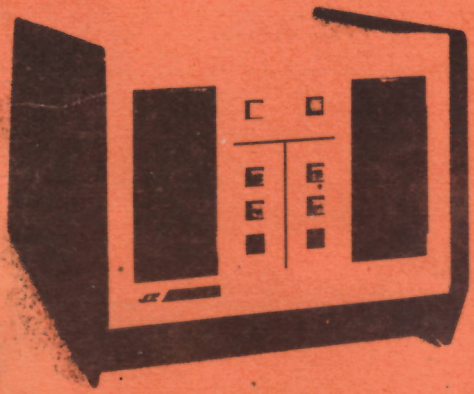
INCOTERM®

SPD® Symbolic Assembly Language Reference Manual



Växjö Data System AB

This manual, assembled in March 1976 by VÄXJÖ DATA SYSTEM AB, SWEDEN, consists of

1. SPD Symbolic Assembly Language Reference Manual MS-7215.0
2. SPD Relocatable Assembly System (75-06-24)

Växjö March 1976

SPD SYMBOLIC

ASSEMBLY LANGUAGE

REFERENCE MANUAL

ORDER NUMBER: MS-7215.0

DATE: AUGUST, 1975

PREFACE

This manual supercedes the SPD 10/20 Assembler Manual, MS-7113, and is the only assembler manual required for the SPD 10/20, SPD 10/24, SPD 10/25, and SPD 20/20.

It should be noted that the SPD 10/24 was especially designed for overseas customers. This system is built and sold solely by a licensee, TRANSAC. The SPD 10/24 is not available within the U. S. A.

Table of Contents

Table of Contents
(cont'd)

Table of Contents
(cont'd)

Table of Contents
(cont'd)

List of Illustrations

List of Tables

SECTION I

INTRODUCTION

GENERAL

The SPD symbolic assembly language is designed to aid in the preparation of programs for execution on any of the following INCOTERM processing units:

SPD 10/20
SPD 10/24
SPD 10/25
SPD 20/20

The assembler relieves the programmer of many of the burdensome tasks which would be associated with programming in machine language while permitting full access to the capabilities of the processing unit.

Mnemonic operation codes are provided for the various machine instructions and the assembler outputs the appropriate bit patterns. A location counter maintained by the assembler allows symbols to be defined as labels for instructions. These labels may be referenced by other instructions to obtain the corresponding location values, thus relieving the programmer of the task of assigning storage locations. A separately maintained load counter controls the final load location. The load counter's value is normally the same as that of the location counter, but the values may be separated to allow assembly of code at one location for eventual execution at some other location.

For locations equipped with an SPD D-250 Dual Diskette and printer, the SPD DOS Assembler is available (See also Appendix D). This assembler translates SPD assembly language into machine language. It is basically a utility program which runs under control of the SPD/DOS system on an SPD 10/20, 10/24, 10/25, or 20/20. The output is a DOS loadable object file on disk.

The DOS COPY utility can be used to obtain object copies on other media.

Notation

A word surrounded by angle brackets, e.g. <label> represents a generic type of operand. The ensuing text describes the allowed possibilities for the operand.

A section of syntax enclosed in square brackets, e.g. [code], indicates an optional part of the construction which may be omitted. The text explains the effect of including or excluding such components.

SECTION II

ASSEMBLY SOURCE FORMAT

GENERAL

An SPD assembly language program consists of a series of lines each containing a single statement. The length of these lines is 1-80 characters, although for convenient use of the SPD/DOS EDIT facility, it may be desirable to restrict the length of all lines to 64 characters for compatibility with the SPD 10/20 screen format.

CHARACTER SET

The character set used is the upper case ASCII set (corresponding to codes X'20' - X'5F'). Any of the 64 characters of this set may appear in character and text constants, but only the following have specific syntactic use:

$	dollar
&	ampersand
'	quote
(	left parenthesis
)	right parenthesis
*	asterisk
#	hash
+	plus
,	comma
-	minus
.	period
/	slash
0-9	digits
=	equal
@	at
A-Z	letters

OPERATION FORMAT

The basic operation format is shown below:

label opcode operands comments

The label, if present, starts in column 1. The first character of the label must be a letter and the remaining characters are from the set 0-9 (digits), A-Z (letters), $ (dollar) and & (ampersand). The label length is from 1-8 characters. If no label is present, column one must be blank.

The opcode is either a machine operation or pseudo-operation mnemonic. It normally starts in column 10 and it must be separated by at least one blank from the label or, if there is no label, must start in or after column 2. In the event that indirect addressing is permitted, it is specified by immediately following the opcode with * (asterisk).

The operands field, if present, consists of one or more operands separated by commas. The number of operands depends on the opcode as described in Section IV. No blanks may appear in the operands except within character or text constants, thus a blank is used as a terminator. The operands field must be separated by at least one and not more than nine blanks from the opcode field. In the case where indirect addressing is specified, no separating blanks are required, but they may be present.

The comments field, which is optional, consists of any sequence of ASCII characters including blanks. The first character of this field must be separated from the operands field by at least one blank. In the case where there is no operands field, the comment field must be separated from the opcode field by at least ten blanks.

Although the above rules allow "free format" input, standard INCOTERM coding conventions require the following format to be followed where possible. See Figure 2-1 for a sample coding form.

cols 1-8	label
cols 10-15	opcode
cols 16-29	operands
cols 30-n	comment

COMMENT LINES

Any assembly source line which contains * (asterisk) or # (hash) in column one is treated as a comment by the assembler. It will be printed if the listing mode is specified (see LIST, LIF pseudo-operations) but will otherwise be ignored. It is also possible to selectively list only the comments starting with *.

TITLE LINES

The printed output has two title lines of forty characters each. The main title is initially set from the source file label, or if no such label is

available, from columns 2-41 of the initial source line.

Subsequently, this main title can be modified by the occurrence of one or more title modification lines in the source program with the following format:

cols 1	& (ampersand)
cols 2-41	title modification data

Blanks in columns 2-41 have no effect. Non-blank characters replace the corresponding character positions of the main title. These title modification lines are interpreted in order in pass one of the assembly and the resulting title used to label the object output and to head the printed output throughout pass two (i. e. title modification lines have no effect in pass two).

Title modification lines are not recognized if deleted by IF-ENDF or DUP. Thus conditional assembly operations may be used to select an appropriate main title.

CHARGE NO. _____

DATE REQ'D _____

INCOTERM® CORPORATION
6 STRATHMORE ROAD, NATICK, MASS. 01760

SPD™ 10/20
ASSEMBLY PROGRAM FORM

FLOW CHART REFERENCE _____

DATE _____ PAGE _____ OF _____

PROGRAM _____

PROGRAMMER _____

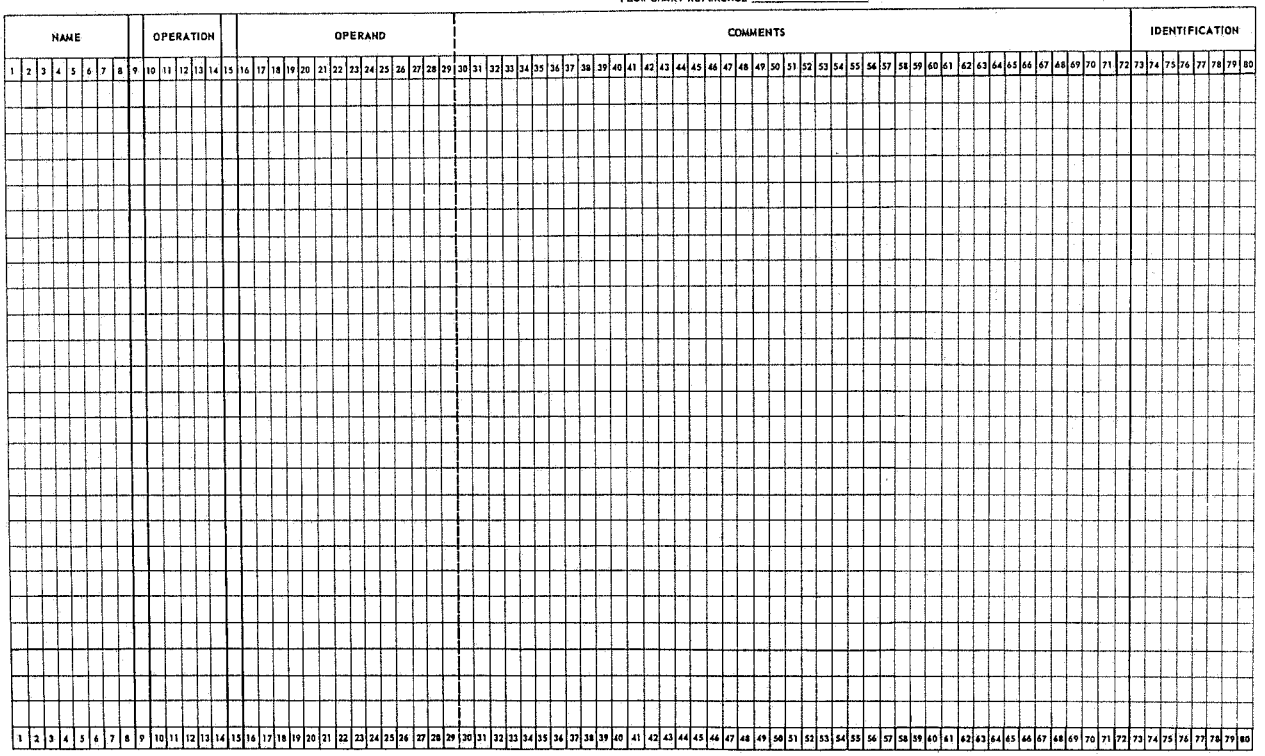


Figure 2-1. Sample Coding Form

The sub-title printed on the listing is initially blank. It may be set by a sub-title line in the source with the following format:

col 1	' (quote)
cols 2-41	sub-title data (40 ASCII characters)

The occurrence of such a line implies an automatic page eject so that subsequent pages (up to the next sub-title line in the source) are printed with the correct sub-title. Sub-title lines are only effective if they occur in a listed section of code (see LIST, LIF pseudo-operations).

SECTION III

EXPRESSIONS

GENERAL

The power of a symbolic assembler lies in its ability to reference values in a symbolic manner. This is achieved by allowing expressions to be used wherever a value is required (e.g. operand location, I/O function code, immediate operand value).

Expressions may either be simple operands (e.g. constants or label references), or they may be calculations involving the use of operators. It is important to understand that such calculations are performed entirely at assembly time and do not correspond to the execution of code at execution time.

OPERANDS

Operands may stand on their own as expressions or they may be joined together with operators to form complex expressions.

Symbols

Any symbol which has been given a value somewhere, by appearing in the label field of a suitable machine or pseudo-operation line, may be used as an operand. The value is the symbol value which is normally (i.e. except in the special cases of EQU, SET, SEG) the location counter value at the definition point.

The syntax of symbol references is the same as that previously described for labels. The length of symbols varies from 1-8 characters and the reference

must be identical to the definition.

Due to the two-pass nature of the assembler, labels may usually be referenced before they are defined. In certain pseudo-operations, such "forward" references are not permitted. These exceptions are noted in the pseudo-operation section.

Constants

Constant operands may be in any of the following formats:

nnn...	Decimal integer. The n's are decimal digits (0-9). Values larger than 65535 are interpreted modulo 65536.
D'nnn...'	An alternate form for a decimal integer. Used particularly in conjunction with repeat counts in ADDR, BYTE, WORD operands (see separate descriptions).
B'bbbb...'	Binary integer. The b's are binary digits (0 or 1). The integer may have one or more digits, the result being right justified. Only the low order 16-bits are retained.
O'cc...'	Octal integer. The c's are octal digits (0-7). The integer may have one or more digits. Numbers larger than O'177777' are interpreted modulo O'200000'.
X'xxx...'	Hexadecimal integer. The x's are hexadecimal digits (0-9, A-F). The number may have one or more digits. Only the low order four digits are retained, the result being right justified if fewer than four digits are given.
'a'	Single character ASCII constant. a is any legal ASCII character except quote. The value of the operand is the corresponding ASCII code for the character. If a constant quote is required, it can be expressed in hexadecimal (i.e. X'27').
'ab'	Double character ASCII constant. a, b are any legal ASCII characters except quote. The value of the operand is 256* (ASCII code for a) + (ASCII code for b).

