
	02 object destroyed	X	
	03 object suspended	X	
24	Pointer specification		
	01 pointer does not exist	X	
	02 pointer type invalid	X	
2A	Program creation		
	06 invalid operand type	X	
	07 invalid operand attribute	X	

Exception		Operand 1	Other
	08 invalid operand value range	X	
	0A invalid operand length	X	
	0C invalid operand odt reference	X	
	0D reserved bits are not zero	X	X
2E	Resource control limit		
	01 user profile storage limit exceeded		X
32	Scalar specification		
	01 scalar type invalid	X	
	02 scalar attributes invalid	X	
36	Space management		
	01 space extension/truncation		X

6.12 Transfer Control (XCTL)

Op Code (Hex)	Operand 1	Operand 2
0282	Program to be called or Call template	Argument list

Operand 1: System pointer or Space pointer Data Object.

Operand 2: Operand list or null.

Description: The instruction destroys the calling invocation and causes control to be passed to the external entry point of the program specified by operand 1.

Operand 1 may be specified as a a system pointer which directly addresses the program that is to receive control or as a space pointer to a call template which identifies the program to receive control. Specifying a template allows for additional controls over how the specified program is to be invoked. The format of the call template is the following:

- Call options — Char(4)
  - Suppress adopted user profiles — Bit 0

    0 = no
    1 = yes
  - Reserved (binary zero) — Bit 1-30
  - Force program state to user state for transfer — Bit 31

    0 = no
    1 = yes
- Reserved (binary zero) — Char(12)
- Program to be called — System Pointer

The suppress adopted user profiles call option specifies whether or not the program adopted and propagated user profiles which may be serving as sources of authority to the process are to be suppressed from supplying authority to the new invocation. Specifying yes causes the propagation of adopted user profiles to be stopped as of the calling invocation, thereby, not allowing the called invocation to benefit from their authority. Specifying no allows the normal propagation of adopted and propagated user profiles to occur. The called program may adopt its owning user profile, if necessary, to supplement the authority available to its invocation.

The force program state to user state option specifies whether or not the transfer control needs to be done in the current program state or as though the transfering program were running in the user state without the it changing to run in the user state.

The invocation count in the activation (if any) of the calling program is decremented by 1. The instruction ensures that the called program is properly activated in the process, if required. See the Activate Program instruction for a definition of this activation verification process.

After any needed static storage has been allocated or located, the invocation entry to the program issuing the Transfer Control instruction is made available for the new invocation. Unless precluded by internal machine alignment requirements, the new invocation's stack control entry and automatic storage overlay that of the invocation issuing the Transfer Control instruction. The new invocation entry is allocated beginning at the same location as that of the current (transferring) invocation. See the Call External instruction for a definition of a PASA (process automatic storage area) entry.

The new invocation's stack control entry is initialized as follows:

- The previous invocation entry pointer and the next invocation entry pointer are the same as that of the invoking program's entry.
- The associated program pointer is copied from the associated activation entry (or from the operand 1 system pointer if no activation entry exists).
- The invocation number entry is unchanged.
- The invocation type value is set to indicate that the program was invoked via a Transfer Control instruction (hex 20).
- The program's automatic storage is allocated and initialized as specified in the program definition.

The invocation entry for the preceding invocation is unchanged by the instruction. The current invocation entry pointer in the PASA base entry is unchanged by the instruction. The next available storage location entry in the PASA base entry is set to address the next available 16-byte aligned area beyond the new invocation entry.

The program is invoked by allocating an area in the PASA space that is sufficient to contain the invocation entry. The area used for allocation is located by the next available storage location pointer in the PASA base entry. This pointer must address a 16-byte aligned area in the space, or a boundary alignment exception is signaled.

The maximum addressable location in the PASA space limits the amount of storage that may be allocated for PASA storage. If this limit is exceeded, the process storage limit exceeded exception is signaled. If the maximum addressable location entry does not address the same space as that addressed by the next available storage location entry, the stack control invalid exception is signaled.

If insufficient space is available in the PASA for the entire new entry, the PASA space is implicitly extended by the machine. If the space is fixed size or may not be extended enough to contain the entry, a space extension/truncation exception is signaled.

Following the allocation and initialization of automatic storage, control is passed to the invoked program.

Operand 2 specifies an operand list that identifies the arguments to be passed to the invocation to which control is being transferred. Automatic objects allocated by the transferring invocation are destroyed as a result of the transfer operation and, therefore, cannot be passed as arguments. A parameter list length excep-

tion is signaled if the number of arguments passed does not correspond to the number required by the parameter list of the target program.

If the transferring invocation has received control to process an exception, an event, or an invocation exit, the return instruction invalid exception is signaled.

If the transferring invocation currently has an invocation exit set, the invocation exit is not given control and is implicitly cleared.

Authorization Required

- Operand 1
  - Operational
- Retrieve
  - Contexts referenced for address resolution

Lock Enforcement

- Materialize
  - Contexts referenced for address resolution

Exceptions

Exception		Operands 1	2	Other
06	Addressing			
	01 space addressing violation	X		
	02 boundary alignment	X		
	03 range	X		
	06 optimized addressability invalid	X		
08	Argument/parameter			
	01 parameter reference violation			X
	02 parameter list length violation		X	
0A	Authorization			
	01 unauthorized for operation	X		
10	Damage encountered			
	04 system object damage state	X	X	X
	05 authority verification terminated due to damaged object			X
	44 partial system object damage	X	X	X
1A	Lock state			
	01 invalid lock state	X		
1C	Machine-dependent exception			
	02 program limitation exceeded			X
	03 machine storage limit exceeded			X
20	Machine support			

Exception		Operands 1	2	Other
	02 machine check			X
	03 function check			X
22	Object access			
	01 object not found	X		
	02 object destroyed	X		
	03 object suspended	X		
	07 authority verification terminated due to destroyed object			X
24	Pointer specification			
	01 pointer does not exist	X		
	02 pointer type invalid	X		
	03 pointer addressing invalid object	X		
2A	Program creation			
	06 invalid operand type	X	X	
	07 invalid operand attribute	X	X	
	08 invalid operand value range	X		
	0C invalid operand odt reference	X	X	
	0D reserved bits are not zero	X	X	X
2C	Program execution			
	01 return instruction invalid			X
	03 stack control invalid			X
2E	Resource control limit			
	01 user profile storage limit exceeded			X
36	Space management			
	01 space extension/truncation			X
38	Template specification			
	01 template value invalid	X		

Chapter 7. Program Creation Control Instructions

This chapter describes all the instructions used to control the create program function. These instructions are arranged in alphabetic order. For an alphabetic summary of all the instructions, see Appendix A, "Instruction Summary."

7.1 No Operation (NOOP)

Op Code (Hex)
0000

Description: No function is performed. The instruction consists of an operation code and no operands. The instruction may not be branched to and is not counted as an instruction in the instruction stream. The instruction may be used for inserting gaps in the instruction stream. These gaps allow instructions with adjacent instruction addresses to be physically separated.

The instruction may precede or follow any machine instruction except the End instruction, and any number of No Operation instructions may exist in succession.

7.2 No Operation and Skip (NOOPS)

Op Code (Hex)	Operand 1
0001	Skip count

Operand 1: Unsigned immediate value.

Description: This instruction performs no function other than to indicate a specific number of bytes within the instruction stream that are to be skipped during encapsulation. It consists of an operation code and 1 operand. Operand 1 is an unsigned immediate value that contains the number of bytes between this instruction and the next instruction to be processed. These bytes are skipped during the encapsulation of this program. A value of zero for operand 1 indicates that no bytes are to be skipped between this instruction and the next instruction to be processed.

If the operand 1 skip count indicates that the next instruction to process is beyond the end of the instruction stream, an invalid operand value range exception is signaled.

This instruction may be used to insert gaps in the instruction stream in such a manner that allows instructions with adjacent instruction addresses to not be physically adjacent.

This instruction may not be branched to, and is not counted as an instruction in the instruction stream.

The instruction may precede or follow any machine instruction except the End instruction, and any number of No Operation and Skip instructions may exist in succession.

Note: When this instruction is used in an existing program template, the following items within the template may be adversely affected:

- The actual count of instructions may be altered to be different than the count of instructions that is specified in the program template header.
- Object definitions that reference instructions may now be out of range or may not reference the intended instruction.

The actual number of bytes skipped includes the bytes containing the instruction plus the number of bytes specified by the skip count value. The number of bytes skipped per template version is as follows:

- Version 0 = 4 plus the skip count value.
- Version 1 = 5 plus the skip count value.

Exceptions

Exception		Operand 1	Other
2A	Program Creation		
	06 Invalid operand type	X	
	08 Invalid operand value range	X	

Exception	Operand 1	Operand Other
0D Reserved bits are not zero	X	X

7.3 Override Program Attributes (OVRPGATR)

Op Code (Hex)	Operand 1	Operand 2
0006	Attribute identification	Attribute modifier

Operand 1: Unsigned immediate value.

Operand 2: Unsigned immediate value.

Description: This program creation control instruction allows one of a set of program attributes specified below to be overriden. The overriden program attribute is in effect until it is changed by another OVRPGATR instruction. The initial program attributes are set to the ones given in the program template for the CRTPG instruction. These same initial program attributes are the ones that are materialized when a MATPG is done. That is, the OVRPGATR instruction has no effect on the materialized attributes.

The OVRPGATR instruction consists of an operation code and two operands. Operand 1 is an unsigned immediate value that contains a representation of which program attribute is to be overriden. Operand 2 is an unsigned immediate value that contains a representation of how the program attribute is to be overriden.

This instruction may not be branched to, and is not counted as an instruction in the instruction stream.

The instruction may precede or follow any machine instruction.

The program attributes defined by operand 1 is overriden according to the following selection values:

Attribute Identification	Attribute Description
1	Array constrainment attribute Allowed values for operand 2: 1 = Constrain array references 2 = Do not constrain array references 3 = Fully unconstrain array references 4 = Terminate override of array constrainment attributes and resume use of the attributes specified in the program template
2	String constrainment attribute Allowed values for operand 2: 1 = Constrain string references 2 = Do not constrain string references

3 = Terminate override of string constrainment attribute and resume use of the attribute specified in the program template

3 Suppress binary size exception attribute

Allowed values for operand 2:

1 = Suppress binary size exceptions

2 = Do not suppress binary size exceptions

3 = Terminate override of suppression of binary size exception attribute and resume use of the attribute specified in the program template

4 Suppress decimal data exception attribute

Allowed values for operand 2:

1 = Suppress decimal data exceptions

2 = Do not suppress decimal data exceptions

3 = Terminate override of suppression of decimal data exception attribute and resume use of the attribute specified in the program template

5 CPYBWP alignment data check attribute

Allowed values for operand 2:

1 = Constrain CPYBWP to require like alignment of operands (default)

2 = Do not constrain CPWBWP to require like alignment of operands

Exceptions

Exception		Operand 1	Other
2A	Program Creation		
	06 Invalid operand type	X	
	08 Invalid operand value range	X	
	0D Reserved bits are not zero	X	X

Chapter 8. Independent Index Instructions

This chapter describes the instructions used for indexes. These instructions are in alphabetic order. For an alphabetic summary of all the instructions, see Appendix A, "Instruction Summary."

8.1 Find Independent Index Entry (FNDINXEN)

Op Code (Hex)	Operand 1	Operand 2	Operand 3	Operand 4
0494	Receiver	Index	Option list	Search argument

Operand 1: Space pointer.

Operand 2: System pointer.

Operand 3: Space pointer.

Operand 4: Space pointer.

Warning: The following information is subject to change from release to release. Use it with caution and be prepared to adjust for changes with each new release.

Description: This instruction searches the independent index identified by operand 2 according to the search criteria specified in the option list (operand 3) and the search argument (operand 4); then it returns the desired entry or entries in the receiver field (operand 1). The maximum size of the independent index entry is 120 bytes.

The option list is a variable-length area that identifies the type of search to be performed, the length of the search argument(s), the number of resultant arguments to be returned, the lengths of the entries returned, and the offsets to the entries within the receiver identified by the operand 1 space pointer. The option list has the following format:

- Rule option — Char(2)
- Argument length — Bin(2)
- Argument offset — Bin(2)
- Occurrence count — Bin(2)
- Return count — Bin(2)

Each entry that is returned to the receiver operand contains the following:

- Entry length — Bin(2)
- Offset — Bin(2)

The rule option identifies the type of search to be performed and has the following meaning:

Search Type	Value (Hex)	Meaning
=	0001	Find equal occurrences of operand 4.
>	0002	Find occurrences that are greater than operand 4.
<	0003	Find occurrences that are less than operand 4.

Search Type	Value (Hex)	Meaning
≥	0004	Find occurrences that are greater than or equal to operand 4.
≤	0005	Find occurrences that are less than or equal to operand 4.
First	0006	Find the first index entry or entries.
Last	0007	Find the last index entry or entries.
Between	0008	Find all entries between the two arguments specified by operand 4 (inclusive).

The option to find between limits requires that operand 4 be a 2-element vector in which element 1 is the starting argument and element 2 is the ending argument. All arguments between (and including) the starting and ending arguments are returned, but the occurrence count specified is not exceeded.

If the index was created to contain both pointers and scalar data, then the search argument must be 16-byte aligned. For the option to find between limits, both search arguments must be 16-byte aligned.

The rule option and the argument length determine the search criteria used for the index search. The argument length must be greater than or equal to one. The argument length for fixed-length entries must be less than or equal to the argument length specified when the index is created.

The argument length entry specifies the length of the search argument (operand 4) to be used for the index search. When the rule option equals first or last, the argument length entry is ignored. For the option to find between limits, the argument length option specifies the lengths of one vector element. The lengths of the vector elements must be equal.

The argument offset is the offset of the second search argument from the beginning of the entire argument field (operand 4). The argument offset field is ignored unless the rule option is find between.

The occurrence count specifies the maximum number of index entries that satisfy the search criteria to be returned. This field is limited to a maximum value of 4095. If this value is exceeded, a template value invalid exception is signaled.

The return count specifies the number of index entries satisfying the search criteria that were returned in the receiver (operand 1). If this field is 0, no index arguments satisfied the search criteria.

There are two fields in the option list for each entry returned in the receiver (operand 1). The entry length is the length of the entry retrieved from the index. The offset has the following meaning:

- For the first entry, the offset is the number of bytes from the beginning of the receiver (operand 1) to the first byte of the first entry.

- For any succeeding entry, the offset is the number of bytes from the beginning of the immediately preceding entry to the first byte of the entry returned.

The entries that are retrieved as a result of the Find Independent Index Entry instruction are always returned starting with the entry that is closest to or equal to the search argument and then proceeding away from the search argument. For example, a search that is for < (less than) or ≤ (less than or equal to) returns the entries in order of decreasing value.

All the entries that satisfy the search criteria (up to the occurrence count) are returned in the space starting at the location designated by the operand 1 space pointer.

If the index was created to contain both pointers and scalar data, then each returned entry is 16-byte aligned.

If the index was created to contain only scalar data, then returned entries are contiguous.

Every entry retrieved causes the count of the find operations to be incremented by 1. The current value of this count is available through the Materialize Index Attributes instruction.

Authorization Required

- Retrieve
  - Operand 2
  - Contexts referenced for address resolution

Lock Enforcement

- Materialize
  - Operand 2
  - Contexts referenced for address resolution

Exceptions

Exception		Operands 1	2	3	4	Other
06	Addressing					
	01 space addressing violation	X	X	X	X	
	02 boundary alignment	X	X	X	X	
	03 range	X	X	X	X	
	06 optimized addressability invalid	X	X	X	X	
08	Argument/parameter					
	01 parameter reference violation	X	X	X	X	
0A	Authorization					
	01 unauthorized for operation		X			
10	Damage encountered					
	04 system object damage state	X	X	X	X	X

Exception		Operands 1	2	3	4	Other
	05 authority verification terminated due to damaged object					X
	44 partial system object damage	X	X	X	X	X
1A	Lock state					
	01 invalid lock state		X			
1C	Machine-dependent exception					
	03 machine storage limit exceeded					X
20	Machine support					
	02 machine check					X
	03 function check					X
22	Object access					
	01 object not found	X	X	X	X	
	02 object destroyed	X	X	X	X	
	03 object suspended	X	X	X	X	
	07 authority verification terminated due to destroyed object					X
24	Pointer specification					
	01 pointer does not exist	X	X	X	X	
	02 pointer type invalid	X	X	X	X	
	03 pointer addressing invalid object		X			
2A	Program creation					
	06 invalid operand type	X	X	X	X	
	07 invalid operand attribute	X	X	X	X	
	08 invalid operand value range	X	X	X	X	
	0A invalid operand length	X	X	X	X	
	0C invalid operand odt reference	X	X	X	X	
	0D reserved bits are not zero	X	X	X	X	X
2E	Resource control limit					
	01 user profile storage limit exceeded					X
36	Space management					
	01 space extension/truncation					X
38	Template specification					
	01 template value invalid			X		
	02 template size invalid			X		

8.2 Insert Independent Index Entry (INSINXEN)

Op Code (Hex)	Operand 1	Operand 2	Operand 3
04A3	Index	Argument	Option list

Operand 1: System pointer.

Operand 2: Space pointer.

Operand 3: Space pointer.

Warning: The following information is subject to change from release to release. Use it with caution and be prepared to adjust for changes with each new release.

Description: This instruction inserts one or more new entries into the independent index identified by operand 1 according to the criteria specified in the option list (operand 3).Each entry is inserted into the index at the appropriate location based on the binary value of the argument. No other collating sequence is supported. The maximum length allowed for the independent index entry is 120 bytes.

The argument (operand 2) and the option list (operand 3) have the same format as the argument and option list for the Find Independent Index Entry instruction.

The rule option identifies the type of insert to be performed and has the following meaning:

Insert Type	Value (Hex)	Meaning	Authorization
Insert	0001	Insert unique argument	Insert
Insert with replacement	0002	Insert argument, replacing the nonkey portion if the key is already in the index	Update
Insert without replacement	0003	Insert argument only if the key is not already in the index	Insert

The insert rule option is valid only for indexes not containing keys. The insert with replacement rule option and the insert without replacement rule option are valid for indexes containing either fixed- or variable-length entries with keys. The duplicate key argument exception is signaled for the following conditions:

- If the rule option is insert and the argument to be inserted (operand 2) is already in the index
- If the rule option is insert without replacement and the key portion of the argument to be inserted (operand 2) is already in the index

The argument length and argument offset fields are ignored.

The occurrence count specifies the number of arguments to be inserted. This field is limited to a maximum value of 4095. If this value is exceeded, a template value invalid exception is signaled.

If the index was created to contain both pointers and data, then each entry to be inserted must be 16-byte aligned. If the index was created to contain variable-length entries, then the entry length and offset fields must be specified in the option list for each argument in the space identified by operand 2. The entry length is the length of the entry to be inserted.

If the index was created to contain both pointer and scalar data, the offset field in the option list must be supplied for each entry to be inserted. The offset is the number of bytes from the beginning of the previous entry to the beginning of the entry to be inserted. For the first entry, this is the offset from the start of the space identified by operand 2.

The return count specifies the number of entries inserted into the index. If the index was created to contain only data, then any pointers inserted are invalidated.

Authorization Required

- Insert or update depending on insert type
  - Operand 1
- Retrieve
  - Contexts referenced for address resolution

Lock Enforcement

- Materialize
  - Contexts referenced for address resolution
- Modify
  - Operand 1

Exceptions

Exception		Operands 1	2	3	Other
02	Access group				
	01 object exceeds available space	X			
06	Addressing				
	01 space addressing violation	X	X	X	
	02 boundary alignment	X	X	X	
	03 range	X	X	X	
	06 optimized addressability invalid	X	X	X	
08	Argument/parameter				
	01 parameter reference violation	X	X	X	
0A	Authorization				
	01 unauthorized for operation	X			
10	Damage encountered				
	04 system object damage state	X	X	X	X

Exception		Operands 1	2	3	Other
	05 authority verification terminated due to damaged object				X
	44 partial system object damage	X	X	X	X
18	Independent index				
	01 duplicate key argument in index	X			
1A	Lock state				
	01 invalid lock state	X			
1C	Machine-dependent exception				
	03 machine storage limit exceeded				X
	04 object storage limit exceeded	X			
20	Machine support				
	02 machine check				X
	03 function check				X
22	Object access				
	01 object not found	X	X	X	
	02 object destroyed	X	X	X	
	03 object suspended	X	X	X	
	07 authority verification terminated due to destroyed object				X
24	Pointer specification				
	01 pointer does not exist	X	X	X	
	02 pointer type invalid	X	X	X	
	03 pointer addressing invalid object	X			
2A	Program creation				
	06 invalid operand type	X	X	X	
	07 invalid operand attribute	X	X	X	
	08 invalid operand value range	X	X	X	
	0C invalid operand odt reference	X	X	X	
	0D reserved bits are not zero	X	X	X	X
2E	Resource control limit				
	01 user profile storage limit exceeded	X			
36	Space management				
	01 space extension/truncation				X
38	Template specification				
	01 template value invalid			X	
	02 template size invalid			X	

8.3 Materialize Independent Index Attributes (MATINXAT)

Op Code (Hex)	Operand 1	Operand 2
0462	Receiver	Index

Operand 1: Space pointer.

Operand 2: System pointer.

Warning: The following information is subject to change from release to release. Use it with caution and be prepared to adjust for changes with each new release.

Description: The instruction materializes the creation attributes and current operational statistics of the independent index identified by operand 2 into the space identified by operand 1. The format of the attributes materialized is as follows:

- Materialization size specification — Char(8)
  - Number of bytes provided for materialization — Bin(4)
  - Number of bytes available for materialization — Bin(4)
- Object identification — Char(32)
  - Object type — Char(1)
  - Object subtype — Char(1)
  - Object name — Char(30)
- Object creation options — Char(4)
  - Existence attributes — Bit 0

    0 = Temporary
    1 = Reserved
  - Space attribute — Bit 1

    0 = Fixed-length
    1 = Variable-length
  - Context — Bit 2

    0 = Addressability not in context
    1 = Addressability in context
  - Access group — Bit 3

    0 = Not a member of access group
    1 = Member of access group
  - Reserved (binary 0) — Bits 4-12
  - Initialize space — Bit 13
  - Reserved (binary 0) — Bits 14-31
- Reserved (binary 0) — Char(4)
- Size of space — Bin(4)

- Initial value of space — Char(1)
- Performance class — Char(4)
  - Space alignment — Bit 0

    0 = The space associated with the object is allocated to allow proper alignment of pointers at 16-byte alignments within the space. If no space is specified for the object, this value must be specified for the performance class.
    1 = The space associated with the object is allocated to allow proper alignment of pointers at 16-byte alignments within the space as well as to allow proper alignment of input/output buffers at 512-byte alignments within the the space.
  - Reserved (binary 0) — Bits 1-4
  - Main storage pool selection — Bit 5

    0 = Process default main storage pool used for object.
    1 = Machine default main storage pool used for object.
  - Reserved (binary 0) — Bit 6
  - Block transfer on implicit access state modification — Bit 7

    0 = The minimum storage transfer size for this object is a value of 1 storage unit.
    1 = The machine default storage transfer size for this object is a value of 8 storage units.
  - Reserved (binary 0) — Bits 8-31
- Reserved (binary 0) — Char(7)
- Context — System pointer
- Access group — System pointer
- Index attributes — Char(1)
- Argument length — Bin(2)
- Key length — Bin(2)
- Index statistics — Char(12)
  - Entries inserted — Bin(4)
  - Entries removed — Bin(4)
  - Find operations — Bin(4)

The number of arguments in the index equals the number of entries inserted minus entries removed. The value of the find operations field is initialized to 0 each time the index is materialized. The value may not be correct after an abnormal system termination.

The first 4 bytes of the materialization identify the total number of bytes that may be used by the instruction. This value is supplied as input to the instruction and is not modified by the instruction. A value of less than 8 causes the materialization length exception to be signaled.

The second 4 bytes of the materialization identify the total number of bytes available to be materialized. The instruction materializes as many bytes as can be contained in the area specified as the receiver. If the byte area identified by the receiver is greater than that required to contain the information requested, then the excess bytes are unchanged.

No exceptions other than the materialization length exception described previously are signaled in the event that the receiver contains insufficient area for the materialization.

The template identified by the operand 1 space pointer must be 16-byte aligned. Values in the template remain the same as the values specified at the creation of the independent index except that the object identification, context, and size of the associated space contain current values.

If the entry length is fixed, then the argument length is the value supplied in the template when the index was created. If the entry length is variable, then the argument length entry is equal to the length of the longest entry that has ever been inserted into the index.

Authorization Required

- Operational
  - Operand 2
- Retrieve
  - Contexts referenced for address resolution

Lock Enforcement

- Materialize
  - Operand 2
  - Contexts referenced for address resolution

Exceptions

Exception		Operands 1	2	Other
06	Addressing			
	01 space addressing violation	X	X	
	02 boundary alignment	X	X	
	03 range	X	X	
	06 optimized addressability invalid	X	X	
08	Argument/parameter			
	01 parameter reference violation	X	X	
0A	Authorization			
	01 unauthorized for operation		X	
10	Damage encountered			
	04 system object damage state	X	X	X
	05 authority verification terminated due to damaged object			X

Exception		Operands 1	2	Other
	44 partial system object damage	X	X	X
1A	Lock state			
	01 invalid lock state		X	
1C	Machine-dependent exception			
	03 machine storage limit exceeded			X
22	Object access			
	01 object not found	X	X	
	02 object destroyed	X	X	
	03 object suspended	X	X	
	07 authority verification terminated due to destroyed object			X
24	Pointer specification			
	01 pointer does not exist	X	X	
	02 pointer type invalid	X	X	
	03 pointer addressing invalid object		X	
2A	Program creation			
	06 invalid operand type	X	X	
	07 invalid operand attribute	X	X	
	08 invalid operand value range	X	X	
	0C invalid operand odt reference	X	X	
	0D reserved bits are not zero	X	X	X
2E	Resource control limit			
	01 user profile storage limit exceeded			X
36	Space management			
	01 space extension/truncation			X
38	Template specification			
	01 template value invalid		X	
	03 materialization length exception	X		

8.4 Modify Independent Index (MODINX)

Op Code (Hex)	Operand 1	Operand 2
0452	Inde-pendent index	Modification option

Operand 1: System pointer.

Operand 2: Character (4) scalar.

Warning: The following information is subject to change from release to release. Use it with caution and be prepared to adjust for changes with each new release.

Description: This instruction modifies the selected attributes of the independent index specified by operand 1 to have the values specified in operand 2. The modification options specified in operand 2 have the following format:

- Modification selection — Char(1)
  - Reserved (binary 0) — Bit 0
  - Immediate update — Bit 1

    0 = Do not change immediate update attribute
    1 = Change immediate update attribute
  - Reserved (binary 0) — Bits 2-7
- New attribute value — Char(1)
  - Reserved (binary 0) — Bit 0
  - Immediate update — Bit 1

    0 = No immediate update
    1 = Immediate update
  - Reserved (binary 0) — Bits 2-7
- Reserved (binary 0) — Char(2)

If the modification selection immediate update is 0, then the immediate update attribute is not changed. If the modification selection immediate update bit is 1, the immediate update attribute is changed to the new immediate update attribute value.

If the immediate update attribute of the index was previously set to no immediate update, and it is being modified to immediate update, then the index is ensured before the attribute is modified.

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for this instruction.

Authorization Required

- Object management
  - Operand 1
- Retrieve
  - Contexts referenced for address resolution

Lock Enforcement

- Modify
  - Operand 1
- Materialization
  - Contexts referenced for address resolution

Exceptions

Exception		Operands 1	Operands 2	Other
06	Addressing			
	01 space addressing violation	X	X	
	02 boundary alignment	X	X	
	03 range	X	X	
	06 optimized addressability invalid	X	X	
08	Argument/parameter			
	01 parameter reference violation	X	X	
0A	Authorization			
	01 unauthorized for operation	X		
10	Damage encountered			
	04 system object damage	X		X
	05 authority verification terminated due to damaged object			X
	44 partial system object damage			X
1A	Lock state			
	01 invalid lock state	X		
1C	Machine-dependent exception			
	03 machine storage limit exceeded			X
20	Machine support			
	02 machine check			X
	03 function check			X
22	Object access			
	01 object not found	X	X	
	02 object destroyed	X	X	
	03 object suspended	X	X	

Exception		Operands 1	2	Other
	07 authority verification terminated due to destroyed object			X
24	Pointer specification			
	01 pointer does not exist	X	X	
	02 pointer type invalid	X	X	
	03 pointer address invalid object	X		
2A	Program creation			
	06 invalid operand type	X	X	
	07 invalid operand attribute	X	X	
	08 invalid operand value range	X	X	
	0A invalid operand length		X	
	0C invalid operand odt reference	X	X	
	0D reserved bits are not zero	X	X	X
2E	Resource control limit			
	01 user profile storage limit exceeded			X
32	Scalar specification			
	01 scalar type invalid	X	X	
	02 scalar attributes invalid		X	
36	Space management			
	01 space extension/truncation			X

8.5 Remove Independent Index Entry (RMVINXEN)

Op Code (Hex)	Operand 1	Operand 2	Operand 3	Operand 4
0484	Receiver	Index	Option list	Argument

Operand 1: Space pointer or null.

Operand 2: System pointer.

Operand 3: Space pointer.

Operand 4: Space pointer.

Warning: The following information is subject to change from release to release. Use it with caution and be prepared to adjust for changes with each new release.

Description: The index entries identified by operands 3 and 4 are removed from the independent index identified by operand 2 and optionally returned in the receiver specified by operand 1. The maximum length of an independent index entry is 120 bytes.

The option list (operand 3) and the argument (operand 4) have the same format and meaning as the option list and argument for the Find Independent Index Entry instruction. The return count designates the number of index entries that were removed from the index.

The arguments removed are returned in the receiver field if a space pointer is specified for operand 1. If operand 1 is null, the entries removed from the index are not returned. If neither space pointer nor null is specified for operand 1, the entries are returned in the same way that entries are returned for the Find Independent Index Entry instruction.

Every entry removed causes the occurrence count to be incremented by 1. The current value of this count is available through the Materialize Index Attributes instruction. The occurrence count field must be less then 4096.

Authorization Required

- Delete
  - Operand 2
- Retrieve
  - Contexts referenced for address resolution

Lock Enforcement

- Materialize
  - Contexts referenced for address resolution
- Modify
  - Operand 2

Exceptions

Exception		Operands 1	2	3	4	Other
02	Access group					
	01 object exceeds available space		X			
06	Addressing					
	01 space addressing violation	X	X	X	X	
	02 boundary alignment	X	X	X	X	
	03 range	X	X	X	X	
	06 optimized addressability invalid	X	X	X	X	
08	Argument/parameter					
	01 parameter reference violation	X	X	X	X	
0A	Authorization					
	01 unauthorized for operation		X			
10	Damage encountered					
	04 system object damage state	X	X	X	X	X
	05 authority verification terminated due to damaged object					X
	44 partial system object damage	X	X	X	X	X
1A	Lock state					
	01 invalid lock state		X			X
1C	Machine-dependent exception					
	03 machine storage limit exceeded					X
	04 object storage limit exceeded		X			
20	Machine support					
	02 machine check					X
	03 function check					X
22	Object access					
	01 object not found	X	X	X	X	
	02 object destroyed	X	X	X	X	
	03 object suspended	X	X	X	X	
	07 authority verification terminated due to destroyed object					X
24	Pointer specification					
	01 pointer does not exist	X	X	X	X	
	02 pointer type invalid	X	X	X	X	
	03 pointer addressing invalid object		X			
2A	Program creation					
	06 invalid operand type	X	X	X	X	

Exception		Operands 1	2	3	4	Other
	07 invalid operand attribute	X	X	X	X	
	08 invalid operand value range	X	X	X	X	
	0C invalid operand odt reference	X	X	X	X	
	0D reserved bits are not zero	X	X	X	X	X
2E	Resource control limit					
	01 user profile storage limit exceeded		X			
36	Space management					
	01 space extension/truncation					X
38	Template specification					
	01 template value invalid			X		
	02 template size invalid			X		

Chapter 9. Queue Management Instructions

This chapter describes the instructions used for queue management. These instructions are in alphabetic order. For an alphabetic summary of all the instructions, see Appendix A, "Instruction Summary."

9.1 Dequeue (DEQ, DEQB, or DEQI)

Op Code (Hex)	Extender	Operand 1	Operand 2	Operand 3	Operand 4-5
1033		Message prefix	Message text	Queue or queue template	
1C33	Branch options				Branch target
1833	Indicator options				Indicator target

Operand 1: Character variable scalar (fixed-length).

Operand 2: Space pointer.

Operand 3: System pointer or space pointer.

Operand 4-5:

- *Branch Form*-Branch point, instruction pointer, relative instruction number, or absolute instruction number.
- *Indicator Form*-Numeric variable scalar or character variable scalar.

Extender: Branch or indicator options.

If the branch or indicator option is indicated in the op code, the extender field is required along with one or two branch operands (for branch option) or one or two indicator operands (for indicator option). See Chapter 1. "Introduction" for the bit encoding of the extender field and the allowed syntax of the branch and indicator operands.

Description: The instruction retrieves a queue message based on the queue type (FIFO, LIFO, or keyed) specified during the queue's creation. If the queue was created with the keyed option, messages can be retrieved by any of the following relationships between an enqueued message key and a selection key specified in operand 1 of the Dequeue instruction: $\neq$, $>$, $<$, $\leq$, and $\geq$. If the queue was created with either the LIFO or FIFO attribute, then only the next message can be retrieved from the queue.

If a message is not found that satisfies the dequeue selection criterion and the branch or options are not specified, the process is put into the wait state until a message arrives to satisfy the dequeue or until the dequeue wait time-out expires. If branch or indicator options are specified, the process is not placed in the dequeue wait state and either the control flow is altered according to the branch options, or indicator values are set based on the presence or absence of a message to be dequeued.

A nonzero dequeue wait time-out value overrides any dequeue wait time-out value specified as the current process attribute. A zero wait time-out value causes the wait time-out value to be taken from the current process attribute. If all wait time-out values are 0 (from the Dequeue instruction and the current process attribute), an immediate wait time-out exception is signaled. The bits in this field are numbered from 0 to 63, and bit 41 is defined as 1024 microseconds.

The maximum wait time-out interval allowed is a value equal to (2^{48} - 1) micro-seconds. Any value that indicates more time than the maximum wait time-out causes the maximum wait time-out to be used.

If operand 3 is a system pointer, the message is dequeued from the queue specified by operand 3. If operand 3 is a space pointer, the message is dequeued from the queue which is specified in the template pointed to by the space pointer. The format of this template is given later in this section. The criteria for message selection are given in the message prefix specified by operand 1. The message text is returned in the space specified by operand 2, and the message prefix is returned in the scalar specified by operand 1. The size of the message text retrieved is returned in the message prefix. The size of the message text can be less than or equal to the maximum size of message specified when the queue was created. When dequeuing from a keyed queue, the length of the search key field and the length of the message key field (in the message key prefix specified in operand 1) are determined implicitly by the attributes of the queue being accessed. If the message text on the queue contains pointers, the message text operand must be 16-byte aligned. Improper alignment results in an exception being signaled. The format of the message prefix is as follows:

- Timestamp of enqueue of message — Char(8)**
- Dequeue wait time-out value (ignored if branch options specified) — Char(8)*
- Size of message dequeued — Bin(4)**
  (The maximum allowable size of a queue message is 65 000 bytes.)
- Access state modification option indicator and message selection criteria — Char(1)*
  - Access state modification option when entering Dequeue wait — Bit 0*

    0 = Access state is not modified
    1 = Access state is modified
  - Access state modification option when leaving Dequeue wait — Bit 1*

    0 = Access state is not modified
    1 = Access state is modified
  - Multiprogramming level option — Bit 2*

    0 = Leave current MPL set at Dequeue wait
    1 = Remain in current MPL set at Dequeue wait
  - Time-out option — Bit 3*

    0 = Wait for specified time, then signal time-out exception
    1 = Wait indefinitely
  - Actual key to input key relationship (for keyed queue) — Bits 4-7*

    0010: Greater than
    0100: Less than
    0110: Not equal
    1000: Equal
    1010: Greater than or equal
    1100: Less than or equal

• Search key (ignored for FIFO/LIFO queues but must be present for FIFO/LIFO queues with nonzero key length values)	Char(key length)*
• Message key	Char(key length)**

Note: Fields shown here with one asterisk indicate input to the instruction, and fields shown here with two asterisks are returned by the machine.

The access state of the process access group is modified when a Dequeue instruction results in a wait and the following conditions exist: the process' instruction wait initiation access state control attribute specifies allow access state modification, the dequeue access state modification option specifies modify access state, and the multiprogramming level option specifies leave MPL set during wait.

The process will remain in the current MPL set for a maximum of two seconds when a Dequeue instruction results in a wait if the multiprogramming level option specifies remain in current MPL set at Dequeue wait and the access state modification when entering Dequeue wait option specifies do not modify access state. After two seconds, the process will automatically be removed from the current MPL set. The automatic removal does not change or affect the total wait time specified for the process by the Dequeue wait time-out value.

Operand 3 can be a system pointer or a space pointer. If it is a system pointer, this pointer will be addressing the queue from which the message is to be dequeued. If it is a space pointer, this pointer will be addressing a template which will contain the system pointer to the queue as well as the Dequeue template extension. The template is 32 bytes in length and must be aligned on a 16-byte boundary with the format as follows:

• Queue	System pointer
• Dequeue template extension	Char(16)
— Extension Options	Char(1)
- Modify process event mask option 0 = Do not modify process event mask 1 = Modify process event mask	Bit 0 *
- Reserved (binary 0)	Bits 1-7
-	
— Extension Area	Char(15)
New process event mask	Bin(2) *
- Previous process event mask	Bin(2) **
- Reserved (binary 0)	Char(11)

The previous process event mask is only returned when the modify process event mask option has been set to 1.

Note: Fields shown here with one asterisk indicate input to the instruction, and fields shown here with two asterisks are returned by the machine.

The work with process event mask option controls the state of the event mask in the process executing this instruction. If the event mask is in the masked state, the machine does not schedule signaled event monitors in the process. The event monitors continue to be signaled by the machine or other processes. When the process is modified to the unmasked state, event handlers are scheduled to handle those events that occurred while the process was masked and those events occurring while in the unmasked state. The number of signals retained while the process is masked is specified by the attributes of the event monitor associated with the process.

The process is automatically masked by the machine when event handlers are invoked. If the process is unmasked in the event handler, other events can be handled if another enabled event monitor within that process is signaled. If the process is masked when it exits from the event handler, the machine explicitly unmasks the process.

Valid masking values are:

0 Masked
256 Unmasked

Other values are reserved and must not be specified. If any other values are specified, a template value invalid exception is signaled.

Whether masking or unmasking the current process, the new mask takes affect upon completion of a satisfied dequeue.

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for this instruction.

Resultant Conditions: Message dequeued (equal), message not dequeued (not equal).

Authorization Required

- Retrieve
  - Operand 3
  - Contexts referenced for address resolution

Lock Enforcement

- Materialize
  - Contexts referenced for address resolution

Exceptions

Exception		Operands 1	2	3	Other
06	Addressing				
	01 space addressing violation	X	X	X	
	02 boundary alignment	X	X	X	
	03 range	X	X	X	
	06 optimized addressability invalid	X	X	X	
08	Argument/parameter				

Exception		Operands 1	2	3	Other
	01 parameter reference violation	X	X	X	
0A	Authorization				
	01 unauthorized for operation			X	X
10	Damage encountered				
	04 system object damage state	X	X	X	X
	05 authority verification terminated due to damaged object				X
	44 partial system object damage	X	X	X	X
1A	Lock state				
	01 invalid lock state		X	X	
1C	Machine-dependent exception				
	03 machine storage limit exceeded				X
20	Machine support				
	02 machine check				X
	03 function check				X
22	Object access				
	01 object not found	X		X	
	02 object destroyed	X	X	X	
	03 object suspended	X	X	X	
	07 authority verification terminated due to destroyed object				X
24	Pointer specification				
	01 pointer does not exist	X	X	X	
	02 pointer type invalid	X	X	X	
	03 pointer address invalid object	X			
2A	Program creation				
	06 invalid operand type	X	X	X	
	07 invalid operand attribute	X			
	08 invalid operand value range	X			
	09 invalid branch target operand				X
	0A invalid operand length	X			
	0C invalid operand odt reference	X	X	X	
	0D reserved bits are not zero	X	X	X	X
2E	Resource control limit				
	01 user profile storage limit exceeded				X
32	Scalar specification				
	03 scalar value invalid	X			

Exception		Operands 1	2	3	Other
36	Space management				
	01 space extension/truncation				X
3A	Wait time-out				
	01 dequeue				X

9.2 Enqueue (ENQ)

Op Code (Hex)	Operand 1	Operand 2	Operand 3
036B	Queue	Message prefix	Message text

Operand 1: System pointer.

Operand 2: Character scalar.

Operand 3: Space pointer.

Description: A message is enqueued according to the queue type attribute specified during the queue's creation.

If keyed sequence is specified, enqueued messages are sequenced in ascending binary collating order according to the key value. If a message to be enqueued has a key value equal to an existing enqueued key value, the message being added is enqueued following the existing message.

If the queue was defined with either last in, first out (LIFO) or first in, first out (FIFO) sequencing, then enqueued messages are ordered chronologically with the latest enqueued message being either first on the queue or last on the queue, respectively. A key can be provided and associated with messages enqueued in a LIFO or FIFO queue; however, the key does not establish a message's position in the queue. The key can contain pointers, but the pointers are not considered to be pointers when they are placed on the queue by an Enqueue instruction.

Operand 1 specifies the queue to which a message is to be enqueued. Operand 2 specifies the message prefix, and operand 3 specifies the message text.

The format of the message prefix is as follows:

• Size of message to be enqueued	Bin(4)*
• Enqueue key value (Ignored for FIFO/LIFO queues with key lengths equal to 0. Must be present for all other queues.)	Char(key length)*

Note: Fields annotated with an asterisk indicate input to the instruction.

The size of the message to be enqueued is supplied to inform the machine of the number of bytes in the space that are to be considered message text. The size of the message is then considered the lesser of the size of the message to be enqueued attribute and the maximum message size specified on queue creation. The message text can contain pointers. When pointers are in message text, the operand 3 space pointer must be 16-byte aligned. Improper alignment will result in an exception being signaled.

If the enqueued message causes the number of messages to exceed the maximum number of messages attribute of the queue, one of the following occurs:

- If the queue is not extendable, the queue message limit exceeded exception and the queue message limit exceeded event are signaled. The message is not enqueued.

- If the queue is extendable, the queue is implicitly extended by the extension value attribute. The message is enqueued. No exception is signaled, but the queue extended event is signaled.

The maximum allowable queue size, including all messages currently enqueued and the machine overhead, is 16 megabytes.

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for this instruction.

Authorization Required

- Insert
  - Operand 1
- Retrieve
  - Contexts referenced for address resolution

Lock Enforcement

- Materialize
  - Contexts referenced for address resolution

Exceptions

Exception		Operands 1	2	3	Other
06	Addressing				
	01 space addressing violation	X	X	X	
	02 boundary alignment	X	X	X	
	03 range	X	X	X	
	06 optimized addressability invalid	X	X	X	
08	Argument/parameter				
	01 parameter reference violation	X	X	X	
0A	Authorization				
	01 unauthorized for operation	X			X
10	Damage encountered				
	04 system object damage state	X	X	X	X
	05 authority verification terminated due to damaged object				X
	44 partial system object damage	X	X	X	X
1A	Lock state				
	01 invalid lock state	X			X
1C	Machine-dependent exception				
	03 machine storage limit exceeded	X			X
	04 object storage limit exceeded	X			
20	Machine support				
	02 machine check				X

Exception		Operands 1	2	3	Other
	03 function check				X
22	Object access				
	01 object not found	X	X	X	
	02 object destroyed	X	X	X	
	03 object suspended	X	X	X	
	07 authority verification terminated due to destroyed object				X
24	Pointer specification				
	01 pointer does not exist	X	X	X	
	02 pointer type invalid	X	X	X	
	03 pointer address invalid object	X			
26	Process management				
	02 queue message limit exceeded	X			
2A	Program creation				
	06 invalid operand type	X	X	X	
	07 invalid operand attribute		X		
	0C invalid operand odt reference	X	X	X	
	0D reserved bits are not zero	X	X	X	X
2E	Resource control limit				
	01 user profile storage limit exceeded				X
36	Space management				
	01 space extension/truncation				X

9.3 Materialize Queue Attributes (MATQAT)

Op Code (Hex)	Operand 1	Operand 2
0336	Receiver	Queue

Operand 1: Space pointer.

Operand 2: System pointer.

Description: The attributes of the queue specified by operand 2 are materialized into the object specified by operand 1. The format of the materialized queue attributes must be aligned on a 16-byte multiple. The format is as follows:

- Materialization size specification — Char(8)
  - Number of bytes provided for materialization — Bin(4)
  - Number of bytes available for materialization — Bin(4)
- Object identification — Char(32)
  - Object type — Char(1)
  - Object subtype — Char(1)
  - Object name — Char(30)
- Object creation options — Char(4)
- Existence attributes — Bit 0

  0 = Temporary
  1 = Permanent
- Space attribute — Bit 1

  0 = Fixed-length
  1 = Variable-length
- Initial context — Bit 2

  0 = Addressability not in context
  1 = Addressability in context
- Access group — Bit 3

  0 = Not a member of access group
  1 = Member of access group
- Reserved (binary 0) — Bits 4-12
- Initialize space — Bit 13
- Reserved (binary 0) — Bits 14-31
- Reserved (binary 0) — Char(4)
- Size of space — Bin(4)
- Initial value of space — Char(1)
- Performance class — Char(4)
  - Space alignment — Bit 0

0 = The space associated with the object is allocated to allow proper alignment of pointers at 16-byte alignments within the space. If no space is specified for the object, this value must be specified for the performance class.
1 = The space associated with the object is allocated to allow proper alignment of pointers at 16-byte alignments within the space as well as to allow proper alignment of input/output buffers at 512-byte alignments within the the space.

- Reserved (binary 0) — Bits 1-4
- Main storage pool selection — Bit 5

  0 = Process default main storage pool is used for object.
  1 = Machine default main storage pool is used for object.

- Reserved (binary 0) — Bit 6
- Block transfer on implicit access state modification — Bit 7

  0 = Transfer the minimum storage transfer size for this object. This value is 1 storage unit.
  1 = Transfer the machine default storage transfer size. This value is 8 storage units.

- Reserved (binary 0) — Bits 8-31

- Reserved (binary 0) — Char(7)
- Context — System pointer
- Access group — System pointer
- Queue attributes — Char(1)
  - Message content — Bit 0

    0 = Contains scalar data only
    1 = Contains pointers and scalar data

  - Queue type — Bits 1-2

    00 = Keyed
    01 = Last in, first out
    10 = First in, first out

  - Queue overflow action — Bit 3

    0 = Signal exception
    1 = Extend queue

  - Reserved (binary 0) — Bits 4-7
- Current maximum number of messages — Bin(4)
- Current number of messages enqueued — Bin(4)
- Extension value — Bin(4)
- Key length — Bin(2)
- Maximum size of message to be enqueued — Bin(4)

The first 4 bytes of the materialization identify the total number of bytes that may be used by the instruction. This value is supplied as input to the instruction and

is not modified by the instruction. A value of less than 8 causes the materialization length exception.

The second 4 bytes of the materialization identify the total number of bytes available to be materialized. The instruction materializes as many bytes as can be contained in the area specified as the receiver. If the byte area identified by the receiver is greater than that required to contain the information requested, then the excess bytes are unchanged. No exceptions (other than the materialization length exception described previously) are signaled when the receiver contains insufficient area for the materialization.

Authorization Required

- Operational
  - Operand 2
- Retrieve
  - Contexts referenced for address resolution

Lock Enforcement

- Materialize
  - Operand 2
  - Contexts referenced for address resolution

Exceptions

Exception		Operands 1	Operands 2	Other
06	Addressing			
	01 space addressing violation	X	X	
	02 boundary alignment	X	X	
	03 range	X	X	
	06 optimized addressability invalid	X	X	
08	Argument/parameter			
	01 parameter reference violation	X	X	
0A	Authorization			
	01 unauthorized for operation		X	X
10	Damage encountered			
	04 system object damage state	X	X	X
	05 authority verification terminated due to damaged object			X
	44 partial system object damage	X	X	X
1A	Lock state			
	01 invalid lock state		X	X
20	Machine support			
	02 machine check			X
	03 function check			X

Exception		Operands 1	2	Other
22	Object access			
	01 object not found	X	X	
	02 object destroyed	X	X	
	03 object suspended	X	X	
	07 authority verification terminated due to destroyed object			X
24	Pointer specification			
	01 pointer does not exist	X	X	
	02 pointer type invalid	X	X	
	03 pointer address invalid object		X	
2A	Program creation			
	06 invalid operand type	X	X	
	07 invalid operand attribute	X		
	08 invalid operand value range	X		
	0A invalid operand length	X		
	0C invalid operand odt reference	X	X	
	0D reserved bits are not zero	X	X	X
2E	Resource control limit			
	01 user profile storage limit exceeded			X
36	Space management			
	01 space extension/truncation			X
38	Template specification			
	03 materialization length exception	X		

9.4 Materialize Queue Messages (MATQMSG)

Op Code (Hex)	Operand 1	Operand 2	Operand 3
033B	Receiver	Queue	Message selection template

Operand 1: Space pointer.

Operand 2: System pointer.

Operand 3: Character(16) scalar.

Description: This instruction materializes selected messages on a queue. One or more messages on the queue specified by operand 2 is selected according to information provided in operand 3 and materialized into operand 1. The number of messages materialized and the amount of key and message text data materialized for each message is governed by the message selection template.

Note that the list of messages on a queue is a dynamic attribute and may be changing on a continual basis. The materialization of messages provided by this instruction is just a picture of the status of the queue at the point of interrogation by this instruction. As such, the actual status of the queue may differ from that described in the materialization when subsequent instructions use the information in the template as a basis for operations against the queue.

Operand 1 specifies a space that is to receive the materialized attribute values.

Operand 2 is a system pointer identifying the queue from which the messages are to be materialized.

Operand 3 is a character (16) scalar specifying which messages are to be materialized.

The operand 1 space pointer must address a 16-byte boundary. The materialization template has the following format:

- Materialization size specification — Char(8)
  - Number of bytes provided for materialization — Bin(4)
  - Number of bytes available for materialization — Bin(4)
- Materialization data — Char(4)
  - Count of messages materialized — Bin(4)
- Queue data — Char(12)
- Count of messages on the queue — Bin(4)
- Maximum message size — Bin(4)
- Key size — Bin(4)
- Reserved — Char(8)
- Message data (repeated for each message) — Char(*)
- Message attributes — Char(16)

  - Message enqueue time — Char(8)
  - Message length — Bin(4)
  - Reserved — Char(4)
- Message key — Char(*)
- Message text — Char(*)

The first 4 bytes of the materialization identify the total quantity of bytes that may be used by the instruction. This value is supplied as input to the instruction and is not modified by the instruction. A value of less than 8 causes the materialization length exception to be signaled.

The second 4 bytes of the materialization identify the total quantity of bytes available to be materialized. The instruction materializes as many bytes as can be contained in the area specified as the receiver. If the byte area identified by the receiver is greater than that required to contain the information requested, then the excess bytes are unchanged. No exceptions are signaled in the event that the receiver contains insufficient area for the materialization, other than the materialization length exception described previously.

The maximum message size and key size are values specified when the queue was created. If the queue is not a keyed queue, the value materialized for the key size is zero.

The length of the message key and message text fields is determined by values supplied in operand 3, message selection data. If the length supplied in operand 3 exceeds the actual data length, the remaining space will be padded with binary zeros.

The message selection template identified by operand 3 must be at least 16 bytes and must be on a 16-byte boundary. The format of the message selection template is as follows:

- Message selection — Char(2)
  - Type — Bits 0-3

    0001 = All messages
    0010 = First
    0100 = Last
    1000 = Keyed

    All other values are reserved
  - Key relationship (if needed) — Bits 4-7

    0010 = Greater than
    0100 = Less than
    0110 = Not equal
    1000 = Equal
    1010 = Greater than or equal
    1100 = Less than or equal

    All other values are reserved
  - Reserved — Bits 8-15
- Lengths — Char(8)

  - Number of key bytes to materialize Bin(4)
  - Number of message text bytes to materialize Bin(4)
- Reserved Char(6)
- Key (if needed) Char(*)

The message selection type must not specify keyed if the queue was not created as a keyed queue.

Both of the fields specified under lengths must be zero or an integer multiple of 16. The maximum value allowed for the key length is 256. The maximum value allowed for the message text is 65536.

Authorization Required

- Retrieve
  - Operand 2
  - Contexts referenced for address resolution

Lock Enforcement

- Materialization
  - Operand 2
  - Contexts referenced for address resolution

Exceptions

Exception		Operands 1	2	3	4	Other
06	Addressing					
	01 space addressing violation	X	X	X		
	02 boundary alignment	X	X	X		
	03 range	X	X	X		
	06 optimized addressability invalid	X	X	X		
08	Argument/parameter					
	01 parameter reference violation	X	X	X		
0A	Authorization					
	01 unauthorized for operation		X			
10	Damage encountered					
	04 system object damage state		X			X
	05 authority verification terminated due to damaged object					X
	44 partial system object damage					X
1A	Lock state					
	01 invalid lock state				X	
20	Machine support					
	02 machine check					X
	03 function check					X

Exception		Operands 1	2	3	4	Other
22	Object access					
	01 object not found	X	X	X		
	02 object destroyed	X	X	X		
	03 object suspended	X	X	X		
	07 authority verification terminated due to destroyed object					X
24	Pointer specification					
	01 pointer does not exist	X	X	X		
	02 pointer type invalid	X	X	X		
	03 pointer address invalid object		X			
28	Process state					
	02 process control space not associated with a process		X			
2A	Program creation					
	06 invalid operand type	X	X	X		
	07 invalid operand attribute	X	X	X		
	08 invalid operand value range	X	X	X		
	0A invalid operand length			X		
	0C invalid operand odt reference	X	X	X		
	0D reserved bits are not zero	X	X	X	X	X
2E	Resource control limit					
	01 user profile storage limit exceeded					X
32	Scalar specification					
	01 scalar type invalid	X	X	X		
	02 scalar attributes invalid	X	X	X		
	03 scalar value invalid			X		
36	Space management					
	01 space extension/truncation					X
38	Template specification					
	03 materialization length exception	X				

Chapter 10. Object Lock Management Instructions

This chapter describes the lock management instructions. The instructions are in alphabetic order. For an alphabetic summary of all the instructions, see Appendix A, "Instruction Summary."

