REVIEW COPY OF

7.0 REALITY

ASSEMBLY LANGUAGE

INSTRUCTION SET

TABLE OF CONTENTS

0 FOREWORD

This document describes the Reality assembly language instruction repertoire for Release D.

Section 1 presents general information about classes and types of instructions. It also explains the format used in describing the instructions.

Section 2 describes each instruction. The instructions are presented in alphabetical order according to their opcode mnemonics combined with their operand types. For example, the instruction MOV Wi,Wj would be in order according to the symbol MOVWW.

The Index includes all of the instruction mnemonics and the instruction opcodes.

1 INTRODUCTION

The REALITY assembly language instruction set has been completely revised with the release 7.0. Many new instructions have been added to the repertoire and the object code has been radically changed. This document describes the executable instructions. It does not discuss assembler directives.

1.1 INFORMATION COMMON TO SEVERAL INSTRUCTIONS

This section discusses information that applies to different classes of instructions, such as arithmetic instructions, string instructions, and so forth.

1.1.1 ARITHMETIC AND ACF ALTERATIVE INSTRUCTIONS

The arithmetic instructions perform arithmetic operations on the contents of accumulators or locations. These contents are loosely referred to as binary integers since the bit configurations represent a binary integer.

Some instructions perform their functions on the accumulators (element-to-accumulator or accumulator-to-element operations). Others perform their functions as element-to-element operations.

The high-order bit of a binary integer is the sign bit. Zero in this bit means positive; one means negative. Negative values are represented in two's complement form.

When an integer is loaded into the accumulator, the sign bit is extended to the left to fill the accumulator (either D0 or FP0 - depending on the operation or operands).

Each operand for an arithmetic instruction must lie entirely within a single frame. That is, all of the bytes of the operand must be in the same frame.

Many of the arithmetic instructions set the Arithmetic Condition Flags (ACF). Whenever the ACF is updated, the entire byte is overwritten and bits not related to the instruction are undefined after the update. The instruction descriptions specify whether the ACF changed.

Element-to-element instructions do not use the accumulators, nor do they change the contents of the accumulators. Furthermore, they do <u>not</u> change the ACF.

The names of the individual flags, their bit positions, and their meanings are as follows:

Name	Bit	Description
NUMBIT	1	Used by the ASCII-number-conversion instructions to indicate that a valid number was converted.
VALBIT	2	Set by ASCII-number-conversion instructions to indicate that the maximum number of digits specified has been converted.
EQUBIT	5	Set by the COMP instruction.
LOWBIT	6	Set by the COMP instruction.
OVFBIT	7	Set when the accumulator overflows (i.e., the result would not fit in the accumulator).

1.1.2 STRING MOVEMENT AND SCAN INSTRUCTIONS

The string instructions may be used to move or scan character strings. A string is defined as a logically contiguous group of bytes that may extend across linked frame boundaries. The location of a string is indicated by an address register pointing at either the byte before or the byte after the string. The length of a string is controlled by either a count of the number of bytes in the string (count in T0 control), a set of characters which terminate the string (delimiter control), or another register pointing at the opposite end of the string (register control). In a move instruction the string destination is indicated by a register pointing at the byte before or the byte after the new location.

Count in T0 Control. When count control is used in the string instructions, T0 (the lower half of D0) is presumed to contain a count before the instruction is executed. The count is decremented before the instruction is executed.

Delimiter Control. Delimiter control refers to the testing of the contents of a byte for a match with one of seven possible characters as defined by a match field (N) in the instruction. For an unsuccessful match, instruction execution is repeated.

The set of characters tested in the match is defined by the match field and the SC0, SC1, and SC2 bytes in the PCB. For each bit position one through seven in the match field that is a 1, a match test is done. If bit position zero in the match field is a 1, the instruction stops on the first equal match. If bit position zero is a 0, the instruction stops when the contents of a byte does not contain one of the characters specified in the match set. The table below shows the test done for each bit position in the match field.

Bit	Test Performed
0	1 = stop on equal; 0 = stop on unequal
1	1 = compare with X'FF', segment mark (SM)
2	1 = compare with X'FE', attribute mark (AM)
3	1 = compare with X'FD', value mark (VM)
4	1 = compare with X'FC', subvalue mark (SVM)
5	1 = compare with contents of SC0
6	1 = compare with contents of SC1
7	1 = compare with contents of SC2

Register Control. Address register 15 (R15) is used with register control string instructions. Before these instructions are executed, R15 must have been loaded with the address of the last byte to be moved. The register pointing at the source string is incremented or decremented each time a character is moved. When the contents of the source register equal the contents of R15, execution ceases.

Cautions Regarding String Addressing. All the string instructions either increment or decrement the addresses of the source and destination bytes. Hence, it must be remembered that the address of a string must point to one byte before or one byte after the string.

The requirement that the address register point to one byte before the string's first byte has some important ramifications when the string's first byte is the first byte of the frame. Three cases must be considered: the frame is the first frame in a linked set; the frame is in a linked set, but it is not the first frame; the frame is unlinked.

Recall that the first data byte of a frame in a linked set requires a displacement of 1 in the address register. If the frame is in a linked set, referencing the byte before the first byte of the first frame requires an address with a displacement of zero.

In the case of the frame being the first frame in a linked set, when this address is attached during the execution of a string instruction, the firmware takes into account that a string instruction is executing and attaches the address register to byte zero of the frame rather than causing a BACKWARD LINK ZERO trap to the debugger. When the address is incremented, the first data byte of the frame will be referenced. Thus, string instructions may reference the "byte before" of the first frame of a linked set without difficulty.

A linked frame other than the first presents a similar situation, but should be avoided. When an address register with a displacement of 0 is attached in this situation, the firmware can attach the register to point to the last byte of the previous frame. This means that the previous frame must be read into memory (if not already in memory) at the start of instruction execution. When the register contents are incremented by the string instruction, the frame containing the string must be read into memory. Hence, when a string instruction references the first data byte of a linked frame, other than the first frame, the previous frame will always be read into memory just to satisfy the attachment of zero displacement. For the sake of efficiency, avoid this situation whenever possible.

When the string starts at byte zero of an unlinked frame (the first data byte), it is impossible to reference the previous byte. But byte zero can be referenced; therefore a string instruction can be used to handle the string starting at byte one. To move the contents of byte zero of an unlinked frame an instruction such as MCC Ri,Rj must be used.

Cautions Regarding String Addressing (cont). When the decrementing string instructions are used at the last data byte of a frame, the results are analogous to the cases of the first data byte. In the case of the last data byte of the last frame of a linked set, the byte cannot be referenced. Attempting to do so with a displacement of 501 will cause a FORWARD LINK ZERO trap to the Debug state.

If the linked frame is not the last frame, referencing the last data byte will cause the next frame to be read into memory to satisfy the attachment of the register (just as the previous one is read when the first byte is referenced).

For an unlinked frame, the last data byte cannot be referenced with a decrementing string instruction.

Once a string instruction has started execution, if the specified control does not stop execution properly, the addresses will be incremented or decremented until they reach one beyond the highest or one below the lowest allowed address. At that point one of the following Debug state traps will occur:

FORWARD LINK ZERO	-	linked set of frames when incrementing
BACKWARD LINK ZERO	-	linked set of frames when decrementing
CROSSING FRAME LIMIT	-	nonlinked frame when either incrementing or decrementing.

1.1.3 SPECIAL ADDRESS REGISTERS

Address registers 0 and 1 are used in special ways in the system. When referenced in an instruction, they should be used with the following information in mind.

Address Register 0. Register 0 (R0) is used in a special way. This register always points to the PCB (i.e., byte 0 of the PCB).

When a virtual process is not active, R0 is in detached form and contains the frame number of the PCB in its FID field and 0 in its displacement field.

Register 0 is attached when the process is activated. Thus, the attached register (R0) points to the (beginning of the) buffer in main memory where the PCB is being held.

Address Register 1. When a process is not active, detached address register 1 (R1) contains the virtual memory address minus 1 (i.e., the FID and displacement (minus one)) for the next instruction to be executed. Thus, R1 acts as a repository for the program instruction counter for that process between activations.

When the process is started, the buffer address of the program frame (as determined from the buffer map) is added to the displacement from R1. This value (a main memory address) is placed into a hardware instruction counter. The register is then converted to the attached form with the buffer address set to byte zero of the program frame.

This allows R1 to be used as a base register to reference data in the program frame because, when the process is active, R1 points to byte 0 of the program frame (mode) in main memory. Never use R1 as a destination or to specify a destination.

When the process is deactivated, the main memory address in the instruction counter is converted to the corresponding FID and displacement, and R1 is detached with these values in it.

NOTE

> **Never update data in an ABS frames using R1. The data will not be written back to disk. It is best, never to change ABS by software.**

1.1.4 STORAGE REGISTERS AND NORMALIZED ADDRESSES

Storage Registers. A six-byte location used to contain a virtual storage address is called a storage register.

The Load Address Register instruction moves six bytes from memory into bytes 2 through 7 of an address register after zeroing bytes 0 and 1 of the register. Note that the address register is detached. As usual, the address register will have to be attached in order for the data referenced by the register to be accessed by the CPU. Correspondingly, with the Store Address Register instruction, the firmware takes the detached form of an address register and stores bytes 2 through 7 in a six-byte memory location (the contents of the address register do not change). In summary the storage register is a virtual storage location that contains a virtual storage address.

Normalized and Unnormalized Addresses. A normalized address is one whose displacement is between 0 and 1023, inclusive, for an unlinked frame and between 1 and 1000, inclusive, for a linked frame. An address in an address register becomes unnormalized when the memory address is incremented, or decremented, beyond the buffer end. This causes the register to become detached with a displacement greater than a frame size or less than zero. When the register is attached, the address will be normalized by the firmware.

If a detached address register is moved to a storage register, the storage register may contain an unnormalized address. This can cause errors if the storage register is used in a Compare Address Register instruction. The Attach Register instruction can be used to attach a register before its contents are moved to a storage register.

1.1.5 INSTRUCTIONS WITH OFFSET

Some instructions reference data by means of a base address register and an offset value within the instruction. When such an instruction is executed, each address register referencing data is attached. This means that the register's memory address field points to a byte in a memory buffer. The instruction's offset value and the contents of the memory address field are then added together in a firmware register to yield the effective address. Since the address register has been attached to a particular buffer, the effective address must point to a byte in that same buffer. Furthermore, the entire operand (either a bit or one, two, four or six bytes) must lie entirely within the buffer.

Therefore, an instruction that references data by means of a register and an offset within the instruction must reference data that are entirely within the frame pointed to by the address register. Although it is possible to initialize a register to point to a frame (say to the end of the frame), and it is (conceptually) possible to use an instruction that references data by means of this register plus some large offset which would point to a byte in the next linked frame, the firmware cannot accommodate such an arrangement because the address register is attached to a buffer before the instruction offset value is added to form the effective address. A "CROSSING FRAME LIMIT" trap to the Debug state will result unless the instruction description specifies otherwise.

1.2 INSTRUCTION DESCRIPTION FORMAT

1.2.1 FORMAT OVERVIEW

Each instruction is explained with a diagram and several specific items. Figure A shows the instruction diagram and the description items. Listed below are the explanations of the items and legends for what may appear under the rubrics.

INSTRUCTION TITLE

This gives the name of the instruction as it is commonly referred to in the text. The title is usually suggestive of the function of the instruction.

MNEMONIC

The source code symbol that represents the machine instruction. The mnemonic is the symbol that is used in the opcode field of the assembly language instruction. All the mnemonics of the assembly language instructions are contained in the OSYM table.

A lowercase c is used in a mnemonic to indicate a comparison code such as E for equal, LE for less than or equal, NZ for non zero. For example, BDc could be BDZ or BDNZ among other possibilities. See section COMPARE CODE INDICATORS.

INSTRUCTION TITLE

MNEMONIC {OPERAND1 {,OPERAND2 {,OPERAND3}}}

INSTRUCTION TYPE

xxxx	xxxx	xxxx	xxxx	xxxx	xxxx
1	2	3	4	5	6

LENGTH IN BITS

Detailed Description of Instruction Execution

Operand Types

Object Code (in Hex)

Example

Cautions and Notes

Figure A. Instruction Diagram and Description Items.

OPERANDS

The types of operands that are allowed in the operand field of the assembly language instruction are represented by the following codes:

A	Absolute memory location
B	Bit reference
C	Character
D	Double tally
F	Triple tally (Full tally)
H	Half tally
K	Literal (represents a constant)
L	Location local to the frame (i.e., displacement within the program frame)
N	Immediate data (constant)
R	Address register
S	Storage register
T	Tally
V	Three-byte value in the low-order bytes of a four-byte field
W	Representative symbol for any of the tally type operands. That is, when W appears as an operand, a half tally (H or C), tally (T), double tally (D), three-byte value (V), or triple tally (F or S) type operand may be valid in the assembly language instruction.

The letters i and j are used to distinguish between two operands. In general the operand with the i is the source of data (the "from" field) and the j operand is the receiver of data (the "to" field). For example, the instruction MOV Wi,Wj moves data from the Wi operand to the Wj operand. On the other hand, instruction AND Rj,Ri forms the logical AND of the two operands and stores the result in the Rj operand.

INSTRUCTION TYPE

There are 13 instruction types. The instruction type is indicated just above the object code format display. See the section INSTRUCTION TYPES.

OBJECT CODE FORMAT

The object code format is displayed in a diagram. The diagram breaks the format into four-bit fields (nibbles).

The operation code bits are shown in binary representation. Other fields that have fixed numeric values are also shown in binary.

Where an instruction allows more than one of the tally type operands, half-tally (H), character (C), tally (T), double tally (D), triple tally (F), storage register (S), and three-byte value (V), the operand is represented by a W. The symbols for these types of operands are defined with three data: base address register, offset from the address register, and size indicator. The position of these data in the object format is indicated by means of subcodes as follows:

Wir	Base address register for operand Wi
Wid	Offset from address register Wir
Wik	Size indicator of operand Wi (See the section OPERAND SIZE INDICATOR.)

Of course, Wj rather than Wi is used where appropriate.

If an instruction requires a particular tally type to be used, only the code for that type is used to explain the instruction. For example, the instruction READ Rj,Hi requires that the second operand be a half tally so H, not W, is used in the instruction description.

Some other descriptor codes are as follows:

f	-	FID (Frame Identification number)
e	-	Entry point number
c	-	Compare code indicator. The indicator is used in conditional branch instructons with meanings such as: less than, non- zero, equal, etc. See the section COMPARE CODE INDICATORS.

DETAILED DESCRIPTION OF INSTRUCTION EXECUTION

This gives a full description of the instruction execution. The definitions of common terms are as follows:

Load - The contents of some location replaces the contents of an address register or accumulator.

Store - The contents of an address register or accumulator replaces the contents of some location.

Move - The contents of a location replaces the contents of some other location, or the contents of a register replaces the contents of another register.

Since most instructions change the contents of a location, register, or accumulator, the following symbols will be used often:

C(element) means the content of an element.

For example,

C(Ri) means the content of address register Ri. The content of an address register is, of course, the address of some byte.

C(C(Rj)) means the content of the content of address register Rj, that is, the content of a byte pointed to by the content of Rj.

C(Wir) means the content of the base address register for operand Wi, that is, the address of a byte.

C(Wir)+Wid means the address of a byte offset by Wid from the byte pointed to by address register Wir.

C(C(Wjr)+Wjd) means the content of an element that is offset by Wjd from the byte pointed to by address register Wjr.

The offset d is a byte offset for all tally-type elements.

The size of the element is determined by the k field of the instruction for a tally type of operand.

LIST OF VALID OPERAND TYPES AND RESPECTIVE OBJECT CODE

This is a cross reference to the object code listings of an assembly language program. The value of k is listed when appropriate, along with the associated valid operand types. The object code is given in hexadecimal notation and the nibble positions noted.

EXAMPLES

The operands are usually defined in the following artificial manner: HTYPI or HTYPJ for half-tally operands, CTYPI or CTYPJ for character type operands, TTYPI or TTYPJ for tally operands, etc.

CAUTIONS AND NOTES

Peculiarities regarding the instruction execution or caveats on instruction usage are noted.

1.2.2 OPERAND LENGTH INDICATOR

The operand length indicator (k) occupies four bits.

The length specifier indicates the following:

k	Operand Size will be:	Associated Operand Type
0	byte	C,H
1	tally	T
2	double tally	D
3	triple tally	F,S
7	four bytes	V
8-F	bit	B

When k is 7, the operand size is four bytes but the data are the contents of the three low-order bytes of the four.

When k is 8 or greater, the operand is a bit (B type) and the three low-order bits of k specify the offset of the bit in the addressed byte.

1.2.3 COMPARE CODE INDICATORS

The compare code indicators are assembled into the c field of the object code. The c field indicates all possible conditions for both arithmetic, logical, and data type compares as follows:

Compare Type	Mnemonic	Meaning	Object Code
Arithmetic		ALWAYS	0000
	L (LZ)	LESS THAN	0001
	E (Z)	EQUAL	0010
	H (HZ)	HIGHER	0011
		NEVER	0100
	HE (HEZ)	HIGHER THAN OR EQUAL	0101
	U (NZ)	UNEQUAL	0110
	LE (LEZ)	LESS THAN OR EQUAL	0111
Logical	BS	BIT SET/ODD	1000
	L	LESS THAN	1001
	E	EQUAL	1010
	H	HIGHER	1011
	BZ	BIT CLEAR/EVEN	1100
	HE	HIGHER THAN OR EQUAL	1101
	U	UNEQUAL	1110
	LE	LESS THAN OR EQUAL	1111
Data Type	A	ALPHABETIC	1000
	N	NUMERIC	1001
	H	HEXADECIMAL	1001
	NA	NON ALPHABETIC	1100
	NN	NON NUMERIC	1101
	NH	NON HEXADECIMAL	1101

The assembler's OSYM does not have mnemonics for ALWAYS and NEVER. These combinations are for patching only.

The mnemonics shown in parentheses are used when comparing with zero.

2 INSTRUCTION SET REPERTOIRE

Each instruction in the repertoire is described on the following pages.

ADD TO ACCUMULATOR

ADD Wi

Type 5 Instruction

1010	0011	Wir	0000	Wik	Wid			32 BITS
1	2	3	4	5	6	7	8	

Detailed Description of Instruction Execution

The integer addressed by the operand is added to the 32-bit accumulator (D0) with sign extension. That is, C(C(Wir)+Wid)+C(D0) replaces C(D0).

C(ACF) is updated to reflect overflow.

Operand Types	Wik	Object Code (Hex) 1	2	3	4	5	6	7	8
Hi	0	A	3	i	0	0	d	d	d
Ti	1	A	3	i	0	1	d	d	d
Di	2	A	3	i	0	2	d	d	d
Vi	7	A	3	i	0	7	d	d	d

Example

```
Label   OpC     Operand         Comment
Field   Field   Field           Field
-----   -----   -------         -------

        ADD     HTYPI
        ADD     DTYPJ
```

Programming Note: The mnemonic ADD may be used with F type operands (six-byte elements), but the object code generated by the assembler is the same as that generated for instruction ADDX.

ADD TO EXTENDED ACCUMULATOR

ADDX Wi

Type 5 Instruction

1010	0010	Wir	0000	Wik	Wid			32 BITS
1	2	3	4	5	6	7	8	

Detailed Description of Instruction Execution

C(C(Wir)+Wid) is added algebraically to C(FP0) and this sum replaces C(FP0). That is, C(C(Wir)+Wid)+C(FP0) replaces C(FP0).

C(ACF) is updated to reflect overflow.

Operand Types	Wik	Object Code (Hex) 1	2	3	4	5	6	7	8
Hi	0	A	2	i	0	0	d	d	d
Ti	1	A	2	i	0	1	d	d	d
Di	2	A	2	i	0	2	d	d	d
Fi	3	A	2	i	0	3	d	d	d
Vi	7	A	2	i	0	7	d	d	d

Example

Label Field	OpC Field	Operand Field	Comment Field
	ADDX	DTYPI	ADD DOUBLE WORD TO FP0

AND WITH IMMEDIATE

AND Rj,N
AND N,Rj

Type 4 Instruction

1000	0001	Rj	0000	N	
1	2	3	4	5	6

24 BITS

Detailed Description of Instruction Execution

C(C(Rj)) are logically ANDed with N. The result replaces C(C(Rj)).