Special Symbols

Special symbols are predefined as follows and may be used as operands.

$ The current value of the location counter. This is always the value at the start of the referencing line before generating any code or otherwise interpreting the line.

$B A sixteen-bit value which can be set externally without affecting the source. The method of setting $B depends on the assembler in use (see separate appendices on specific assemblers). The use of $B allows one source program to assemble more than one version of a program by means of conditional assembly operations.

$C The location of the cursor register: X'0FFE' in CNFG 10 or CNFG 25 assembly, X'7FFE' in a CNFG 0, CNFG 24, or CNFG 20 assembly. $C has the same value as $X but $C is preferred usage if the register is being used as a cursor register.

$D Duplication count. When a line is duplicated by means of the DUP pseudo-operation, $D has successive values of 1, 2, 3,...for duplications of the line. The value of $D is zero in non-DUPed lines.

$L The current value of the load counter. This is always the value at the start of the referencing line before generating any code or otherwise interpreting the line.

$X The location of the index register: X'0FFE' in CNFG 10 or CNFG 25 assembly, X'7FFE' in CNFG 0, CNFG 20, or CNFG 24 assembly. $X has the same value as $C but $X is preferred usage when the register is used as an index register.

** An alternate form of a constant zero operand. This form is conventionally used when the value is to be set at execution time (e.g. DAC ** for the entry point of a subroutine called by JSR.)

OPERATORS

Operators are written between two operands and perform a computational operation on the two operand values. The following paragraphs describe the available operators in alphabetical order.

Addition

The addition operator + adds its two operands as 16-bit unsigned quantities, ignoring overflow. This is equivalent to a twos- complement add if the operands are regarded as signed twos-complement values.

Examples:

```
2+2                 = 4
X'8000'+X'8000'     = 0
4+X'FFFF'           = 3
X'4000'+X'4000'     = X'8000'
```

Compare Equal

The compare equal operator .EQ. compares its two 16-bit operands and gives a result of one if the operands are identical and zero otherwise.

Examples:

```
X'FFFF'.EQ.X'FFFF'  = 1
2.EQ.1              = 0
```

Compare Greater Than or Equal

The compare greater than or equal operator .GE. compares its two 16-bit operands as unsigned integers and gives a result of one if the left operand is greater than or equal to its right operand and zero otherwise.

Examples: 3.GE.3 = 1

X'FFFF'.GE.0 = 1

1.GE.2 = 0

Compare Greater Than

The compare greater than operator .GT. compares its two 16-bit operands as unsigned integers and gives a result of one if the left operand is strictly greater than its right operand and zero otherwise.

Examples: 3.GT.3 = 0

X'FFFF'.GT.0 = 1

1.GT.2 = 0

Compare Less Than or Equal

The compare less than or equal operator .LE. compares its two 16-bit operands as unsigned integers and gives a result of one if the left operand is less than or equal to the right operand and zero otherwise.

Examples: 3.LE.3 = 1

X'FFFF'.LE.0 = 0

1.LE.2 = 1

Compare Less Than

The compare less than operator .LT. compares its two 16-bit operands as unsigned integers and gives a result of one if the left operand is strictly less than the right operand and zero otherwise.

Examples:

```
3.LT.3          = 0
X'FFFF'.LT.0    = 0
1.LT.2          = 1
```

Compare Not Equal

The compare not equal operator .NE. compares its two 16-bit operands and gives a result of zero if they are identical and one otherwise.

Examples:

```
3.NE.3          = 0
X'FFFF'.NE.0    = 1
```

Division

The division operator / yields the quotient of the result of dividing the left operand by the right operand. Both operands are regarded as unsigned 16-bit operands and the result is truncated to an integer by dropping the fractional part. Division by zero is undefined and results in an error.

Examples:

```
3/2             = 1
3200/100        = 32
X'1234'/256     = X'0012'
X'FEDC'/256     = X'00FE'
1/X'FFFF'       = 0
2/0             = (error)
```

Left Shift

The left shift operator .LS. shifts its left operand to the left the number of bits indicated by the right operand. The shift is a logical end-off shift with zero bits entering on the right as required. The shift count must be in the range 0-15 inclusive.

Examples:

```
B'101010'.LS.2 = B'10101000'
X'FF7F'.LS.4   = X'F7F0'
320.LS.1       = 640
12.LS.16       = (error)
12.LS.X'FFFF'  = (error)
```

Logical And

The logical and operator .AND. performs a 16-bit bit-wise "and" operation on its two operands.

Examples:

```
'34'.AND.X'0F0F'    = X'0304'
B'0011'.AND.0101'   = B'0001'
```

Logical Exclusive Or

The logical exclusive or operator .XOR. performs a 16-bit bit-wise "exclusive or" operation on its two operands.

Examples:

```
X'41'.XOR.X'41'       = 0
B'0011'.XOR.B'0101'   = B'0110'
X'FFF'.XOR.X'F0F0'    = X'FF0F'
```

Logical Or

The logical or operator .OR. performs a 16-bit bit-wise "inclusive or" operation on its two operands.

Examples:

```
X'41'.OR.X'14'      = X'55'
X'0304'.OR.'00'     = '34'
B'0011'.OR.B'0101'  = B'0111'
```

Multiplication

The multiplication operator * yields the product of its two operands which are treated as unsigned 16-bit integers. If necessary, the result is truncated by dropping all but the least significant bits.

Examples:

```
5*5              = 25
X'4000'*X'10'    = 0
X'FEDC'*X'100'= X'DC00'
```

Right Shift

The right shift operator .RS. shifts its left operand to the right the number of bits indicated by the right operand. The shift is a logical 16-bit end-off shift with zero bits entering on the left as required. The shift count must be in the range 0-15 inclusive.

Examples:

```
B'101010'.RS.3 = B'101'
X'FF7F.RS.5    = X'07FB'
320.RS.2       = 160
10.RS.1        = 10
```

12.RS.16 = (error)

12.RS.X'FFFF' = (error)

Subtraction

The subtraction operator - subtracts its right operand from its left operand to yield a result. The subtraction is performed on 16-bit unsigned quantities ignoring overflow. This is equivalent to a twos-complement subtract if the operands are regarded as signed twos-complement values.

Examples:

3-4	= X'FFFF' (-1)
200-100	= 100
200-X'FFFF'	= 201 (200-(-1))

OPERATOR PRECEDENCE AND ORDER OF OPERATIONS

It is permissible to write an alternating sequence of operands and operators in an expression. In this case, the question arises of which operator is performed first, and is resolved by the following precedence table.

Precedence Level	Operators
7	* /
6	+ -
5	.RS. .LS.
4	.EQ. .NE. .GT. .LT. .LE. .GE.
3	.AND.
2	.OR.
1	.XOR.

When two operators have a common operand, the operator with the higher precedence is always performed first. If two such operators have the same precedence, the one on the left is performed first. Parentheses may be used to group sub-expressions and alter the normal order of operations as specified by above rules. For example, the following two examples are equivalent:

A+B.EQ.3.AND.4*6*7

((A+B).EQ.3).AND.((4*6)*7)

Up to five levels of parentheses are permitted.

OMISSION OF ZERO OPERANDS

If the left operand of + or - is zero:

```
0+
0-
```

the zero may be omitted and is understood to be present. Thus:

-5	is the same as 0-5 (=X'FFFB')
+A*B	is the same as 0+A*B
(-A)*(-B)	is the same as (0-A)*(0-B)
1*-2	is the same as 1*0-2

EXAMPLES

The following examples illustrate the rules for forming expressions and also show several techniques which are made possible by the expression evaluation mechanism:

A*(A.GE.B)+B*(B.GT.A)	(maximum of A, B)
10+10*($X.RS.14)	(10 if CNFG 10 or 25, else 20)
X-X/Y*Y	(remainder of X divided by Y)
(A.XOR.X'8000').GE.(B.XOR.X'8000')	(signed comparison)
NLINES*64	(screen size)
$X-4*(3+D)-2	(auto-exec area, device D)
$B.RS.4.AND.1	(isolate bit 4 of $B)
'0'+N+('A'-'('-1)*(N.GT.9)	(ASCII-hex value of N ($0 \leq N \leq 15$))

SECTION IV

MACHINE OPERATIONS

GENERAL

This section describes assembler operations used to generate code for executable machine operations. They are grouped by functional class. Note that automatic word alignment takes place for all machine instructions just as if they had been preceded by an ALGN pseudo-operation. As indicated, any instruction may be labeled, in which case the symbol used as <label> is defined to have the value of the location counter after any required word alignment.

WORD CLASS INSTRUCTIONS

	MNEMONIC OP CODE	HEXADECIMAL OP CODE	DESCRIPTION
[< label >]	CMC < a >	{ X'70aa' }	compare cursor
[< label >]	CMX < a >	{ X'70aa' }	compare index
[< label >]	DEC < a >	{ X'58aa' }	decrement
[< label >]	INC < a >	{ X'50aa' }	increment
[< label >]	IN2 < a >	{ X'60aa' }	increment by two
[< label >]	JMP < a >	{ X'B8aa' }	jump
[< label >]	JSR < a >	{ X'78aa' }	jump to subroutine
[< label >]	LDC < a >	{ X'40aa' }	load cursor
[< label >]	LDX < a >	{ X'40aa' }	load index
[< label >]	STC < a >	{ X'48aa' }	store cursor
[< label >]	STX < a >	{ X'48aa' }	store index

The listed operations generate one word (short address) instructions which reference a word location in memory. < a > is an expression whose value is the operand address which must be even. The assembler sets the sector reference and addressing bits of the instruction in accordance with the appropriate hardware rules. In the event that the referenced sector is not directly accessible, an automatic entry is made in the literal table (called a desectorizing link) which points to the desired address and the generated instruction is setup to reference indirectly through this location.

Another possibility for the operand < a > is a literal consisting of an expression immediately preceded by =. In this case, a word containing the 16 bit value of the expression is created in the literal table and the instruction references this word. See description of the SIZE pseudo-operation for details of the positioning of this table.

Any of the above instructions may specify indirect addressing by the use of an * immediately following the opcode. In this case, the indirect bit of the instruction (if the sector is directly referencable) and the desectorizing link (if one is created) is set on to obtain indirect addressing.

```
Examples:  BGLOOP    JMP      $
                     JSR      SUBR
                     INC*     $X
                     LDX      =0
                     CMC      =NLINES*64
                     JMP*     =XX (forces desectorizing)
```

BYTE CLASS INSTRUCTIONS

	MNEMONIC OP CODE		HEXADECIMAL OP CODE	DESCRIPTION
[< label >]	AD	< a >	{ X'10aa' }	add byte
[< label >]	AN	< a >	{ X'28aa' }	and byte
[< label >]	CM	< a >	{ X'20aa' }	compare byte
[< label >]	LD	< a >	{ X'00aa' }	load byte
[< label >]	OR	< a >	{ X'30aa' }	or byte
[< label >]	SB	< a >	{ X'18aa' }	subtract byte
[< label >]	ST	< a >	{ X'08aa' }	store byte

The above operations generate one-word (short address) instructions which reference a byte location in memory. < a > is an expression whose value is the operand address. The assembler sets the sector reference and addressing bits of the instruction in accordance with the appropriate hardware rules. In the event that the referenced sector is not directly accessible, an automatic entry is made in the literal table (called a desectorizing link) which points to the desired address and the generated instruction is set up to reference indirectly through this location. Except as described below, literals may not be used as operands for byte class instructions.

Any of the above instructions may specify indirect addressing by the use of a * immediately following the opcode. In this case the indirect bit of the instruction (if the sector is referencable) and the desectorizing link (if one is created) is set on to obtain desectorizing. Indirectly addressed byte

class instructions are treated as word class instructions. This means that the specified address must be even and that literal references are permitted.

```
Examples:        LD   A+4
                 ST*  $X
           LBL   CM*  =A (forces desectorizing)
```

INDEXING

On the 20/20 only (i.e. in CNFG 20), indexing notation can be used on any word class or byte class instruction. Preceding the operand field with @ (at sign) causes the assembler to generate an indexing (X'C94F') instruction prior to generating code for the instruction itself. In this case, <label>, if present, is set to the location counter value prior to generating the indexing instruction.