10.1 Lock Object (LOCK)

Op Code (Hex)	Operand 1
03F5	Lock request template

Operand 1 Space pointer.

Description: The instruction requests that locks for system objects identified by system pointers in the space object (operand 1) be allocated to the issuing process. The lock state desired for each object is specified by a value associated with each system pointer in the lock template (operand 1).

The lock request template must be aligned on a 16-byte boundary. The format is as follows:

- Number of lock requests in template — Bin(4)
- Offset to lock state selection values — Bin(2)
- Wait time-out value for instruction — Char(8)
- Lock request options — Char(1)
  - Lock request type — Bits 0-1

    00 = Immediate request-If all locks cannot be immediately granted, signal exception.
    01 = Synchronous request-Wait until all locks can be granted.
    10 = Asynchronous request-Allow processing to continue and signal event when the object is available.
  - Access state modifications — Bits 2-3

    When the process is entering lock wait for synchronous request: — Bit 2

    0= Access state should not be modified.
    1= Access state should be modified.

    When the process is leaving lock wait: — Bit 3

    0= Access state should not be modified.
    1= Access state should be modified.
  - Reserved (binary 0) — Bits 4-5*
  - Time-out option — Bit 6

    0= Wait for specified time, then signal time-out exception.
    1= Wait indefinitely.
  - Template extension specified — Bit 7

    0= Template is not specified.
    1= Template is specified.
- Reserved (binary 0) — Char(1)
- Lock Object template extension — Char(16)
  - Extension Options — Char(1)
    - Modify process event mask option — Bit 0

0 = Do not modify process event mask
1 = Modify process event mask

- Reserved (binary 0) Bits 1-7

— Extension Area Char(15)

- New process event mask Bin(2)
- Previous process event mask Bin(2)
- Reserved (binary 0) Char(11)

Modify process event mask being set to 1 is not allowed when the lock request type is asynchronous.

The previous process event mask is only returned when the modify process event mask option has been set to 1.

The work with process event mask option controls the state of the event mask in the process executing this instruction. If the event mask is in the masked state, the machine does not schedule signaled event monitors in the process. The event monitors continue to be signaled by the machine or other processes. When the process is modified to the unmasked state, event handlers are scheduled to handle those events that occurred while the process was masked and those events occurring while in the unmasked state. The number of signals retained while the process is masked is specified by the attributes of the event monitor associated with the process.

The process is automatically masked by the machine when event handlers are invoked. If the process is unmasked in the event handler, other events can be handled if another enabled event monitor within that process is signaled. If the process is masked when it exits from the event handler, the machine explicitly unmasks the process.

Valid masking values are:

0 Masked
256 Unmasked

Other values are reserved and must not be specified. If any other values are specified, a template value invalid exception is signaled.

Whether masking or unmasking the current process, the new mask takes affect upon completion of a satisfied lock object.

Lock state selection values located by the offset to lock state selection values field.

- Object(s) to be locked — System pointer (one for each object to be locked)
- Lock state selection — Char(1)
  (repeated for each pointer in the template)

  — Requested lock state — Bits 0-4
  (1 = lock requested, 0 = lock not requested)

  Only one state may be requested.

LSRD lock	Bit 0
LSRO lock	Bit 1
LSUP lock	Bit 2
LEAR lock	Bit 3

LENR lock	Bit 4
— Reserved (binary 0)	Bits 5-6*
— Entry active indicator	Bit 7

0 = Entry not active-This entry is not used.
1 = Entry active-Obtain this lock.

Note: Entries indicated with an asterisk are ignored by the instruction.

Lock Allocation Procedure

A single Lock instruction can request the allocation of one or more lock states on one or more objects. Locks are allocated sequentially until all locks requested are allocated.

When a requested lock state cannot be immediately granted, any locks already allocated by this Lock instruction are released, and the lock request option specified in the lock request template establishes the machine action. The lock request options are described in the following paragraphs.

- Immediate Request-If the requested locks cannot be granted immediately, this option causes the lock request not grantable exception to be signaled. No locks are granted, and the lock request is canceled.
- Synchronous Request-This option causes the process requesting the locks to be placed in the wait state until all requested locks can be granted. If the locks cannot be granted in the time interval established by the wait time-out parameter specified in the lock request template, the lock wait time-out exception is signaled to the requesting process at the end of the interval. No locks are granted, and the lock request is canceled.
- Asynchronous Request-This option allows the requesting process to proceed with execution while the machine asynchronously attempts to satisfy the lock request.

When the synchronous request option is specified and the requested locks cannot be immediately allocated, the access state modification parameter in the lock request template specifies whether the access state of the process access group is to be modified on entering and/or returning from the lock wait. The parameter has no effect if the process instruction wait access state control attribute specifies that no access state modification is allowed. If the process attribute value specifies that access state modification is allowed and the wait on event access state modification option specifies modify access state, the machine modifies the access state for the specified process access group.

If a synchronous lock wait is requested and the invocation containing the lock instruction is terminated, then the lock request is canceled.

If the lock request is satisfied, then the object locked event is signaled to the requesting process. If the request is not satisfied in the time interval established by the wait time-out parameter specified in the lock request template, the wait time-out for pending lock event is signaled to the requesting process. No locks are granted, and the lock request is canceled. If an object is destroyed while a process has a pending request to lock the object, the object destroyed event is signaled to the waiting process.

If an asynchronous lock wait is requested and the invocation containing the Lock instruction is terminated, then the lock request remains active.

The wait time-out parameter establishes the maximum amount of time that a process competes for the requested set of locks when either the synchronous or asynchronous wait options are specified. The bits in this field are numbered from 0 to 63, and bit 41 is defined as 1024 microseconds. The maximum wait time-out interval allowed is a value equal to (2^{48} - 1) microseconds. Any value that indicates more time than the maximum wait time-out causes the maximum wait time-out to be used. If the wait time-out parameter is specified with a value of binary 0, then the value associated with the default wait time-out parameter in the process definition template establishes the time interval.

When two or more processes are competing for a conflicting lock allocation on a system object, the machine attempts to first satisfy the lock allocation request of the process with the highest priority. Within that priority, the machine attempts to satisfy the request that has been waiting longest.

If any exception is identified during the instruction's execution, any locks already granted by the instruction are released, and the lock request is canceled.

For each system object lock counts are kept by lock state and by process. When a lock request is granted, the appropriate lock count(s) of each lock state specified is incremented by 1.

If a previously unsatisfied lock request is satisfied by the transfer of a lock from another process, the lock request and transfer lock are treated as independent events relative to lock accounting. The appropriate lock counts are incremented for both the lock request and the transfer lock function.

Authorization Required

- Some authority or ownership
  - Objects to be locked
- Retrieve
  - Contexts referenced for address resolution

Lock Enforcement

- Materialize
  - Contexts referenced for address resolution

Exceptions

Exception		Operands 1	Operands Other
06	Addressing		
	01 space addressing violation	X	
	02 boundary alignment	X	
	03 range	X	
	06 optimized addressability invalid	X	
08	Argument/parameter		
	01 parameter reference violation	X	

Exception		Operands 1	Other
0A	Authorization		
	01 unauthorized for operation	X	
10	Damage encountered		
	04 system object damage state	X	X
	05 authority verification terminated due to damaged object		X
	44 partial system object damage	X	X
1A	Lock state		
	01 invalid lock state		X
	02 lock request not grantable	X	
1C	Machine-dependent exception		
	03 machine storage limit exceeded		X
	06 machine lock limit exceeded		X
20	Machine support		
	02 machine check		X
	03 function check		X
22	Object access		
	01 object not found	X	
	02 object destroyed	X	
	03 object suspended	X	
	07 authority verification terminated due to destroyed object		X
24	Pointer specification		
	01 pointer does not exist	X	
	02 pointer type invalid	X	
2A	Program creation		
	06 invalid operand type	X	
	07 invalid operand attribute	X	
	08 invalid operand value range	X	
	0C invalid operand odt reference	X	
	0D reserved bits are not zero	X	X
2E	Resource control limit		
	01 user profile storage limit exceeded		X
36	Space management		
	01 space extension/truncation		X
38	Template specification		
	01 template value invalid	X	

Exception		Operands 1	Other
3A	Wait time-out		
	02 lock		X

10.2 Lock Space Location (LOCKSL)

Op Code (Hex)	Operand 1	Operand 2
03F6	Space location	Lock type request

*Operand 1:*Space pointer data object.

*Operand 2:*Char(1) scalar.

Description: The space location identified by operand 1 is locked according to the request specified by operand 2. Locking the space location does not prevent any byte operation from referencing that location, nor does it prevent the space from being extended, truncated, or destroyed. Space location locks follow the normal locking rules with respect to conflicts and waits but are strictly symbolic in nature.

Following is the format of operand 2:

- Requested lock state Char(1)

  Hex 80 = LSRD lock
  Hex 40 = LSRO lock
  Hex 20 = LSUP lock
  Hex 10 = LEAR lock
  Hex 08 = LENR lock

  All other values are reserved.

If the requested lock cannot be immediately granted, the process will enter a synchronous wait for the lock, for a period of up to the interval specified by the process default time-out value. If the wait exceeds this time limit, a space location lock wait exception is signaled, and the requested lock is not granted.

During the wait, the process access state may be modified. This can occur if the process' instruction wait access state control attribute is set to allow access state modification.

A space pointer machine object cannot be specified for operand 1.

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for this instruction.

Exceptions

Exception		Operands 1	Operands 2	Other
06	Addressing			
	01 space addressing violation	X	X	
	02 boundary alignment	X	X	
	03 range	X	X	
	06 optimized addressability invalid	X	X	
08	Argument/parameter			
	01 parameter reference violation	X	X	

Exception		Operands 1	2	Other
10	Damage encountered			
	04 system object damage state			X
	44 partial system object damage			X
1C	Machine-dependent exception			
	03 machine storage limit exceeded			X
	06 machine lock limit exceeded			X
20	Machine support			
	02 machine check			X
	03 function check			X
22	Object access			
	02 object destroyed	X	X	
24	Pointer specification			
	01 pointer does not exist	X	X	
	02 pointer type invalid	X	X	
2A	Program creation			
	06 invalid operand type	X	X	
	07 invalid operand attribute	X	X	
	08 invalid operand value range	X	X	
	0C invalid operand odt reference	X	X	
	0D reserved bits are not zero	X	X	X
2E	Resource control limit			
	01 user profile storage limit exceeded			X
32	Scalar specification			
	01 scalar type invalid	X	X	
	03 scalar value invalid		X	
36	Space management			
	01 space extension/truncation			X
3A	Wait time-out			
	04 space location lock wait	X		

10.3 Materialize Allocated Object Locks (MATAOL)

Op Code (Hex)	Operand 1	Operand 2
03FA	Receiver	System object or space location

Operand 1: Space pointer.

Operand 2: System pointer or space pointer data object.

Description: This instruction materializes the current allocated locks on a designated object. If operand 2 is a system pointer, the current allocated locks on the object identified by the system pointer specified by operand 2 are materialized into the template specified by operand 1. If operand 2 is a space pointer, the current allocated locks on the specified space location are materialized into the template specified by operand 1. The materialization template identified by operand 1 must be 16-byte aligned. The format of the materialization is as follows:

- Materialization size specification — Char(8)
- Number of bytes provided for materialization — Bin(4)
- Number of bytes available for materialization — Bin(4)
- Current cumulative lock status — Char(3)
- Lock states currently allocated (1 = yes) — Char(1)

LSRD	Bit 0
LSRO	Bit 1
LSUP	Bit 2
LEAR	Bit 3
LENR	Bit 4
Locks implicitly set	Bit 5
Reserved (binary 0)	Bits 6-7

- Reserved (binary 0) — Char(2)
- Reserved (binary 0) — Char(1)
- Number of lock descriptions following — Bin(2)
- Reserved (binary 0) — Char(2)
- Lock state descriptors (repeated for each lock currently allocated) — Char(32)
  - Process control space — System pointer
  - Lock state being described — Char(1)

    Hex 80 = LSRD lock request
    Hex 40 = LSRO lock request
    Hex 20 = LSUP lock request
    Hex 10 = LEAR lock request
    Hex 08 = LENR lock request

    All other values are reserved

— Status of lock request Char(1)

A value of 1 in the corresponding bit indicates the condition is true:

Reserved (binary 0)	Bits 0-5
Implicit lock (machine applied)	Bit 6
Lock held by process	Bit 7

— Reserved (binary 0) Char(14)

Locks may be implicitly applied by the machine (status code = hex 02). If the implicit lock is held for a process, a pointer to the associated process control space is returned. Locks held by the machine but not related to a specific process, cause the process control space entry to be assigned a value of binary zero.

Only a single lock state is returned for each lock state descriptor entry.

The first 4 bytes of the materialization identify the total quantity of bytes that may be used by the instruction. This value is supplied as input to the instruction and is not modified by the instruction. A value of less than eight causes the materialization length exception to be signaled.

The second 4 bytes of the materialization identify the total quantity of bytes available to be materialized. The instruction materializes as many bytes as can be contained in the area specified as the receiver. If the byte area identified by the receiver is greater than that required to contain the information requested, then the excess bytes are unchanged. No exceptions are signaled in the event the receiver contains insufficient area for the materialization, other than the materialization length exception.

A space pointer machine object cannot be specified for operand 2.

Exceptions

Exception		Operands 1	Operands 2	Other
06	Addressing			
	01 space addressing violation	X	X	
	02 boundary alignment	X	X	
	03 range	X	X	
	06 optimized addressability invalid	X	X	
08	Argument/parameter			
	01 parameter reference violation	X	X	
0A	Authorization			
	01 unauthorized for operation		X	X
10	Damage encountered			
	04 system object damage		X	X
	44 partial system object damage			X
1A	Lock state			
	01 invalid lock state		X	

Exception		Operands 1	2	Other
1C	Machine-dependent exception			
	03 machine storage limit exceeded			X
20	Machine support			
	02 machine check			X
	03 function check			X
22	Object access			
	01 object not found	X	X	
	02 object destroyed	X	X	
	03 object suspended	X	X	
24	Pointer specification			
	01 pointer does not exist	X	X	
	02 pointer type invalid	X	X	
2A	Program creation			
	06 invalid operand type	X	X	
	07 invalid operand attribute	X	X	
	08 invalid operand value range	X	X	
	0A invalid operand length	X		
	0C invalid operand odt reference	X	X	
	0D reserved bits are not zero	X	X	X
2E	Resource control limit			
	01 user profile storage limit exceeded			X
32	Scalar specification			
	01 scalar type invalid	X	X	
36	Space management			
	01 space extension/truncation			X
38	Template specification			
	03 materialization length exception	X		

10.4 Materialize Data Space Record Locks (MATDRECL)

Op Code (Hex)	Operand 1	Operand 2
032E	Receiver	Record selection template

Operand 1: Space pointer.

Operand 2: Space pointer.

Description: This instruction materializes the current allocated locks on the specified data space record.

The current lock status of the data space record identified by the template in operand 2 is materialized into the space identified by operand 1.

The record selection template identified by operand 2 must be 16-byte aligned. The format of the record selection template is as follows.

- Record selection — Char(24)
  - Data space identification — System pointer
  - Record number — Bin(4)
  - Reserved — Char(4)
- Lock selection — Char(8)
  - Materialize data space locks held — Bit 0

    1 = Materialize
    0 = Do not materialize
  - Materialize data space locks waited for — Bit 1

    1 = Materialize
    0 = Do not materialize
  - Reserved — Bits 2-7
  - Reserved — Char(7)

The data space identification must be a system pointer to a data space.

The record number is a relative record number within that data space. If the record number is zero then all locks on the specified data space will be materialized. If the record number is not valid for the specified data space a template value invalid exception is signaled.

Both of the fields specified under lock selection are bits which determine the locks to be materialized. If the first bit is on, the current holders of the specified data space record lock are materialized. If the second bit is on, any process waiting to lock the specified data space record is materialized.

The materialization template identified by operand 1 must be 16-byte aligned. The format of the materialization is as follows:

- Materialization size specification — Char(8)

- Number of bytes provided for materialization — Bin(4)
- Number of bytes available for materialization — Bin(4)

• Materialization data — Char(8)
- Count of locks held — Bin(2)
- Count of locks waited for — Bin(2)
- Reserved — Char(4)

• Locks held identification (repeated for each lock held) — Char(32)
- Process identification — System pointer
- Record number — Bin(4)
- Lock state being described — Char(1)

  Hex C0 = DLRD lock state
  Hex F8 = DLUP lock state

  All other values are reserved.
- Reserved — Char(11)

• Locks waited for identification (repeated for each lock waited for) — Char(32)
- Process identification — System pointer
- Record number — Bin(4)
- Lock state being described — Char(1)

  Hex C0 = DLRD lock state
  Hex F8 = DLUP lock state

  All other values are reserved.
- Reserved — Char(11)

The first 4 bytes of the materialization identify the total quantity of bytes that may be used by the instruction. This value is supplied as input to the instruction and is not modified by the instruction. A value of less than 8 causes the materialization length exception to be signaled.

The second 4 bytes of the materialization identify the total quantity of bytes available to be materialized. The instruction materializes as many bytes as can be contained in the area specified as the receiver. If the byte area identified by the receiver is greater than that required to contain the information requested, the excess bytes are unchanged. No exceptions are signaled in the event that the receiver contains insufficient area for the materialization, other than the materialization length exception described previously.

The count of locks held contains the number of locks held. One system pointer to the PCS (process control space) of each process holding a lock, the relative record number which is locked, and the lock state are materialized in the area identified as locks held identification. These fields contain data only if held data space locks are selected for materialization.

The count of locks waited for contains the number of locks being waited for. One system pointer to the PCS (process control space) of each process waiting for a

lock, the relative record number, and the lock state which the process is waitng for are materialized in the area identified as locks waited for identification. These fields contain data only if data space record locks waited for are selected for materialization.

Authorization Required

- Retrieve
  - Contexts referenced for address resolution

Lock Enforcement

- Materialize
  - Contexts referenced for address resolution

Exceptions

Exception		Operands 1	Operands 2	Other
06	Addressing			
	01 space addressing violation	X	X	
	02 boundary alignment	X	X	
	03 range	X	X	
	06 optimized addressability invalid	X	X	
08	Argument/parameter			
	01 parameter reference violation	X	X	
0A	Authorization			
	01 unauthorized for operation		X	X
10	Damage encountered			
	04 system object damage state		X	X
	05 authority verification terminated due to damaged object			X
	44 partial system object damage			X
1A	Lock state			
	01 invalid lock state		X	
1C	Machine-dependent exception			
	03 machine storage limit exceeded			X
20	Machine support			
	02 machine check			X
	03 function check			X
22	Object access			
	01 object not found	X	X	
	02 object destroyed	X	X	
	03 object suspended	X	X	
	07 authority verification terminated due to destroyed object			X

Exception		Operands 1	2	Other
24	Pointer specification			
	01 pointer does not exist	X	X	
	02 pointer type invalid	X	X	
2A	Program creation			
	06 invalid operand type	X	X	
	07 invalid operand attribute	X	X	
	08 invalid operand value range	X	X	
	0A invalid operand length	X		
	0C invalid operand odt reference	X	X	
	0D reserved bits are not zero	X	X	X
2E	Resource control limit			
	01 user profile storage limit exceeded			X
32	Scalar specification			
	01 scalar type invalid	X	X	
36	Space management			
	01 space extension/truncation			X
38	Template specification			
	01 template value invalid		X	
	03 materialization length exception	X		

10.5 Materialize Object Locks (MATOBJLK)

Op Code (Hex)	Operand 1	Operand 2
033A	Receiver	System object or space location

Operand 1: Space pointer.

Operand 2: System pointer or space pointer data object.

Description: If operand 2 is a system pointer, the current lock status of the object identified by the system pointer is materialized into the template specified by operand 1. If operand 2 is a space pointer, the current lock status of the specified space location is materialized into the template specified by operand 1. The materialization template identified by operand 1 must be aligned on a 16-byte boundary. The format of the materialization is as follows:

- Materialization size specification — Char(8)
  - Number of bytes provided for materialization — Bin(4)
  - Number of bytes available for materialization — Bin(4)
- Current cumulative lock status — Char(3)
  - Lock states currently allocated (1 = yes) — Char(1)

LSRD	Bit 0
LSRO	Bit 1
LSUP	Bit 2
LEAR	Bit 3
LENR	Bit 4
Locks implicitly set	Bit 5
Reserved (binary 0)	Bits 6-7

  - Lock states for which processes are in synchronous wait (1 = yes) — Char(1)

LSRD	Bit 0
LSRO	Bit 1
LSUP	Bit 2
LEAR	Bit 3
LENR	Bit 4
Implicit lock request	Bit 5
Reserved (binary 0)	Bits 6-7

  - Lock states for which processes are in asynchronous wait (1 = yes) — Char(1)

LSRD	Bit 0
LSRO	Bit 1
LSUP	Bit 2
LEAR	Bit 3
LENR	Bit 4
Reserved (binary 0)	Bits 5-7

- Reserved (binary 0) — Char(1)

- Number of lock descriptions that follow — Bin(2)
- Reserved (binary 0) — Char(2)
- Lock state descriptors (repeated for each lock currently allocated or waited for) — Char(32)
  - Process control space — System pointer
  - Lock state being described — Char(1)

LSRD	Bit 0
LSRO	Bit 1
LSUP	Bit 2
LEAR	Bit 3
LENR	Bit 4
Reserved (binary 0)	Bits 5-7

  - Status of lock request — Char(1)

Reserved	Bits 0-2
Waiting because this lock is not available	Bit 3
Process in asynchronous wait for lock	Bit 4
Process in synchronous wait for lock	Bit 5
Implicit lock (machine- applied)	Bit 6
Lock held by process	Bit 7

  - Reserved (binary 0) — Char(14)

Locks may be applied by the machine (status code = hex 02). If the implicit lock is held for a process, a pointer to the associated process control space is returned. Locks held by the machine but not related to a specific process cause the process control space entry to be assigned a value of binary 0.

Only a single lock state is returned for each lock state descriptor entry.

The first 4 bytes of the materialization identify the total number of bytes that may be used by the instruction. This total is supplied as input to the instruction and is not modified by the instruction. A total of less than 8 causes the materialization length exception to be signaled.

The second 4 bytes of the materialization identify the total number of bytes available to be materialized. The instruction materializes as many bytes as can be contained in the area specified as the receiver. If the byte area identified by the receiver is greater than that required to contain the information requested, then the excess bytes are unchanged. No exceptions (other than the materialization length exception described previously) are signaled if the receiver contains insufficient area for the materialization.

A space pointer machine object cannot be specified for operand 2.

Authorization Required

- Retrieve
  - Contexts referenced for address resolution

Lock Enforcement

- Materialize
  - Contexts referenced for address resolution

Exceptions

Exception		Operands 1	Operands 2	Other
06	Addressing			
	01 space addressing violation	X	X	
	02 boundary alignment	X	X	
	03 range	X	X	
	06 optimized addressability invalid	X	X	
08	Argument/parameter			
	01 parameter reference violation	X	X	
0A	Authorization		X	X
	01 unauthorized for operation		X	
10	Damage encountered			
	04 system object damage state	X	X	X
	05 authority verification terminated due to damaged object			X
	44 partial system object damage	X	X	X
1A	Lock state			
	01 invalid lock state		X	
1C	Machine-dependent exception			
	03 machine storage limit exceeded			X
20	Machine support			
	02 machine check			X
	03 function check			X
22	Object access			
	01 object not found	X	X	
	02 object destroyed	X	X	
	03 object suspended	X	X	
	07 authority verification terminated due to destroyed object			X
24	Pointer specification			
	01 pointer does not exist	X	X	
	02 pointer type invalid	X	X	
2A	Program creation			
	06 invalid operand type	X	X	
	07 invalid operand attribute	X	X	

Exception		Operands 1	2	Other
	08 invalid operand value range	X	X	
	0A invalid operand length	X		
	0C invalid operand odt reference	X	X	
	0D reserved bits are not zero	X	X	X
2E	Resource control limit			
	01 user profile storage limit exceeded			X
36	Space management			
	01 space extension/truncation			X
38	Template specification			
	03 materialization length exception	X		

10.6 Materialize Process Locks (MATPRLK)

Op Code (Hex)	Operand 1	Operand 2
0312	Receiver	Process control space

Operand 1: Space pointer.

Operand 2: System pointer or null.

Description: The lock status of the process identified by operand 2 is materialized into the receiver specified by operand 1. If operand 2 is null, the lock status is materialized for the process issuing the instruction. The materialization identifies each object or space location for which the process has a lock allocated or for which the process is in a synchronous or asynchronous wait. The format of the materialization is as follows:

- Materialization size specification — Char(8)
  - Number of bytes provided for materialization — Bin(4)
  - Number of bytes available for materialization — Bin(4)
- Number of lock entries — Bin(2)
- Expanded number of lock entries — Bin(4)
- Reserved (binary 0) — Char(2)
- Lock status (repeated for each lock currently allocated or waited for by the process) — Char(32)
  - Object, space location, or binary 0 if no pointer exists — System pointer or space pointer
  - Lock state — Char(1)

LSRD	Bit 0
LSRO	Bit 1
LSUP	Bit 2
LEAR	Bit 3
LENR	Bit 4
Reserved (binary 0)	Bits 5-7

  - Status of lock state for process — Char(1)

Reserved	Bits 0-1
Object or space location no longer exists	Bit 2
Waiting because this lock is not available	Bit 3
Process in asynchronous wait for lock	Bit 4
Process in synchronous wait for lock	Bit 5
Implicit lock (machine-applied)	Bit 6
Lock held by process	Bit 7

- Reserved (binary 0) — Char(14)

The first 4 bytes of the materialization identify the total number of bytes that may be used by the instruction. This value is supplied as input to the instruction and is not modified by the instruction. A value of less than 8 causes the materialization length exception to be signaled.

The second 4 bytes of the materialization identify the total number of bytes available to be materialized. The instruction materializes as many bytes as can be contained in the area specified as the receiver. If the byte area identified by the receiver is greater than that required to contain the information requested, then the excess bytes are unchanged. No exceptions (other than the materialization length exception described previously) are signaled if the receiver contains insufficient area for the materialization.

The number of lock entries field identifies the number of lock entries that are materialized. When a process holds more than 32,767 locks, this field is set with its maximum value of 32,767. This field has been retained in the template for compatibility with programs using the template prior to the changes made to support materialization of more than 32,767 lock entries.

The expanded number of lock entries field identifies the number of lock entries that are materialized. This field is always set in addition to the number of lock entries field described previously; however, it does not have a maximum limit of 32,767, so it can be used to specify that more than 32,767 locks have been materialized. When a process holds more than 32,767 locks, the number of lock entries field will equal 32,767, which would be incorrect. The expanded number of lock entries field, however, will identify the correct number of lock entries materialized. In all cases, this field should be used instead of the number of lock entries field to get the correct count of lock entries materialized.

Authorization Required

- Retrieve
  - Context referenced by address resolution

Lock Enforcement

- Materialize
  - Contexts referenced for address resolution

Exceptions

Exception		Operands 1	2	Other
06	Addressing			
	01 space addressing violation	X	X	
	02 boundary alignment	X	X	
	03 range	X	X	
	06 optimized addressability invalid	X	X	
08	Argument/parameter			
	01 parameter reference violation	X	X	
10	Damage encountered			
	04 system object damage state	X	X	X
	05 authority verification terminated due to damaged object			X
	44 partial system object damage	X	X	X
1C	Machine-dependent exception			

Exception		Operands 1	2	Other
	03 machine storage limit exceeded			X
20	Machine support			
	02 machine check			X
	03 function check			X
22	Object access			
	01 object not found	X	X	
	02 object destroyed	X	X	
	03 object suspended	X	X	
	07 authority verification terminated due to destroyed object			X
24	Pointer specification			
	01 pointer does not exist	X	X	
	02 pointer type invalid	X	X	
	03 pointer addressing invalid object		X	
28	Process state			
	02 process control space not associated with a process		X	
2A	Program creation			
	06 invalid operand type	X	X	
	07 invalid operand attribute	X	X	
	08 invalid operand value range	X	X	
	0A invalid operand length	X		
	0C invalid operand odt reference	X	X	
	0D reserved bits are not zero	X	X	X
2E	Resource control limit			
	01 user profile storage limit exceeded			X
36	Space management			
	01 space extension/truncation			X
38	Template specification			
	03 materialization length exception	X		

10.7 Materialize Process Record Locks (MATPRECL)

Op Code (Hex)	Operand 1	Operand 2
031E	Receiver	Process selection template

Operand 1: Space pointer.

Operand 2: Space pointer.

Description: This instruction materializes the current allocated data space record locks held by the process. The current lock status of the process identified in the process selection template specified by operand 2 is materialized into the receiver identified by operand 1. The materialization identifies each data space record lock which the process has or the process is waiting to obtain.

If the PCS (process control space) pointer is null or all zeros, the lock activity for the process issuing the instruction is materialized.

The process selection template identified by operand 2 must be 16-byte aligned. The format of the process selection template is as follows:

- Process selection — Char(16)
  - Process identification — System pointer
- Lock selection — Char(8)
  - Materialize held locks — Bit 0

    1 = Materialize
    0 = Do not materialize
  - Materialize locks waited for — Bit 1

    1 = Materialize
    0 = Do not materialize
  - Reserved — Bits 2-7
  - Reserved — Char(7)

The process identification must be a system pointer to a PCS (process control space) or null, all zeros.

Both of the fields specified under lock selection are bits which determine the locks to be materialized. If the first bit is on, any data base record lock held by the process is materialized. If the second bit is on, any data base record lock the process is waiting for is materialized.

The materialization template identified by operand 1 must be 16-byte aligned. The format of the materialization is as follows:

- Materialization size specification — Char(8)
  - Number of bytes provided for materialization — Bin(4)
  - Number of bytes available for materialization — Bin(4)
- Materialization data — Char(8)

– Count of locks held	Bin(2)
– Count of locks waited for	Bin(2)
– Reserved	Char(4)
• Locks held identification (repeated for each lock held)	Char(32)
– Data space identification	System pointer
– Relative record number	Bin(4)
– Lock state being described Hex C0 = DLRD lock state Hex F8 = DLUP lock state All other values are reserved.	Char(1)
– Reserved	Char(11)
• Locks waited for identification (repeated for each lock waited for)	Char(32)
– Data space identification	System pointer
– Relative record number	Bin(4)
– Lock state being described Hex C0 = DLRD lock state Hex F8 = DLUP lock state All other values are reserved.	Char(1)
– Reserved	Char(11)

The first 4 bytes of the materialization identify the total quantity of bytes that may be used by the instruction. This value is supplied as input to the instruction and is not modified by the instruction. A value of less than 8 causes the materialization length exception to be signaled.

The second 4 bytes of the materialization identify the total quantity of bytes available to be materialized. The instruction materializes as many bytes as can be contained in the area specified as the receiver. If the byte area identified by the receiver is greater than that required to contain the information requested, the excess bytes are unchanged. No exceptions are signaled in the event that the receiver contains insufficient area for the materialization, other than the materialization length exception described previously.

The count of locks held contains the number of locks held by the process. One system pointer to the data space, relative record number in the data space, and lock state is materialized in the area identified as locks held identification for each lock. These fields contain data only if held locks are selected for materialization.

The count of locks waited for contains the number of locks that the process is waiting for. One system pointer to the data space, relative record number in the data space, and lock state is materialized in the area identified as locks waited for identification for each lock waited for. These fields contain data only if locks waited for are selected for materialization.

Authorization Required

- Retrieve
  - Contexts referenced for address resolution

Lock Enforcement

- Materialize
  - Contexts referenced for address resolution

Exceptions

Exception		Operands 1	Operands 2	Other
06	Addressing			
	01 space addressing violation	X	X	
	02 boundary alignment	X	X	
	03 range	X	X	
	06 optimized addressability invalid	X	X	
08	Argument/parameter			
	01 parameter reference violation	X	X	
0A	Authorization			
	01 unauthorized for operation		X	X
10	Damage encountered			
	04 system object damage state		X	X
	05 authority verification terminated due to damaged object			X
	44 partial system object damage			X
1A	Lock state			
	01 invalid lock state		X	
1C	Machine-dependent exception			
	03 machine storage limit exceeded			X
20	Machine support			
	02 machine check			X
	03 function check			X
22	Object access			
	01 object not found	X	X	
	02 object destroyed	X	X	
	03 object suspended	X	X	
	07 authority verification terminated due to destroyed object			X
24	Pointer specification			
	01 pointer does not exist	X	X	
	02 pointer type invalid	X	X	

Exception		Operands 1	2	Other
2A	Program creation			
	06 invalid operand type	X	X	
	07 invalid operand attribute	X	X	
	08 invalid operand value range	X	X	
	0A invalid operand length	X		
	0C invalid operand odt reference	X	X	
	0D reserved bits are not zero	X	X	X
2E	Resource control limit			
	01 user profile storage limit exceeded			X
32	Scalar specification			
	01 scalar type invalid	X	X	
36	Space management			
	01 space extension/truncation			X
38	Template specification			
	01 template value invalid		X	
	03 materialization length exception	X		

10.8 Materialize Selected Locks (MATSELLK)

Op Code (Hex)	Operand 1	Operand 2
033E	Receiver	Object or space location template

Operand 1: Space pointer.

Operand 2: System pointer or space pointer data object.

Description: The locks held by the process issuing this instruction for the object or space location referenced by operand 2 are materialized into the template specified by operand 1. The format of the materialization template is as follows:

- Materialization size specification — Char(8)
  - Number of bytes provided for materialization — Bin(4)
  - Number of bytes available for materialization — Bin(4)
- Cumulative lock status for all locks on operand 2 — Char(1)
  - Lock state — Char(1)

LSRD	Bit 0
LSRO	Bit 1
LSUP	Bit 2
LEAR	Bit 3
LENR	Bit 4

  - Reserved (binary 0) — Bits 5-7
- Reserved — Char(3)
- Number of lock entries — Bin(2)
- Reserved — Char(2)
- Lock status (repeated for each lock currently allocated) — Char(2)
  - Lock state — Char(1)

    Hex 80 = LSRD lock request
    Hex 40 = LSRO lock request
    Hex 20 = LSUP lock request
    Hex 10 = LEAR lock request
    Hex 08 = LENR lock request

    All other values are reserved
  - Status of lock — Char(1)

Reserved (binary 0)	Bits 0-5
Implicit lock	Bit 6

    0 = Not implicit lock
    1 = Is implicit lock

Reserved (binary 1)	Bit 7

The first 4 bytes of the materialization identifies the total number of bytes that may be used by the instruction. This value is supplied as input to the instruction and is not modified by the instruction. A value of less than 8 causes the materialization length exception to be signaled.

The second 4 bytes of the materialization identifies the total quantity of bytes available to be materialized. The instruction materializes as many bytes as can be contained in the area specified as the receiver. If the byte area identified by the receiver is greater than that required to contain the information requested, then the excess bytes are unchanged. No exceptions are signaled in the event that the receiver contains insufficient area for the materialization, other than the materialization length exception described previously.

A space pointer machine object cannot be specified for operand 2.

Authorization

- Retrieve
  - Context referenced by address resolution

Lock Enforcement

- Materialize
  - Contexts referenced for address resolution

Exceptions

Exception		Operands 1	Operands 2	Other
06	Addressing			
	01 space addressing violation	X	X	
	02 boundary alignment	X	X	
	03 range	X	X	
	06 optimized addressability invalid	X	X	
08	Argument/parameter			
	01 parameter reference violation	X	X	
0A	Authorization			
	02 unauthorized for operation		X	
10	Damage encountered			
	04 system object	X	X	X
	05 authority verification terminated due to damaged object			X
	44 partial system object damage			X
1A	Lock state			
	01 invalid lock state		X	
1C	Machine-dependent exception			
	03 machine storage limit exceeded			X
20	Machine support			

Exception		Operands 1	2	Other
	02 machine check			X
	03 function check			X
22	Object access			
	01 object not found	X	X	
	02 object destroyed	X	X	
	03 object suspended	X	X	
	07 authority verification terminated due to destroyed object			X
24	Pointer specification			
	01 pointer does not exist	X	X	
	02 pointer type invalid	X	X	
	03 pointer addressing invalid object		X	
28	Process state			
	02 process control space not associated with a process		X	
2A	Program creation			
	06 invalid operand type	X	X	
	07 invalid operand attribute	X	X	
	08 invalid operand value range	X	X	
	0A invalid operand length	X		
	0C invalid operand odt reference	X	X	
	0D reserved bits are not zero	X	X	X
2E	Resource control limit			
	01 user profile storage limit exceeded			X
32	Scalar specification			
	01 scalar type invalid	X	X	
36	Space management			
	01 space extension/truncation			X
38	Template specification			
	03 materialization length exception	X		

10.9 Transfer Object Lock (XFRLOCK)

Op Code (Hex)	Operand 1	Operand 2
0382	Receiving process control space	Lock transfer template

Operand 1: System pointer.

Operand 2: Space pointer.

Description: The receiving process (operand 1) is allocated the locks designated in the lock transfer template (operand 2). Upon completion of the transfer lock request, the current process no longer holds the transferred lock(s).

Operand 2 identifies the objects and the associated lock states that are to be transferred to the receiving process. The space contains a system pointer to each object that is to have a lock transferred and a byte which defines whether this entry is active. If the entry is active, the space also contains the lock states to be transferred. Operand 2 must be aligned on a 16-byte boundary. The format is as follows:

- Number of lock transfer requests in template — Bin(4)
- Offset to lock state selection bytes (1 byte for each lock transfer request) — Bin(2)
- Reserved (binary 0) — Char(8)*
- Reserved — Char(1)
  - Reserved — Bits 0-6*
  - Reserved (binary 0) — Bit 7
- Reserved (binary 0) — Char(1)
- Object lock(s) to be transferred — System pointer (one for each object lock to be transferred)
- Lock state selection (repeated for each pointer in the template) — Char(1)
  - Lock state to transfer. Only one state may be requested. (1 = transfer) — Bits 0-4

    LSRD — Bit 0
    LSRO — Bit 1
    LSUP — Bit 2
    LEAR — Bit 3
    LENR — Bit 4
  - Reserved (binary 0) — Bit 5*
  - Lock count — Bit 6

    0 = The current lock count is transferred.
    1 = A lock count of 1 is transferred.

- Entry active indicator Bit 7

  0 = Entry not active This entry is not used.
  1 = Entry active This lock is transferred.

Note: Entries indicated by an asterisk are ignored by the instruction.

If the receiving process is issuing the instruction, then no operation is performed, and no exception is signaled. The lock count transferred is either the lock count held by the transferring process or a count of 1. If the receiving process already holds an identical lock, then the final lock count is the sum of the count originally held by the receiving process and the transferred count.

Only locks currently allocated to the process issuing the instruction can be transferred. If the transfer of an allocated lock would result in the violation of the lock allocation rules, then the lock cannot be transferred. An implicit lock may not be transferred.

No locks are transferred if an entry in the template is invalid.

The locks specified by operand 2 are transferred sequentially and individually. If one lock cannot be transferred because the process does not hold the indicated lock on the object, then exception data is saved to identify the lock that could not be transferred. Processing of the next lock to be transferred continues.

After all locks specified in operand 2 have been processed, the object lock transferred event is signaled to the process receiving the locks if any locks were transferred. If any lock was not transferred, the invalid object lock transfer request exception is signaled.

When an object lock is transferred, the transferring process synchronously loses the record of the lock, and the object is locked to the receiving process. However, the receiving process obtains the lock asynchronously after the instruction currently being executed is completed. If the transferring process holds multiple locks for the object, any lock states not transferred are retained in the process.

Authorization Required

- Retrieve
  - Contexts referenced for address resolution

Lock Enforcement

- Materialize
  - Contexts referenced for address resolution

Exceptions

Exception		Operands 1	Operands 2	Other
06	Addressing			
	01 space addressing violation	X	X	
	02 boundary alignment	X	X	
	03 range	X	X	
	06 optimized addressability invalid	X	X	

Exception		Operands 1	2	Other
08	Argument/parameter			
	01 parameter reference violation	X	X	
0A	Authorization			
	01 unauthorized for operation		X	
10	Damage encountered			
	04 system object damage state	X	X	X
	05 authority verification terminated due to damaged object			X
	44 partial system object damage	X	X	X
1A	Lock state			
	01 invalid lock state			X
	04 invalid object lock transfer request		X	
1C	Machine-dependent exception			
	03 machine storage limit exceeded			X
20	Machine support			
	02 machine check			X
	03 function check			X
22	Object access			
	01 object not found	X	X	
	02 object destroyed	X	X	
	03 object suspended	X	X	
	07 authority verification terminated due to destroyed object			X
24	Pointer specification			
	01 pointer does not exist	X	X	
	02 pointer type invalid	X	X	
	03 pointer addressing invalid object		X	
28	Process state			
	02 process control space not associated with a process	X		
2A	Program creation			
	06 invalid operand type	X	X	
	07 invalid operand attribute	X	X	
	08 invalid operand value range	X	X	
	0C invalid operand odt reference	X	X	
	0D reserved bits are not zero	X	X	X
2E	Resource control limit			

Exception		Operands 1	2	Other
	01 user profile storage limit exceeded			X
36	Space management			
	01 space extension/truncation			X
38	Template specification			
	01 template value invalid		X	

10.10 Unlock Object (UNLOCK)

Op Code (Hex)	Operand 1
03F1	Unlock template

Operand 1: Space pointer.

Description: The instruction releases the object locks that are specified in the unlock template. The template specified by operand 1 identifies the system objects and the lock states (on those objects) that are to be released. The unlock template must be aligned on a 16-byte boundary. The format is as follows:

- Number of unlock requests in template — Bin(4)
- Offset to lock state selection bytes — Bin(2)
- Reserved (binary 0) — Char(8)*
- Unlock option — Char(1)
  - Reserved (binary 0) — Bits 0-3*
  - Unlock type — Bits 4-5

    00 = Unlock specific locks now allocated to process
    01 = Cancel specific asynchronously waiting lock request or allocated locks
    10 = Cancel all asynchronously waiting lock requests
    11 = Invalid
  - Reserved (binary 0) — Bit 6*
  - Reserved (binary 0) — Bit 7
  - Reserved (binary 0) — Char(1)
- Object to unlock (one for each unlock request) — System pointer
- Unlock options (repeated for unlock request) — Char(1)
  - Lock state to unlock (only one state can be selected) (1 = unlock) — Bits 0-4

LSRD	Bit 0
LSRO	Bit 1
LSUP	Bit 2
LEAR	Bit 3
LENR	Bit 4
  - Lock count option — Bit 5

    0 = Lock count reduced by 1
    1 = All locks are unlocked- The set lock count = 0
  - Reserved (binary 0) — Bit 6*
  - Entry active indicators — Bit 7

    0 = Entry not active- This entry is not used.
    1 = Entry active- These locks are unlocked.

Note: Entries indicated by an asterisk are ignored by the instruction.

If all asynchronous lock waits are being canceled (unlock type 10), then system pointers to the objects and unlock options for each object are not required. If the asynchronous lock fields are provided in the template, then the data is ignored.

Unlock type 01 attempts to cancel an asynchronous lock request that is identical to the one defined in the template. After the instruction attempts to cancel the specified request, program execution continues just as if unlock type 00 had been selected. A waiting lock request is canceled if the number of active requests in the template, the objects, the objects corresponding lock states, and the order of the active entries in the template all match.

When a lock is released, the lock count is reduced by 1 or set to 0 in the specified state. This option is specified by the lock count option parameter.

If unlock type 01 is specified and the unlock count option for an object lock is 0 (lock count reduced by 1), then a successful cancel satisfies this request, and no additional locks on the object are unlocked. If the unlock count option for an object lock is set to 1 (set lock count to 0), the results of the cancel are disregarded, and all held locks on the object are unlocked.

Specific locks can be unlocked only if they are allocated to the process issuing the unlock instruction. Implicit locks may not be unlocked with this instruction. No locks are unlocked if an entry in the template is invalid.

Object locks to unlock are processed sequentially and individually. If one specific object lock cannot be unlocked because the process does not hold the indicated lock on the object, then exception data is saved, but processing of the instruction continues.

After all requested object locks have been processed, the invalid unlock request exception is signaled if any object lock was not unlocked.

If unlock type 01 is selected and the cancel attempt is unsuccessful, an invalid unlock request exception is signaled when any object lock in the template is not unlocked.

Authorization Required

- Retrieve
  - Contexts referenced for address resolution

Lock Enforcement

- Materialize
  - Contexts referenced for address resolution

Exceptions

Exception		Operands 1	Other
06	Addressing		
	01 space addressing violation	X	
	02 boundary alignment	X	
	03 range	X	

Exception		Operands 1	Other
	06 optimized addressability invalid	X	
08	Argument/parameter		
	01 parameter reference violation	X	
0A	Authorization		
	01 unauthorized for operation	X	
10	Damage encountered		
	04 system object damage state	X	X
	05 authority verification terminated due to damaged object		X
	44 partial system object damage	X	X
1A	Lock state		
	01 invalid lock state		X
	03 invalid unlock request	X	
1C	Machine-dependent exception		
	03 machine storage limit exceeded		X
20	Machine support		
	02 machine check		X
	03 function check		X
22	Object access		
	01 object not found	X	
	02 object destroyed	X	
	03 object suspended	X	
	07 authority verification terminated due to destroyed object		X
24	Pointer specification		
	01 pointer does not exist	X	
	02 pointer type invalid	X	
2A	Program creation		
	06 invalid operand type	X	
	07 invalid operand attribute	X	
	08 invalid operand value range	X	
	0C invalid operand odt reference	X	
	0D reserved bits are not zero	X	X
2E	Resource control limit		
	01 user profile storage limit exceeded		X
36	Space management		
	01 space extension/truncation		X

Exception		Operands 1	Other
38	Template specification		
	01 template value invalid	X	

10.11 Unlock Space Location (UNLOCKSL)

Op Code (Hex)	Operand 1	Operand 2
03F2	Space location	Lock type

Operand 1: Space pointer data object.

Operand 2: Char(1) scalar.

Description: The lock type specified by operand 2 is removed from the space location identified by operand 1 (the lock must be held by the process that issues the instruction). The space location specified by operand 1 need not exist when this instruction is issued, although the space pointer must be a valid pointer as used to lock the space location. When multiple locks of the same lock state for the same space location need to be unlocked, this instruction must be issued for each lock held for the space location. If an attempt is made to unlock a space location lock not held by the process, an invalid space location unlock exception is signaled.

Following is the format of operand 2:

- Lock state to be unlocked Char(1)

  Hex 80 = LSRD lock
  Hex 40 = LSRO lock
  Hex 20 = LSUP lock
  Hex 10 = LEAR lock
  Hex 08 = LENR lock

  All other values are reserved.

A space pointer machine object cannot be specified for operand 1.

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for this instruction.

Exceptions

Exception		Operands 1	Operands 2	Other
06	Addressing			
	01 space addressing violation	X		
	02 boundary alignment	X		
	03 range	X		
	06 optimized addressability invalid	X		
08	Argument/parameter			
	01 parameter reference violation	X		
10	Damage encountered			
	04 system object	X		X
	44 partial system object damage			X
1A	Lock state			

Exception		Operands 1	2	Other
	05 invalid space location unlock	X		
1C	Machine-dependent exception			
	03 machine storage limit exceeded			X
	06 machine lock limit exceeded			X
20	Machine support			
	02 machine check			X
	03 function check			X
22	Object access			
	02 object destroyed	X	X	
24	Pointer specification			
	01 pointer does not exist	X		
	02 pointer type invalid	X		
2A	Program creation			
	06 invalid operand type	X		
	07 invalid operand attribute	X		
	08 invalid operand value range	X		
	0C invalid operand odt reference	X		
	0D reserved bits are not zero	X	X	X
2E	Resource control limit			
	01 user profile storage limit exceeded			X
32	Scalar specification			
	01 scalar type invalid	X	X	
	03 scalar value invalid		X	
36	Space management			
	01 space extension/truncation			X

Chapter 11. Exception Management Instructions

This chapter describes all instructions used for exception management. These instructions are in alphabetic order. For an alphabetic summary of all the instructions, see Appendix A, "Instruction Summary."

11.1 Materialize Exception Description (MATEXCPD)

Op Code (Hex)	Operand 1	Operand 2	Operand 3
03D7	Attribute receiver	Exception description	Materialization option

Operand 1: Space pointer.

Operand 2: Exception description.

Operand 3: Character(1) scalar.

Description: The instruction materializes the attributes (operand 3) of an exception description (operand 2) into the receiver specified by operand 1.

The template identified by operand 1 must be a 16-byte aligned area in the space if the materialization option is hex 00.

The first 4 bytes of the materialization identify the total number of bytes that may be used by the instruction. This value is supplied as input to the instruction and is not modified by the instruction. A value of less than 8 causes the materialization length exception to be signaled.

The second 4 bytes of the materialization identify the total number of bytes available to be materialized. The instruction materializes as many bytes as can be contained in the area specified as the receiver. If the byte area identified by the receiver is greater than that required to contain the information requested, then the excess bytes are unchanged. No exceptions (other than the materialization length exception) are signaled in the event that the receiver operand contains insufficient area for the materialization.

Operand 2 identifies the exception description to be materialized.

The value of operand 3 specifies the materialization option. If the materialization option is hex 00, the format of the exception description materialization is as follows:

- Template size — Char(8)
  - Number of bytes provided for materialization — Bin(4)
  - Number of bytes available for materialization — Bin(4)
- Control flags — Char(2)
  - Exception handling action — Bits 0-2

000 = Do not handle. (Ignore occurrence of exception and continue processing.)
001 = Do not handle. (Disable this exception description and continue to search this invocation for another exception description to handle the exception.)
010 = Do not handle. (Continue to search for an exception description by resignaling the exception to the preceding invocation.)
100 = Defer handling. (Save exception data for later exception handling.)
101 = Pass control to the specified exception handler.

- No data — Bit 3

  0 = Exception data is returned
  1 = Exception data is not returned

- Reserved (binary 0) — Bit 4
- User data indicator — Bit 5

  0 = User data not present
  1 = User data present

- Reserved (binary 0) — Bits 6-7
- Exception handler type — Bits 8-9

  00 = External entry point
  01 = Internal entry point
  10 = Branch point

- Reserved (binary 0) — Bits 10-15

- Instruction number to be given control (if internal entry point or branch point; otherwise, 0) — Ubin(2)
- Length of compare value (maximum of 32 bytes) — Bin(2)
- Compare value (size established by value of length of compare value parameter) — Char(32)
- Number of exception IDs — Bin(2)
- System pointer to the exception handling program if an external exception handler is specified — System pointer
- Pointer to user data (not present if value of user data indicator is binary 0) — Space pointer
- Exception ID (one for each exception ID dictated by the number of exception IDs attribute) — Char(2)

If the materialization option is hex 01, the format of the materialization is as follows:

- Template size — Char(8)
  - Number of bytes provided for materialization — Bin(4)
  - Number of bytes available for materialization — Bin(4)
- Control flags — Char(2)
  - Exception handling action — Bits 0-2

000 = Do not handle. (Ignore occurrence of exception and continue processing.)
001 = Do not handle. (Disable this exception description and continue to search this invocation for another exception description to handle the exception.)
010 = Do not handle. (Continue to search for an exception description by resignaling the ex- ception to the preceding invocation.)
100 = Defer handling. (Save exception data for later exception handling.)
101 = Pass control to the specified exception handler.

- No data — Bit 3

  0 = Exception data is returned
  1 = Exception data is not returned

- Reserved (binary 0) — Bit 4
- User data indicator — Bit 5

  0 = User data not present
  1 = User data present

- Reserved (binary 0) — Bits 6-15

If the materialization option is hex 02, the format of the materialization is as follows:

- Template size — Char(8)
  - Number of bytes provided for materialization — Bin(4)
  - Number of bytes available for materialization — Bin(4)
- Compare value length (maximum of 32 bytes) — Bin(2)
- Compare value — Char(32)

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for this instruction.

Exceptions

Exception		Operands 1	2	3	Other
06	Addressing				
	01 space addressing violation	X		X	
	02 boundary alignment	X		X	
	03 range	X		X	
	06 optimized addressability invalid	X		X	
08	Argument/parameter				
	01 parameter reference violation	X		X	
10	Damage encountered				
	04 system object damage state	X	X	X	X
	44 partial system object damage	X	X	X	X
1C	Machine-dependent exception				

Exception		Operands 1	2	3	Other
	03 machine storage limit exceeded				X
20	Machine support				
	02 machine check				X
	03 function check				X
22	Object access				
	01 object not found	X		X	
	02 object destroyed	X		X	
	03 object suspended	X		X	
24	Pointer specification				
	01 pointer does not exist	X		X	
	02 pointer type invalid	X		X	
2A	Program creation				
	06 invalid operand type	X	X	X	
	07 invalid operand attribute	X		X	
	08 invalid operand value range	X		X	
	0A invalid operand length	X		X	
	0C invalid operand odt reference	X	X	X	
	0D reserved bits are not zero	X	X	X	X
2E	Resource control limit				
	01 user profile storage limit exceeded				X
32	Scalar specification				
	03 scalar value invalid			X	
36	Space management				
	01 space extension/truncation				X
38	Template specification				
	03 materialization length exception	X			

11.2 Modify Exception Description (MODEXCPD)

Op Code (Hex)	Operand 1	Operand 2	Operand 3
03EF	Exception description	Modifying attributes	Modification option

Operand 1: Exception description.

Operand 2: Space pointer, or character(2) constant.

Operand 3: Character(1) scalar.

Description: The exception description attributes specified by operand 3 are modified with the values of operand 2.

Operand 1 references the exception description.