Operand Types	Object Code (Hex) 1 2 3 4 5 6
Rj,N	8 1 j 0 n n
N,Rj	8 1 j 0 n n

Example

```
Label   OpC     Operand         Comment
Field   Field   Field           Field
-----   -----   -------         -------

        AND     R3,X'B'
        AND     X'C',R5
```

NOTE : **Provided for compatibility. Preferred usage is AND WW.**

AND WITH STORAGE

AND Rj,Ri

Type 3 Instruction

1110	0001	Rj	Ri
1	2	3	4

16 BITS

Detailed Description of Instruction Execution

C(C(Rj)) are logically ANDed with C(C(Ri)).

The result replaces C(C(Rj)).

Operand Types	Object Code (Hex) 1 2 3 4
Rj,Ri	E 1 j i

Example

Label Field	OpC Field	Operand Field	Comment Field
	AND	R15,R14	

NOTE : Provided for compatibility. Preferred usage is AND WW.

AND ELEMENT WITH ELEMENT

AND Wi,Wj

Type 6 Instruction

1111	1011	Wir	Wjr	Wik	Wid		
1	2	3	4	5	6	7	8

Wjk	Wjd		
9	10	11	12

48 BITS

Detailed Description of Instruction Execution

C(C(Wir)+Wid) is logically ANDed with C(C(Wjr)+Wjd). The result replaces C(C(Wjr)+Wjd).

Operand Types	Wik	Wjk	Object Code (Hex) 1	2	3	4	5	6	7	8	9	0	1	2
Ci,Cj	0	0	F	B	r	r	0	d	d	d	0	d	d	d
Hi,Hj	0	0	F	B	r	r	0	d	d	d	0	d	d	d
Ti,Tj	1	1	F	B	r	r	1	d	d	d	1	d	d	d
Di,Dj	2	2	F	B	r	r	2	d	d	d	2	d	d	d
Fi,Fj	3	3	F	B	r	r	3	d	d	d	3	d	d	d
Si,Sj	3	3	F	B	r	r	3	d	d	d	3	d	d	d
Vi,Vj	7	7	F	B	r	r	7	d	d	d	7	d	d	d

Example

Label Field	OpC Field	Operand Field	Comment Field
	AND	HTYPI,HTYPJ	
	AND	CTYPI,HTYPJ	
	AND	TTYPI,TTYPJ	
	AND	DTYPI,DTYPJ	
	AND	FTYPI,FTYPJ	
	AND	STYPI,STYPJ	

ATTACH REGISTER

ATT Rj

Type 3 Instruction

0011	1101	Rj	0000	16 BITS
1	2	3	4	

Detailed Description of Instruction Execution

If Rj is detached, it is attached.

Operand Types	Object Code (Hex) 1 2 3 4
Rj	3 D j 0

Example

Label Field	OpC Field	Operand Field	Comment Field
	ATT	R12	ATTACH REG

ATTACH REGISTER AND SET REGISTER

ATT Ri,Rj

Type 3 Instruction

0011	1110	Ri	Rj
1	2	3	4

16 BITS

Detailed Description of Instruction Execution

This instruction attaches Ri if it is detached. It then sets the C(Rj) to point in unlinked format to byte zero of the buffer that the C(Ri) addresses.

Operand Types	Object Code (Hex) 1 2 3 4
Ri,Rj	3 E i j

Example

Label Field	OpC Field	Operand Field	Comment Field
	ATT	R12,R14	POINT R14 AT R12'S BUFFER

BRANCH TO LOCAL LOCATION

B L

Type 12 Instruction

0011	0101	0000	L-1			24 BITS
1	2	3	4	5	6	

Detailed Description of Instruction Execution

Control is transferred to the location in the current buffer defined by the local address of the label L. The local address is a byte displacement into the executing frame.

Operand Types	Object Code (Hex) 1	2	3	4	5	6
L	3	5	0	1	1	1

Example

Label Field	OpC Field	Operand Field	Comment Field
	B	LOCLAB	

Programming Note: For entry points into a mode OSYM has the mnemonic instruction EP L, which assembles to the same object code as this instruction.

BRANCH ON BIT CONDITION (SET OR ZERO)

BBc Bi,L

Type 9 Instruction

1011	0101	Bir	0000	Bik	Bid		
1	2	3	4	5	6	7	8

c	L-1		
9	10	11	12

48 BITS

Detailed Description of Instruction Execution

The code c can take on the following values:

	Complete Mnemonic	In the Instruction	Compare Relation
Z	BBZ	0010	(BIT) ZERO
S	BBS	0110	(BIT) SET

C(C(Bir) + Bid) is tested for set (one) or zero.

If the relation corresponds to the comparison code c, control transfers to L.

Operand Types	Wik	Object Code (Hex) 1 2 3 4 5 6 7 8 9 0 1 2
Bi,L	b	B 5 i 0 b d d d c 1 1 1

Example

Label Field	OpC Field	Operand Field	Comment Field
	BBZ	BTYPI,LOCLAB	
	BBS	BTYPJ,LOCLAB	

BRANCH ON BIT CONDITION (SET OR ZERO) WITH RELATIVE OFFSET

BBc Ri,Hj,L

Type 9 Instruction

1101	0010	Hjr	Ri	Hjk	Hjd		
1	2	3	4	5	6	7	8

c	L-1		
9	10	11	12

48 BITS

Detailed Description of Instruction Execution

The code C can take on the following values:

	Complete Mnemonic	In the Instruction	Compare Relation
Z	BBZ	0010	(BIT) ZERO
S	BBS	0110	(BIT) SET

C(C(Ri)+C(C(Hjr)+Hjd)) is tested for set (one) or zero.

If the relation corresponds to the comparison code c, control transfers to L.

Note: The bit offset, that is, C(C(Hjr)+Hjd)), must be zero or positive.

Operand Types	Wik	Object Code (Hex) 1 2 3 4 5 6 7 8 9 0 1 2
Ri,Hj,L	0	D 2 j i 0 d d d c 1 1 1

Example

Label Field	OpC Field	Operand Field	Comment Field
	BBZ	R15,HTYPI,LOCLAB	
	BBS	R6,HTYPJ,LOCLAB	

BRANCH ON CHARACTER ALPHABETIC

BCcA Ri,L

Type 8 Instruction

1000	1000	Ri	0000	c	L-1			32 BITS
1	2	3	4	5	6	7	8	

Detailed Description of Instruction Execution

The code c can take on the following values:

	Complete Mnemonic	In the Instruction	Compare Relation
	BCA	1000	ALPHABETIC
N	BCNA	1100	NON ALPHABETIC

C(C(Ri)) is tested as to whether or not it is alphabetic by using a bit map in the PCB at location X '3E0'.

If the relation corresponds to the comparison code c, the instruction causes a branch to L.

Operand Types	Object Code (Hex) 1 2 3 4 5 6 7 8
Ri,L	8 8 i 0 c l l l

Example

Label Field	OpC Field	Operand Field	Comment Field
	BCA	R6,LOCLAB	
	BCNA	R5,LOCLAB	

BRANCH ON RELATIVE CHARACTER COMPARE

BCc Ci,Rj,L

Type 9 Instruction

1101	0111	Cir	Rj	Cik	Cid		
1	2	3	4	5	6	7	8

c	L-1		
9	10	11	12

48 BITS

Detailed Description of Instruction Execution

The code c can take on the following values:

	Complete Mnemonic	In the Instruction	Compare Relation
L	BCL	1001	LESS THAN
E	BCE	1010	EQUAL
H	BCH	1011	HIGHER THAN
HE	BCHE	1101	HIGHER THAN OR EQUAL
U	BCU	1110	UNEQUAL
LE	BCLE	1111	LESS THAN OR EQUAL

C(C(Cir)+Cid) is compared logically with C(C(Rj)).

If the relation corresponds to the comparison code c, the instruction causes a branch to L to occur.

Operand Types	Cik	Object Code (Hex) 1 2 3 4 5 6 7 8 9 0 1 2
Ci,Rj,L	0	D 7 i j 0 d d d c 1 1 1

Example

Label Field	OpC Field	Operand Field	Comment Field
	BCE	CTYPI,R7,LLB	
	BCLE	CTYPJ,R13,LOCB	

BRANCH IMMEDIATE COMPARED TO CHARACTER

BCc N,Ri,L

Type 10 Instruction

1000	0111	Ri	0000	N	c
1	2	3	4	5 6	7

L-1	40 BITS
8 9 10	

Detailed Description of Instruction Execution

The code c can take on the following values:

	Complete Mnemonic	In the Instruction	Compare Relation
L	BCL	1001	LESS THAN
E	BCE	1010	EQUAL
H	BCH	1011	HIGHER THAN
HE	BCHE	1101	HIGHER THAN OR EQUAL
U	BCU	1110	UNEQUAL
LE	BCLE	1111	LESS THAN OR EQUAL

N is compared logically with C(C(Ri)); the instruction causes a branch to L if the relation corresponds to the comparison code c.

Operand Types	Object Code (Hex) 1 2 3 4 5 6 7 8 9 0
Ri,N,L	~~4 6~~ i 0 n n c 1 1 1
	87

Example

Label Field	OpC Field	Operand Field	Comment Field
	BCU	C'D',R14,LOCLAB	
	BCLE	C'J',R9,LOCABS	

BRANCH ON CHARACTER NUMERIC

BCcN Ri,L

Type 8 Instruction

1000	1001	Ri	0000	c	L-1			32 BITS
1	2	3	4	5	6	7	8	

Detailed Description of Instruction Execution

The code c can take on the following values:

	Complete Mnemonic	In the Instruction	Compare Relation
	BCN	1001	NUMERIC
N	BCNN	1101	NON NUMERIC

C(C(Ri)) is tested as to whether or not it is numeric, that is, ASCII 0 to 9.

If the relation corresponds to the comparison code c, the instruction causes a branch to L.

Operand Types	Object Code (Hex) 1 2 3 4 5 6 7 8
Ri,L	8 9 i 0 c l l l

Example

Label Field	OpC Field	Operand Field	Comment Field
	BCN	R6,LOCLAB	
	BCNN	R5,LOCLAB	

BRANCH ON CHARACTER COMPARE

BCc Ri,Rj,L

Type 8 Instruction

1110	0111	Ri	Rj	c	L-1			32 BITS
1	2	3	4	5	6	7	8	

Detailed Description of Instruction Execution

The code c can take on the following values:

	Complete Mnemonic	In the Instruction	Compare Relation
L	BCL	1001	LESS THAN
E	BCE	1010	EQUAL
H	BCH	1011	HIGHER THAN
HE	BCHE	1101	HIGHER THAN OR EQUAL
U	BCU	1110	UNEQUAL
LE	BCLE	1111	LESS THAN OR EQUAL

C(C(Ri)) are compared logically with C(C(Rj)); the instruction causes a branch to L if the relation corresponds to the comparison code c.

Operand Types	Object Code (Hex) 1 2 3 4 5 6 7 8
Ri,Rj	E 7 i j c 1 1 1

Example

Label Field	OpC Field	Operand Field	Comment Field
	BCE	R14,R15,LOCLAB	
	BCLE	R7,R9,LOCABS	

BRANCH ON CHARACTER HEXADECIMAL

BCcX Ri,L

Type 8 Instruction

1000	1010	Ri	0000	c	L-1			32 BITS
1	2	3	4	5	6	7	8	

Detailed Description of Instruction Execution

The code c can take on the following values:

	Complete Mnemonic	In the Instruction	Compare Relation
	BCX	1001	HEXADECIMAL
N	BCNX	1101	NON HEXADECIMAL

C(C(Ri)) is tested as to whether or not it is hexadecimal, that is, ASCII 0 to 9 or A to F.

If the relation corresponds to the comparison code c, the instruction causes a branch to L.

Operand Types	Object Code (Hex) 1 2 3 4 5 6 7 8
Ri,L	8 A i 0 c l l l

Example

Label Field	OpC Field	Operand Field	Comment Field
	BCX	R6,LOCLAB	
	BCNX	R5,LOCLAB	

BRANCH ON REGISTER TO REGISTER COMPARE

Bc Ri,Rj,L

Type 8 Instruction

0101	0100	Ri	Rj	c	L-1			32 BITS
1	2	3	4	5	6	7	8	

Detailed Description of Instruction Execution

The code c can take on the following values:

	Complete Mnemonic	In the Instruction	Compare Relation
E	BE	1010	EQUAL
U	BU	1110	UNEQUAL

Ri and Rj are normalized and attached. The FID fields of the two registers are compared. If they are equal, the displacement fields are compared. The instruction causes a branch to L if the result of the compares corresponds to the comparison code c. C(Ri) and C(Rj) are not changed other than being normalized and attached.

Operand Types	Object Code (Hex) 1 2 3 4 5 6 7 8
Ri,Rj,L	5 4 i j c l l l

BRANCH ON ADDRESS REGISTER COMPARE

Bc Si,Rj,L
Bc Rj,Si,L

Type 9 Instruction

1101	0100	Sir	Rj	Sik	Sid		
1	2	3	4	5	6	7	8

c	L-1		
9	10	11	12

48 BITS

Detailed Description of Instruction Execution

The code c can take on the following values:

	Complete Mnemonic	In the Instruction	Compare Relation
E	BE	1010	EQUAL
U	BU	1110	UNEQUAL

This instruction is always executed as Si compared to Rj. If the second form is used, the c field conditions are inverted.

If Rj is attached, the detached form is calculated. If Rj is detached, the register is attached and then the detached form is calculated.

In either case, the virtual storage address in Rj is normalized (displacement between 0 and 511 for unlinked frames and displacement between 0 and 500 for linked frames).

After Rj is normalized, the displacement fields and the FID fields of the two operands are compared. If the relation corresponds to the comparison code c, the instruction causes a branch to L.

BRANCH ON ADDRESS REGISTER COMPARE (cont)

Operand Types	Wik	Object Code (Hex) 1 2 3 4 5 6 7 8 9 0 1 2
Si,Rj,L	3	B 4 i j 3 d d d c 1 1 1

Example

Label Field	OpC Field	Operand Field	Comment Field
	BE	R14,STYPI,LOCLAB	
	BU	STYPJ,R5,LOCLAB	

BRANCH ON ZERO CONDITION

Bc Wi,L

Type 9 Instruction

1011	0100	Wir	0000	Wik	Wid		
1	2	3	4	5	6	7	8

c	L-1		
9	10	11	12

48 BITS

The code c can take on the following values:

	Complete Mnemonic	In the Instruction	Compare Relation
LZ	BLZ	0001	LESS THAN
Z	BZ	0010	EQUAL
HZ	BHZ	0011	HIGHER THAN
HEZ	BHEZ	0101	HIGHER THAN OR EQUAL
NZ	BNZ	0110	UNEQUAL
LEZ	BLEZ	0111	LESS THAN OR EQUAL

C(C(Wir)+Wid) is compared arithmetically with zero; the instruction causes a branch to L if the relation corresponds to the comparison code c.

C(ACF) is not changed.

Operand Types	Wik	Object Code (Hex) 1	2	3	4	5	6	7	8	9	0	1	2
Hi,L	0	B	4	i	0	0	d	d	d	c	1	1	1
Ti,L	1	B	4	i	0	1	d	d	d	c	1	1	1
Di,L	2	B	4	i	0	2	d	d	d	c	1	1	1
Fi,L	3	B	4	i	0	3	d	d	d	c	1	1	1
Vi,L	7	B	4	i	0	7	d	d	d	c	1	1	1

Example

```
Label   OpC     Operand          Comment
Field   Field   Field            Field
-----   -----   -------          -------

        BNZ     HTYPI,LOCLAB
        BLEZ    FTYPI,LOCLAB
```

BRANCH ON ELEMENT COMPARE

Bc Wi,Wj,L

Type 11 Instruction

1111	0100	Wir	Wjr	Wik	Wid		
1	2	3	4	5	6	7	8

Wjk	Wjd			c	L-1			64 BITS
9	10	11	12	13	14	15	16	

Detailed Description of Instruction Execution

The code c can take on the following values:

	Complete Mnemonic	In the Instruction	Compare Relation
L	BL	0001	LESS THAN
E	BE	0010	EQUAL
H	BH	0011	HIGHER THAN
HE	BHE	0101	HIGHER THAN OR EQUAL
U	BU	0110	UNEQUAL
LE	BLE	0111	LESS THAN OR EQUAL

C(C(Wir)+Wid) is compared arithmetically with C(C(Wjr)+Wjd). If the relation corresponds to the comparison code c, the instruction causes a branch to L.

Note: Storage registers (element type S) can only be used with mnemonics BE and BU.

Operand Types	Wik	Wjk	Object Code (Hex) 1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6
Hi,Hj,L	0	0	F	4	i	j	0	d	d	d	0	d	d	d	c	1	1	1
Ti,Tj,L	1	1	F	4	i	j	1	d	d	d	1	d	d	d	c	1	1	1
Di,Dj,L	2	2	F	4	i	j	2	d	d	d	2	d	d	d	c	1	1	1
Fi,Fj,L	3	3	F	4	i	j	3	d	d	d	3	d	d	d	c	1	1	1
Si,Sj,L	3	3	F	4	i	j	3	d	d	d	3	d	d	d	c	1	1	1
Vi,Vj,L	7	7	F	4	i	j	7	d	d	d	7	d	d	d	c	1	1	1

BRANCH ON STORAGE COMPARE (cont)

Example

```
Label   OpC      Operand          Comment
Field   Field    Field            Field
-----   -----    -------          -------

        BE       HTYPI,HSTYPJ,LLB
        BLE      FTYPI,FTYPJ,LOCB
```

BASIC DECODE

BDCD

Type 3 Instruction

0100	1000	0110	0011
1	2	3	4

16 BITS

Detailed Description of Instruction Execution

This instruction assumes that R6 is pointing one byte before a compiler object code (COC) instruction. The instruction pointer R6 is incremented by one so that it points at the COC instruction operation code byte. The operation code may be followed by an operand. In any case, R6 will be incremented subsequently by the amount necessary to make it point one before the next COC instruction.

The one-byte operation code of the COC instruction is read. If the code is implemented in microcode, the firmware uses the code as an index into the branch table to get the address of a software routine to execute the operation. The firmware branches to that routine. If the code is implemented in microcode, th microcode branches to one of its own routines to execute it.

Refer to the Basic Runtime System for the codes that are implemented in microcode.

Operand Types	Object Code (Hex) 1 2 3 4
	4 8 i j

Example

Label Field	OpC Field	Operand Field	Comment Field
	BDCD	*	

BRANCH DECREMENTING BY ONE COMPARE

BDc Wj,L

Type 9 Instruction

1011	1100	Wjr	0000	Wjk	Wjd		
1	2	3	4	5	6	7	8

c	L-1		
9	10	11	12

48 BITS

Detailed Description of Instruction Execution

The code c can take on the following values:

	Complete Mnemonic	In the Instruction	Compare Relation
LZ	BDLZ	0001	LESS THAN
Z	BDZ	0010	EQUAL
HZ	BDHZ	0011	HIGHER THAN
HEZ	BDHEZ	0101	HIGHER THAN OR EQUAL
NZ	BDNZ	0110	UNEQUAL
LEZ	BDLEZ	0111	LESS THAN OR EQUAL

One is subtracted from C(C(Wjr)+Wjd) and this result replaces C(C(Wjr)+Wjd). Then C(C(Wjr)+Wjd) is tested for its relation to zero.

If the relation corresponds to the comparison code c, the instruction causes a branch to L to occur.

C(ACF) is NOT changed.

Operand Types	Wjk	Object Code (Hex) 1	2	3	4	5	6	7	8	9	0	1	2
Hj,L	0	B	C	j	0	0	d	d	d	c	1	1	1
Tj,L	1	B	C	j	0	1	d	d	d	c	1	1	1
Dj,L	2	B	C	j	0	2	d	d	d	c	1	1	1
Fj,L	3	B	C	j	0	3	d	d	d	c	1	1	1
Vj,L	7	B	C	j	0	7	d	d	d	c	1	1	1

BRANCH DECREMENTING BY ONE COMPARE (cont)

Example

Label Field	OpC Field	Operand Field	Comment Field
	BDZ	TTYPJ,LLB	
	BDLEZ	DTYPJ,LOCB	

BRANCH DECREMENTING ON ELEMENT COMPARE

BDc Wj,Wi,L

Type 11 Instruction

1111	0010	Wjr	Wir	Wjk	Wjd		
1	2	3	4	5	6	7	8

Wik	Wid			c	L-1			64 BITS
9	10	11	12	13	14	15	16	

Detailed Description of Instruction Execution

The code c can take on the following values:

	Complete Mnemonic	In the Instruction	Compare Relation
LZ	BDLZ	0001	LESS THAN
Z	BDZ	0010	EQUAL
HZ	BDHZ	0011	HIGHER THAN
HEZ	BDHEZ	0101	HIGHER THAN OR EQUAL
NZ	BDNZ	0110	UNEQUAL
LEZ	BDLEZ	0111	LESS THAN OR EQUAL

C(C(Wir)+Wid) is subtracted from C(C(Wjr)+Wjd) and this result replaces C(C(Wjr)+Wjd). Then C(C(Wjr)+Wjd) is tested for its relation to zero.