For word class and indirectly addressed byte class instructions, it is permissible to use indexing in conjunction with a literal reference, the @ sign precedes the = in this case.

```
Examples:  LOAD  LD   @4
                 LDX  @LINKP
                 CM*  @=NEXT (forces desectorizing)
```

OPERATE CLASS INSTRUCTIONS

	MNEMONIC OPCODE	HEXADECIMAL OPCODE	DESCRIPTION
[<label>]	NOP	{X'C000'}	no operation
[<label>]	HALT	{X'C001'}	halt
[<label>]	MLA	{X'C002'}	move LIR to ACR
[<label>]	MAL	{X'C003'}	move ACR to LIR
[<label>]	MCA	{X'C004'}	move CHR to ACR
[<label>]	MAC	{X'C005'}	move ACR to CHR
[<label>]	ENB	{X'C006'}	enable interrupts
[<label>]	DSB	{X'C007'}	disable interrupts
[<label>]	IOR	{X'C008'}	I/O reset
[<label>]	SHL4	{X'C009'}	shift left four
[<label>]	RTL1	{X'C00A'}	rotate left one
[<label>]	SRL1	{X'C00B'}	shift and rotate left one
[<label>]	CLL	{X'C00C'}	clear LIR
[<label>]	CLC	{X'C00D'}	clear CHR
[<label>]	WAIT	{X'C00F'}	wait

The above operations generate one-word operate class instructions as shown. No operands are permitted. MLA, MAL, MCA, MAC, CLL, CLC may be used only in CNFG 10 or CNFG 24 or CNFG 25 modes. RTL1 may be used only in CNFG 20 mode. SRL1 may be used only in CNFG 24 mode.

IMMEDIATE CLASS INSTRUCTIONS

	MNEMONIC OP CODE	HEXADECIMAL OPCODE	DESCRIPTION
[<label>]	LDI <v>	{X'80vv'}	load immediate
[<label>]	ADI <v>	{X'90vv'}	add immediate
[<label>]	SBI <v>	{X'98vv'}	subtract immediate
[<label>]	ANI <v>	{X'A8vv'}	and immediate
[<label>]	ORI <v>	{X'B0vv'}	or immediate
[<label>]	XOR <v>	{X'D0vv'}	exclusive or immediate

The above operations generate one-word immediate class instructions. The operand< v >is an expression whose value is the immediate operand. This value must be in the range 0 to 255 or -128 to -1 (X'FF80' to X'FFFF'). Indirect addressing is not permitted.

Examples:

```
ADI     '0'
ANI     X'FF'
XOR     $.AND.X'FF'
```

JUMP ON CONDITION INSTRUCTIONS

	MNEMONIC OP CODE		HEXADECIMAL OP CODE	DESCRIPTION
[<label>]	JCFAL	<a>	{X'8800aaaa'}	never jump
[<label>]	JCLT	<a>	{X'8900aaaa'}	jump on less than
[<label>]	JCEQ	<a>	{X'8A00aaaa'}	jump on equal
[<label>]	JCLE	<a>	{X'8B00aaaa'}	jump on less than or equal
[<label>]	JCTRU	<a>	{X'8C00aaaa'}	always jump
[<label>]	JCGE	<a>	{X'8D00aaaa'}	jump on greater than or equal
[<label>]	JCNE	<a>	{X'8E00aaaa'}	jump on not equal
[<label>]	JCGT	<a>	{X'8F00aaaa'}	jump on greater than
[<label>]	JCNG	<a>	{X'8801aaaa'}	jump on negative
[<label>]	JCPO	<a>	{X'8C01aaaa'}	jump on positive
[<label>]	JCEV	<a>	{X'8C02aaaa'}	jump on even
[<label>]	JCOD	<a>	{X'8802aaaa'}	jump on odd
[<label>]	JCNC	<a>	{X'8900aaaa'}	jump on no carry
[<label>]	JCCO	<a>	{X'8D00aaaa'}	jump on carry
[<label>]	WJMP	<a>	{X'8C00aaaa'}	word jump

The above operations generate two-word jump on condition instructions as shown. The operand <a> is an expression whose value, which must be even, is the target jump address.

Indirect addressing may be specified by placing a * after the opcode. In this case the indirect addressing bit of the second word generated is set on.

Examples:

```
                JCGT    LOC+8
                JCEQ*   RTNAD
HANG            JCTRU   HANG
                JCNE    $-10
```

COMPARE AND JUMP INSTRUCTIONS

	MNEMONIC OP CODE		HEXADECIMAL OPCODE	DESCRIPTION
[< label >]	CJFAL	< v >,< a >	{X'A0vvaaaa'}	compare and never jump
[< label >]	CJLT	< v >,< a >	{X'A1vvaaaa'}	compare and jump on less than
[< label >]	CJEQ	< v >,< a >	{X'A2vvaaaa'}	compare and jump on equal
[< label >]	CJLE	< v >,< a >	{X'A3vvaaaa'}	compare and jump on less than or equal
[< label >]	CJTRU	< v >,< a >	{X'A4vvaaaa'}	compare and always jump
[< label >]	CJGE	< v >,< a >	{X'A5vvaaaa'}	compare and jump on greater than or equal
[< label >]	CJNE	< v >,< a >	{X'A6vvaaaa'}	compare and jump on not equal
[< label >]	CJGT	< v >,< a >	{X'A7vvaaaa'}	compare and jump on greater than

The above operations generate two-word compare and jump class instructions. The first field< v >is an expression whose value is the comparison operand. The value of this expression must be in the range 0 to 255 or -128 to -1 (X'FF80' - X'FFFF'). The second field< a >is an expression whose value,

which must be even, is the target jump address.

Indirect addressing may be specified by placing a * after the opcode. In this case, the most significant bit of the second word generated is set on.

```
Examples:   LBL     CJEQ    X'F0',$+8
                    CJNE    $.AND.X'FF',MATCH
                    CJGE*   'C',ENTRY
                    CJEQ    -1,BOR
```

INPUT-OUTPUT INSTRUCTIONS

	MNEMONIC OP CODE		HEXADECIMAL OP CODE	DESCRIPTION
[<label>]	CIO	<f>,<d>	{ X'C9fd' }	control I/O
[<label>]	RIO	<f>,<d>	{ X'CAfd' }	read I/O
[<label>]	WIO	<f>,<d>	{ X'CBfd' }	write I/O

The above operations generate the one word input-output instructions. <f>,<d> are expressions whose value is in the range 0 to 15 which are used to provide the function and device codes as shown.

	MNEMONIC OP CODE		HEXADECIMAL OP CODE	DESCRIPTION
[<label>]	JFACK	< f>, <d>, <a>	{X'CCfdaaaa'}	jump on no I/O acknowledge
[<label>]	JTACK	< f>, <d>, <a>	{X'C8fdaaaa'}	jump on I/O acknowledge

The above operations generate the two word test input-output instructions. < f>, <d> are two expressions whose value is in the range 0 to 15 which are used to provide the function and device codes as shown. < a> is an expression whose value is the target jump address which must be even. Indirect addressing may be specified by placing a * immediately following the opcode, in which case the most significant bit of the second word is set on.

```
Examples:        CIO     4, X'A'
                 RIO     SDATA, SYNC
LOOP             JFACK*  1, 2, WRITE
                 JTACK   BUSY, DISK, $
                 WIO     1, $B. AND. X'F'
```

EXTENDED MNEMONICS

The assembler provides a set of opcodes representing special purpose uses of standard instructions. Although these are not individual machine operations, they may be regarded as such by the assembly language programmer.

	MNEMONIC OP CODE	HEXADECIMAL OPCODE	DESCRIPTION
[< label>]	CLA	{X'8000'}	clear accumulator

CLA takes no operands, and as can be seen from the generated code, is actually equivalent to LDI 0.

	MNEMONIC OPCODE	HEXADECIMAL OPCODE	DESCRIPTION
[<label>]	SKP	{X'8800'}	skip next instruction

SKP takes no operands, and as can be seen from the generated code, is equivalent to the first word of a JCFAL instruction, the skipped word actually being the unreferenced address operand.

SECTION V

ASSEMBLER PSEUDO-OPERATIONS

GENERAL

Assembler pseudo-operations are written in normal LABEL OPCODE OPERANDS COMMENT format but do not correspond to actual hardware instructions. Instead, they are used to generate data in various ways or to control the assembly process. This chapter lists all pseudo-operations in alphabetical order.

ADDR -- Generate two-byte data

The ADDR pseudo-operation is used to assemble one or more two-byte values. These values are generated in sequence and are not necessarily aligned to a word boundary.

The form is:

[<label>] ADDR <operand>,<operand>,...,<operand>

<label>, if present, is set to the value of the location counter prior to generation of the first value. No word alignment takes place.

As shown, the number of <operand>s is variable from one to as many as will fit on a single line.

Any expression may be used as an <operand>, resulting in generation of two bytes containing its value.

Alternately, an <operand> can have the following format:

<repeat><constant>

where < repeat > is an explicit decimal integer and <constant> is one of the following:

B'....'	binary constant
O'....'	octal constant
D'....'	decimal constant
X'....'	hexadecimal constant
'x'	one character ASCII constant
'xx'	two character ASCII constant

The result of such a specification is to generate two bytes containing the constant value the number of times specified by repeat. A zero repeat factor results in no code being generated.

Examples:

```
        ADDR    'AB',3D'999'        (generates 4 words)
KONS    ADDR    B'101',99'**'       (generates 100 words)
```

ALGN -- Align to word boundary

The ALGN pseudo-operation increments the load and location counter if the location counter is odd. It thus ensures that the following data is generated starting on an even byte (i.e. word) boundary. In the event that the location counter is already even, ALGN has no effect. The form is:

```
[<label>]       ALGN
```

As shown, there is no operand. <label>, if given, is assigned the location counter value following incrementation (if any) of the counters i.e. the label value will always be even. Note that the incrementation in the odd case is equivalent to BSS 1; no zero byte is generated.

Examples:

```
WORDST      ALGN
```

BOOT -- Set boot mode

The BOOT pseudo-operation sets the assembler in boot mode. The form is: BOOT

As shown, no operand or label is permitted.

Boot must precede the first object code generation. It is used to assemble a program which is intended to load directly using the hardware bootstrap mechanism. The following restrictions are placed on a boot mode program:

ORG operations are not allowed. BOOT sets the origin to zero (X'000') automatically. The normal prohibition against code below location X'100' is not enforced.

XORG may not specify a second operand.

BSS is not allowed.

Literals may not be used.

Cross-sector references not resolvable by the hardware are not permitted.

SEG and ESEG operations are not permitted.

END may not specify an entry point.

BSS -- Reserve storage

The BSS pseudo-operation is used to reserve storage without assembling any data. The form is:

[<label>] BSS <count>

<label>, if present, is set to the location counter value at the start. No word alignment takes place.

<count> is an expression whose value is the number of bytes to reserve. This value is added to both the location and load counters. Any symbols used must have been previously defined.

Note: INCOTERM standard loaders do not clear memory before loading. This means that the contents of any BSS areas following initial load is whatever was there before the load started, since these areas are not included in the object output.

Examples:	BUFR	BSS	128	
	HERE	BSS	0	(no effect, except to define label)
	ALGNR	BSS	$.AND.1	(like ALGN)

BSZ -- Generate zeros

The BSZ pseudo-operation is used to generate a specified number of zeros. The form is:

[<label>] BSZ <count>

<label>, if present, is set to the location counter value at the start. No word alignment takes place.

<count> is an expression giving the number of zero bytes to be generated. A value of zero is permitted, resulting in generation of no data. Any symbols appearing in the expression must be previously defined.

Note that the use of BSZ with large counts can result in increasing the size of the generated object program considerably depending on the medium. Where possible, it is desirable to replace such BSZ operations with initialization code which performs the required zeroing operation.

Examples:	BUFFER	BSZ	128	(generates 128 zeros)
		BSZ	1256-$	(zeros up to location 1256 decimal)
		BSZ	$.AND.1	(acts like ALGN, except that a byte of zeros may be generated)
		BSZ	64-($.AND.63)	(zero to end of 10/20 screen line)

BYTE -- Generate byte data

The BYTE pseudo-operation is used to assemble one or more bytes of data according to the value of expressions given. The form is as follows:

[<label>] BYTE <operand>, <operand>, ... <operand>

<label>, if present, is set to the value of the location counter prior to generation of the first value. No word alignment takes place.

As shown, the number of <operand>s is variable from one to as many as will fit on a single line.