Operand 2 specifies the new attribute values. Operand 2 may be either a character constant or a space pointer to the modification template. When operand 3 is a constant, operand 2 is a character constant; when operand 3 is not a constant, operand 2 is a space pointer.

The value of operand 3 specifies the modification option. If the modification option is hex 01 and operand 2 specifies a space pointer, the format of the modifying attributes pointed to by operand 2 is as follows:

- Template size — Char(8)
  - Number of bytes provided for materialization (must be at least 10) — Bin(4)
  - Number of bytes available for materialization — Bin(4)*
- Control flags — Char(2)
  - Exception handling action — Bits 0-2

    000 = Do not handle. (Ignore occurrence of exception and continue processing.)
    001 = Do not handle. (Disable this exception description and continue to search this invocation for another exception description to handle the exception.)
    010 = Do not handle. (Continue to search for an exception description by resignaling the exception to the preceding invocation.)
    100 = Defer handling. (Save exception data for later exception handling.)
    101 = Pass control to the specified exception handler.
  - No data — Bit 3

    0 = Exception data is returned
    1 = Exception data is not returned
  - Reserved (binary 0) — Bits 4-15

If the exception description was in the deferred state prior to the modification, the deferred signal, if present, is lost.

When the option to not return exception data is selected, no data is returned for the Retrieve Exception Data or Test Exception instructions, and the number of bytes available for the materialization field is set to 0. This option can also be selected in the ODT definition of the exception description.

If the modification option of operand 3 is a constant value of hex 01, then operand 2 may specify a character constant. The operand 2 constant has the same format as the control flags entry previously described.

If the modification option is hex 02, then operand 2 must specify a space pointer. The format of the modification is as follows:

- Template size — Char(8)
  - Number of bytes provided (must be at least 10 plus the length of the compare value in the exception description) — Bin(4)
  - Number of bytes available for materialization — Bin(4)*
- Compare value length (maximum of 32 bytes) — Bin(2)*
- Compare value — Char(32)

Note: Entries shown here with an asterisk (*) are ignored by the instruction.

The number of bytes in the compare value is dictated by the compare value length specified in the exception description as originally specified in the object definition table.

An external exception handling program can be modified by resolving addressability to a new program into the system pointer designated for the exception description.

The presence of user data is not a modifiable attribute of exception descriptions. If the exception description has user data, it can be modified by changing the value of the data object specified in the exception description.

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for this instruction.

Exceptions

Exception		Operands 1	2	3	Other
06	Addressing				
	01 space addressing violation		X	X	
	03 range		X	X	
	06 optimized addressability invalid		X	X	
08	Argument/parameter				
	01 parameter reference violation		X	X	
10	Damage encountered				
	04 system object damage state	X	X	X	X

Exception		Operands 1	2	3	Other
	44 partial system object damage	X	X	X	X
1C	Machine-dependent exception				
	03 machine storage limit exceeded				X
20	Machine support				
	02 machine check				X
	03 function check				X
22	Object access				
	01 object not found		X	X	
	02 object destroyed		X	X	
	03 object suspended		X	X	
24	Pointer specification				
	01 pointer does not exist		X	X	
	02 pointer type invalid		X	X	
2A	Program creation				
	06 invalid operand type	X	X	X	
	07 invalid operand attribute		X	X	
	08 invalid operand value range		X	X	
	0A invalid operand length			X	
	0C invalid operand odt reference	X	X	X	
	0D reserved bits are not zero	X	X	X	X
2E	Resource control limit				
	01 user profile storage limit exceeded				X
32	Scalar specification				
	03 scalar value invalid			X	
36	Space management				
	01 space extension/truncation				X
38	Template specification				
	01 template value invalid		X		
	02 template size invalid		X		

11.3 Retrieve Exception Data (RETEXCPD)

Op Code (Hex)	Operand 1	Operand 2
03E2	Receiver	Retrieve options

Operand 1: Space pointer.

Operand 2: Character(1) scalar (fixed-length).

Description: The data related to a particular occurrence of an exception is returned and placed in the specified space.

Operand 1 is a space pointer that identifies the receiver template. The template identified by operand 1 must be 16-byte aligned in the space.

The value of operand 2 specifies the type of exception handler for which the exception data is to be retrieved. The exception handler may be a branch point exception handler, an internal entry point exception handler, or an external entry point exception handler.

An exception state of process invalid exception is signaled to the invocation issuing the Retrieve Exception Data instruction if the retrieve option is not consistent with the process's exception handling state. For example, the exception is signaled if the retrieve option specifies retrieve for internal entry point exception handler and the process exception state indicates that an internal exception handler has not been invoked.

The first 4 bytes of the materialization identify the total number of bytes that may be used by the instruction. This value is supplied as input to the instruction and is not modified by the instruction. A value of less than 8 causes the materialization length exception.

The second 4 bytes of the materialization identify the total number of bytes available to be materialized. The instruction materializes as many bytes as can be contained in the area specified as the receiver. If the byte area identified by the receiver is greater than that required to contain the information requested, then the excess bytes are unchanged. No exceptions (other than the materialization length exception) are signaled in the event that the receiver contains insufficient area for the materialization.

After an invocation has been destroyed, exception data associated with a signaled exception description within that invocation is lost.

The format of operand 1 for the materialization is as follows:

- Template size — Char(8)
  - Number of bytes provided for retrieval — Bin(4)
  - Number of bytes available for retrieval — Bin(4)
- Exception identification — Char(2)
- Compare value length (maximum of 32 bytes) — Bin(2)

- Compare value — Char(32)
- Reserved (binary 0) — Char(4)
- Exception specific data — Char(*)
- Signaling program invocation — Space pointer
- Signaled program invocation — Space pointer
- Signaling program instruction address — Ubin(2)
- Signaled program instruction address — Ubin(2)
- Machine-dependent data — Char(10)

The signaling program invocation address entry locates the invocation entry in the PASA (process automatic storage area) that corresponds to the invocation that caused the exception to be signaled. For machine exceptions, this space pointer locates the invocation executing when the exception occurred. For user-signaled exceptions, this space pointer locates the invocation that executed the Signal Exception instruction. The signaling program instruction address entry locates the instruction that caused the exception to be signaled.

The signaled program invocation entry locates the invocation entry in the PASA that is signaled to handle the exception. This invocation is the last invocation signaled or resignaled to handle the exception. For machine exceptions, the first invocation signaled is the invocation incurring the exception. For user-signaled exceptions, the Signal Exception instruction may initially locate the current or any previous invocation. If the invocation to be signaled handles the exception by resignaling the exception, the immediately previous invocation is considered to be the last signaled invocation. This may occur repetitively until no more prior invocations exist in the process and the signaled program invocation entry is assigned a value of binary 0. If an invocation to be signaled handles the exception in any manner other than resignaling or does not handle the exception, that invocation is considered to be the last signaled.

The signaled program instruction address entry specifies the number of the instruction that is currently being executed in the signaled invocation.

The machine extends the area beyond the exception specific data area with binary 0's so that the pointers to program invocations are properly aligned.

The operand 2 values are defined as follows:

- Retrieve options — Char(1)

  - Hex 00 = Retrieve for a branch point exception handler
  - Hex 01 = Retrieve for an internal entry point exception handler
  - Hex 02 = Retrieve for an external entry point exception handler

If the exception data retention option is set to 1 (do not save), the number of bytes available for retrieval is set to 0.

Exception data is always available to the process default exception handler.

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for this instruction.

Exceptions

Exception		Operands 1	Operands 2	Other
06	Addressing			
	01 space addressing violation	X	X	
	02 boundary alignment	X	X	
	03 range	X	X	
	06 optimized addressability invalid	X	X	
08	Argument/parameter			
	01 parameter reference violation	X	X	
10	Damage encountered			
	04 system object damage state	X	X	X
	44 partial system object damage	X	X	X
16	Exception management			
	02 exception state of process invalid		X	
1C	Machine-dependent exception			
	03 machine storage limit exceeded			X
20	Machine support			
	02 machine check			X
	03 function check			X
22	Object access			
	01 object not found	X	X	
	02 object destroyed	X	X	
	03 object suspended	X	X	
24	Pointer specification			
	01 pointer does not exist	X	X	
2A	Program creation			
	06 invalid operand type	X	X	
	07 invalid operand attribute	X	X	
	08 invalid operand value range	X	X	
	0A invalid operand length	X	X	
	0C invalid operand odt reference	X	X	
	0D reserved bits are not zero	X	X	X
2E	Resource control limit			
	01 user profile storage limit exceeded			X
32	Scalar specification			
	03 scalar value invalid		X	
36	Space management			

Exception		Operands 1	2	Other
	01 space extension/truncation			X
38	Template specification			
	03 materialization length exception	X		

11.4 Return From Exception (RTNEXCP)

Op Code (Hex)	Operand 1
03E1	Return target

Operand 1: Space pointer.

Description: An internal exception handler subinvocation or an external exception handler invocation is terminated, and control is passed to the specified instruction in the specified invocation.

The template identified by operand 1 must be 16-byte aligned in the space. It specifies the target invocation and target instruction in the invocation where control is to be passed. The format of operand 1 is as follows:

- Invocation address — Space pointer
- Reserved (binary 0) — Char(1)
- Action — Char(1)
  - Reserved (binary 0) — Bits 0-5
  - Unstack Option — Bit 6

    0 = The action performed is determined by the setting of the following Action Code (bit 7).
    1 = If the excpetion handler is an internal exception handler, resume execution with the instruction that follows RTNEXCP instruction and terminate the internal exception handler subinvocation.
  - Action Code — Bit 7

    0 = Re-execute the instruction that caused the exception or the instruction that invoked the invocation.
    1 = Resume execution with the instruction that follows the instruction that caused the exception or resume execution with the instruction that follows the instruction that invoked the invocation.
  - Reserved (binary 0) — Char(1)

The invocation address entry is a space pointer that locates an invocation entry in the PASA (process automatic storage area) chain to which control will be passed. The current instruction in an invocation is the one that caused another invocation to be created. If an event handler was invoked, then the current instruction is the instruction that executed prior to the invocation of the event handler.

The unstack option is only valid when issued in an internal exception handler subinvocation and is ignored for an external exception handler invocation. This option will cause the internal exception handler subinvocation to be terminated and control will resume at the intruction immediatly following the RTNEXCP instruction. In effect, this option will cause the current subinvocation to be unstacked.

If the action code is 0, then the current instruction of the addressed invocation is reexecuted. If the action code is 1, execution resumes with the instruction following the current instruction of the addressed invocation.

When a Return From Exception instruction returns control to an invocation that was interrupted by an event, the action code in the operand 1 template is ignored and execution continues at the point of interruption. That is, the interrupted instruction is not reexecuted and execution of the instruction is completed as if no interruption occurred. For example, if a Dequeue instruction is waiting for a message to arrive on a queue when an event handler is invoked that produces an exception, the exception handler returns control to the interrupted Dequeue instruction and the instruction continues to wait for the message.

The Return From Exception instruction may be issued only from the initial invocation of an external exception handling sequence or from an invocation that has an active internal exception handler.

If the instruction is issued from an invocation that is not an external exception handler and has no internal exception handler subinvocations, the return instruction invalid exception is signaled.

The following table shows the actions performed by the Return From Exception instruction:

Invocation Issuing Instruction	Addressing Own Invocation/Option	Addressing Higher Invocation/Option
Not handling exception	Error 1	Error 1
Handling internal exception(s)	Allowed 2	Allowed 3
Handling external exception(s)	Error 1	Allowed 3
Handling external exception(s) and internal exception(s)	Allowed 2	Allowed 3

1. A return instruction invalid exception is signaled. If there are no more internal exception handler subinvocations active and this invocation is not an external exception handler, the instruction may not be issued.
2. The current internal exception handler subinvocation is terminated.
3. All invocations after the addressed invocation are terminated and execution proceeds within the addressed invocation. Any invocation exit programs set for the terminated invocations will be given control before execution proceeds within the addressed invocation.

Whenever an invocation is terminated, the invocation count in the corresponding activation entry (if any) is decremented by 1.

An action code of 1 specifies completion of an instruction rather than execution of the following instruction if the current instruction in the addressed invocation signaled a size exception or a floating-point inexact result exception.

Note: The previous condition does not apply if any of the above exceptions were explicitly signaled by a Signal Exception instruction.

A Return From Exception instruction cannot be used or recognized in conjunction with a branch point internal exception handler.

If a failure to invoke an invocation exit handler occurs, a failure to invoke program event is signaled.

Exceptions

Exception		Operands 1	Other
06	Addressing		
	01 space addressing violation	X	
	02 boundary alignment	X	
	03 range	X	
	06 optimized addressability invalid	X	
08	Argument/parameter		
	01 parameter reference violation	X	
10	Damage encountered		
	04 system object damage state	X	X
	44 partial system object damage	X	X
16	Exception management		
	03 invalid invocation	X	
1C	Machine-dependent exception		
	03 machine storage limit exceeded		X
20	Machine support		
	02 machine check		X
	03 function check		X
22	Object access		
	02 object destroyed	X	
	03 object suspended	X	
24	Pointer specification		
	01 pointer does not exist	X	
	02 pointer type invalid	X	
2A	Program creation		
	06 invalid operand type	X	
	07 invalid operand attribute	X	
	08 invalid operand value range	X	
	09 invalid branch target operand		X
	0A invalid operand length	X	
	0C invalid operand odt reference	X	
	0D reserved bits are not zero	X	X
2C	Program execution		
	01 return instruction invalid		X

Exception		Operands 1	Other
2E	Resource control limit		
	01 user profile storage limit exceeded		X
36	Space management		
	01 space extension/truncation		X
38	Template specification		
	01 template value invalid	X	

11.5 Sense Exception Description (SNSEXCPD)

Op Code (Hex)	Operand 1	Operand 2	Operand 3
03E3	Attribute receiver	Invocation template	Exception template

Operand 1: Space pointer.

Operand 2: Space pointer.

Operand 3: Space pointer.

Description: The Sense Exception Description instruction searches the invocation specified by operand 2 for an exception description that matches the exception identifier and compare value specified by operand 3 and returns the user data and exception handling action specified in the exception description. The exception descriptions of the invocation are searched in ascending ODT number sequence.

The exception identifier in the exception description can be specified in one of the following ways:

Hex 0000 = Any exception ID will result in a match

Hex nn00 = Any exception ID in class nn will result in a match

Hex nnmm = Only exception ID nnmm will result in a match

If a match on exception ID is detected, the corresponding compare values are matched. If the compare value length in the exception description is less than the compare value in the search template, the length of the compare value in the exception description is used for the match. If the compare value length in the exception description is greater than the compare value in the search template, an automatic mismatch results.

If a match on exception ID and compare value is detected, the exception handling action of the exception description determines which of the following actions is taken:

IGNORE The operand 1 template is materialized.

DISABLE The exception description is bypassed and the search for an exception description continues with the next exception description defined for the invocation.

RESIGNAL The operand 1 template is materialized.

DEFER The operand 1 template is materialized.

HANDLE The operand 1 template is materialized.

If no exception description of the invocation matches the exception ID and compare value of operand 3, the number of bytes available for materialization on the operand 1 template is set to 0.

The template identified by operand 1 must be 16-byte aligned.

The first 4 bytes of the materialization identify the total number of bytes that may be used by the instruction. This value is supplied as input to the instruction and is not modified by the instruction. A value of less than 8 causes the materialization length exception to be signaled.

The second 4 bytes of the materialization identify the total number of bytes available to be materialized. The instruction materializes as many bytes as can be contained in the area specified as the receiver. If the byte area identified by the receiver is greater than that required to contain the information requested, then the excess bytes are unchanged. No exception is signaled in the event the receiver contains insufficient area for the materialization, other than the materialization length exception described previously.

The format of the attribute receiver is as follows:

- Template size — Char(8)
  - Number of bytes provided for materialization — Bin(4)
  - Number of bytes available for materialization — Bin(4)
- Control flags — Char(2)
  - Exception handling action — Bits 0-2

    000 = Do not handle-ignore occurrence of exception and continue processing
    010 = Do not handle-continue search for an exception description by resignaling the exception to the immediately preceding invocation
    100 = Defer handling-save exception data for later exception handling
    101 = Pass control to the specified exception handler
  - No data — Bit 3

    0 = Exception data is returned
    1 = Exception data is not returned
  - Reserved (binary 0) — Bit 4
  - User data indicator — Bit 5

    0 = User data not present
    1 = User data present
  - Reserved (binary 0) — Bits 6-7
  - Exception handler type — Bits 8-9

    00 = External entry point
    01 = Internal entry point
    10 = Branch point
  - Reserved (binary 0) — Bits 10-15
- Relative exception description number — Bin(2)
- Reserved (binary 0) — Char(4)
- Pointer to user data (binary 0 if value of user data indicator is binary 0) — Space pointer

The relative exception description number entry identifies the relative number of the exception description that matched the search criteria. The order of definition of the exception descriptions in the ODT determines the value of the index.

A value of 1 indicates that the first exception description defined in the ODT matched the search criteria.

The template identified by operand 1 must be 16-byte aligned. The invocation address entry is a space pointer that locates an invocation entry in the PASA (process automatic storage area). The invocation is searched for a matching exception description. If the space pointer locates the PASA base entry, the operand 1 template is materialized with the number of bytes available for materialization set to 0. If the space pointer locates neither a valid invocation entry nor the PASA base entry, the invalid invocation address exception is signaled.

The first exception description to search entry specifies the relative number of the exception description to be used to start the search. The number must be a nonzero positive binary number determined by the order of definition of exception descriptions in the ODT. A value of 1 indicates that the first exception description in the invocation is to be used to begin the search. If the value is greater than the number of exception descriptions for the invocation, the operand 1 template is materialized with the number of bytes available for materialization set to 0.

The format of the invocation template is as follows:

- Invocation address — Space pointer
- Reserved (binary 0) — Char(2)
- First exception description to search — Bin(2)

The operand 3 exception template specifies the exception-related data to be used as a search argument. The format of the template is as follows:

- Template size — Char(8)
  - Number of bytes provided for materialization (must be at least 44) — Bin(4)
  - Number of bytes available for materialization — Bin(4)*
- Exception identifier — Char(2)
- Compare value length (maximum of 32) — Bin(2)
- Compare value — Char(32)

Note: Entries noted with an asterisk (*) are ignored by the instruction.

Exceptions

Exception		Operands 1	2	3	Other
06	Addressing				
	01 space addressing violation	X	X	X	
	02 boundary alignment	X	X	X	
	03 range	X	X	X	
	06 optimized addressability invalid	X	X	X	
08	Argument/parameter				
	01 parameter reference violation	X	X	X	

Exception		Operands 1	2	3	Other
10	Damage encountered				
	04 system object damage				X
	44 partial system object damage				X
16	Exception management				
	03 invalid invocation address		X		
1C	Machine-dependent exception				
	03 machine storage limit exceeded				X
20	Machine support				
	02 machine check				X
	03 function check				X
22	Object access				
	01 object not found	X	X	X	
	02 object destroyed	X	X	X	
	03 object suspended	X	X	X	
24	Pointer specification				
	01 pointer does not exist	X	X	X	
	02 pointer type invalid	X	X	X	
2A	Program creation				
	06 invalid operand type	X	X	X	
	07 invalid operand attribute	X	X	X	
	08 invalid operand value range	X	X	X	
	0A invalid operand length	X	X	X	
	0C invalid operand odt reference	X	X	X	
	0D reserved bits are not zero	X	X	X	X
2E	Resource control limit				
	01 user profile storage limit exceeded				X
32	Scalar specification				
	01 scalar type invalid	X	X	X	
36	Space management				
	01 space extension/truncation				X
38	Template specification				
	01 template value invalid		X	X	
	02 template size invalid			X	
	03 materialization length exception	X			

11.6 Signal Exception (SIGEXCP)

Op Code (Hex)	Op Code Operand 1	Operand 2
10CA	Attribute template	Exception data

Operand 1: Space pointer.

Operand 2: Space pointer.

Optional Forms

Mnemonic	Op Code (Hex)	Form Type
SIGEXCPI	18CA	Indicator
SIGEXCPB	1CCA	Branch

Extender: Branch options or indicator options.

If the branch or indicator option is specified in the op code, the extender field must be present along with one or two branch targets (for branch options) or one or two indicator operands (for indicator options). The branch or indicator operands immediately follow the last operand listed above. See Chapter 1. "Introduction" for the encoding of the extender field and the allowed syntax of the branch and indicator operands.

Description: The Signal Exception instruction signals a new exception or resignals an existing exception to the process. Optionally, the instruction branches to one of the specified targets based on the results of the signal and the selected branch options in the extender field, or it sets indicators based on the results of the signal. The signal is presented starting at the invocation identified in the signal template.

The template identified by operand 1 specifies the signal option and starting point. It must be 16-byte aligned in the space with the following format.

- Signaled to invocation address — Space pointer
- Signal option — Char(1)
  - Signal/resignal option — Bit 0

    0 = Signal new exception.
    1 = Resignal currently handled exception (valid only for an external exception handler).
  - Invoke PDEH (process default exception handler) option — Bit 1

    0 = Invoke PDEH if no exception description found for invocation.
    1 = Do not invoke PDEH if no exception description found for invocation (ignore if PASA base entry specified).
  - Exception description search control — Bit 2

    0 = Exception description search control not present
    1 = Exception description present

  - Reserved (binary 0) Bits 3-7
- Reserved (binary 0) Char(1)
- First exception description to search Bin(2)

The signaled to invocation address entry is a space pointer that locates an invocation entry in the PASA (process automatic storage area). The exception is signaled to this invocation. If the space pointer locates the PASA base entry, the exception is signaled to the PDEH. If the space pointer locates neither a valid invocation entry nor the PASA base entry, the invalid invocation address exception is signaled. If the program associated with the invocation has defined an exception description to handle the exception, the specified action is taken; otherwise, the PDEH is invoked unless the invoke PDEH option bit is 1 (the exception is considered ignored). If the PASA base entry is addressed instead of an existing invocation, the PDEH will be invoked.

Exception descriptions of an invocation are searched in ascending ODT number sequence. If the exception description search control is not present, the search begins with the first exception description defined in the ODT. Otherwise, the first exception description to search value identifies the relative number of the exception description to be used to start the search. The value must be a nonzero positive binary number determined by the order of definition of exception descriptions in the ODT. This value is also returned by the Sense Exception Description instruction. A value of 1 indicates that the first exception description in the invocation is to be used to begin the search. If the value is greater than the number of exception descriptions for the invocation, the template value invalid exception is signaled.

The template identified by operand 2 must be 16-byte aligned in the space. It specifies the exception-related data to be passed with the exception signal. The format of the exception data is the same as that returned by the Retrieve Exception Data instruction. The format is as follows:

- Template size Char(8)
  - Number of bytes of data to be signaled (must be at least 48 bytes) Bin(4)
  - Number of bytes available for materialization Bin(4)*
- Exception identification Char(2)
- Compare value length (maximum of 32 bytes) Bin (2)
- Compare value Char(32)
- Reserved (binary 0) Char(4)
- Exception specific data Char(*)

Note: Entries shown here with an asterisk (*) are ignored by the instruction.

Operand 2 is ignored if operand 1 specifies the resignal option, because the exception-related data is the same as for the exception currently being processed; however, it must be specified when signaling a new exception.

The maximum size for exception-related data that is to accompany an exception signaled by the Signal Exception instruction is 32 608 bytes, including the standard signal data.

If an exception ID in an exception description corresponds to the signaled exception, the corresponding compare values are verified. If the compare value length in the exception description is less than the compare value length in the signal template, the length of the compare value in the exception description is used for the match. If the compare value length in the exception description is greater than the compare value length in the signal template, an automatic mismatch results. Machine-signaled exceptions have a 4-byte compare value of binary 0's.

An exception description may monitor for an exception with a generic ID as follows:

Hex 0000 = Any signaled exception ID results in a match.

Hex nn00 = Any signaled exception ID in class nn results in a match.

Hex nnmm = The signaled exception ID must be exactly nnmm in order for a match to occur.

An exception description may be in one of five states, each of which determines an action to be taken when the match criteria on the exception ID and compare value are met.

IGNORE No exception handling occurs. The Signal Exception instruction is assigned a resultant condition of ignored. If a corresponding branch or indicator setting is present, that action takes place.

DISABLE The exception description is bypassed, and the search for a monitor continues with the next exception description defined for the invocation.

RESIGNAL The search for a monitoring exception description is to be reinitiated at the preceding invocation. A resignal from the initial invocation in the process results in the invocation of the process default exception handler. A resignal from an invocation exit program results in an unhandled exception that causes process termination.

DEFER The exception description is signaled, and the Signal Exception instruction is assigned the resultant condition of deferred. If a corresponding branch or indicator setting is present, that action takes place. To take future action on a deferred exception, the exception description must be synchronously tested with the Test Exception instruction in the signaled invocation.

HANDLE Control is passed to the indicated exception handler, which may be a branch point, an internal subinvocation, or an external invocation.

If the exception description is in the ignore or defer state and if the Signal Exception instruction does not specify a branch or indicator condition or if it specifies branch or indicator conditions that are not met, then the instruction following the Signal Exception instruction is executed.

When control is given to an internal or branch point exception handler, all invocations up to, but not including, the exception handling invocation are terminated. Any invocation exit programs set for the terminated invocations will be given control before execution proceeds in the signaled exception handler.

If a failure to invoke an external exception handler or an invocation exit occurs, a failure to invoke program event is signaled. For each destroyed invocation, the

invocation count in the corresponding activation entry (if any) is decremented by 1.

Resultant Conditions: Exception ignored or exception deferred.

Exceptions

Exception		Operands 1	Operands 2	Other
06	Addressing			
	01 space addressing violation	X	X	
	02 boundary alignment	X	X	
	03 range	X	X	
	06 optimized addressability invalid	X	X	
08	Argument/parameter			
	01 parameter reference violation	X	X	
10	Damage encountered			
	04 system object damage state	X	X	X
	44 partial system object damage	X	X	X
16	Exception management			
	02 exception state of process invalid			X
	03 invalid invocation	X		
1C	Machine-dependent exception			
	03 machine storage limit exceeded			X
20	Machine support			
	02 machine check			X
	03 function check			X
22	Object access			
	01 object not found	X	X	
	02 object destroyed	X	X	
	03 object suspended	X	X	
24	Pointer specification			
	01 pointer does not exist	X	X	
	02 pointer type invalid	X	X	
2A	Program creation			
	05 invalid op code extender field			X
	06 invalid operand type	X	X	
	07 invalid operand attribute	X	X	
	08 invalid operand value range	X	X	
	09 invalid branch target operand			X
	0C invalid operand odt reference	X	X	

Exception		Operands 1	2	Other
	0D reserved bits are not zero	X	X	X
2E	Resource control limit			
	01 user profile storage limit exceeded			X
36	Space management			
	01 space extension/truncation			X
38	Template specification			
	01 template value invalid	X		
	02 template size invalid	X		

11.7 Test Exception (TESTEXCP)

Op Code (Hex)	Operand 1	Operand 2
104A	Receiver	Exception description

Operand 1: Space pointer.

Operand 2: Exception description.

Optional Forms

Mnemonic	Op Code (Hex)	Form Type
TESTEXCPI	184A	Indicator
TESTEXCPB	1C4A	Branch

Extender: Branch options.

If the branch or indicator option is specified in the op code, the extender field must be present along with one or two branch targets (for branch options) or one or two indicator targets (for indicator options). The branch or indicator targets immediately follow the last operand listed above. See Chapter 1. "Introduction" for the encoding of the extender field and the allowed syntax of the branch and indicator operands.

Description: The instruction tests the signaled status of the exception description specified in operand 2, and optionally alters the control flow or sets the specified indicators based on the test. Exception data is returned at the location identified by operand 1. The branch or indicator setting occurs based on the conditions specified in the extender field depending on whether or not the specified exception description is signaled.

Operand 2 is an exception description whose signaled status is to be tested. An exception can be signaled only if the referenced exception description is in the deferred state.

Operand 1 addresses a space into which the exception data is placed if an exception identified by the exception description has been signaled.

The template identified by operand 1 must be 16-byte aligned in the space.

The first 4 bytes of the materialization identify the total number of bytes that may be used by the instruction. This value is supplied as input to the instruction and is not modified by the instruction. A value of less than 8 causes the materialization length exception to be signaled.

The second 4 bytes of the materialization identify the total number of bytes available to be materialized. The instruction materializes as many bytes as can be contained in the area specified as the receiver. If the byte area identified by the receiver is greater than that required to contain the information requested, then the excess bytes are unchanged. No exceptions (other than the materialization length exception) are signaled in the event that the receiver contains insufficient area for the materialization.

If the exception description is not in the signaled state, the number of bytes available for the materialization entry is set to binary 0's, and no other bytes are modified. The format of the data returned in operand 1 is as follows:

- Template size — Char(8)
  - Number of bytes provided for materialization — Bin(4)
  - Number of bytes available for materialization (0 if exception description is not signaled) — Bin (4)
- Exception identification — Char(2)
- Compare value length (maximum of 32 bytes) — Bin(2)
- Compare value — Char(32)
- Reserved (binary 0) — Char(4)
- Exception-specific data — Char(*)
- Signaling program invocation address — Space pointer
- Signaled program invocation address — Space pointer
- Signaling program instruction address — Ubin(2)
- Signaled program instruction address — Ubin(2)
- Machine-dependent data — Char(10)

The area beyond the exception-specific data area is extended with binary 0's so that pointers to program invocations are properly aligned.

If no branch options are specified, instruction execution proceeds at the instruction following the Test Exception instruction.

If the exception data retention option is set to 1 (do not save), no data is returned by this instruction.

Resultant Conditions: Exception signaled or exception not signaled.

Exceptions

Exception		Operands 1	2	Other
06	Addressing			
	01 space addressing violation	X		
	02 boundary alignment	X		
	03 range	X		
	06 optimized addressability invalid	X		
08	Argument/parameter			
	01 parameter reference violation	X		
10	Damage encountered			
	04 system object damage state	X	X	X
	44 partial system object damage	X	X	X
16	Exception management			

Exception		Operands 1	2	Other
	01 exception description status invalid		X	
1C	Machine-dependent exception			
	03 machine storage limit exceeded			X
20	Machine support			
	02 machine check			X
	03 function check			X
24	Pointer specification			
	01 pointer does not exist	X		
	02 pointer type invalid	X		
2A	Program creation			
	05 invalid op code extender field			X
	06 invalid operand type	X	X	
	09 invalid branch target operand			X
	0A invalid operand length	X		
	0C invalid operand odt reference	X	X	
	0D reserved bits are not zero	X	X	X
2E	Resource control limit			
	01 user profile storage limit exceeded			X
36	Space management			
	01 space extension/truncation			X
38	Template specification			
	03 materialization length exception	X		

Extended Function Instructions

These instructions provide an extended set of functions which can be used to control and monitor the operation of the machine. Because of the more complicated nature of these instructions, they are more exposed to changes in their operation in different machine implementations than the basic function instructions. Therefore, it is recommended that, where possible, programs avoid using these extended function instructions to minimize the impacts which can arise in moving to different machine implementations.

Chapter 12. Context Management Instructions

This chapter describes the instructions used for context management. These instructions are in alphabetic order. See Appendix A, "Instruction Summary," for an alphabetic summary of all the instructions.

12.1 Materialize Context (MATCTX)

Op Code (Hex)	Operand 1	Operand 2	Operand 3
0133	Receiver	Permanent context, temporary context, or machine context	Materialization options

Operand 1: Space pointer.

Operand 2: System pointer or null.

Operand 3: Character scalar (fixed-length).

Description: Based on the contents of the materialization options specified by operand 3, the symbolic identification and/or system pointers to all or a selected set of the objects addressed by the context specified by operand 2 are materialized into the receiver specified by operand 1. If operand 2 is null, then the machine context is materialized.

The materialization control information requirements field in the materialization options operand specifies the information to be materialized for each selected entry. Symbolic identification and system pointers identifying objects addressed by the context can be materialized based on the bit setting of this parameter. The materialization control selection criteria field specifies the context entries from which information is to be presented. The type code, subtype code, and name fields contain the selection criteria when a selective materialization is specified.

When type code or type/subtype codes are part of the selection criteria, only entries that have the specified codes are considered. When a name is specified as part of the selection criteria, the N characters in the search criteria are compared against the N characters of the context entry, where N is defined by the name length field in the materialization options. The remaining characters (if any) in the context entry are not used in the comparison.

The materialization options operand has the following format:

- Materialization control — Char(2)
  - Information requirements (1 = materialize) — Char(1)

    Reserved (binary 0) — Bits 0-3
    Validation — Bit 5
    0 = Validate system pointers
    1 = No validation
    System pointers — Bit 6
    Symbolic identification — Bit 7
  - Selection criteria — Char(1)

    Hex 00 — All context entries
    Hex 01 — Type code selection
    Hex 02 — Type code/subtype code selection
    Hex 04 — Name selection
    Hex 05 — Type code/name selection
    Hex 06 — Type code/subtype code/name selection

Hex 0E-Context entries collating at and above the specified Type code/subtype code/name selection

- Length of name to be used for search argument — Bin(2)
- Type code — Char(1)
- Subtype code — Char(1)
- Name — Char(30)

If the information requirements parameter is binary 0, the context attributes are materialized with no context entries. In this case, the selection criteria field is meaningless.

If the validation attribute indicates no validation is to be performed, no object validation occurs and a significant performance improvement results.

Selection criteria value Hex 00, when the number of bytes provided in the receiver does not allow for materialization of at least one context entry, requests that as much of the context attributes as will fit be materialized into the receiver and that an estimate of the the byte size correlating to the full list of context entries currently in the context be set into the number of bytes available for materialization field of the receiver. This capability of requesting an estimate of the size of a full materialization of the context provides a low overhead way of getting a close approximation of the amount of space that will be needed for an actual materialize of all context entries.

Selection criteria value Hex 00, when the number of bytes provided in the receiver allow for materialization of at least one context entry, and values X'01' through X'06' request that all context entries matching the associated type code/subtype code/name criteria be materialized into the receiver. The number of bytes available for materialization field is set with the byte size correlating to the full list of context entries that matched the selection criteria whether or not the receiver provided enough room for the full list to be materialized.

Selection criteria value Hex 0E requests that as many context entries as will fit which collate at or higher (are equal to or greater) than the specified type code/subtype code/name criteria be materialized into the receiver. The number of bytes available for materialization field is set with the byte size correlating to the list of context entries that were actually materialized into the receiver rather than the full list that may have been available in the context.

When no validation occurs, some of the following pointers may be erroneous:

- Pointers to destroyed objects
- Pointers to objects that are no longer in the context
- Multiple pointers to the same object

The first 4 bytes of the materialization output identify the total number of bytes available for use by the instruction. This value is supplied as input to the instruction and is not modified by the instruction. A value of less than 8 causes the materialization length exception to be signaled. The instruction materializes as many bytes and pointers as can be contained in the receiver. If the byte area identified by the receiver is greater than that required to contain the information requested for materialization, the excess bytes are unchanged. No exceptions

are signaled in the event that the receiver contains insufficient area for the materialization, other than the materialization length exception signaled above.

The format of the materialization is as follows:

- Materialization size specification — Char(8)
  - Number of bytes provided for materialization — Bin(4)
  - Number of bytes available for materialization — Bin(4)
- Context identification — Char(32)
  - Object type — Char(1)
  - Object subtype — Char(1)
  - Object name — Char(30)
- Context options — Char(4)
  - Existence attributes — Bit 0

    0 = Temporary
    1 = Permanent
  - Space attribute — Bit 1

    0 = Fixed-length
    1 = Variable-length
  - Reserved (binary 0) — Bit 2
  - Access group — Bit 3

    0 = Not a member of access group
    1 = Member of access group
  - Reserved (binary 0) — Bits 4-31
- Recovery options — Char(4)
  - Automatic damaged context rebuild option — Bit 0

    0 = Do not rebuild at IMPL
    1 = Rebuild at IMPL
- Size of space — Bin(4)
- Initial value of space — Char(1)
- Performance class — Char(4)
  - Space alignment — Bit 0

    0 = The space associated with the object is allocated to allow proper alignment of pointers at 16-byte alignments within the space. If no space is specified for the object, this value must be specified for the performance class.
    1 = The space associated with the object is allocated to allow proper alignment of pointers at 16-byte alignments within the space as well as to allow proper alignment of input/output buffers at 512-byte alignments within the the space.
  - Reserved (binary 0) — Bits 1-4
  - Main storage pool selection — Bit 5

0 = Process default main storage pool is used for object.
1 = Machine default main storage pool is used for object.

- Reserved (binary 0) Bit 6
- Block transfer on implicit access state modification Bit 7

  0 = Transfer the minimum storage transfer size for this object. This value is 1 storage unit.
  1 = Transfer the machine default storage transfer size. This value is 8 storage units.

- Reserved (binary 0) Bits 8-31

- Reserved (binary 0) Char(7)
- Reserved (binary 0) Char(16)
- Access group System pointer
- Context entry (repeated for each selected entry) Char(16-48)
  - Object identification (if requested) Char(32)

    Type code Char(1)
    Subtype code Char(1)
    Name Char(30)

  - Object pointer (if requested) System pointer

The context entry object identification information, if requested by the materialization options parameter, is present for each entry in the context that satisfies the search criteria. If both system pointers and symbolic identification are requested by the materialization options operand, the system pointer immediately follows the object identification for each entry.

The order of the materialization of a context is by object type code, object subtype code, and object name, all in ascending sequence.

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for this instruction.

Authorization Required

- Retrieve
  - Operand 2

Lock Enforcement

- Materialization
  - Operand 2

Exceptions

Exception		Operands 1	2	3	Other
06	Addressing				
	01 Space addressing violation	X	X	X	
	02 Boundary alignment	X	X	X	
	03 Range	X	X	X	

Exception		Operands 1	2	3	Other
	06 Optimized addressability invalid	X	X	X	
08	Argument/parameter				
	01 Parameter reference violation	X	X	X	
0A	Authorization				
	01 Unauthorized for operation		X		
10	Damage encountered				
	02 Machine context damage state				X
	04 System object damage state	X	X	X	X
	05 authority verification terminated due to damaged object				X
	44 Partial system object damage	X	X	X	X
1A	Lock state				
	01 Invalid lock state		X		
1C	Machine-dependent exception				
	03 Machine storage limit exceeded		X		X
20	Machine support				
	02 Machine check				X
	03 Function check				X
22	Object access				
	01 Object not found	X	X	X	
	02 Object destroyed	X	X	X	
	03 Object suspended	X	X	X	
	07 authority verification terminated due to destroyed object				X
24	Pointer specification				
	01 Pointer does not exist	X	X	X	
	02 Pointer type invalid	X	X	X	
	03 Pointer addressing invalid object		X		
2A	Program creation				
	06 Invalid operand type	X	X	X	
	07 Invalid operand attribute	X		X	
	08 Invalid operand value range	X		X	
	0A Invalid operand length	X		X	
	0C Invalid operand odt reference	X	X	X	
	0D Reserved bits are not zero	X	X	X	X
2E	Resource control limit				
	01 user profile storage limit exceeded				X

Exception		Operands 1	2	3	Other
32	Scalar specification				
	02 Scalar attributes invalid			X	
	03 Scalar value invalid			X	
36	Space management				
	01 space extension/truncation				X
38	Template specification				
	03 Materialization length exception	X			

Chapter 13. Authorization Management Instructions

This chapter describes the instructions used for authorization management. These instructions are in alphabetic order. For an alphabetic summary of all the instructions, see Appendix A, "Instruction Summary."

13.1 Materialize Authority (MATAU)

Op Code (Hex)	Operand 1	Operand 2	Operand 3
0153	Receiver	System object	User profile or Source Template

Operand 1: Space pointer.

Operand 2: System pointer.

Operand 3: System pointer or Space pointer Data Object or null.

Description: This instruction materializes the specific types of authority for a system object available to the specified user profile. The private authorization that the user profile specified by operand 3 has to the permanent system object specified by operand 2, and the object's public authorization is materialized in operand 1. If operand 3 is null, then only the object's public authorization is materialized, and the private authorization field in the materialization is set to binary 0.

Except for certain special cases, the authority to be materialized is determined by first checking for direct authority to the object itself, then checking for indirect authority to the object through authority to an authorization list containing the object. The first source of authority found is materialized and the source is indicated in the materialization.

The special case of the operand 3 user profile having all object special authority overrides any explicit private authorities that the user profile might hold to the object or its containing authorization list and results in a materialization showing that the profile holds all private authorities directly to the object.

The special case of the operand 2 object being in an authorization list which has the override specific object authority attribute in effect results in the authorization or lack of authorization held to the authorizaiton list completely overriding the explicit private authorities that the user profile might hold to the object. This case results in a materialization showing that the profile has just the private authorities it holds or doesn't hold to the authorization list. That is, if the user profile has private authority to the object, but doesn't have private authority to the authorization list, the materialization will show that the user does not have any private authority to the object. Similarly, if the user profile has both private authority to the object and to the authorization list, the materialization will show that the user has only the private authority through the authorization list. If operand 3 is null, then only the object's public authorization is materialized, and the private authorization field in the materialization is set to binary zeros.

Operand 3 may be specified as a a system pointer which directly addresses the user profile to be checked as a source of authority or as a space pointer to a source template which identifies the source user profile. Specifying a template allows for additional controls over how the materialize operation is to be performed. The format of the source template is the following:

- Source flags — Char(2)
  - Ignore all object special authority — Bit 0

0 = No
1 = Yes

Reserved (binary zero)	Bit1-15
• Reserved (binary zero)	Char(14)
• User profile pointer	System Pointer

The ignore all object special authority source flag specifies whether or not that special authority is to be ignored during the materialize operation. When yes is specified, just the explicitly held private authority that the specified user profile holds either directly to the object or indirectly to an authorization list containing the object will be materialized. When no is specified, the authority provided by all object special authority, if held by the source user profile, is included and results in a materialization showing that the profile holds all private authorities directly to the object. No is the default for this flag value when the source template is not specified.

The user profile pointer field specifies the address of the user profile to be checked as a source of authority.

The first 4 bytes of the materialization identify the total number of bytes that may be used by the instruction. This value is supplied as input to the instruction and is not modified by the instruction. A value of less than 8 causes the materialization length exception.

The second 4 bytes of the materialization identify the total number of bytes available to be materialized (16 for this instruction). The instruction materializes as many bytes as can be contained in the area specified as the receiver. If the byte area identified by the receiver is greater than that required to contain the information requested, then the excess bytes are unchanged. No exceptions (other than the materialization length exception) are signaled in the event that the receiver contains insufficient area for the materialization.

The format of the materialization is as follows:

• Materialization size specification	Char(8)
— Number of bytes provided for materialization	Bin(4)
— Number of bytes available for materialization (contains a value of 16 for this instruction)	Bin(4)
• Private authorization (1 = authorized)	Char(2)
— Object control	Bit 0
— Object management	Bit 1
— Authorized pointer	Bit 2
— Space authority	Bit 3
— Retrieve	Bit 4
— Insert	Bit 5
— Delete	Bit 6
— Update	Bit 7
— Ownership (1 = yes)	Bit 8

  - Excluded — Bit 9
  - Authority List Management — Bit 10
  - Reserved (binary zero) — Bit 11-15
- Public authorization (1 = authorized) — Char(2)
  - Object control — Bit 0
  - Object management — Bit 1
  - Authorized pointer — Bit 2
  - Space authority — Bit 3
  - Retrieve — Bit 4
  - Insert — Bit 5
  - Delete — Bit 6
  - Update — Bit 7
  - Reserved (binary zero) — Bit 8
  - Excluded — Bit 9
  - Authority List Management — Bit 10
  - Reserved (binary zero) — Bit 11-15
- Private authorization source — Bin(2)

  0 = authority to object
  1 = authority to authorization list
- Public authorization source — Bin(2)

  0 = authority from object
  1 = authority from authorization list

Any of the four authorizations-retrieve, insert, delete, or update-constitute operational authority.

If this instruction references a temporary object, all public authority states are materialized. Private authority states are not materialized.

Authorization Required

- Operational
  - Operand 3
- Retrieve
  - Contexts referenced for address resolution

Lock Enforcement

- Materialize
  - Operand 2
  - Operand 3
  - Contexts referenced for address resolution

Exceptions

Exception		Operands 1	2	3	Other
06	Addressing				
	01 space addressing violation	X	X	X	
	02 boundary alignment	X	X	X	
	03 range	X	X	X	
	06 optimized addressability invalid	X	X	X	
08	Argument/parameter				
	01 parameter reference violation	X	X	X	
0A	Authorization				
	01 unauthorized for operation		X	X	
10	Damage encountered				
	02 machine context damage state				X
	04 system object damage state	X	X	X	X
	05 authority verification terminated due to damaged object				X
	44 partial system object damage	X	X	X	X
1A	Lock state				
	01 invalid lock state		X	X	
1C	Machine-dependent exception				
	03 machine storage limit exceeded				X
20	Machine support				
	02 machine check				X
	03 function check				X
22	Object access				
	01 object not found	X	X	X	
	02 object destroyed	X	X	X	
	03 object suspended	X	X	X	
	07 authority verification terminated due to destroyed object				X
24	Pointer specification				
	01 pointer does not exist	X	X	X	
	02 pointer type invalid	X	X	X	
	03 pointer addressing invalid object		X	X	
2A	Program creation				
	06 invalid operand type	X	X	X	
	07 invalid operand attribute	X	X	X	
	08 invalid operand value range	X	X	X	
	0A invalid operand length	X			

Exception		Operands 1	2	3	Other
	0C invalid operand odt reference	X	X	X	
	0D reserved bits are not zero	X	X	X	X
2E	Resource control limit				
	01 user profile storage limit exceeded				X
36	Space management				
	01 space extension/truncation				X
38	Template specification				
	03 materialization length exception	X			

13.2 Materialize Authority List (MATAL)

Op Code (Hex)	Operand 1	Operand 2	Operand 3
01B3	Receiver	Authori-zation List	Materializations Options

Operand 1: Space pointer.

Operand 2: System pointer.

Operand 3: Space pointer.

Description: Based on the contents of the materialization options specified by operand 3, the symbolic identification and/or system pointers to all or a selected set of the objects contained in the authorization list specified by operand 2 are materialized into the receiver specified by operand 1.

The materialization options operand has the following format:

- Materialization control — Char(2)

  Information Requirements — Char(1)

Value (Hex)	Meaning
12	Materialize count of entries matching the criteria.
22	Materialize identification of entries matching the criteria with short description.
32	Materialize identification of entries matching the criteria with long description.

  Selection Criteria — Char(1)

00	All authorization list entries
01	Type code selection
02	Type code/subtype code selection

- Reserved (binary zero) — Bin(2)
- Type code — Char(1)
- Subtype code — Char(1)
- Reserved (binary zero) — Char(30)

The information requirements field specifies the type of materialization, just a count of entries, short descriptions, or long descriptions, which is being requested.

The selection criteria field specifies the criteria to be used in selecting the authorization list entries for which information is to be presented. The type code and subtype code fields contain the selection criteria when a selective materialization is specified.

When type code or type/subtype codes are part of the selection criteria, only entries that have the specified codes are considered.

The format of the materialization is as follows:

• Materialization size specification	Char(8)
— Number of bytes provided for materialization	Bin(4)
— Number of bytes available for materialization	Bin(4)
• Authorization List Identification	Char(32)
— Object type	Char(1)
— Object subtype	Char(1)
— Object name	Char(30)
• Authorization List Options	Char(4)
— Existence attributes	Bit 0
1 = Permanent (always permanent)	
— Space Attribute	Bit 1
0 = Fixed length 1 = Variable length	
— Reserved	Bit 2-31
• Reserved	Char(4)
• Size of space	Bin(4)
• Initial value of space	Char(1)
• Performance class	Char(4)
• Reserved	Char(7)
• Context	System Pointer
• Reserved	Char(16)
• Authorization list attributes	Char(4)
— Override specific object authority	Bit 0
0 = No 1 = Yes	
— Reserved (binary zero)	Bit 1-31
• Reserved (binary zero)	Char(28)
• Entries header	Char(16)
— Number of entries available	Bin(4)
— Reserved	Char(12)

If no description is requested in the materialization options parameter, the above constitutes the information available for materialization. If a description (short or long) is requested by the materialization options operand, a description entry is present (assuming a sufficient size receiver) for each object materialized into the receiver. Either of the following entry formats may be selected.

• Short description entry	Char(32)
— Type code	Char(1)
— Subtype code	Char(1)
— Reserved	Char(14)
— System object	System Pointer
• Long description entry	Char(128)
— Type code	Char(1)
— Subtype code	Char(1)
— Object name	Char(30)
— Reserved	Char(16)
— System object	System Pointer
— Object owning user profile	System Pointer
• Context	Char(48)
— Type code	Char(1)
— Subtype code	Char(1)
— Context name	Char(30)
— Context pointer	System Pointer

The first four bytes of the materialization output identify the total quantity of bytes available for use by the instruction. This value is supplied as input to the instruction and is not modified by the instruction. A value of less than 8 causes the materialization length exception to be signaled.

The instruction materializes as many bytes and pointers as can be contained in the receiver. If the byte area identified by the receiver is greater than that required to contain the information requested for materialization, the excess bytes are unchanged. No exceptions are signaled in the event that the receiver contains insufficient area for the materialization, other than the materialization length exception signaled above.

Refer to the Create Authorization List instruction for a discussion of the creation attributes materialized in the above template.

The number of entries available field specifies the number of authorization list entries which satisfied the selection criteria and were therefore materialized. A value of zero indicates no entries were available.

The object identification information, if requested by the materialization options parameter, is present for each entry in the authorization list that satisfies the search criteria.

The object pointer information, if requested by the materialization options parameter, is present for each entry in the authorization list that satisfies the search criteria.

If the object addressed by the system pointer is not addressed by a context, the context type entry is set to Hex 00 or if the object is addressed by the machine context, the context type entry is set to Hes 81. Additionally, in either of these cases, the context pointer is set to the system default "pointer does not exist" value.

Authorization Required

- Retrieve

  Operand 2

Lock Enforcement

- Materialization

  Operand 2

Exceptions

Exception		Operands 1	2	3	Other
06 addressing					
	01 space addressing violation	X	X	X	
	02 boundary alignment violation	X	X	X	
	03 range	X	X	X	
	06 optimized addressability invalid	X	X	X	
08	Argument/parameter				
	01 parameter reference violation	X	X	X	
0A	Authorization				
	01 unauthorized for operation		X		
10	Damage encountered				
	04 system object damage		X		X
	05 authority verification terminated due to damaged object				X
	44 partial system object damage				X
1A	Lock state				
	01 invalid lock state		X		
1C	Machine dependent exception				
	03 machine storage limit exceeded		X		X
20	Machine support				
	02 machine check				X
	03 function check				X
22	Object access				

Exception		Operands 1	2	3	Other
	01 object not found	X	X	X	
	02 object destroyed	X	X	X	
	03 object suspended	X	X	X	
	07 authority verification terminated due to destroyed object				X
24	Pointer specification				
	01 pointer does not exist	X	X	X	
	02 pointer type invalid	X	X	X	
	03 pointer addressing invalid object		X		
2A	Program creation				
	06 invalid operand type	X	X	X	
	07 invalid operand attribute	X		X	
	08 invalid operand value range	X		X	
	0A invalid operand length	X		X	
	0C invalid operand odt reference	X	X	X	
	0D reserved bits are not zero	X	X	X	X
2E	Resource control limit				
	01 user profile storage limit exceeded				X
32	Scalar specification				
	01 scalar type invalid	X	X	X	
	02 scalar attributes invalid			X	
	03 scalar value invalid			X	
36	Space management				
	01 space extension/truncation				X
3E	Template specification				
	03 materialization length	X			

13.3 Materialize Authorized Objects (MATAUOBJ)

Op Code (Hex)	Operand 1	Operand 2	Operand 3
013B	Receiver	User profile	Materializtion options

Operand 1: Space pointer.

Operand 2: System pointer.

Operand 3: Character(1) scalar (fixed-length).

Description: This instruction materializes the identification and the system pointers to system objects that are privately owned or that are owned by a specified user profile. The materialization options (operand 3) for the user profile (operand 2) are returned in the receiver (operand 1). The materialization options for operand 3 for the short template header have the following format:

Value (Hex)	Meaning
11	Materialize count of owned objects with no description.
12	Materialize count of authorized objects with no description (excludes owned objects).
13	Materialize count of all authorized and owned objects with no description.
21	Materialize identification of owned objects with short description.
22	Materialize identification of authorized objects with short description (excludes owned objects.)
23	Materialize identification of all authorized and owned objects with short description.
31	Materialize identification of owned objects with long description.
32	Materialize identification of authorized objects with long description (excludes owned objects).
33	Materialize identification of all authorized and owned objects with long description.

The long template header materialization options hex 51 through hex 63 are the same as the short template materialization options hex 11 through 23.

The long template header materialization options hex 71 through hex 73 are the same as the short template materialization options hex 31 through hex 33 except that context extension is materialized for each object as well.

The first 4 bytes of the materialization identify the total number of bytes that may be used by the instruction. This value is supplied as input to the instruction and is not modified by the instruction. A value of less than 8 causes the materialization length exception to be signaled.

The second 4 bytes of the materialization identify the total number of bytes available to be materialized. The instruction materializes as many bytes as can be contained in the area specified as the receiver. If the byte area identified by the

receiver is greater than that required to contain the information requested, then the excess bytes are unchanged. No exceptions (other than the materialization length exception) are signaled in the event that the receiver contains insufficient area for the materialization.

The order of materialization is owned objects (if requested by the materialization options operand) followed by objects privately authorized to the user profile (if requested by the materialization options operand). No authorizations are stored in the system pointers that are returned.

The template identified by operand 1 must be 16-byte aligned in the space. For options hex 11 through hex 33, the short template header is materialized. It has the following format:

- Materialization size specification — Char(8)
  - Number of bytes provided for materialization — Bin(4)
  - Number of bytes available for materialization — Bin(4)
- Number of objects owned by user profile — Bin(2)
- Number of objects privately authorized to user profile — Bin(2)
- Reserved (binary 0) — Char(4)

For options hex 51 through 73, the long template header is materialized. It has the following format:

- Materialization size specification — Char(8)
  - Number of bytes provided for materialization — Bin(4)
  - Number of bytes available for materialization — Bin(4)
- Number of objects owned by user profile — Bin(4)
- Number of objects privately authorized to user profile — Bin(4)

If no description is requested in the materialization options parameter, the above constitutes the information available for materialization. If a description (short or long) is requested by the materialization options parameter, a description entry is present (assuming there is a sufficient sized receiver) for each object materialized into the receiver. Either of the following entries may be selected.