If the relation corresponds to the comparison code c, the instruction causes a branch to L to occur.

C(ACF) is NOT changed.

Operand Types	Wjk	Wik	Object Code (Hex) 1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6
Hj,Hi,L	0	0	F	2	j	i	0	d	d	d	0	d	d	d	c	1	1	1
Tj,Ti,L	1	1	F	2	j	i	1	d	d	d	1	d	d	d	c	1	1	1
Dj,Di,L	2	2	F	2	j	i	2	d	d	d	2	d	d	d	c	1	1	1
Fj,Fi,L	3	3	F	2	j	i	3	d	d	d	3	d	d	d	c	1	1	1
Vj,Vi,L	7	7	F	2	j	i	7	d	d	d	7	d	d	d	c	1	1	1

BRANCH DECREMENTING ON STORAGE COMPARE (cont)

Example

Label Field	OpC Field	Operand Field	Comment Field
	BDZ	TTYPJ,TSTYPI,LLB	
	BDLEZ	DTYPJ,DTYPI,LOCB	

Programming Note: If the first operand (Wj) is being decremented by one, use the BDc Wj,L instruction, which decrements by one implicitly.

BRANCH ON LOGICAL COMPARE

BL.c Wi,Wj,L

Type 11 Instruction

1	2	3	4	5	6	7	8
1111	0001	Wir	Wjr	Wik	Wid		
9	10	11	12	13	14	15	16
Wjk	Wjd			c	L-1		

64 BITS

Detailed Description of Instruction Execution

The code c can take on the following values:

	Complete Mnemonic	In the Instruction	Compare Relation
L	BL.L	1001	LESS THAN
E	BL.E	1010	EQUAL
H	BL.H	1011	HIGHER THAN
HE	BL.HE	1101	HIGHER THAN OR EQUAL
U	BL.U	1110	UNEQUAL
LE	BL.LE	1111	LESS THAN OR EQUAL

C(C(Wir)+Wid) is compared logically with C(C(Wjr)+Wjd). If the relationship matches the comparison code c, control transfers to L.

C(ACF) is NOT changed.

Operand Types	Wik	Wjk	Object Code (Hex) 1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6
Ci,Cj,L	0	0	F	1	i	j	0	d	d	d	0	d	d	d	c	1	1	1
Hi,Hj,L	0	0	F	1	i	j	0	d	d	d	0	d	d	d	c	1	1	1
Ti,Tj,L	1	1	F	1	i	j	1	d	d	d	1	d	d	d	c	1	1	1
Di,Dj,L	2	2	F	1	i	j	2	d	d	d	2	d	d	d	c	1	1	1
Fi,Fj,L	3	3	F	1	i	j	3	d	d	d	3	d	d	d	c	1	1	1
Vi,Vj,L	7	7	F	1	i	j	7	d	d	d	7	d	d	d	c	1	1	1

BRANCH ON LOGICAL COMPARE (cont)

Example

```
Label   OpC      Operand          Comment
Field   Field    Field            Field
-----   -----    -------          -------

        BL.E     TTYPI,TSTYPJ,LLB
        BL.L     DTYPI,DTYPJ,LOCB
```

Programming Note: The mnemonic form BCc Ci,Cj,L results in the same object code as BL.c Ci,Cj,L.

BRANCH AND STACK INDIRECT TO A MODAL ENTRY

BSLI

Type 1 Instruction

0000	0101	8 BITS
1	2	

Detailed Description of Instruction Execution

The address of the instruction following this instruction is pushed onto the return stack.

C(T0) is assumed to be a mode-id with the entry point number (Me) in bits 0-3 and the FID (Mf) in bits 4-15.

Control is transferred to frame Mf at location 1+(2*Me).

When this instruction is executed, register one is updated to point to byte zero of the ABS frame being entered.

Operand Types	Object Code (Hex) 1	2
	0	5

Example

Label Field	OpC Field	Operand Field	Comment Field
	BSLI		

Caution: This instruction generates a RETURN STACK FULL abort if RSCWA is incremented beyond the end of the stack.

BRANCH AND STACK TO LOCAL LOCATION

BSL L

Type 12 Instruction

0011	0001	0000	L-1			24 BITS
1	2	3	4	5	6	

Detailed Description of Instruction Execution

The address of the instruction following this instruction is pushed onto the return stack. Control is transferred to the instruction at L.

Operand Types	Object Code (Hex) 1	2	3	4	5	6
L	3	1	0	1	1	1

Example

Label Field	OpC Field	Operand Field	Comment Field
	BSL	LOCLAB	

Caution: This instruction generates a RETURN STACK FULL abort if RSCWA is incremented beyond the end of the stack.

BRANCH AND STACK TO A MODAL ENTRY

BSL M
BSL N,M

Type 12 Instruction

0011	0000	Me	Mf			24 BITS
1	2	3	4	5	6	

Detailed Description of Instruction Execution

The address of the instruction following this instruction is pushed onto the return stack.

M is a mode-id made up of an entry point number (Me) and a FID (Mf). Control transfers to frame Mf at location 1+(2*Me).

When this instruction is executed, register one is updated to point to byte zero of the ABS frame being entered.

Operand Types	Object Code (Hex) 1 2 3 4 5 6
M	3 0 e f f f
N,M	3 0 n f f f

Where e,n are entry point numbers and fff are the 3 nibbles (12 bits) of the FID (Mf). When N is given, n overrides e.

Example

Label Field	OpC Field	Operand Field	Comment Field
MTYPJ	DEFM	12,128	
	BSL	MTYPJ	
	BSL	6,MTYPJ	

Caution: This instruction generates a RETURN STACK FULL abort if RSCWA is incremented beyond the end of the stack.

BRANCH ON STRING COMPARE

BSTc Ri,Rj,N,L

Type 10 Instruction

0100	1100	Ri	Rj	N		c
1	2	3	4	5	6	7

L-1		
8	9	10

40 BITS

Detailed Description of Instruction Execution

The code c can take on the following values:

	Complete Mnemonic	In the Instruction	Compare Relation
E	BCE	1010	EQUAL
U	BCU	1110	UNEQUAL

This instruction may be used to compare two strings. Each string must end with a delimiter.

C(Ri) and C(Rj) are incremented by 1.

If C(C(Ri)) and C(C(Rj)) are equal and less than N (delimiter character), the instruction increments the registers and compares again.

If the contents of both bytes are greater than or equal to N, the strings are considered equal and the instruction branches or not depending on the comparison code c.

If the contents of both bytes are less than N but unequal to each other, the strings are considered unequal and the instruction branches or not depending on the comparison code c.

Operand Types	Object Code (Hex) 1 2 3 4 5 6 7 8 9 0
Ri,Rj,N,L	4 C i j n n c 1 1 1

BRANCH ON STRING EQUAL (cont)

Example

Label Field	OpC Field	Operand Field	Comment Field
	BSTE	R3,R5,X'FC',LOCLAB	

Programming Note: This instruction can be used in conjunction with the BRANCH CHARACTER LOW instruction to compare two strings:

```
BSTE   R14,R15,M,EQUAL

BCL    R14,R15,LOW

B      HIGH
```

where M is a delimiter value that is defined elsewhere. EQUAL, LOW, and HIGH are local labels.

The first instruction compares strings. If the delimiter is encountered, the strings are equal.

The second instruction compares the two bytes that were found unequal by the first instruction.

COMPARE STRINGS

COMP Ri,Rj,L

Type 8 Instruction

0100	1101	Ri	Rj	0000	L-1			32 BITS
1	2	3	4	5	6	7	8	

Detailed Description of Instruction Execution

This instruction compares two strings until it finds two unequal characters or until it finds a delimiter in one or both strings.

Before instruction execution C(Ri) and C(Rj) must point one byte before their respective strings. C(Ri) and C(Rj) are each incremented by one. If C(C(Ri)) and C(C(Rj)) are equal and not delimiters, the registers are again incremented and the next characters compared. This process continues until the characters are unequal or until a delimiter is found in one or both strings.

The instruction recognizes three classes of delimiters. The first class comprises all the values in the range X'FB'-X'FF'. Any of these values signal the end of the string.

The next delimiter class is the value X'F9'. This delimiter tells the firmware two things: Numeric characters follow, and they are being sorted in ascending sequence.

The third class of delimiter is the value X'FA'. It tells the firmware that numeric characters follow but they are being sorted in descending sequence.

If the corresponding characters from each string are both numeric delimiters (X'F9' or X'FA'), the instruction branches to location L. If only one of the characters is a numeric delimiter, the branch is not taken. The instruction does not convert the numeric characters following an X'F9' or X'FA' delimiter. Also, the instruction does not look at a character to see if it is numeric. The X'F9' or X'FA' delimiter must be inserted in the string by the software before the COMP instruction is executed.

When the branch is taken, the software knows that both delimiters are numeric. Otherwise, the instruction does not inform the software that a numeric delimiter was encountered .

When the branch is taken, the ACF is not changed. When the instruction falls through, ACF OVFBIT is zeroed, NUMBIT is undefined, and EQUBIT and LOWBIT are set as summarized in Figure A.

COMPARE STRINGS (cont)

If String 1 C(C(Ri)) is	AND String 2 C(C(Rj)) is	Then it means that	and it sets EQUBIT	LOWBIT
F9-FA	F9-FA	Both are numerics. Branch is taken.	N/C	N/C
FB-FF	FB-FF	String 1 = String 2	1	0
<F9 and >C(C(Rj))	<F9	String 1 > String 2	0	0
<F9 and <C(C(Rj))	<F9	String 1 < String 2	0	1
F9-FA	FB-FF	String 1 > String 2 String 1 is longer.	0	0
FB-FF	F9-FA	String 1 < String 2 String 2 is longer.	0	1
FB-FF	<F9	String 1 < String 2 String 2 is longer.	0	1
<F9	FB-FF	String 1 > String 2 String 1 is longer.	0	0
F9	<30	String 1 > String 2 Numeric is greater, ascending.	0	0
F9	<F9 and >=30	String 1 < String 2 Numeric is less, ascending.	0	1
FA	<D0	String 1 > String 2 Numeric is greater, descending.	0	0
FA	<F9 and >=D0	String 1 < String 2 Numeric is less, descending.	0	1
<30	F9	String 1 < String 2 Numeric is greater, ascending	0	1
<F9 and >=30	F9	String 1 > String 2 Numeric is less, ascending	0	0
<D0	FA	String 1 < String 2 Numeric is greater, descending	0	1
<F9 and >=D0	FA	String 1 > String 2 Numeric is less, descending	0	0

Figure A. Summary of COMP Instruction Results

COMPARE STRINGS (cont)

Operand Types	Object Code (Hex) 1 2 3 4 5 6 7 8
Ri,Rj,L	4 D i j 0 1 1 1

Example

```
Label  OpC      Operand          Comment
Field  Field    Field            Field
-----  -----    -------          -------

       COMP     R14,R15,NUMST    COMPARE STRINGS
       BBS      EQUBIT,EQST      BRANCH IF EQUAL
       BBS      LOWBIT,LEST      BRANCH IF STRING 1 LESS
```

Programming Note: This instruction was designed to speed up the ENGLISH SORT verb. For an ascending, left-justified sort, the verb software puts an X'FF' after each string but does not change it otherwise. Specifically, the X'F9' delimiter is not used. For an ascending, right-justified sort, the software puts X'F9' before each string of numeric characters. When this delimiter is found in both strings, the software converts the numerics to binary and sorts then by value. At this writing, the SORT verb does not use the X'FA' delimiter.

For a descending sort the software uses the two's complement of each character value for the sort. Since the software does not use the X'FA' delimiter, a descending, right-justified sort of numerics is not the reverse of an ascending, right-justified sort.

DISABLE BASIC DEBUGGER

DBDB

Type 1 Instruction

0010	0110	8 BITS
1	2	

Detailed Description of Instruction Execution

This instruction resets the bits set by EBDB to disable traps to the Basic Debugger when the BDCD instruction encounters one of the following compiler object code instrucions:

01 EOL

06 BRANCH

Operand Types	Object Code (Hex) 1	2
	2	6

Example

Label Field	OpC Field	Operand Field	Comment Field
	DBDB		DISABLE D/B DEBUGGER TRAPS

Programming Note: This instruction along with EBDB replaces the use of bit DFLG to indicate a trap to the Basic Debugger.

DECODE

DCDRR Ri,Rj
DCD (The assembler's OSYM uses R6 for Ri and R3 for Rj)

Type 3 Instruction

0100	0011	Ri	Rj
1	2	3	4

16 BITS

Detailed Description of Instruction Execution

This instruction assumes that Ri is pointing one byte before a compiler object code (COC) instruction. The instruction pointer Ri is incremented by one so that it points at the COC instruction operation code byte. The operation code may be followed by an operand. In any case, Ri will be incremented subsequently by the amount necessary to make it point one before the next COC instruction.

The one-byte operation code of the COC instruction is read. The firmware uses the code as an index into the branch table to get the address of a software routine to execute the operation. The firmware branches to that routine. The table address is in double tally DCDFID in the PCB.

Operand Types	Object Code (Hex) 1 2 3 4
Ri,Rj	4 3 i j

Example

Label Field	OpC Field	Operand Field	Comment Field
	DCD		
	DCDR	R12,R15	

DECREMENT ADDRESS REGISTER BY ONE

DEC Rj

Type 3 Instruction

0000	0110	Rj	0000
1	2	3	4

16 BITS

Detailed Description of Instruction Execution

If Rj is attached, C(Rj) is decremented by one. If Rj is detached, the displacement field of Rj is decremented by one; then the register is attached.

Operand Types	Object Code (Hex) 1 2 3 4
Rj	0 6 j 0

Example

Label Field	OpC Field	Operand Field	Comment Field
	DEC	R14	

In the linked format, if the resulting memory address is XY17 (XY represents any even hexadecimal number), that is, if the address points to byte 23 of a buffer:

1. If the backward link of the current frame is zero, C(R) remains attached to data byte zero of the current frame;

2. Otherwise, an attempt is made to attach C(R) to the last data byte of the frame pointed to by the backward link of the current frame. The "REFERENCING ILLEGAL FRAME" debug trap could occur in this case.

3. If C(Rj) is decremented again so that the resulting memory address is XY16, a "BACKWARD LINK ZERO" debug trap will occur.

Programming Note: Rj is always attached when this instruction completes execution.

DECREMENT ADDRESS REGISTER

DEC Rj,Wi

Type 5 Instruction

1011	0110	Wir	Rj	Wik	Wid			32 BITS
1	2	3	4	5	6	7	8	

Detailed Description of Instruction Execution

If Rj is attached, the memory address field of Rj is decremented by C(C(Wir)+Wid). If the resulting address crosses the frame boundary, the register is detached. If Rj is initially detached, the displacement portion of Rj is decremented by C(C(Wir)+Wid).

Operand Types	Wik	Object Code (Hex) 1	2	3	4	5	6	7	8
Rj,Hi	1	B	6	i	j	0	d	d	d
Rj,Ti	1	B	6	i	j	1	d	d	d
Rj,Di	1	B	6	i	j	2	d	d	d
Rj,Fi	1	B	6	i	j	3	d	d	d

Example

Label Field	OpC Field	Operand Field	Comment Field
	DEC	R14,TTYPJ	

Programming Note: This instruction may cause an attached register to detach. It will never cause a detached register to attach.

SUBTRACT ONE FROM ELEMENT

DEC Wj

Type 5 Instruction

1010	0110	Wjr	0000	Wjk	Wjd			32 BITS
1	2	3	4	5	6	7	8	

Detailed Description of Instruction Execution

C(C(Wjr)+Wjd) is decreased by one.

C(ACF) is NOT changed.

Operand Types	Wjik	Object Code (Hex) 1 2 3 4 5 6 7 8
Hj	0	A 6 r 0 0 d d d
Tj	1	A 6 r 0 1 d d d
Dj	2	A 6 r 0 2 d d d
Fj	3	A 6 r 0 3 d d d
Vj	7	A 6 r 0 7 d d d

Example

Label Field	OpC Field	Operand Field	Comment Field
	DEC	DTYPJ	DECREMENT DTYPJ BY 1

SUBTRACT ELEMENT FROM ELEMENT

DEC Wj,Wi

Type 6 Instruction

1	2	3	4	5	6	7	8
1111	0110	Wjr	Wir	Wjk	Wjd		

9	10	11	12
Wik	Wid		

48 BITS

Detailed Description of Instruction Execution

C(C(Wir)+Wid) is subtracted from C(C(Wjr)+Wjd). The difference replaces C(C(Wjr)+Wjd).

C(ACF) is NOT changed.

Operand Types	Wjk	Wik	Object Code (Hex) 1	2	3	4	5	6	7	8	9	0	1	2
Hj,Hi	0	0	F	6	j	i	0	d	d	d	0	d	d	d
Tj,Ti	1	1	F	6	j	i	1	d	d	d	1	d	d	d
Dj,Di	2	2	F	6	j	i	2	d	d	d	2	d	d	d
Fj,Fi	3	3	F	6	j	i	3	d	d	d	3	d	d	d
Vj,Vi	7	7	F	6	j	i	7	d	d	d	7	d	d	d

Example

Label Field	OpC Field	Operand Field	Comment Field
	DEC	HTYPJ,HTYPI	
	DEC	STYPJ,STYPI	

DIVIDE

DIV Wi

Type 5 Instruction

1010	1001	Wir	0000	Wik	Wid			32 BITS
1	2	3	4	5	6	7	8	

Detailed Description of Instruction Execution

C(D0) is divided by C(C(Wir)+Wid). A 4-byte quotient replaces C(D0); a 4-byte remainder replaces C(D1).

The sign of the quotient is determined by the rules of algebra. The sign of the remainder is the sign of the dividend.

C(ACF) is changed: OVFBIT is set only when a divide by zero is attempted. It is not set for any other overflow condition.

Operand Types	Wik	Object Code (Hex) 1 2 3 4 5 6 7 8
Hi	0	A 9 i 0 0 d d d
Ti	1	A 9 i 0 1 d d d
Di	2	A 9 i 0 2 d d d
Vi	7	A 9 i 0 7 d d d

Example

Label Field	OpC Field	Operand Field	Comment Field
	DIV	TTYPJ	D0/TTYPJ
	DIV	FTYPI	FP0/FTYPI

Programming Note: Data can be moved out of FPY using a Move Storage to Storage instruction.

Programming Note: The mnemonic DIV may be used with F type operands (six-byte elements), but the object code generated by the assembler is the same as that generated for instruction DIVX.

DIVIDE EXTENDED

DIVX Wi

Type 5 Instruction

1010	1000	Wir	0000	Wik	Wid			32 BITS
1	2	3	4	5	6	7	8	

Detailed Description of Instruction Execution

C(FP0) is divided by C(C(Wir)+Wid). A 6-byte quotient replaces C(FP0); a 6-byte remainder replaces C(FPY).

The sign of the quotient is determined by the rules of algebra. The sign of the remainder is the sign of the dividend.

C(ACF) is changed: OVFBIT is set only when a divide by zero is attempted. It is not set for any other overflow condition.

Operand Types	Wik	Object Code (Hex) 1 2 3 4 5 6 7 8
Hi	0	A 8 i 0 0 d d d
Ti	1	A 8 i 0 1 d d d
Di	2	A 8 i 0 2 d d d
Fi	3	A 8 i 0 3 d d d
Vi	7	A 8 i 0 7 d d d

Programming Note: Data can be moved out of FPY by using a Move Storage to Storage instruction.