Any expression whose value is in the range 0 to 255 or -128 to -1 (X'FF80' to X'FFFF') may be used as an <operand> with the resulting generation of a single byte of data (upper X'FF' byte is ignored for "negative" values).

Alternately, an <operand> can have the following format:

<repeat><constant>

where <repeat> is an explicit decimal integer and <constant> is one of the following:

B'....'	binary constant
O'....'	octal constant
D'....'	decimal constant
X'....'	hexadecimal constant
'x'	single character ASCII constant

The result of such a specification is to generate a byte containing the constant value the number of times specified by repeat. A zero repeat value is legal and causes no data to be generated for that value.

```
Examples:  LOOP    BYTE    X'FF'
                   BYTE    A+B*C,-1
           L1      BYTE    'E','R',3'*'              (generates 5 bytes)
                   BYTE    $.AND.X'FF',79B'1001'     (generates 80 bytes)
                   BYTE    'E',0'*','R'              (generates 2 bytes)
```

CNFG -- Specify configuration mode

The CNFG pseudo-operation sets the configuration mode to match the hardware in use. The form is:

```
CNFG    [<mode>]
```

As shown, no label is permitted.

<mode> is an expression which has one of the following five values:

10 10/20 mode assembly. AXN, RTL1, SRL1 instructions not recognized. Addressing restrictions and link resolution compatible with SPD 10/20 architecture.

20 20/20 mode assembly. MCA, MAC, MAL, MLA, CLC, CLL, SRL1 operations not recognized. Addressing restrictions and link resolution compatible with SPD 20/20 architecture.

24 10/24 mode assembly. AXN, RTL1 operations not recognized. Addressing restrictions and link resolution compatible with SPD 10/24 architecture.

25 10/25 mode assembly. AXN, RTL1, SRL1 operations not recognized. Addressing restrictions and link resolution compatible with SPD 10/25 architecture.

0 Compatible mode assembly. AXN, RTL1, MCA, MAC, MLA, CLC, CLL, MAL, SRL1 operations not recognized. Addressing restrictions and link resolution compatible with SPD 10/20, 10/24, 10/25 and 20/20 architectures.

Any symbols appearing in the expression for mode must have been previously defined. See section on addressing for further details on address restrictions and link resolution.

The first CNFG pseudo-operation in the program specifies the program configuration mode and must occur before the first generation of object code. Subsequent CNFG operations may occur but they affect only the allowed operation set, not the addressing mode. This may be useful, for example, in assembling a section of code in a CNFG 0 assembly which is executed only in the 20/20 case and uses RTL1.

If the <mode> operand is omitted, the effect is to return to the mode of the program (i.e. that of the first CNFG). The mode operand may not be omitted on the initial CNFG line.

If no CNFG operation appears in a program, CNFG 10 is assumed throughout.

Examples:	CNFG 10	(set 10/20 mode)
	CNFG 0	(set compatible mode)
	CNFG 10+10*($B.AND.1)	(10 or 20 depending on $B)

DAC -- Generate address constant

The DAC pseudo-operation is used to generate a single word containing an address. The form is:

[<label>] DAC[*] <address>

<label>, if present, is set to the location counter value prior to generation of the word (following possible word alignment).

<address> is an expression which evaluates to an address in the permissible range (see section on addressing). If indirect addressing is specified, the indirect addressing bit of the result is set. (DAC is the only pseudo-operation which permits indirect addressing).

```
Examples:  KPTR    DAC     K
                   DAC     $+3*4
                   DAC*    256
```

DUP -- Duplicate source line

The DUP pseudo-operation allows the effects of a specified source line to be duplicated. The form is:

```
    DUP       < count>
<source line>
```

The effect is as if <source line> had appeared consecutively the number of times specified by the value of the count expression. Any line may be duplicated in this manner except DUP, END, IF, ENDF, sub-title and comment lines.

Any symbols appearing in <count> must be previously defined. As shown, no label is permitted.

A duplication count of zero is valid, and causes <source line> to be assembled zero times (i.e. to be deleted).

Within expressions appearing in <source line>, the special symbol $D takes on values of 1, 2, 3,... for the successive duplications.

```
Examples:  DUP     X.NE.0                  (assemble if X non-zero)
           DAC     X

           DUP     26                      (26 ASCII letter codes)
           BYTE    'A'-1+$D

           DUP     64-($.AND.63)           (blanks to end of 10/20 line)
           BYTE    ' '

           DUP     16                      (16 ASCII codes for hex digits)
           BYTE    '0'-1+$D+7*($D.GE.10)
```

EJECT -- Eject assembly listing to new page

The EJECT pseudo-operation causes the assembly listing to eject to a fresh page. The form is:

```
EJECT
```

As shown, no label or operand is permitted. The use of EJECT has no effect on the generated code. EJECT is effective only in LIST 2 or LIST 3 mode.

END -- End assembly

The END pseudo-operation occurs once at the end of the source text as the last source line and marks the end of the text. The form is:

```
END        [<entry>]
```

<entry> is an expression whose value (which must be in address range) specifies the location to which control is to be passed following loading the program. As shown, it is optional. If omitted, the entry point is set to the first assembled location in segment zero. As shown, no label is permitted.

Examples:

```
END
END        BEGIN
```

ENDF -- End conditional assembly range

The ENDF pseudo-operation marks the end of a conditional assembly range and occurs only in conjunction with a previously occuring matching IF pseudo-operation. The form is:

```
ENDF
```

As shown, no label or operand is permitted.

EQU -- Equate symbol to value

The EQU pseudo-operation is used to assign a given value to a specified symbol. The form is:

<label> EQU <value>

<label>, which is required, is the symbol whose value is to be set.

<value> is an expression giving the value to be set. Any symbols appearing in <value> must have been previously defined.

Examples:	SYMQ	EQU	X'27'	(value of ASCII quote)
	HERE	EQU	$	(like HERE BSS 0)
	PRA	EQU	$X-2	(location of power restart)
	F2	EQU	F1+1	(F1 must be previously defined)

ESEG -- End overlay segment

The ESEG pseudo-operation terminates an overlay segment. The form is:

ESEG [< entry >]

<entry> is an expression whose value specifies the segment entry point. If omitted, the segment entry point is set to the first assembled location in the segment. The significance of this entry point is defined by the routines used to load the segment. It is normally the location to which control is passed initially if the segment is executed. As shown, no label is permitted.

Following the ESEG, the load and location counters are unchanged and assembly resumes in segment zero (the main segment). All literals and links required by the code in overlay segments are added to the main (segment zero) literal table which is always resident.

Examples: ESEG SEGMENT

ESEG

HEX -- Generate hexadecimal data

The HEX pseudo-operation is used to generate a string of bytes whose value is given in hexadecimal. The form is:

[<label>] HEX <hex>

<label>, if present, is set to the location counter value prior to generation of the first data byte. No word alignment takes place.

<hex> is an even length string of two or more hexadecimal digits (0-9, A-F). Each successive pair of digits corresponds to a single byte of generated data.

```
Examples:   ENDREF   HEX      7474
                     HEX      A3FF010D
                     HEX      4142          (like TEXT 'AB')
```

IF -- Start conditional assembly range

The IF pseudo-operation provides for conditional assembly of a section of code. The form is:

```
        IF    <test>
```

<test> is an expression which is evaluated. If the value is non-zero, there is no effect. If the value is zero, than all code between the IF and its matching ENDF is ignored. Any symbols appearing in<test> must have been previously defined. As shown, no label is permitted. IF-ENDF pairs may be nested to any reasonable level.

```
Examples:   IF    $B.AND.X'200'          (test bit 9 of $B)
            IF    X.GT.Y
            IF    X.GE.3.AND.X.LT.17
```

LBL -- Generate label data

The LBL pseudo-operation is used to generate the primary title data. The form is:

```
        [<label>]   LBL
```

<label>, if given, is set to the location counter value prior to the generation of the first byte. No word alignment takes place. As shown, no operand is permitted.

The effect of LBL is to assemble 40 bytes corresponding to the ASCII codes for the 40 characters of the primary title, as modified by any assembled & lines in the text. The result is exactly as if this title had been assembled by TEXT. This operation may be useful in controlling program version number information.

Examples: LABELD LBL

LIF -- Control Listing of IF Ranges

The LIF pseudo-operation is used to control the listing of IF's, ENDF's and code deleted in conditional ranges. The form is:

LIF <code>

As shown, no label is permitted.

<code> is an expression with a value of 0, 1, or 2. Any symbols used must have been predefined. The effects are as follows:

<code> = 0 IF and ENDF lines are never printed and code deleted by "false" ranges is also not printed. The effect is to generate an assembly listing which (except for line numbers) is identical to that which would have been obtained if no IF's had been used.

<code> = 1 (The default setting if no LIF appears). Code deleted by "false" ranges is not listed. IF and ENDF lines list except when they are contained within a deleted range.

<code> = 2 All IF and ENDF lines are printed. Deleted code is also printed.

In LIST 0 mode, IF and ENDF lines and deleted code are not printed regardless of the LIF setting. Note that the assembler has an option to control LIF mode externally, thus overriding the affect of LIF operations in the source.

LIST -- Control Listing Mode

The LIST pseudo-operation is used to control the format of the listing. The form is:

```
        LIST        <code>
```

As shown, no label is permitted.

<code> is an expression with a value of 0, 1, 2, or 3. Any symbols used must have been predefined. The effects are as follows:

<code> = 0 Only flagged lines are printed, no sub-titles are printed.

<code> = 1 All lines are printed except comments starting with #. EJECT and subtitle lines are printed as comments but have no other effect. No sub-titles are printed.

<code> = 2 All lines are printed except comments starting with #. EJECT and subtitle lines are interpreted and used to format the listing.

<code> = 3 All lines are printed including comments starting with #. EJECT and sub-title lines are interpreted and used to format the listing.

In the case of IF, ENDF and lines deleted by IF-ENDF, both the LIF and LIST conditions must be met in order for unflagged lines to print. Note that the assembler has options to control LIST mode externally, thus overriding the effect of LIST operations in the source.

LTXT -- Generate lower case text string

The LTXT pseudo-operation is used to generate a string of bytes corresponding to ASCII values of characters in a supplied text string except that any letters are converted to lower case (other characters unaffected). The form is:

[<label>] LTXT <d> <text><d>

<label>, if present, is set to the location counter prior to generation of the first byte. No word alignment occurs.

<d> is any non-blank character not appearing in <text>. It acts as a delimiter and is not included in the generated data.

<text> is a string of any characters except the delimiter character. It may range in length from zero (no data generated) to as long as is permitted by the restriction to a single source line.

Examples:

```
        LTXT  '0123'          (like TEXT '0123')
XX      LTXT  *AB*            (generates X'6162')
```

LTX8 -- Generate lower case text string with 8th bit set.

The LTX8 pseudo-operation is used to generate a string of bytes corresponding to ASCII values of characters in a supplied text string except that letters are converted to lower case as in LTXT and the most significant (parity) bit of each byte is set. The form is:

[<label>] LTX8 <d><text><d>

<label>, if present, is set to the location counter prior to generation of the first byte. No word alignment occurs.

<d> is any non-blank character not appearing in <text>. It acts as a delimiter and is not included in the generated data.

<text> is a string of any characters except the delimiter character. It may range in length from zero (no data generated) to as long as is permitted by the restriction to a single source line.

```
Examples:       LTX8  '0123'          (like TXT8 '0123')
          LB    LTX8  *AB*            (generates X'E1E2')
```

NOBJ -- Turn off object output mode

The NOBJ pseudo-operation is used to discontinue generation of object code. The form is:

```
        NOBJ
```

As shown, no label or operand is permitted.

Following the occurence of NOBJ and until an OBJ pseudo-operation is encountered no object data will be generated. However, in all other respects the assembly proceeds as usual, with the load and location counters being incremented in the normal manner.

NOBJ may be useful in assembling prototype (dummy) sections of code for purposes of documentation or symbol definition. OBJ mode is automatically

set when an END or ESEG line is encountered, thus literals and entry point records are always generated.

OBJ -- Turn on object output mode

The OBJ pseudo-operation is used to resume generation of object code. The form is:

```
OBJ
```

As shown, no label or operand is permitted.

Following the occurence of OBJ, normal output of generated code is resumed. Since this is the normal (default) mode, OBJ is used only in conjunction with NOBJ to selectively inhibit object code generation for parts of the program.