- Short description entry — Char(32)
  - Type code — Char(1)
  - Subtype code — Char(1)
  - Private authorization (1 = authorized) — Char(2)

Object control	Bit 0
Object management	Bit 1
Authorized pointer	Bit 2
Space authority	Bit 3
Retreive	Bit 4
Insert	Bit 5
Delete	Bit 6
Update	Bit 7

Ownership (1 = yes)	Bit 8
Excluded	Bit 9
Authority List Management	Bit 10
Reserved (binary zero)	Bit 11-15

- — Reserved (binary 0) — Char(12)
- — System object — System pointer

• Long description entry — Char(64)

- — Type code — Char(1)
- — Subtype code — Char(1)
- — Object name — Char(30)
- — Private authorization (1 = authorized) — Char(2)

Object control	Bit 0
Object management	Bit 1
Authorized pointer	Bit 2
Space authority	Bit 3
Retreive	Bit 4
Insert	Bit 5
Delete	Bit 6
Update	Bit 7
Ownership (1 = yes)	Bit 8
Excluded	Bit 9
Authority List Management	Bit 10
Reserved (binary zero)	Bit 11-15

- — Public authorization — Char(2)

Object control	Bit 0
Object management	Bit 1
Authorized pointer	Bit 2
Space authority	Bit 3
Retreive	Bit 4
Insert	Bit 5
Delete	Bit 6
Update	Bit 7
Reserved (binary zero)	Bit 8
Excluded	Bit 9
Authority List Management	Bit 10
Reserved (binary zero)	Bit 11-15

- — Reserved (binary 0) — Char(12)
- — System object — System pointer

• Context extension (options hex 71-73) — Char(48)

- — Type code — Char(1)
- — Subtype code — Char(1)
- — Context name — Char(30)
- — Context pointer — System pointer

The context extension portion of the long description entry is optional. It is only provided as an extension to the base form of the long description entry when options hex 71 through hex 73 are requested. For these options, if the object

addressed by the system pointer is not addressed by a context, the context type entry is set to hex 00 or if the object is addressed by the machine context, the context type entry is set to hex 81. Additionally, in either of these cases, the context pointer is set to the system default pointer does not exist value.

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for this instruction.

Authorization Required

- Operational
  - Operand 2
- Retrieve
  - Contexts referenced for address resolution
  - Operand 2 if materializing owned objects

Lock Enforcement

- Materialize
  - Contexts referenced for address resolution
  - Operand 2 if materializing owned objects

Exceptions

Exception		Operands 1	2	3	Other
06	Addressing				
	01 space addressing violation	X	X	X	
	02 boundary alignment	X	X	X	
	03 range	X	X	X	
08	Argument/parameter				
	01 parameter reference violation	X	X	X	
0A	Authorization				
	01 unauthorized for operation		X		
10	Damage encountered				
	02 machine context damage state				X
	04 system object damage state	X	X	X	X
	05 authority verifcation terminated due to damaged object				X
	44 partial system object damage	X	X	X	X
1A	Lock state				
	01 invalid lock state		X		
1C	Machine-dependent exception				
	03 machine storage limit exceeded				
20	Machine support				
	02 machine check				X

Exception		Operands 1	2	3	Other
	03 function check				X
22	Object access				
	01 object not found	X	X	X	
	02 object destroyed	X	X	X	
	03 object suspended	X		X	
	04 object not eligible for operation	X	X		
	07 authority verifcation terminated due to destroyed object				X
24	Pointer specification				
	01 pointer does not exist	X	X	X	
	02 pointer type invalid	X	X	X	
	03 pointer addressing invalid object		X		
2A	Program creation				
	06 invalid operand type	X	X	X	
	07 invalid operand attribute	X	X	X	
	08 invalid operand value range	X	X	X	
	0A invalid operand length	X			
	0C invalid operand odt reference	X	X	X	
	0D reserved bits are not zero	X	X	X	X
2E	Resource control limit				
	01 user profile storage limit exceeded				X
32	Scalar specification				
	03 scalar value invalid			X	
36	Space management				
	01 space extension/truncation				X
38	Template specification				
	03 materialization length exception	X			

13.4 Materialize Authorized Users (MATAUU)

Op Code (Hex)	Operand 1	Operand 2	Operand 3
0143	Receiver	System object	Materialization options

Operand 1: Space pointer.

Operand 2: System pointer.

Operand 3: Character(1) scalar (fixed-length).

Description: The instruction materializes the authorization states and the identification of the user profile(s). The materialization options (operand 3) for the system object (operand 2) are returned in the receiver (operand 1). The materialization options for operand 3 have the following format:

Value (Hex)	Meaning
11	Materialize public authority with no description.
12	Materialize public authority and number of privately authorized profiles with no description.
21	Materialize identificataon of owning profile with short description.
22	Materialize identification of privately authorized profiles with short description.
23	Materialize identification of owning and privately authorized profiles with short description.
31	Materialize identification of owning profile with long description.
32	Materialize identification of privately authorized profiles with long description.
33	Materialize identification of owning and privately authorized profiles with long description.

The first 4 bytes of the materialization identify the total number of bytes that may be used by the instruction. This value is supplied as input to the instruction and is not modified by the instruction. A value of less than 8 causes the materialization length exception.

The second 4 bytes of the materialization identify the total number of bytes available to be materialized. The instruction materializes as many bytes as can be contained in the area specified as the receiver. If the byte area identified by the receiver is greater than that required to contain the information requested, then the excess bytes are unchanged. No exceptions (other than the materialization length exception) are signaled in the event that the receiver contains insufficient area for the materialization.

The order of materialization is an entry for the owning user profile (if requested by the materialization options operand) followed by a list (0 to n entries) of entries for user profiles having private authorization to the object (if requested by the materialization options operand). The authorization field within the system pointers will not be set.

The template identified by operand 1 must be 16-byte aligned in the space and has the following format:

- Materialization size specification — Char(8)
  - Number of bytes provided for materialization — Bin(4)
  - Number of bytes available for materialization — Bin(4)
- Public authorization (1 = authorized) — Char(2)
  - Object control — Bit 0
  - Object management — Bit 1
  - Authorized pointer — Bit 2
  - Space authority — Bit 3
  - Retrieve — Bit 4
  - Insert — Bit 5
  - Delete — Bit 6
  - Update — Bit 7
  - Reserved (binary zero) — Bit 8
  - Excluded — Bit 9
  - Authority List Management — Bit 10
  - Reserved (binary zero) — Bits 11-15
- Number of privately authorized user profiles — Bin(2)
- Reserved (binary 0) — Char(4)

If no description is requested by the materialization options operand, the template identified by operand 1 constitutes the information available for materialization. If a description (short or long) is requested by the materialization options operand, a description entry is present (assuming there is a sufficient sized receiver) for each user profile materialized or available to be materialized into the receiver. Either of the following entry types may be selected.

- Short description entry — Char(32)
  - User profile type code — Char(1)
  - User profile subtype code — Char(1)
  - Private authorization (1 = authorized) — Char(2)

    Object control — Bit 0
    Object management — Bit 1
    Authorized pointer — Bit 2
    Space authority — Bit 3
    Retreive — Bit 4
    Insert — Bit 5
    Delete — Bit 6
    Update — Bit 7
    Ownership (1 = yes) — Bit 8
    Excluded — Bit 9
    Authority List Management — Bit 10

Reserved (binary zero)	Bits 11-15
– Reserved (binary 0)	Char(12)
– User profile	System pointer
• Long description entry	Char(64)
– User profile type code	Char(1)
– User profile subtype code	Char(1)
– User profile name	Char(30)
– Private authorization (1 = authorized)	Char(2)
Object control	Bit 0
Object management	Bit 1
Authorized pointer	Bit 2
Space authority	Bit 3
Retreive	Bit 4
Insert	Bit 5
Delete	Bit 6
Update	Bit 7
Ownership (1 = yes)	Bit 8
Excluded	Bit 9
Authority List Management	Bit 10
Reserved (binary zero)	Bits 11-15
– Reserved (binary 0)	Char(14)
– User profile	System pointer

If this instruction references a temporary object, all public authority states are materialized. The privately authorized user and owner profile(s) description is not materialized (binary 0).

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for this instruction.

Authorization Required

- Retrieve
  - Contexts referenced for address resolution
- Object management or ownership
  - Operand 2 object (when object is not an authorizaiton list)
- Authorization list management or ownership
  - Operand 2 object (when object is an authorizaiton list)

Lock Enforcement

- Materialize
  - Operand 2
  - Contexts referenced for address resolution

Exceptions

Exception		Operands 1	2	3	Other
06	Addressing				
	01 space addressing violation	X	X	X	
	02 boundary alignment	X	X	X	
	03 range	X	X	X	
	06 optimized addressability invalid	X	X	X	
08	Argument/parameter				
	01 parameter reference violation	X	X	X	
0A	Authorization				
	01 unauthorized for operation		X		
10	Damage encountered				
	04 system object damage state	X	X	X	X
	05 authority verification terminated due to damaged object				X
	44 partial system object damage	X	X	X	X
1A	Lock state				
	01 invalid lock state		X		
1C	Machine-dependent exception				
	03 machine storage limit exceeded				X
20	Machine support				
	02 machine check				X
	03 function check				X
22	Object access				
	01 object not found	X	X	X	
	02 object destroyed	X	X	X	
	03 object suspended	X	X	X	
	07 authority verification terminated due to destroyed object				X
24	Pointer specification				
	01 pointer does not exist	X	X	X	
	02 pointer type invalid	X	X	X	
2A	Program creation				
	06 invalid operand type	X	X	X	
	07 invalid operand attribute	X	X	X	
	08 invalid operand value range	X	X	X	
	0A invalid operand length	X			
	0C invalid operand odt reference	X	X	X	
	0D reserved bits are not zero	X	X	X	X

Exception		Operands 1	2	3	Other
2E	Resource control limit				
	01 user profile storage limit exceeded				X
32	Scalar specification				
	03 scalar value invalid			X	
36	Space management				
	01 space extension/truncation				X
38	Template specification				
	03 materialization length exception	X			

13.5 Materialize User Profile (MATUP)

Op Code (Hex)	Operand 1	Operand 2
013E	Receiver	User profile

Operand 1: Space pointer.

Operand 2: System pointer.

Description: The attributes of the user profile specified by operand 2 are materialized into the receiver specified by operand 1.

The first 4 bytes of the materialization identify the total number of bytes that may be used by the instruction. This value is supplied as input to the instruction and is not modified by the instruction. A value of less than 8 causes the materialization length exception to be signaled.

The second 4 bytes of the materialization identify the total number of bytes available to be materialized. The instruction materializes as many bytes as can be contained in the area specified as the receiver. If the byte area identified by the receiver is greater than that required to contain the information requested, then the excess bytes are unchanged. No exceptions (other than the materialization length exception) are signaled in the event that the receiver contains insufficient area for the materialization.

The receiver identified by operand 1 must be 16-byte aligned in the space. The following is the format of the materialized information:

- Materialization size specification — Char(8)
  - Number of bytes provided for materialization — Bin(4)
  - Number of bytes available for materialization — Bin(4)
- Object identification — Char(32)
  - Object type — Char(1)
  - Object subtype — Char(1)
  - Object name — Char(30)
- Object creation options — Char(4)
  - Existence attribute — Bit 0

    1 = Permanent
  - Space attribute — Bit 1

    0 = Fixed-length
    1 = Variable-length
  - Reserved (binary 1) — Bit 2
  - Reserved (binary 0) — Bits 3-12
  - Initialize space — Bit 13
  - Reserved (binary 0) — Bits 14-31
- Reserved (binary 0) — Char(4)

- Size of space — Bin(4)
- Initial value of space — Char(1)
- Performance class — Char(4)
- Reserved (binary 0) — Char(7)
- Reserved (binary 0) — Char(16)
- Reserved (binary 0) — Char(16)
- Privileged instructions (1 = authorized) — Char(4)
  - Create logical unit description — Bit 0
  - Create network description — Bit 1
  - Create controller description — Bit 2
  - Create user profile — Bit 3
  - Modify user profile — Bit 4
  - Diagnose — Bit 5
  - Terminate machine processing — Bit 6
  - Initiate process — Bit 7
  - Modify resource management control — Bit 8
  - Create mode description — Bit 9
  - Create class of service description — Bit 10
  - Reserved (binary zero) — Bits 11-31
- Special authorizations (1 = authorized) — Char(4)
  - All object authority — Bit 0
  - Load (unrestricted) — Bit 1
  - Dump (unrestricted) — Bit 2
  - Suspend object (unrestricted) — Bit 3
  - Load (restricted) — Bit 4
  - Dump (restricted) — Bit 5
  - Suspend object (restricted) — Bit 6
  - Process control — Bit 7
  - Reserved (binary 0) — Bit 8
  - Service authority — Bit 9
  - Auditor authority — Bit 10
  - Spool Control — Bit 11
  - Reserved (binary 0) — Bits 12-23
  - Modify machine attributes — Bits 24-31
    - Group 2 — Bit 24
    - Group 3 — Bit 25
    - Group 4 — Bit 26

Group 5	Bit 27
Group 6	Bit 28
Group 7	Bit 29
Group 8	Bit 30
Group 9	Bit 31

Note: Group 1 requires no authorization.

- Storage authorization-The maximum amount of auxiliary storage (in units of 1024 bytes) that can be allocated for thestorage of objects owned by this user profile — Bin(4)
- Storage utilization- The current amount of auxiliary storage (in units of 1024 bytes) allocated for the storage ofobjects owned by this user profile — Bin(4)

The attributes that the instruction can materialize are described in the Create User Profile instruction.

Authorization Required

- Operational
  - Operand 2

Lock Enforcement

- Materialize
  - Operand 2

Exceptions

Exception		Operands 1	Operands 2	Other
06	Addressing			
	01 space addressing violation	X	X	
	02 boundary alignment	X	X	
	03 range	X	X	
	06 optimized addressability invalid	X	X	
08	Argument/parameter			
	01 parameter reference violation	X	X	
0A	Authorization			
	01 unauthorized for operation		X	
10	Damage encountered			
	02 machine context			X
	04 system object damage state	X	X	X
	05 authority verification terminated due to damaged object			X
	44 partial system object damage	X	X	X
1A	Lock state			

Exception		Operands 1	2	Other
	01 invalid lock state		X	
1C	Machine-dependent exception			X
	03 machine storage limit exceeded			X
20	Machine support			
	02 machine check			X
	03 function check			X
22	Object access			
	01 object not found	X	X	
	02 object destroyed	X	X	
	03 object suspended	X	X	
	07 authority verification terminated due to destroyed object			X
24	Pointer specification			
	01 pointer does not exist	X	X	
	02 pointer type invalid	X	X	
	03 pointer addressing invalid object		X	
2A	Program creation			
	06 invalid operand type	X	X	
	07 invalid operand attribute	X	X	
	08 invalid operand value range	X	X	
	0A invalid operand length	X		
	0C invalid operand odt reference	X	X	
	0D reserved bits are not zero	X	X	X
2E	Resource control limit			
	01 user profile storage limit exceeded			X
36	Space management			
	01 space extension/truncation			X
38	Template specification			
	03 materialization length exception	X		

13.6 Test Authority (TESTAU)

Op Code (Hex)	Operand 1	Operand 2	Operand 3
10F7	Available authority template receiver	System object or object template	Required authority template

Operand 1: Character(2) variable scalar or null (fixed-length).

Operand 2: System pointer or space pointer data object.

Operand 3: Character(2) scalar (fixed-length).

Optional Forms

Mnemonic	Op Code (Hex)	Form Type
TESTAUI	18F7	Indicator
TESTAUB	1CF7	Branch

Extender: Branch or indicator options

If the branch option is specified in the op code, the extender field must be present along with one or two branch targets. If the indicator option is specified in the op code, the extender field must be present along with one or two indicator operands. The branch or indicator operands immediately follow operand 3. See Chapter 1. "Introduction" for the encoding of the extender field and the allowed syntax of the branch and indicator operands.

Description: This instruction verifies that the object authorities and/or ownership rights specified by operand 3 are currently available to the process for the object specified by operand 2.

If operand 1 is not null, all of the authorities and/or ownership specified by operand 3 that are currently available to the process are returned in operand 1.

If an object template is not specified, operand 2 is a system pointer, the authority verification is performed relative to the invocation executing this instruction. If an object template is specified, operand 2 is a space pointer, the authority verification is performed relative to the invocation specified in the template. Specifying an invocation causes the invocations subsequent to it to be bypassed in the authority verification process. This has the influence of excluding the program adopted user profiles for any of these excluded invocations from acting as a source of authority to the authority verification process.

The required authorities and/or ownership are specified by the required authority template of operand 3. This template includes a test option that indicates whether all of the specified authorities are required or whether any one or more of the specified authorities is sufficient. This option can be used, for example, to test for operational authority by coding a template value of hex 0F01 in operand 3. Using the *any* option does not affect what is returned in operand 1. If

operand 1 is not null and the *any* option is specified, all of the authorities specified by operand 3 that are available to the process are returned in operand 1.

If the required authority is available, one of the following occurs:

- Branch form indicated
  - Conditional transfer of control to the instruction indicated by the appropriate branch target operand.
- Indicator form specified
  - The leftmost byte of each of the indicator operands is assigned the following values.

    Hex F1- If the result of the test matches the corresponding indicator option

    Hex F0- If the result of the test does not match the corresponding indicator option

If no branch options are specified, instruction execution proceeds to the next instruction. If operand 1 is null and neither the branch or indicator form is used, an invalid operand type exception is signaled.

The format for the available authority template (operand 1) is as follows: (1 = authorized)

• Authorization template	Char(2)
– Object control	Bit 0
– Object management	Bit 1
– Authorized pointer	Bit 2
– Space authority	Bit 3
– Retrieve	Bit 4
– Insert	Bit 5
– Delete	Bit 6
– Update	Bit 7
– Ownership (1 = yes)	Bit 8
– Excluded	Bit 9
– Authority List Management	Bit 10
– Reserved (binary zero)	Bits 11-15

If operand 2 is a system pointer, it identifies the object for which authority is to be tested. If operand 2 is a space pointer, it provides addressability to the object template. The format for the optional object template is as follows:

• Object template	Char(32)
– Relative invocation	Bin(2)
– Reserved (binary 0)	Char(14)
– System object	System pointer

The *relative invocation* field in the object template identifies an invocation relative to the current invocation at which the authority verification is to be per-

formed. The value of the relative invocation field must be less than or equal to zero. A value of zero identifies the current invocation, -1 identifies the prior invocation, -2, the invocation prior to that, and so on. A value larger than the number of invocations currently on the invocation stack or a positive value results in the signaling of the template value invalid exception. The program adopted and propagated user profiles for the identified invocation and older invocations will be included in the authority verification process. Program adopted user profiles for invocations newer than the identified invocation will not be included in the authority verification process. If the current invocation is specified, its program adopted user profile is included whether or not it is to be propagated.

The *system object* field specifies a system pointer which identifies the object for which authority is to be tested.

The format for the required authority template (operand 3) is as follows: (1 = authorized)

- Authorization template — Char(2)
  - Object control — Bit 0
  - Object management — Bit 1
  - Authorized pointer — Bit 2
  - Space authority — Bit 3
  - Retrieve — Bit 4
  - Insert — Bit 5
  - Delete — Bit 6
  - Update — Bit 7
  - Ownership (1 = yes) — Bit 8
  - Excluded — Bit 9
  - Authority List Management — Bit 10
  - Reserved (binary zero) — Bits 11-14
  - Test option — Bit 15

    0 = All of the above authorities must be present.
    1 = Any one or more of the above authorities must be present.

This instruction will tolerate a damaged object referenced by operand 2 when the reference is a resolved pointer. The instruction will not tolerate damaged contexts or programs when resolving pointers. Damaged user profiles encountered during the authority verification processing result in the signaling of the authority verification terminated due to damaged object exception.

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for this instruction.

Resultant Conditions: Authorized - the required authority is available. Unauthorized - the required authority is not available.

Authorization Required

- Retrieve
  - Contexts referenced for address resolution

Lock Enforcement

- Materialize
  - Contexts referenced for address resolution

Exceptions

Exception		Operands 1	2	3	Other
06	Addressing				
	01 space addressing violation	X	X	X	
	02 boundary alignment	X	X	X	
	03 range	X	X	X	
	06 optimized addressability invalid	X	X	X	
08	Argument/parameter				
	01 parameter reference violation	X	X	X	
0A	Authorization				
	01 unauthorized for operation		X		
10	Damage encountered				
	02 machine context damage state		X		
	04 system object damage state	X	X	X	X
	05 authority verification terminated due to damaged object				X
	44 partial system object damage	X	X	X	X
1A	Lock state				
	01 invalid lock state		X		
1C	Machine-dependent exception				
	03 machine storage limit exceeded				X
20	Machine support				
	02 machine check				X
	03 function check				X
22	Object access				
	01 object not found	X	X	X	
	02 object destroyed	X	X	X	
	07 authority verification terminated due to destroyed object				X
24	Pointer specification				
	01 pointer does not exist	X	X	X	
	02 pointer type invalid	X	X	X	

Exception		Operands 1	2	3	Other
2A	Program creation				
	05 invalid op code extender field				X
	06 invalid operand type	X	X	X	X
	07 invalid operand attribute	X		X	X
	09 invalid branch target operand				X
	0C invalid operand odt reference	X	X	X	X
	0D reserved bits are not zero	X	X	X	X
2C	Program execution				
	04 invalid branch target				X
2E	Resource control limit				
	01 user profile storage limit exceeded				X
32	Scalar specification				
	01 scalar type invalid	X	X	X	
	03 scalar value invalid			X	
36	Space management				
	01 space extension/truncation				X
38	Template specification				
	01 template value invalid		X		

13.7 Test Extended Authorities (TESTEAU)

Op Code (Hex)	Operand 1	Operand 2	Operand 3
10FB	Available authority template receiver	Required authority template	Relative invocation

Operand 1: Character(8) variable scalar or null (fixed-length).

Operand 2: Character(8) scalar (fixed-length).

Operand 3: Bin(2) scalar or null (fixed-length).

Optional Forms

Mnemonic	Op Code (Hex)	Form Type
TESTEAUI	18FB	Indicator
TESTEAUB	1CFB	Branch

Extender: Branch or indicator options

If the branch option is specified in the op code, the extender field must be present along with one or two branch targets. If the indicator option is specified in the op code, the extender field must be present along with one or two indicator operands. The branch or indicator operands immediately follow operand 3. See Chapter 1. "Introduction" for the encoding of the extender field and the allowed syntax of the branch and indicator operands.

Description: This instruction verifies that the privileged instructions and special authorities specified by operand 2 are currently available to the process.

If operand 1 is not null, all of the privileged instructions and special authorities specified by operand 2 that are currently available to the process are returned in operand 1.

Note: The term *authority verification* refers to the testing of the required privileged instruction and special authorities.

If operand 3 is null, the authority verification is performed relative to the invocation executing this instruction. If an operand 3 is specified, the authority verification is performed relative to the invocation specified. Specifying an invocation causes the invocations subsequent to it to be bypassed in the authority verification process. This has the influence of excluding the program adopted user profiles for any of these excluded invocations from acting as a source of authority to the authority verification process.

The required privileged instructions and special authorities are specified by the required authority template of operand 2.

If the required authority is available, one of the following occurs:

- Branch form indicated

  - Conditional transfer of control to the instruction indicated by the appropriate branch target operand.

- Indicator form specified
  - The leftmost byte of each of the indicator operands is assigned the following values.

    Hex F1- If the result of the test matches the corresponding indicator option

    Hex F0- If the result of the test does not match the corresponding indicator option

If no branch options are specified, instruction execution proceeds to the next instruction. If operand 1 is null and neither the branch or indicator form is used, an invalid operand type exception is signaled.

The format for the available authority template (operand 1) is as follows: (1 = authorized)

• Authority Template	Char(8)
– Privileged Instruction Template	Char(4)
- Create logical unit description	Bit 0
- Create network description	Bit 1
- Create controller desciption	Bit 2
- Create User Profile	Bit 3
- Modify User Profile	Bit 4
- Diagnose	Bit 5
- Terminate Machine Processing	Bit 6
- Initiate Process	Bit 7
- Modify resource management control	Bit 8
- Create mode description	Bit 9
- Create class of service description	Bit 10
- Reserved (binary zero)	Bits 11-31
– Special Authority template	Char(4)
- All Object	Bit 0
- Load (unrestricted)	Bit 1
- Dump (unrestricted)	Bit 2
- Suspend (unrestricted)	Bit 3
- Load (restricted)	Bit 4
- Dump (restricted)	Bit 5
- Suspend (restricted)	Bit 6
- Process control	Bit 7
- Reserved (binary zero)	Bit 8
- Service	Bit 9

- Auditor Authority — Bit 10
- Spool Control — Bit 11
- Reserved (binary zero) — Bit 12-23
- Modify machine attributes — Bit 24-31
  - Group 2 — Bit 24
  - Group 3 — Bit 25
  - Group 4 — Bit 26
  - Group 5 — Bit 27
  - Group 6 — Bit 28
  - Group 7 — Bit 29
  - Group 8 — Bit 30
  - Group 9 — Bit 31

The format for the required authority template (operand 2) is as follows: (1 = authorized)

- Required Authority — Char(8)
  - Privileged Instruction Template — Char(4)
    - Create logoical unit description — Bit 0
    - Create network description — Bit 1
    - Create controller desciption — Bit 2
    - Create User Profile — Bit 3
    - Modify User Profile — Bit 4
    - Diagnose — Bit 5
    - Terminate Machine Processing — Bit 6
    - Initiate Process — Bit 7
    - Modify resource management control — Bit 8
    - Create mode description — Bit 9
    - Create class of service description — Bit 10
    - Reserved (binary zero) — Bits 11-31
  - Special Authority template — Char(4)
    - All Object — Bit 0
    - Load (unrestricted) — Bit 1
    - Dump (unrestricted) — Bit 2
    - Suspend (unrestricted) — Bit 3
    - Load (restricted) — Bit 4
    - Dump (restricted) — Bit 5
    - Suspend (restricted) — Bit 6

- Process control — Bit 7
- Reserved (binary zero) — Bit 8
- Service — Bit 9
- Reserved (binary zero) — Bit 10-23
- Modify machine attributes — Bit 24-31
  - Group 2 — Bit 24
  - Group 3 — Bit 25
  - Group 4 — Bit 26
  - Group 5 — Bit 27
  - Group 6 — Bit 28
  - Group 7 — Bit 29
  - Group 8 — Bit 30
  - Group 9 — Bit 31

The *relative invocation* operand (operand 3) identifies an invocation relative to the current invocation at which the authority verification is to be performed. The value of the relative invocation field must be less than or equal to zero. A value of zero identifies the current invocation, -1 identifies the prior invocation, -2, the invocation prior to that, and so on. A value larger than the number of invocations currently on the invocation stack or a positive value results in the signaling of the template value invalid exception. The program adopted and propagated user profiles for the identified invocation and older invocations will be included in the authority verification process. Program adopted user profiles for invocations newer than the identified invocation will not be included in the authority verification process. If the current invocation is specified, its program adopted user profile is included whether or not it is to be propagated.

Damaged user profiles encountered during the authority verification processing result in the signaling of the authority verification terminated due to damaged object exception.

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for this instruction.

Resultant Conditions: Authorized - the required authority is available. Unauthorized - the required authority is not available.

Authorization Required

- Retrieve
  - — Contexts referenced for address resolution

Lock Enforcement

- Materialize
  - — Contexts referenced for address resolution

Exceptions

Exception		Operands 1	2	3	Other
06	Addressing				
	01 space addressing violation	X	X	X	
	02 boundary alignment	X	X	X	
	03 range	X	X	X	
	06 optimized addressability invalid	X	X	X	
08	Argument/parameter				
	01 parameter reference violation	X	X	X	
0A	Authorization				
	01 unauthorized for operation		X		
10	Damage encountered				
	02 machine context damage state		X		
	04 system object damage state	X	X	X	X
	05 authority verification terminated due to damaged object				X
	44 partial system object damage	X	X	X	X
1A	Lock state				
	01 invalid lock state		X		
1C	Machine-dependent exception				
	03 machine storage limit exceeded				X
20	Machine support				
	02 machine check				X
	03 function check				X
22	Object access				
	01 object not found	X	X	X	
	02 object destroyed	X	X	X	
	07 authority verification terminated due to destroyed object				X
24	Pointer specification				
	01 pointer does not exist	X	X	X	
	02 pointer type invalid	X	X	X	
2A	Program creation				
	05 invalid op code extender field				X
	06 invalid operand type	X	X	X	X
	07 invalid operand attribute	X		X	X
	09 invalid branch target operand				X
	0C invalid operand odt reference	X	X	X	X
	0D reserved bits are not zero	X	X	X	X

Exception		Operands 1	2	3	Other
2C	Program execution				
	04 invalid branch target				X
2E	Resource control limit				
	01 user profile storage limit exceeded				X
32	Scalar specification				
	01 scalar type invalid	X	X	X	
	03 scalar value invalid			X	
36	Space management				
	01 space extension/truncation				X
38	Template specification				
	01 template value invalid		X		

Chapter 14. Process Management Instructions

This chapter describes instructions used for process management. These instructions are in alphabetic order. For an alphabetic summary of all the instructions, See Appendix A, "Instruction Summary."

14.1 Materialize Process Attributes (MATPRATR)

Op Code (Hex)	Operand 1	Operand 2	Operand 3
0333	Receiver	Process control space	Materialization options

Operand 1: Space pointer.

Operand 2: System pointer or null.

Operand 3: Character scalar(1).

Description: The instruction causes either one specific attribute or all the attributes of the designated process to be materialized.

Operand 1 specifies a space that is to receive the materialized attribute values. The space pointer specified in operand 1 must address a 16-byte aligned area.

Operand 2 is a system pointer identifying the process control space associated with the process whose attributes are to be materialized. If operand 2 is null, the process issuing the instruction is the subject process. If the subject process's attributes are being materialized by another process, that process must be the original initiator of the subject process or the governing user profile(s) must have process control special authorization.

Operand 3 is a character scalar(1) specifying which process attribute is to be materialized. A value of hex 00 results in all the attributes of a process being materialized in the format described in the Initiate Process instruction for the process definition template. Other options allow materialization of specialized process attributes.

The materialization template has the following general format when a process scalar attribute is materialized:

- Materialization size specification — Char(8)
  - Number of bytes provided for materialization — Bin(4)
  - Number of bytes available for materialization — Bin(4)
- Process scalar attributes — Char(*)

The materialization template has the following general format when a process pointer attribute is materialized:

- Materialization size specification — Char(8)
  - Number of bytes provided for materialization — Bin(4)
  - Number of bytes available for materialization — Bin(4)
- Reserved (binary 0) — Char(8)
- Process pointer attribute — System pointer or Space pointer

Note: The values of the entry associated with an asterisk (*) are ignored by this instruction.

The following attributes require materialization targets of varying lengths. The attributes to be materialized and their operand 3 materialization option values follow.

- Process control attributes Char(4)

  Values hex 01 through hex 0B or hex 27 cause the 4-byte process control attributes value to be placed in the byte area identified by operand 1. The individual attributes and the corresponding values are as follows:

  - Process type Bit 0

    0 = Dependent process
    1 = Independent process

  - Instruction wait access state control Bit 1

    0 = Access state modification is not allowed
    1 = Access state modification is allowed if specified

  - Time slice end access state control Bit 2

    0 = Access state modification is not allowed
    1 = Access state modification is allowed if specified

  - Time slice end event option Bit 3

    0 = Time slice expired without entering instruction wait event is not signaled
    1 = Time slice expired without entering instruction wait event is signaled

  - Reserved (binary 0) Bit 4

  - Initiation phase program option Bit 5

    0 = No initiation phase program specified (do not enter initiation phase)
    1 = Initiation phase program specified (enter initiation phase)

  - Problem phase program option Bit 6

    0 = No problem phase program specified (do not enter problem phase)
    1 = Problem phase program specified (enter problem phase)

  - Termination phase program option Bit 7

    0 = No termination phase program specified (do not enter termination phase)
    1 = Termination phase program specified (enter termination phase)

  - Process default exception handler option Bit 8

    0 = No process default exception handler
    1 = Process default exception handler specified

  - Process NRL (name resolution list) option Bit 9

    0 = No process NRL specified
    1 = Process NRL specified

  - Process access group option Bit 10

    0 = No process access group specified
    1 = Process access group specified

  - Process adopted user profile list option Bit 11

0 = No process adopted user profile list specified
1 = Process adopted user profile list specified

- Reserved (binary 0) Bits 12-31

- Signal event control mask

  The materialization of the control mask is as follows:

  - Hex 0C = Signal event control mask Char(2)

- Number of event monitors

  The materialization of this attribute is as follows:

  - Hex 0D = Number of event monitors Bin(2)

The resource management attributes and data types are as follows:

- Hex 0E = Process priority Char(1)
- Hex 0F = Process storage pool ID Char(1)
- Hex 10 = Maximum temporary auxiliary storage allowed Bin(4)
- Hex 11 = Time slice interval Char(8)
- Hex 12 = Default time-out interval Char(8)
- Hex 13 = Maximum processor time allowed Char(8)
- Hex 14 = Process multiprogramming level class ID Char(1)
- Hex 28 = Process category Char(2)
- Hex 15 = Modification control indicators Char(8)

The modification control indicators are materialized when the operand 3 value is hex 15. Each indicator specifies the modification options allowed to a process upon itself by the initiating process. The possible values of each modification control indicator are as follows:

00 = Modification of the attribute is not allowed.

01 = Modification is allowed in the initiation or termination phases only.

10 = Modification is allowed in all phases (initiation, problem, and termination).

The bit assignments of the modification control indicators are as follows:

- Instruction wait access state control Bits 0-1
- Time slice end access state control Bits 2-3
- Time slice event option Bits 4-5
- Reserved (binary 0) Bits 6-7
- Problem phase program option Bits 8-9
- Termination phase program option Bits 10-11
- Process default exception handler option Bits 12-13
- Process NRL option Bits 14-15
- Signal event control mask Bits 16-17

- Process priority — Bits 18-19
- Process storage pool identification — Bits 20-21
- Maximum temporary auxiliary storage allowed — Bits 22-23
- Time slice interval — Bits 24-25
- Default time-out interval — Bits 26-27
- Maximum processor time allowed — Bits 28-29
- Process MPL class ID — Bits 30-31
- User profile pointer — Bits 32-33
- Process communication object pointer — Bits 34-35
- Process NRL pointer — Bits 36-37
- Termination phase program pointer — Bits 38-39
- Problem phase program pointer — Bits 40-41
- Process default exception handler — Bits 42-43
- Process adopted user profile list — Bits 44-45
- Process adopted user profile list option — Bits 46-47
- Process category — Bits 48-49
- Reserved (binary 0) — Bits 50-63

The process pointer attributes which may be materialized are the following:

- Hex 16 = Process user profile pointer

  The system pointer with addressability to the user profile is placed into the space addressed by operand 1. If the materialization option (hex 00) is specified in operand 3, a reserved character(9) field is included at this point. This user profile is the process user profile assigned by the Initiate Process or Modify Process Attribute instruction.

- Hex 17 = Process communication object (PCO) pointer

  The PCO pointer is placed in the space addressed by operand 1.

- Hex 18 = Process name resolution List

  The space pointer to the NRL is placed in the space addressed by operand 1.

- Hex 19 = Initiation phase program pointer

  The system pointer to the program is placed in the space addressed by operand 1.

- Hex 1A = Termination phase program pointer

  The system pointer to the program is placed in the space addressed by operand 1.

- Hex 1B = Problem phase program pointer

  The system pointer to the program is placed in the space addressed by operand 1.

- Hex 1C = PDEH (process default exception handler program)

The system pointer to the program is placed in the space addressed by operand 1.

- Hex 1D = Process automatic storage area

  The space pointer with addressability to the PASA is placed in the space addressed by operand 1.

- Hex 1E = Process static storage area

  The space pointer with addressability to the PSSA is placed in the space addressed by operand 1.

- Hex 1F = Process access group

  The system pointer with addressability to the PAG is placed in the space addressed by operand 1.

Process status indicators are materialized when the value of operand 3 is hex 20. The format and associated values of this attribute are as follows:

• Process states	Char(2)
— External existence state	Bits 0-2
000 = Suspended 010 = Suspended, in instruction wait 100 = Active, in ineligible wait 101 = Active, in current MPL 110 = Active, in instruction wait	
— Invocation exit active	Bit 3
— Reserved (binary 0)	Bits 4-7
— Internal processing phase	Bits 8-10
001 = Initiation phase 010 = Problem phase 100 = Termination phase	
— Reserved (binary 0)	Bits 11-15
• Process interrupt status (bit 1 denotes pending)	Char(2)
— Time slice end pending	Bit 0
— Transfer lock pending	Bit 1
— Asynchronous lock retry pending	Bit 2
— Suspend process pending	Bit 3
— Resume process pending	Bit 4
— Resource management attribute modify pending	Bit 5
— Process attribute modify pending	Bit 6
— Terminate machine processing pending	Bit 7
— Terminate process pending	Bit 8
— Wait time-out pending	Bit 9
— Event schedule pending	Bit 10
— Machine service pending	Bit 11

- Invocation exit active — Bit 12
- Reserved (binary 0) — Bits 13-15

- Process initial internal termination status — Char(3)
  - Initial internal termination reason — Bits 0-7

    Hex 80 = Return from first invocation in problem phase.
    Hex 40 = Return from first invocation in initiation phase, and no problem phase program specified.
    Hex 20 = Terminate Process instruction issued by this process to itself.
    Hex 10 = Exception was not handled by the process.
    Hex 00 = Process terminated externally.

  - Initial internal termination code — Bits 8-23

    The code is assigned in one of the following ways:

    1. If the termination is caused by a Return External instruction from the first invocation, then this code is binary 0's.
    2. The code is assigned by operand 2 of the Terminate Process instruction. This code is also given to subordinate processes involved in the termination.
    3. The code is assigned by the original exception code that caused process termination to commence. This code is also given to subordinate processes involved in the termination.

- Process initial external termination status — Char(3)
  - Initial external termination reason: — Bits 0-7

    Hex 80 = Terminate Process instruction issued explicitly to this process from another process.
    Hex 40 = A superordinate process has been terminated.
    Hex 00 = Process terminated internally.

  - Initial external termination code: — Bits 8-23

    This code is supplied by the termination code in operand 2 of the Terminate Process instruction.

- Process final termination status — Char(3)
  - Final termination reason: — Bits 0-7

    Hex 80 = Return instruction from first invocation.
    Hex 40 = Terminate Process instruction issued by the process being materialized.
    Hex 20 = Terminate Process instruction issued to the process being materialized by another process.
    Hex 10 = Exception not handled by this process.
    Hex 08 = Terminate Process instruction issued to superordinate of the process being materialized.
    Hex 04 = Superordinate process of the process being materialized completed termination phase.

  - Final termination code is assigned in one of the following ways: — Bits 8-23

    1. If the termination is caused by a Return External instruction from first invocation, then this code is binary 0's.

2. The termination code is assigned by the Terminate Process instruction.
3. The termination code is assigned by the original exception code that caused process termination.

   The process final termination status is presented as event-related data in the terminate process event. Usually the event is the only source of the process final termination status since the process will cease to exist before its attributes can be materialized.

Process resource usage attributes are materialized when the value of operand 3 is hex 21. The format and associated values of this attribute are as follows:

- Total temporary auxiliary storage used — Bin(4)
- Total processor time used — Char(8)
- Number of locks currently held by the process (including implicit locks) — Bin(2)

Subordinate processes identification attributes are materialized when the value of operand 3 is hex 22. The format and associated values of this attribute are as follows:

- Materialization size specification — Char(8)
  - Number of bytes provided for materialization — Bin(4)
  - Number of bytes available for materialization — Bin(4)
- Number of immediately subordinate processes — Bin(2)
- Reserved (binary 0) — Char(6)
- System pointer to the process control space for each subordinate process (repeated for each immediately subordinate process) — System pointer(s)

Process performance attributes are materialized when the value of operand 3 is hex 23. The format and associated values of this attribute are as follows:

- Materialization size specification — Char(8)
  - Number of bytes provided for materialization — Bin(4)
  - Number of bytes available for materialization — Bin(4)
- Number of synchronous page reads into main storage associated with data base — Bin(4)
- Number of synchronous page reads into main storage not associated with data base — Bin(4)
- Total number of synchronous page writes from main storage. This includes writes associated with and not associated with data base. — Bin(4)

• Number of transitions into ineligible wait state	Bin(2)
• Number of transitions into an instruction wait	Bin(2)
• Number of transitions into ineligible wait state from an instruction wait	Bin(2)
• Timestamp of materialization	Char(8)
• Number of asynchronous reads into main storage associated with data base	Bin(4)
• Number of asynchronous reads into main storage not associated with data base	Bin(4)
• Number of synchronous writes from main storage associated with data base	Bin(4)
• Number of synchronous writes from main storage not associated with data base	Bin(4)
• Number of asynchronous writes from main storage associated with data base	Bin(4)
• Number of asynchronous writes from main storage not associated with data base	Bin(4)
• Total number of writes from main storage of permanent objects	Bin(4)
• Total reads and writes performed for checksum updating due to writes of checksum protected objects	Bin(4)
• Number of page faults on process access group objects	Bin(4)
• Number of internal effective address overflow exceptions	Bin(4)
• Number of internal binary overflow exceptions	Bin(4)
• Number of internal decimal overflow exceptions	Bin(4)
• Number of internal floating point overflow exceptions	Bin(4)
• Number of times a page fault occurred on an address that was currently part of an auxiliary storage I/O operation	Bin(4)
• Number of times the process explicitly waited for outstanding asynchronous I/O operations to complete	Bin(4)

Each of the Bin(2) counters has a limit of 32 767. If this limit is exceeded, the count is set to 0, and no exception is signaled.

The process performance attributes are not supplied with materialization option hex 00.

The first 4 bytes of the materialization identify the total number of bytes that may be used by the instruction. This value is supplied as input to the instruction and is not modified by the instruction. A value of less than 8 causes the materialization length exception to be signaled.

The second 4 bytes of the materialization identify the total number of bytes available to be materialized. The instruction materializes as many bytes as can be contained in the area specified as the receiver. If the byte area identified by the receiver is greater than that required to contain the information requested, then the excess bytes a No exceptions (other than the materialization length exception described previously) are signaled in the event that the receiver contains insufficient area for the materialization.

Process execution status attributes are materialized when the value of operand 3 is hex 24. The format and associated values of this attribute are as follows:

- Process priority — Char(2)
  - Machine interface priority — Char(1)
  - Machine adjusted priority — Char(1)

    Normal value is hex 80. This value is dynamically modified by the machine.
- Pending interruptions — Char(2)
  - Time slice end — Bit 0
  - Transfer lock — Bit 1
  - Asynchronous lock retry — Bit 2
  - Suspend process — Bit 3
  - Resume process — Bit 4
  - Modify resource management attribute — Bit 5
  - Modify process attribute — Bit 6
  - Terminate machine processing — Bit 7
  - Terminate process — Bit 8
  - Wait time-out — Bit 9
  - Event — Bit 10
  - Machine service — Bit 11
  - Reserved (binary 0) — Bits 12-15
- Execution status — Char(2)
  - Suspended — Bit 0
  - Instruction wait — Bit 1
  - In MPL — Bit 2
  - Ineligible wait — Bit 3
  - Reserved (binary 0) — Bits 4-15
- Wait status — Char(2)
- Wait on event — Bit 0

- Dequeue — Bit 1
- Lock — Bit 2
- Wait on time — Bit 3
- Reserved (binary 0) — Bits 4-15
- Process class identification — Char(2)
- Storage pool class — Char(1)
- MPL class — Char(1)
- Processor time used — Char(8)
- Performance attributes — Char(70)
  - Number of synchronous reads into main storage associated with data base — Bin(4)
  - Number of synchronous reads into main storage not associated with data base — Bin(4)
  - Total number of synchronous page writes from main storage. This includes writes associated with and not associated with data base. — Bin(4)
  - Transitions to ineligible wait — Bin(2)
  - Transitions to instruction wait — Bin(2)
  - Transitions to ineligible from instruction wait — Bin(2)
  - Number of asynchronous reads into main storage associated with data base — Bin(4)
  - Number of asynchronous reads into main storage not associated with data base — Bin(4)
  - Number of synchronous writes from main storage associated with data base — Bin(4)
  - Number of synchronous writes from main storage not associated with data base — Bin(4)
  - Number of asynchronous writes from main storage associated with data base — Bin(4)
  - Number of asynchronous writes from main storage not associated with data base — Bin(4)
  - Total number of writes from main storage of permanent objects — Bin(4)
  - Total reads and writes performed for checksum updating due to writes of checksum protected objects — Bin(4)
  - Number of page faults on process access group objects — Bin(4)
  - Number of internal effective address overflow exceptions — Bin(4)
  - Number of internal binary overflow exceptions — Bin(4)

- Number of internal decimal overflow exceptions — Bin(4)
- Number of internal floating point overflow exceptions — Bin(4)
- Number of times a page fault occurred on an address that was currently part of an auxiliary storage I/O operation — Bin(4)
- Number of times the process explicitly waited for outstanding asynchronous I/O operations to complete — Bin(4)

A system pointer to the process control space is materialized when the value of operand 3 is hex 25. If a process control space pointer is supplied in operand 2, it is ignored. A pointer to the process that is executing the MATPRATR instruction is always materialized.

A materialization option's value of hex 26 causes the adopted user profile list attributes to be materialized as follows:

- Materialization size specification — Char(8)
  - Number of bytes provided for materialization — Bin(4)
  - Number of bytes available for materialization — Bin(4)
- Reserved (binary 0) — Char(8)
- Pointer to the adopted user profile list last used to set this attribute — Space pointer
- Number of user profiles in the encapsulated adopted user profile list — Bin(2)
- Reserved — Char(14)
- List of user profiles in the encapsulated adopted user profile list (one system pointer to each user profile in the list) — System pointers

Due to verifications performed on the user profiles specified in an adopted process user profile list input to either the Initiate Process or Modify Process instructions, the encapsulated adopted user profile list may differ from the input list. When verification of an input user profile fails, it is not included in the encapsulated list.

The adopted user profile list attributes are not supplied with materialization option hex 00.

A materialization option's value of hex 27 causes the process control attributes to be materialized. Refer to the description of this materialization provided in prior text for this instruction.

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for this instruction.

Authorization Required

- Process control special authorization
  - For materializing a process other than the one executing this instruction
- Retrieve
  - Contexts referenced for address resolution

Lock Enforcement

- Materialize
  - Contexts referenced for address resolution

Exceptions

Exception		Operands 1	2	3	4	Other
06	Addressing					
	01 space addressing violation	X	X	X		
	02 boundary alignment	X	X	X		
	03 range	X	X	X		
	06 optimized addressability invalid	X	X	X		
08	Argument/parameter					
	01 parameter reference violation	X	X	X		
0A	Authorization					
	01 unauthorized for operation		X			
	04 unauthorized for process control		X			
10	Damage encountered					
	04 system object damage state	X	X	X	X	X
	05 authority verification terminated due to damaged object					X
	44 partial system object damage	X	X	X	X	X
1A	Lock state					
	01 invalid lock state				X	
1C	Machine-dependent exception					
	03 machine storage limit exceeded					X
20	Machine support					
	02 machine check					X
	03 function check					X
22	Object access					
	01 object not found	X	X	X		
	02 object destroyed	X	X	X		
	03 object suspended	X	X	X		

Exception		Operands 1	2	3	4	Other
	07 authority verification terminated due to destroyed object					X
24	Pointer specification					
	01 pointer does not exist	X	X	X		
	02 pointer type invalid	X	X	X		
	03 pointer addressing invalid object		X			
28	Process state					
	02 process control space not associated with a process		X			
2A	Program creation					
	06 invalid operand type	X	X	X		
	07 invalid operand attribute	X	X	X		
	08 invalid operand value range	X	X	X		
	0A invalid operand length	X				
	0C invalid operand odt reference	X	X	X		
	0D reserved bits are not zero	X	X	X	X	X
2E	Resource control limit					
	01 user profile storage limit exceeded					X
32	Scalar specification					
	03 scalar value invalid		X			
36	Space management					
	01 space extension/truncation					X
38	Template specification					
	03 materialization length exception	X				

14.2 Wait On Time (WAITTIME)

Op Code (Hex)	Operand 1
0349	Wait template

Operand 1: Character(16) scalar.

Description: This instruction causes the process to wait for a specified time interval. The current process is placed in wait state for the amount of time specified by the wait template in accordance with the specified wait options.

The format of the wait template for operand 1 is:

- Wait time interval — Char(8)
- Wait options — Char(2)
  - Access state control for entering wait — Bit 0

    0 = Do not modify access state
    1 = Modify access state
  - Access state control for leaving wait — Bit 1

    0 = Do not modify access state
    1 = Modify access state
  - MPL (multiprogramming level) control during wait — Bit 2

    0 = Do not remain in current MPL set
    1 = Remain in current MPL set
  - Reserved — Bits 3-15
- Reserved — Char(6)

The format of the wait time interval value is that of a 64-bit unsigned binary value where bit 41 is equal to 1024 microseconds, assuming the bits are numbered from 0 to 63.

The access state control options control whether the process access group (PAG) will be explicitly transferred between main and auxiliary storage when entering and leaving a wait as a result of execution of this instruction. Specification of modify access state requests that the PAG be purged from main to auxiliary storage for entering a wait and requests that the PAG be transferred from auxiliary to main storage for leaving a wait. Specification of do not modify access state requests that the PAG not be explicitly transferred between main and auxiliary storage as a result of executing this instruction.

The access state of the PAG is modified when entering the wait if the process' instruction wait initiation access state control attribute specifies allow access state modification, if the access state control for entering wait option specifies modify access state, and if the MPL control during wait option specifies do not remain in current MPL set.

The MPL control during wait option controls whether the process will be removed from the current MPL (multiprogramming level) set or remain in the

current MPL set when the process enters a wait as a result of executing this instruction.

When the MPL control during wait option specifies remain in current MPL set and the access state control for entering wait option specifies do not modify access state, the process will remain in the current MPL set for a maximum of 2 seconds. After 2 seconds, the process will automatically be removed from the current MPL set. The automatic removal does not change or affect the total wait time specified for the process in the wait time interval.

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for this instruction.

Exceptions

Exception		Operands 1	2	3	4	Other
06	Addressing					
	01 space addressing violation	X				
	02 boundary alignment	X				
	03 range	X				
	06 optimized addressability invalid	X				
08	Argument/parameter					
	01 parameter reference violation	X				
10	Damage encountered					
	04 system object damage state					X
	05 authority verification terminated due to damaged object					X
	44 partial system object damage					X
20	Machine support					
	02 machine check					X
	03 function check					X
22	Object access					
	02 object destroyed	X				
	03 object suspended	X				
	07 authority verification terminated due to destroyed object					X
24	Pointer specification					
	01 pointer does not exist	X				
	02 pointer type invalid	X				
2A	Program creation					
	06 invalid operand type	X				
	07 invalid operand attribute	X				
	0A invalid operand length	X				
	0C invalid operand odt reference	X				

Exception		Operands 1	2	3	4	Other
	0D reserved bits are not zero	X				X
2E	Resource control limit					
	01 user profile storage limit exceeded					X
32	Scalar specification					
	01 scalar type invalid	X				
	02 scalar attributes invalid	X				
	03 scalar value invalid	X				
36	Space management					
	01 space extension/truncation					X

Chapter 15. Resource Management Instructions

This chapter describes the storage and resource management instructions. These instructions are in alphabetic order. For an alphabetic summary of all the instructions, see Appendix A, "Instruction Summary."

15.1 Ensure Object (ENSOBJ)

Op Code (Hex)	Operand 1
0381	System pointer

Operand 1: System pointer.

Description: The object identified by operand 1 is protected from volatile storage loss. The machine ensures that any changes made to the specified object are recorded on nonvolatile storage media. The access state of the object is not changed by this instruction. If operand 1 addresses a temporary object, no operation is performed because temporary objects are not preserved during a machine failure. No exception is signaled if temporary objects are referenced.

Authorization Required

- Retrieve
  - Contexts referenced for address resolution

Lock Enforcement

- Materialize
  - Contexts referenced for address resolution

Exceptions

Exception		Operands 1	Operands Other
06	Addressing		
	01 space addressing violation	X	
	02 boundary alignment	X	
	03 range	X	
	06 optimized addressability invalid	X	
08	Argument/parameter		
	01 parameter reference violation	X	
0A	Authorization		
	01 unauthorized for operation	X	
10	Damage encountered		
	04 system object damage state	X	
	05 authority verification terminated due to damaged object		X
	44 partial system object damage	X	X
1A	Lock state		
	01 invalid lock state	X	
1C	Machine-dependent exception		
	03 machine storage limit exceeded		X
20	Machine support		

Exception		Operands 1	Other
	02 machine check		X
	03 function check		X
22	Object access		
	01 object not found	X	
	02 object destroyed	X	
	03 object suspended	X	
	04 object not eligible for operation	X	
	07 authority verification terminated due to destroyed object		X
24	Pointer specification		
	01 pointer does not exist	X	
	02 pointer type invalid	X	
	03 pointer addressing invalid object	X	
2A	Program creation		
	06 invalid operand type	X	
	07 invalid operand attributes	X	
	08 invalid operand value range	X	
	0C invalid operand odt reference	X	
	0D reserved bits are not zero	X	X
2E	Resource control limit		
	01 user profile storage limit exceeded		X
30	Journal management		
	02 entry not journaled	X	
32	Scalar specification		
	01 scalar type invalid	X	
36	Space management		
	01 space extension/truncation		X

15.2 Materialize Access Group Attributes (MATAGAT)

Op Code (Hex)	Operand 1	Operand 2
03A2	Receiver	Access group

Operand 1: Space pointer.