DIVIDE DOUBLE-EXTENDED

DIVXX Fi

Type 5 Instruction

1010	1100	Fir	0000	Fik	Fid			32 BITS
1	2	3	4	5	6	7	8	

Detailed Description of Instruction Execution

This instruction divides a ten-byte binary number by C(C(Fir)+Fid). The most significant four bytes of the number must be in the four low-order bytes of FPY; the least significant six bytes must be in FP0. After the division the quotient is placed in FP0 and the remainder in FPY.

The sign of the quotient is determined by the rules of algebra. The sign of the remainder is the sign of the dividend.

C(ACF) is changed: OVFBIT is set only when a divide by zero is attempted. It is not set for any other overflow condition.

Operand Types	Fik	Object Code (Hex) 1 2 3 4 5 6 7 8
Fi	3	A C i 0 3 d d d

Example

Label Field	OpC Field	Operand Field	Comment Field
	DIVXX	FTYPJ	

DEQUEUE I/O REQUEST

DQIO Rj,N

Type 4 Instruction

0101	1111	Rj	0000	N	
1	2	3	4	5	6

24 BITS

0000 1 1
'09'

Detailed Description of Instruction Execution

This instruction retrieves a completed task descriptor (TD) from its Completions Queue, returning the virtual storage address of the TD in Rj.

N is an eight-bit literal defining the dequeue criteria as follows:

Bit	Meaning
0-3	Undefined;
4	0 = Don't wait if request is not satisfied (No-Wait option); 1 = Wait for completion (Wait option);
5-6	Undefined;
7	0 = Dequeue first TD on queue (Dequeue option); 1 = Search queue for matching Stream number (Stream option).

No-Wait Option

When executed with the No-Wait option, the instruction always completes immediately with the status code in H0 indicating whether or not a TD was actually dequeued as follows:

Code (Hex)	Meaning
00	TD successfully dequeued;
01	Completions Queue empty;
11	No TD on queue with requested Stream number.

DEQUEUE I/O REQUEST (cont)

Wait Option

With the Wait option this instruction completes immediately if there is a suitable TD on the Completions Queue. If there is no suitable TD on the queue, the instruction roadblocks the process by setting the IOWAIT/ roadblock (removing the process from the Priority Queue) until a completed TD is queued onto the process's Completions Queue. The effect of this will be that the process will be re-activated executing the same DQIO instruction.

A process can be roadblocked indefinitely if the instruction is executed with the Wait option in the following situations:

N specifies the Dequeue option, but the queue is empty and there are no outstanding TDs;

N specifies the Search option, but there are no TDs with the specified Stream number in the queue or outstanding.

Dequeue Option

The Dequeue option dequeues the first TD on the Completions Queue.

Stream Option

When the Stream option is specified, the Stream number to search for must be specified in H0.

Operand Types	Object Code (Hex) 1	2	3	4	5	6
Rj,N	5	F	j	0	n	n

Example

Label Field	OpC Field	Operand Field	Comment Field
	DQIO	R3,X'0'	GET NEXT TD

ENABLE BASIC DEBUGGER

EBDB

Type 1 Instruction

0010	0101	8 BITS
1	2	

Detailed Description of Instruction Execution

This instruction sets a bit in the PIB and a hardware bit directly testable by microcode to cause a trap to the Basic Debugger when the BDCD instruction encounters one of the following compiler object code instrucions:

01 EOL

06 BRANCH

Operand Types	Object Code (Hex) 1	2
	2	5

Example

Label Field	OpC Field	Operand Field	Comment Field
	EBDB		ENABLE D/B DEBUGGER TRAPS

Programming Note: This instruction along with DBDB replaces the use of bit DFLG to indicate a trap to the Basic Debugger.

EXTERNAL BRANCH INDIRECT TO A MODAL ENTRY

ENTI

Type 1 Instruction

0000	0100
1	2

8 BITS

Detailed Description of Instruction Execution

C(T0) is assumed to be a Mode-ID with the entry point number (Me) in bits 0-3 and the FID (Mf) in bits 4-15.

Control is transferred to the specified frame (Mf) at location 1+(2*Me).

Operand Types	Object Code (Hex) 1	2
	0	4

Example

Label Field	OpC Field	Operand Field	Comment Field
	ENTI		

EXTERNAL BRANCH TO A MODAL ENTRY

ENT M

Type 12 Instruction

0011	0100	Me	Mf			24 BITS
1	2	3	4	5	6	

Detailed Description of Instruction Execution

Control is transferred to the mode with FID Mf at location 1+(2*Me).

Operand Types	Object Code (Hex) 1 2 3 4 5 6
M	3 4 e f f f

Where e is the entry point number and fff are the 3 nibbles (12 bits) of the FID (Mf).

Example

Label Field	OpC Field	Operand Field	Comment Field
MTYPJ	DEFM	12,128	
	ENT	MTYPJ	

Programming Note: The assembler also recognizes the mnemonic format

ENT N,M 1 0 n f f f

where N overrides the Me for M. The opcode and Mf are the same, however. Execution of the object code instruction is also the same.

HALT

HALT

Type 1 Instruction

0000	1000	8 BITS
1	2	

Detailed Description of Instruction Execution

This instruction causes a trap to a debugger.

The instruction traps to the Software Debugger, which displays the message "HALT INSTRUCTION."

Operand Types	Object Code (Hex) 1	2
	0	8

Example

Label Field	OpC Field	Operand Field	Comment Field
	HALT		

INCREMENT ADDRESS REGISTER BY ONE

INC Rj

Type 3 Instruction

0000	0111	Rj	0000
1	2	3	4

16 BITS

Detailed Description of Instruction Execution

If Rj is attached, C(Rj) is incremented by one. If Rj is detached, the displacement field of Rj is incremented by one; then the register is attached.

Operand Types	Object Code (Hex) 1 2 3 4
Rj	0 7 j 0

Example

Label Field	OpC Field	Operand Field	Comment Field
	INC	R15	

Caution: If the resulting memory address is not in the same buffer, then either:

1. A "CROSSING FRAME LIMIT" debug trap occurs if C(Rj) is in unlinked format; or

2. An attempt is made to attach C(Rj) to the first data byte of the frame pointed to by the forward link of the current frame. In this case, "FORWARD LINK ZERO" and "REFERENCING ILLEGAL FRAME" are debug traps that could occur.

Programming Note: Rj is always attached when this instruction completes execution.

INCREMENT ADDRESS REGISTER

INC Rj,Wi

Type 5 Instruction

1011	0111	Wir	Rj	Wik	Wid			32 BITS
1	2	3	4	5	6	7	8	

Detailed Description of Instruction Execution

If R is attached, the memory address field of Rj is incremented by C(C(Wir)+Wid). If the resulting address crosses the frame boundary, the register is detached. If Rj is initially detached, the displacement portion of Rj is incremented by C(C(Wir)+Wid).

Operand Types	Wik	Object Code (Hex) 1 2 3 4 5 6 7 8
Rj,Hi	1	B 7 i j 0 d d d
Rj,Ti	1	B 7 i j 1 d d d
Rj,Di	1	B 7 i j 2 d d d
Rj,Fi	1	B 7 i j 3 d d d

Example

Label Field	OpC Field	Operand Field	Comment Field
	INC	R13,TTYPI	

Programming Note: This instruction may cause an attached register to detach. It will never cause a detached register to attach.

ADD ONE TO ELEMENT

INC Wj

Type 5 Instruction

1010	0111	Wjr	0000	Wjk	Wjd			32 BITS
1	2	3	4	5	6	7	8	

Detailed Description of Instruction Execution

C(C(Wjr)+Wjd) is increased by one.

Operand Types	Wik	Object Code (Hex) 1 2 3 4 5 6 7 8
Hj	0	A 7 j 0 0 d d d
Tj	1	A 7 j 0 1 d d d
Dj	2	A 7 j 0 2 d d d
Fj	3	A 7 j 0 3 d d d
Vj	7	A 7 j 0 7 d d d

Example

Label Field	OpC Field	Operand Field	Comment Field
	INC	DTYPJ	BUMP DTYPJ BY 1

ADD ELEMENT TO ELEMENT

INC Wj,Wi

Type 6 Instruction

1111	0111	Wjr	Wir	Wjk	Wjd		
1	2	3	4	5	6	7	8

Wik	Wid		
9	10	11	12

48 BITS

Detailed Description of Instruction Execution

C(C(Wir)+Wid) is added to C(C(Wjr)+Wjd). The sum replaces C(C(Wjr)+Wjd).

Operand Types	Wjk	Wik	Object Code (Hex) 1	2	3	4	5	6	7	8	9	0	1	2
Hj,Hi	0	0	F	7	j	i	0	d	d	d	0	d	d	d
Tj,Ti	1	1	F	7	j	i	1	d	d	d	1	d	d	d
Dj,Di	2	2	F	7	j	i	2	d	d	d	2	d	d	d
Fj,Fi	3	3	F	7	j	i	3	d	d	d	3	d	d	d
Sj,Si	3	3	F	7	j	i	3	d	d	d	3	d	d	d
Vj,Vi	7	7	F	7	j	i	7	d	d	d	7	d	d	d

Example

```
Label   OpC      Operand          Comment
Field   Field    Field            Field
-----   -----    -------          -------

        INC      HTYPJ,HTYPI
        INC      STYPJ,STYPI
```

LOAD ADDRESS DIFFERENCE

LAD Si,Rj
LAD Rj,Si

Type 5 Instruction

1011	1110	Sir	Rj	Sik	Sid			32 BITS
1	2	3	4	5	6	7	8	

Detailed Description of Instruction Execution

This instruction subtracts C(C(Sir)+Sid) from C(Rj), and the difference replaces C(TO). The subtraction is done in the following manner. The detached form of C(Rj) is calculated. The C(C(Sir)+Sid) is treated as a storage address (2 bytes of displacement, 1 link byte, 3-byte FID). For unlinked frames both operands must reference the same FID: The difference between the displacements is the result in this case. The instruction is valid for unequal frame numbers only if both frames are in the same group of contiguously linked frames, and the difference between the FIDs is equal to or less than 32. For contiguously linked frames the result is calculated by multiplying the difference between the FIDs by 1000 and adding to that product the difference between the displacements.

Operand Types	Wik	Object Code (Hex) 1	2	3	4	5	6	7	8
Si,Rj	3	B	E	i	j	3	d	d	d
Rj,Si	3	B	E	i	j	3	d	d	d

Example

Label Field	OpC Field	Operand Field	Comment Field
	LAD	STYPJ,R14	

LOAD ABSOLUTE

LOADA **A**

Type 13a Instruction

0011	1111	Ak	A					32 BITS
1	2	3	4	5	6	7	8	

Detailed Description of Instruction Execution

C(A) replaces C(D0) for a 1, 2 or 4-byte operand.

For a 6-byte operand, C(FP0) is replaced by C(A).

Operand Types	Aik	Object Code (Hex) .1	2	3	4	5	6	7	8
Hi	0	3	F	0	a	a	a	a	a
Ti	1	3	F	1	a	a	a	a	a
Di	2	3	F	2	a	a	a	a	a
Fi	3	3	F	3	a	a	a	a	a

Example

Label Field	OpC Field	Operand Field	Comment Field
	LOADA	@ALOC	LOAD FROM ABSOLUTE

Caution: Remember that a 6-byte operand affects the 6-byte accumulator FP0. Shorter operands affect only D0, the 4-byte accumulator.

Note: Only items defined in PSYM as '@' types can be used as operands.

LOAD ACCUMULATOR

LOAD Wi

Type 5 Instruction

1010	1111	Wir	0000	Wik	Wid			32 BITS
1	2	3	4	5	6	7	8	

Detailed Description of Instruction Execution

The integer addressed by the operand is loaded into the 32-bit accumulator (D0). That is, C(C(Wir)+Wid) replaces C(D0). For half tally and tally operands, the sign bit is extended.

Operand Types	Wik	Object Code (Hex) 1 2 3 4 5 6 7 8
Hi	0	A F i 0 0 d d d
Ti	1	A F i 0 1 d d d
Di	2	A F i 0 2 d d d
Vi	7	A F i 0 7 d d d

Example

```
Label   OpC      Operand          Comment
Field   Field    Field            Field
-----   -----    -------          -------

TTYPJ   DEFT     14,8             TALLY TYPE J
        LOAD     TTYPJ            LOAD D0
```

Programming Note: The mnemonic LOAD may be used with F type operands (six-byte elements), but the object code generated by the assembler is the same as that generated for instruction LOADX.

LOAD EXTENDED ACCUMULATOR

LOADX Wi

Type 5 Instruction

1010	1110	Wir	0000	Wik	Wid			32 BITS
1	2	3	4	5	6	7	8	

Detailed Description of Instruction Execution

The integer addressed by the operand is loaded into the 48-bit accumulator (FP0), and the sign bit is extended. That is, C(C(Wir)+Wid) replaces C(FP0).

Operand Types	Wik	Object Code (Hex) 1	2	3	4	5	6	7	8
Hi	0	A	E	i	0	0	d	d	d
Ti	1	A	E	i	0	1	d	d	d
Di	2	A	E	i	0	2	d	d	d
Fi	3	A	E	i	0	3	d	d	d
Vi	7	A	E	i	0	7	d	d	d

Example

```
Label   OpC      Operand        Comment
Field   Field    Field          Field
-----   -----    -------        -------

HTYPI   DEFH     15,5           HALF TALLY TYPE I
        LOADX    HTYPI          MOVE VALUE TO FP0
```

LOCK PROCESSESSAND INCREMENT INHIBITH

LOCKINH N

Type 2 Instruction

0001	0011	N	
1	2	3	4

16 BITS

Detailed Description of Instruction Execution

This instruction performs the same actions as the LOCK N instruction with the addition that if the lock is locked successfully, INHIBITH is incremented by one.

Operand Types	Object Code (Hex) 1	2	3	4
N	1	3	n	n

Example

Label Field	OpC Field	Operand Field	Comment Field
	LOCKINH	OVRFLW*	

LOCK COMPETING PROCESSES FOR SYSTEM RESOURCE

LOCK N

Type 2 Instruction

0001	0010	N	
1	2	3	4

16 BITS

Detailed Description of Instruction Execution

This instruction enables processes to compete for a system resource by means of a lock. The first process to set the lock may use the resource. Other processes that attempt to set the lock will be roadblocked until the first process opens the lock with the UNLOCK N instruction. UNLOCK N removes the roadblock from the first process in the priority queue that is waiting on the lock.

N is the lock number.

There are three conditions handled by this instruction.

If lock N contains the unlocked value, the firmware stores th executing process's number in lock N, and instruction execution ends: The process has control of the resource.

If lock N already contains the process number of the executing process, instruction execution ends: The process retains control of the resource.

If lock N contains the process number of another process, the instruction deposits the value N in the executing process's PIB and enters the Monitor. The process is roadblocked until the lock is opened by the UNLOCK N instruction.

Operand Types	Object Code (Hex) 1	2	3	4
N	1	2	n	n

Example

Label Field	OpC Field	Operand Field	Comment Field
	LOCK	OVRFLW*	

LOCK COMPETING PROCESSES (cont)

Programming Note: This instruction does not know the resources that are being locked. The correlation of a lock with a resource is a software convention. In fact there is nothing in the system to prevent a process from using a resource without using this instruction.

If more than one lock must be locked at one time, there should also be a software convention that specifies the order in which the locks should be locked. This avoids the problem of two processes roadblocking each other by locking two locks in different order.

LOCK COMPETING PROCESS FOR SOFTWARE RESOURCE

LOCK Tj

Type 5 Instruction

1011	1000	Tjr	0000	Tjk	Tjd			32 BITS
1	2	3	4	5	6	7	8	

Detailed Description of Instruction Execution

This instruction enables processes to compete for a software resource by means of a lock. The first process to close the lock may use the resource; other processes that attempt to set the lock are deactivated.

The instruction sets C(C(Tjr)+Tjd) with a value that indicates that the lock is closed.

You unlock the lock by means of the SET instruction, that is,

SET Tj

Operand Types	Tjk	Object Code (Hex) 1 2 3 4 5 6 7 8
Tj	1	A A j 0 1 d d d

Example

Label Field	OpC Field	Operand Field	Comment Field
	LOCK	TTYPJ	LOCK A SOFTWARE RESOURCE
	.		
	.		
	.		
	SET	TTYPJ	UNLOCK IT

LOAD PIB ADDRESS TO REGISTER

LPIB Rj

Type 3 Instruction

0011	1010	Rj	0000
1	2	3	4

16 BITS

Detailed Description of Instruction Execution

This instruction assumes that C(T0) is a process number. The instruction replaces C(Rj) with that process's PIB address. The instruction also returns the PIB software flag byte in H1 and one of the following codes in H0:

Code	Meaning
0	Successful
1	No PIB pointer
2	Invalid process number
3	No FID allocated to PIB

Operand Types	Object Code (Hex) 1 2 3 4
Rj	3 A j 0

Example

Label Field	OpC Field	Operand Field	Comment Field
	LPIB	R15	

MOVE BINARY TO DECIMAL

MBD Ti,Rj

Type 5 Instruction

1101	1011	Tir	Rj	Tik	Tid			32 BITS
1	2	3	4	5	6	7	8	

Detailed Description of Instruction Execution

This instruction assumes that the low-order byte of C(C(Tir)+Tid) contains a binary number with a value in the range 0 to 9. The instruction converts the binary number to an ASCII character, which replaces the C(C(Rj)). C(C(Tir)+Tid) is not changed.

Operand Types	Tik	Object Code (Hex) 1 2 3 4 5 6 7 8
Ti,Rj	1	D B i j 1 d d d

Example

Label Field	OpC Field	Operand Field	Comment Field
	MBD	TTYPI,R14	

MOVE BINARY TO HEXADECIMAL STRING

MBX Wi,Rj

Type 5 Instruction

1011	1111	Wir	Rj	Wik	Wid			32 BITS
1	2	3	4	5	6	7	8	

Detailed Description of Instruction Execution

The binary integer which is C(C(Wir)+Wid) is converted into an ASCII string of hexadecimal numbers starting at C(Rj)+1. Bits 4-7 of H0 (the low-order byte of D0) contain a count of the maximum number of ASCII bytes to be generated. If bit zero of H0 is a zero, the leading zeros of the hexadecimal string are suppressed; that is, they are not moved to the output string. if bit zero of H0 is a one, zero suppression will not take place. C(Rj) is incremented before each hexadecimal character is stored. C(H0) is unpredictable after this instruction is executed. If the digit count in H0 exceeds the size allowed, no operation is performed.

The maximum digit count is twice the size of the first operand: 12 for S, 8 for D, 4 for T, etc. A count of zero is not allowed.

Operand Types	Wik	Object Code (Hex) 1	2	3	4	5	6	7	8
Ci,Rj	0	B	F	i	j	0	d	d	d
Hi,Rj	0	B	F	i	j	0	d	d	d
Ti,Rj	1	B	F	i	j	1	d	d	d
Di,Rj	2	B	F	i	j	2	d	d	d
Fi,Rj	3	B	F	i	j	3	d	d	d

Example

```
Label   OpC      Operand          Comment
Field   Field    Field            Field
-----   -----    -------          -------

=H6     HTLY     6
        MOV      =H6,H0
        MBX      FTYPI,R15
```

Programming Note: At the conclusion of instruction execution Rj points to the last character converted.

MOVE RELATIVE CHARACTER TO CHARACTER

MCC Ci,Rj

Type 5 Instruction

1101	0110	Cir	Rj	Cik	Cid			32 BITS
1	2	3	4	5	6	7	8	

Detailed Description of Instruction Execution

C(C(Cir)+Cid) replaces C(C(Rj)).

Operand Types	Cik	Object Code (Hex) 1 2 3 4 5 6 7 8
Ci,Rj	0	D 6 i j 0 d d d

Example

Label Field	OpC Field	Operand Field	Comment Field
	MCC	CTYPI,R15	

MOVE IMMEDIATE CHARACTER

MCC N,Rj

Type 4 Instruction

0101	0110	Rj	0000	N	
1	2	3	4	5	6

24 BITS

Detailed Description of Instruction Execution

N replaces C(C(Rj)).

Operand Types	Object Code (Hex) 1 2 3 4 5 6
N,Rj	5 6 j 0 n n

Example

Label Field	OpC Field	Operand Field	Comment Field
	MCC	6,R13	

MOVE CHARACTER TO RELATIVE CHARACTER

MCC Ri,Cj

Type 5 Instruction

1	2	3	4	5	6	7	8	
1100	0110	Cjr	Ri	Cjk	Cjd			32 BITS

Detailed Description of Instruction Execution

C(C(Ri)) replaces C(C(Cjr)+Cjd).