ORG -- Set assembly origin

The ORG pseudo-operation is used to set or reset the location and load counters. The form is:

[<label>] ORG <loc>

<label>, if present, is set to the value of the location counter after the origin operation is completed (i.e. to the value of loc).

<loc> is an expression whose value is used to reset both the load and location counters. All symbols appearing in the expression must be previously defined.

Note: In the absence of an ORG statement at the start of an assembly, the default starting value for both load and location counters is X'100'.

Examples:		ORG	$+10	(like BSS 10)
	TOPS	ORG	X'7E00'	
		ORG	$X.AND.X'7E00'	(CNFG independent top sector)
		ORG	$L	(resets location counter to match load counter)

PAGE -- Set page depth

The PAGE pseudo-operation allows specification of the page depth for the assembly listing. The form is:

PAGE <count>

As shown, no label is permitted.

<count> is an expression whose value is the maximum number of lines per printed assembly page (not counting the title lines). When this value is exceeded, an automatic eject occurs. The allowed range is 1-255. Any symbols used must have been previously defined.

More than one PAGE pseudo-operation may occur in assembly, in which case each one affects the listing for the following section. If no PAGE pseudo-operation is used, an appropriate default which depends on the assembler and environment is set.

The use of PAGE does not affect the generated code.

Examples:

```
PAGE    41      (41 lines/page)
PAGE    255     (max allowed value)
```

SEG -- Start of overlay segment

The SEG pseudo-operation specifies the start of an overlay segment. The form is:

<label> SEG [<org>]

<label>, which is required, is set to the segment number. Segments are numbered serially starting at 1 and ranging up to a maximum value of 250.

<org> is an expression whose value is used to set the load and location counters in the same manner as the ORG pseudo-operation. It is optional and, if omitted, the load and location counters are unchanged. Any symbols appearing in <org> must have been previously defined.

The range of segment code is from the SEG operation to its matching ESEG. No nesting of SEG-ESEG pairs is permitted. ORG and XORG pseudo-operations may be used within the segment to modify the location and load counters.

Standard INCOTERM loaders load only segment zero and ignore other segments. The loading of overlay segments is accomplished using special routines appropriate to the load medium.

Examples:
```
SEG1    SEG
SEG2    SEG     SEGORG
```

SET -- Set symbol to value

The SET pseudo-operation is used to assign or reassign the value of a specified symbol. The form is:

<label> SET <value>

<label>, which is required, is the symbol whose value is to be set.

<value> is an expression giving the value to be set. Any symbols appearing in <value> must have been previously defined.

SET is the only operation which permits a symbol to be assigned more than one value. This is accomplished by having more than one SET for the same symbol. In this case the value of the symbol is whatever was assigned by the most recent SET operation. If a SET symbol is referenced before any SET for the SYMBOL has occured, the value (in pass 2) is that set by the last SET in pass 1.

A symbol defined by SET must not be defined by any other method (i.e. must not appear in the label field except on another SET operation).

```
Examples:   Q       SET   Q+1          (increment previously SET value)
            SEGHI   SET   SEGHI+($-SEGHI)*($.GT.SEGHI) (SEGHI=max(SEGHI, $))
```

SIZE -- Specify memory size

The SIZE pseudo-operation is used to specify the memory size and the origin for literals. The form is:

SIZE <size>[,<links>[, <xsize>]]

As shown, no label is permitted.

<size> gives the address of the last byte in memory and must have one of the following values:

CNFG 0		X'0FFF'	(4K)
	or	X'1FFF'	(8K)
	or	X'3FFF'	(16K)
	or	X'7FFF'	(32K)
CNFG 10		X'0FFF'	(4K)
CNFG 20		X'3FFF'	(16K)
	or	X'7FFF'	(32K)
CNFG 24		X'1FFF'	(8K)
	or	X'3FFF'	(16K)
CNFG 25		X'0FFF'	(4K)

In the cases where more than one operand can be given, the significance is that the value given represents the minimum workable memory size for the program.

In addition to the values shown, X'0000' (zero) may be used. See special section "SIZE 0" in Chapter VI for details.

<links> gives the origin for the literal table. Literals and desectorizing links are built downward from the specified location and extend down as far as

necessary. Thus the specified even (word) location is the lowest unused address above the table (i.e. the lowest reserved location). A high memory address suitable to the program CNFG must be specified. As shown, the links operand is optional. If omitted, the following default values are used:

CNFG 0	X'7FBA'
CNFG 10	X'FD2'
CNFG 20	X'7FBA'
CNFG 24	X'7FD2'
CNFG 25	X'FD2'

<xsize> is omitted in normal assemblies. If set it allows the load counter (but not the location counter) to exceed the normal limit and range up to xsize without error. The resulting program will not be loadable by standard INCOTERM loaders, but special purpose loaders may load such data into external memory for use in overlay systems. The low order byte of the value must be X'FF'.

The SIZE line may be omitted entirely, in which case the size defaults to X'FFF' (CNFG 0, CNFG 10, CNFG 25), X'3FFF' (CNFG 20), or X'1FFF' (CNFG 24). If present, it must precede the occurence of any generated object code.

Examples:	SIZE	X'3FFF', X'7FBA'	(CNFG 20 default)
	SIZE	X'FFF', X'FEO'	(CNFG 10, links higher than default)
	SIZE	X'7FFF', X'7FBA', X'FFFF'	(32K external memory)

TEXT -- Generate text string

The TEXT pseudo-operation is used to generate a string of bytes corresponding to ASCII values of characters in a supplied text string. The form is:

[<label>] TEXT <d>< text> <d>

<label>, if present, is set to the location counter prior to generation of the first byte. No word alignment occurs.

< d > is any non-blank character not appearing in <text>. It acts as a delimiter and is not included in the generated data.

< text > is a string of any characters except the delimiter character. It may range in length from zero (no data generated) to as long as is permitted by the restriction to a single source line.

Examples:
```
        TEXT    'THIS IS A MESSAGE'
MSG     TEXT    /QUOTE --'/
        TEXT    *AB*                    (generates X'4142')
```

TXT8 -- Generate text string with 8th bit

The TXT8 pseudo-operation is used to generate a string of bytes corresponding to ASCII values of the characters in a supplied text string with the X'80' (parity) bit set. The form is:

[<label>] TXT8 <d><text><d>

<label>, if present, is set to the location counter prior to generation of the first byte. No word alignment occurs.

<d> is any non-blank character not appearing in <text>. It acts as a delimiter and is not included in the generated data.

<text> is a string of any characters except the delimiter character. It may range in length from zero (no data generated) to as long as is permitted by the restriction to a single source line.

Examples: BMS TXT8 'BLINK MESSAGE'

TXT8 *AB* (generates X'C1C2')

WORD -- Generate word data

The WORD pseudo-operation is used to assemble one or more word values. These values are generated in sequence and are word aligned. The form is:

[<label>] WORD <operand>,<operand>,...<operand>

<label>, if present, is set to the value of the location counter prior to generation of the first value. Normal word alignment takes place.

As shown, the number of <operand>s is variable from one to as many as will fit on a single line. Any expression may be used as an <operand>, resulting in generation of a word containing its value.

Alternately, an <operand> can have the following format:

<repeat><constant>

where <repeat> is an explicit non-zero decimal integer and <constant> is one of the following:

B'....'	binary constant
O'....'	octal constant
D'....'	decimal constant
X'....'	hexadecimal constant
'x'	one character ASCII constant
'xx'	two character ASCII constant

The result of such a specification is to generate words containing the constant value the number of times specified by repeat.

Note: WORD is identical to ADDR except that a prior word alignment occurs.

```
Examples:               WORD    'AB', 3D'999'     (generates 4 words)
        KONS            WORD    B'101', 99'**'
```

XORG -- Set execution origin

The XORG pseudo-operation allows separate specification of the load and location counters for use where code is to be assembled and loaded in one location and moved to another location before it is executed. The form is:

[<label>] XORG <location>[,< load>]

<location> is an expression whose value is the new value of the location counter. Any symbols used must have been previously defined.

<load> is an expression whose value is the new value of the load counter. Any symbols used must have been previously defined. As shown, this operand is optional. If omitted, the load counter value ($L) is unchanged.

<label>, if present, is set to the new value of the location counter (i.e. to the value of the first operand).

Note that when XORG is used, it is the responsibility of the executing program to move the code into its proper location before actually executing the code.

Examples:		XORG	$, $L	(has no effect)
		XORG	X, X	(like ORG X)
	LDR	XORG	0	(typical start of code for loader)
		XORG	$L	(resynchronize load, location counters)

XREF -- Set cross reference mode

The XREF pseudo-operation is used to control collection of cross references. The form is:

```
XREF        < code>
```

As shown, no label is permitted.

<code> is an expression with a value of 0, 1, or 2. Any symbols used must have been previously defined. The effects are as follows:

<code> = 0	Collection of symbol cross reference is completely inhibited.
<code> = 1	Symbol cross references will be collected in assembled areas.
<code> = 2	Symbol cross references will be collected in both assembled and unassembled (IF-ENDF deleted) areas.

Note that the assembler has options to control XREF mode externally, thus overriding the effect of XREF operations in the source.

SECTION VI

ADDRESSING RESTRICTIONS

GENERAL

Although the load and location counters are full 16-bit quantities, only a limited range of values are permissible depending on the TPU model and memory size. These limitations are described separately for each CNFG value.

CNFG 10 ADDRESSING

The SPD 10/20 and SPD 10/25 always have 4K bytes of memory and the hardware program counter is only 12 bits. Consequently, the assembler limits the range of address values to be between X'0000' and X'0FFF'. Top sector is at location X'0E00'.

CNFG 20 ADDRESSING

The SPD 20/20 has 16K, or 32K of memory. To allow smaller programs to run unchanged on machines with different sized memories, it is desirable to have a uniform addressing structure. This is achieved by consistently addressing the special top sectors with addresses of the form X'7x00'. Thus we have.

TOP always at X'7E00'
TOP-1 always at X'7C00'
TOP-2 always at X'7A00'

This works on smaller memories because unused high order address bits are ignored. Thus an address X'7C54' is treated as X'3C54' on a 16K machine.

To implement the restrictions implied by a SIZE less than 32K, the assembler limits the allowed addresses to the following ranges:

SIZE (1st parameter)		Allowed Ranges
X'3FFF'	(16K)	X'0000' - X'39FF' X'7A00' - X'7FFF'
X'7FFF'	(32K)	X'0000' - X'7FFF'

In the case of 16K, there is a "gap" in the allowed address range. It is the programmers responsibility to avoid assembling code in this gap by suitable use of ORG commands.

CNFG 24 ADDRESSING

The SPD 10/24 has 8K or 16K of memory. As in the case of the SPD 20/20, a uniform addressing scheme is used as follows:

Top always at X'7E00'
Top-1 always at X'7C00'

The following are the allowed addressing ranges:

SIZE (1st parameter)		Allowed Ranges
X'1FFF'	(8K)	X'0000' - X'1BFF' X'7C00' - X'7FFF'
X'3FFF'	(16K)	X'0000' - X'3BFF' X'7C00' - X'7FFF'

CNFG 25 ADDRESSING

The rules for CNFG 25 addressing are exactly the same as for the SPD 10/20.

CNFG 0 ADDRESSING

CNFG 0 specifies a program which is to run compatibly on more than one SPD model. Top sector is always addressed at X'7E00'. Since the 10/20 and 10/25 ignore the upper four bits and the 10/24 ignores the upper two bits, this always produces the intended effect. The allowed address ranges depend on the size as follows:

SIZE (1st parameter)		Allowed Ranges
X'0FFF'	(4K)	X'0000' - X'0DFF' X'7E00' - X'7FFF'
X'1FFF'	(8K)	X'0000' - X'1DFF' X'7E00' - X'7FFF'
X'3FFF'	(16K)	X'0000' - X'3DFF' X'7E00' - X'7FFF'
X'7FFF'	(32K)	X'0000' - X'7FFF'

As usual, the significance of the SIZE parameter is the minimum required memory. Note that this addressing convention precludes the possibility of direct references from TOP sector to TOP-1. Thus in a CNFG 0 assembly, desectorizing links are created for such references.

Following these addressing rules results in a program whose addressing structure is compatible. To execute compatibly, the code must be model insensitive. In particular, the following points should be observed:

(1) INC, DEC, IN2, STX, LDX, CMX are 12 bit operations on the 10/20 and 10/25, 14 bit operations on the 10/24, and 16 bit operations on the 20/20.