Operand 2: System pointer.

Description: The attributes of the access group and the identification of objects currently contained in the access group are materialized into the receiving object specified by operand 1.

The materialization must be aligned on a 16-byte boundary. The format is:

- Materialization size specification — Char(8)
  - Number of bytes provided for materialization — Bin(4)
  - Number of bytes available for materialization — Bin(4)
- Object identification — Char(32)
  - Object type — Char(1)
  - Object subtype — Char(1)
  - Object name — Char(30)
- Object creation options — Char(4)
  - Existence attributes — Bit 0

    0 = Temporary
    1 = Reserved
  - Space attribute — Bit 1

    0 = Fixed-length
    1 = Variable-length
  - Context — Bit 2

    0 = Addressability not in context
    1 = Addressability in context
  - Reserved (binary 0) — Bits 3-12
  - Initialize space — Bit 13
  - Reserved (binary 0) — Bits 14-31
- Reserved (binary 0) — Char(4)
- Size of space — Bin(4)
- Initial value of space — Char(1)
- Performance class — Char(4)
  - Space alignment — Bit 0

0 = The space associated with the object is allocated to allow proper alignment of pointers at 16-byte alignments within the space. If no space is specified for the object, this value must be specified for the performance class.
1 = The space associated with the object is allocated to allow proper alignment of pointers at 16-byte alignments within the space as well as to allow proper alignment of input/output buffers at 512-byte alignments within the space.

- Reserved (binary 0) — Bits 1-4
- Default main storage pool — Bit 5

  0 = Process main storage pool is used for this object.
  1 = Machine default main storage pool is used for this object.

- Reserved (binary 0) — Bit 6
- Block transfer on implicit access state modification — Bit 7

  0 = Minimum storage transfer size for this object is transferred. This value is 1 storage unit.
  1 = Machine default storage transfer size is transferred. This value is 8 storage units.

- Reserved (binary 0) — Bits 8-31

- Reserved (binary 0) — Char(7)
- Context — System pointer
- Reserved (binary 0) — Char(16)
- Access group size — Bin(4)
- Reserved (binary 0) — Bin(4)
- Number of objects in the access group — Bin(4)
- Reserved (binary 0) — Char(4)
- Access group object system pointer (repeated for each object currently contained in the access group) — System pointer

The receiver space contains the access group's attributes (as defined by the Create Access Group instruction), the current status of the access group, and a system pointer to each object assigned to the access group.

The access group size represents the total amount of space that has been allocated to the access group. The amount of available space represents the amount of space that is available in the access group for additional objects.

There is one access group object system pointer for each object currently assigned to the access group. The authorization field within each system pointer is not set.

Authorization Required

- Retrieve
  - Operand 2
  - Contexts referenced for address resolution

Lock Enforcement

- Materialize
  - Operand 2
  - Contexts referenced for address resolution

Exceptions

Exception		Operands 1	Operands 2	Other
06	Addressing			
	01 space addressing violation	X	X	
	02 boundary alignment	X	X	
	03 range	X	X	
	06 optimized addressability invalid	X	X	
08	Argument/parameter			
	01 parameter reference violation	X	X	
0A	Authorization			
	01 unauthorized for operation		X	
10	Damage encountered			
	04 system object damage state	X	X	X
	05 authority verification terminated due to damaged object			X
	44 partial system object damage	X	X	X
1A	Lock state			
	01 invalid lock state		X	X
1C	Machine-dependent exception			
	03 machine storage limit exceeded			X
20	Machine support			
	02 machine check			X
	03 function check			X
22	Object access			
	01 object not found	X	X	
	02 object destroyed	X	X	
	03 object suspended	X	X	
	07 authority verification terminated due to destroyed object			X
24	Pointer specification			

Exception		Operands 1	2	Other
	01 pointer does not exist	X	X	
	02 pointer type invalid	X	X	
	03 pointer addressing invalid object		X	
2A	Program creation			
	06 invalid operand type	X	X	
	07 invalid operand attribute	X	X	
	08 invalid operand value range	X	X	
	0A invalid operand length	X		
	0C invalid operand odt reference	X	X	
	0D reserved bits are not zero	X	X	X
2E	Resource control limit			
	01 user profile storage limit exceeded			X
36	Space management			
	01 space extension/truncation			X
38	Template specification			
	03 materialization length exception	X		

15.3 Materialize Resource Management Data (MATRMD)

Op Code (Hex)	Operand 1	Operand 2
0352	Receiver	Control data

Operand 1: Space pointer.

Operand 2: Character(8) scalar (fixed-length).

Description: The data items requested by operand 2 are materialized into the receiving object specified by operand 1. Operand 2 is an 8-byte character scalar. The first byte identifies the generic type of information being materialized, and the remaining 7 bytes further qualify the information desired.

Operand 1 contains the materialization and has the following format:

- Materialization size specification Char(8)
  - Number of bytes provided for materialization Bin(4)
  - Number of bytes available for materialization Bin(4)
- Time of day Char(8)
- Resource management data Char(*)

The remainder of the materialization depends on operand 2 and on the machine implementation. The following values are allowed for operand 2:

- Selection option Char(1)

  Hex 01 = Materialize processor utilization data
  Hex 03 = Materialize storage management counters
  Hex 04 = Materialize storage transient pool information
  Hex 08 = Materialize machine address threshold data
  Hex 09 = Materialize main storage pool information
  Hex 0A = Materialize MPL control information
  Hex 0C = Materialize machine reserved storage pool information
  Hex 11 = Materialize user storage area 1
  Hex 12 = Materialize auxiliary storage information
- Reserved (binary 0) Char(7)

The following defines the formats and values associated with each of the above materializations of resource management data.

Processor Utilization (Hex 01):

- Processor time since IPL (initial program load)Char(8)

Processor time since IPL is the total amount of processor time used, both by instruction processes and internal machine functions, since IPL. The significance of bits within the field is the same as that defined for the time-of-day clock.

Storage Management Counters (Hex 03):

- Access pending Bin(2)

- Storage pool delays Bin(2)
- Directory look-up operations Bin(4)
- Directory page faults Bin(4)
- Access group member page faults Bin(4)
- Microcode page faults Bin(4)
- Microtask read operations Bin(4)
- Microtask write operations Bin(4)
- Reserved Bin(4)

Access pending is a count of the number of times that a paging request must wait for the completion of a different request for the same page.

Storage pool delays is a count of the number of times that processes have been momentarily delayed by the unavailability of a main storage frame in the proper pool.

Directory look-up operations is a count of the number of times that auxiliary storage directories were interrogated, exclusive of storage allocation or deallocation.

Directory page faults is a count of the number of times that a page of the auxiliary storage directory was transferred to main storage, to perform either a look-up or an allocation operation.

Access group member page faults is a count of the number of times that a page of an object contained in an access group was transferred to main storage independently of the containing access group. This occurs when the containing access group has been purged or because portions of the containing access group have been displaced from main storage.

Microcode page faults is a count of the number of times a page of microcode was transferred to main storage.

Microtask read operations is a count of the number of transfers of one or more pages of data from auxiliary main storage on behalf of a microtask rather than a process.

Microtask write operations is a count of the number of transfers of one or more pages of data from main storage to auxiliary storage on behalf of a microtask, rather than a process.

Storage Transient Pool Information (Hex 04):

- Storage pool to be used for the transient pool Bin(2)

The pool number materialized is the number of the main storage pool, which is being used as the transient storage pool. A value of 0 indicates that the transient pool attribute is being ignored.

Machine Address Threshold Data (Hex 08):

- Total permanent addresses possible Char(8)
- Total temporary addresses possible Char(8)

- Permanent addresses remaining Char(8)
- Temporary addresses remaining Char(8)
- Permanent addresses threshold Char(8)
- Temporary addresses threshold Char(8)

Total permanent addresses possible is the maximum number of permanent addresses that can exist on the machine.

Total temporary addresses possible is the maximum number of temporary addresses that can exist on the machine.

Permanent addresses remaining is the number of permanent addresses that can still be created before address regeneration must be run.

Temporary addresses remaining is the number of temporary addresses that can still be created before address regeneration must be run.

Permanent addresses threshold is a number that, when it exceeds the number of permanent addresses remaining, causes the event machine address threshold exceeded to be signaled. When the event is signaled, the threshold is reset to 0.

Temporary addresses threshold is a number that, when it exceeds the number of temporary addresses remaining, causes the event machine address threshold exceeded to be signaled. When the event is signaled, the threshold is reset to 0.

Main Storage Pool Information (Hex 09):

- Machine minimum transfer size Bin(2)
- Maximum number of pools Bin(2)
- Current number of pools Bin(2)
- Main storage size Bin(4)
- Reserved (binary 0) Char(2)
- Pool 1 minimum size Bin(4)
- Individual main storage pool information (repeated once for each pool, up to the current number of pools) Char(32)
  - Pool size Bin(4)
  - Pool maintenance Bin(4)
  - Process interruptions (data base) Bin(4)
  - Process interruptions (nondata base) Bin(4)
  - Data transferred to pool (data base) Bin(4)
  - Data transferred to pool (nondata base) Bin(4)
  - Reserved (binary 0) Char(8)

Machine minimum transfer size is the smallest number of bytes that may be transferred as a block to or from main storage.

Maximum number of pools is the maximum number of storage pools into which main storage may be partitioned. These pools will be assigned the logical identification beginning with 1 and continuing to the maximum number of pools.

Current number of pools is a user-specified value for the number of storage pools the user wishes to utilize. These are assumed to be numbered from 1 to the number specified. This number is fixed by the machine to be equal to the maximum number of pools.

Main storage size is the amount of main storage, in units equal to the machine minimum transfer size, which may be apportioned among main storage pools.

Pool 1 minimum size is the amount of main storage, in units equal to the machine minimum transfer size, which must remain in pool 1. This amount is machine and configuration dependent.

Individual main storage pool information is data in an array that is associated with a main storage pool by virtue of its ordinal position within the array. In the descriptions below, data base refers to all other data, including internal machine fields. Pool size, pool maintenance, and data transferred information is expressed in units equal to the machine minimum transfer size described above.

Pool size is the amount of main storage assigned to the pool.

Pool maintenance is the amount of data written from a pool to secondary storage by the machine to satisfy demand for resources from the pool. It does not represent total transfers from the pool to secondary storage, but rather is an indication of machine overhead required to provide primary storage within a pool to requesting processes.

Process interruptions (data base and nondata base) is the total number of interruptions to processes (not necessarily assigned to this pool) which were required to transfer data into the pool to permit instruction execution.

Data transferred to pool (data base and nondata base) is the amount of data transferred from auxiliary storage to the pool to permit instruction execution and as a consequence of set access state, implicit access group movement, and internal machine actions.

Multiprogramming Level Control Information (Hex 0A):

- Machine-wide MPL control Char(16)
  - Machine maximum number of MPL classes Bin(2)
  - Machine current number of MPL classes Bin(2)
  - MPL (max) Bin(2)
  - Ineligible event threshold Bin(2)
  - MPL (current) Bin(2)
  - Number of processes in ineligible state Bin(2)
  - Reserved Char(4)

- MPL class information (repeated for each MPLChar(16) class, from 1 to the current number of MPL classes)
  - MPL (max) Bin(2)
  - Ineligible event threshold Bin(2)
  - Current MPL Bin(2)
  - Number of processes ineligible state Bin(2)
  - Number of processes assigned to class Bin(2)
  - Transitions (active to ineligible) Bin(2)
  - Transitions (active to MI wait) Bin(2)
  - Transitions (MI wait to ineligible) Bin(2)

Machine-Wide MPL Control:

Maximum number of MPL classes is the largest number of MPL classes allowed in the machine. These are assumed to be numbered from 1 to the maximum.

Current number of MPL classes is a user-specified value for the number of MPL classes in use. They are assumed to be numbered from 1 to the current number.

MPL (max) is the maximum number of processes which may concurrently be in the active state in the machine.

Ineligible event threshold is a number which, if exceeded by the machine number of ineligible processes defined below, will cause the machine ineligible threshold exceeded event to be signaled. When the event is signaled, this value is set by the machine to 65 535.

MPL (current) is the current number of processes in the active state.

Number of processes in the ineligible state is the number of processes not currently active because of enforcement of both the machine and class MPL rules.

MPL Class Information:

MPL class controls is data in an array that is associated with an MPL class by virtue of its ordinal position within the array.

MPL (max) is the number of processes assigned to the class which may be concurrently active.

Ineligible event threshold, MPL (current), and number of processes in ineligible state are as defined above but apply only to processes assigned to the class.

Number of processes assigned to class is the total number of processes, in any state, assigned to the pool.

Transitions count is the total number of transitions by processes assigned to a class as follows:

1. Active state to ineligible state
2. Active state to wait
3. Wait state to ineligible state

Note that transitions from wait state to active state can be derived as (2 - 3) and transitions from ineligible state to active state as (1 + 3). These numbers are unsigned Bin(2) and are maintained by the machine without regard to overflow conditions.

Machine Reserved Storage Pool Information (Hex 0C):

- Current number of pools — Bin(2)
- Reserved — Char(6)
- Individual main storage pool information (repeated once for each pool, up to the current number of pools) — Char(16)
  - Pool size — Bin(4)
  - Machine portion of the pool — Bin(4)
  - Number of load/dump sessions — Bin(2)
  - Reserved — Char(6)

Pool size is the amount of main storage assigned to the pool (including the machine reserved portion).

Machine portion of the pool specifies the amount of storage from the pool that is dedicated to machine functions.

User storage area 1 (Hex 11):

- User data — Char(*)

The user data previously stored internally in the machine through usage of the corresponding option on the Modify Resource Management Controls instruction is materialized into the receiver. The operand 1 template, for this option, must start on a 16 byte boundary and any pointers contained in the user data are preserved in the materialization.

The length value materialized in the number of bytes available for materialization field of operand 1 specifies the length of the entire operand 1 template and is limited, through checks performed on the modify operation, to a maximum value of 65 504 (64K-32) bytes. The actual length of the user data materialized is calculated by subtracting 16 from the length value for the total template length.

Auxiliary Storage Information (Hex 12):

The auxiliary storage information describes the ASPs (auxiliary storage pools) which are configured within the machine and the units of auxiliary storage currently allocated to an ASP or known to the machine but not allocated to an ASP.

Note that contrary to the normal case of being able to modify the values materialized by this option through use of the Modify Resource Management Controls instruction, modification of most of the auxiliary storage configuration is performed using functions available in DST (the Dedicated Service Tool).

Also note that through appropriate setting of the number of bytes provided field for operand 1, the amount of information to be materialized for this option can be reduced thus avoiding the processing for unneeded information. As an example, by setting this field to only provide enough bytes for the common 16 byte header, plus the option Hex 12 control information, plus the system ASP entry of the ASP information, you can get just the information up through the system ASP entry returned and avoid the overhead for the user ASPs and unit information.

Control information (occurs just once)	Char(64)
• Number of ASPs	Bin(2)
• Number of allocated auxiliary storage units Note: Number of configured, non-mirrored units + number of mirrored pairs	Bin(2)
• Number of unallocated auxiliary storage units	Bin(2)
• Control flags	Char(1)
– Main storage dump area unavailable	Bit 0
– Reserved	Bits 1-7
• Reserved	Char(1)
• Maximum unprotected space used	Bin(8)
• Current unprotected space in use	Bin(8)
• Checksum main storage	Bin(4)
• Unit information offset	Bin(4)
• Number of pairs of mirrored units	Bin(2)
• Mirroring main storage	Bin(4)
• Reserved	Char(26)
ASP information (Repeated once for each ASP. Located immediately after the control information above. ASP 1, always configured, is first. Configured user ASPs follow in ascending numerical order.)	Char(160)
• ASP number	Char(2)
• ASP control flags	Char(1)
– Suppress threshold exceeded event	Bit 0
– User ASP overflow	Bit 1
– Checksum protection	Bit 2
– Unprotected space overflow	Bit 3
– ASP mirrored	Bit 4
– User ASP MI State	Bit 5
– Reserved	Bits 6-7
• Reserved	Char(5)
• ASP media capacity	Bin(8)

- Reserved Bin(8)
- ASP space available Bin(8)
- ASP event threshold Bin(8)
- ASP event threshold percentage Bin(2)
- Reserved Char(54)
- ASP checksum information Char(64)
  - Protected space capacity Bin(8)
  - Unprotected space capacity Bin(8)
  - Protected space available Bin(8)
  - Unprotected space available Bin(8)
  - Unprotected space on each checksummed unit Bin(4)
  - Reserved Char(28)

Unit information Char(208)
(Consists of one entry each for the configured, non-mirrored units and one unit of the mirrored pairs, the non-configured units, and the other unit of the mirrored pairs.

An allocated storage unit (ASU) is either an allocated, non-mirrored unit or a mirrored pair. Note that the mirrored pair counts only as one ASU. When used without qualification, the term unit refers to an ASU.

Unit information start may be located by the Unit Information Offset in the control information.)

- Device type Char(8)
  - Disk Type Char(4)
  - Disk Model Char(4)
- Device identification Char(8)
  - Unit number Char(2)
  - Serial number Char(4)
  - Reserved Char(2)
- Unit relationship Char(4)
  - Reserved Char(1)
  - Bus information Char(1)
    - Bus number Bits 0-2
    - Bus unit (IOP) Bits 3-7
  - Controller identification Char(1)
  - Actuator identification Char(1)
- Unit ASP number Char(2)
- Logical mirrored pair status Char(1)

  - Unit mirrored — Bit 0
  - Mirrored unit protected — Bit 1
  - Mirrored pair reported — Bits 2
  - Reserved — Bits 3-7
- Mirrored unit status — Char(1)
- Unit media capacity — Bin(8)
- Reserved — Bin(8)
- Unit space available — Bin(8)
- Unit reserved system space — Bin(8)
- Reserved — Char(24)
- Unit checksum information — Char(64)
  - Unit redundancy space — Bin(8)
  - Unit protected space capacity — Bin(8)
  - Unit protected space available — Bin(8)
  - Unit unprotected space capacity — Bin(8)
  - Unit unprotected space available — Bin(8)
  - Unit checksum set number — Char(2)
  - Reserved — Char(22)
- Unit usage information — Char(64)
  - Blocks transferred to main storage — Bin(4)
  - Blocks transferred from main storage — Bin(4)
  - Requests for data transfer to main storage — Bin(4)
  - Requests for data transfer from main storage — Bin(4)
  - Permanent blocks transferred from main storage — Bin(4)
  - Requests for permanent data transfer from main storage — Bin(4)
  - Redundancy blocks transferred from main storage — Bin(4)
  - Requests for redundancy data transfer from main storage — Bin(4)
  - Reserved — Char(32)

Number of ASPs - is the number of ASPs configured within the machine. One, the minimum value, indicates just the system ASP exists and that there are no user ASPs configured. Up to 15 user ASPs can be configured. Values greater than one indicate how many user ASPs are configured in addition to the system ASP. The system ASP always exists.

Number of allocated auxiliary storage units - is the number of configured units logically addressable by the system as units. This is the number of configured, non-mirrored units plus the number of mirrored pairs allocated to the ASPs. The total number of units (actuator arms) on the system is the sum of the allocated auxiliary storage units plus the number of unallocated

auxiliary storage units plus the number of pairs of mirrored units. For example, each 9335 enclosure represents two units. Information on these units is materialized as part of the unit information. Any two units of the same type and size may be associated to form a mirrored pair. Association of two units as a mirrored pair reduces the amount of logically available storage by the number of bytes contained on one of the mirrored units in the mirrored pair.

Number of unallocated auxiliary storage units - is the number of auxiliary storage units that are currently not allocated to an ASP. Information on these units is materialized as part of the unit information.

Main storage dump area unavailable flag - indicates whether or not the main storage dump area on the load source disk unit is unavailable. A value of binary 1 indicates it is unavailable; binary 0 indicates it is available. The condition where it is unavailable can arise when main storage is added to the machine, but during subsequent IPL processing the machine can not free up space on the load source disk unit for the additional dump area needed. This occurs when there is insufficient space available on the other disk units in the system ASP to allow for movement of object allocations off of the load source unit. The corrective action is to free up space in the system ASP and reIPL the machine so the allocation of additional space to the dump area can be completed.

The main storage dump area is important for recovery and diagnostic purposes. It is used by the machine during certain hardware and power failures to capture a main storage dump which is used to minimize the object damage which can result. It is also used by the machine during certain software logic failures to capture a main storage dump which is used to determine the cause of the failure.

Maximum unprotected space used (Checksum field) - is the largest number of bytes of unprotected storage used at any one time since the last IPL of the machine. When checksum protection is not in effect for the system ASP, this field describes the amount of unprotected storage that would have been used if checksum protection had been in effect.

Current unprotected space used (Checksum field) - is the current number of bytes of unprotected storage in use. When checksum protection is not in effect for the system ASP, this field describes the amount of unprotected storage that would be in use if checksum protection was in effect.

Checksum main storage (Checksum field) - is the number of bytes of main storage reserved in the machine storage pool for checksum usage.

Unit information offset - is the offset, in bytes, from the start of the operand 1 materialization template to the start of the unit information. This value can be added to a space pointer addressing the start of operand 1 to address the start of the unit information.

Number of pairs of mirrored units - represents the number of mirrored pairs in the system. Each mirrored pair consists of two mirrored units; however, only one of the two mirrored units is guaranteed to be operational.

Mirroring main storage - is the number of bytes of main storage in the machine storage pool used by mirroring. This increases when mirror synchronization is active. This amount of storage is directly related to the number of mirrored pairs.

ASP information - is repeated once for each ASP configured within the machine. The number of ASPs configured is specified by the number of ASPs field. ASP 1, the system ASP is materialized first. Because the system ASP always exists, its materialization is always available. The user ASPs which are configured are materialized after the system ASP in ascending numerical order. There may be gaps in the numerical order. That is, if just user ASPs 3 and 5 are configured, only information for them is materialized producing information on just ASP 1, ASP 3 and ASP 5 in that order.

ASP number - uniquely identifies the auxiliary storage pool. The ASP number may have a value from 1 through 16. A value of 1 indicates the system ASP. A value of 2 through 16 indicates a user ASP.

Suppress threshold exceeded event flag - indicates whether or not the machine is suppressing signaling of the related event when the event threshold in effect for this ASP has been exceeded. A value of binary 1 indicates that the signaling is being suppressed; binary 0 indicates that the signaling is not being suppressed. The default after each IPL of the machine is that the signaling is not suppressed; i.e. default is binary 0. For the system ASP, this flag is implicitly set to binary 1 by the machine when the machine auxiliary storage threshold exceeded event is signaled. For a user ASP, this flag is implicitly set to binary 1 by the machine when the user auxiliary storage threshold exceeded event is signaled. This is done to avoid repetitive signaling of the event when the threshold exceeded condition occurs. Option Hex 12 of the Modify Resource Management Controls instruction can be used to explicitly reset the suppression of the threshold exceeded event when it is desirable to again have the machine detect the threshold exceeded condition and signal the related event.

User ASP overflow flag (Checksum field) - indicates whether or not object allocations directed into the user ASP have overflowed into the system ASP. A value of binary 1 indicates overflow; binary 0 indicates no overflow. This flag does not apply to the system ASP, and is always set to a binary 0 for it.

Checksum protection flag - specifies whether or not the ASP is under checksum protection. A value of binary 1 indicates that checksum protection is in effect; a value of binary 0 indicates it is not. Because checksum protection is only allowed for the system ASP, this flag is only applicable to the system ASP.

Unprotected space overflow flag (Checksum field) - specifies whether or not allocations for unprotected data in the ASP have exceeded the unprotected space capacity and overflowed into the area normally used for allocation of protected data. A value of binary 1 indicates that such overflow has occurred; a value of binary 0 indicates it has not. This status is set when the ASP unprotected space overflow event is signaled; it is reset automatically on the subsequent IMPL of the machine. Because unprotected storage is used primarily for allocation of temporary objects which are automatically deallocated as part of the IPL process, the overflowed allocations are freed up at IPL, providing for the automatic reset of the overflow condition.

Because checksum protection is only allowed for the system ASP, this flag is only applicable to the system ASP.

ASP mirrored flag - specifies whether or not the ASP is configured to be mirror protected. A value of binary 1 indicates that ASP mirror protection is configured. Refer to the mirror unit protected flag to determine if mirror protection is active for each mirrored pair. A value of binary 0 indicates that none of the units associated with the ASP are mirrored.

User ASP MI State - indicates the state of the User ASP. A value of binary 1 indicates that the User ASP is in the 'new' state. This means that a context may be allocated in this User ASP. A value of binary 0 indicates that the User ASP is in the 'old' state. This means that there are no contexts allocated in this User ASP. This flag has no meaning for the System ASP and will always be set to binary 0 for it.

ASP media capacity - specifies the total space, in number of bytes of auxiliary storage, on the storage media allocated to the ASP. This is just the sum of the unit media capacity fields for (1) the units allocated to the ASP or (2) the mirrored pairs in the ASP.

ASP space available - is the number of bytes of secondary storage space that is not currently assigned to objects or internal machine functions, and therefore, is available for allocation in the ASP when the ASP is not under checksum protection. Note that a mirrored pair counts for only one unit. When the ASP is under checksum protection, this value is meaningless and the ASP checksum information describes the space available values.

ASP event threshold - specifies the minimum value for the number of bytes of auxiliary storage space available in the ASP prior to the signaling of the appropriate threshold exceeded event. The threshold exceeded condition occurs when either the protected space available value or the ASP space available value, depending upon whether checksum protection is or isn't in effect for the ASP, becomes equal to or less than the ASP event threshold value. This condition causes either the auxiliary storage threshold exceeded event, for the system ASP, or the user ASP threshold exceeded event, for a user ASP, to be signaled. Redundant signaling of these events is suppressed as indicated by the setting of the suppress threshold exceeded event flag. Refer to the definition of the suppress threshold exceeded event flag for more information.

The ASP event threshold value is calculated from the the ASP event threshold percentage value by multiplying either the protected space capacity value or the ASP media capacity value, depending upon whether checksum protection is or isn't in effect for the ASP, by the ASP event threshold percentage and subtracting the product from that same capacity value.

ASP event threshold percentage - specifies the auxiliary storage space utilization threshold as a percentage of either the protected space capacity or the ASP media capacity, depending upon whether checksum protection is or isn't in effect for the ASP. This value is used, as described above, to calculate the ASP event threshold value. This value can be modified through use of Dedicated Service Tool DASD configuration options.

ASP checksum information (Checksum field) - specifies capacity and space available values that apply when the ASP is under checksum protection. In this case, the units of auxiliary storage allocated to ASP are formatted with areas for protected data, unprotected data, and redundancy data. Information on the protected and unprotected space is provided both here in these fields on an ASP basis and under unit information on a per unit basis. Information on redundancy space is only provided under unit information on a per unit basis. When the ASP is not under checksum protection, the values of these fields are meaningless. Because checksum protection is only allowed for the system ASP, this information is only applicable to the system ASP.

Protected space capacity (Checksum field) - specifies the total number of bytes of auxiliary storage formatted for the storage of protected data in the ASP.

Unprotected space capacity (Checksum field) - specifies the total number of bytes of auxiliary storage formatted for storage of unprotected data in the ASP.

Protected space available (Checksum field) - specifies the number of bytes of auxiliary storage formatted for storage of protected data that are not currently assigned to objects or internal machine functions, and therefore, are available for allocation in the ASP.

Unprotected space available (Checksum field) - specifies the number of bytes of auxiliary storage formatted for storage of unprotected data that are not currently assigned to objects or internal machine functions, and therefore, are available for allocation in the ASP.

Unprotected space on each checksummed unit (Checksum field) - specifies the number of megabytes (millions of bytes) of auxiliary storage formatted for storage of unprotected data on each unit allocated to a checksum set in the ASP. Using the Dedicated Service Tool to modify this value provides for altering the relation of the protected versus unprotected space capacity values.

Unit information - is materialized in the following order:

Group 1: Configured units consisting of non-mirrored units and mirrored units.

Group 2: Non-configured units

Group 3: Configured units consisting of mirrored units.

Internal designators are used to guarantee consistency across DST, SST, and XPF ***.No code dependencies may be based on the order in which these units are materialized.*** The unit information is located by the unit information offset field which specifies the offset from the beginning of the operand 1 template to the start of the unit information. The number of entries for each of the three groups listed above is defined as follows:

Group 1: Number of non-mirrored, configured units + number of mirrored pairs

Group 2: Number of non-configured storage units

Group 3: Number of mirrored pairs

For unallocated units, the device type, device identification, unit relationship, and unit media capacity fields contain meaningful information. The remaining fields have no meaning for unallocated units because the units are not currently in use by the system. Mirrored unit entries contain either current or last known information. The last known data consists of the mirrored unit status, disk type, disk model, unit ASP number, disk serial number, and unit address. Last knows information is provided when the Mirrored Unit Reported field is a binary zero.

Disk type - identifies the type of disk enclosure containing this auxiliary storage unit. This is the four byte character field from the vital product data for the disk device which identifies the type of drive. For example, the value is character string '9332' for a 9332 device and '9335' for a 9335 device.

Disk model - identifies the model of the type of disk enclosure containing this auxiliary storage unit. This is the four byte character field from the vital product data for the disk device which identifies the model of the drive. For example, the value is character string '0200' for a model 200 9332 device and '0400' for a model 400 9332 device.

Unit number - Uniquely identifies each non-mirrored unit or mirrored pair among the configured units. Both mirrored units of a mirrored pair have the same unit number. The value of the unit number is assigned by the system when the unit is allocated to an ASP. For unallocated units, the unit number is set to binary zero.

Serial number - specifies the serial number of the device containing this auxiliary storage unit. This is the four byte serial number field from the vital product data for the disk device.

Bus number - identifies the I/O Bus to which the disk device containing this auxiliary storage unit is connected.

Bus unit (IOP) - identifies the I/O Processor used to access the controller for the disk device containing this auxiliary storage unit.

Controller identification - specifies the controller for the disk device containing this auxiliary storage unit.

Actuator identification - specifies the actuator associated with this auxiliary storage unit in the disk device containing it.

Unit ASP number - specifies the ASP to which this unit is currently allocated. A value of 1 specifies the system ASP. A value from 2 through 16 specifies a user ASP and correlates to the user ASP number field in the user ASP information entries. A value of 0 indicates that this unit is currently unallocated.

Unit mirrored flag - Indicates that this unit number represents a mirrored pair. This bit is materialized with both mirrored units of a mirrored pair.

Mirrored unit protected flag - indicates the mirror status of a mirrored pair. A value of binary 1 indicates that both mirrored units of a mirrored pair are active. A binary 0 indicates that one mirrored unit of a mirrored pair is not

active. Active means that both units are on line and fully synchronized (ie. the data is identical on both mirrored units).

Mirrored unit reported flag - indicates that a mirrored unit reported as present. The mirrored unit reported present during or following IMPL. Current attachment of a mirrored unit to the system **cannot** be inferred from this bit. A binary 0 indicates that the mirrored unit being materialized is missing. The last known information pertaining to the missing mirrored unit is materialized. A binary 1 indicates that the mirrored unit being materialized has reported. The information for this reported unit is current to the last time it reported status to the system.

Mirrored unit status - indicates mirrored unit status.

A value of 1 indicates that this mirrored unit of a mirrored pair is active (ie. on-line with current data).

A value of 2 indicates that this mirrored unit is being synchronized.

A value of 3 indicates that this mirrored unit is suspended.

Mirrored unit status is stored as binary data and is valid only when the unit mirrored flag is on.

Unit media capacity - is the space, in number of bytes of auxiliary storage, on the non-mirrored unit or mirrored pair, that is, the capacity of the unit prior to any formatting or allocation of space by the system it is attached to. For a mirrored pair, this space is the number of bytes of auxiliary storage on either one of the mirrored units. The space is identical on both of the mirrored units. Caution, do not attempt to add the capacities of the two units of a mirrored pair together.

Unit space available - is the number of bytes of secondary storage space that is not currently assigned to objects or internal machine functions, and therefore, is available for allocation on the unit (or the mirrored pair) when the ASP containing it is not under checksum protection. When the ASP containing the unit is under checksum protection, this value is meaningless and the Unit checksum information describes the space available values. For a mirrored pair, this space is the number of bytes of auxiliary storage available on either one of the mirrored units. The space is identical on both of the mirrored units. Caution, do not attempt to add the capacities of the two units of a mirrored pair together.

Unit reserved system space - is the total number of bytes of auxiliary storage on the unit reserved for use by the machine. This storage is not available for storing objects, redundancy data, and other internal machine data. When the unit is not in a checksum set, the unit checksum set number contains a value of zero, this reserved space is included in the ASP and unit media capacity fields and reduces the corresponding space available values. When the unit is in a checksum set, the unit checksum set number is nonzero, this reserved space is not included in the ASP and unit checksum information fields which provide capacity and space information and, therefore, does not reduce the corresponding space available values.

Unit checksum information (Checksum field) - specifies capacity and space available values that apply when the ASP containing the unit is under checksum protection. In this case, when the unit is in a checksum set, the

unit checksum set number is nonzero, the unit is formatted with areas for protected data, unprotected data, and redundancy data and these fields provide information relating to those areas. If the unit is not allocated to a checksum set, the unit checksum set number contains a value of zero, it is only formatted for the storage of unprotected data and the other capacity values will be zero. When the ASP containing the unit is not under checksum protection, the values of these fields are meaningless, except that the unit checksum set number field will contain a zero value. Because checksum protection is only allowed for the system ASP, this information is only applicable to units allocated to the system ASP.

Unit redundancy space - (Checksum field) is the total number of bytes of auxiliary storage on the unit formatted for use for redundancy data. This storage is not available for storing objects and other internal machine data.

Unit protected space capacity (Checksum field) - is the number of bytes of auxiliary storage formatted for storage of protected data on the unit. This field is only nonzero if this unit is allocated to a checksum set. Units not allocated to a checksum set contain no protected storage area. This value does not include the size of any data redundancy area which may have been formatted on the unit as well.

Unit protected space available (Checksum field) - is the number of bytes of protected space on secondary storage available for allocation on the unit; that is, not currently assigned to objects or internal machine functions. This field is only nonzero if this unit is allocated to a checksum set. Units not allocated to a checksum set contain no protected storage area, unless they are mirrored. All space of a mirrored pair is protected.

Unit unprotected space capacity (Checksum field) - is the number of bytes of auxiliary storage formatted for storage of unprotected data on the unit. This value does not include the size of any data redundancy area which may have been formatted on the unit as well.

Unit unprotected space available (Checksum field) - is the number of bytes of unprotected space on secondary storage that are not currently assigned to objects or internal machine functions, and therefore, are available for allocation on the unit.

Unit checksum set number (Checksum field) - specifies the checksum set to which this unit is currently allocated. A nonzero value specifies the number of the checksum set. A zero value specifies that the unit is currently not assigned to a checksum set.

Unit usage information - specifies statistics relating to usage of the unit. For unallocated units, these fields are meaningless.

Blocks transferred to/from main storage fields - specify the number of 512-byte blocks transferred for the unit since the last IMPL. These values wrap around to zero and continue counting in the case of an overflow of the field with no indication of the overflow having occurred.

Requests for data transfer to/from main storage fields - specify the number of data transfer (I/O) requests processed for the unit since the last IMPL. These values wrap around to zero and continue counting in the case of an

overflow of the field with no indication of the overflow having occurred. These values are not directly related to the number of blocks transferred for the unit because the number of blocks to be transferred for a given transfer request can vary greatly.

Permanent blocks transferred from main storage - specifies the number of 512-byte blocks of permanent data transferred from main storage to auxiliary storage for the unit since the last IMPL. In the case of an overflow of the field, this value wraps around back to zero and continues counting, with no indication of the overflow condition having occurred.

Requests for permanent data transfer from main storage - specifies the number of transfer (I/O) requests for transfers of permanent data from main storage to auxiliary storage that have been processed for the unit since the last IMPL. In the case of an overflow of the field, this value wraps around back to zero and continues counting, with no indication of the overflow condition having occurred. This value is not directly related to the permanent blocks transferred from main storage value for the unit ASP because the number of blocks to be transferred for any particular transfer request can vary greatly.

Redundancy blocks transferred from main storage (Checksum field) - specifies the number of 512-byte blocks of redundancy data transferred from main storage to auxiliary storage for the unit since the last IMPL. In the case of an overflow of the field, this value wraps around back to zero and continues counting, with no indication of the overflow condition having occurred. This field is only meaningful for a unit in a checksum set.

Requests for redundancy data transfer from main storage (Checksum field) - specifies the number of transfer (I/O) requests for transfers of redundancy data from main storage to auxiliary storage that have been processed for the unit since the last IMPL. In the case of an overflow of the field, this value wraps around back to zero and continues counting, with no indication of the overflow condition having occurred. This value is not directly related to the redundancy blocks transferred from main storage value for the unit because the number of blocks to be transferred for any particular transfer request can vary greatly. This field is only meaningful for a unit in a checksum set.

Exceptions

Exception		Operands 1	2	Other
06	Addressing			
	01 space addressing violation	X	X	
	02 boundary alignment	X	X	
	03 range	X	X	
	06 optimized addressability invalid	X	X	
08	Argument/parameter			
	01 parameter reference violation	X	X	
10	Damage encountered			
	04 system object damage state	X		X
	44 partial system object damage	X	X	X

Exception		Operands 1	Operands 2	Other
1C	Machine-dependent exception			
	03 machine storage limit exceeded			X
20	Machine support			
	02 machine check			X
	03 function check			X
22	Object access			
	01 object not found	X	X	
	02 object destroyed	X	X	
	03 object suspended	X	X	
24	Pointer specification			
	01 pointer does not exist	X	X	
	02 pointer type invalid	X	X	
2A	Program creation			
	06 invalid operand type	X	X	
	07 invalid operand attribute	X	X	
	08 invalid operand value range	X	X	
	0C invalid operand odt reference	X	X	
	0D reserved bits are not zero	X	X	X
2E	Resource control limit			
	01 user profile storage limit exceeded			X
32	Scalar specification			
	02 scalar attribute invalid		X	
	03 scalar value invalid	X		
36	Space management			
	01 space extension/truncation			X
38	Template specification			
	03 materialization length exception	X		

15.4 Set Access State (SETACST)

Op Code (Hex)	Operand 1
0341	Access state template

Operand 1: Space pointer.

Description: The instruction specifies the access state (which specifies the desired speed of access) that the issuing process has for a set of objects or subobject elements in the execution interval following the execution of the instruction. The specification of an access state for an object momentarily preempts the machine's normal management of an object.

The Set Access State instruction template must be aligned on a 16-byte boundary. The format is:

- Number of objects to be acted upon — Bin(4)
- Reserved (binary 0) — Char(12)
- Access state specifications (repeated as many times as necessary) — Char(32)
  - Pointer to object whose access state is to be changed — Space pointer or system pointer
  - Access state code — Char(1)
  - Reserved (binary 0) — Char(3)
  - Access state parameter — Char(12)

    Access pool ID — Char(4)
    Space length — Bin(4)
    Reserved (binary 0) — Char(4)

The number of objects entry specifies how many objects are potential candidates for access state modification. An access state specification entry is included for each object to be acted upon.

The pointer to object entry identifies the object or space which is to be acted upon. For the space associated with a system object, the space pointer may address any byte in the space. This pointer is followed by parameters that define in detail the action to be applied to the object.

The access state code designates the desired access state. The allowed values are as follows:

Access State Code (Hex)	Function and Required Parameter
00	No operations are performed.
01	Associated object is moved into main storage (if not already there) synchronously with the execution of the instruction.
02	Associated object is moved into main storage (if not already there) asynchronously with the execution of the instruction.

Access State Code (Hex)	Function and Required Parameter
03	Associated object is placed in main storage without regard to the current contents of the object. This causes access to secondary storage to be reduced or eliminated. For this access state code, a space pointer must be provided.
04	Associated object is removed from mainstore in a manner which reduces or eliminates access to secondary storage. Content of the object is unpredictable after this operation. For this access state code, a space pointer must be provided.
20	Associated object attributes are moved into main storage synchronous with the instruction's execution. The associated attributes are the attributes that are common to all system objects. The associated pointer to object must be a resolved system pointer.
21	Associated object attributes are moved into main storage asynchronous with the instruction's execution. The associated attributes are the attributes that are common to all system objects. The associated pointer to object must be a resolved system pointer.
40	Perform no operation on the associated object. The main storage occupied by this object is to be used, if possible, to satisfy the request in the next access state specification entry. Either a space or system pointer may be provided for this access state code.
41	Wait for any previously issued but incomplete X'81' or X'91' access state code operations to complete. This includes all previous X'81' and X'91' operations that may have been performed on previous Set Access State instructions within the current process as well as those that may have been issued in previous access state specification entries in the current instruction. The pointer is ignored for this access state code entry.
80	Object should be written and it is not needed in main storage by issuing process. Object is written to nonvolatile storage synchronously with the execution of the instruction. Any main storage that the object occupied is then marked as to make it quickly available for replacement.
81	Object should be written and it is not needed in main storage by issuing process. Object is written to nonvolatile storage asynchronously with the execution of the instruction. Any main storage that the object occupied is then marked as to make it quickly available for replacement. If desired, the process can synchronize with any outstanding X'81' access state operations by issuing a X'41' access state operations either within the current instruction or during a subsequent Set Access State instruction.
90	Associated object must be insured, but is still needed in main storage. Object is written to nonvolatile storage synchronously with the execution of the instruction. Unlike access state codes X'80' and X'81', this access state code does not mark any main storage occupied by the object as to make it quickly available for replacement.

Access State Code (Hex)	Function and Required Parameter
91	Associated object must be insured, but is still needed in main storage. Object is written to nonvolatile storage asynchronously with the execution of the instruction. Unlike access state codes X'80' and X'81', this access state code does not mark any main storage occupied by the object as to make it quickly available for replacement. If desired, the process can synchronize with any outstanding X'91' access state operations by issuing a X'41' access state operations either within the current instruction or during a subsequent Set Access State instruction.

Access state codes hex 03 and hex 04 may be used for spaces only. The pointer to the object in the access state specification must be a space pointer. Otherwise, the pointer type invalid exception is signaled.

Access state code hex 40 may be used in conjunction with access state codes hex 01, hex 02, or hex 03. The access state specification entry with access state code hex 40 must immediately precede the access state specification entry with access state code hex 01, hex 02, or hex 03 with which it is to be combined. The pointer to the object in both entries must be a space pointer. Otherwise, the pointer type invalid exception is signaled. The access state parameter field in the access state specification entry with code hex 40 is ignored. The access pool ID and the space length in the entry with access state code hex 01, hex 02, or hex 03 are used.

The access/pool ID entry indicates the desired main storage pool in which the object is to be placed (0-16). The storage pool ID entry is treated as a 4-byte logical binary value. When a 0 storage pool ID is specified, the storage pool associated with the issuing process is used.

The space length entry designates the part of the space associated with the object to be operated on. If the pointer to the object entry is a system pointer, the operation begins with the first byte of the space. If the pointer to the object entry is a space pointer that specifies a location, the operation proceeds for the number of storage units that are designated. No exception is signaled when the number of referenced bytes of the space are not allocated. When operations on objects are designated by system pointers, this operation is performed in addition to the access state modification of the object. This entry is ignored for access state codes hex 20 and hex 21.

Authorization Required

- Retrieve
  - Contexts referenced for address resolution

Lock Enforcement

- Materialize
  - Contexts referenced for address resolution

Exceptions

Exception		Operands 1	Operands Other
04	Access state		
	01 access state specification invalid	X	
06	Addressing		
	01 space addressing violation	X	
	02 boundary alignment	X	
	03 range	X	
	06 optimized addressability invalid	X	
08	Argument/parameter		
	01 parameter reference violation	X	
0A	Authorization		
	01 unauthorized for operation	X	
10	Damage encountered		
	04 system object damage state	X	
	05 authority verification terminated due to damaged object		X
	44 partial system object damage	X	X
1A	Lock state		
	01 invalid lock state	X	
1C	Machine-dependent exception		
	03 machine storage limit exceeded		X
20	Machine support		
	02 machine check		X
	03 function check		X
22	Object access		
	01 object not found	X	
	02 object destroyed	X	
	03 object suspended	X	
	07 authority verification terminated due to destroyed object		X
24	Pointer specification		
	01 pointer does not exist	X	
	02 pointer type invalid	X	
	03 pointer addressing invalid object	X	
	04 pointer not resolved	X	
2A	Program creation		
	06 invalid operand type	X	
	07 invalid operand attribute	X	

Exception		Operands 1	Other
	08 invalid operand value range	X	
	0C invalid operand odt reference	X	
	0D reserved bits are not zero	X	X
2E	Resource control limit		
	01 user profile storage limit exceeded		X
36	Space management		
	01 space extension/truncation		X
38	Template specification		
	01 template value invalid	X	

Chapter 16. Dump Space Management Instructions

This chapter describes all the instructions used for dump space management. These instructions are arranged in alphabetical order. For an alphabetic summary of all the instructions, see Appendix A, "Instruction Summary."

16.1 Materialize Dump Space (MATDMPS)

Op Code (Hex)	Operand 1	Operand 2
04DA	Receiver	Dump space

Operand 1: Space pointer.

Operand 2: System pointer.

Description: The current attributes of the dump space specified by operand 2 are materialized into the receiver specified by operand 1.

The first 4 bytes of the materialization identify the total quantity of bytes that may be used by the instruction. This value is supplied as input to the instruction and is not modified by the instruction. A value of less then eight causes the materialization length exception to be signaled.

The second 4 bytes of the materialization identify the total quantity of bytes available to be materialized. The instruction materializes as many bytes as can be contained in the area specified as the receiver. If the byte area identified by the receiver is greater than that required to contain the information requested, then the excess bytes are unchanged. No exceptions are signaled in the event that the receiver contains insufficient area for the materialization, other than the materialization length exception described previously.

The template identified by operand 1 must be 16-byte aligned in the space. The format of the materialization is as follows:

- Materialization Size Specification — Char(8)
  - Number of bytes provided for — Bin(4) materialization
  - Number of bytes available for materialization (always 128 for this instruction) — Bin(4)
- Object Identification — Char(32)
  - Object type — Char(1)
  - Object subtype — Char(1)
  - Object name — Char(30)
- Object Creation Options — Char(4)
  - Existence attributes — Bit 0

    0 = Temporary
    1 = Permanent
  - Space attribute — Bit 1

    0 = Fixed length
    1 = Variable length
  - Context — Bit 2

    0 = Addressability not in context
    1 = Addressability in context

  - Reserved (binary 0) Bit 3-12
  - Initialize space Bit 13
  - Reserved (binary 0) Bit 14-31
- Recovery Options Char(4)
- Size of Space Bin(4)
- Initial Value of Space Char(1)
- Performance Class Char(4)
- Reserved Char(7)
- Context System pointer
- Reserved Char(16)
- Dump Space Size Char(4)
- Dump Data Size Char(4)
- Dump Data Size Limit Char(4)
- Reserved Char(20)

The dump space size entry is set with the current size value for the number of 512-byte blocks of space allocated for storage of dump data within the dump space.

The dump data size entry is set with the current size value for the number of 512-byte blocks of dump data contained in the dump space. This value specifies the number of blocks from the start of the dump space through the block of dump data which has been placed into the dump space at the largest dump space offset value. A value of zero indicates that the dump space currently contains no dump data.

The dump data size limit entry is set with the current size limit for the number of 512-byte blocks of dump data which may be stored in the dump space. A value of zero indicates that no explicit limitation is placed on the amount of dump data which may be stored in the dump space. The machine implicitly places a limit on the maximum size of a dump space. This value of this limitation is dependent upon the specific implementation of the machine.

Authorization Required

- Operational
  - Operand 2
- Retrieve
  - Contexts referenced for address resolution

Lock Enforcement

- Materialize
  - Operand 2
  - Contexts referenced for address resolution

Exceptions

Exception		Operands 1	Operands 2	Other
06	Addressing			
	01 space addressing violation	X	X	
	02 boundary alignment violation	X	X	
	03 range	X	X	
	06 optimized addressability invalid	X	X	
08	Argument/parameter			
	01 parameter reference violation	X	X	
0A	Authorization			
	01 unauthorized for operation		X	
10	Damage encountered			
	04 system object damage state		X	X
	05 authority verification terminated due to damaged object			X
	44 partial system object damage			X
1A	Lock state			
	01 invalid lock state		X	
1C	Machine-dependent exception			
	03 machine storage limit exceeded			X
20	Machine support			
	02 machine check			X
	03 function check			X
22	Object access			
	01 object not found	X	X	
	02 object destroyed	X	X	
	03 object suspended	X	X	
	07 authority verification terminated due to destroyed object			X
24	Pointer specification			
	01 pointer does not exist	X	X	
	02 pointer type invalid	X	X	
	03 pointer addressing invalid object		X	
2A	Program creation			
	06 invalid operand type	X	X	
	07 invalid operand attribute	X	X	
	08 invalid operand value range	X	X	
	0A invalid operand length	X		
	0C invalid operand odt reference	X	X	

Exception		Operands 1	2	Other
	0D reserved bits are not zero	X	X	X
2E	Resource control limit			
	01 user profile storage limit exceeded			X
32	Scalar specification			
	01 scalar type invalid	X	X	
36	Space management			
	01 space extension/truncation			X
38	Template specification			
	01 template value invalid	X		
	03 materialization length exception	X		

Chapter 17. Machine Observation Instructions

This chapter describes all instructions used for machine observation. These instructions are arranged alphabetically. For an alphabetic summary of all the instructions, see Appendix A, "Instruction Summary."

17.1 Materialize Instruction Attributes (MATINAT)

Op Code (Hex)	Operand 1	Operand 2
0526	Receiver	Selection information

Operand 1: Space pointer.

Operand 2: Character scalar.

Description: This instruction materializes the attributes of the instruction that are selected in operand 2 and places them in the receiver (operand 1).

Operand 2 is a 16-byte template. Only the first 16 bytes are used. Any excess bytes are ignored. Operand 2 has the following format:

- Selection template — Char(16)
  - Invocation number — Bin(2)
  - Instruction number — Bin(4)
  - Reserved (binary 0) — Char(10)

The invocation number is a specific identifier for the target invocation, in the process, that is to be materialized. This program must be observable or the program not observable exception is signaled.

The instruction number specifies the instruction in the specified program invocation that is to be materialized.

Operand 1 is a space pointer that addresses a 16-byte aligned template where the materialized data is placed. The format of the data is as follows:

- Materialization size specification — Char(8)
  - Number of bytes provided for materialization — Bin(4)
  - Number of bytes available for materialization — Bin(4)
- Object identification — Char(32)
  - Program type — Char(1)
  - Program subtype — Char(1)
  - Program name — Char(30)
- Offset to instruction attributes — Bin(4)
- Reserved (binary 0) — Char(8)
- Instruction attributes — Char(*)
  - Instruction type — Char(2)
    - Instruction version — Bits 0-3

      Hex 0000 = 2-byte operand references
      Hex 0001 = 3-byte operand references

    - Reserved (binary 0) — Bits 4-15

- Instruction length as input to Create Program — Bin(2)
- Offset to instruction form specified as input to Create Program — Bin(4)
- Reserved (binary 0) — Char(4)
- Number of instruction operands — Bin(2)
- Operand attributes offsets — Char(*)
  - An offset is materialized for each of the the operands of the instruction specifying the offset to the attributes for the operand — Bin(4)

• Instruction form specified as input to Create Program — Char(*)

- Instruction operation code — Char(2)
- Optional extender field and operand fields — Char(*)

• Operand attributes — Char(*)

A set of attributes following this format is materialized for each of the operands of the instruction. Compound operand references result in materialization of only one set of attributes for the operand which describe the substring or array element as is appropriate. See the specific format described below for each operand type.

- Operand type — Bin(2)

  1 = Data object
  2 = Constant data object
  3 = Instruction number reference
  4 = Argument list
  5 = Exception description
  6 = Null operand
  7 = Space pointer machine object

- Operand specific attributes — Char(*)

  See descriptions below for detailed formats. Nothing is provided for null operands.

• Data object — Char(32)

For a data object, the following operand attributes are materialized.

- Operand type = 1 — Bin(2)
- Data object specific attributes — Char(7)

  Element type — Char(1)
  Hex 00 = Binary
  Hex 01 = Floating-point
  Hex 02 = Zoned decimal
  Hex 03 = Packed decimal
  Hex 04 = Character
  Hex 08 = Pointer
  Element length — Char(2)

  If binary, or character, or floating-point:
  Length — Bits 0-15

If zoned decimal or packed decimal:
  Fractional digits — Bits 0-7
  Total digits — Bits 8-15
If pointer:
  Length = 16 — Bits 0-15
Array size — Bin(4)
  If scalar, then value of 0.
  If array, then number of elements.

- Reserved (binary 0) — Char(6)
- Data object addressability — Char(17)

  Addressability indicator — Char(1)

  Hex 00 = Addressability was not established
  Hex 01 = Addressability was established

  Space pointer to the object if addressability could be established — Space pointer

• Constant data object — Char(*)

For a constant data object, the following operand attributes are materialized (immediate operands as constants, signed immediates as binary, and unsigned immediates as character).

- Operand type = 2 — Bin(2)
- Constant specific attributes — Char(7)

  Element type — Char(1)
  Hex 00 = Binary
  Hex 01 = Floating-point
  Hex 02 = Zoned decimal
  Hex 03 = Packed decimal
  Hex 04 = Character
  Element length — Char(2)

  If binary, or character, or floating-point:
    Length — Bits 0-15
  If zoned decimal or packed decimal:
    Fractional digits — Bits 0-7
    Total digits — Bits 8-15
  Reserved (binary 0) — Bin(4)

- Reserved (binary 0) — Char(7)
- Constant value — Char(*)

• Instruction references — Char(*)

For instruction references, either through instruction definition lists or immediate operands, the following operand attributes are materialized.

- Operand type = 3 — Bin(2)
- Number of instruction reference elements — Bin(2)

  1 = Single instruction reference
  >1 = Instruction definition list

- Reserved (binary 0) — Char(12)
- Reference list — Char(*)

The instruction number of each instruction reference is materialized in the order in which they are defined.

- Argument list — Char(*)

  For an argument list, the following operand attributes are materialized.