Operand Types	Cik	Object Code (Hex) 1 2 3 4 5 6 7 8
Ri,Cj	0	C 6 j i 0 d d d

Example

Label Field	OpC Field	Operand Field	Comment Field
	MCC	R14,HTYPJ	

MOVE CHARACTER TO CHARACTER

MCC Ri,Rj

Type 3 Instruction

1110	0000	Ri	Rj
1	2	3	4

16 BITS

Detailed Description of Instruction Execution

C(C(Ri)) replaces C(C(Rj)).

Operand Types	Object Code (Hex) 1 2 3 4
Ri,Rj	E 0 i j

Example

Label Field	OpC Field	Operand Field	Comment Field
	MCC	R7,R14	

INCREMENT DESTINATION REGISTER AND MOVE IMMEDIATE CHARACTER

MCI N,Rj

Type 4 Instruction

0100	0110	Rj	0000	N	
1	2	3	4	5	6

24 BITS

Detailed Description of Instruction Execution

C(Rj) is incremented by one. Then N replaces C(C(Rj)).

Operand Types	Object Code (Hex) 1	2	3	4	5	6
N,Ri	4	6	j	0	n	n

Example

Label Field	OpC Field	Operand Field	Comment Field
	MCI	C'K',R15	

MOVE IMMEDIATE CHARACTER UNDER REGISTER CONTROL

MCI N,Ri,Rj

Type 4 Instruction

0101	0101	Ri	Rj	N	
1	2	3	4	5	6

24 BITS

Detailed Description of Instruction Execution

C(Ri) and C(Rj) are compared. If C(Ri) and C(Rj) are equal, instruction execution is finished. If not, C(Ri) is incremented by one and N replaces C(C(Ri)). The process repeats until C(Ri) equals C(Rj).

Operand Types	Object Code (Hex) 1 2 3 4 5 6
N,Ri,Rj	5 5 i j n n

Example

```
Label   OpC     Operand         Comment
Field   Field   Field           Field
-----   -----   -------         -------

        MCI     0,R14,R15
        MCI     C' ',R14,R15
```

Programming Note: This instructions compares the addresses contained in the two registers before incrementing and moving the data. This is different from the string instructions, which increment before comparing.

INCREMENT DESTINATION REGISTER AND MOVE CHARACTER

MCI Ri,Rj

Type 3 Instruction

1000	0101	Ri	Rj
1	2	3	4

16 BITS

Detailed Description of Instruction Execution

C(Rj) is incremented by one. C(C(Ri)) replaces C(C(Rj)).

Operand Types	Object Code (Hex) 1	2	3	4
Ri,Rj	8	5	i	j

Example

Label Field	OpC Field	Operand Field	Comment Field
	MCI	R5,R14	

MOVE DECIMAL CHARACTER TO BINARY

MDB Ri,Wj

Type 5 Instruction

1101	1010	Wjr	Ri	Wjk	Wjd			32 BITS
1	2	3	4	5	6	7	8	

Detailed Description of Instruction Execution

C(C(Wjr)+Wjd) is multiplied by ten. The binary value of the ASCII digit in C(C(Ri)) is added to the product, and the sum replaces C(C(Wjr)+Wjd).

Operand Types	Wik	Object Code (Hex) 1 2 3 4 5 6 7 8
Ri,Hj	0	D A j i 0 d d d
Ri,Tj	1	D A j i 1 d d d
Ri,Dj	2	D A j i 2 d d d
Ri,Fj	3	D A j i 3 d d d

Example

Label Field	OpC Field	Operand Field	Comment Field
	MDB	R11,STYPI	

Programming Note: If C(C(Wjr)+Wjd) is zero initially, repeated use of this instruction with incrementing of C(Ri) will convert an ASCII string representing a decimal value into a binary integer.

DECREMENT, MOVE STRING UNDER DELIMITER CONTROL WHILE COUNTING

MDDDC Ri,Rj,N

Type 4 Instruction

0110	0010	Ri	Rj	N	
1	2	3	4	5 6	24 BITS

Detailed Description of Instruction Execution

C(Ri) and C(Rj) are each decremented by one. The character which is C(C(Ri)) replaces C(C(Rj)). C(T0) is decremented by one. The character that was moved is then matched with the characters specified by N. If the match is not successful, the operation is repeated.

Operand Types	Object Code (Hex) 1 2 3 4 5 6
Ri,Rj,N	6 2 i j n n

Example

```
Label   OpC     Operand          Comment
Field   Field   Field            Field
-----   -----   -------          -------

        MDDDC   R4,R5,X'15'
```

Programming Note: Assuming that T0 contains zero before this instruction is executed, when execution is completed, the number of characters moved will be recorded in T0 as a negative number.

DECREMENT AND MOVE STRING UNDER DELIMITER CONTROL

MDDD Ri,Rj,N

Type 4 Instruction

0110	0001	Ri	Rj	N		24 BITS
1	2	3	4	5	6	

Detailed Description of Instruction Execution

C(Ri) and C(Rj) are each decremented by one. The character which is C(C(Ri)) replaces C(C(Rj)). The character that was moved is then matched with the characters specified by N. If the match is not successful, the operation is repeated.

Operand Types	Object Code (Hex) 1 2 3 4 5 6
Ri,Rj,N	6 1 i j n n

Example

Label Field	OpC Field	Operand Field	Comment Field
	MDDD	R11,R7,X'84'	

Programming Note: At least one character is always moved by the instruction since the match test is made after the move.

DECREMENT BOTH REGISTERS AND MOVE CHARACTER

MDD Ri,Rj

Type 3 Instruction

0110	0000	Ri	Rj
1	2	3	4

16 BITS

Detailed Description of Instruction Execution

C(Rj) and C(Rj) are each decremented by one.

Then C(C(Ri)) replaces C(C(Rj)).

Operand Types	Object Code (Hex) 1 2 3 4
Ri,Rj	6 0 i j

Example

Label Field	OpC Field	Operand Field	Comment Field
	MDD	R6,R11	DEC R6 AND R11, MOVE BYTE

DECREMENT AND MOVE STRING UNDER REGISTER CONTROL

MDDR Ri,Rj

Type 3 Instruction

0101	1000	Ri	Rj
1	2	3	4

16 BITS

Detailed Description of Instruction Execution

C(Ri) and C(Rj) are each decremented by one. The character which is C(C(Ri)) replaces C(C(Rj)). C(Ri) is compared with C(R15). If the addresses are not equal, the operation is repeated. If C(Ri) is equal to C(R15) initially, no operation is performed.

Operand Types	Object Code (Hex) 1 2 3 4
Ri,Rj	5 8 i j

Example

Label Field	OpC Field	Operand Field	Comment Field
	MDDR	R4,R7	

Programming Note: This instruction assumes that R15 contains an address equal to or less than C(Ri).

DECREMENT AND MOVE STRING UNDER T0 AND DELIMITER CONTROL

MDDTD Ri,Rj,N

Type 4 Instruction C 1010

0101	0100	Ri	Rj	N	
1	2	3	4	5	6

24 BITS

Detailed Description of Instruction Execution

C(Ri) and C(Rj) are each decremented by one. The character which is C(C(Ri)) replaces C(C(Rj)). C(T0) is decremented by one. If C(T0) is zero or if the character moved matches one of the characters specified by N, execution ceases; otherwise, the operation is repeated. If C(T0) equals zero initially, no operation is performed.

Operand Types	Object Code (Hex) 1 2 3 4 5 6
Ri,Rj,N	5 4 i j n n

A

Example

Label Field	OpC Field	Operand Field	Comment Field
	MDDTD	R5,R11,X'C'	

Programming Note: This instruction may be used to limit the maximum number of characters moved. T0 can be initialized with a maximum count before this instruction is executed. After execution the number of characters moved may be computed by subtracting the final count in T0 from the initial count.

DECREMENT AND MOVE STRING UNDER T0 CONTROL

MDDT Ri,Rj

Type 3 Instruction

0101	1001	Ri	Rj
1	2	3	4

16 BITS

Detailed Description of Instruction Execution

C(Ri) and C(Rj) are each decremented by one. The character which is C(C(Ri)) replaces C(C(Rj)). C(T0) is decremented by one. If C(T0) is non-zero, the operation is repeated. If C(T0) equals zero initially, no operation is performed.

Operand Types	Object Code (Hex) 1 2 3 4
Ri,Rj	5 9 i j

Example

Label Field	OpC Field	Operand Field	Comment Field
	MDDT	R4,R7	

MOVE FLOATING DECIMAL STRING TO BINARY

MFBN Ri

Type 3 Instruction

0100	1111	Ri	0000
1	2	3	4

16 BITS

Detailed Description of Instruction Execution

This instruction converts a floating decimal string, pointed to by Ri, to a six byte binary number in FP0. A floating decimal string is a string of ASCII decimal digits that includes an optional period (decimal point). The digits that precede the period are called integer digits. Digits that follow the period are called fraction digits. The first character of the string may be a plus sign, a minus sign, a period, or a decimal digit. The string must include at least one digit and must be terminated by a system delimiter or a period.

Several PCB elements must be initialized before the instruction executes. FP0, which will receive the converted number, must contain zero, H6 must contain a maximum count of integer digits. Bits 0-3 of H7 must be zero because they will be used for indicators by the firmware. Bits 4-7 of H7 must contain th maximum count of fraction digits. Ri must point to the byt preceding the first character in the string. Note that a zero in H6 specifies a maximum count of 256 integer digits whereas a zero in H7 specifies that there are no fraction digits. Hence, the lowest integer digit count is one, and the highest fraction digit count is 15.

This instruction sets C(ACF). If a number was converted and a proper terminator was found, NUMBIT is set to one. FP0 contains the six-byte result, and Ri points to the terminator.

The complete ACF byte, including NUMBIT, is set to zero if there is a non-decimal digit in the string or if there is no terminator where one is expected. If NUMBIT is zero, Ri points to the last character processed.

The ASCII string is converted to a binary number in the six byte accumulator FP0. All of the digits, both integer and fraction, are converted into one magnitude. if a period is encountered before the specified number of integer digits have been processed, the succeeding fraction digits will be converted up to the number specified in H7. if C(H7) are greater than the number of fraction digits, the results in FP0 will be multiplied by 10 for each missing digit. If C(H7) are zero, integer digits only will be converted; that is, processing will stop at the first period or system delimiter.

MOVE FLOATING DECIMAL STRING TO BINARY (cont)

The following situations should be kept in mind:

1. If the first character is a period, there must be some fraction digits or NUMBIT will be set to zero.
2. If the number of integer digits equals C(H6), the instruction terminates after converting the integer digits. At the conclusion of execution C(Ri) points to the last digit converted, not at the terminator. NUMBIT is set to zero.
3. If C(H7) is not zero, processing will terminate at a system delimiter. If the number of fraction digits equals C(H7), the terminator may be a second period rather than a system delimiter.
4. The instruction does not indicate that a magnitude was too large for the accumulator.

Operand Types	Object Code (Hex) 1 2 3 4
Ri	4 F i 0

Example

```
Label   OpC     Operand         Comment
Field   Field   Field           Field
-----   -----   -------         -------

=H4     HTLY    4
=H8     HTLY    8
        ZERO    DO              ZERO FPO AND
        ZERO    D1              H6 AND H7
        MOV     =H4,H6          SET INTEGER SIZE
        MOV     =H8,H7          SET FRACTION SIZE
        MFBN    R15
```

MOVE HEXADECIMAL STRING TO BINARY

MXBN Ri

Type 3 Instruction

0100	1111	Ri	0000
1	2	3	4

16 BITS

Detailed Description of Instruction Execution

This instruction converts a hexadecimal string, pointed to by Ri, to a six-byte binary number in FP0. A hexadecimal string is a sequence of ASCII characters that represent hexadecimal digits. The first character of the string may be a plus sign, a minus sign, or a hexadecimal digit. The string must include at least one digit and must be terminated by a system delimiter.

Several PCB elements must be initialized before the instruction executes. FP0, which will receive the converted number, must contain zero. H6 must contain a maximum count of hexadecimal digits. Bits 0-3 of H7 must be zero because they will be used for indicators by the firmware. Ri must point to the byte preceding the first character in the string. Note that a zero in H6 specifies a maximum count of 256 hexadecimal digits. Hence, the lowest digit count is one.

This instruction sets C(ACF). In particular, NUMBIT is set to one if a number was converted and a proper terminator was found. FP0 contains the six-byte result, and Ri points to the terminator.

NUMBIT is set to zero if there is a non-hexadecimal digit in the string or if there is no terminator where one is expected. If NUMBIT is zero, Ri points to the last character processed.

The ASCII string is converted to a binary number in the six-byte accumulator FP0. All of the digits are converted into one magnitude. Processing will stop at the first ????? period or system delimiter. The following situations should be kept in mind:

1. If the number of hexadecimal digits equals C(H6), the instruction terminates after converting the digits. At the conclusion of execution C(Ri) points to the last digit converted, not at the terminator. NUMBIT is set to zero.

2. The instruction does not indicate that a magnitude was too large for the accumulator.

INSTRUCTIONS

Operand Types	Object Code (Hex) 1 2 3 4
Ri	4 F i 0

Example

```
Label   OpC     Operand         Comment
Field   Field   Field           Field
-----   -----   -------         -------

K10     HTLY    10
        ZERO    D0              ZERO FP0 AND
        ZERO    D1              H6 AND H7
        MOV     K10,H6          SET INTEGER COUNT
        MXBN    R14
```

INCREMENT SOURCE REGISTER AND MOVE CHARACTER

MIC Ri,Rj

Type 3 Instruction

0100	0111	Ri	Rj
1	2	3	4

16 BITS

Detailed Description of Instruction Execution

C(Ri) is incremented by one. C(C(Ri)) replaces C(C(Rj)).

Operand Types	Object Code (Hex) 1 2 3 4
Ri,Rj	4 7 i j

Example

Label Field	OpC Field	Operand Field	Comment Field
	MIC	R5,R14	

INCREMENT, MOVE STRING UNDER DELIMITER CONTROL WHILE COUNTING

MIIDC Ri,Rj,N

Type 4 Instruction

0100	0010	Ri	Rj	N	
1	2	3	4	5	6

24 BITS

Detailed Description of Instruction Execution

C(Ri) C(Rj) are each incremented by one. The character which is C(C(Ri)) replaces (C(C(Rj)). C(T0) is decremented by one. the character that was moved is then matched with the characters specified by N. If the match is not successful, the operation is repeated.

Operand Types	Object Code (Hex) 1 2 3 4 5 6
Ri,Rj,N	4 2 i j n n

Example

Label Field	OpC Field	Operand Field	Comment Field
	MIIDC	R14,R8,X'84'	

Programming Note: Assuming that T0 contains zero before this instruction is executed, when execution is completed, the number of characters moved will be recorded in T0 as a negative number.

INCREMENT AND MOVE STRING UNDER DELIMITER CONTROL

MIID Ri,Rj,N

Type 4 Instruction

0100	0001	Ri	Rj	N		24 BITS
1	2	3	4	5	6	

Detailed Description of Instruction Execution

C(Ri) and C(Rj) are incremented by one. The character which is C(C(Ri)) replaces C(C(Rj)). The character that was moved is then matched with the characters specified by N. If the match is not successful, the operation is repeated.

Operand Types	Object Code (Hex) 1 2 3 4 5 6
Ri,Rj,N	4 1 i j n n

Example

Label Field	OpC Field	Operand Field	Comment Field
	MIID	R7,R11,X'DO'	

Programming Note: At least one character is always moved by this instruction since the match test is made after the move.

INCREMENT BOTH REGISTERS AND MOVE CHARACTER

MII Ri,Rj

Type 3 Instruction

0100	0000	Ri	Rj
1	2	3	4

16 BITS

Detailed Description of Instruction Execution

C(Ri) and C(Rj) are each incremented by one. Then C(C(Ri)) replaces C(C(Rj)).

Operand Types	Object Code (Hex) 1 2 3 4
Ri,Rj	4 0 i j

Example

Label Field	OpC Field	Operand Field	Comment Field
	MII	R14,R15	

INCREMENT AND MOVE STRING UNDER REGISTER CONTROL

MIIR Ri,Rj

Type 3 Instruction

0101	0000	Ri	Rj
1	2	3	4

16 BITS

Detailed Description of Instruction Execution

C(Ri) and C(Rj) are each incremented by one. The character which is C(C(Ri)) replaces C(C(Rj)). C(Ri) is compared with C(R15). If the addresses are not equal, the operation is repeated. If C(Ri) is initially equal to C(R15), no operation is performed.

Operand Types	Object Code (Hex) 1	2	3	4
Ri,Rj	5	0	i	j

Example

Label Field	OpC Field	Operand Field	Comment Field
	MIIR	R6,R3	

Programming Note: This instruction assumes that R15 contains an address equal to or greater than C(Ri).

INCREMENT AND MOVE STRING UNDER T0 AND DELIMITER CONTROL

MIITD Ri,Rj,N

Type 4 Instruction

0101	0010	Ri	Rj	N	
1	2	3	4	5	6

24 BITS

Detailed Description of Instruction Execution

C(Ri) and C(Rj) are each incremented by one. The character which is C(C(Ri)) replaces C(C(Rj)). C(T0) is decremented by one. If C(T0) is zero or if the character moved matches one of the characters specified by N, execution ceases; otherwise, the operation is repeated. if C(T0) equals zero initially, no operation is performed.

Operand Types	Object Code (Hex) 1 2 3 4 5 6
Ri,Rj,N	5 2 i j n n

Example

Label Field	OpC Field	Operand Field	Comment Field
	MIITD	R9,R10,5	

Programming Note: This instruction may be used to limit the maximum number of characters moved. T0 can be initialized with a maximum count before this instruction is executed. After execution the number of characters moved may be computed by subtracting the final count in T0 from the initial count.

INCREMENT AND MOVE STRING UNDER T0 CONTROL

MIIT Ri,Rj

Type 3 Instruction

0101	0001	Ri	Rj
1	2	3	4

16 BITS

Detailed Description of Instruction Execution

C(Ri) and C(Rj) are each incremented by one. The character which is C(C(Ri)) replaces C(C(Rj)). C(T0) is decremented by one. If C(T0) is non-zero, the operation is repeated. If C(T0) equals zero initially, no operation is performed.

Operand Types	Object Code (Hex) 1	2	3	4
Ri,Rj	5	1	i	j

Example

Label Field	OpC Field	Operand Field	Comment Field
	MIIT	R7,R4	

MOVE, ATTACHED, STORAGE REGISTER TO REGISTER

MOVA Si,Rj

Type 5 Instruction

1011	1001	Sir	Rj	Sik	Sid			32 BITS
1	2	3	4	5	6	7	8	

Detailed Description of Instruction Execution

This instruction moves C(C(Sir)+Sid) to Rj. When the instruction has finished execution, Rj will be attached if three conditions hold: 1) Rj was already attached, 2) Si's FID field contains the same value as Rj's FID field, and 3) Si contains a normalized displacement. If all three conditions are not met, the instruction works just like the MOV Si,Rj instruction.

Rj's linked/unlinked flag is set to agree with Si's flag byte.

Operand Types	Wik	Object Code (Hex) 1 2 3 4 5 6 7 8
Si,Rj		B 9 i j 3 d d d

Example

Label Field	OpC Field	Operand Field	Comment Field
	MOVA	STYPI,R12	

Programming Note: Functionally, this instruction is the same as MOV Si,Rj, but it may leave the register in an attached state. This only affects code that depends on MOV Si,Rj to detach a register. The instruction is intended to speed up software when data are being manipulated within one frame. If the register cannot remain attached, this instruction is slightly slower than MOV Si,Rj; otherwise, it is several times faster.

MOVE BIT TO BIT

MOV Bi,Bj

Type 6 Instruction

1111	0101	Bir	Bjr	io	Bid		
1	2	3	4	5	6	7	8

jo	Bjd		
9	10	11	12

48 BITS

Detailed Description of Instruction Execution

C(C(Bir)+Bid) is moved to C(C(Bjr)+Bjd).

Operand Types	Object Code (Hex) 1 2 3 4 5 6 7 8 9 0 1 2
Bi,Bj	F 5 i j o d d d o d d d

o is 8 plus the bit offset in the byte.