(2) Instructions specific to one model (e.g. MCA, AXN) can only be used in sections of code which are executed only on the appropriate model.

(3) Byte references to locations X'7FFE', X'7FFF' may have different effects.

(4) The RTL1 instruction (X'C00A') acts as a NOP on the 10/2x and may be used to distinguish between machines at run time.

SIZE 0

Regardless of the CNFG setting, the first parameter to SIZE can be given as zero. In this case, all address checks are omitted and the resulting program can be loaded into any sized memory. If there is segment zero code in a memory location not available, the address will be "folded" by ignoring high order bits. It is the responsibility of the executing program in this case to test the memory size or otherwise ensure that sensible processing occurs.

EFFECT OF RESTRICTIONS

The following address operands must be within the above specified ranges:

Byte class instruction operands,
Word class instruction operands,
Jump instruction target addresses,
DAC pseudo-operation operands.

Except when external memory is present as described below, it is invalid to assemble code or data with a load counter value outside the specified ranges. Note that the operand values of ORG and XORG are not restricted in any way, only the actual assembly of code and address reference values.

EXTERNAL MEMORY

The specifications of external memory (SIZE third parameter) allows the load counter value for generated code to range up to the given value. However, address operands restrictions are unaffected. Thus the XORG pseudo-operation must be used to assemble code in external memory.

LOADER CONSIDERATIONS

INCOTERM standard loaders use locations X'0000' - X'00FF' for the loader program itself. Thus any non-BOOT mode program must avoid assembling code or data with a load counter value in this region and the assembler enforces this restriction in segment zero.

Certain INCOTERM loaders use the AUTO-EXEC areas and index/cursor register. Programs using such loaders should avoid assembling code into these locations. In the case where a program may be loaded from more than one device, the safe rule is to avoid the AUTO-EXEC area completely, although the assembler does not enforce this restriction. Note, however, that the power restart location is not used by any loader and may be set at assembly time.

LITERAL POOL

The literal pool containing referenced literals and created links is built down from the address specified as the second parameter to SIZE (or the default value as described under the SIZE pseudo-operation).

In no case will the assembler generate literals or links which are not addressable from every location. This means that all entries must be in the TOP sector except on the SPD 20/20 (i.e. in a CNFG 20 assembly) where TOP-1 and TOP-2 may also be used.

It is considered an error to assemble code which overlaps this generated literal pool and this check will be performed by the assembler. Note, however, that it is perfectly admissible to assemble data above the literal pool.

BSS OPERATIONS

Even though BSS does not actually generate any object output, it is treated exactly like BSZ from the point of view of the above address checks. Thus BSS with a non-zero operand may not appear in the loader area, or in a restricted area, or overlap the literal pool. In NOBJ mode, the load counter value checks are omitted, thus NOBJ can be used to avoid error flags on BSS operations (or any other generating lines) if such overlap is required.

SECTION VII

FORMAT OF PRINTED LISTING

GENERAL

This chapter describes the general format of the printed assembly listing.

TITLE LINES

The first line of every page contains the main title line as modified by any assembled & lines in the program. It also contains the page number. & lines themselves are listed with line numbers in the same format as comment lines.

The second line is blank. The third line contains the current subtitle as set by the most recently listed subtitle line in the source. The subtitle line itself is printed in comment format at the top of the page (i.e. it always causes an EJECT so that the new subtitle is printed immediately). Subtitle lines which are not listed (in LIST 0 mode or in unassembled areas with LIF 0 or LIF 1 modes set) do not affect the listed subtitle.

Line four following the subtitle is blank. In LIST 1 mode, subtitle lines are printed as comments and lines three and four are omitted.

The remaining lines are source lines and their generated code as described below. The maximum number of such lines is set by the current page depth (see PAGE pseudo-operation). If this maximum is exceeded, an automatic page eject occurs.

GENERATED MACHINE INSTRUCTIONS

The form of the line printed for a machine instruction is as follows:

```
ffffss-dddd//1111p  cccccccc s(nnnnn) xxxxx
```

ffff	Up to 4 error flags (see separate section). Blank if no errors detected.
ss	Current segment number (two hexadecimal digits). Omitted, together with following minus, if no SEG statements appear in the program.
dddd	Load counter at start of line, following any required word boundary alignment. Omitted, together with the following slashes, if no XORG statements appear in the program.
1111	Location counter at start of line, following any required word boundary alignment.
p	Colon in OBJ mode. Asterisk in NOBJ mode.
cccccccc	Generated code, four or eight hexadecimal digits. If only two bytes generated, then the last four columns of this field are blank. If indexing is specified, all four bytes are printed on a single line.
s	Blank if no desectorizing link generated. Asterisk if desectorizing link generated.
nnnnn	Line number, 1-5 decimal digits with leading zeros as required.
xxxxx	Reproduction of source program line.

COMMENT LINES

Comment lines print simply as:

```
(nnnnn) xxxxx
```

Comment lines starting with # are printed only in LIST 3 mode.

PSEUDO-OPERATIONS

Data generation pseudo-operations list exactly like machine instruction lines except that cccccccc may contain from 0-4 bytes (0-8 hexadecimal digits), the case of zero occuring for zero repeat counts, null operand fields etc. Also, if more than four bytes are generated from a single source line, additional lines are printed in the format:

ffffss-dddd//1111p cccccccc

where cccccccc contains 2-8 hexadecimal digits. The pseudo-operations listing in the above format are ADDR, BYTE, DAC, HEX, LBL, LTXT, LTX8, TEXT, TXT8, WORD.

In the following descriptions, "location" means:

ffffss-dddd//1111p

as described for machine instructions, page 7-2.

ALGN	Prints new location after any required alignment.
BOOT	Prints in comment line format.
BSS	Prints initial location.
BSZ	Prints like data generation pseudo-operation except that only the first line prints when the operand value exceeds 4.
CNFG	Prints in comment format.
DUP	Prints the duplication count in the first four columns of the cccccccc field as four hexadecimal digits. The duplicated line prints in normal format except that if the duplication results in the target line generating more than four bytes of output, additional short lines are printed as for data generation pseudo-operations.

EJECT	In LIST modes 2 or 3, EJECT does not print, but causes an eject to the top of the next page. Blank pages cannot occur through use of multiple EJECT lines or EJECT lines and a subtitle line. In LIST 1 mode, EJECT prints in comment format. In LIST 0 mode, EJECT has no effect. In unassembled IF-ENDF areas with LIF modes 0 or 1, EJECT has no effect. In LIF 2 mode, the behavior depends on the LIST mode as described above.
END	The program entry point address prints in the first four columns of the cccccccc field as four hexadecimal digits.
ENDF	Prints in comment format.
EQU	The operand value prints in the first four columns of the cccccccc field as four hexadecimal digits.
ESEG	The segment entry point address prints in the first four columns of the cccccccc field as four hexadecimal digits.
IF	Prints in comment format.
LIF	Prints in comment format.
LIST	Prints in comment format.
NLIF	Prints in comment format.
NLST	Prints in comment format.
NOBJ	Prints in comment format.
NXRF	Prints in comment format.
OBJ	Prints in comment format.
ORG	Prints updated location.
PAGE	Prints in comment format.
SIZE	The first operand value is printed in the first four columns of the cccccccc field as four hexadecimal digits.

XORG Prints updated location.

XREF Prints in comment format.

Note: The above rules mean that, whenever a label is defined, its value is the current value of the location counter as printed in the 1111 field of the line.

UPDATE/DELETION FLAGS

To warn the reader of possible discrepancies between the listing and the source program or generated object program, the characters // are placed on columns 5, 6 of the cccccccc field in the following lines.

(1) All IF lines printed.

(2) All ENDF lines printed.

(3) Source line unassembled in range of DUP 0.

(4) Source lines unassembled in range of IF-ENDF if listed in LIF 2 mode.

(5) LIST lines except LIST 0 lines in LIST 0 mode.

LITERAL TABLE

After the end line, the literal table is printed providing that the assembly does not end in LIST 0 mode. This table prints one line for each unique literal in the format:

```
f  00-dddd//1111: cccc
```

The only possible flag is X, which is generated on the highest address literal which is overlapped by previously assembled code. Although the literal table is built backwards, it is listed and generated forwards, i.e. by successively higher addresses.

In LIST 0 mode, the literal table is not printed except that the single line flagged X is printed if literal overlap has occured.

CROSS-REFERENCE LISTING

Following the assembly listing, a cross-reference listing is generated containing a sorted list of defined symbols together with a list of line numbers in which the symbol is referenced. The SPD/DOS assembler also generates a literal cross-reference table showing line numbers which reference each literal or generated desectorizing link.

ERROR FLAGS

Error flags are letters placed at the left side of erroneous lines. This section gives the meaning of the various flags. It should be noted that the various assemblers may post flags in a slightly different manner. Furthermore, a limited number of flags can be placed on each line, so that errors beyond this number (usually four) remain unflagged.

Flag	Meaning
A	Violation of addressing restriction (see Chapter VI). The assembler ignores the violation and proceeds with the code generation.
B	Violation of BOOT mode restrictions (see description of BOOT pseudo-operation in Chapter V). The attempt is ignored. Also posted for a BOOT pseudo-operation which occurs after generated code. BSS in BOOT mode is flagged and treated as BSZ.
C	Erroneous character (e.g. missing quote in character constant, missing terminator in text constant, invalid numeric digit).
E	An operand which must be even (i.e. a word address) is odd.

Flag	Meaning
F	Forward reference. An operand requiring all symbols to be predefined references a symbol which is defined later on. A value of zero is used at the point of reference.
H	Missing END line. The assembler supplies one if possible.
I	IF-ENDF nesting is incorrect. ENDF with no previous IF or IF with no matching ENDF.
L	Label error. Invalid label syntax or label where none permitted or missing required label.
M	Multiple definition of a label. The second definition is ignored. Some assemblers also post this on all references to multiply defined labels.
N	Numeric error (e.g. operand outside permitted range, division by zero, multiplication of negative quantities), a value of zero is used.
O	Unrecognized operation code. Two words of zeros are generated. Also caused by a wrong CNFG setting, e.g. RTL1 in CNFG 10 in which case the correct code is generated.
Q	Invalid SIZE or CNFG parameter is ignored.
S	SEG-ESEG sequence error (i.e. missing ESEG for previous SEG or ESEG with no previous SEG) or more than 250 segments.
T	Table overflow. Symbol or literal table overflow, or unaddressable literal.
U	Reference to undefined symbol. A value of zero is used.
V	Too few operands or missing operand. Zero value used.
W	Too many operands. Extra operands are ignored.
X	Attempt to assemble segment zero code in loader region (X'0000' - X'00FF') or overlap of code and literals, or attempt to assemble code in a region forbidden by the CNFG and SIZE parameters in effect.

Flag	Meaning
%	Symbol not defined due to symbol table overflow, may also be posted on references to such symbols.
@	Invalid use of indexing: incorrect format, or used where indexing not allowed, or used in other than CNFG 20 mode.
	Invalid use of literal: incorrect format, or used where literal not allowed.
'	Invalid use of indirect addressing: used on an opcode not permitting indirect addressing.