  - Operand type = 4 — Bin(2)
  - Argument list specific attributes — Char(4)

    Actual number of list entries — Bin(2)
    Maximum number of list entries — Bin(2)
  - Reserved (binary 0) — Char(10)
  - Addressability to list entries — Char(*)

    Space pointer to each list entry for the number of actual list entries. A value of all zeros is materialized if addressability could not be established. — Space pointer

- Exception description — Char(48)

  For an exception description, the following operand attributes are materialized.

  - Operand type = 5 — Bin(2)
  - Reserved (binary 0) — Char(10
  - Control flags — Char(2)

    Exception handling action — Bits 0-2

    000 = Ignore occurrence of exception and continue processing
    001 = Disabled exception description
    010 = Continue search for an exception description by resignaling the exception to the immediately preceding invocation
    100 = Defer handling
    101 = Pass control to the specified exception handler

    Reserved (binary 0) — Bits 3-15
  - Compare value length — Bin(2)
  - Compare value — Char(32)

- Space pointer machine object — Char(32)

  For a space pointer machine object, the following operand attributes are materialized.

  - Operand type = 7 — Bin(2)
  - Reserved (binary 0) — Char(13)
  - Pointer addressability — Char(17)

    - Pointer value indicator — Char(1)

      Hex 00 = Addressability value is not valid
      Hex 01 = Addressability value is valid
  - Space pointer data object containing the space pointer machine object value if addressability value is valid. — Space pointer

The first 4 bytes of the materialization identify the total number of bytes that may be used by the instruction. This value is supplied as input to the instruction and is not modified by the instruction. A value of less than 8 causes the materialization length exception to be signaled.

The second 4 bytes of the materialization identify the total number of bytes available to be materialized. The instruction materializes as many bytes as can be contained in the area specified as the receiver. If the byte area identified by the receiver is greater than that required to contain the information requested, then excess bytes are unchanged.

The materialization available for an instruction depends on the execution status of the program that the instruction is in. If the program has not executed to the point of the instruction, little or no meaningful information about the instruction can be materialized. If the program executes the instruction multiple times, the materialization will vary with each execution.

No exceptions are signaled in the event that the receiver contains insufficient area for the materialization, other than the materialization length exception described previously.

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for this instruction.

Exceptions

Exception		Operands 1	Operands 2	Other
06	Addressing			
	01 Space addressing violation	X	X	
	02 Boundary alignment	X	X	
	03 Range	X	X	
	06 Optimized addressability invalid	X	X	
08	Argument/Parameter			
	01 Parameter reference violation	X	X	
10	Damage Encountered			
	04 System object damage state			X
	44 Partial system object damage			X
1C	Machine-Dependent Exception			
	03 Machine storage limit exceeded			X
1E	Machine Observation			
	01 Program not observable		X	
20	Machine Support			
	02 Machine check			X
	03 Function check			X
22	Object Access			
	01 Object not found		X	

Exception		Operands 1	2	Other
	02 Object destroyed	X	X	
	03 Object suspended	X	X	
24	Pointer Specification			
	01 Pointer does not exist	X	X	
	02 Pointer type invalid	X	X	
2A	Program Creation			
	06 Invalid operand type	X	X	
	07 Invalid operand attribute	X	X	
	0C Invalid operand ODT reference	X	X	
	0D Reserved bits are not zero	X	X	X
2E	Resource Control Limit			
	01 User Profile storage limit exceeded			X
32	Scalar Specification			
	01 Scalar type invalid	X	X	
	02 Scalar attributes invalid	X	X	
	03 Scalar value invalid	X	X	
36	Space Management			
	01 Space Extension/Truncation			X
38	Template Specification			
	01 Template value invalid		X	
	03 Materialization length exception	X		

17.2 Materialize Invocation (MATINV)

Op Code (Hex)	Operand 1	Operand 2
0516	Receiver	Selection information

Operand 1: Space pointer.

Operand 2: Space pointer.

Description: The attributes of the invocation selected through operand 2 are materialized into the receiver designated by operand 1.

Operand 2 is a space pointer that addresses a template of the following form:

- Invocation number — Bin(2)
- Offset to list of parameters — Bin(4)
- Number of parameter ODV numbers — Char(2)
- Offset to list of exception descriptions — Bin(4)
- Number of exception description ODV (object definition table) numbers — Char(2)

The offset to the list of parameters and the offset to the list of exception descriptions are both relative to the start of the operand 2 template. Each list is an array of Char(2) ODV numbers. The number of parameter ODV numbers and the number of exception description ODV numbers define the sizes of the arrays.

Operand 2 is a space pointer that addresses a template that has the following format:

- Control information — Char(2)
  - Template extension — Bit 0

    0 = Template extension is not present.
    1 = Template extension is present.
  - Invocation number — Bits 1-15
- Offset to list of parameters — Bin(4)
- Number of parameter ODV numbers — Char(2)
- Offset to list of exception descriptions — Bin(4)
- Number of exception description ODV numbers — Char(2)
- Template extension (optional) — Char(14)
  - Offset to list of space pointer machine objects — Bin(4)
  -

    Number of space pointer machine object ODV numbers — Char(2)

- Reserved (binary 0) Char(8)

The offset to the list of space pointer machine objects, offset to the list of parameters, and the offset to the list of exception descriptions are relative to the start of the operand 2 template. Each list is an array of Char(2) ODV numbers. The number of space pointer machine object ODV numbers, number of parameter ODV numbers, and the number of exception description ODV numbers define the sizes of the arrays.

Operand 1 is a space pointer that addresses a 16-byte aligned template into which the materialized data is placed. The format of the data is:

- Materialization size specification Char(8)
  - Number of bytes provided for materialization Bin(4)
  - Number of bytes available for materialization Bin(4)
- Object identification Char(32)
  - Program type Char(1)
  - Program subtype Char(1)
  - Program name Char(30)
- Trace specification Char(2)
  - Invocation trace status Bit 0

    0 = Not tracing new invocations
    1 = Tracing new invocations
  - Return trace Bit 1

    0 = Not tracing returns
    1 = Tracing returns
  - Invocation trace propagation Bit 2

    0 = Not propagating invocation trace
    1 = Propagating invocation trace
  - Return trace propagation Bit 3

    0 = Not propagating return trace
    1 = Propagating return trace
  - Reserved (binary 0) Bits 4-15
- Instruction number Ubin(2)
- Offset to parameter values Bin(4)
- Offset to exception description value Bin(4)
- Offset to space pointer machine object values (Optional-This data is present only if the template extension is present in the selection information.) Bin(4)
- Space pointer machine objects (Optional-This data is present Char(*)

only if the template extension is present in the selection information.)

- For each ODV number specified for a space pointer machine object, the value of the space pointer machine object is materialized as follows: Char(32)

  Reserved (binary 0) Char(15)

  Pointer value indicator Char(1)

  00 = Addressability value is not valid
  01 = Addressability value is valid

  Space pointer data object containing the space pointer machine object value if addressability value is valid. Space pointer

- Parameters Char(*)
  - For each parameter ODT number specified the address of the parameter data is materialized (If no parameter ODT numbers are materialized, this parameter is binary 0.) Space pointer

- Exception description Char(*)
  - For each exception description ODT number specified, the following is materialized: Char(36)
  - Control flags Char(2)

    Exception handling action Bits 0-2

    000 = Ignore occurrence of exception and continue processing
    001 = Disabled exception description
    010 = Continue search for an exception description by resignaling the exception to the immediately preceding invocation
    100 = Defer handling
    101 = Pass control to the specified exception handler

    Reserved (binary 0) Bits 3-15
  - Compare value length Bin(2)
  - Compare value Char(32)

The first 4 bytes of the materialization identify the total number of bytes that may be used by the instruction. This value is supplied as input to the instruction and is not modified by the instruction. A value of less than 8 causes the materialization length exception to be signaled.

The second 4 bytes of the materialization identify the total number of bytes available to be materialized. The instruction materializes as many bytes as can be contained in the area specified as the receiver. If the byte area identified by the receiver is greater than that required to contain the information requested, then excess bytes are unchanged.

No exceptions (other than the materialization length exception) are signaled in the event that the receiver contains insufficient area for the materialization.

The instruction number returned depends on how control was passed from the invocation:

Exit Type	Instruction Number
Call External	Locates the Call External instruction
Event	Locates the next instruction to execute
Exception	Locates the instruction that caused the exception

The space pointers that address parameter values are returned in the same order as the corresponding ODT numbers in the input array. The same is true for the exception description values.

If the offset to the list of parameters or the number of parameter ODT numbers is 0, no parameters are returned and the offset to parameters value is 0. If any parameters are returned, they are 16-byte aligned. If the offset to list of exception descriptions or the number of exception description ODT numbers is 0, no exception descriptions are returned and the offset to exception description values are 0.

Exceptions

Exception		Operands 1	Operands 2	Other
06	Addressing			
	01 space addressing violation	X	X	
	02 boundary alignment	X	X	
	03 range	X	X	
	06 optimized addressability invalid	X	X	
08	Argument/parameter			
	01 parameter reference violation	X	X	
10	Damage encountered			
	04 system object damage state	X	X	X
	44 partial system object damage	X	X	X
1C	Machine-dependent exception			
	03 machine storage limit exceeded		X	
1E	Machine observation			
	01 program not observable			X
20	Machine support			
	02 machine check			X
	03 function check			X
22	Object access			
	01 object not found		X	
	02 object destroyed	X	X	
	03 object suspended	X	X	
24	Pointer specification			

Exception		Operands 1	2	Other
	01 pointer does not exist	X	X	
	02 pointer type invalid	X	X	
2A	Program creation			
	06 invalid operand type	X	X	
	07 invalid operand attribute	X	X	
	0C invalid operand odt reference	X	X	
	0D reserved bits are not zero	X	X	X
2E	Resource control limit			
	01 user profile storage limit exceeded			X
32	Scalar specification			
	01 scalar type invalid	X	X	
	02 scalar attributes invalid	X	X	
36	Space management			
	01 space extension/truncation			X
38	Template specification			
	01 template value invalid		X	
	03 materialization length exception	X		

17.3 Materialize Invocation Entry (MATINVE)

Op Code (Hex)	Operand 1	Operand 2	Operand 3
0547	Receiver	Selection information	Materialization options

Operand 1: Character variable scalar.

Operand 2: Character(8) scalar or null.

Operand 3: Character(1) scalar or null.

Description: This instruction materializes the attributes of the specified invocation entry within the process issuing the instruction. The attributes specified by operand 3 of the invocation selected through operand 2 are materialized into the receiver designated by operand 1.

Operand 2 is an 8-byte template or a null operand. If operand 2 is null, it indicates that the attributes of the current invocation are to be materialized. If operand 2 is not null, it must be an 8-byte template which specifies the invocation to be materialized. Only the first 8 bytes are used. Any excess bytes are ignored. It has the following format:

- Selection information — Char(8)
  - Relative invocation number — Char(2)
  - Reserved — Char(6)

If operand 2 is not null, it is restricted to a constant with the relative invocation number field specifying a value of zero, which indicates that the attributes of the current invocation are to be materialized.

Operand 3 is a 1-byte value or a null operand. If operand 3 is null, it indicates that the attributes for a materialization option value of hex 00 are to be materialized. If operand 3 is not null, it must be a 1-byte value which specifies the type of materialization to be performed. Option values that are not defined below are reserved values and may not be specified. Only the first byte is used. Any excess bytes are ignored. It has the following format:

- Materialization options — Char(1)

  Hex 00 = Long materialization
  Hex 01 = Short materialization type 1
  Hex 02 = Short materialization type 2
  Hex 03 = Short materialization type 3
  Hex 04 = Short materialization type 4
  Hex 05 = Short materialization type 5

If operand 3 is not null, it is restricted to a constant character scalar or an immediate value.

Operand 1 specifies a receiver into which the materialized data is placed. It must specify a character scalar with a minimum length which is dependent upon the materialization option specified for operand 3. If the length specified for operand 1 is less than the required minimum, an exception is signaled. Only the bytes up to the required minimum length are used. Any excess bytes are ignored. For the materialization options which produce

pointers in the materialized data, 16-byte space alignment is required for the receiver. The data placed into the receiver differs depending upon the materialization option specified. The following descriptions detail the formats of the optional materializations.

Long Materialization: For a materialization option value of hex 00, the minimum length for the receiver is 144 bytes. It has the following format:

- Hex 00 = Long materialization — Char(144)
  - Reserved — Char(12)
  - Mark counter — Bin(4)
  - Reserved — Char(32)
  - Associated program pointer (zero for data base select/omit program) — System pointer
  - Invocation number — Bin(2)
  - Invocation type — Char(1)

    Hex 00 = Data base select/omit program
    Hex 01 = Call external
    Hex 02 = Transfer control
    Hex 03 = Event handler
    Hex 04 = External exception handler
    Hex 05 = Initial program in process problem state
    Hex 06 = Initial program in process initiation state
    Hex 07 = Initial program in process termination state
    Hex 08 = Invocation exit
  - Reserved (binary 0) — Char(1)
  - Invocation mark — Bin(4)
  - State invocation was invoked with — Char(2)
  - State for invocation — Char(2)
  - Reserved — Char(4)
  - PASA entry pointer — Space pointer
  - PSSA entry pointer — Space pointer
  - Reserved — Char(32)

Short Materialization Type 1: For a materialization option value of hex 01, the minimum length for the receiver is 16 bytes. It has the following format:

Hex 01 = Short materialization type 1 — Char(16)

- Associated program pointer (null for data base select/omit program) — System pointer

Short Materialization Type 2: For a materialization option value of hex 02, the minimum length for the receiver is 4 bytes. It has the following format:

Hex 02 = Short materialization type 2 — Char(4)

- Invocation mark — Bin(4)

Short Materialization Type 3: For a materialization option value of hex 03, the minimum length for the receiver is 16 bytes. It has the following format:

Hex 03 = Short materialization type 3	Char(16)
• PASA entry pointer	Space pointer

Short Materialization Type 4: For a materialization option value of hex 04, the minimum length for the receiver is 16 bytes. It has the following format:

Hex 04 = Short materialization type 4	Char(16)
• PSSA entry pointer	Space pointer

Short Materialization Type 5: For a materialization option value of hex 05, the minimum length for the receiver is 4 bytes. It has the following format:

Hex 05 = Short materialization type 5	Char(4)
• State invocation was invoked with	Char(2)
• State for invocation	Char(2)

The mark counter value represents the current value of a counter used by the machine to mark all activations and invocations created during the execution of a process with a unique value. This mark indicates the point at which the specific entry was allocated relative to the sequence of all activations and invocations that have been created over time within the process.

The associated program pointer is a system pointer that locates the program associated with the invocation entry.

The invocation number is a number that uniquely identifies each invocation in the PASA. When an invocation is allocated, the invocation number of the new invocation entry is one more than that in the calling invocation. The first invocation in the current process state has an invocation number of one.

The invocation type indicates how the associated program was invoked.

The invocation mark indicates the point at which this invocation entry was allocated relative to the sequence of all activations and invocations that have been created over time within the process. This is set from the incremented mark counter value for each new invocation added to the invocation stack.

The state invocation was invoked with value represents the state in which the machine was running when the program was called or transfered to.

State for invocation value represents the state in which the machine is running the program.

The PASA entry pointer is a space pointer that is set to address the start of the PASAE (program automatic storage area entry) associated with the invocation. The associated program's automatic data starts 64 bytes after the area addressed by this pointer.

The PSSA entry pointer is a space pointer that is set to address the start of the PSSAE (program static storage area entry) associated with the invocation. The associated program's static data starts 64 bytes after the area addressed by this pointer. The first 64 bytes contain the header information

for the PSSAE. Refer to the Create Activation instruction for a description of this header information. This pointer will be set to a value of all zeros if the invoked program does not have static data.

The fields labeled reserved in the descriptions of the optional materializations are currently reserved for future use. These fields may be altered by this instruction depending upon the particular implementation of the machine. Any values set into these fields are meaningless.

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for this instruction.

Exceptions

Exception		Operands 1	2	3	Other
06	Addressing				
	01 space addressing violation	X	X	X	
	02 boundary alignment	X	X	X	
	03 range	X	X	X	
	06 optimized addressability invalid	X	X	X	
08	Argument/parameter				
	01 parameter reference violation	X	X	X	
10	Damage encountered				
	04 system object damage state				X
	44 partial system object damage				X
1C	Machine-dependent exception				
	03 machine storage limit exceeded				X
20	Machine support				
	02 machine check				X
	03 function check				X
22	Object access				
	01 object not found	X	X	X	
	02 object destroyed	X	X	X	
	03 object suspended	X	X	X	
24	Pointer specification				
	01 pointer does not exist	X	X	X	
	02 pointer type invalid	X	X	X	
2A	Program creation				
	06 invalid operand type	X	X	X	
	07 invalid operand attribute	X	X	X	
	08 invalid operand value range		X	X	
	0A invalid operand length	X	X	X	
	0C invalid operand odt reference	X	X	X	

Exception		Operands 1	2	3	Other
	0D reserved bits are not zero	X	X	X	X
2E	Resource control limit				
	01 user profile storage limit exceeded				X
32	Scalar specification				
	01 scalar type invalid	X	X	X	
	02 scalar attributes invalid	X	X	X	
	03 scalar value invalid	X	X	X	
36	Space management				
	01 space extension/truncation				X

17.4 Materialize Invocation Stack (MATINVS)

Op Code (Hex)	Operand 1	Operand 2
0546	Receiver	Process

Operand 1: Space pointer.

Operand 2: System pointer or null.

Description: This instruction materializes the current invocation stack within the specified process.

The attributes of the invocation entries currently on the invocation stack of the process specified by operand 2 are materialized into the template specified by operand 1.

Operand 2 is a system pointer or a null operand. If operand 2 is null, it indicates that the invocation stack of the current process is to be materialized. If operand 2 is not null, it is a system pointer identifying the process control space associated with the process for which the invocation stack is to be materialized. If the subject process, identified by operand 2, is different from the process executing this instruction, the executing process must be the original initiator of the subject process or must have process control special authorization to the process control space associated with the subject process.

Operand 1 is a space pointer that addresses a 16-byte aligned template into which is placed the materialized data. The format of the data is:

- Materialization size specification — Char(8)
  - Number of bytes provided for materialization — Bin(4)
  - Number of bytes available for materialization — Bin(4)
- Number of invocation entries — Bin(4)
- Mark counter — Bin(4)
- Invocation entries — Char(*)
  (An invocation entry is materialized for each of the invocations currently on the invocation stack of the specified process.)

The invocation entries materialized are each 128 bytes long and have the following format:

- Reserved — Char(32)
- Associated program pointer — System pointer
  (null for data base select/omit program or a destroyed program)
- Invocation number — Bin(2)
- Invocation type — Char(1)

  Hex 00 = Data base select/omit program

Hex 01 = Call external
Hex 02 = Transfer control
Hex 03 = Event handler
Hex 04 = External exception handler
Hex 05 = Initial program in process problem state
Hex 06 = Initial program in process initiation state
Hex 07 = Initial program in process termination state
Hex 08 = Invocation exit

- Reserved Char(1)
- Invocation mark Bin(4)
- Instruction number Bin(4)
- Reserved Char(68)

The number of invocations value specifies the number of invocation entries provided in the materialization.

The mark counter value represents the current value of a counter used by the machine to mark all activations and invocations created during the execution of a process with a unique value. This mark indicates the point at which the specific entry was allocated relative to the sequence of all activations and invocations that have been created over time within the process.

The associated program pointer is a system pointer that locates the program associated with the invocation entry.

The invocation number is a number that uniquely identifies each invocation in the PASA. When an invocation is allocated, the invocation number of the new invocation entry is one more than that in the calling invocation. The first invocation in the current process state has an invocation number of one.

The invocation type indicates how the associated program was invoked.

The invocation mark indicates the point at which this invocation entry was allocated relative to the sequence of all activations and invocations that have been created over time within the process. This is set from the incremented mark counter value for each new invocation added to the invocation stack.

The instruction number specifies the number of the instruction last being executed when the invocation passed control to the next invocation on the stack.

The fields labeled reserved are currently reserved for future use. These fields may be altered by this instruction depending upon the particular implementation of the machine. Any values set into these fields are meaningless.

The first 4 bytes of the materialization identifies the total quantity of bytes that may be used by the instruction. This value is supplied as input to the instruction and is not modified by the instruction. A value of less than 8 causes the materialization length exception to be signaled.

The second 4 bytes of the materialization identifies the total quantity of bytes available to be materialized. The instruction materializes as many bytes as can be contained in the area specified as the receiver. If the byte area iden-

tified by the receiver is greater than that required to contain the information requested, the excess bytes are unchanged.

No exceptions are signaled in the event that the receiver contains insufficient area for the materialization, other than the materialization length exception described previously.

When the materialization is performed for a process other than the one executing this instruction, the instruction attempts to interrogate, snapshot, the invocation stack of the other process concurrently with the ongoing execution of that process. In this case, the interrogating process and subject process may be interleaving usage of the processor resource. Due to this, the accuracy and integrity of the materialization is relative to the state, static or dynamic, of the invocation stack in the subject process over the time of the interrogation. If the invocation stack in the subject process is in a very static state, not changing over the period of interrogation, the materialization may represent a good approximation of a snapshot of its invocation stack. To the contrary, if the invocation stack in the subject process is in a very dynamic state, radically changing over the period of interrogation, the materialization is potentially totally inaccurate and may describe a sequence of invocations that was never an actual sequence that occurred within the process. In addition to the above exposures to inaccuracy in attempting to take the snapshot, the ongoing status of the invocation stack of the subject process may substantially differ from that reflected in the materialization, due to its continuing execution after completion of this instruction.

When the materialization is performed for the process executing this instruction, it does provide an accurate reflection of the status of the process' invocation stack. In this case, concurrent execution of this instruction with execution of other instructions in the process is precluded.

Authorization Required

- Process control special authorization
  - For materializing a different process than the one executing this instruction
- Retrieve
  - Contexts referenced for address resolution

Lock Enforcement

- Materialization
  - Contexts referenced for address resolution

Exceptions

Exception		Operands 1	Operands 2	Other
06	Addressing			
	01 space addressing violation	X	X	
	02 boundary alignment	X	X	
	03 range	X	X	
	06 optimized addressability invalid	X	X	

Exception		Operands 1	2	Other
08	Argument/parameter			
	01 parameter reference violation	X	X	
0A	Authorization			
	01 unauthorized for operation	X	X	
10	Damage encountered			
	04 system object damage state			X
	44 partial system object damage			X
1C	Machine-dependent exception			
	03 machine storage limit exceeded		X	
20	Machine support			
	02 machine check			X
	03 function check			X
22	Object access			
	01 object not found	X	X	
	02 object destroyed	X	X	
	03 object suspended	X	X	
24	Pointer specification			
	01 pointer does not exist	X	X	
	02 pointer type invalid	X	X	
28	Process state			
	02 process control space not associated with a process		X	
2A	Program creation			
	06 invalid operand type	X	X	
	07 invalid operand attribute	X	X	
	0C invalid operand odt reference	X	X	
	0D reserved bits are not zero	X	X	X
2E	Resource control limit			
	01 user profile storage limit exceeded			X
32	Scalar specification			
	01 scalar type invalid	X	X	
	02 scalar attributes invalid	X	X	
36	Space management			
	01 space extension/truncation			X
38	Template specification			
	03 materialization length	X		

17.5 Materialize Pointer (MATPTR)

Op Code (Hex)	Operand 1	Operand 2
0512	Receiver	Pointer

Operand 1: Space pointer.

Operand 2: System pointer, space pointer data object, data pointer, or instruction pointer.

Description: The materialized form of the pointer object referenced by operand 2 is placed in operand 1.

The first 4 bytes of the materialization identify the total number of bytes that may be used by the instruction. This value is supplied as input to the instruction and is not modified by the instruction. A value of less than 8 causes the materialization length exception to be signaled.

The second 4 bytes of the materialization identify the total number of bytes available to be materialized. The instruction materializes as many bytes as can be contained in the area specified as the receiver. If the byte area identified by the receiver is greater than that required to contain the information requested, then the excess bytes are unchanged. No exceptions (other than the materialization length exception) are signaled in the event that the receiver contains insufficient area for the materialization.

The format of the materialization is:

- Materialization size specification — Char(8)
  - — Number of bytes provided for materialization — Bin(4)
  - — Number of bytes available for materialization — Bin(4)
- Pointer type — Char(1)

  Hex 01 = System pointer
  Hex 02 = Space pointer
  Hex 03 = Data pointer
  Hex 04 = Instruction pointer

Pointer value materialization depends on the pointer type. One of the following pointer type formats is used.

- System pointer description — Char(66)

  The system pointer description identifies the object addressed by the pointer and the context which the object specifies as its addressing context.
  - — Context identification — Char(32)

    Context type — Char(1)
    Context subtype — Char(1)
    Context name — Char(30)
  - — Object identification — Char(32)

Object type	Char(1)
Object subtype	Char(1)
Object name	Char(30)

– Pointer authorization Char(2)

Object control	Bit 0
Object management	Bit 1
Authorization pointer	Bit 2
Space authority	Bit 3
Retrieve	Bit 4
Insert	Bit 5
Delete	Bit 6
Update	Bit 7
Ownership	Bit 8
Reserved (binary 0)	Bits 9-15

Note: If the object addressed by the system pointer specifies that it is not addressed by a context or if the context is destroyed, the context entry is hex 00. If the object is addressed by the machine context, a context type entry of hex 81 is returned. No verification is made that the specified context actually addresses the object.

The following lists the object type codes for system object references:

Value (Hex)	Object Type
01	Access group
02	Program
04	Context
06	Byte string space
07	Journal space
08	User profile
09	Journal port
0A	Queue
0B	Data space
0C	Data space index
0D	Cursor
0E	Index
0F	Commit block
10	Logical unit description
11	Network description
12	Controller description
14	Class of Service Description
15	Mode Description
19	Space
1A	Process control space
1B	Authorization List
1C	Dictionary

Note: Only the authority currently stored in the system pointer is materialized.

- Data pointer description Char(75)

The data pointer description describes the current scalar and array attributes and identifies the space addressability contained in the data pointer.

- Scalar and array attributes — Char(7)

  Scalar type — Char(1)

  Hex 00 = Signed binary
  Hex 01 = Floating-point
  Hex 02 = Zoned decimal
  Hex 03 = Packed decimal
  Hex 04 = Character
  Hex 06 = Onlyns
  Hex 07 = Onlys
  Hex 08 = Either
  Hex 09 = Open
  Hex 0A = Unsigned binary

  Scalar length — Char(2)

  If binary, character, floating-point, Onlyns, Onlys, Either, or Open:

  Length — Bits 0-15

  If zoned decimal or packed decimal:

  Fractional digits — Bits 0-7
  Total digits — Bits 8-15

  Reserved (binary 0) — Bin(4)

- Data pointer space addressability — Char(68)

  Context identification — Char(32)

  Context type — Char(1)
  Context subtype — Char(1)
  Context name — Char(30)

  Object identification — Char(32)

  Object type — Char(1)
  Object subtype — Char(1)
  Object name — Char(30)

  Offset into space — Bin(4)

  Note: If the object containing the space addressed by the data pointer is not addressed by a context, the context entry is hex 00. If the object is addressed by the machine context, a context type entry of hex 81 is returned.

  Support for usage of a Data Pointer describing an Onlyns, Onlys, Either, or Open scalar value is limited. For more information, refer to the Copy Extended Characters Left Adjusted With Pad, Set Data Pointer Attributes, and Create Cursor instructions.

- Space pointer description — Char(68)

  The space pointer description describes space addressability contained in the space pointer.

  - Context identification — Char(32)

    Context type — Char(1)
    Context subtype — Char(1)
    Context name — Char(30)

- Object identification — Char(32)
  - Object type — Char(1)
  - Object subtype — Char(1)
  - Object name — Char(30)
- Offset into space — Bin(4)

Note: If the object containing the space addressed by the space pointer is not addressed by a context, the context entry is hex 00. If the object is addressed by the machine context, a context type entry of hex 81 is returned.

- Instruction pointer description

  The instruction pointer description describes instruction addressability contained in the instruction pointer.

  - Context identification — Char(32)
    - Context type — Char(1)
    - Context subtype — Char(1)
    - Context name — Char(30)
  - Program identification — Char(32)
    - Program type — Char(1)
    - Program subtype — Char(1)
    - Program name — Char(30)
  - Instruction number — Bin(4)

If the program containing the instruction currently being addressed by the instruction pointer is not addressed by a context, the context entry is hex 00.

If the pointer is a system pointer or a data pointer and is initialized but unresolved, the pointer is resolved before the materialization occurs.

This instruction will tolerate a damaged object referenced by operand 2 when operand 2 is a resolved pointer. The instruction will not tolerate a damaged context(s) or damaged programs when resolving pointers. Also, as a result of damage or abnormal machine termination, this instruction can indicate that an object is addressed by a context, when in fact the context will not show this as an addressed object.

A space pointer machine object cannot be specified for operand 2.

Exceptions

Exception		Operands 1	Operands 2	Other
06	Addressing			
	01 space addressing violation	X	X	
	02 boundary alignment	X	X	
	03 range	X	X	
	04 external data object not found		X	
	06 optimized addressability invalid	X	X	

Exception		Operands 1	2	Other
08	Argument/parameter			
	01 parameter reference violation	X	X	
10	Damage encountered			
	04 system object damage state	X	X	X
	05 authority verification terminated due to damaged object			X
	44 partial system object damage	X	X	X
1C	Machine-dependent exception			
	03 machine storage limit exceeded			X
20	Machine support			
	02 machine check			X
	03 function check			X
22	Object access			
	01 object not found	X	X	
	02 object destroyed	X	X	
	03 object suspended	X	X	
	07 authority verification terminated due to destroyed object			X
24	Pointer specification			
	01 pointer does not exist	X	X	
	02 pointer type invalid	X	X	
2A	Program creation			
	06 invalid operand type	X	X	
	07 invalid operand attribute	X	X	
	08 invalid operand value range	X	X	
	0C invalid operand odt reference	X	X	
	0D reserved bits are not zero	X	X	X
2E	Resource control limit			
	01 user profile storage limit exceeded			X
32	Scalar specification			
	01 scalar type invalid	X		
36	Space management			
	01 space extension/truncation			X
38	Template specification			
	03 materialization length exception	X		

17.6 Materialize Pointer Locations (MATPTRL)

Op Code (Hex)	Operand 1	Operand 2	Operand 3
0513	Receiver	Source	Length

Operand 1: Space pointer.

Operand 2: Space pointer.

Operand 3: Binary scalar.

Description: This instruction finds the pointers in a subset of a space and produces a bit mapping of their relative locations.

The area addressed by the operand 2 space pointer is scanned for a length equal to that specified in operand 3. A bit in operand 1 is set for each 16 bytes of operand 2. The bit is set to binary 1 if a pointer exists in the operand 2 space, or the bit is set to binary 0 if no pointer exists in the operand 2 space.

Operand 1 is a space pointer addressing the receiver area. One bit of the receiver is used for each 16 bytes specified by operand 3. If operand 3 is not a 16-byte multiple, then the bit position in operand 1 that corresponds to the last (odd) bytes of operand 2 is set to 0. Bits are set from left to right (bit 0, bit 1,...) in operand 1 as 16-byte areas are interrogated from left to right in operand 2. The number of bits set in the receiver is always a multiple of 8. Those rightmost bits positions that do not have a corresponding area in operand 2 are set to 0.

The format of the operand 1 receiver is:

- Template size specification — Char(8)
  - Number of bytes provided for materialization — Bin(4)
  - Number of bytes available for materialization — Bin(4)
- Pointer locations — Char(*)

Operand 2 must address a 16-byte aligned area; otherwise, a boundary alignment exception is signaled. If the value specified by operand 3 is not positive, the scalar value invalid exception is signaled.

The first 4 bytes of the materialization identify the total number of bytes that may be used by the instruction. This value is supplied as input to the instruction and is not modified by the instruction. A value of less than 8 causes the materialization length exception to be signaled.

The second 4 bytes of the materialization identify the total number of bytes available to be materialized. The instruction materializes as many bytes as can be contained in the area specified as the receiver. If the byte area identified by the receiver is greater than that required to contain the information requested, then the excess bytes are unchanged. No exceptions (other than the materialization length exception) are signaled in the event that the receiver contains insufficient area for materialization.

Exceptions

Exception		Operands 1	2	3	Other
06	Addressing				
	01 space addressing violation	X	X	X	
	02 boundary alignment	X	X	X	
	03 range	X	X	X	
	06 optimized addressability invalid	X	X	X	
08	Argument/parameter				
	01 parameter reference violation	X	X	X	
10	Damage encountered				
	04 system object damage state	X	X	X	X
	44 partial system object damage	X	X	X	X
1C	Machine-dependent exception				
	03 machine storage limit exceeded				X
20	Machine support				
	02 machine check				X
	03 function check				X
22	Object access				
	01 object not found			X	
	02 object destroyed	X	X	X	
	03 object suspended	X	X	X	
24	Pointer specification				
	01 pointer does not exist	X	X	X	
	02 pointer type invalid	X	X	X	
2A	Program creation				
	06 invalid operand type	X	X	X	
	07 invalid operand attribute	X	X	X	
	08 invalid operand value range	X	X	X	
	0A invalid operand length			X	
	0C invalid operand odt reference	X	X	X	
	0D reserved bits are not zero	X	X	X	X
2E	Resource control limit				
	01 user profile storage limit exceeded				X
32	Scalar specification				
	03 scalar value invalid			X	
36	Space management				
	01 space extension/truncation				X

Exception		Operands 1	2	3	Other
38	Template specification				
	03 materialization length exception			X	

17.7 Materialize System Object (MATSOBJ)

Op Code (Hex)	Operand 1	Operand 2
053E	Receiver	Object

Operand 1: Space pointer.

Operand 2: System pointer.

Description: This instruction materializes the identity and size of a system object addressed by the system pointer identified by operand 2. It can be used whenever addressability to a system object is contained in a system pointer.

The first 4 bytes of the materialization identify the total number of bytes that may be caused by the instruction. This value is supplied as input to the instruction and is not modified by the instruction. A value of less than 8 raises the materialization length exception.

The second 4 bytes of the materialization identify the total number of bytes available to be materialized. The instruction materializes as many bytes as can be contained in the area specified as the receiver. If the byte area identified by the receiver is greater than that required to contain the information requested, then the excess bytes are unchanged. No exceptions (other than the materialization length exception) are signaled in the event that the receiver contains insufficient area for the materialization.

The format of the materialization is:

- Materialization size specification — Char(8)
  - Number of bytes provided for materialization — Bin(4)
  - Number of bytes available for materialization — Bin(4)
- Object state attributes — Char(2)
  - Suspended state — Bit 0

    0 = Not suspended
    1 = Suspended
  - Damage state — Bit 1

    0 = Not damaged
    1 = Damaged
  - Partial damage state — Bit 2

    0 = No partial damage
    1 = Partial damage
  - Existence of addressing context — Bit 3

    0 = Not addressed by a temporary context
    1 = Addressed by a temporary context
  - Dump for previous release permitted — Bit 4

0 = Dump for previous release not permitted.
1 = Dump for previous release permitted.

- Reserved (binary 0) — Bits 5-15

- Context identification — Char(32)
  - Context type — Char(1)
  - Control subtype — Char(1)
  - Context name — Char(30)
- Object identification — Char(32)
  - Object type — Char(1)
  - Object subtype — Char(1)
  - Object name — Char(30)
- Timestamp of creation — Char(8)
- Size of associated space — Bin(4)
- Object size — Bin(4)
- Owning user profile identification — Char(32)
  - User profile type — Char(1)
  - User profile subtype — Char(1)
  - User profile name — Char(30)
- Timestamp of last modification — Char(8)
- Recovery options — Char(4)
- Performance class — Char(4)
- Initial value of space — Char(1)
- Reserved — Char(3)
- Object Authorization List (AL) status — Bin(2)

  0 = object not in an AL
  1 = object in AL
- Authorization List identification — Char(48)
  - Authorization list (AL) status — Bin(2)

    0 = Valid AL
    1 = Damaged AL
    2 = Destroyed AL (no name below)
  - Reserved — Char(14)
  - Authorization list type — Char(1)
  - Authorization list subtype — Char(1)
  - Authorization list name — Char(30)
- Dump for previous release reason code — Bit(64)
- Maximum possible associated space size — Bin(4)
- Timestamp of last use of object — Char(8)

- Count of number of days object was used — Ubin(2)
- Object domain attributes — Char(2)
  - Program state provided — Bit 0

    0 = No program state value
    1 = Program state value present
  - Reserved (binary 0) — Bits 1-15
- Domain of object — Char(2)
- State for program — Char(2)
- Reserved — Char(124)

The timestamp field is materialized as an 8-byte unsigned binary number in which bit 41 is equal to 1024 microseconds. The timestamp of creation field is implicitly set when an object is created.

The timestamp of last modification field is explicitly set by the Modify System Object instruction. It is implicitly set, except for the objects restricted below, by any instruction or IMPL function that modifies or attempts to modify an object attribute value or an object state. The timestamp of last modification field is only ensured as part of the normal ensuring of objects.

Implicit setting of the timestamp of last modification field is restricted for the following objects and will only occur for generic, nonobject specific, operations on them such as Rename Object for example.

- Logical unit description
- Controller description
- Network description
- Access group
- Queue

No modification time stamp will be provided for the following objects and a value of zero will be returned in the materialization template for the modification time stamp

- Process control space

If the object addressed by the system pointer specifies that it is not addressed by a context or if the context is destroyed, the context type entry is hex 00. If the object is addressed by the machine context, a context type entry of hex 81 is returned. No verification is made that the specified context actually addresses the object.

If the object is a temporary object and is, therefore, owned by no user profile, the user profile type entry is assigned a value of hex 00. The object authorization list status field indicates whether or not the object is contained in an authorization list. If it is, the authorization list identification information provides the name of the authorization list, except when the authorization list is indicated as destroyed, in which case, the name information is meaningless.

This instruction will tolerate a damaged object referenced by operand 2 when operand 2 is a resolved pointer. The instruction will not tolerate a damaged context(s) or damaged programs when resolving pointers. Also, as a result of damage or abnormal machine termination, this instruction can indicate that an object is addressed by a context, when in fact the context will not show this as an addressed object. The Modify Addressability instruction can be used to correct this problem. The existence of addressing context attribute indicates whether the previously (or currently) addressing context was (is) temporary. This field is 0 if the object was (is) not addressed by a temporary context.

The Dump for Previous Release Permitted field will indicate if the object is eligible for a Request I/O instruction in which a dump for previous is requested. When this field indicates that the object is not eligible, the Dump for Previous Release Reason Code can be used to determine why the object is not eligible.

Currently reason codes are only architected for programs. The reason code structure for programs is mapped as follows. Note that more than one reason may be returned.

- Program dump for previous release reason code — Bit(64)
  - Language version and release reason — Bit 0

    0 = Language version and release is not a reason
    1 = Language version and release is one reason
  - Level of machine instructions used reason — Bit 1

    0 = The level of machine instructions used in the program is not a reason
    1 = Machine instructions not available in the previous release are used
  - Program observability reason — Bit 2

    0 = Lack of program observability is not a reason
    1 = Program is not observable and must be to be moved to previous release
  - Reserved — Bits 3-63

If the object has an associated space, the maximum possible associated space size field will be returned with a value which represents the maximum size to which the associated space can be extended. This value depends on the internal packaging of the object and its associated space as well as (possibly) the maximum space size field as optionally specified during the create of the object (or on the Create Duplicate Object instruction, if that is how the object was created).

The timestamp of last use of object field and the count of number of days object was used field are set by the Modify System Object instruction or by the Call External or Transfer Control instructions on the objects first use on that day. The timestamp value is only good for the date, the time value obtained from this timestamp is not accurate.

The domain of object field contains the value which represents which state the program which accesses this object must be running in.

The state for program field contains the state the program runs in. It is only present when the program state provided flag is on.

Valid object type fields and their meanings are:

Value (Hex)	Object Type
01	Access group
02	Program
04	Context
06	Byte string space
07	Journal space
08	User profile
09	Journal port
0A	Queue
0B	Data space
0C	Data space index
0D	Cursor
0E	Index
0F	Commit block
10	Logical unit description
11	Network description
12	Controller description
14	Class of Service Description
15	Mode Description
19	Space
1A	Process control space
1B	Authorization List
1C	Dictionary

Authorization Required

- Retrieve
  - Contexts referenced for address resolution

Lock Enforcement

- Materialize
  - Operand 2
  - Contexts referenced for address resolution

Exceptions

Exception		Operands 1	Operands 2	Other
06	Addressing			
	01 space addressing violation	X	X	
	02 boundary alignment	X	X	
	03 range	X	X	
	06 optimized addressability invalid	X	X	

Exception		Operands 1	2	Other
08	Argument/parameter			
	01 parameter reference violation	X	X	
0A	Authorization			
	01 unauthorized for operation		X	
10	Damage encountered			
	04 system object damage state	X	X	X
	05 authority verification terminated due to damaged object			X
	44 partial system object damage	X	X	X
1A	Lock state			
	01 invalid lock state		X	
1C	Machine-dependent exception			
	03 machine storage limit exceeded			X
20	Machine support			
	02 machine check			X
	03 function check			X
22	Object access			
	01 object not found	X	X	
	02 object destroyed	X	X	
	03 object suspended	X	X	
	07 authority verification terminated due to dastroyed object			X
24	Pointer specification			
	01 pointer does not exist	X	X	
	02 pointer type invalid	X	X	
2A	Program creation			
	06 invalid operand type	X	X	
	07 invalid operand attribute	X	X	
	08 invalid operand value range	X	X	
	0C invalid operand odt reference	X	X	
	0D reserved bits are not zero	X	X	X
2E	Resource control limit			
	01 user profile storage limit exceeded			X
32	Scalar specification			
	01 scalar type invalid	X	X	
36	Space management			
	01 space extension/truncation			X

Exception		Operands 1	2	Other
38	Template specification			
	03 materialization length exception	X		

Chapter 18. Machine Interface Support Functions Instructions

This chapter describes all instructions used for machine interface support functions. These instructions are arranged in alphabetic order. For an alphabetic summary of all the instructions, see Appendix A, "Instruction Summary."

18.1 Materialize Machine Attributes (MATMATR)

Op Code (Hex)	Operand 1	Operand 2
0636	Materialization	Machine attributes

Operand 1: Space pointer.

Operand 2: Character(2) scalar (fixed-length) or Space pointer.

Warning: The following information is subject to change from release to release. Use it with caution and be prepared to adjust for changes with each new release.

Description: The instruction makes available the unique values of machine attributes. The values of various machine attributes are placed in the receiver.

Operand 2 - specifies options for the type of information to be materialized. Operand 2 may be specified as either an attribute selection value (Character(2) scalar), or as an attribute selection template (Space Pointer).

When operand 2 is specified as an attribute selection value, the machine attributes are divided into nine groups. Byte 0 of the attribute selection operand specifies from which group the machine attributes are to be materialized. Byte 1 of the options operand selects a specific subset of that group of machine attributes.

When operand 2 is specified as an attribute selection template, particular machine attributes may be selected for materialization through specification of appropriate selection criteria. At this time, only selector value Hex 0000, materialize RCR (Resource Configuration Record) information, can be requested in the attribute selection template. The resource configuration record is an internal machine data structure which contains information on the configuration of I/O devices attached to the machine. The format of the attribute selection template is defined as follows:

- Selector Value (Only '0000'X, materialize RCR is allowed) — Char(2)
- Reserved — Char(14)
- Size of Selection Template — Bin(4)
- Materialization options — Char(2)
- Reserved — Char(10)

The size of selection template field must specify a value of 22 bytes or greater for the materialization options to apply. If a value of less than 22 is specified, then all of the RCR data is materialized.

The materialization options field allows for specification of various bit values which can be used to select the data to be materialized. The definition of the options which can be selected is provided in the following tables.

- Bit positions within the table specified as a 0 or a 1 must contain that value for the specific option.
- Bit positions specified as an 'x' may contain a value of 0 or 1 and provide additional selection criteria for that option.
- Bit positions specified as a letter other than 'r' may contain values specific to the option being selected and are defined under the applicable option.
- Bit positions specified as an 'r' indicate the bit position is reserved. These bit positions must contain a value of zero, otherwise a template value invalid exception is signaled.

Options	Description
rrrr r1rr aaaa aaaa	List names of objects active to an IOP. Where 'aaaa aaaa' specifies the Logical Bus Address associated with the IOP.
xxxx x0rr rrrr rxxx	RCR data

The following sub options may be specified in conjunction with the selection of RCR data. The 'R' values below indicates the value of 0, requesting RCR data, has been specified for bit 6.

Sub Option	Description
0xxx xRrr rrrr rrrr	All Buses
1xxx xRrr rrrr rnnn	Specific Bus Where 'nnn' = a binary value from 0 to 7 specifying the bus number being selected.
x0rr rRrr rrrr rxxx	All IOPs
x1ws cRrr rrrr rxxx	By IOP Type (1 or more of the following must be set) w=1 specifies Workstation IOP type s=1 specifies Storage IOP type c=1 specifies Communications IOP type

Operand 1 - specifies a space pointer to the area where the materialization is to be placed. The format of the materialization is as follows:

- Materialization size specification — Char(8)
  - Number of bytes provided for materialization — Bin(4)
  - Number of bytes available for materialization — Bin(4)
- Attribute specification (as defined by the attribute selection) — Char(*)

The first 4 bytes of the materialization (operand 1) identify the total number of bytes that can be used by the instruction. This value is supplied as input to the instruction and is not modified by the instruction. A value of less than 8 causes the materialization length exception to be signaled.

The second 4 bytes of the materialization identify the total number of bytes available to be materialized. The instruction materializes as many bytes as can be contained in the area specified as the receiver. If the byte are identified by the receiver is greater than that required to contain the information requested for materialization, then the a excess bytes are unchanged. No exceptions (other than the materialization length exception) are signaled in the event that the receiver contains insufficient area for the materialization.

The machine attributes selected by operand 2 are materialized according to the following selection values:

Selection Value	Attribute Description
Hex 0000	RCR information (only allowed in attribute selection template)

The format of the data materialized is dependent on the materialization option(s) selected in the attribute selection template.

When the list names of objects active to an IOP option is selected, the following format is used:

- Number of active object names returned — Bin(4)
- Total number of active object names of the specify IOP — Bin(4)
- List of all the active object names — Char(10*Y)

When the RCR data option is selected, the following format will be used:

- Number of entries returned by selection criteria — Bin(4)
- Total number of entries in RCR by selection criteria — Bin(4)
- RCR data — char(*)

  The RCR is materialized as a contiguous character string of binary data.

The following figure represents the base structure of the materialization of the attribute specification data.

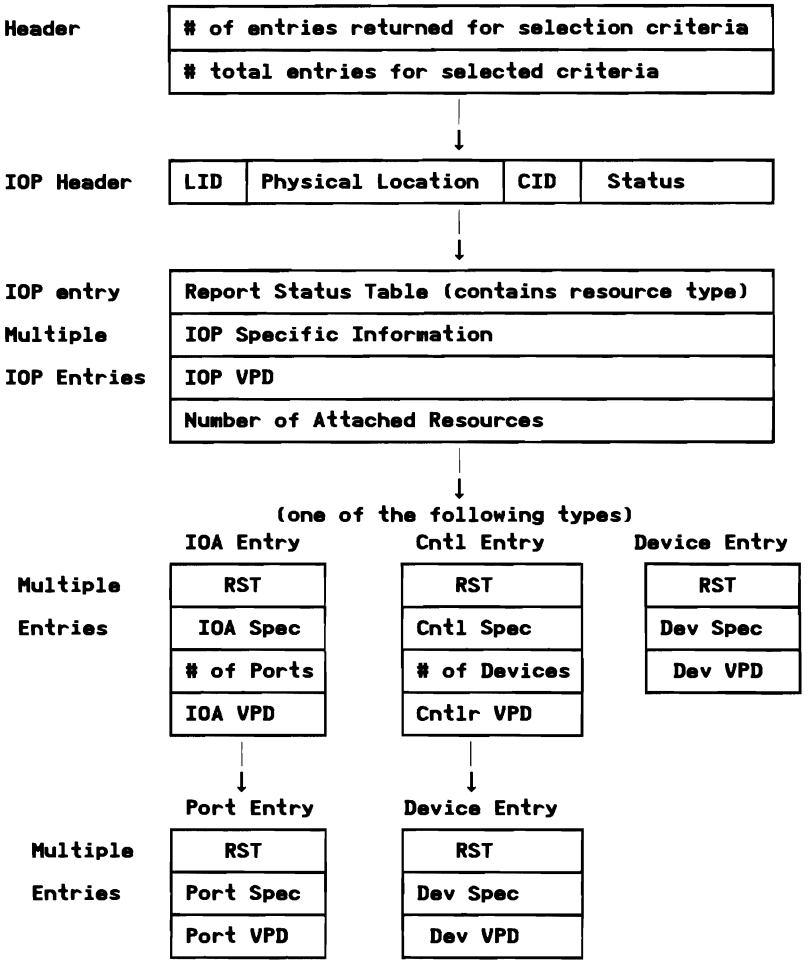


Figure 18-1. Materialization Format

The header portion contains the number of entries for the selection criteria and total entries for the selection criteria. If these two fields are not equal, then there was not enough space provided for a complete materialization.

Following the header will be the start of IOP specific data as well as the offset to the next IOP entry. There can be a total 96 IOP entries.

- Structure of IOP Entry

  The following list describes in a general form the structure of each IOP entry that will be materialized. The IOP entry is common for all lower levels whether they be for communications, storage or workstation device entries. If the IOP is describing communications or storage, the next lower level items will be for either a communications IOA or a storage controller. If the IOP is describing workstations, the next lower levels will be defining device entries.

  IOP entry data
  offset to next IOP entry
  offset to next lower level entry
  number of next lower level entries

The data following an IOP entry is dependent on the type of IOP materialized. It will be an IOA, controller or device for IOP types communications, storage or workstation respectively.

- Structure of IOA/Controller entry

  The following list describes in a general form the structure of each Communications IOA or storage device controller entry.

  IOA entry data
  offset to next IOA entry
  offset to port/Device entry
  number of ports/devices

If the following entry is IOA or controller, a port or device entry will follow.

- Structure of Port/Storage Device Entry

  The following list describes in a general form the structure of each Communications Port or storage device entry.

  Port/Device entry data
  offset to next port/Device entry

If the IOP type is workstation, a device entry will follow. The workstation IOP entry contains data used to identify the device entries.

- Structure of Device Entry

  The following list describes in a general form the structure of each device associated with a workstation IOP. Note the this structure is NOT preceeded by an IOA or Controller structure.

  Device entry data
  offset to next device entry

General RCR Structure

The following sections describe the structure of each of the various RCR structures. Note that each of these sections contain the data structure below and will be called out as 'STRUCTURE COMMON DATA'.

- RCR level id Char(16)
- Unit Address (always 'FFFF'X for IOP's) Char(2)
- Reserved Char(2)
- Resource ID Char(4)
- Resource Type Char(4)
- RCTT level Char(1)
- Model Char(3)
- Base LID Char(4)
- Reserved Char(4)
- Serial Number Char(4)
- Status Bytes Char(2)
- Reserved Char(2)

- Part Number — Char(12)
- Manufacturing info — Char(2)
- Manufacturing Plant number — Char(2)
- Reserved — Char(32)

IOP Entry

- Base Load ID — Char(4)
- Direct Select Address — Char(2)

  IOP Bus Number — Char(1)
  IOP Card Number — Bit(4)
  IOP Board Number — Bit(4)

- Logical Bus Address — Char(1)

  Bus Number — Bit(3)
  Bus Address — Bit(5)

- RAS Connection ID — Char(4)
- Status — Char(1)

  1xxx xxxx Operational
  0xxx xxxx Not operational
  x1xx xxxx Contains Load Source DASD
  x0xx xxxx Does not contain Load Source DASD

- IOP Resource Type — Char(4)
- Reserved — Char(8)
- Offset to next IOP — Bin(2)
- Offset to next lower level — Bin(2)
- Structure Common Data — Char(48)
- Reserved — Char(6)
- Object name for workstation CD — Char(10)
- Number of resources attached to IOP — Bin(2)
- Reserved — Char(8)
- IOP entry change flag — Char(1)
- IOP sub-type — Char(1)
- Reserved — Char(8)
- IOP VPD — Char(600)

IOA/Controller Entry

- Offset to next IOA/Controller entry — Bin(2)
- Offset to port/device entry — Bin(2)
- Structure Common Data — Char(48)
- IOA direct select address — Char(2)

  IOP Bus Number — Char(1)
  IOP Card Number — Bit(4)

IOP Board Number	Bit(4)
• Number of attached Ports/Devices	Bin(2)
• Reserved	Char(8)
• Second level entry change flag	Char(1)
• Level type	Char(2)
• Reserved	Char(14)
• Status	Char(2)
Operational Status	Char(1)
Reserved	Char(1)
• IOP VPD	Char(600)

Port Facility Entry

• Offset to next Port entry	Bin(2)
• Reserved	Bin(2)
• Structure Common Data	Char(48)
• Reserved	Char(6)
• Object name for ND	Char(10)
• Protocol name	Char(1)
• Reserved	Char(7)
• Cable attached off IOA	Char(2)
• Port entry change flag	Char(1)
• Reserved	Char(11)
• Port VPD	Char(600)

Device Entry

• Offset to next Device entry	Bin(2)
• Reserved	Bin(2)
• Structure Common Data	Char(48)
• Reserved	Char(6)
• Object name for the LUD	Char(10)
• IOP type	Char(1)
• Device entry change flag	Char(1)
• Reserved	Char(12)
• Device VPD	Char(600)

Hex 0004 Machine serial identification
(can only be materialized)
(only allowed in attribute selection value)

The machine serial identification that is materialized is an 8-byte character field that contains the unique machine identifier.

Hex 0100 Time-of-day clock
(can be materialized and modified)
(only allowed in attribute selection value)

The time-of-day clock provides a consistent measure of elapsed time. The maximum elapsed time the clock can indicate is approximately 143 years.

The time-of-day clock is a 64-bit unsigned binary counter with the following format:

0.....41 42 reserved 63

The bit positions of the clock are numbered from 0 to 63.