Example

Label Field	OpC Field	Operand Field	Comment Field
	MOV	BTYPI,BTYPJ	

MOVE ADDRESS REGISTER TO ADDRESS REGISTER

MOV Ri,Rj

Type 3 Instruction

0000	0011	Ri	Rj
1	2	3	4

16 BITS

Detailed Description of Instruction Execution

C(Ri) replaces C(Rj). C(Ri) is not changed. If Ri was attached, Rj becomes attached; otherwise, Rj is detached.

Operand Types	Object Code (Hex) 1	2	3	4
Ri,Rj	0	3	i	j

Example

Label Field	OpC Field	Operand Field	Comment Field
	MOV	R7,R15	

STORE ADDRESS REGISTER

MOV Ri,Sj

Type 5 Instruction

1100	1010	Sjr	Ri	Sjk	Sjd			32 BITS
1	2	3	4	5	6	7	8	

Detailed Description of Instruction Execution

The detached form of C(Ri), that is, displacement, link and FID, replaces C(C(Sjr)+Sjd). C(Ri) is not changed, not even detached.

Operand Types	Wik	Object Code (Hex) 1	2	3	4	5	6	7	8
Ri,Sj	3	C	A	j	i	3	d	d	d

Example

Label Field	OpC Field	Operand Field	Comment Field
	MOV	R2,STYPJ	

LOAD ADDRESS REGISTER

MOV Si,Rj

Type 5 Instruction

1011	1010	Sir	Rj	Sik	Sid			32 BITS
1	2	3	4	5	6	7	8	

Detailed Description of Instruction Execution

Rj is detached and then C(C(Sir)+Sid) replaces the six low-order bytes (displacement, link and FID) of Rj. The high-order two bytes of Rj are set to zero.

Operand Types	Wik	Object Code (Hex) 1 2 3 4 5 6 7 8
Si,Rj	3	B A i j 3 d d d

Example

Label Field	OpC Field	Operand Field	Comment Field
	MOV	STYPI,R15	

Programming Note: Since the address registers can be addressed as storage locations, they can be changed by using other move instructions. However, this may lead to problems in the future if the address register philosophy changes. For simplicity use only the Load Address Register instruction to move an address into an address register.

MOVE ELEMENT TO ELEMENT

MOV Wi,Wj

Type 6 Instruction

1	2	3	4	5	6	7	8
1111	0011	Wir	Wjr	Wik	Wid		

9	10	11	12
Wjk	Wjd		

48 BITS

Detailed Description of Instruction Execution

C(C(Wir)+Wid) replaces C(C(Wjr)+Wjd).

Operand Types	Wik	Wjk	Object Code (Hex) 1	2	3	4	5	6	7	8	9	0	1	2
Ci,Cj	0	0	F	3	r	r	0	d	d	d	0	d	d	d
Hi,Hj	0	0	F	3	r	r	0	d	d	d	0	d	d	d
Ti,Tj	1	1	F	3	r	r	1	d	d	d	1	d	d	d
Di,Dj	2	2	F	3	r	r	2	d	d	d	2	d	d	d
Fi,Fj	3	3	F	3	r	r	3	d	d	d	3	d	d	d
Si,Sj	3	3	F	3	r	r	3	d	d	d	3	d	d	d
Vi,Vj	7	7	F	3	r	r	7	d	d	d	7	d	d	d

Example

```
Label   OpC      Operand          Comment
Field   Field    Field            Field
-----   -----    -------          -------

        MOV      HTYPI,HTYPJ
        MOV      CTYPI,CTYPJ
        MOV      TTYPI,TTYPJ
        MOV      DTYPI,DTYPJ
        MOV      FTYPI,FTYPJ
        MOV      STYPI,STYPJ
```

Programming note: The assembler allows the mnemonic format MCC C,C as well as MOV C,C.

Although this is the only instruction that moves tallies, double tallies or triple tallies, there are other instructions that move single characters or character strings from storage to storage.

MULTIPLY BY 10

MUL10 Wj

Type 5 Instruction

1011	0001	Wjr	0000	Wjk	Wjd			32 BITS
1	2	3	4	5	6	7	8	

Detailed Description of Instruction Execution

C(C(Wjr)+Wjd) is multiplied by 10.

Operand Types	Wjk	Object Code (Hex) 1	2	3	4	5	6	7	8
Hj	0	B	1	j	0	0	d	d	d
Tj	1	B	1	j	0	1	d	d	d
Dj	2	B	1	j	0	2	d	d	d
Fj	3	B	1	j	0	3	d	d	d

Example

```
Label   OpC      Operand          Comment
Field   Field    Field            Field
-----   -----    -------          -------

        MUL10    HTYPI
        MUL10    FTYPJ
```

MULTIPLY BY SCALE

MULS Wi

Type 5 Instruction

1011	0000	Wir	0000	Wik	Wid			32 BITS
1	2	3	4	5	6	7	8	

Detailed Description of Instruction Execution

The six-byte accumulator (FP0) is multiplied by 10 the number of times specified by the low-order byte of C(C(Wir)+Wid). A 12-byte number is the result. The high-order six bytes of the result are placed in FPY, and the low-order six bytes are placed in FP0.

?????Maybe this will set OVFBIT?????

Operand Types	Wik	Object Code (Hex) 1 2 3 4 5 6 7 8
Hi	0	B 0 i 0 0 d d d
Ti	1	B 0 i 0 1 d d d
Di	2	B 0 i 0 2 d d d
Fi	3	B 0 i 0 3 d d d

Example

Label Field	OpC Field	Operand Field	Comment Field
	MULS	TTYPI	
	MULS	FTYPJ	

Programming Note: Although large operands can be addressed, only the low-order byte of Wi is used so the largest scale value is 255.

MULTIPLY

MUL Wi

Type 5 Instruction

1011	0011	Wir	0000	Wik	Wid			32 BITS
1	2	3	4	5	6	7	8	

Detailed Description of Instruction Execution

C(D0) is multiplied by C(C(Wir)+Wid). An 8-byte product replaces C(D1) and C(D0).

The sign of the product is determined by the rules of algebra.

C(ACF) is updated to reflect overflow.

Operand Types	Wik	Object Code (Hex) 1	2	3	4	5	6	7	8
Hi	0	B	3	i	0	0	d	d	d
Ti	1	B	3	i	0	1	d	d	d
Di	2	B	3	i	0	2	d	d	d
Vi	7	B	3	i	0	7	d	d	d

Example

```
Label   OpC      Operand          Comment
Field   Field    Field            Field
-----   -----    -------          -------

        MUL      DTYPI            DTYPI * D0
        MUL      FTYPJ            FTYPJ * FP0
```

Programming Note: The mnemonic MUL may be used with F type operands (six-byte elements), but the object code generated by the assembler is the same as that generated for instruction MULX.

MULTIPLY EXTENDED

MULX Wi

Type 5 Instruction

1011	0010	Wir	0000	Wik	Wid			32 BITS
1	2	3	4	5	6	7	8	

Detailed Description of Instruction Execution

C(FP0) is multiplied by C(C(Wir)+Wid). A 12-byte product replaces C(FPY) and C(FP0). The sign of the product is determined by the rules of algebra.

Operand Types	Wik	Object Code (Hex) 1 2 3 4 5 6 7 8
Hi	0	B 2 i 0 0 d d d
Ti	1	B 2 i 0 1 d d d
Di	2	B 2 i 0 2 d d d
Fi	3	B 2 i 0 3 d d d
Vi	7	B 2 i 0 7 d d d

Example

Label Field	OpC Field	Operand Field	Comment Field
	MULX	HTYPI	MULTIPLY FP0

Programming Note: The mnemonic MUL may be used with F type operands (six-byte elements), but the object code generated by the assembler is the same as that generated for instruction MULX.

MOVE HEXADECIMAL CHARACTER TO BINARY

MXB Ri,Wj

Type 5 Instruction

1101	1110	Wjr	Ri	Wjk	Wjd			32 BITS
1	2	3	4	5	6	7	8	

Detailed Description of Instruction Execution

C(C(Wjr)+Wjd) is multiplied by sixteen (that is, shifted left four bits). The binary value of the ASCII hexadecimal digit in C(C(Ri) is added to the product, and the result replaces C(C(Wjr)+Wjd).

Operand Types	Wik	Object Code (Hex) 1	2	3	4	5	6	7	8
Ri,Cj	0	D	E	j	i	0	d	d	d
Ri,Hj	0	D	E	j	i	0	d	d	d
Ri,Tj	1	D	E	j	i	1	d	d	d
Ri,Dj	2	D	E	j	i	2	d	d	d
Ri,Fj	3	D	E	j	i	3	d	d	d

Example

Label Field	OpC Field	Operand Field	Comment Field
	MXB	R14,FTYPJ	

Programming Note: If C(C(Wjr)+Wjd) is initially set to zero, repeated use of this instruction with incrementing of C(R) will convert an ASCII string representing a hexadecimal value into a binary integer.

NEGATE

NEG Wj

Type 5 Instruction

1010	1010	Wjr	0000	Wjk	Wjd			32 BITS
1	2	3	4	5	6	7	8	

Detailed Description of Instruction Execution

C(C(Wjr)+Wjd) is replaced with its two's complement form.

Operand Types	Wjk	Object Code (Hex) 1	2	3	4	5	6	7	8
Hj	0	A	A	j	0	0	d	d	d
Tj	1	A	A	j	0	1	d	d	d
Dj	2	A	A	j	0	2	d	d	d
Fj	3	A	A	j	0	3	d	d	d
Vj	7	A	A	j	0	7	d	d	d

Example

Label Field	OpC Field	Operand Field	Comment Field
	NEG	TTYPJ	

NO OPERATION

NOP

Type 1 Instruction

0000	1111
1	2

8 BITS

Detailed Description of Instruction Execution

This instruction causes the CPU to take the next instruction in sequence.

Operand Types	Object Code (Hex) 1	2
	0	F

Example

Label Field	OpC Field	Operand Field	Comment Field
	NOP		

STORE A ONE

ONE Wj

Type 5 Instruction

1100	0001	Wjr	0000	Wjk	Wjd			32 BITS
1	2	3	4	5	6	7	8	

Detailed Description of Instruction Execution

C(C(Wjr)+Wjd) is replaced by a binary one with leading zeros.

Operand Types	Wjk	Object Code (Hex) 1	2	3	4	5	6	7	8
Hj	0	C	1	j	0	0	d	d	d
Tj	1	C	1	j	0	1	d	d	d
Dj	2	C	1	j	0	2	d	d	d
Fj	3	C	1	j	0	3	d	d	d
Vj	7	C	1	j	0	7	d	d	d

Example

Label Field	OpC Field	Operand Field	Comment Field
	ONE	FTYPJ	SET TO 1

OR WITH IMMEDIATE

OR Rj,N
OR N,Rj

Type 4 Instruction

1000	0010	Rj	0000	N	
1	2	3	4	5	6

24 BITS

Detailed Description of Instruction Execution

C(C(Rj)) is logically ORed with N. The result replaces C(C(Rj)).

Operand Types	Object Code (Hex) 1	2	3	4	5	6
Rj,N	8	2	j	0	n	n
N,Rj	8	2	j	0	n	n

Example

Label Field	OpC Field	Operand Field	Comment Field
	OR	R6,X'C'	
	OR	X'D',R5	

OR WITH STORAGE

OR Rj,Ri

Type 3 Instruction

1110	0010	Rj	Ri	16 BITS
1	2	3	4	

Detailed Description of Instruction Execution

C(C(Rj)) is logically ORed with C(C(Ri)).

The result replaces C(C(Rj)).

Operand Types	Object Code (Hex) 1 2 3 4
Rj,Ri	E 2 j i

Example

Label Field	OpC Field	Operand Field	Comment Field
	OR	R15,R14	

OR ELEMENT WITH ELEMENT

OR Wi,Wj

Type 6 Instruction

1	2	3	4	5	6	7	8
1111	1100	Wir	Wjr	Wik	Wid		

9	10	11	12
Wjk	Wjd		

48 BITS

Detailed Description of Instruction Execution

C(C(Wir)+Wid) is logically ORed with C(C(Wjr)+Wjd). The result replaces C(C(Wjr)+Wjd).

Operand Types	Wik	Wjk	Object Code (Hex) 1	2	3	4	5	6	7	8	9	0	1	2
Ci,Cj	0	0	F	B	r	r	0	d	d	d	0	d	d	d
Hi,Hj	0	0	F	B	r	r	0	d	d	d	0	d	d	d
Ti,Tj	1	1	F	B	r	r	1	d	d	d	1	d	d	d
Di,Dj	2	2	F	B	r	r	2	d	d	d	2	d	d	d
Fi,Fj	3	3	F	B	r	r	3	d	d	d	3	d	d	d
Vi,Vj	7	7	F	B	r	r	7	d	d	d	7	d	d	d

Example

```
Label   OpC      Operand          Comment
Field   Field    Field            Field
-----   -----    -------          -------

        OR       HTYPI,HTYPJ
        OR       CTYPI,CTYPJ
        OR       TTYPI,TTYPJ
        OR       DTYPI,DTYPJ
        OR       FTYPI,FTYPJ
```

POP NUMBER

POPNR Ri
POPN (The assembler's OSYM uses R3 for Ri)

Type 3 Instruction

1001	1111	Ri	0000	16 BITS
1	2	3	4	

Detailed Description of Instruction Execution

The purpose of this instruction is to pop a number off the stack and store it in the extended accumulator.

The stack pointer (Ri) is decremented by 10.

If the descriptor type code at Ri;C0 has the low-order bit set, the number in bytes Ri;C2 through Ri;C7 is copied to the extended accumulator (FP0).

If the stack pointer would be decremented past the beginning of the buffer, or if the low-order bit of the type code is zero, the pointer is not decremented. A BSL using the second entry of the branch table is executed instead.

Operand Types	Object Code (Hex) 1 2 3 4
Ri	9 F i 0

Example

Label Field	OpC Field	Operand Field	Comment Field
	POPNR	R4	
	POPN		

Caution: The software that is invoked by the BSL may assume that R3 is the stack pointer.

POP STRING

POPSRR Ri,Rj
POPS (The assembler's OSYM uses R3 for Ri and R10 for Rj)

Type 3 Instruction

1110	1111	Ri	Rj
1	2	3	4

16 BITS

Detailed Description of Instruction Execution

The purpose of this instruction is to pop a string off the stack.

The stack pointer (Ri) is decremented by 10.

If the descriptor type code at byte Ri;C0 is the direct string type code (i.e., X'02'), the stack pointer (Ri) is moved to the string pointer (Rj). That is, C(Ri) replaces C(Rj).

If the descriptor type code at byte Ri;C0 has the high order bit set, the contents of the storage register at bytes Ri;C2 through Ri;C7 are moved to the string pointer (Rj). That is, the six bytes starting at the location C(Ri)+2 replace C(Rj).

If the stack pointer would be decremented past the beginning of the buffer, or if the descriptor type does not have the high-order bit set and is not equal to X'02', the pointer is not decremented. A BSL using the fourth entry in the branch table is executed instead.

Operand Types	Object Code (Hex) 1 2 3 4
Ri,Rj	E F i j

Example

Label Field	OpC Field	Operand Field	Comment Field
	POPSRR	R4, R5	
	POPSR	R4	
	POPS		

Caution: The software that is invoked by the BSL may assume that R3 is the stack pointer.

PUSH DESCRIPTOR

PUSHD Ri,Rj
PUSHD Ri (The assembler's OSYM uses R3 for Rj)

Type 3 Instruction

1110	1100	Ri	Rj
1	2	3	4

16 BITS

Detailed Description of Instruction Execution

This instruction pushes a descriptor onto the stack. It copies the 10 bytes referenced by Ri;C0 through Ri;C9 (the descriptor) to the stack referenced by Rj;C0 through Rj;C9. Rj is then incremented by 10.

If the descriptor type code is zero (that is, if the byte referenced by Ri;C0 contains X'00'), a BSL is executed using the third entry of the branch table.

Operand Types	Object Code (Hex) 1 2 3 4
Ri,Rj	E C i j

Example

Label Field	OpC Field	Operand Field	Comment Field
	PUSHDRR	R4, R5	
	PUSHD	R4	

Caution: If the 10 bytes referenced by Rj cross a frame limit, a CROSSING FRAME LIMIT abort will result.

Caution: If the 10 bytes referenced by Ri cross a frame limit a CROSSING FRAME LIMIT abort will result.

Caution: The software that is invoked by the BSL may assume that R3 is the stack pointer.

PUSH NUMBER

PUSHNR Ri
PUSHN (The assembler's OSYM uses R3 for Ri)

Type 3 Instruction

1	2	3	4
1001	1011	Ri	0000

16 BITS

Detailed Description of Instruction Execution

This instruction pushes a numeric descriptor onto the stack. It writes the numeric code X'01' into the byte referenced by Ri;C0 and copies the number from the extended accumulator (FP0) to bytes Ri;C2 through Ri;C7. The stack pointer (Ri) is then incremented by 10.

Operand Types	Object Code (Hex) 1	2	3	4
Ri	9	B	i	0

Example

Label Field	OpC Field	Operand Field	Comment Field
	PUSHNR	R4	
	PUSHN		

Caution: If the 10 bytes referenced by Ri cross a frame limit, a CROSSING FRAME LIMIT abort will result.

PUSH INDIRECT STRING

PUSHS Ri,Rj
PUSHS Ri (The assembler's OSYM uses R3 for Rj)

Type 3 Instruction

1110	1101	Ri	Rj
1	2	3	4

16 BITS

Detailed Description of Instruction Execution

This instruction pushes an indirect string descriptor onto the stack. It writes the indirect string type code X'82' into the byte referenced by Rj;C0 and moves the contents of Ri to the storage register at bytes Rj;C2 through Rj;C7. The stack printer (Rj) is then incremented by 10.

Operand Types	Object Code (Hex) 1 2 3 4
Ri,Rj	E D i j

Example

```
Label   OpC      Operand          Comment
Field   Field    Field            Field
-----   -----    -------          -------

        PUSHSRR  R4,R5
        PUSHS    R4
```

Caution: If the 10 bytes referenced by Rj cross a frame limit, a CROSSING FRAME LIMIT abort will result.

PUSH TEMPORARY STRING

PUSHTS Ri,Rj
PUSHTS Ri (The assembler's OSYM uses R3 for Ri)

Type 3 Instruction

1	2	3	4	
1110	1110	Ri	Rj	16 BITS

Detailed Description of Instruction Execution

This instruction pushes a temporary string descriptor onto the stack. It writes the temporary string type code X'C2' into the byte referenced by Rj;C0 and moves the contents of Ri to the storage register at bytes Rj;C2 through Rj;C7. The stack pointer (Rj) is then incremented by 10.

Operand Types	Object Code (Hex) 1	2	3	4
Ri,Rj	E	E	i	j

Example

```
Label  OpC      Operand        Comment
Field  Field    Field          Field
-----  -----    -------        -------

       PUSHTSRR R4,R5
       PUSHTS   R4
```

Caution: If the 10 bytes referenced by Rj cross a frame limit, a CROSSING FRAME LIMIT abort will result.

QUEUE COMMAND

QCMD N

Type 2 Instruction

0110	0110	N	
1	2	3	4

16 BITS

Detailed Description of Instruction Execution

This instruction queues a command for the input/output processor. The command is the value N.

Note that there are several mnemonics in OSYM that generate this instruction with different values for N, for example, IORESET and IOKILL. The mnemonic QCMD would not ordinarily be used in a mode.

The following input interface is required :

Element	Description
H0	Channel number
H1	Controller number
H2	IOP2 address

Operand Types	Object Code (Hex) 1	2	3	4
N	6	6	n	n

Example

Label Field	OpC Field	Operand Field	Comment Field
	IORESUME	*	CONTINUE I/O

QUEUE I/O REQUEST

QIO Ri,N

Type 4 Instruction

0110	1000	Ri	0000	N	
1	2	3	4	5	6

24 BITS

Detailed Description of Instruction Execution

This instruction initiates the transfer of a task descriptor (TD) to the input/output processor. C(Ri) points to the task descriptor defining an I/O operation. The TD contains the number of the controller for which action is desired.