APPENDIX A

ASSEMBLER SYNTAX, QUICK REFERENCE

EXPRESSION OPERANDS

ddd..	Decimal constant
D'ddd..'	Decimal constant
0'ddd..'	Octal constant
B'ddd..'	Binary constant
X'ddd..'	Hexadecimal constant
'c'	ASCII character constant
'cc'	ASCII word constant
ll..	Label name
$	Location counter
$L	Load counter
$C, $X	Cursor/index register location
$B	$B setting
$D	Duplication count
**	Zero

EXPRESSION OPERATORS (in order of increasing precedence)

.XOR.	Exclusive Or
.OR.	Inclusive Or
.AND.	And
.NE. .GT. .LT. .GE. .EQ. .LE.	Comparisons
.RS. .LS.	Shifts
+ -	Addition, Subtraction
* /	Multiplication, Division

TITLE LINES

&text	Primary Title Modification	text is in columns 2-41
'text	Subtitle Line	

OPERATION FORMATS

[1]	AD	b	[1]	CJGT	i, e
[1]	AD*	w	[1]	CJLE	i, e
[1]	ADDR	xr, xr, xr, ... xr	[1]	CJLT	i, e
[1]	ADI	i	[1]	CJNE	i, e
[1]	ALGN		[1]	CJTRU	i, e
[1]	AN	b	[1]	CLA	
[1]	AN*	w	[1]	CLC	
[1]	ANI	i	[1]	CLL	
[1]	BOOT		[1]	CM	b
[1]	BSS	p	[1]	CM*	w
[1]	BSZ	p	[1]	CMC	w
[1]	BYTE	br, br, ... br	[1]	CMC*	w
[1]	CIO	f, f	[1]	CMX	w
[1]	CJEQ	i, e	[1]	CMX*	w
[1]	CJEQ*	i, e		CNFG	p (0, 10, 20, 24 or 25)
[1]	CJFAL	i, e	[1]	DAC	a
[1]	CJFAL*	i, e	[1]	DAC*	a
[1]	CJGE	i, e	[1]	DEC	w
[1]	CJGE*	i, e	[1]	DEC*	w
			[1]	DSB	
				DUP	p

	EJECT	
[1]	ENB	
	END	[e]
	ENDF	
1	EQU	p
	ESEG	[e]
[1]	HALT	
	HEX	h
	IF	p
[1]	IN2	w
[1]	IN2*	w
[1]	INC	w
[1]	INC*	w
[1]	IOR	
[1]	JCCO	e
[1]	JCCO*	e
[1]	JCEQ	e
[1]	JCEV	e
[1]	JCFAL	e
[1]	JCGE	e
[1]	JCGT	e
[1]	JCLE	e
[1]	JCLT	e
[1]	JCNC	e
[1]	JCNE	e
[1]	JCNG	e
[1]	JCOD	e
[1]	JCPO	e
[1]	JCTRU	e
[1]	JFACK	f, f, e
[1]	JFACK*	f, f, e
[1]	JMP	w
[1]	JMP*	w
[1]	JSR	w
[1]	JSR*	w
[1]	JTACK	f, f, e
[1]	JTACK*	f, f, e
[1]	LBL	
[1]	LD	b
[1]	LD*	w
[1]	LDC	w
[1]	LDC*	w
[1]	LDX	w
[1]	LDX*	w
[1]	LDI	i
	LIF	p (0-2)
	LIST	p (0-3)
[1]	LTXT	t
[1]	LTX8	t
[1]	MAC	

[1]	MAL	
[1]	MCA	
[1]	MLA	
	NOBJ	
[1]	NOP	
	OBJ	
[1]	OR	b
[12	OR*	w
[1]	ORG	p
[1]	ORI	i
	PAGE	p (1-255)
[1]	RIO	f,f
[1]	RTL1	
[1]	SB	b
[1]	SB*	w
[1]	SBI	i
1	SEG	p
1	SET	p
[1]	SHL4	
	SIZE	a [,a[,a]]
[1]	SKP	
[1]	ST	b
[1]	ST*	w
[1]	STC	w
[1]	STC*	w
[1]	STX	w
[1]	STX*	w
[1]	TEXT	t
[1]	TXT8	t
[1]	WAIT	
[1]	WIO	f,f
[1]	WJMP	e
[1]	WJMP*	e
[1]	WORD	xr,xr,...xr
[1]	XOR	i
[1]	XORG	p[,p]
	XREF	p (0-2)

KEY TO SYMBOLS

b Expression whose value is valid byte address, indexing (preceding @) permitted in CNFG 20.

e Expression whose value is valid even address.

f Expression with value in range 0-15.

h Even length string of hexadecimal digits.

i Expression with byte value (0 to 255 or -128 to -1).

l Label

p Expression whose value is previously defined.

r Repeat factors allowed.

t Text string enclosed in delimiters.

w Expression whose value is even address or literal (= followed by any expression), indexing (preceding @) permitted in CNFG 20.

x Any expression.

APPENDIX B

ERROR FLAGS, QUICK REFERENCE

A	Invalid address
B	Boot mode violation
C	Erroneous character.
E	Odd operand where even required.
F	Forward reference
H	Missing END line.
I	IF-ENDF nesting error.
L	Label error.
M	Multiple definition.
N	Numeric error.
O	Invalid operation code.
Q	Invalid SIZE or CNFG parameter.
S	SEG-ESEG sequence error.
T	Table overflow.
U	Undefined symbol.
V	Missing operand.
W	Extra operand.
X	Invalid load location.
Z	Symbol not defined, table overflow.
@	Indexing error.
=	Invalid literal.
*	Invalid use of indirect addressing.

APPENDIX C

COMPATIBILITY WITH PREVIOUS VERSIONS

The following constructions, though not considered part of the SPD assembly language, are permitted by all current versions of the absolute assembler for compatibility with previous versions of the language. Always use the new forms where possible.

The least significant byte of the first parameter to SIZE may be X'FE' instead of X'FF'.

LIST with no operand is equivalent to LIST 3.

NLST is equivalent to LIST 0.

LIF with no operand is equivalent to LIF 2.

NLIF is equivalent to LIF 0.

XREF with no operand is equivalent to XREF 1.

NXRF is equivalent to XREF 0.

AXN, which may be labeled, generates an indexing instruction X'C94F'. AXN can be used only in CNFG 20 mode.

A SIZE first parameter of X'1FFF' (8K) may be specified in CNFG 20.

APPENDIX D

SPD/DOS ASSEMBLE OPERATING NOTES

SPD/DOS ASSEMBLE is an assembler which runs under control of SPD/DOS (diskette operating system) on any SPD TPU equipped with a diskette unit and a printer. See the SPD/DOS Operator's Reference Manual (Order Number MS-7177) for general details on loading and operating SPD/DOS.

Source Input Format

Source programs to be processed by SPD/DOS ASSEMBLE are stored in DOS source files. The maximum permitted record length is 80 characters (longer records are truncated). The 64 character upper case ASCII set is accepted, the standard ASCII representation in the source file being assumed. Lower case letters and special characters may be used in comments if the printer in use will accept them, but lower case letters are not acceptable in symbolic names.

If the source file data is entered from the keyboard (for example, by using the DOS EDIT utility), the layout of the relevant ASCII characters depends on the keyboard as shown in Figures D-1 through D-4. Note that the maximum record length for files created by EDIT will be 64 characters.

If the source file data is entered from the card reader (for example, by using the DOS COPY utility) the punch code is standard ASCII. Table D-1 shows these punches together with corresponding EBCDIC graphics for use when an 029 (or similar) model EBCDIC keypunch is used to prepare card input.

If the source file is entered from paper tape (for example, by using the DOS COPY utility), the standard 8 channel ASCII code is used. The parity bit is ignored and may be even, odd, always off or always on. For formats on other external media, see the SPD/DOS Programmer's Reference Manual (Order Number MS-7178).

Figure D-1. Keyboard Layout for Preparation of Input to SPD/DOS ASSEMBLE (10/20 Upper Case Only Keyboard)

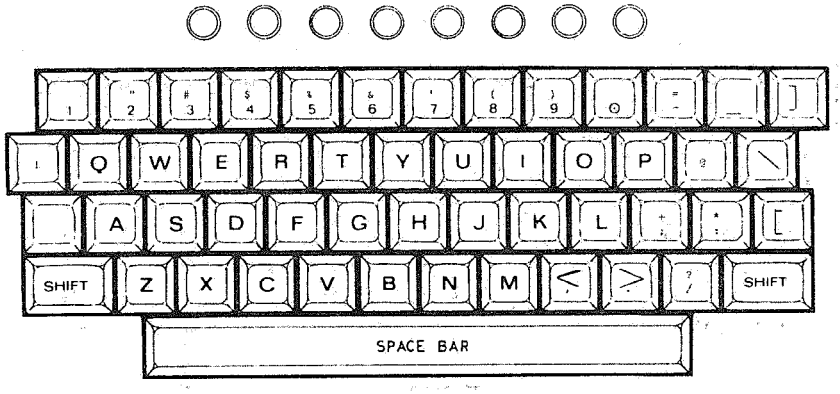


Figure D-2. Keyboard Layout for Preparation of Input to SPD/DOS ASSEMBLE (10/20 Upper/Lower Case Keyboard)

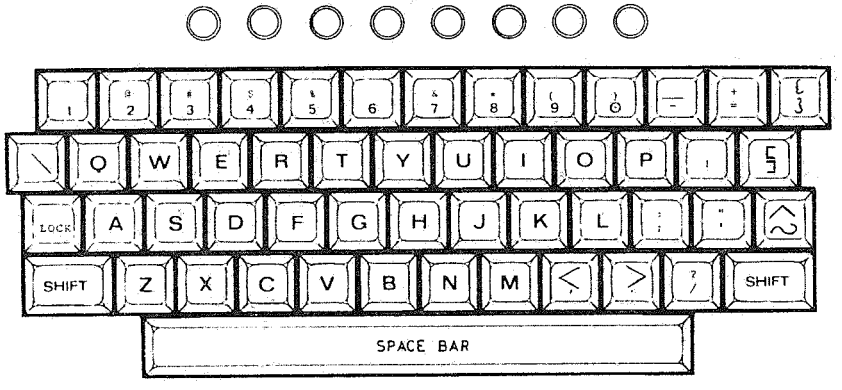


Figure D-3. Keyboard Layout for Preparation of Input to SPD/DOS ASSEMBLE (20/20 Upper Case Only Keyboard)

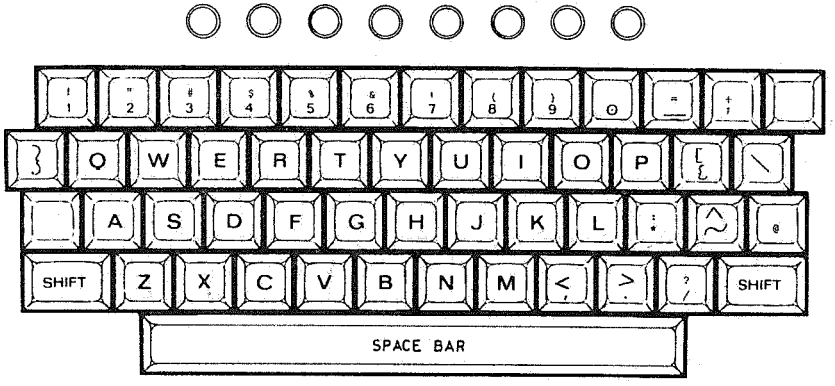


Figure D-4. Keyboard Layout for Preparation of Input to SPD/DOS ASSEMBLE (20/20 Upper/Lower Case Keyboard)

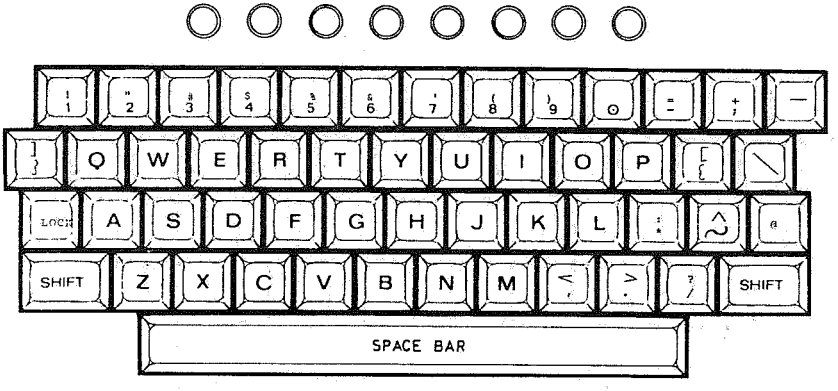


Table D-1

Card Punch Codes for Input to SPD/DOS ASSEMBLE

ASCII Graphic	Card Punch Code	Corresponding EBCDIC Graphic
space	no punch	space
!	12-8-7	|
"	8-7	"
#	8-3	#
$	11-8-3	$
%	0-8-4	%
&	12	&
'	8-5	'
(	12-8-5	(
)	11-8-5	)
*	11-8-4	*
+	12-8-6	+
,	0-8-3	,
-	11	-
.	12-8-3	.
/	0-1	/
0	0	0
1	1	1
2	2	2
3	3	3

ASCII Graphic	Card Punch Code	Corresponding EBCDIC Graphic
4	4	4
5	5	5
6	6	6
7	7	7
8	8	8
9	9	9
:	8-2	:
;	11-8-6	;
<	12-8-4	<
=	8-6	=
>	0-8-6	>
?	0-8-7	?
@	8-4	@
A	12-1	A
B	12-2	B
C	12-3	C
D	12-4	D
E	12-5	E
F	12-6	F
G	12-7	G
H	12-8	H
I	12-9	I
J	11-1	J

ASCII Graphic	Card Punch Code	Corresponding EBCDIC Graphic
K	11-2	K
L	11-3	L
M	11-4	M
N	11-5	N
O	11-6	O
P	11-7	P
Q	11-8	Q
R	11-9	R
S	0-2	S
T	0-3	T
U	0-4	U
V	0-5	V
W	0-6	W
X	0-7	X
Y	0-8	Y
Z	0-9	Z
[	12-8-2	¢
\	0-8-2	none (0-8-2)
]	11-8-2	!
^	11-8-7	¬
-	0-8-5	-

Object Output Format

The output from SPD/DOS ASSEMBLE is an object file on diskette in standard SPD/DOS object file format. See DOS Programmer's Reference Manual (Order Number MS-7178) for full details.