The clock is incremented by adding a 1 in bit position 41 every 1024 microseconds. Bit positions 42 through 63 are used by the machine and have no special meaning to the user. Note that these bits (42-63) may contain either binary 1's or binary 0's.

Unpredictable results occur if the time of day is materialized before it is set.

The maximum unsigned binary value that the time of day clock can be modified to contain is hex DFFFFFFFFFFFFFFF.

Hex 0104 Primary Initial process definition template
(can be materialized and modified)
(only allowed in attribute selection value)

The primary initial process definition template is used by the machine to perform an initial process load.

No check is made and no exception is signaled if the values in the template are invalid; however, the next initial process load will not be successful.

Hex 0108 Machine initialization status record
(only allowed in attribute selection value)

The entire template can be materialized but only specific fields in the template are modifiable.

The MISR (machine initialization status record) is used to report the status of the machine. The status is collected at IMPL (initial microprogram load) or IMPLA (initial microprogram load abbreviated).

The materialize format of the MISR is as follows:

- MISR status Char(8)
  - Restart IMPL Bit 0

    0 = IMPL was not initiated by the Terminate instruction
    1 = IMPL was initiated by the Terminate instruction
  - Manual power on Bit 1

    0 = Power on not due to Manual power on
    1 = Manual power on occurred
  - Timed power on Bit 2

    0 = Power on not due to Timed power on

1 = Timed power on occurred

- Remote power on Bit 3

  0 = Power on not due to Remote power on
  1 = Remote power on occurred

- Auto-power restart power on Bit 4

  0 = Power on not due to Auto-power restart power on
  1 = Auto-power restart power on occurred

- Uninterrupted power supply (UPS) battery low Bit 5

  0 = UPS battery not low
  1 = UPS battery low

- Uninterrupted power supply (UPS) bypass active Bit 6

  0 = UPS bypass not active
  1 = UPS bypass active

- Utility power failed, running on UPS Bit 7

  0 = Running on utility power
  1 = Running on UPS

- Uninterrupted power supply installed Bit 8

  0 = UPS not installed
  1 = UPS installed, ready for use

- Operation Panel battery failed Bit 9

  0 = Operation Panel battery good
  1 = Operation Panel battery failed

- Operation Panel self test failed Bit 10

  0 = Operation Panel self test successful
  1 = Operation Panel self test failed

- Console Status Bit 11

  0 = Console is operative
  1 = Console is inoperative

- Console State Bit 12

  0 = Console is not ready
  1 = Console is ready

- Reserved Bit 13

- Reserved Bit 14

- Primary console status Bit 15

  0 = Not using Primary console
  1 = Using Primary console

- Reserved Bit 16

- ASCII console status Bit 17

0 = Not using ASCII console
1 = Using ASCII console

- Termination status Bit 18

  0 = Normal (TERMMPR)
  1 = Abnormal

- Duplicate user profile (AIPL only) Bit 19

  0 = Not duplicate, new user profile created
  1 = Duplicate found and used by AIPL

- Damaged user profile (AIPL only) Bit 20

  0 = Not damaged, user profile used
  1 = Damaged user profile, profile deleted and recreated

- Damaged machine context Bit 21

  0 = Not damaged
  1 = Machine context damaged

- Object recovery list status Bit 22

  0 = Complete
  1 = Incomplete

- Recovery phase completion Bit 23

  0 = Complete
  1 = Incomplete

- Most recent machine termination Bit 24

  0 = Objects ensured
  1 = Object(s) not ensured at most recent machine termination

- Last MISR reset Bit 25

  0 = Object(s) ensured on every machine termination
  1 = Object(s) not ensured on every machine termination since last MISR reset

- Reserved Bit 26-27

- IPL Mode (can be materialized and modified) Bit 28-29

  00 = DST and BOSS in unattended mode
  10 = DST and BOSS is attended mode

- Service Processor power on Bit 30

  0 = Not first service processor power on
  1 = First service processor power on

- MISR damage Bit 31

  0 = MISR not damaged
  1 = MISR damaged, information reset to default values

- Auto keylock position Bit 32

0 = Keylock not in auto position
1 = Keylock in auto position

- Normal keylock position Bit 33

  0 = Keylock not in normal position
  1 = Keylock in normal position

- Manual keylock position Bit 34

  0 = Keylock not in manual position
  1 = Keylock in manual position

- Secure keylock position Bit 35

  0 = Keylock not in secure position
  1 = Keylock in secure position

- Tower two presence on 9404 system unit Bit 36

  0 = Tower two not present
  1 = Tower two present

- Battery status for tower one on 9404 system unit Bit 37

  0 = Battery good for tower one
  1 = Battery low for tower one

- Battery status for tower two on 9404 Bit 38

  0 = Battery good for tower two
  1 = Battery low for tower two

- Termination due to utility power failure and user specified delay time exceeded Bit 39

  0 = Delay time not exceeded
  1 = Utility failure and delay time exceeded

- Termination due to utility power failure and battery low Bit 40

  0 = Battery not low
  1 = Utility failure and battery low

- Termination due to forced microcode completion Bit 41

  0 = Not forced microcode completion
  1 = Termination due to forced microcode completion

- Auto power restart disabled due to utility failure Bit 42

  0 = Auto power restart not disabled
  1 = Auto power restart disabled

- Reset utility power bits (valid only on modify) Bit 43

  0 = Do not reset utility power bits
  1 = Reset utility power bits

- Spread the Operating System Bit 44
  (can be materialized and modified)

  0 = Do not spread the Operating System
  1 = Spread the Operating System

- Install from Disk/Tape Bit 45
  (can be materialized and modified)

  0 = Install from tape
  1 = Install from disk

- Use Primary/Alternate PDT Bit 46
  (can be materialized and modified)

  0 = Use Primary Process Definition Template
  1 = Use Alternate Process Definition Template

- Time/Date source Bit 47

  0 = Time/Date is accurate
  1 = Time/Date default value used

- Install Type Bin(2)

  0 = Normal IPL
  1 = Manual Install
  2 = Automated Install

- Number of damaged main storage units Bin(2)
- National language index Bin(2)
  (Can be materialized and modified)
- Number of entries in object Bin(4) recovery list
- Tape sequence number for an AIPL Bin(4)
- Tape volume number for an AIPL Bin(4)
- Address of object recovery list Space pointer
- Process control space created as the result of IPL or AIPL System pointer
- Process static storage area space System pointer
- Process automatic storage System pointer area space
- Console Information list Char(400)
  (Can be materialized and modified)
  (Array of five entries each 80 bytes in size)
  (1st = Primary, 2-5 = Reserved)

  Console entry Char(80)
  Display LUD System pointer
  Display CD System pointer
  Controller model Char(4)
  Controller type Char(4)
  Controller serial number Char(4)
  Controller object data Char(12)
  Direct select address Char(2)
  IOP bus number Bit(8)
  IOP card, Bit(8) board structure

IOP card	Bit(4) number
IOP board	Bit(4) number
Logical bus address	Char(1)
IOP unit address	Char(2)
Resource Identifier	Char(4)
Reserved	Char(3)
Work station object data	Char(12)
Direct select address	Char(2)
IOP bus number	Bit(8)
IOP card board structure	Bit(8)
IOP card number	Bit(4)
IOP board number	Bit(4)
Logical bus address	Char(1)
Device unit address	Char(2)
Port	Char(1)
Switch Setting	Char(1)
Reserved	Char(7)
Device type	Char(4)
Device model	Char(4)
Information valid in entry	Bit(1)
Reserved	Bit(7)
Console keyboard type	Char(1)
Console extended keyboard type	Char(1)
Reserved	Char(1)

- Load/Dump Tape device information list — Char(96)
  (Can be materialized and modified)
  (Array of two entries each 48 bytes in size)

  (1st = LUD information, 2nd = CD information)

Load/Dump tape device entry	Char(48)
Reserved	Char(16)
LUD/CD information	Char(12)
Direct select address	Char(2)
IOP bus number	Bit(8)
IOP card board structure	Bit(8)
IOP card number	Bit(4)
IOP board number	Bit(4)
Logical bus address	Char(1)
Device unit address	Char(2)
Information valid in entry	Bit(1)
Reserved	Bit(7)
Reserved	Char(6)
Device type	Char(4)
Device model	Char(4)

Reserved	Char(12)	
• Recovery object list	Char(*)	(located by recovery object list pointer)
Recovery entry	Char(32)	(repeated for number of entries)
Object pointer	System pointer	
Object type	Char(1)	
Object status	Char(15)	

Restart IMPL indicates that a Terminate Machine Processing instruction was issued with the restart option set to yes. The machine performed an IMPL without powering down the machine.

Manual power on indicates the power switch on the operation panel was pressed to power the system on.

Timed power on indicates the system was powered on using the system value specified by the customer. This option will only be honored when the Timed power on function is enabled.

Remote power on indicates the system was power on by a phone call placed by the customer. This option will only be honored when the Remote power on function is enabled.

Auto-power restart indicates the system was automatically powered on after a utility failure occurred and power was restored. This option will only be honored when the Auto-power restart function is enabled.

UPS battery low indicates that a UPS battery is installed on the system and the battery is low.

UPS bypass active indicates that the UPS has been bypassed. If a utility power failure occurs, the UPS will not supply power.

UPS power failed indicates that a utility failure has occurred and the system is currently running on battery power.

UPS installed indicates that a Uninterrupted Power Supply is installed on the system and is available for use should the power fail.

Operation Panel battery failure indicates the battery in the operation panel has failed and the system will not be able to determine the correct time and date upon the next IMPL. An approximate time and date will be given to the customer for verification.

Operation Panel self test failed indicates the Operation Panel is possibly bad and some function concerning the operation panel may not work correctly.

Console status indicates whether the selected console is functioning normally or is inoperative.

Console state indicates whether the selected console is ready to be used.

Primary console status is set when the customer selected primary console is being used as the system console.

ASCII console status is set when a ASCII console is being used as the system console.

Termination status indicates how the previous IMPL was terminated. If normal, the Terminate Machine Processing instruction successfully terminated the previous IMPL. If abnormal, the Terminate Machine Processing instruction did not successfully terminate the previous IMPL. This also implies that some cleanup of permanent objects may be required by the user.

The duplicate user profile is valid only for AIPL and indicates if a user profile that is the same as the AIPL user profile to be created already exists in the machine context. The machine in this instance does not create the user profile for AIPL but rather uses the one located with the same name.

Damaged AIPL user profile indicates if the currently existing user profile was detected as damaged and a new user profile was created as specified in the AIPL user profile creation template.

Damaged machine context indicates if damage was detected in the machine context when an attempt was made to locate the duplicate user profile or to insert addressability to a newly created user profile. In either case, all current addressability is removed from the machine context, the new AIPL user profile is created, its addressability is inserted into the machine context, and the AIPL continues. Objects whose addressability was removed may have it reinserted using the Reclaim instruction for all objects or the Modify Addressability instruction for a specific object.

The object recovery list status entry indicates that the status is complete unless one of the following conditions is true:

- The recovery list was lost.
- More objects were to be placed in the list but there was insufficient space.

The recovery phase completion entry indicates that the status is complete unless one of the following conditions occurs:

- An object to be recovered and/or inserted into the object recovery list no longer exists.
- The objects to be recovered could not be determined due to loss of internal machine indicators that specified which objects were in use at machine termination.

The most recent machine termination entry is set to 0 unless all objects were not ensured at the most recent machine termination.

The last MISR reset entry is set to 0 if all objects were ensured at every machine termination since the MISR was last reset (to 0) using the Modify Machine Attributes instruction.

IPL mode indicates which mode DST and BOSS will be IPL'ed. Either both will be attended or both will be unattended.

Service Processor power on indicates if this is the first time the service processor card has been powered on.

MISR Damage indicates if the microcode detected that the MISR was damaged and it's contents has to be reset to the default system values.

Auto keylock position indicates if the keylock was is the auto position on the operation panel on the most recent IMPL.

Normal keylock position indicates if the keylock was is the normal position on the operation panel on the most recent IMPL.

Manual keylock position indicates if the keylock was is the manual position on the operation panel on the most recent IMPL.

Secure keylock position indicates if the keylock was is the secure position on the operation panel on the most recent IMPL.

Tower two present on 9404 system unit indicates if the system has second tower when the system is a 9404 system unit.

Battery status for tower one on 9404 system unit indicates if a UPS battery is installed on the first tower of a 9404 system unit, the battery is low.

Battery status for tower two on 9404 system unit indicates if a UPS battery is installed on the second tower of a 9404 system unit, the battery is low.

Termination due to utility power failure and user specified delay time exceeded indicates the last termination of the system was due to a utility power failure and the system value specified by the delay time had elapsed so the system was terminated.

Termination due to utility power failure and battery low indicates the last termination of the system was due to a utility power failure and while running on battery power the voltage dropped below a level to continue to power the system so the system was terminated.

Termination due to forced microcode completion indicates that the system when down by the user selecting power down from DST or the delayed power off switch was pressed on the operation panel.

Auto power restart disabled due to a utility failure indicates the microcode disabled the auto power restart option when a condition was detected that would prevent the auto power restart to function properly.

Reset utility power bits indicates that the power bits should be reset. This bit is only looked at when modifying the MISR.

Spread/No Spread indicates to spread the operating system on the next install instead of overlaying the existing objects. This bit is set to spread after a new dasd has been added.

Install from Disk/Tape indicates when performing an install to use the initial OS/400 install program off of disk or to load the initial OS/400 install program off of tape.

Primary/Alternate Process Definition Template indicates on IPL to initiate the initial OS/400 process using the Primary or the Alternate Process Definition Template.

Time/Date source informs OS/400 if VLIC was able to determine the correct time/date or if it was forced to use the default time/date.

Install type is set indicate whether an IPL or install was performed and if an install was performed, what type of install occurred.

The number of damaged main storage transfer blocks entry indicates the number of main storage transfer blocks that were detected as damaged by the machine during IMPL.

National language index is the value used to index to the the National language array kept by the system.

The number of entries in the object recovery list entry indicates how many objects are listed in the space located by the address of object recovery list entry.

The tape sequence number is set by the microcode to allow BOSS to perform their install.

The tape volume number is set by the microcode to allow BOSS to perform their install.

The address of object recovery list entry contains a space pointer to the list of the potentially damaged objects that were identified during machine initialization. The machine maintains this list of objects until a Modify Machine Attribute instruction for the MISR is executed. The number of such objects is indicated by the number of entries in the object recovery list entry.

The process control space created results from IPL or AIPL and is identified by a system pointer returned in this field.

Process static storage space system pointer addresses the space object that contains the PSSA created and initialized at IPL time. The space containing the PSSA is a temporary space and is not addressed by a context. This field contains binary 0's if the machine to programming transition is done via an IPL.

Process automatic storage area system pointer addresses the space object that contains the PASA created and initialized at IPL time. The space containing the PASA is a temporary space and is not addressed by any context. This field contains binary 0's if the machine to programming transition is done via an IPL.

The console information list contains information for each console device as obtained from the Resource Configuration Record or set by the customer.

The Load/Dump tape device information is information needed to build a LUD and CD for the device used to install BOSS.

The recovery object list identifies objects that required some activity performed on the object(s) during IPL. The list is located by the recovery object list pointer.

Each entry in the list has the following general format:

- Object — System pointer
- Object type — Char(1)

- Object status Char(15)
  - General status Char(2)

    Damaged Bit 0

    0 = Object not damaged
    1 = Object damaged

    Reserved Bit 1

    Suspended Bit 2

    0 = Object not suspended
    1 = Object suspended

    Partially damaged Bit 3

    0 = Object not partially damaged
    1 = Object partially damaged

    Journal synchronization Bit 4

    0 = Synchronization complete or not necessary
    1 = Synchronization failure

    Reserved Bits 5-6

    IPL detected damage Bit 7

    0 = Any indicated damage was not detected by directory recovery
    1 = Indicated damage was detected by directory recovery

    Reserved Bits 8-15
  - Object specific status Char(13)

    (The format for the IPL recovery status for this portion of the object recovery list entries is different for each object type. A description of each follows by object type.)
  - Commit block status Char(2)

    Decommit Bit 0

    0 = The journal has successfully been read backwards until either a start commit or a decommit entry was found. An attempt has been made to decommit all the data base changes but the attempt may not have been successful if the data space is damaged or if the function check flag is on.
    1 = The journal has not successfully been read backwards to a start commit or decommit entry and all the changes have not been decommitted.

    Journal read errors Bit 1

    0 = No journal read errors
    1 = Journal read errors occurred during decommit

    Journal write errors Bit 2

    0 = No journal write errors
    1 = Journal write errors occurred during decommit

Partial damage to data space

0 = No partial damage encountered
1 = Partial damage encountered on 1 or more data spaces

Damage to data space Bit 4

0 = No damage encountered

1 = Damage encountered on 1 or more data spaces

Function check Bit 5

0= No function check encountered
1= Function check encountered

Reserved Bit 6

Data space during IMPL Bit 7

0= Data space is synchronized with the journal
1= Data space is not synchronized with the journal. All changes may not be decommitted.

Decommit reason code Bits 8-10

000 = Decommit not performed
001 = Decommit at IPL
010 = Process termination
100 = Decommit instruction (all other values reserved)

Reserved Bits 11-15

Reserved (binary 0) Char(7)

Start commit journal Bin(4)

Sequence number

- Data space
  - Status Char(13)

    Indexes detached from data space Bit 0

    0 = Indexes remain attached
    1 = All indexes detached from this data space

    Reserved (binary 0) Bits 1-15
  - Reserved (binary 0) Char(7)
  - Ordinal entry number of last entry Bin(4)
- Data space index
  - Status Char(13)

    Invalidated Bit 0

    0 = Not invalidated
    1 = Invalidated

    Recovered by journal Bit 1

0 = Not recovered
1 = Recovered

Reserved (binary 0) Bits 2-15

- Reserved (binary 0) Char(11)

- Journal port
  - Status Char(13)

    Synchronization status Bit 0

    0 = All objects synchronized
    1 = Not all objects synchronized

    Reserved Bits 1-7
  - Reserved Char(10)
  - Number of journal spaces attached Bin(2)
- Journal space
  - Status Char(13)

    Journal space usable Bit 0

    0 = Journal space is usable
    1 = Journal space is not usable

    Threshold reached Bit 1

    0= Threshold has not been reached
    1= Threshold has been reached

    Reserved Bits 2-7
  - Reserved Char(4)
  - First journal sequence number Bin(4)
  - Last journal sequence number Bin(4)
  - Reserved (binary 0) Char(13)

All objects-Any damage detected during IPL is reported in the general status information. If this damage is detected as a result of special processing performed during directory rebuild, it is indicated in the IPL detected damage bit. A journal synchronization failure indicates the designated object was not made current with respect to the journal. Subsequent attempts to apply journal changes from the journal to this object will not be allowed.

Commit block-All commit blocks that were attached to an active process during the previous IPL are interrogated at the following IPL. The system attempts to decommit any uncommitted changes referenced through these commit blocks. The results of this attempted decommit is reported in the status field. The system also returns the journal entry sequence number of the start commit journal entry (hex 0500) last created for this commit block if there were any uncommitted changes. If the number is not returned, a value of binary zero is returned.

Data space-If object damage was detected during IPL, the object is marked as damaged, damage is indicated in the object status field, and an event is signaled. In this case, the highest ordinal entry number is 0. In certain situations, the data space indexes over the data space become detached and therefore must be recreated. If the object is not damaged, the data space is usable and the highest ordinal entry number is set. The ordinal entry number of last entry indicates the last entry in the data space. Updates are not guaranteed. Updates may be out of sequence or partially applied and must be verified by the user for correctness.

Data space index-If object damage was detected during IPL, the object is marked as damaged, damage is indicated in the object status field, and an event is signaled. If the object was invalidated because changes were made in a data space addressed by the data space index, the data space index is included in the list and marked as invalidated. The associated data space is also included elsewhere in the object recovery list. Only damaged or invalidated data space indexes are included in the list.

Journal port-Each journal port in the system is interrogated at IPL. The status field contains the result of this checking and also the result of the attempt to synchronize the objects (if necessary) being journaled through the indicated journal port. The system also returns the number of journal spaces attached to the journal port after IPL is complete.

Journal space-Each journal space that was attached to a journal port or used by the system to synchronize an object which was being journaled at the time of the previous machine termination is interrogated during IPL. The status field reports the results of this interrogation and synchronization use. Journal spaces are only referenced by the object recovery list if this IPL was preceded by an abnormal failure or some unexpected condition was discovered during the IPL. The first journal sequence number on the journal space is returned. The last usable entry on the journal space is also identified. If the journal space is damaged, these fields will contain zeroes.

Hex 0118 Uninterruptible power supply delay time and calculated delay time. (only allowed in attribute selection value)

The UPS delay time can be materialized and modified. The UPS calculated delay time can only be materialized. Note: The UPS delay time is meaningful only if a UPS is installed.

The delay time interval is the amount of time the system waits for the return of utility power. If a utility power failure occurs, the system will continue operating on the UPS supplied power. If utility power does not return within the user specified delay time, the system will perform a quick power down. The delay time interval is set by the customer. The calculated delay time is determined by the amount of main storage and DASD that exists on the system. Both values are in seconds.

The format of the template for the uninterruptible power supply delay time (including the 8-byte prefix) is as follows:

- Number of bytes available Bin(4)

- Number of bytes provided Bin(4)
- UPS Delay time Bin(4)
- Calculated UPS Delay time Bin(4)

Hex 012C Vital Product Data
(can only be materialized)
(only allowed in attribute selection value)

The VPD (vital product data) is a template that contains information for memory card VPD, Customer Card Identification Number (CCIN) values for the memory card, non-memory card VPD, Central Electronics Complex (CEC) VPD and the minimum memory fields.

The materialize format of the VPD is as follows:

- IMPI VPD Char(456)
  - Main Store Memory VPD Char(64)
    (An array of eight entries each 8 bytes in size with an entry existing for each main store card on the system up to a maximum of eight cards.)

      Main Store card entry Char(8)
        Main Store VPD Status Bit(8)
          Card not usable Bit(1)
          Card has failed Bit(1)
          Reserved Bit(6)
        Reserved Char(7)
  - CCIN size list Char(8)
    (An array of eight entries each 1 byte in size that contains the number of CCIN numbers associated with each memory card)
  - CCIN value list Char(256)
    (An eight by eight matrix with each element 4 bytes in size containing a CCIN number associated with a memory card)
  - Non-Memory VPD Char(128)
    (An array of four entries each 32 bytes in size)

      Non-memory card entry Char(32)
        Load Identifier Char(4)
        CCIN number Char(4)
        Model Number Char(4)
        Part Number Char(12)
        Serial Number Char(4)
        Plant of manufacture Char(4)
- CEC VPD Char(70)

  Reserved Char(8)

  Service processor part number Char(8)

  Service processor unique identification Char(4)

Service processor load identification	Char(4)
Reserved	Char(4)
System plant of manufacture	Char(4)
System serial number	Char(4)
Reserved	Char(13)
System type	Char(4)
System model number	Char(4)
Processor plant of manufacture	Char(4)
Processor serial number	Char(4)
Reserved	Char(5)

- Minimum Memory Fields Char(4)

Minimum memory required	Bin(2)
Minimum memory available	Bin(2)

Main store memory VPD is an array that contains eight entries. The first entry in the array corresponds to the memory card in the first physical card slot, the second entry corresponds to the memory card in the second physical card slot etc. for a maximum of eight memory cards. The main store card VPD status should be interpreted by taking Card not useable together with Card has failed. The two bits together should be interpreted as follows:

- Card not useable = 0 and Card failed = 0

  Use the main store entry, no failures detected

- Card not useable = 0 and Card failed = 1

  Use the main store entry but the card contains bad frames

- Card not useable = 1 and Card failed = 0

  Do not use the main store entry, card does not exist

- Card not useable = 1 and Card failed = 1

  Do not use the main store entry, card exists but is bad

CCIN size list is an array with the first entry corresponding to the first main store memory VPD entry. The value in this entry determines the number of CCIN values that exist for the first main store card. This value determines the number of entries in the CCIN value list matrix that exist for the first card. The rest of the entries in the CCIN size list correspond just like the first entry did.

CCIN value list is a matrix that holds the CCIN values for each main store card. The first subscript determines which main store card the information corresponds to and the second subscript determines which CCIN number we are dealing with for that card. We can have up to eight memory cards with up to eight CCIN values for each memory card. Note that the CCIN size list determines how many CCIN values exist for each card.

Non-memory VPD is an array with four entries. The first entry in the array contains the information for the processor card, the second entry contains the information for the SP/SBA card, the third entry contains the information for additional BCU cards and the fourth entry is reserved.

Minimum memory fields contain the amount of memory required for the system to run at optimum performance and the amount of memory that is actually available on the system. These values are in megabytes of main storage.

Hex 0130 Network Attributes
(can be materialized and modified)
(only allowed in attribute selection value)

The Network Attributes is a template that contains information concerning APPN network attributes.

The materialize format of the Network attributes is as follows:

- Network Data — Char(190)

System name	Char(8)
System name length	Bin(2)
New System name	Char(8)
New System name length	Bin(2)
Local system network identification	Char(8)
Local system network identification length	Bin(2)
Reserved	Char(10)
Local system control point name	Char(8)
Local system control point name length	Bin(2)
Reserved	Char(10)
Default local location name	Char(8)
Default local location name length	Bin(2)
Default mode name	Char(8)
Default mode name length	Bin(2)
Maximum number of intermediate sessions	Bin(2)
Maximum number of conversations per APPN LUD	Bin(2)
Local system node type	Bit(8)
Reserved	Bit(8)
Route addition resistance	Bin(2)
List of network server network ID's	Char(40)
(An array of five entries each 8 bytes in size)	
List of network server network ID lengths	Char(10)
(An array of five entries each 2 bytes in size)	
List of network server control point names	Char(40)
(An array of five entries each 8 bytes in size)	
List of network server control point name lengths	Char(10)
(An array of five entries each 2 bytes in size)	
Alert Flags	Char(1)
Alert priority focal point	Bit(1)

Alert default focal point	Bit(1)
Reserved	Bit(6)
Network Attribute Flags (Materializable only)	Char(1)
Network attributes initialized	Bit(1)
Pending system name made current system name	Bit(1)
Reserved	Bit(6)

The machine system name is defaulted to the system serial number with a 'S' in the first position. Thereafter, it may be modified to any value of 1 through 8 characters with the first character alphabetic.

The machine system name length is kept to determine how long the system name is. The default value for the length is eight.

The new system name is a tentative new value chosen for the machine system name. This value will become the machine system name at the next IPL. The initial value is null and the syntax rules are the same as those for the machine system name.

The new system name length is kept to determine how long the new system name is. The default value for the length is zero.

The local system network identification default is 'APPN' and the default local system network identification length is four.

The local system control point name default is the system serial number with a character 'S' in the first position and the default control point name length is eight.

The local location name default is the system serial number with a character 'S' in the first position and the default local location name length is eight.

The mode name default is all blanks and the default mode length is eight.

The maximum number of intermediate sessions default is 200

The maximum number of conversations per APPN LUD is 64

The local system node type default is '01' hex

The route addition resistance default is 128

All entries of the network server network ID's are defaulted to blanks with all entries of the network server network ID's lengths defaulting to zero.

All entries of the network server control point names are defaulted to blanks with all entries of the network server control point name lengths defaulting to zero.

Hex 0134 Date Format
(can be materialized and modified).
(only allowed in attribute selection value)

The date format is the format is which the date will be presented to the customer. The possible values are YMD, MDY, DMY, JUL where Y = Year, M = Month, D = Day and JUL = Julian.

The format of the template for date format is as follows:

- Number of bytes available Bin(4)
- Number of bytes provided Bin(4)
- Date Format Char(3)

Hex 0138 Leap Year Adjustment
(can be materialized and modified).
(only allowed in attribute selection value)

The leap year adjustment is added to the leap year calculations to determine the year in which the leap should occur. The valid values are 0, 1, 2, 3.

The format of the template for leap year adjustment is as follows:

- Number of bytes available Bin(4)
- Number of bytes provided Bin(4)
- Leap year adjustment Bin(2)

Hex 013C Timed Power On
(can be materialized and modified).
(only allowed in attribute selection value)

The timed power on is the time and date at which the system should automatically power on if it is not already powered on.

The format of the template for timed power on is as follows:

- Number of bytes available Bin(4)
- Number of bytes provided Bin(4)
- Minute Bin(2)
- Hour Bin(2)
- Day Bin(2)
- Month Bin(2)
- Year Bin(2)

Hex 0140 Timed Power On Enable/Disable
(can be materialized and modified).
(only allowed in attribute selection value)

The timed power on enable/disable allows the timed power on function to be queried to determined if the function is enabled or disabled.

The format of the template for timed power on enable/disable is as follows:

- Number of bytes available Bin(4)

- Number of bytes provided Bin(4)
- Enable/Disable Bin(2)

  HEX 8000-indicates timed power on is enabled

  HEX 0000-indicates timed power on is disabled

Hex 0144 Remote Power On Enable/Disable
(can be materialized and modified).
(only allowed in attribute selection value)

The remote power on enable/disable allows the remote power on function to be queried to determined if the function is enabled or disabled.

The format of the template for remote power on enable/disable is as follows:

- Number of bytes available Bin(4)
- Number of bytes provided Bin(4)
- Enable/Disable Bin(2)

  HEX 8000-indicates remote power on is enabled

  HEX 0000-indicates remote power on is disabled

Hex 0148 Auto power restart Enable/Disable
(can be materialized and modified).
(only allowed in attribute selection value)

The auto power restart enable/disable allows the auto power restart function to be queried to determined if the function is enabled or disabled.

The format of the template for auto power restart enable/disable is as follows:

- Number of bytes available Bin(4)
- Number of bytes provided Bin(4)
- Enable/Disable Bin(2)

  HEX 8000-indicates auto power restart is enabled

  HEX 0000-indicates auto power restart is disabled

Hex 014C Date separator
(can be materialized and modified).
(only allowed in attribute selection value)

The date separator is used when the date is presented to the customer. The valid values are a slash(/), dash(-), period(.) and a comma(,).

The format of the template for the date separator is as follows:

- Number of bytes available Bin(4)
- Number of bytes provided Bin(4)
- Date Separator Char(1)

Hex 0164 Uninterruptible power supply type
(can be materialized and modified).
(only allowed in attribute selection value)

Note: The UPS type is meaningful only if a UPS is installed.

The uninterruptible power supply type option allows BOSS to tell the microcode how much of the system is powered by a UPS (ie, what type of UPS is installed). A full UPS will power all racks in the system. A mini UPS will power the racks containing the CEC and the load source.

The format of the template for UPS Type is as follows (including the usual 8-byte prefix):

- Number of bytes available Bin(4)
- Number of bytes provided Bin(4)
- UPS Type Bin(2)

  HEX 0000-indicates a full UPS is installed (all racks have a UPS installed)

  HEX 8000-indicates a mini UPS is installed (only the minimum number of racks are powered)

Hex 0168 Panel Status Request
(can be materialized and modified).
(only allowed in attribute selection value)

The Panel Status Request option allows BOSS to determine what current status of the operations panel.

The entire template can be materialized but only specific fields in the template are modifiable.

The format of the template for Panel Status Request is as follows (including the usual 8-byte prefix):

- Number of bytes available Bin(4)
- Number of bytes provided Bin(4)
- Current IPL Type Char(1)
  (Can be materialized and modified)
- Panel Status Char(2)
  - Uninterrupted power supply installed Bit 0

    0 = UPS not installed
    1 = UPS installed, ready for use
  - Utility power failed, running on UPS Bit 1

    0 = Running on utility power
    1 = Running on UPS
  - Uninterrupted power supply (UPS) bypass active Bit 2

    0 = UPS bypass not active
    1 = UPS bypass active

  - Uninterrupted power supply (UPS) battery low — Bit 3

    0 = UPS battery not low
    1 = UPS battery low

  - Auto keylock position — Bit 4

    0 = Keylock not in auto position
    1 = Keylock in auto position

  - Normal keylock position — Bit 5

    0 = Keylock not in normal position
    1 = Keylock in normal position

  - Manual keylock position — Bit 6

    0 = Keylock not in manual position
    1 = Keylock in manual position

  - Secure keylock position — Bit 7

    0 = Keylock not in secure position
    1 = Keylock in secure position

  - Reserved — Char(1)

- Reserved — Char(5)
- Most recent IPL Type — Char(1)

The Current IPL Type is the state of the IPL type at the operations panel. Possible values are A, B, C, D.

UPS installed indicates that a Uninterrupted Power Supply is installed on the system and is available for use should the power fail.

UPS power failed indicates that a utility failure has occurred and the system is currently running on battery power.

UPS bypass active indicates that the UPS has been bypassed. If a utility power failure occurs, the UPS will not supply power.

UPS battery low indicates that a UPS battery is installed on the system and the battery is low.

Auto keylock position indicates that the keylock is currently in the auto position on the operation panel.

Normal keylock position indicates that the keylock is currently in the normal position on the operation panel.

Manual keylock position indicates that the keylock is currently in the manual position on the operation panel.

Secure keylock position indicates that the keylock is currently in the secure position on the operation panel.

The Most Recent IPL Type is the type of IPL that was performed on the most recent IPL. Possible values are A, B, C, D.

Hex 016C Extended machine initialization status record
(only allowed in attribute selection value)

The entire template can be materialized but only specific fields are modifiable.

The XMISR (extended machine initialization status record) is used to report the status of the machine.

The materialize format of the XMISR is as follows:

- Number of bytes available Bin(4)
- Number of bytes provided Bin(4)
- Save storage status Char(4)
  - Checksumming status Bit(0)

    0 = Checksumming was not stopped
    1 = Checksumming was stopped
  - Completion status Bit(1)

    0 = Save storage did not complete ok
    1 = Save storage completed ok
  - System restored status Bit(2)

    0 = Save storage did not restore the system
    1 = Save storage restored the system
  - Save storage attempted Bit(3)

    0 = Save storage not attempted
    1 = Save storage was attempted
  - Unreadable sectors Bit(4)

    0 = Unreadable sectors were not found
    1 = Unreadable sectors were found during save operation
  - Check for active files on save storage media Bit(5)

    0 = Do not check for active files on save storage media
    1 = Check for active files on save storage media
  - Reserved Bit(6)
  - Reserved Bit(7)
  - Reserved Char(3)
- Save storage information Char(118)

  Tape device information Char(18)
  Number of tape device entries Bin(16)

  Tape device address Char(16)

  (Array of four entries, each 4 bytes in size)

  Tape device IOP address Char(2)

  Tape device Char(2)

device address

Tape volume names structure	Char(62)
Number of tape volume entries	Bin(16)
Tape volume names (Array of ten entries, each 6 bytes in size)	Char(60)
Tape expiration date	Char(6)
Bad sector count	Char(4)
Date from save tape	Char(6)
Time last successful save started	Char(8)
Reserved	Char(14)

- Install tape Volume ID Char(6)
- IPL sequence number ID Bin(4)

Hex 0170 Alternate initial process definition template
(can be materialized and modified)
(only allowed in attribute selection value)

The alternate initial process definition template is used by the machine when performing an automated install.

No check is made and no exception is signaled if the values in the template are invalid; however, the next automated install will not be successful.

Limitations:

Data-pointer-defined scalars are not allowed as a primary operand for this instruction. An invalid operand type exception is signaled if this occurs.

Substring operand references that allow for a null substring reference (a length value of zero) may not be specified for this instruction.

Exceptions

Exception		Operands 1	2	Other
06	Addressing			
	01 space addressing violation	X	X	
	02 boundary alignment	X	X	
	03 range	X	X	
	06 optimized addressability invalid	X	X	
10	Damage encountered			
	04 system object damage state	X	X	X
	44 partial system object damage	X	X	X
1C	Machine-dependent exception			
	03 machine storage limit exceeded			X
	0A service processor unable to process request			X

Exception		Operands 1	2	Other
20	Machine support			
	02 machine check			X
	03 function check			X
22	Object access			
	01 object not found	X	X	
	02 object destroyed	X	X	
	03 object suspended	X	X	
24	Pointer specification			
	01 pointer does not exist	X	X	
	02 pointer type invalid	X	X	
2A	Program creation			
	06 invalid operand type	X	X	
	07 invalid operand attribute	X	X	
	08 invalid operand value range	X	X	
	0A invalid operand length		X	
	0C invalid operand odt reference	X	X	
	0D reserved bits are not zero	X	X	X
2E	Resource control limit			
	01 user profile storage limit exceeded			X
32	Scalar specification			
	01 scalar type invalid	X	X	
	02 scalar attributes invalid		X	
	03 scalar value invalid		X	
36	Space management			
	01 space extension/truncation			X
38	Template specification			
	03 materialization length exception	X		

Chapter 19. Exception Specifications

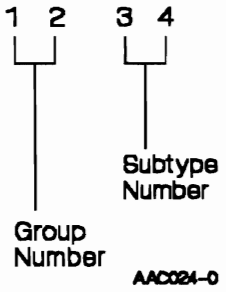
This chapter describes the exceptions which can be signaled by the machine. Exception generation is the only facility for synchronously communicating error conditions that are a direct result of AS/400 instruction processing. Machine exceptions identify error conditions that require processing before the next sequential AS/400 instruction is executed. Instructions that cause a particular exception may not function identically before execution is stopped; however, each instruction produces consistent results. These results ensure machine integrity and reliability. The results are inherent in a particular exception definition or in the detailed instruction definition.

The user can monitor any number of exceptions. There are three basic techniques for the user to handle an exception. One technique is to provide detailed handling specified by a program defined exception description object. The second technique is to provide a default exception handler for the process. This exception handler is invoked whenever an invocation fails to handle an exception. The third technique is to accept the machine default of process termination by not providing an appropriate exception handling mechanism. for a general description of exception management.

19.1 Machine Interface Exception Data

Exception data is communicated across the machine interface through a Retrieve Exception Data instruction. Certain information is available for all exceptions when an appropriate exception description has been defined by the user. That information includes the following:

- Exception identification-This is a 2-byte hexadecimal field formed by concatenating to the high-order 1-byte exception group number a low-order 1-byte exception subtype number. The format of the exception identification is as follows:



- Compare value length
- Compare value (machines signaled have a compare value of hex 00000000 with a length of 4)
- Exception-specific data
- Signaling program invocation address
- Signaled program invocation address
- Signaling program instruction address
- Signaled program instruction address
- Machine-dependent data identifying the component that generated the exception

The exception-specific data provides additional pointers and data that may be required for an individual exception.

19.2 Exception List

The following is a list of all exceptions in alphabetic and numeric order by group. The subtypes within each group are in numeric order.

02 Access Group

- 01 Object ineligible for access group

04 Access State

- 01 Access state specification invalid

06 Addressing

- 01 Space addressing violation
- 02 Boundary alignment
- 03 Range
- 04 External data object not found
- 05 Invalid space reference
- 06 Optimized addressability invalid

08 Argument/Parameter

- 01 Parameter reference violation
- 02 Argument list length violation
- 03 Argument list length modification violation

0A Authorization

- 01 Unauthorized for operation
- 02 Privileged instruction
- 03 Attempt to grant/retract authority state to an object that is not authorized
- 04 Special authorization required
- 05 Create/modify user profile beyond level of authorization
- 06 Grant/retract authority invalid.

0C Computation

- 01 Conversion
- 02 Decimal data
- 03 Decimal point alignment
- 04 Edit digit count
- 05 Edit mask syntax
- 06 Floating-point overflow
- 07 Floating-point underflow
- 08 Length conformance
- 09 Floating-point invalid operand

- 0A Size
- 0B Zero divide
- 0C Invalid floating-point conversion
- 0D Floating-point inexact result
- 0E Floating-point zero divide
- 0F Master key not defined
- 10 Weak key not valid
- 11 Key parity invalid
- 12 Invalid extended character data
- 13 Invalid extended character operation

0E Context Operation

- 01 Duplicate object identification
- 02 Object ineligible for context

10 Damage Encountered

- 02 Machine context damage state
- 04 System object damage state
- 05 Authority verification terminated due to damaged object
- 44 Partial system object damage state

12 Data Base Management

- 01 Conversion mapping error
- 02 Key mapping error
- 03 Cursor not set
- 04 Data space entry limit exceeded
- 05 Data space entry already locked
- 06 Data space entry not found
- 07 Data space index invalid
- 08 Incomplete key description
- 09 Duplicate key value in existing data space entry
- 0A End of path
- 0B Duplicate key value detected while building unique data space index
- 0D No entries locked
- 0F Duplicate key value in uncommitted data space entry
- 13 Invalid mapping template
- 14 Invalid selection template
- 15 Data space not addressed by index

16 Data space not addressed by cursor

17 Key changed since set cursor

18 Invalid key value modification

19 Invalid rule option

1A Data space entry size exceeded

1B Logical space entry size limit exceeded

1C Key size limit exceeded

1D Logical key size limit exceeded

21 Unable to maintain a unique key data space index

25 Invalid data base operation

26 Data space index with invalid floating-point field build termination

27 Data space index key with invalid floating-point field

30 Specified data space entry rejected

32 Join value changed

33 Data space index with non-user exit selection routine build termination

34 Non-user exit selection routine failure

36 No mapping code specified

37 Operation not valid with join cursor

38 Derived field operation error

39 Derived field operation error during build index

40 Invalid entry definition table

16 Exception Management

01 Exception description status invalid

02 Exception state of process invalid

03 Invalid invocation address

18 Independent Index

01 Duplicate key argument in index

1A Lock State

01 Invalid lock state

02 Lock request not grantable

03 Invalid unlock request

04 Invalid object lock transfer request

05 Invalid space location unlock

1C Machine-Dependent Exception

01 Machine-dependent request invalid

- 02 Program limitation exceeded
- 03 Machine storage limit exceeded
- 04 Object storage limit exceeded
- 06 Lock limit exceeded
- 07 Modify main storage pool controls invalid
- 08 Requested function not valid
- 09 Auxiliary storage pool number invalid

1E Machine Observation

- 01 Program not observable

20 Machine Support

- 01 Diagnose
- 02 Machine check
- 03 Function check

22 Object Access

- 01 Object not found
- 02 Object destroyed
- 03 Object suspended
- 04 Object not eligible for operation
- 05 Object not available to process
- 06 Object not eligible for destruction
- 07 Authority verification terminated due to destroyed object

24 Pointer Specification

- 01 Pointer does not exist
- 02 Pointer type invalid
- 03 Pointer addressing invalid object
- 04 Pointer not resolved

26 Process Management

- 02 Queue full

28 Process State

- 01 Process ineligible for operation
- 02 Process control space not associated with a process
- 0A Process attribute modification invalid

2A Program Creation

- 01 Program header invalid
- 02 ODT syntax error
- 03 ODT relational error

- 04 Operation code invalid
- 05 Invalid op code extender field
- 06 Invalid operand type
- 07 Invalid operand attribute
- 08 Invalid operand value range
- 09 Invalid branch target operand
- 0A Invalid operand length
- 0B Invalid number of operands
- 0C Invalid operand ODT reference
- 0D Reserved bits are not zero

2C Program Execution

- 01 Return instruction invalid
- 02 Return point invalid
- 03 Stack control invalid
- 04 Branch target invalid
- 05 Activation in use by invocation
- 06 Instruction cancellation
- 07 Instruction termination

2E Resource Control Limit

- 01 User profile storage limit exceeded

30 Journal

- 01 Apply journal changes failure
- 02 Entry not journaled
- 03 Maximum objects through a journal port limit exceeded
- 04 Invalid journal space
- 05 Maximum journal spaces attached
- 06 Journal space not at a recoverable boundary
- 07 Journal ID not unique
- 08 Object already being journaled
- 09 Transaction limit list exceeded
- 0A Data space index currently journaled
- 0B Data space index currently in force mode
- 0C Underlying data space not journaled to same journal

32 Scalar Specification

- 01 Scalar type invalid
- 02 Scalar attributes invalid

- 03 Scalar value invalid

34 Source/Sink Management

- 01 Source/sink configuration invalid
- 02 Source/sink physical address invalid
- 03 Source/sink object state invalid
- 04 Source/sink resource not available

36 Space Management

- 01 Space extension/truncation
- 02 Invalid space modification

38 Template Specification

- 01 Template value invalid
- 02 Template size invalid
- 03 Materialization length exception

3A Wait Time-Out

- 01 Dequeue
- 02 Lock
- 03 Wait on event
- 04 Space location lock wait

3C Service

- 01 Invalid service session state
- 02 Unable to start service session

3E Commitment Control

- 01 Invalid commit block status change
- 03 Commit block is attached to process
- 04 Commit block controls uncommitted changes
- 06 Commitment control resource limit exceeded
- 08 Object under commitment control being incorrectly journaled
- 10 Operation not valid under commitment control
- 11 Process has attached commit block
- 12 Objects under commitment control
- 13 Commit block not journaled
- 14 Errors during decommit
- 15 Object ineligible for commitment control
- 16 Object ineligible for removal from commitment control

40 Dump Space Management

- 01 Dump data size limit exceeded„

02 Invalid dump data insertion

03 Invalid dump space modification

04 Invalid dump data retrieval

02 Access Group

0201 Object Ineligible for Access Group

An attempt was made to insert an object into an access group. The operation could not be performed for one of the following reasons:

- The object is temporary, or the object is permanent and the access group is temporary.
- The object is restricted by the machine from membership in an access group.

Information Passed:

- Access group — System pointer
- Object to be inserted (binary 0 for objects not yet created) — System pointer

Instructions Causing Exception:

- Any create instruction that specifies an access group in the create template
- Signal Exception

04 Access State

0401 *Access State Specification Invalid*

An access state not supported by the machine was specified for an object.

Information Passed

- The invalid access state — Char(1)

Instructions Causing Exception:

- Set Access State
- Signal Exception

06 Addressing

0601 Space Addressing Violation

An attempt has been made to operate outside the current extent of the space contained in a system object.

Information Passed:

- Object referenced — System pointer
- Space offset reference attempted — Bin(4)
  This value may be zero when not available.

Instructions Causing Exception:

- Any instruction using a pointer or scalar as an operand
- Any instruction using a scalar as an index, a length suboperand, or a space pointer as a base suboperand
- Signal Exception

0602 Boundary Alignment

A program object has been referenced, and it does not have the proper alignment relative to the beginning of a space. Pointers must always be 16-byte aligned. Program objects that are not pointers must have at least the alignment specified by the ODT entry.

Information Passed:

- Addressability to pointer or template — Space pointer

Instructions Causing Exception:

- Any instruction having a pointer operand or a template operand that requires a specific boundary alignment

0603 Range

A subscript value in a compound operand array reference is outside the range defined for the array. A subscript value of less than 1 or greater than the number of elements defined by the array causes this exception.

A reference to a string has a position and/or length that exceeds the bounds of the string. A compound operand that defines a character string that does not completely fall within the bounds of the base character string was referenced. A substring with position (P) e1 and length (L) e1 does not meet the following constraint (n is the length of the base string):

$$P + L - 1 \leq n$$

Instructions Causing Exception:

- All instructions that use scalar or pointer operands
- Signal Exception

0604 External Data Object Not Found

An unsuccessful attempt was made to resolve a data pointer. The external data object specified by the initial value of the data pointer was not found in the process activations. If a program name was specified in the symbolic address, then only that program's activation

is considered for resolution. If no program was specified, then all of the programs with activations in the process are considered for data pointer resolution.

Information Passed:

- External data object name Char(32)

Instructions Causing Exception:

- Any instruction that references an external data object through a data pointer.
- Any instruction where a data pointer is used as the scalar value for an index of a length suboperand. This includes scalar and pointer operands that may be subscripted.
- Signal Exception
- Compare Pointer Addressability
- Compare Pointer for Space Addressability
- Convert Character to Numeric
- Convert External Form to Numeric
- Convert Numeric to Character
- Copy Bytes Left Adjusted
- Copy Bytes Left Adjusted With Pad
- Copy Bytes Right Adjusted
- Copy Bytes Right Adjusted With Pad
- Copy Numeric Value
- Edit
- Materialize Pointer
- Resolve Data Pointer
- Set Data Pointer Addressability
- Set Data Pointer Attributes
- Set Space Pointer From Pointer
- Set System Pointer From Pointer

0605 Invalid Space Reference

An attempt was made to address a space contained in an object that has no space.

Instruction Causing Exception:

- Set Space Pointer from Pointer

0606 Optimized Addressability Invalid

An instruction attempted to use the internally optimized value of a space pointer that was invalid due to the failure of a prior instruction in trying to access the pointer's value.

The machine may optimize the retrieval of a pointer's value by using the value retrieved on one instruction for use by multiple instructions that have need to reference the pointer's value. This avoids the overhead of continually retrieving the pointer's value from storage for every instruction that would have need to use it. If, in attempting to retrieve the pointer's value, an exception is detected, the machine marks the internally optimized value as invalid. This is done to provide for detecting the invalid addressability upon subsequent execution of instructions that depend on the internally optimized value. These instructions have no provision for retrieving the pointer's value from storage. These instructions will not redetect the original exception, but instead detect the optimized addressability invalid exception for this condition. This condition can occur when an exception is detected during an attempt to retrieve a pointer's value and the exception is ignored which causes execution of the program to continue without successfully retrieving the pointer's value.

This exception may not be detected on certain cases of internal machine optimizations that may be performed on references to space pointer machine objects. A reference to the space data addressed by the pointer is necessary to ensure consistent detection of this exception. Although the exception may not be detected for initial operations, it will be detected on any subsequent operation that references the space data addressed by the space pointer machine objects.

The optimization of retrieving a pointer's value can be prevented by specifying the abnormal attribute for the pointer.

This exception may not be detected on the operations listed below under certain cases of internal machine optimizations which may be performed on references to space pointer machine objects. The operations listed below refer to the value of a space pointer machine object, but do not have need to reference the space data the pointer addresses. A reference to the space data addressed by the pointer is necessary to insure consistent detection of this exception. Although the exception may not be detected for these operations, it will be detected upon any subsequent operation which references the space data addressed by the space pointer machine object.

The following instructions may not detect this exception upon references to a space pointer machine object.

- Add Space Pointer
- Compare Pointer for Space Addressability
- Compare Space Addressability
- Set Space Pointer
- Set Space Pointer with Displacement
- Ser Space Pointer from Pointer
- Subtract Space Pointer Offset

See the particular instruction description for more detail. *Instructions Causing Exception:*

- Any instruction using a pointer or scalar as an operand
  - Signal Exception

0's on the right. Nonzero digits would have to be truncated on the left to fit the aligned value into a 31-digit decimal field.

Instructions Causing Exception:

- Add Numeric
- Compare Numeric Value
- Subtract Numeric
- Signal Exception

0C04 Edit Digit Count

The number of digit position characters in the mask operand of an Edit instruction is not equal to the number of digits in the source operand value.

Instructions Causing Exception:

- Edit
- Signal Exception

0C05 Edit Mask Syntax

The characters of the mask operand do not follow the valid syntax rules for an Edit instruction.

Instructions Causing Exception:

- Edit
- Signal Exception

0C06 Floating-Point Overflow

The result of a floating-point operation is finite and not an invalid value, but its exponent is too large for the target floating-point format. The signed exponent has exceeded 127 in short format or 1023 in long format.

Information Passed:

• Floating-point value attributes	Char(1)
— Normal bias	Bit 0
— Modified bias	Bit 1
— Rounded to short floating-point precision	Bit 2
— NaN	Bit 3
— Reserved (binary 0)	Bits 4-7
• Reserved (binary 0)	Char(7)
• Overflowed floating-point value	Floating-(8) point
• Reserved (binary 0)	Char(16)

Instructions Causing Exception:

- Add Numeric
- Compute Math Function Using One Input Value

- Compute Math Function Using Two Input Values
- Convert Character to Numeric
- Convert Numeric to Character
- Convert Decimal Form to Floating-Point
- Copy Numeric Value
- Divide
- Extract Magnitude
- Multiply
- Negate
- Scale
- Subtract Numeric
- Signal Exception

0C07 Floating-Point Underflow

The result of a floating-point operation is not zero but has too small an exponent for the destination's format without being denormalized. The signed exponent is less than -126 in short format or less than -1022 in long format.

Information Passed:

• Floating-point value attributes	Char(1)
— Normal bias	Bit 0
— Modified bias	Bit 1
— Rounded to short floating-point precision	Bit 2
— NaN	Bit 3
— Reserved (binary 0)	Bits 4-7
• Reserved (binary 0)	Char(7)
• Underflowed floating-point value	Floating-(8) point
• Reserved (binary 0)	Char(16)

Instructions Causing Exception:

- Add Numeric
- Compute Math Function Using One Input Value
- Compute Math Function Using Two Input Values
- Convert Character to Numeric
- Convert Numeric to Character
- Convert Decimal Form to Floating-Point
- Copy Numeric Value
- Divide
- Extract Magnitude

- Multiply
- Negate
- Scale
- Subtract Numeric
- Signal Exception

0C08 Length Conformance

The operand lengths or resultant value length or both do not conform to the rules of the instruction:

CVTHC — Twice the length of the source operand must be less than or equal to the length of the receiver operand.

CVTCH — The length of the operand must be less than or equal to twice the length of the receiver operand.

CVTMC — The length of a record in the receiver was not enough to contain the converted form of a record from the source.

EDIT — The length of the resultant edited value must be equal to the length of the receiver operand.

SCAN — The length of the compare operand must not be greater than the length of the base string.

SEARCH — The length of the find operand plus the value of the location operand must be less than or equal to the length of an element of the array operand.

XLATE — The source and receiver operands must be the same length.

Instructions Causing Exception:

- Convert Character to Hex
- Convert Hex to Character
- Convert MRJE to Character
- Edit
- Extended Character Scan
- Scan
- Search
- Signal Exception
- Translate

0C09 Floating-Point Invalid Operand

A floating-point invalid operand condition is caused by one of the following conditions:

- A source operand is an unmasked NaN.
- Addition of infinities of different signs and subtraction of infinities of the same sign.
- Multiplication of zero times infinity.