N defines the service request with one of the following values:

Value (Hex)	Meaning
01	Queue the TD to the tail of the queue of TDs for this controller (normal data flow);
02	Queue the TD to the head of the queue of TDs for this controller (expedited data flow);
03	Execute this TD without regard to the queue of TDs for this controller (error recovery).

If the controller number specified in the TD indicates that the transfer is to another process in the same computer, the TD will be transferred to the other process provided the intended recipient process has initiated a receive TD. Otherwise, the effect of this instruction is that the queue command and TD are queued to the IO-Circular-Queue of an IOP. The specific IOP is specified in the body of the TD.

On completion of the instruction, H0 will contain the result of the operation with one of the following codes:

Code (Hex)	Meaning
00	I/O issued;
0D	I/O queue full;
0E	Invalid IOP address in TD;
0F	Invalid type field in TD (completion handler invalid)

QUEUE I/O REQUEST (cont)

H0 containing X'00' indicates that the transfer was issued, not that the transfer completed without error. When a TD is dequeued by means of the DQIO Rj,N instruction, it's status bytes must be checked to determine the outcome of the request.

Code X'0D' can only be returned in monitor mode. In virtual mode the process is roadblocked until there is room in the queue. The Monitor should loop on the QIO instruction while counting to some number that indicates remedial action should be taken.

Codes X'0E' and X'0F' mean that the I/O request was rejected.

Operand Types	Object Code (Hex) 1 2 3 4 5 6
Ri,N	6 8 i 0 n n

Example

Label Field	OpC Field	Operand Field	Comment Field
	QIO	R3,X'01'	QUEUE UP TD

REGISTER DETACH AND SET DISPLACEMENT TO ONE

RDETO Rj

Type 3 Instruction

0001	0111	Rj	0000
1	2	3	4

16 BITS

Detailed Description of Instruction Execution

If Rj is attached, it is detached, and its displacement field is set to one.

If Rj is detached, its displacement field is set to one.

Operand Types	Object Code (Hex) 1 2 3 4
Rj	1 7 j 0

Example

Label Field	OpC Field	Operand Field	Comment Field
	RDETO	R12	

Programming Note: This instruction replaces the use of the ONE W instruction to detach a register and set its displacement field to one.

REGISTER DETACH AND SET DISPLACEMENT TO ZERO

RDETZ Rj

Type 3 Instruction

0001	0110	Rj	0000	16 BITS
1	2	3	4	

Detailed Description of Instruction Execution

If Rj is attached, it is detached, and its displacement field is set to zero.

If Rj is detached, its displacement field is set to zero.

Operand Types	Object Code (Hex) 1 2 3 4
Rj	1 6 j 0

Example

Label Field	OpC Field	Operand Field	Comment Field
	RDETZ	R12	

Programming Note: This instruction replaces the use of the ZERO W instruction to detach a register and set its displacement field to zero.

READ INDIRECT

READI Di,Rj

Type 5 Instruction

1010	1011	Dir	Rj	Dik	Did			32 BITS
1	2	3	4	5	6	7	8	

General Description of Instruction Execution

Because this instruction is quite complex, a general description useful to assembly language programmers is given here and further details are provided below.

This instruction commands the I/O processor to accept characters from a process's I/O device and transfer them to main memory.

The C(C(Dir)+Did) is as follows:

tally	timeout count in tenths of a second
byte	read command
byte	mask of active delimiters

Rj points to the input buffer.

The mask specifies which characters are to be treated as terminators. Some read commands do not use this mask, in which cases it will be ignored.

Before the instruction is executed, T0 must contain the maximum number of characters to be read. The maximum number of characters that may be specified in T0 is 256. A value less than one will result in no operation.

It is not valid to use this instruction to request a read that may cross a frame boundary. Hence, Rj must point to a byte in a frame that will allow the number of characters specified in T0 to be read.

There are several read commands described in the I/O processor firmware specification which can be used in Di. However, assembly language programmers are encouraged to use macros defined in OSYM. These macros are READ, READW, READX, and READL.

The I/O processor uses the mask for the active delimiters if the read command specifies its use. Each mask bit that is equal to 1 tells the I/O processor that it should terminate the input if the corresponding character is entered:

READI (cont)

Bits (L to R)	Meaning
0	Unassigned
1	Stop input on control-I (TAB)
2	Stop input on ESCape
3	Stop input on control-P
4	Stop input on control-N
5	Stop input on either carriage return or linefeed
6	Stop input on the character in Terminal I/O Workspace byte TSC3
7	Stop input on the character in Terminal I/O Workspace byte TSC2

If the variable delimiters specified in the two low-order bits of the mask are to be used, they must be loaded into Terminal I/O Workspace bytes TSC2 and/or TSC3 prior to executing the instruction. The delimiter character will not be echoed to the I/O device but will be included in the transfer to main memory.

Instruction execution ends when either the number of characters specified in T0 has been read or the specified terminator is entered. T0 contains the original count less the number of characters read. T1 contains the the number of characters read.

If the virtual process is a TIPH process, that is, if the PIB PHANTOM bit is a 1, the instruction traps to the Software Debugger and does not perform any I/O.

Operand Types	Wik	Object Code (Hex) 1 2 3 4 5 6 7 8
Di,Rj	1	A B i j 1 d d d

Example

```
Label  OpC     Operand            Comment
Field  Field   Field              Field
-----  -----   -------            -------

       MOV     TIMOUTREAD,DTYPI   GET COUNT, COMMAND, MASK
       READI   DTYPI,R13
```

READ (cont)

Detailed Description of Instruction Execution

When the firmware executes this instruction, it first checks COPY.FLG in the Terminal I/O Workspace. If COPY.FLG is not set, the firmware sets up the Read task descriptor. This includes moving the two bytes in TSC2 and TSC3 from the Terminal I/O Workspace to IOADDR1 and IOADDR2, respectively, and storing Nm in the IOADDR0 field of the READ task descriptor. The firmware issues a QIO command to the I/O processor. The Read task descriptor points to the corelocked input buffer pointed to by the INPUT.BUF field in the Terminal I/O Workspace. The firmware next sets input roadblock, sets READ.IN.PROGRESS to 1, removes the PIB from the Priority Queue, and backs up the program counter to point to the READ instruction.

Since the PIB has been removed from the Priority Queue, the process will not run. When the I/O processor has completed the read, it interrupts the CPU. The CPU firmware clears the input roadblock, sets COPY.FLG to 1, and puts the PIB at the top of the Priority Queue. When the process runs again, the READ instruction will be re-executed. This time COPY.FLG will be set. The firmware checks the validity of the virtual data address, copies the data from the corelocked input buffer to the virtual process's buffer, sets that buffer's status to write required, and resets COPY.FLG. It subtracts the contents of LENGTH.TRANSFERRED (in the Read task descriptor) from the contents of T0, putting the difference back into T0, and copies the contents of LENGTH.TRANSFERRED into T1. The firmware then goes to RNI to execute the next instruction.

Terminal I/O errors are reported by the I/O processor to the firmware via the Status task descriptor and never by the Read task descriptor itself. Therefore, when a terminal error is reported, the Read task descriptor remains active and the process enters the Software Debugger. The firmware transfers to Software Debugger entry-point 10 for the BREAK key and to entry-point 12 for other terminal errors. If one or more of PRGERR (PIB byte 1, bit 2) or INDEBUG (PIB byte 1, bit 1) or INCCB (PIB byte X'10', bit 0) or PHANTOM (PIB byte 0, bit 7) is set to 1, BREAK key is ignored.

Handshaking errors between the controllers and the CPU may occur during I/O operations. These errors may be reported using the Read task descriptor as well as the Status task descriptor. Whenever the firmware detects a handshake error, it aborts to entry point 20 of the Software Debugger.

The Monitor may not use the READ instruction for terminal input; it must issue terminal I/O task descriptors in the same manner that it does for other I/O devices. If this instruction is executed in monitor mode, the Firmware Debugger will be entered.

READ (cont)

If the corelocked circular output buffer in the Sequel memory (the buffer pointed to by the OUTPUT.BUF field in the Terminal I/O Workspace) is not empty when this instruction is encountered, the output roadblock is set (OBYTEBLK/ is zeroed), the READ.PENDING bit is set, a firmware release quantum entry to the Monitor is taken, and the instruction is not executed. When the output buffer becomes empty, the output roadblock will be cleared so that the READ instruction will be re-executed.

Generally, the I/O processor will terminate input when either the maximum byte count runs out or a terminator character is entered. The actual method of termination is determined by the definition of the specific read command in Nc.

The read commands are defined in the "IOP2 Firmware Specification", FS20032441.

If the process is a TIPH process, that is, if PIB PHANTOM bit is a 1, the READ instruction will trap to the Software Debugger and not perform any I/O. First, all the error checks are made. The virtual storage address of the terminal I/O buffer will be stored in the PIB at field PH.WRITER2. The Software Debugger is then entered at entry point 17.

READ PROM

RPROM Rj

Type 3 Instruction

1001	0101	Rj	0000	16 BITS
1	2	3	4	

Detailed Description of Instruction Execution

After attaching Rj, this instruction performs a PROM read and stores the data in the bytes pointed at by C(Rj).

The following information is stored:

Byte	Information
0	Reserved
1	Reserved
2	Reserved
3	Reserved
4	PROM microcode version
5	WCS microdoce version
6-9	4-byte system serial number

The system serial number is read from a system configuration PROM which includes a checksum byte. If the checksum is correct, ACF bit NUMBIT is set: if the checksum is not correct, NUMBIT is cleared.

Operand Types	Object Code (Hex) 1	2	3	4
Rj	9	5	j	0

Example

Label Field	OpC Field	Operand Field	Comment Field
	RPROM	R14	

RETURN

RTN

Type 1 Instruction

0000	0001
1	2

8 BITS

Detailed Description of Instruction Execution

Control is transferred to the location saved in the topmost entry of the return stack.

The return stack pointer is decremented.

If the return stack is empty when instruction execution begins, control traps to the Debug state with an error code of 1. The stack is empty when C(RSCWA) is less than or equal to the displacement of the first stack entry.

Operand Types	Object Code (Hex) 1	2
	0	1

Example

Label Field	OpC Field	Operand Field	Comment Field
	RTN		

SET BIT

SB Bj

Type 5 Instruction

1	2	3	4	5	6	7	8	
1101	1000	Bjr	0000	o	Bjd			32 BITS

Detailed Description of Instruction Execution

C(C(Bjr)+Bjd) is set to one.

Operand Types	Object Code (Hex) 1 2 3 4 5 6 7 8
Bj	D 8 j 0 o d d d

o is 8 plus the bit offset within the byte.

Example

Label Field	OpC Field	Operand Field	Comment Field
	SB	BTYPJ	

SET BIT WITH RELATIVE OFFSET

SB Rj,Wi

Type 5 Instruction

1101	0001	Wir	Rj	Wik	Wid			32 BITS
1	2	3	4	5	6	7	8	

Detailed Description of Instruction Execution

C(C(Rj)+C(C(Wir)+Wid)) is set to one.

Operand Types	Wik	Object Code (Hex) 1	2	3	4	5	6	7	8
Rj,Hi	0	D	1	i	j	0	d	d	d
Rj,Ti	0	D	1	i	j	1	d	d	d
Rj,Di	0	D	1	i	j	2	d	d	d
Rj,Fi	0	D	1	i	j	3	d	d	d

Example

```
Label   OpC      Operand          Comment
Field   Field    Field            Field
-----   -----    -------          -------

        SB       R7,HTYPEJ
```

SCAN STRING DECREMENTING UNDER DELIMITER CONTROL WITH COUNT

SDDC Ri,N

Type 4 Instruction

0110	1010	Ri	0000	N	
1	2	3	4	5	6

24 BITS

Detailed Description of Instruction Execution

C(Ri) is decremented by one. C(T0) is decremented by one. C(C(Ri)) is tested for a match with one of the characters specified by N. If the match is not successful, the operation is repeated.

Operand Types	Object Code (Hex) 1 2 3 4 5 6
Ri,N	6 A i 0 n n

Example

Label Field	OpC Field	Operand Field	Comment Field
	SDDC	R9,X'B'	

Programming Note: Assuming that T0 contains zero before the instruction is executed, when execution is completed, the number of characters scanned will be represented in T0 as a negative number.

Programming Note: The older mnemonic form SCDDC Ri,N produces the same object code.

SCAN STRING DECREMENTING UNDER DELIMITER CONTROL

SDD Ri,N

Type 4 Instruction

0110	1001	Ri	0000	N		24 BITS
1	2	3	4	5	6	

Detailed Description of Instruction Execution

C(Ri) is decremented by one. C(C(Ri)) is tested for a match with one of the characters specified by N. If the match is not successful, the operation is repeated.

Operand Types	Object Code (Hex) 1 2 3 4 5 6
Ri,N	6 9 i 0 n n

Example

Label Field	OpC Field	Operand Field	Comment Field
	SDD	R6,X'C'	

Programming Note: The older mnemonic form SCDD Ri,N produces the same object code.

SET TO ALL ONES

SET Wj

Type 5 Instruction

1100	0011	Wjr	0000	Wjk	Wjd			32 BITS
1	2	3	4	5	6	7	8	

Detailed Description of Instruction Execution

Each bit of C(C(Wjr)+Wjd) is set to a one, that is C(C(Wjr)+Wjd) is set to negative one.

Operand Types	Wjk	Object Code (Hex) 1	2	3	4	5	6	7	8
Hj	0	C	3	j	0	0	d	d	d
Tj	1	C	3	j	0	1	d	d	d
Dj	2	C	3	j	0	2	d	d	d
Fj	3	C	3	j	0	3	d	d	d
Vj	7	C	3	j	0	7	d	d	d

Example

Label Field	OpC Field	Operand Field	Comment Field
	SET	FTYPJ	SET TO -1

SHIFT FROM STORAGE

SHIFT Rj,Ri

Type 3 Instruction

1110	0101	Rj	Ri
1	2	3	4

16 BITS

Detailed Description of Instruction Execution

C(C(Ri)) is shifted right one bit with zero entering the high-order bit position.

The result replaces C(C(Rj)).

C(C(Ri)) is not changed.

Operand Types	Object Code (Hex) 1 2 3 4
Rj,Ri	E 5 j i

Example

Label Field	OpC Field	Operand Field	Comment Field
	SHIFT	R9,R5	

SCAN STRING INCREMENTING, COUNTING DELIMITERS

SICD Ri,Tj,N

Type 7 Instruction

1101	1111	Ri	Tjr	Tjk	Tjd		
1	2	3	4	5	6	7	8

N	40 BITS
9	10

Detailed Description of Instruction Execution

This instruction searches a string for a specified character. Ri points at the string, and SC1 contains the character. If the character is found, the tally Tj is decremented. The search continues until either Tj is decremented to zero or a specified system delimiter is found. The mask N must specify the search character in SC1, and it specifies any system delimiters to stop execution. SC0 and SC2 are not used by this instruction.

Operand Types	Wjk	Object Code (Hex) 1 2 3 4 5 6 7 8 9 0
Ri,Tj,N	1	D F i j 1 d d d n n

Example

Label Field	OpC Field	Operand Field	Comment Field
	SICD	R14,T3,X'42'	

Programming Note: The older mnemonic forms SCCD Ri,Tj,N and SCD Ri,Tj,N each produce the same object code.

SCAN STRING INCREMENTING UNDER DELIMITER CONTROL WITH COUNT

SIDC Ri,N

Type 4 Instruction

0100	1010	Ri	0000	N	
1	2	3	4	5	6

24 BITS

Detailed Description of Instruction Execution

C(Ri) is incremented by one. C(T0) is decremented by one. C(C(Ri)) is tested for a match with one of the characters specified by N. If the match is not successful, the operation is repeated.

Operand Types	Object Code (Hex) 1 2 3 4 5 6
Ri,N	5 2 i 0 n n

Example

Label Field	OpC Field	Operand Field	Comment Field
	SIDC	R5,9	

Programming Note: Assuming that T0 contains zero before the instruction is executed, when execution is completed, the number of characters scanned will be represented in T0 as a negative number.

Programming Note: The older mnemonic form SCDC Ri,N produces the same object code.

SCAN STRING INCREMENTING UNDER DELIMITER CONTROL

SID Ri,N

Type 4 Instruction

0100	1001	Ri	0000	N		24 BITS
1	2	3	4	5	6	

Detailed Description of Instruction Execution

C(Ri) is incremented by one. C(C(Ri)) is tested for a match with one of the characters specified by N. If the match is not successful, the operation is repeated.

Operand Types	Object Code (Hex) 1 2 3 4 5 6
Ri,N	4 9 i 0 n n

Example

Label Field	OpC Field	Operand Field	Comment Field
	SID	R15,3	

Programming Note: The older mnemonic form SCD Ri,N produces the same object code.

SCAN STRING UNDER DELIMITER CONTROL ACCUMULATING CHECK SUM

SIDX Ri,N

Type 4 Instruction

0100	1011	Ri	0000	N		24 BITS
1	2	3	4	5	6	

Detailed Description of Instruction Execution

C(Ri) is incremented by one. C(C(Ri)) is tested for a match with one of the characters specified by N. If the match is unsuccessful, C(C(Ri)) is added to C(T0) and the operation is repeated; otherwise, instruction execution terminates. Hence, the terminating character is not added to the check sum.

Operand Types	Object Code (Hex) 1 2 3 4 5 6
Ri,N	4 B i 0 n n

Example

Label Field	OpC Field	Operand Field	Comment Field
	SIDX	R10,X'A'	

Programming Note: If T0 contains zero before instruction execution begins, a two-byte check sum of all characters scanned will be accumulated in T0.

Programming Note: The older mnemonic form SCDX Ri,N produces the same object code.

INCREMENT AND SCAN STRING UNDER T0 AND DELIMITER CONTROL

SITD Ri,N

Type 4 Instruction

0101	0011	Ri	0000	N	
1	2	3	4	5	6

24 BITS

Detailed Description of Instruction Execution

If C(T0) equals zero initially, no operation is performed.

C(Ri) is incremented by one. C(T0) is decremented by one. If C(T0) is zero or if C(C(Ri)) matches one of the characters specified by N, execution ceases. If the match is not successful, the operation is repeated.

Operand Types	Object Code (Hex) 1 2 3 4 5 6
Ri,N	5 3 i 0 n n

Example

Label Field	OpC Field	Operand Field	Comment Field
	SITD	R15,3	

Programming Note: This instruction may be used to limit the maximum number of characters scanned. T0 can be initialized with a maximum count before the instruction is executed. After execution the number of characters scanned may be computed by subtracting the final count in T0 from the initial count.

SET PCB ADDRESS LINKED

SPCBL Rj,N

Type 3 Instruction

0011	1001	Rj	N
1	2	3	4

16 BITS

Detailed Description of Instruction Execution

This instruction sets a register pointing to a linked workspace that starts in one of the process's primary workspace frames. The instruction

adds N to the FID that R0 points to and

sets the C(Rj):

linked,

displacement points one byte before data byte one of a linked frame,

FID-field contains the calculated FID.

Operand Types	Object Code (Hex) 1 2 3 4
Rj,N	3 9 j n

Example

Label Field	OpC Field	Operand Field	Comment Field
	SPCBL	R7,39	GET DATA STACK READ

Programming Note: This instruction presumes a knowledge of the format of a process's primary workspace.

SET PCB ADDRESS UNLINKED

SPCBU Rj,N

Type 3 Instruction

0011	1000	Rj	N
1	2	3	4

16 BITS

Detailed Description of Instruction Execution

This instruction sets a registers to point to an unlinked block in one of the process's primary workspace frames. The instruction

adds N to the FID that R0 points to and

sets the C(Rj):

unlinked,

displacement points to byte zero of the unlink frame,

FID-field contains the calculated FID.

Operand Types	Object Code (Hex) 1 2 3 4
Rj,N	3 8 j n

Example

Label Field	OpC Field	Operand Field	Comment Field
	SPCBU	R11,28	GET USER CONTROL BLOCK

Programming Note: This instruction presumes a knowledge of the format of a process's primary workspace.

SET PIB ADDRESS

SP Rj

Type 3 Instruction

0010	0111	Rj	0000
1	2	3	4

16 BITS

Detailed Description of Instruction Execution

The special FID of the PIB is placed into Rj's FID field. The displacement of the PIB in the buffer is placed into Rj's displacement field. The flag field is set to X'80' to indicate an unlinked frame, and the register is attached.

Operand Types	Object Code (Hex) 1	2	3	4
Rj	2	7	j	0

Example

Label Field	OpC Field	Operand Field	Comment Field
	SP	R11	

Programming Note: Unlike most of the data associated with a virtual process, PIBs do not reside on disc; they reside only in main memory. In order to allow a virtual process to reference a PIB through the address registers with the usual fields, the Configurator program puts special FIDs into the Buffer Map for the buffers that hold PIBs. These special FIDs have the format:

X'EFnnnn'

where nnnn is the process number.

SET REGISTER TO ADDRESS

SRA Rj,Wi

Type 5 Instruction

1100	1000	Wir	Rj	Wik	Wid			32 BITS
1	2	3	4	5	6	7	8	

Detailed Description of Instruction Execution

The effective address, C(Wir)+Wid, is computed. The resulting effective address replaces C(Rj).

Operand Types	Wik	Object Code (Hex) 1	2	3	4	5	6	7	8
Rj,Ci	0	C	8	i	j	0	d	d	d
Rj,Hi	0	C	8	i	j	0	d	d	d
Rj,Ti	1	C	8	i	j	1	d	d	d
Rj,Di	2	C	8	i	j	2	d	d	d
Rj,Fi	3	C	8	i	j	3	d	d	d
Rj,Si	3	C	8	i	j	3	d	d	d

Example

Label Field	OpC Field	Operand Field	Comment Field
	SRA	R15,STYPJ	PUT ADR OF STYPJ IN R15

STORE ACCUMULATOR

STORE Wj

Type 5 Instruction

1100	0010	Wjr	0000	Wjk	Wjd			32 BITS
1	2	3	4	5	6	7	8	

Detailed Description of Instruction Execution

The contents of the 32-bit accumulator (D0) are stored into the location defined by the operand if the word type of the operand is H,T, or D. That is, C(D0) replaces C(C(Wjr)+Wjd) for operands of four bytes or less. For half tally and tally operands, the high order bits are lost.

For a six-byte operand, C(FP0) replaces C(C(Wjr)+Wjd).

Operand Types	Wjk	Object Code (Hex) 1	2	3	4	5	6	7	8
Hj	0	C	2	j	0	0	d	d	d
Tj	1	C	2	j	0	1	d	d	d
Dj	2	C	2	j	0	2	d	d	d
Fj	3	C	2	j	0	3	d	d	d
Vj	7	C	2	j	0	7	d	d	d

Example

Label Field	OpC Field	Operand Field	Comment Field
	STORE	FTYPEJ	STORE INTO FP0
	STORE	TTYPI	STORE INTO (LOW ORDER WORD)

SUBTRACT FROM ACCUMULATOR

SUB Wi

Type 5 Instruction

1010	0101	Wir	0000	Wik	Wid			32 BITS
1	2	3	4	5	6	7	8	

Detailed Description of Instruction Execution

The integer addressed by the operand is subtracted from the 32-bit accumulator (D0) with sign extension. That is, C(D0) - C(C(Wir)+Wid) replaces C(D0).

C(ACF) is updated to reflect overflow.

Operand Types	Wik	Object Code (Hex) 1	2	3	4	5	6	7	8
Hi	0	A	5	i	0	0	d	d	d
Ti	1	A	5	i	0	1	d	d	d
Di	2	A	5	i	0	2	d	d	d
Vi	7	A	5	i	0	7	d	d	d

Example

Label Field	OpC Field	Operand Field	Comment Field
	SUB	HTYPI	SUBTRACT FROM D0

Programming Note: The mnemonic SUB may be used with F type operands (six-byte elements), but the object code generated by the assembler is the same as that generated for instruction SUBX.

SUBTRACT FROM EXTENDED ACCUMULATOR

SUBX Wi

Type 5 Instruction

1010	0100	Wir	0000	Wik	Wid			32 BITS
1	2	3	4	5	6	7	8	

Detailed Description of Instruction Execution

C(C(Wir)+Wid) is subtracted algebraically from C(FP0) and this difference replaces C(FP0). That is, C(FP0) - C(C(Wir)+Wid) replaces C(FP0).

C(ACF) is updated to reflect overflow.

Operand Types	Wik	Object Code (Hex) 1	2	3	4	5	6	7	8
Hi	0	A	4	i	0	0	d	d	d
Ti	1	A	4	i	0	1	d	d	d
Di	2	A	4	i	0	2	d	d	d
Fi	3	A	4	i	0	3	d	d	d
Vi	7	A	4	i	0	7	d	d	d

Example

Label Field	OpC Field	Operand Field	Comment Field
	SUBX	DTYPI	SUBTRACT FROM FP0
	SUBX	HTYPI	SUBTRACT (BYTE) FROM FP0

SWAP STORAGE ELEMENTS

SWAP Wi,Wj

Type 6 Instruction

1	2	3	4	5	6	7	8
1111	1010	Wir	Wjr	Wik	Wid		

9	10	11	12
Wjk	Wjd		

48 BITS

Detailed Description of Instruction Execution

C(C(Wir)+Wid) and C(C(Wjr)+Wjd) are swapped.

Operand Types	Wik	Wjk	Object Code (Hex) 1	2	3	4	5	6	7	8	9	0	1	2
Ci,Cj	0	0	F	A	i	j	0	d	d	d	0	d	d	d
Hi,Hj	0	0	F	A	i	j	0	d	d	d	0	d	d	d
Ti,Tj	1	1	F	A	i	j	1	d	d	d	1	d	d	d
Di,Dj	2	2	F	A	i	j	2	d	d	d	2	d	d	d
Fi,Fj	3	3	F	A	i	j	3	d	d	d	3	d	d	d
Si,Sj	3	3	F	A	i	j	3	d	d	d	3	d	d	d
Vi,Vj	7	7	F	A	i	j	7	d	d	d	7	d	d	d

Example

```
Label   OpC      Operand          Comment
Field   Field    Field            Field
-----   -----    -------          -------

        SWAP     HTYPI,HTYPJ
        SWAP     CTYPI,CTYPJ
        SWAP     TTYPI,TTYPJ
        SWAP     DTYPI,DTYPJ
        SWAP     FTYPI,FTYPJ
        SWAP     STYPI,STYPJ
```

UNLOCK COMPETING PROCESSES

UNLOCK N

Type 2 Instruction

0000	0010	N	
1	2	3	4

16 BITS

Detailed Description of Instruction Execution

This instruction opens locks set by the LOCK N instruction.

N is the lock number.

The instruction handles two conditions.

If lock N contains the unlocked value or the process number of another process, this instruction falls through to the next instruction in sequence.

If lock N contains the process number of the executing process, the firmware puts the unlocked value into lock N and searches the Priority Queue for all PIBs with the value N in the PIB lock number hold byte. For each PIB with N the firmware removes the lock roadblock.

Operand Types	Object Code (Hex) 1 2 3 4
N	0 2 n n

Example

Label Field	OpC Field	Operand Field	Comment Field
	UNLOCK	OVRFLW*	

WRITE FILLER CHARACTERS

WRITEF

Type 1 Instruction

0010	1110
1	2

8 BITS

Detailed Description of Instruction Execution

This instruction causes the CPU firmware to send nulls to the process's terminal.

The characters are transmitted in a manner similar to writing data to the terminal using the WRITE instruction, except that no registers are involved.

The following input interface is required :

Element	Description
T0	Number of nulls to send

C(T0) is undefined after execution of this instruction.

If the process is a TIPH process, this instruction is a NOP.

Operand Types	Object Code (Hex) 1	2
	2	E

Example

```
Label   OpC      Operand          Comment
Field   Field    Field            Field
-----   -----    -------          -------

        LOAD     20               SEND 20 NULLS
        WRITEF   *                TO TERMINAL
```

WRITE NEW LINE

WRITEOL

Type 1 Instruction

0010	1111	8 BITS
1	2	

General Description of Instruction Execution

Because this instruction is quite complex, a general description useful to assembly language programmers is presented here, and further details are provided below.

This instruction outputs a carriage return, line feed and nulls to the process's terminal.

The CPU uses the DELYCNT field in the PIB for the null character count. The maximum count that may be specified in DELYCNT is 127. Anything larger will cause the instruction to abort into the Software Debugger.

If the process is a TIPH process, that is, if the PIB TRAP.WRITES bit is set to 1, this instruction traps to the Software Debugger and does not perform any I/O.

Operand Types	Object Code (Hex) 1	2
	2	F

Example

Label Field	OpC Field	Operand Field	Comment Field
	WRITEOL	*	WRITE NEW LINE

WRITE NEW LINE (cont)

Detailed Description of Instruction Execution

This instruction uses the same firmware logic as the WRITE instruction for putting characters into the output buffer.

The firmware will first check to see if the corelocked output buffer has enough free space for the two end-of-line and DELYCNT null characters. If there is not enough room, the output roadblock is set, the program counter is set to the WRITEOL instruction, the process is deactivated, and the firmware enters the Monitor to release the process's time quantum. When the output buffer has enough room, the output roadblock will be cleared and the WRITEOL instruction will be executed.

If there is enough room for the characters, the firmware puts them into the output buffer. If the buffer is empty when the instruction is executed, the firmware sets up the Write task descriptor and issues it to the IOP2 to write the characters. The firmware then goes to RNI to execute the next instruction.

For a TIPH process, this instruction sets two PCB fields before entering the Software Debugger.

WRITE

WRITE Ri,Rj

Type 3 Instruction

0101	1100	Ri	Rj
1	2	3	4

16 BITS

General Description of Instruction Execution

Because this instruction is quite complex, a general description useful to assembly language programmers is presented here, and further details are provided below.

This instruction commands the IOP2 to transfer characters to a process's terminal. Ri points to the first byte of a character string, and Rj points to the last character of the string.

If Ri points to a byte in an unlinked frame, Rj must point to the same or a subsequent byte in the same frame. If Ri points to a byte in a linked frame, Rj must point to the same or a subsequent byte in the same linked set. If either of these rules is broken, a CROSSING FRAME LIMIT abort is generated.

This instruction is restricted to virtual mode. If it is executed in monitor mode, the Firmware Debugger is entered.

If the virtual process is a TIPH process, that is, if the PIB PHANTOM bit is set to 1, the WRITE instruction traps to the Software Debugger and does not perform any I/O.

Operand Types	Object Code (Hex) 1 2 3 4
Ri,Rj	5 C i j

Example

Label Field	OpC Field	Operand Field	Comment Field
	WRITE	R14,R15	

Programming notes: As much as possible, output characters in strings, not individually. Use the instruction WRITEOL to output carriage return, line feed, and nulls. Use WRITEF to output nulls.

WRITE (cont)

Detailed Description of Instruction Execution

This instruction commands the IOP2 to transfer data from main memory to a terminal. If the corelocked output buffer is full (the buffer pointed to by the OUTPUT.BUF field in the Terminal I/O Workspace), the output roadblock is set (OBYTEBLK/ is zeroed) and a firmware release quantum entry to the Monitor is taken. When the output buffer becomes at least half-empty the output roadblock will be cleared. When the process is re-activated, the WRITE instruction will be re-executed.

The bytes from Ri to Rj, inclusive, are displayed on the terminal of the process executing the WRITE. The details of the WRITE instruction's function are governed by the relationship between the two registers, the number of characters to output, and the amount of free space in the output buffer. These are the possible cases:

1. The first register points before the second and the output buffer has room for all the bytes to transmit. If the registers are not pointing at the same frame, the frames must be linked.

   The CPU firmware moves the data from the virtual process's buffer to the corelocked output buffer and advances the buffer's producer pointer (the TO.PROD field in the Terminal I/O Workspace) by the number of bytes moved. If the output buffer is empty when the instruction is executed, the Write task descriptor is set up and the firmware commands the IOP2 to write the data to the terminal. The first register (Ri) is set pointing to the last byte to be output. The firmware returns to RNI to execute the next instruction.

2. The first register points before the second register and there are more characters to transmit than there is free space in the output buffer. If the registers are not pointing at the same frame, the frames must be linked.

   The firmware moves data from the process's buffer into the corelocked output buffer until it is full. If the buffer is empty when this instruction is executed, the firmware issues a write command to the IOP2. Output roadblock is set. The program counter is set pointing at the WRITE instruction. Ri is left pointing at the next data to be transmitted, and the process is deactivated. The firmware enters the Monitor to release the process's time quantum.

WRITE (cont)

When the interrupt signaling completion is received, the firmware will advance the output buffer's consumer pointer (the TO.CONS field in the Terminal I/O Workspace) by the number of bytes written. If there are more data in the buffer, the task descriptor is re-issued to write them. If that makes the buffer at least half-empty, the output roadblock is cleared; otherwise, the process is not reactivated until the next interrupt is received. When the process is reactivated, the WRITE instruction is re-executed so that output may continue from where it left off. This continues until the data are reduced to fewer bytes than the free space in the output buffer, at which time case 1 applies.

Use of the WRITE instruction in this way enables a process to output an unlimited number of characters with one command.

3. The registers point to different frames, and the first register is in unlinked format; or they point to the same frame, and the second register is before the first.

   A CROSSING FRAME LIMIT abort is generated.

Terminal I/O errors are reported by the IOP2 to the firmware via the Status task descriptor and never by the Write task descriptor itself. Therefore, when a terminal error is reported, the Write task descriptor remains active and the process enters the Software Debugger. The firmware transfers to Software Debugger entry-point 10 for the BREAK key and to entry-point 12 for other terminal errors. If one or more of PRGERR (PIB byte 1, bit 2) or INDEBUG (PIB byte 1, bit 1) or INCCB (PIB byte X'10', bit 0) or PHANTOM (PIB byte 0, bit 7) is set to 1, BREAK key is ignored.

Handshaking errors between the controllers and the CPU may occur during I/O operations. These errors may be reported using the Write task descriptor as well as the Status task descriptor. Whenever the firmware detects a handshake error, it aborts to entry point 20 of the Software Debugger.

The Monitor may not use the WRITE instruction for terminal output; it must issue terminal I/O task descriptors in the same manner that it does for other I/O devices. If this instruction is executed in monitor mode, the Firmware Debugger will be entered.

If the process is a TIPH process, that is, if the PIB PHANTOM bit is set to 1, the WRITE instruction with one exception traps to the Software Debugger and does not perform any I/O. The exception is when TIPH.OPT is 1. This means the TIPH process is using the T option to write to a terminal instead of to a Spooler file. The firmware transfers the data to the terminal and does not enter the debugger.

WRITE (cont)

For all other TIPH cases the firmware enters the debugger. First, all the error checks are made. PIB field PH.WRITER1 will be set pointing to the first byte of the string to be written. PH.WRITER2 will be set pointing to the last byte of the string. Ri will be updated to point to the last byte of the string and the Software Debugger entered at entry point 19.

EXCHANGE CHARACTERS

XCC Ri,Rj

Type 3 Instruction

1110	1010	Ri	Rj
1	2	3	4

16 BITS

Detailed Description of Instruction Execution

C(C(Ri)) and C(C(Rj)) are exchanged.

Operand Types	Object Code (Hex) 1 2 3 4
Ri,Rj	E A i j

Example

Label Field	OpC Field	Operand Field	Comment Field
	XCC	R8,R9	

XOR WITH IMMEDIATE

XOR Rj,N
XOR N,Rj

Type 4 Instruction

1000	0011	Rj	0000	N	
1	2	3	4	5	6

24 BITS

Detailed Description of Instruction Execution

C(C(Rj)) are logically exclusive ORed with N.

The result replaces C(C(Rj)).

Operand Types	Object Code (Hex) 1 2 3 4 5 6
Rj,N	8 3 j 0 n n
N,Rj	8 3 j 0 n n

Example

Label Field	OpC Field	Operand Field	Comment Field
	XOR	R8,7	
	XOR	X'C',R5	

XOR WITH STORAGE

XOR Rj,Ri

Type 3 Instruction

1110	0011	Rj	Ri
1	2	3	4

16 BITS

Detailed Description of Instruction Execution

C(C(Rj)) are logically exclusive ORed with C(C(Ri)).

The result replaces C(C(Rj)).

Operand Types	Object Code (Hex) 1 2 3 4
Rj,Ri	E 3 j i

Example

Label Field	OpC Field	Operand Field	Comment Field
	XOR	R10,R4	

XOR ELEMENT WITH ELEMENT

XOR Wi,Wj

Type 6 Instruction

1111	1101	Wir	Wjr	Wik	Wid		
1	2	3	4	5	6	7	8

Wjk	Wjd		
9	10	11	12

48 BITS

Detailed Description of Instruction Execution

C(C(Wir)+Wid) is eXclusively ORed with C(C(Wjr)+Wjd). The result replaces C(C(Wjr)+Wjd).

Operand Types	Wik	Wjk	Object Code (Hex) 1	2	3	4	5	6	7	8	9	0	1	2
Ci,Cj	0	0	F	D	r	r	0	d	d	d	0	d	d	d
Hi,Hj	0	0	F	D	r	r	0	d	d	d	0	d	d	d
Ti,Tj	1	1	F	D	r	r	1	d	d	d	1	d	d	d
Di,Dj	2	2	F	D	r	r	2	d	d	d	2	d	d	d
Fi,Fj	3	3	F	D	r	r	3	d	d	d	3	d	d	d
Vi,Vj	7	7	F	D	r	r	7	d	d	d	7	d	d	d

Example

```
Label   OpC      Operand          Comment
Field   Field    Field            Field
-----   -----    -------          -------

        XOR      HTYPI,HTYPJ
        XOR      CTYPI,CTYPJ
        XOR      TTYPI,TTYPJ
        XOR      DTYPI,DTYPJ
        XOR      FTYPI,FTYPJ
```

EXCHANGE ADDRESS REGISTERS

XRR Ri,Rj

Type 3 Instruction

0000	1010	Ri	Rj
1	2	3	4

16 BITS

Detailed Description of Instruction Execution

C(Ri) and C(Rj) are exchanged. All eight bytes of each register are exchanged. hence, if Ri was attached, Rj becomes attached; otherwise it is detached. If Rj was attached, Ri becomes attached; otherwise, it is detached.

Operand Types	Object Code (Hex) 1 2 3 4
Ri,Rj	0 A i j

Example

Label Field	OpC Field	Operand Field	Comment Field
	XRR	R5,R6	

ZERO BIT

ZB Bj

Type 5 Instruction

1101	0000	Bjr	0000	o	Bjd			32 BITS
1	2	3	4	5	6	7	8	

Detailed Description of Instruction Execution

C(C(Bjr)+Bjd) is set to zero.

Operand Types	Object Code (Hex) 1 2 3 4 5 6 7 8
Bj	D 0 j 0 o d d d

o is 8 plus the bit offset within the byte.

Example

Label Field	OpC Field	Operand Field	Comment Field
	ZB	BTYPJ	

ZERO BIT WITH RELATIVE OFFSET

ZB Rj,Hi

Type 5 Instruction

1	2	3	4	5	6	7	8	
1101	0000	Hir	j	Hik	Hid			32 BITS

Detailed Description of Instruction Execution

C(C(Rj)+C(C(Hir)+Hid)) is set to zero.

Operand Types	Hik	Object Code (Hex) 1 2 3 4 5 6 7 8
Rj,Hi	0	D 0 i j 0 d d d

Example

Label Field	OpC Field	Operand Field	Comment Field
	ZB	R9,HTYPEJ	

STORE A ZERO

ZERO Wj

Type 5 Instruction

1100	0000	Wjr	0000	Wjk	Wjd			32 BITS
1	2	3	4	5	6	7	8	

Detailed Description of Instruction Execution

C(C(Wjr)+Wjd) is replaced by binary zeros.

C(ACF) is NOT changed.

Operand Types	Wjk	Object Code (Hex) 1	2	3	4	5	6	7	8
Cj	0	C	0	j	0	0	d	d	d
Hj	0	C	0	j	0	0	d	d	d
Tj	1	C	0	j	0	1	d	d	d
Dj	2	C	0	j	0	2	d	d	d
Fj	3	C	0	j	0	3	d	d	d
Sj	3	C	0	j	0	3	d	d	d
Vj	7	C	0	j	0	7	d	d	d

Example

Label Field	OpC Field	Operand Field	Comment Field
	ZERO	CTYPJ	ZERO THE CONTENTS OF CTYPJ

INDEX

INDEX

7.0 VIRTUAL ASSEMBLY LANGUAGE CPU INSTRUCTIONS

INDEX

INDEX

INDEX

INDEX

INDEX