- Division of zero by zero or infinity by infinity.
- Computing the sine, cosine, or tangent function for infinity.
- Computing the arc tangent, exponential, logarithm, square root, or power function for infinity when in projective infinity mode.
- Floating-point values compared unordered and no branch or indicator options are specified for the unordered, negation of unordered, equal, or negation of equal conditions on compare numeric value.
- An unordered resultant condition occurred on a computational instruction because the result was NaN, and branch or indicator conditions are specified but unordered, negation of unordered, zero, or negation of zero conditions are not specified.

Information Passed:

- Exception type Char(1)

  Hex 00 = Invalid arithmetic operation or operand is unmasked NaN.
  Hex 01 = Invalid branch or indicator conditions.

  Hex 02 through hex FF are reserved.

- Reserved (binary 0) Char(31)

Instructions Causing Exception:

- Add Numeric
- Compare Numeric Value
- Compute Math Function Using One Input Value
- Compute Math Function Using Two Input Values
- Convert Character to Numeric
- Convert Floating-Point to Decimal Form
- Convert Numeric to Character
- Copy Numeric Value
- Divide
- Extract Magnitude
- Multiply
- Negate
- Scale
- Subtract Numeric
- Signal Exception

0C0A Size

An operand was too small to contain a result. This condition is detected only when a fixed-point result is too large to be assigned to a fixed-point receiver. The receiver operand is set with the result of the operation truncated to the receiver size.

Instructions Causing Exception:

- Add Numeric
- Compute Math Function Using One Input Value
- Compute Math Function Using Two Input Values
- Convert Character to Numeric
- Convert External Form
- Convert Numeric to Character
- Copy Numeric Value
- Divide
- Divide With Remainder
- Extract Magnitude
- Multiply
- Negate
- Remainder
- Scale
- Subtract Numeric
- Sum
- Signal Exception
- Trim Length

0C0B Zero Divide

An attempt was made to divide by 0 on a fixed-point divide operation.

Instructions Causing Exception:

- Divide
- Divide With Remainder
- Remainder
- Signal Exception

0C0C Invalid Floating-Point Conversion

This exception is detected on a conversion from binary floating-point to other than a binary floating-point format because overflow, infinity, or NaN is detected before conversion is complete.

Information Passed:

• Floating-point value attributes	Char(1)
— Normal bias	Bit 0
— Modified bias	Bit 1
— Reserved (binary 0)	Bit 2
— NaN	Bit 3
— Infinity	Bit 4

  - Reserved (binary 0) Bits 5-7
- Reserved (binary 0) Char(7)
- Invalid floating-point value Floating-point (8)
- Reserved (binary 0) Char(16)

Instructions Causing Exception:

- Add Numeric
- Compute Math Function Using One Input Value
- Compute Math Function Using Two Input Values
- Convert Character to Numeric
- Convert Floating-Point to Decimal Form
- Convert Numeric to Character
- Copy Numeric Value
- Divide
- Multiply
- Negate
- Scale
- Subtract Numeric
- Signal Exception

0C0D Floating-Point Inexact Result

This exception is signaled when the rounded result of an operation is not exact because of one of the following conditions:

- The rounded result of an operation is not exact because a value is modified (rounded) to fit in a receiver operand and nonzero fraction digits are lost.
- The occurrence of a floating-point overflow condition when that condition is masked and the result is no longer exact because it is set to infinity.

Information Passed:

- Reserved (binary 0) Char(32)

Instructions Causing Exception:

- Add Numeric
- Compare Numeric Value
- Compute Math Function Using One Input Value
- Compute Math Function Using Two Input Values
- Convert Character to Numeric
- Convert Decimal Form to Floating-Point
- Convert Floating-Point to Decimal Form
- Copy Numeric Value

0E Context Operation

0E01 Duplicate Object Identification

An attempt was made to place addressability in a context to an object having the same name, type, and subtype as an existing entry in the context.

Information Passed:

- System pointer to the existing object
- Object identification Char(32)
  - Object type Char(1)
  - Object subtype Char(1)
  - Object name Char(30)

Instructions Causing Exception:

- All create instructions
- Modify Addressability
- Rename Object
- Signal Exception

0E02 Object Ineligible For Context

An attempt was made to delete addressability to an object of a type that may be addressed only by the machine context, or an attempt was made to place addressability to an object in a temporary or permanent context that may be addressed only by the machine context.

Information Passed:

- System pointer to object
- Object identification Char(32)
  - Object type Char(1)
  - Object subtype code Char(1)
  - Object name Char(30)

Instructions Causing Exception:

- Modify Addressability
- Signal Exception

10 Damage

1002 Machine Context Damage State

The machine context cannot be referenced because it is in the damaged state. The machine context rebuild option of the Reclaim instruction can be used to correct the problem or an IPL can correct the problem.

Information Passed:

- Reserved (binary 0) Char(16)
- VLOG dump ID Char(8)
- Error class Bin(2)
- The error class codes for the type of damage detected are as follows:

  Hex 0000 = Previously marked damaged
  Hex 0001 = Detected abnormal condition
  Hex 0002 = Locally invalid device sector
  Hex 0003 = Device failure

- Auxiliary storage device failure Bin(2)

  This field is defined for error classes hex 0002 and hex 0003. It is the unit number of the failing device or 0 for a main storage failure.

- Reserved (binary 0) Char(100)

Instructions Causing Exception:

- Materialize Context
- Resolve System Pointer
- Any instruction that resolves a system object that is located by the machine context
- Signal Exception

1004 System Object Damage State

A system object cannot be accessed because it is in the damaged state.

Information Passed:

- System pointer to the damaged object System pointer
- VLOG dump ID Char(8)
- Error class Bin(2)
- The error class codes for the type of damage detected are as follows:

  Hex 0000 = Previously marked damaged
  Hex 0001 = Detected abnormal condition
  Hex 0002 = Locally invalid device sector
  Hex 0003 = Device failure

- Auxiliary storage device indicator Bin(2)

This field is defined for error classes hex 0002 and hex 0003. It is the unit number of the failing device or 0 for a main storage failure.

- Reserved (binary 0) Char(100)

Instructions Causing Exception:

- Any instruction that references a system object
- Signal Exception

1005 Authority verification terminated due to damaged object

Authority verification processing terminated due to a damaged user profile or authorization list found during the check of authority for a permanent system object.

Information Passed

- System Pointer to the object for which authority was being checked Sys Ptr
- Reason Code Bin(2)

  0 = Damaged User Profile
  1 = Damaged Authorization List

  (all other values reserved)

- Reserved Char(14)
- System Pointer to the damaged User Profile or Authorization List Sys Ptr

Instructions Causing Exception

- Any instruction with operands or operand lists that refer to a permanent system object
- Signal Exception

1044 Partial System Object Damage

Partial damage to a system object has been detected.

Information Passed:

- System pointer to the damaged object System pointer
- VLOG dump ID Char(8)
- Error Class Bin(2)
- The error class codes for the type of damage detected are as follows:

  Hex 0000 = Previously marked damaged
  Hex 0001 = Detected abnormal condition
  Hex 0002 = Locally invalid device sector
  Hex 0003 = Device failure

- Auxiliary storage device indicator Bin(2)

  This field is defined for error classes hex 0002 and hex 0003. It is the unit number of the failing device or 0 for a main storage failure.

- Reserved (binary 0) Char(100)

Instructions Causing Exception:

- Any instruction that references a system object
- Signal Exception

16 Exception Management

1601 Exception Description Status Invalid

The tested exception description was not in the deferred state.

Instructions Causing Exception:

- Test Exception
- Signal Exception

1602 Exception State of Process Invalid

An attempt was made to retrieve exception data or resignal an exception when the process is not in an exception handling state; that is, the process is not in an external program, internal entry point, or branch point exception handler. The re-signal option is valid only for an external exception handler.

Instructions Causing Exception:

- Signal Exception
- Retrieve Exception Data

1603 Invalid Invocation Address

The invocation address specified in the space pointer on a Return From Exception instruction or Signal Exception instruction did not represent an existing program invocation.

Information Passed:

- Space pointer

Instructions Causing Exception:

- Return From Exception
- Sense Exception Description
- Signal Exception

1A Lock State

1A01 Invalid Lock State

The lock enforcement rule or rules were violated when an attempt was made to access an object.

Information Passed:

- System pointer to the object

Instructions Causing Exception:

- All instructions that enforce the lock rules
- Signal Exception

1A02 Lock Request Not Grantable

The lock request cannot be granted immediately and neither the synchronous nor asynchronous wait option was specified.

Information Passed:

- Pointer to lock request template — Space pointer
- Failing request number (relative entry position) — Bin(2)

Instructions Causing Exception:

- Lock Object
- Signal Exception

1A03 Invalid Unlock Request

An attempt was made to unlock a lock state not held by the current requesting process.

Information Passed:

- Pointer to unlock request template — Space pointer
- Number of requests not unlocked — Bin(2)
- Request number (relative entry position for each lock not unlocked) — Bin(2)

Instructions Causing Exception:

- Unlock Object
- Signal Exception

1A04 Invalid Object Lock Transfer Request

An attempt was made to transfer locks that were not held by the transferring process, or the transfer lock request was not granted because the lock granting rules would have been violated.

Information Passed:

- Pointer to lock transfer request template — Space pointer
- Number of requests not transferred — Bin(2)
- Request number (relative entry position for each lock not transferred) — Bin(2)

20 Machine Support

2001 Diagnose

An error or discrepancy was found when a Diagnose instruction was processed.

Information Passed:

- Space element to the subelement in the operand 2 object that was being processed
- Data — Bin(4)
  - Subidentifier unique to the requested function — Bin(2)
  - Indicator of the pointer in operand 2 that was being processed — Bin(2)

Instructions Causing Exception:

- Diagnose
- Signal Exception

2002 Machine Check

A machine malfunction affecting system-wide operation has been detected during execution of an instruction in this process.

Information Passed:

- Timestamp that gives the current value of the machine time-of-day clock. — Char(8)
- Error code indicating nature of machine check. (This value is machine-dependent and is only defined in the machine service documentation.) — Char(2)
- Reserved (binary 0) — Char(6)
- VLOG dump ID — Char(8)
- Error class — Bin(2)

  The error class codes for the type of damage detected are as follows:

  Hex 0000 = Unspecified abnormal condition
  Hex 0002 = Logically invalid device sector
  Hex 0003 = Device failure
- Auxiliary storage device indicator — Bin(2)

  This field is defined for error classes hex 0002 and hex 0003. It is the OU number of the failing device or 0 for a main storage failure.
- Reserved (binary 0) — Char(100)

Instructions Causing Exception:

- Any instruction
- Signal Exception

2003 Function Check

The executing instruction has failed unexpectedly during execution within the process.

Information Passed:

- Timestamp giving the current value of the machine time-of-day clock. Char(8)
- Error code indicating the nature of the function check. (This value is machine-dependent.) Char(2)
- Reserved (binary 0) Char(6)
- VLOG dump ID Char(8)
- Error class Bin(2)

  The error class codes for the type of damage detected are as follows:

  Hex 0000 = Unspecified abnormal condition
  Hex 0002 = Logically invalid device sector
  Hex 0003 = Device failure

- Auxiliary storage device indicator Bin(2)

  This field is defined for error classes hex 0002 and hex 0003. It is the OU number of the failing device or 0 for a main storage failure.

- Reserved (binary 0) Char(100)

Instructions Causing Exception:

- Any instruction
- Signal Exception

22 Object Access

2201 Object Not Found

An attempt to resolve addressability into a system pointer was not successful for one of the following reasons:

- The named object was not located in the context specified in the symbolic address or in any context referenced in the name resolution list.
- An object with a corresponding name was found but the user profile(s) governing execution of the instruction did not have the authority required for resolution.

Information Passed:

- Object identification Char(32)
  - Object type Char(1)
  - Object subtype Char(1)
  - Object name Char(30)
- Required authorization Char(2)

Instructions Causing Exception:

- Any instruction that references an object through a system pointer
- Signal Exception

2202 Object Destroyed

An attempt was made to reference an object that no longer exists.

This exception may not be signaled for operations which refer to the value of a space pointer machine object, but which do not attempt to reference the space data the pointer addresses. The following instructions may not signal this exception upon references to a space pointer machine object.

- Add Space Pointer
- Compare Pointer for Space Addressibility
- Compare Space Addressibilty
- Set Space Pointer
- Set Space Pointer wtih Displacement
- Set Space Pointer from Pointer
- Subtract Space Pointer Offset

See the particular instruction description for more detail. *Instructions Causing Exception:*

- Any instruction that references an object through a system pointer, a space pointer, or a data pointer
- Any instruction that references a scalar or a pointer operand when the object and the space containing the scalar or pointer have been destroyed
- Signal Exception

24 Pointer Specification

2401 Pointer Does Not Exist

A pointer reference was made to a storage location in a space that does not contain a pointer data object, or a reference was made to a space pointer machine object that was not set to address a space.

This exception may not be signaled for operations which refer to the value of a space pointer machine object, but which do not attempt to reference the space data the pointer addresses. The following instructions may not signal this exception upon references to a space pointer machine object.

- Add Space Pointer
- Compare Pointer for Space Addressability
- Compare Space Addressability
- Set Space Pointer
- Set Space Pointer With Displacement
- Set Space Pointer from Pointer
- Subtract Space Pointer Offset

See the particular instruction description for more detail.

Instructions Causing Exception:

- Any instruction that has pointer operands
- Any instruction that references a base operand (scalar or pointer) when the base pointer is not a space pointer
- Any instruction that allows a scalar defined by a data pointer to be an operand
- Any instruction that requires a pointer as part of the input template
- Signal Exception

2402 Pointer Type Invalid

An instruction has referenced a pointer object that contains an incorrect pointer type for the operation requested.

Instructions Causing Exception:

- Any instruction that has pointer operands
- Any instruction that contains a base operand (scalar or pointer) when the base pointer is not a space pointer
- Any instruction that allows a scalar defined by a data pointer to be an operand
- Any instruction that requires a pointer as part of the input template
- Signal Exception

2403 Pointer Addressing Invalid Object

An instruction has referenced a system pointer that addresses an incorrect type of system object for this operation.

Information Passed:

- The invalid system pointer

Instructions Causing Exception:

- Any instruction that references a system pointer, either as an operand or within a template operand, and that requires a specific object type as a part of its operation
- Signal Exception

2404 Pointer Not Resolved

The operation did not find a resolved system pointer. For example, NRL (name resolution list) entries must be resolved system pointers that address contents.

Information Passed:

- The invalid pointer

Instructions Causing Exception:

- Resolve System Pointer
- Any instruction that causes a system pointer to be implicitly resolved when the NRL is used in the resolution. All entries in the NRL must be resolved.
- Resolved Data Pointer
- Any instruction that causes a data pointer to be implicitly resolved. all activation entries in the process must contain a resolved pointer to the associated program.
- Signal Exception

26 Process Management

2602 Queue Full

An attempt was made to enqueue a message to a queue that is full and is not extendable.

Information Passed:

- System pointer to the queue for which the enqueue was attempted

Instructions Causing Exception:

- Enqueue
- Request I/O
- Signal Exception

2A Program Creation

2A01 Program Header Invalid

The data in the program header was invalid.

Instructions Causing Exception:

- Signal Exception

2A02 ODT Syntax Error

The syntax (bit setting) of an ODT (object definition table) entry was invalid.

Information Passed:

- ODT entry number — Char(2)

Instructions Causing Exception:

- Signal Exception

2A03 ODT Relational Error

An ODT (object definition table) entry reference to another ODT entry was invalid.

Information Passed:

- ODT entry number — Char(2)

Instructions Causing Exception:

- Signal Exception

2A04 Operation Code Invalid

One of the following conditions occurred.

- The operation code did not exist.
- The optional form was not allowed.

Information Passed:

- Instruction number of the instruction being analyzed — Ubin(2)

Instructions Causing Exception:

- Signal Exception

2A05 Invalid Op Code Extender Field

The branch/indicator options were invalid.

Information Passed:

- Instruction number of the instruction being analyzed — Ubin(2)

Instructions Causing Exception:

- Signal Exception

2A06 Invalid Operand Type

One of the following conditions was detected:

- Instruction number of the instruction being analyzed — Ubin(2)

Instructions Causing Exception:

- Signal Exception

2A0C Invalid Operand ODT Reference

The ODT reference was not within the range of the ODV.

Information Passed:

- Instruction number of the instruction being analyzed — Ubin(2)

Instructions Causing Exception:

- Signal Exception

2A0D Reserved Bits Are Not Zero

The reserved bits in an opcode or operand are nonzero.

Information Passed:

- Instruction number of the instruction being analyzed — Ubin(2)

Instructions Causing Exception:

- Signal Exception

2C Program Execution

2C01 Return Instruction Invalid

Improper usage of the Return, Transfer Control, or Return From Exception instruction occurred for one of the following reasons:

- A Return From Exception instruction was executed in an invocation that was not defined as an exception handler.
- A Return External or Transfer Control instruction was issued from a first-invocation-level exception handler.
- A Transfer Control instruction was issued from a first-invocation-level event handler.

Instructions Causing Exception:

- Return External
- Return From Exception
- Transfer Control
- Signal Exception

2C02 Return Point Invalid

An attempt was made to use a Return External instruction with a return point that was invalid for one of the following reasons:

- The return point value was outside the range of the return list specified on the preceding Call External instruction.
- A nonzero return point was supplied, but no return list was supplied on the preceding Call External instruction.
- A nonzero return point was supplied when a Return External instruction was issued in the first invocation in the process.
- A nonzero return point was supplied when the Return External instruction was issued by an invocation acting as an event handler.

Instructions Causing Exception:

- Return External
- Signal Exception

2C03 Stack Control Invalid

Information Passed:

- Cause indicator Bin(2)

  Hex 0003 = The chain being modified bit in the PSSA base entry was on when it was necessary for the machine to use the chain of PSSA activations or it was necessary for the machine to modify the chain of PSSA activations.

Instructions Causing Exception:

- Activate Program
- Call External
- De-activate Program

- Modify Automatic Storage Allocation
- Resolve Data Pointer
- Transfer Control
- Signal Exception

2C04 Branch Target Invalid

An attempt was made to branch to an instruction defined through an instruction pointer, but the instruction pointer was set by a program other than the one that issued the branch.

Information Passed:

- Instruction pointer causing the exception

Instructions Causing Exception:

- All instructions that have a branch form
- Signal Exception

2C05 Activation in Use by Invocation

An attempt was made to de-activate a program that has an existing invocation which is not the invocation issuing the instruction.

Information Passed:

- Program — System pointer

Instructions Causing Exception:

- De-activate Program
- Signal Exception

2E Resource Control Limit

2E01 User Profile Storage Limit Exceeded

The user profile specified insufficient auxiliary storage to create or extend a permanent object.

Information Passed:

- System pointer to the user profile

Instructions Causing Exception:

- All create instructions creating a permanent object
- All instructions extending a permanent object

  Any instruction which references bytes of data within a space object can cause an automatic extension of the space if the space has the attribute of being automatically extendible. Therefore, this exception may be signaled for any instruction which has an operand which references bytes of data in a space.
- Signal Exception

32 Scalar Specification

3201 Scalar Type Invalid

A scalar operand did not have the following data types required by the instruction:

- Character
- Packed decimal
- Zoned decimal
- Binary
- Floating-point

Instructions Causing Exception:

- Any instruction using a late bound (data pointer) scalar operand
- Signal Exception

3202 Scalar Attributes Invalid

A scalar operand did not have the following attributes required by the instruction:

- Length
- Precision
- Boundary

Instructions Causing Exception:

- Any instruction using a late-bound (data pointer) scalar operand
- Any instruction that verifies the length of a character scalar in a space object operand
- Signal Exception

3203 Scalar Value Invalid

A scalar operand does not contain a correct value as required by the instruction.

Information Passed:

- Length of data passed Bin(2)
- Bit offset to invalid field (relative to 0) Bin(2)
- Operand number Bin(2)
- Invalid data Char(*)

Instructions Causing Exception:

- Any instruction using a scalar operand
- Signal Exception

36 Space Management

3601 Space Extension/Truncation

A Modify Space Attributes instruction made one of the following invalid attempts to modify the size of the space:

- Truncate the space to a negative size.
- Extend or truncate a fixed size space.
- Extend a space beyond the space allowed in the referenced object.
- An operation which required an automatic extension of a space occurred when the extended space would not fit in the access group which contained it.

For information on the maximum size space allowed for a particular object, refer to the *Limitations* topic within the definition of the create instruction for that type of object.

Information Passed

- System pointer to the space

Instructions Causing Exception

- Activate Program
- Call External
- Modify Automatic Storage Allocation
- Modify Space Attributes
- Signal Exception
- Transfer Control
- Any instruction that invokes an external exception handler or an external event handler or an invocation exit
- Any instruction which has an operand which references bytes of data in a space.

  Any instruction which references bytes of data within a space object can cause an automatic extension of the space if the space has the attribute of being automatically extendible.

3602 Invalid Space Modification

A Modify Space Attributes instruction made an attempt to modify the attributes of a space but the requested modification is invalid.

Information Passed:

- System pointer to the object
- Error code Char(2)

  Error codes and their meanings are as follows:

Code	Meaning
0001	An attempt was made to modify the performance class attribute of the system object containing the space and the space was not a fixed length of size zero.

- Signal Exception

3803 Materialization Length Exception

Less than 8 bytes was specified to be available in the receiver operand of a materialize instruction.

Instructions Causing Exception:

- Any materialize instruction
- Any retrieve instruction
- Signal Exception

3A Wait Time-Out

3A01 Dequeue

A specified time period elapsed, and a Dequeue instruction was not satisfied.

Information Passed:

- The queue waited for — System pointer
- Time-out value — Char (8)

Instructions Causing Exception:

- Dequeue
- Signal Exception

3A02 Lock

A specified time period elapsed, and a Lock Object instruction was not satisfied.

Information Passed:

- System pointer to the object waited for
- Time-out value — Char(8)

Instructions Causing Exception:

- Lock Object
- Signal Exception

3A04 Space Location Lock Wait

A specified time period has elapsed and a Lock Space Location instruction has not been satisfied.

Information Passed:

- Space location — Space pointer
- Time-out value — Char(8)

Instructions Causing Exception:

- Lock Space Location
- Signal Exception

3C Service

3C01 Invalid Service Session State

The process is not in the proper service session for the request service command because of one of the following conditions:

- No service session exits for the process, and the command is other than start service session.
- The process is in service session, and the command is to start service session.
- The process is in service session, but a previous stop service session command was issued.

Instructions Causing Exception:

- Signal Exception

3C02 Unable to Start Service Session

The machine was unable to start a valid service session.

Instructions Causing Exception:

- Signal Exception

Appendix A. Instruction Summary

This appendix provides an abbreviated format of all the instructions. The instructions are listed alphabetically by instruction mnemonic.

The summary list includes the following items for each instruction.

- *Operation Description*-The name of the instruction.
- *Mnemonic*-The mnemonic assigned to the instruction.
- *Operation Code*-The operation code assigned to the instruction.
- *Number of Operands*-The number of operands (excluding the extender) in the instruction.
- *Extender*-A description of the use of the extender field.
- *Operand Syntax*-The objects allowed as operands in the instruction.
- *Resulting Conditions*-The conditions that can be set at the end of the standard operation in order to perform a conditional branch or set a conditional indicator.
- *Optional Forms*-A notation for the optional forms that are allowed for the computational instructions.

Note: This summary list can also be used as an index to identify the page where a complete description of each instruction can be found in this manual. The page number is the last item included with each instruction in this summary.

The following paragraphs further describe the summary list format of the last five items in the previous list.

Number Of Operands

Certain computational instructions allow a variable number of operands and are identified in the summary list by the following form:

number + B

The number defines the number of fixed operands. The B indicates the existence of variable operands (branch targets or indicator operands). A pair of braces around the letter indicates that the variable operands are optional.

Extender Usage

Instructions that use an extender field have a brief description of the use of the extender. Hyphens indicate that the extender is not used. Brackets indicate that the extender is optional. The abbreviation BR/IND is used to mean branch or indicator options. The extender field defines the use of the branch or indicator operands with respect to the resulting conditions of the instruction.

Resulting Conditions

Resulting conditions are the status result of the operation that is used for determining a branch target, if any.

The following conditions are indicated in the instruction summary.

P, N, Z	Positive, negative, zero
Z, NZ	Zero, not zero
H, L, E	High, low, equal
E, NE	Equal, not equal
P, Z	Positive, zero
H, L, E, U	High, low, equal, unequal
Z, O, M	Zero, ones, mixed
[N]Z[N]C	Zero and no carry, not zero and no carry, zero and carry, not zero and carry
S, NS	Signaled, not signaled
DE, I	Exception deferred, exception ignored
DQ, NDQ	Dequeued, not dequeued

Optional Forms

All instructions are classified as computational or noncomputational format. The format determines how the operation code is interpreted and whether optional forms of the instruction are allowed. (See "Instruction Format" in Chapter 1. "Introduction").

Certain computational instructions allow optional forms. The following optional forms can be specified:

- *B (Branch Form)*-The resulting conditions of the operation are compared with the branch options specified in the extender field. If one of the options is satisfied, a branch is executed to the branch target corresponding to the branch option.
- *I (Indicator Form)*-The resulting conditions of the operation are compared with the indicator options specified in the extender field. If one of the options is satisfied, the indicator corresponding to that option is assigned a value of hex F1. The other indicators referred to by the operation are assigned a value of hex F0.
- *S (Short Form)*-The operand that acts as a receiver in the instruction can also be one of the source operands.
- *R (Round Form)*-If the result of the operation is to be truncated before being placed in the receiver, rounding is performed.

A.1 Instruction Stream Syntax

In this instruction summary, the following metalanguage is used to describe the machine interface instruction set operand syntax.

Metasymbol	Meaning
{}	Choose from a series of alternatives
[ ]	Enclose an optional entry or entries
|	OR - used to separate alternatives
.N.	Repeat previous entry, up to N times
::=	Is defined as - define a metavariable Metavariable ::= Metadefinition
DESC-{}	Description of a metavariable in English

Notes:

1. Some of the computational op codes require an extender field while on other op codes an extender field is optional. Some computational op codes may be optionally short, or round.

Program Object Definitions

ARG-LIST ::= DESC-{operand list which defines an argument list}

B-ARRAY ::= DESC-{array of binary variables} B-PT ::= DESC-{branch point} BIN ::= DESC-{binary} BIN[N] ::= DESC-{binary object with precision N} BT ::= DESC-{instruction number | relative instruction number | instruction pointer | branch pointer | IDL element | null}

C-ARRAY ::= DESC-{array of character string variables} CHAR ::= DESC-{character string which is either variable or constant} CHAR[N] ::= DESC-{string at least N bytes long} CHARV ::= DESC-{char variable} CHARC ::= DESC-{char constant}

D-PTR ::= DESC-{data pointer}

EXCP-DESC ::= DESC-{exception description}

F-BT ::= DESC-{instruction number | relative instruction number | branch point} F-P ::= DESC-{floating-point value}

IDL ::= DESC-{instruction definition list} IT ::= DESC-{char|numeric variable used as an indicator target} I-ENT PT ::= DESC-{internal entry point} I-PTR ::= DESC-{instruction pointer}

NULL ::= DESC-{indicates a null operand [X'0000']} NUMERIC ::= DESC-{binary | zoned | packed | numeric scalar} N-ARRAY ::= DESC-{array of numeric variable}

OP-LIST ::= DESC-{operand list}

PROCESS ::= DESC-{character string that names a process} PTR ::= DESC-{a 16-byte, 16-byte-boundary-aligned pointer element} P-ARRAY ::= DESC-{an array of 16 bytes, 16-byte-boundary-aligned pointer(s)}

SPDO ::= DESC-{space pointer data object} S-PTR ::= DESC-{system pointer} SPP ::= DESC-{space pointer} SPP-ARRAY ::= DESC-{an array of space pointer variables}

Notes:

1. NUMERIC, CHAR, BIN, and UBIN may be followed by the special characters S, C, V. CHAR, BIN, and UBIN may also be followed by the special character I. These characters further qualify the object as being scalar, constant, variable or immediate, respectively.
2. All array objects are variable.

System Object Declarations

ACTV ENTRY ::= DESC-{SPP that addresses an activation}
AG ::= DESC-{S-PTR that addresses an access group}
AL ::= DESC-{S-PTR that addresses an authorization list}

CD ::= DESC-{S-PTR that addresses a controller description} CSD ::= DESC-{S-PTR that addresses a class of service description} CONTEXT ::= DESC-{S-PTR that addresses a context} CURSOR ::= DESC-{S-PTR that addresses a cursor}

DATA SPACE ::= DESC-{S-PTR that addresses a data space} DCT ::= DESC-{S-PTR that addresses a dictionary} DS-INDEX ::= {S-PTR that addresses a data space index}

INDEX ::= DESC-{S-PTR that addresses an index}

LUD ::= DESC-{S-PTR that addresses a logical unit description}

MD ::= DESC-{S-PTR that addresses a mode description}

ND ::= DESC-{S-PTR that addresses a network description}

PCS ::= DESC-{S-PTR to process control space} PROGRAM ::= DESC-{S-PTR that addresses a program}

SPACE ::= DESC-{a system pointer pointing to a space object}

QUEUE ::= DESC-{S-PTR that addresses a queue}

USER PROFILE ::= DESC-{S-PTR that addresses a user profile}

Resulting Conditions Definitions

ZC ::= DESC-{zero with carry}

[N]ZC ::= DESC-{[not] zero with carry}

Z[N]C ::= DESC-{zero with [no] carry}

[N]Z[N]C ::= DESC-{[not] zero with [no] carry}

CR ::= DESC-{completed record}

DE ::= DESC-{deferred}

DEN ::= DESC-{denormalized}

DQ ::= DESC-{dequeued}

NDQ ::= DESC-{not dequeued}

ECE ::= DESC-{escape code encountered}

E ::= DESC-{equal}

H ::= DESC-{high}

I ::= DESC-{ignored}

IN ::= DESC-{infinity}

L ::= DESC-{low}

M ::= DESC-{mixed}

N ::= DESC-{negative}

NaN ::= DESC-{symbolic not-a-number}

NE ::= DESC-{not equal}

NRN ::= DESC-{normalized real number}

NS ::= DESC-{not signaled}

NZ ::= DESC-{not zero}

O ::= DESC-{ones}

P ::= DESC-{positive}

RO ::= DESC-{receiver overrun}

S ::= DESC-{signaled}

SE ::= DESC-{source exhausted}

TR ::= DESC-{truncated record}

U ::= DESC-{unequal}

UN ::= DESC-{unordered}

Z ::= DESC-{zero}

Instruction Summary (Alphabetical Listing by Mnemonic)

Operation Description	Mnemonic	Op Code	Number of Operands	Operand Syntax	Resulting Condi-tions	Optional Forms	Page
Activate Program	ACTPG	0212	2	{ACTV ENTRY | PROGRAM},PROGRAM	-	-	6-1
Add Logical Character	ADDLC	1023	3+[B]	CHARV, CHARS.2., [BT.4. | IT.4.]	[N]Z[N]C	[B | I, S]	1-1
Add Numeric	ADDN	1043	3+[B]	NUMERICV, NUMERICS.2., [BT.3. | IT.3.]	P, N, Z	[B | I, S, R]	1-4
Add Space Pointer	ADDSPP	0083	3	SPP.2., BINS	-	-	3-1
And	AND	1093	3+[B]	CHARV, CHARS.2., [BT.3. | IT.3.]	Z, NZ	[B | I, S]	1-8
Branch	B	1011	1	BT	-	-	1-11
Compute Array Index	CAI	1044	4	BINV, BINS.3.	-	-	1-30
Call Internal	CALLI	0293	3	I-ENT PT, {ARG LIST | NULL}, I-PTR	-	-	6-11
Call External	CALLX	0283	3	PROGRAM | SPP, {ARG LIST | NULL}, {IDL | NULL}	-	-	6-5
Concatenate	CAT	10F3	3	CHARV, CHARS.2.	-	-	1-47
Clear Bit in String	CLRBTS	102E	2	{CHARV | NUMERICV}, BINS	-	-	1-13
Clear Invocation Exit	CLRIEXIT	0250	0	-	-	-	6-13
Compute Math Function Using One Input Value	CMF1	100B	3+[B]	NUMERICV, CHARS[2], NUMERICS, [BT.4. | IT.4.]	P, N, Z, UN	[B | I]	1-32
Compute Math Function Using Two Input Values	CMF2	100C	4+[B]	NUMERICV, CHARS[2], NUMERICS, NUMERICS, [BT.4. | IT.4.]	P, N, Z, UN	[B | I]	1-42
Compare Bytes Left-Adjusted	CMPBLA	10C2	2+B	{CHARS | NUMERICS}.2., {BT.3. | IT.3.}	H, L, E	{B | I}	1-15
Compare Bytes Left-Adjusted With Pad	CMPBLAP	10C3	3+B	{CHARS | NUMERICS}.3., {BT.3. | IT.3.}	H, L, E	{B | I}	1-18

Operation Description	Mnemonic	Op Code	Number of Operands	Operand Syntax	Resulting Condi-tions	Optional Forms	Page
Compare Bytes Right-Adjusted	CMPBRA	10C6	2+B	{CHARS | NUMERICS}.2., {BT.3. | IT.3.}	H, L, E	{B | I}	1-21
Compare Bytes Right-Adjusted With Pad	CMPBRAP	10C7	3+B	{CHARS | NUMERICS}.3., {BT.3. | IT.3.}	H, L, E	{B | I}	1-24
Compare Numeric Value	CMPNV	1046	2+B	NUMERICS.2., {BT.3. | IT.3.}	H, L, E	{B | I}	1-27
Compare Pointer for Space Address-ability	CMPPSPAD	10E6	2+B	{SPP | D-PTR}, {NUMERICV | CHARV | C-N-ARRAY | SPP | D-PTR}	H, L, E, U	{B | I}	3-3
Compare Pointer for Object Address-ability	CMPPTRA	10D2	2+B	{D-PTR | SPP | S-PTR | I-PTR}.2.	E, NE	[B | I]	2-1
Compare Pointer Type	CMPPTRT	10E2	2+B	{D-PTR | SPP | S-PTR | I-PTR}, {CHARS[1]NULL}	E, NE	{B | I}	2-4
Compare Space Addressability	CMPSPAD	10F2	2+B	{CHARV | C-ARRAY | NUMERICV | N-ARRAY | PTR | P-ARRAY.2.	H, L, E, U	{B | I}	3-6
Copy Bytes to Bits Arithmetic	CPYBBTA	104C	4	{NUMERICV | CHARV}, BINI.2., {NUMERICV | CHARV}	-	-	1-135
Copy Bytes to Bits Logical	CPYBBTL	103C	4	{NUMERICV | CHARV}, BINI.2., {NUMERICV | CHARV}	-	-	1-137
Copy Bytes Left-Adjusted	CPYBLA	10B2	2	{NUMERICV | CHARV}, {NUMERICS | CHARS}	-	-	1-121
Copy Bytes Left-Adjusted With Pad	CPYBLAP	10B3	3	{NUMERICV | CHARV}, {NUMERICS | CHARS}.2.	-	-	1-123
Copy Bytes Overlap Left-Adjusted	CPYBOLA	10BA	2	{NUMERICV | CHARV}.2.	-	-	1-125
Copy Bytes Overlap Left-Adjusted With Pad	CPYBOLAP	10BB	3	{NUMERICV | CHARV}.2., {NUMERICS | CHARS}	-	-	1-127

Operation Description	Mnemonic	Op Code	Number of Operands	Operand Syntax	Resulting Condi-tions	Optional Forms	Page
Copy Bytes Right-Adjusted	CPYBRA	10B6	2	{NUMERICV | CHARV}, {NUMERICS | CHARS}	-	-	1-131
Copy Bytes Right-Adjusted With Pad	CPYBRAP	10B7	3	{NUMERICV | CHARV}, {NUMERICS | CHARS}.2.	-	-	1-133
Copy Bytes Repeatedly	CPYBREP	10BE	2	{NUMERICV | CHARV}, {NUMERICS | CHARS}	-	-	1-129
Copy Bits Arith-metic	CPYBTA	102C	4	{NUMERICV | CHARV}.2., BINI.2.	-	-	1-109
Copy Bits Logical	CPYBTL	101C	4	{NUMERICV | CHARV}.2., BINI.2.	-	-	1-111
Copy Bits With Left Logical Shift	CPYBTLLS	102F	3	{CHARV | NUMERICV}, {CHARS | NUMERICS}, CHARS[2]	-	-	1-113
Copy Bits With Right Arithmetic Shift	CPYBTRAS	101B	3	{CHARV | NUMERICV}, {CHARS | NUMERICS}, CHARS[2]	-	-	1-115
Copy Bits With Right Logical Shift	CPYBTRLS	103F	3	{CHARV | NUMERICV}, {CHARS | NUMERICS}, CHARS[2]	-	-	1-118
Copy Bytes With Pointers	CPYBWP	0132	2	{CHARV | PTR}, {CHARV | PTR | NULL}	-	-	2-7
Copy Extended Characters Left-Adjusted With Pad	CPYECLAP	1053	3	D-PTR CHARS,D-PTR CHARS,CHAR	-	-	1-139
Copy Hex Digit Numeric to Numeric	CPYHEXNN	1092	2	{NUMERICV | CHARV}, {NUMERICS | CHARS}	-	-	1-143
Copy Hex Digit Numeric to Zone	CPYHEXNZ	1096	2	{NUMERICV | CHARV}, {NUMERICS | CHARS}	-	-	1-145
Copy Hex Digit Zone to Numeric	CPYHEXZN	109A	2	{NUMERICV | CHARV}, {NUMERICS | CHARS}	-	-	1-147

Operation Description	Mnemonic	Op Code	Number of Operands	Operand Syntax	Resulting Condi-tions	Optional Forms	Page
Copy Hex Digit Zone to Zone	CPYHEXZZ	109E	2	{NUMERICV | CHARV}, {NUMERICS | CHARS}	-	-	1-149
Copy Numeric Value	CPYNV	1042	2+[B]	NUMERICV, NUMERICS, [BT.3. | IT.3.]	P, N, Z	[B | I, R]	1-151
Convert BSC to Character	CVTBC	10AF	3	CHARV, CHARV[3], CHARS	CR, SE, TR	[B | I]	1-49
Convert Char-acter to BSC	CVTCB	108F	3	CHARV, CHARV[3], CHARS	SE, RO	[B | I]	1-53
Convert Char-acter to Hex	CVTCH	1082	2	CHARV, CHARS	-	-	1-57
Convert Char-acter to MRJE	CVTCM	108B	3	CHARV, CHARV[13], CHARS	SE, RO	[B | I]	1-59
Convert Char-acter to Numeric	CVTCN	1083	3	NUMERICV, CHARS, CHARS[7]	-	-	1-65
Convert Char-acter to SNA	CVTCS	10CB	3	CHARV, CHARV[15], CHARS	SE, RO	[B | I]	1-68
Convert Decimal Form to Floating-Point	CVTDFFP	107F	3	F-PS, NUMERICS, NUMERICS	-	-	1-78
Convert External Form to Numeric Value	CVTEFN	1087	3	NUMERICV, CHARS, {CHARS[3] | NULL}	-	-	1-81
Convert Floating-Point to Decimal Form	CVTFPDF	10BF	3	NUMERICV, NUMERICV, F-PS	-	Round	1-84
Convert Hex to Character	CVTHC	1086	2	CHARV, CHARS	-	-	1-87
Convert MRJE to Character	CVTMC	10AB	3	CHARV, CHARV[6], CHARS	SE, RO	[B | I]	1-89
Convert Numeric to Character	CVTNC	10A3	3	CHARV, NUMERICS, CHARS[7]	-	-	1-94
Convert SNA to Character	CVTSC	10DB	3	CHARV, CHARV[14], CHARS	SE, RO, ECE	[B | I]	1-97
De-activate Program	DEACTPG	0225	1	PROGRAM | NULL	-	-	6-14
Dequeue	DEQ	1033	3+[B]	CHARV, SPP, QUEUE, [BT.2. | IT.2.]	DQ, NDQ	[B | I]	9-2

Operation Description	Mnemonic	Op Code	Number of Operands	Operand Syntax	Resulting Conditions	Optional Forms	Page
Divide	DIV	104F	3+[B]	NUMERICV, NUMERICS.2., [BT.3. | IT.3.]	P, N, Z	[B | I, S, R]	1-154
Divide With Remainders	DIVREM	1074	4+[B]	NUMERICV, NUMERICS.2., NUMERICV	P, N, Z	[B | I, S, R]	1-158
Extended Character Scan	ECSCAN	10D4	4	B-ARRAY, CHARS, CHARS, CHARS[1]	P, Z, ECE	[B | I]	1-176
Edit	EDIT	10E3	3	CHARV, NUMERICS, CHARS	-	-	1-162
End	END	0260	0	-	-	-	6-16
Enqueue	ENQ	036B	3	QUEUE, CHARS, SPP	-	-	9-8
Ensure Object	ENSOBJ	0381	1	S-PTR	-	-	15-2
Exchange Bytes	EXCHBY	10CE	2	{CHARV | NUMERICV}.2.	-	-	1-171
Extract Exponent	EXTREXP	1072	2	BINV, F-PS	NRN, DEN, IN, NaN	[B | I]	1-180
Extract Magnitude	EXTRMAG	1052	2+[B]	NUMERICV, NUMERICS, [BT.3. | IT.3.]	P, Z	[B | I, S]	1-183
Find Independent Index Entry	FNDINXEN	0494	4	SPP, INDEX, SPP.2.	-	-	8-2
Insert Independent Index Entry	INSINXEN	04A3	3	INDEX, SPP.2.	-	-	8-6
Lock Object	LOCK	03F5	1	SPP	-	-	10-2
Lock Space Location	LOCKSL	03F6	2	SPP, CHARS[1]	-	-	10-8
Materialize Access Group Attributes	MATAGAT	03A2	2	SPP, AG	-	-	15-4
Materialize Authority List	MATAL	01B3	3	SPP, AL, SPP	-	-	13-7
Materialize Allocated Object Locks	MATAOL	03FA	2	SPP, {S-PTR | SPDO}	-	-	10-10
Materialize Authority	MATAU	0153	3	SPP, S-PTR, {USER PROFILE | NULL}	-	-	13-2
Materialize Authorized Objects	MATAUOBJ	013B	3	SPP, USER PROFILE, CHARS[1]	-	-	13-12

Operation Description	Mnemonic	Op Code	Number of Operands	Operand Syntax	Resulting Condi-tions	Optional Forms	Page
Materialize Authorized Users	MATAUU	0143	3	SPP, S-PTR, CHARS[1]	-	-	13-17
Materialize Context	MATCTX	0133	3	SPP, {CONTEXT | NULL}, CHARS	-	-	12-2
Materialize Dump Space	MATDMPS	04DA	2	SPP, S-PTR	-	-	16-2
Materialize Data Space Record Locks	MATDRECL	032E	2	SPP, SPP	-	-	10-13
Materialize Exception Description	MATEXCPD	03D7	3	SPP, EXCP-DESC, CHARS[1]	-	-	11-1
Materialize Instruction Attri-butes	MATINAT	0526	2	SPP, CHARS	-	-	17-2
Materialize Invocation	MATINV	0516	2	SPP.2.	-	-	17-8
Materialize Invocation Entry	MATINVE	0547	3	CHARV, {CHARV.1. | NULL}, CHARS.1. | NULL}	-	-	17-13
Materialize Invocation Stack	MATINVS	0546	2	SPP, {S-PTR | NULL}	-	-	17-18
Materialize Machine Attri-butes	MATMATR	0636	2	SPP, CHARS[2] | SPP	-	-	18-2
Materialize Object Locks	MATOBJLK	033A	2	SPP, S-PTR	-	-	10-17
Materialize Program	MATPG	0232	2	SPP, PROGRAM	-	-	5-2
Materialize Process Attri-butes	MATPRATR	0333	3	SPP, {PCS | NULL}, CHARS[1]	-	-	14-2
Materialize Process Record Locks	MATPRECL	031E	2	SPP, SPP	-	-	10-24
Materialize Process Locks	MATPRLK	0312	2	SPP, {PCS | NULL}	-	-	10-21
Materialize Pointer	MATPTR	0512	2	SPP, {S-PTR | D-PTR | SPP | I-PTR}	-	-	17-22
Materialize Pointer Locations	MATPTRL	0513	3	SPP.2., BINS	-	-	17-27

Operation Description	Mnemonic	Op Code	Number of Operands	Operand Syntax	Resulting Conditions	Optional Forms	Page
Materialize Queue Attributes	MATQAT	0336	2	SPP, QUEUE	-	-	9-11
Materialize Queue Messages	MATQMSG	033B	3	SPP, S-PTR, CHARS[16]	-	-	9-15
Materialize Resource Management Data	MATRMD	0352	2	SPP, CHARS[8]	-	-	15-8
Materialize Space Attributes	MATS	0036	2	SPP, S-PTR	-	-	4-2
Materialize Selected Locks	MATSELLK	033E	2	SPP, {S-PTR | SPP}	-	-	10-28
Materialize System Object	MATSOBJ	053E	2	SPP, S-PTR	-	-	17-30
Materialize User Profile	MATUP	013E	2	SPP, USER PROFILE	-	-	13-22
Modify Automatic Storage Allocation	MODASA	02F2	2	{SPP | NULL}, BINS	-	-	6-17
Modify Exception Description	MODEXCPD	03EF	3	EXCP-DESC, SPP, CHARS[4]	-	-	11-5
Modify Independent Index	MODINX	0452	2	S-PTR, CHARS[4]	-	-	8-13
Modify Space Attributes	MODS	0062	2	S-PTR, BINS	-	-	4-6
Multiply	MULT	104B	3+[B]	NUMERICV, NUMERICS.2., [BT.3. | IT.3.]	P, N, Z	[B | I, S, R]	1-186
Negate	NEG	1056	2+[B]	NUMERICV, NUMERICS, [BT.3. | IT.3.]	P, N, Z	[B | I, S]	1-190
No Operation	NOOP	0000	0	-	-	-	7-2
No Operation and Skip	NOOPS	0001	1	UBINI	-	-	7-3
Not	NOT	108A	2+[B]	CHARV, CHARS, [BT.3. | IT.3.]	Z, NZ	[B | I, S]	1-193
Or	OR	1097	3+[B]	CHARV, CHARS.2., [BT.3. | IT.3.]	Z, NZ	[B | I, S]	1-196
Override Program Attributes	OVRPGATR	0006	2	UBINI.2.	-	-	7-5
Remainder	REM	1073	3+[B]	NUMERICV, NUMERICS.2., [BT.3. | IT.3.]	P, N, Z	[B | I, S]	1-199

Operation Description	Mnemonic	Op Code	Number of Operands	Operand Syntax	Resulting Condi-tions	Optional Forms	Page
Retrieve Excep-tion Data	RETEXCPD	03E2	2	SPP, CHARS[1]	-	-	11-8
Remove Inde-pendent Index Entry	RMVINXEN	0484	4	{SPP | NULL}, INDEX, SPP.2.	-	-	8-16
Resolve Data Pointer	RSLVDP	0163	3	D-PTR, {CHARS[32] | NULL}, {S-PTR | NULL}	-	-	2-10
Resolve System Pointer	RSLVSP	0164	4	S-PTR, {CHARS[34] | NULL}, {S-PTR | NULL}, {CHARS[2" | NULL}	-	-	2-13
Return From Exception	RTNEXCP	03E1	1	SPP	-	-	11-12
Return External	RTX	02A1	1	{BINS | NULL}	-	-	6-20
Scale	SCALE	1063	3+[B]	NUMERICV, NUMERICS, BINS, [BT.3. | IT.3.]	P, N, Z	[B | I, S]	1-203
Scan	SCAN	10D3	3+[B]	{BINV | B-ARRAY}, CHARS.2., [BT.3. | IT.3.]	P, Z	[B | I]	1-207
Scan With Control	SCANWC	10E4	4+[B]	SPP, CHARV[8], CHARS[4], [BT.4. | IT.4.]	-	-	1-210
Search	SEARCH	1084	4+[B]	{BINV | B-ARRAY}, {N-ARRAY | C-ARRAY}, {CHARV | NUMERICV}, BINS	P, Z	[B | I]	1-219
Set Access State	SETACST	0341	1	SPP	-	-	15-26
Set Argument List Length	SETALLEN	0242	2	ARG-LIST, BINS	-	-	6-23
Set Bit in String	SETBTS	101E	2	{CHARV | NUMERICV}, BINS	-	-	1-222
Set Data Pointer	SETDP	0096	2	D-PTR {NUMERICV | N-ARRAY | CHARV | C-ARRAY}	-	-	3-9

Operation Description	Mnemonic	Op Code	Number of Operands	Operand Syntax	Resulting Conditions	Optional Forms	Page
Set Data Pointer Addressability	SETDPADR	0046	2	D-PTR {NUMERICV | N-ARRAY | CHARV | C-ARRAY}	-	-	3-11
Set Data Pointer Attributes	SETDPAT	004A	2	D-PTR, CHARS[7]	-	-	3-13
Set Invocation Exit	SETIEXIT	0252	2	S-PTR, ARG LIST NULL	-	-	6-25
Set Instruction Pointer	SETIP	1022	2	I-PTR, F-BT	-	-	1-224
Set System Pointer From Pointer	SETSPFP	0032	2	S-PTR, {D-PTR | SPP | S-PTR | I-PTR}	-	-	3-25
Set Space Pointer	SETSPP	0082	2	SPP, {CHARV | C-ARRAY | NUMERICV | N-ARRAY | PTR | P-ARRAY}	-	-	3-16
Set Space Pointer with Displacement	SETSPPD	0093	3	SPP, {CHARV | C-ARRAY | NUMERICV | N-ARRAY | PTR | P-ARRAY}	-	-	3-18
Set Space Pointer From Pointer	SETSPPFP	0022	2	SPP, {S-PTR | D-PTR | SPP}	-	-	3-20
Set Space Pointer Offset	SETSPPO	0092	2	SPP, BINS	-	-	3-23
Signal Exception	SIGEXCP	10CA	2+[B]	SPP.2., [BT.2. | IT.2.]	I, DE	[B | I]	11-20
Sense Exception Description	SNSEXCPD	03E3	3	SPP.3.	-	-	11-16
Store and Set Computational Attributes	SSCA	107B	3	CHARV[5], {CHARS[5] | NULL}, {CHARS[5] | NULL}	-	-	1-226
Store Parameter List Length	STPLLEN	0241	1	BINV	-	-	6-28
Store Space Pointer Offset	STSPPO	00A2	2	BINV, SPP	-	-	3-27
Subtract Logical Character	SUBLC	1027	3+[B]	CHARV, CHARS.2., [BT.3. | IT.3.]	[N]Z[N]C	[B | I, S]	1-231
Subtract Numeric	SUBN	1047	3+[B]	NUMERICV, NUMERICS.2., [BT.3. | IT.3.]	P, N, Z	[B | I, S, R]	1-234

Operation Description	Mnemonic	Op Code	Number of Operands	Operand Syntax	Resulting Condi-tions	Optional Forms	Page
Subtract Space Pointer Offset	SUBSPP	0087	3	SPP.2., BINS	-	-	3-29
Test Authority	TESTAU	10F7	3	{CHARV[2] | NULL}, {S-PTR | SPDO}, CHARS[2]	-	-	13-26
Test Extended Authorities	TESTEAU	10FB	3	{CHARV[8] | NULL}, CHARS[8], {BINS[2] | NULL}	-	-	13-31
Test Exception	TESTEXCP	104A	2+[B]	SPP, EXCP-DESC, [BT.2. | IT.2.]	S, NS	[B | I]	11-25
Trim Length	TRIML	10A7	3	NUMERICV, CHARS, CHARS[1]	-	-	1-252
Test Bit in String	TSTBTS	100E	2+B	{CHARS | NUMERICS}, BINS, {BT.2. | IT.2.}	Z, O	{B | I}	1-240
Test Bits Under Mask	TSTBUM	102A	2+B	{CHARS | NUMERICS}.2., {BT.3. | IT.3.}	Z, O, M	{B | I}	1-243
Test and Replace Char-acters	TSTRPLC	10A2	2	CHARV, CHARS	-	-	1-238
Unlock Object	UNLOCK	03F1	1	SPP	-	-	10-35
Unlock Space Location	UNLOCKSL	03F2	2	SPP, CHARS[1]	-	-	10-39
Verify	VERIFY	10D7	3+[B]	{BINV | B-ARRAY}, CHARS.2. [BT.3. | IT.3.]	P, Z	[B | I]	1-254
Wait On Time	WAITTIME	0349	1	CHARS.16.	-	-	14-15
Transfer Control	XCTL	0282	2	PROGRAM | SPP, {ARG LIST | NULL}	-	-	6-30
Transfer Object Lock	XFRLOCK	0382	2	PCS, SPP	-	-	10-31
Translate	XLATE	1094	4	CHARV, CHARS, {CHARS | NULL}, CHARS	-	-	1-246
Translate With Table	XLATEWT	109F	3	CHARV, CHARS, CHARS	-	-	1-249

Operation Description	Mnemonic	Op Code	Number of Operands	Operand Syntax	Resulting Condi-tions	Optional Forms	Page
Exclusive Or	XOR	109B	3+[B]	CHARV, CHARS.2., [BT.3. | IT.3.]	Z, NZ	[B | I, S]	1-173

Index

Reader's Comments

Application System/400™
Machine Interface
Functional Reference

Publication No. SC21-8226-0

Use this form to tell us what you think about this manual. If you have found errors in it, or if you want to express your opinion about it (such as organization, subject matter, appearance) or make suggestions for improvement, this is the form to use.

To request additional publications, or to ask questions or make comments about the functions of IBM products or systems, you should talk to your IBM representative or to your IBM authorized remarketer. This form is provided for comments about the information in this manual and the way it is presented.

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you.

Be sure to print your name and address below if you would like a reply.

Name

Address

Company or Organization

Phone No.

Reader's Comments
SC21-8226-0

IBM®

Cut or Fold
Along Line

Fold and Tape | **Please do not staple** | Fold and Tape

NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS MAIL PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

International Business Machines Corporation
Information Development
Department 245
3605 North Hwy 52
ROCHESTER MN 55901-9986

Fold and Tape | **Please do not staple** | Fold and Tape

SC21-8226-0

Cut or Fold
Along Line

IBM®

SC21-8226-0

Printed in U.S.A.