Operating Procedures

ASSEMBLE is loaded with a DOS command of the form:

AS[,<options>] <sfile>,<b>,<ofile>,<olabel>

<sfile> specifies the source program. If the program is in a single file <sfile> is a normal format DOS filename. If the source program is contained in more than one file, <sfile> is given in the format:

<fname>.<dsn1>.<dsn2>[.<dsn3>]

<fname> specifies the name of all the files (which must be the same) and the unit of the initial file. <dsn1>,<dsn2>(and <dsn3> if three files are used) specify in sequence the serial numbers of the disks containing the files. As shown, the parts of this parameter are separated by periods.

<b> is 4 hexadecimal digits (0-9, A-F) specifying the value of the assembly parameter $B.

<ofile> is the name of the output object file to be created.

<olabel> is the label of the object file to be created.

If the fourth parameter is omitted, the label on the object file is copied from the label of the source input file.

If the third and fourth parameters are omitted, the name and label of the object file are the same as those of the source input file.

If all but the first parameter are omitted, then $B is set to X'0000' and the name and label of the object file are the same as those of the source input file.

If, during the assembly, a disk must be mounted, a pause will occur for mounting. The disk containing the work file, object file and assemble load file must never be reloaded.

ASSEMBLE requires a D (data) type work file to be established prior to initialization of the assembly. The file is called WORK and may be conveniently created using the CREATE utility. The minimum size is 6 tracks. The number of symbols and cross references which can be handled is a function of the size of this work file as shown in Table D-2.

Table D-2

ASSEMBLE Work File Capacity

Tracks	Maximum Symbols	Maximum Cross References
6	100	2048
9	350	3072
12	600	4096
15	850	5120
18	1100	6144
21	1350	7168
24	1600	8192
27	1850	9216
30	2100	10240
33	2350	11264
36	2600	12288
39	2850	13312
42	3100	14336
45	3350	15360
48	3600	16384

Option Letters

The <options> field consists of one or more of the following letters in any order.

A Alternate unit. The object file is generated on the opposite unit from that implied by the call.

B Both pass list. A listing is given during pass 1 as well as pass 2. Usually used only for diagnosing system errors.

C Clean list. LIF 0 mode is enforced throughout the assembly regardless of the occurence of LIF operations in the source program.

E Erase. Any previous object file of the same name is erased. In the absence of this option, it is an error to have such a duplicate file name.

F Full list. LIF 2 mode is enforced throughout the assembly regardless of the occurence of LIF operations in the source.

G Generate included code.
LIN 1 mode - . -

H Hold generated code.
LIN 0 mode . . - '

I Inhibit object. The assembly proceeds normally, but output of the object file is inhibited.

K Kill hash comments. LIST 2 mode is enforced throughout the assembly regardless of the occurence of LIST operations in the source program.

L List mode. LIST 3 mode is enforced throughout the assembly regardless of LIST operations in the source program.

N No printer. "Listing" output is written to diskette using a pre-existing source file called LIST. The CREATE utility with the S option may be used to create this file. EDIT may be used to examine the file following assembly.

P Paper save. LIST 1 mode is enforced throughout the assembly regardless of the occurence of LIST operations in the source program.

Q Quick assembly. XREF 0 mode is enforced throughout the assembly and no cross reference output is generated.

R Reference unassembled. XREF 2 mode is enforced throughout the assembly regardless of the occurence of XREF operations in the source program.

S Short list. LIST 0 mode is enforced throughout the assembly regardless of the occurence of LIST operations in the source program.

T Table of contents. A table of contents showing the initial page number for each listed sub-title line is printed immediately before the listing of the first sub-title line in the program. This allows header comments to precede the table of contents.

U Unlist deleted code. LIF 1 mode is enforced throughout the assembly regardless of the occurence of LIF operations in the source program.

V Verify. Object file output is verified using a reread check. Other disk write operations (to the WORK and LIST) files are never verified.

X Xref. XREF 1 mode is enforced throughout the assembly regardless of the occurence of XREF operations in the source program.

File Allocation

The assembler will work no matter how the files are positioned, but the following rules should be follwed for maximum efficiency.

The WORK file and ASSEMBLE program load file should be on one unit and the source on the other unit in a two disk system. ASSEMBLE will preferentially select the WORK file on the unit opposite to the source if WORK files exist on both units.

The WORK file should immediately follow the ASSEMBLE program load file. This is of particular importance from an efficiency point of view on machines with more than 8K memory (SPD 10/24, 20/20).

The output object file may be on either unit with little impact on efficiency.

The LIST file (N option set) may be on either unit. ASSEMBLE preferentially selects the LIST file on the unit opposite to the source if LIST files exist on both units. In the case where a full listing is obtained on disk, an approximate guide is to make the LIST file one and a half times as large as the source file itself.

If disks must be reloaded during assembly (more than one source file), then the WORK file, LIST file, ASSEMBLE program load file and generated object file must all be on one unit, the unit which is not reloaded.

Definition of Standard Symbols

Standard DOS symbols, as defined in the DOS Programmers Reference Manual, Order Number MS-7178, may be referenced in an assembly without being defined.

These definitions are accessed only for otherwise undefined symbols, thus the program is free to use names of standard symbols for its own purposes.

Standard symbols do not count towards the limits shown in Table D-2.

Display Messages

A one line display is active throughout the assembly. The first forty characters contain the version number identification for ASSEMBLE. The remaining twenty-four characters are used to display various messages as follows.

INITIALIZATION	Displayed during the assembly initialization process.
PASS: 1 REC: nnnnn ERR: mmm	Displayed throughout pass one. nnnnn is the number of the current record. mmm is the number of lines with errors detected. Note that not all errors are detected in pass one.
PASS: 2 REC: nnnnn ERR: mmm	Displayed throughout pass two. nnnnn is the record number, mmm is the number of lines with errors.
CROSS REFERENCE ERR: mmm	Displayed during cross reference table output. mmm is the total number of lines with errors.
TERMINATION ERR: mmm	Displayed during assembler termination processing. mmm is the number of lines with errors.
MOUNT UNIT=X DSN=nnnnnnnn	Displayed during pause for disk remount in the case where the source program is split over more than one diskette. The required diskette should be mounted on the indicated unit and then the space bar pressed.

Examples

AS, EX 1, FRED

This command causes the source file FRED on unit 1 to be assembled generating an object file on unit 1 also called FRED with the same label as

the source file. The value of $B is set to X'0000'. XREF 1 mode will be enforced and any previously existing object file called FRED on unit 1 will be erased before generating the new object file output.

ASSEMBLE, L XY, 1000, 1. YZ, VERSION 1

This command assembles the source file XY on the currently selected unit generating an object file with name YZ on unit 1 with label VERSION 1. The value of $B is set to X'1000' and LIST 3 mode is enforced throughout the assembly regardless of the occurence of LIST statements in the assembly source.

AS XX. DISK1. DISK2. DISK3, 177A

The source program is made up of three files all called XX on disks DISK1, DISK2, and DISK3. $B will be set to X'177A'.

AS, INS PROG1

The source file PROG1 on the currently selected unit is assembled with no object file output. An errors-only listing will be written to the LIST file which must exist before starting the assembly.

Listing Format

The format of the assembly listing itself is exactly as described in Chapter VII. The main title is taken from the file label of the generated object file as implied by the command line used to load ASSEMBLE. The default page depth is taken from the current DOS configuration parameter (allowing four lines for the title and sub-title).

The cross reference listing is in two parts. The symbol cross reference listing contains an entry for each referenced symbol. In this listing, the line numbers for definitions occur first using <> to surround definition numbers (a definition corresponds to the appearance of the symbol in the label field of an assembled instruction). References follow using () to surround line numbers. A "U" flag is posted on reference lines for symbols with no definition and "M" flag on reference lines for symbols which were multiply defined.

The literal cross reference contains an entry for each unique generated literal or desectorizing link. The entry gives the literal value, literal address, and numbers of referencing lines. This listing is unaffected by the use of XREF operations and is given in its entirety unless the Q option is set, in which case it is deleted.

In the case where the source program is split over more than one file, the listed line numbers start at 10001 for the second file and 20001 for

the third file. This aids in editing the component files since the low order four digits give the line number within the file. To avoid confusion in the cross reference listing, the number of lines in each component file should not exceed 10000.

Following the cross reference listing, a termination summary is printed containing the following messages:

ASSEMBLED BY xxxxxx	xxxxxx is the version of ASSEMBLE used to obtain this assembly.
SOURCE FILE DSN-dddddddd NAME-nnnnnnnn LABEL-llllllll	This message shows the disk serial number, file name and file label of the source input file. It is printed more than once if the program is composed of more than one source file.
OBJECT FILE DSN-dddddddd	This message shows the diskette serial number, name and label of the generated object file. It is omitted if there is no object file (I option set).
SECTORS OF OBJECT CODE nn	This message gives the number (nn=decimal integar) of sectors of object code. It is omitted if there is no object file (I option set)
NUMBER OF OVERLAY SEGMENTS nn	This message gives the number of overlay segments (SEG-ESEG regions) in the program (nn=decimal integer). nn is 0 for unsegmented programs.
$B SETTING X'xxxx'	This message shows the value (xxxx=four hexadecimal digits) of the assembly parameter $B. It is printed only if $B is non-zero.

CNFG-cc SIZE-X'ssss' EXTERNAL SIZE-X'xxxx'	This message shows the program configuration value (cc=10, 20, 24 25 or 00), the program size (SIZE first parameter) and the external size (SIZE third parameter). The EXTERNAL SIZE part of this message is omitted if no third parameter is given to SIZE.
BOOT MODE	This message is printed only if the program was in BOOT mode, i.e. it contained a BOOT pseudo-operation.
NO ERRORS	This message is printed if no syntax errors were detected in the source program.
ERRORS IN LINES (nnn) (nnn) ...	This message lists the numbers of lines in which syntax errors were detected.
CROSS REFERENCE TABLE INCOMPLETE	This message is printed if the cross reference table is incomplete due to work file overflow.
NUMBER OF LINES WITH ERRORS nnn	This message shows the number of lines in error. It is printed instead of the normal error cross reference if the cross reference table was incomplete.

Format of Listing on Diskette

If the N option is set, the format of the listing is the same as on a printer except as follows:

The maximum allowed LIST mode is 1; LIST modes 2, 3 are treated as LIST 1.

Titles and sub-titles are never generated.

The Q option is enforced. No cross reference listing can be obtained and the termination message gives the number of error lines, not a cross reference listing.

Lines with errors are preceded with a period, thus allowing the use of the EDIT search to locate error lines.

Full images are generated so that the file may be printed although the lines will be truncated to 64 characters when the file is examined with EDIT.

APPENDIX E

LIST OF OTHER USEFUL PUBLICATIONS

TITLE	ORDER NUMBER
PUBLICATIONS	
PUBLICATIONS CATALOG - SECOND EDITION	MS-7159
SPD/DOS MANUALS	
SPD D-250 DISKETTE REFERENCE MANUAL	MS-7143
SPD/DOS DISKETTE OPERATING SYSTEM OPERATORS REF. MAN.	MS-7177
SPD/DOS DISKETTE OPERATING SYSTEM PROGRAMMERS REF.	MS-7178
ASSEMBLER MANUAL	
SPD SYMBOLIC ASSEMBLY LANGUAGE REFERENCE MANUAL	MS-7215
SPD 10/20 MANUALS	
SPD 10/20 INTELLIGENT TERMINAL SYSTEM DESCRIPTION	MS-7145
SPD 10/20 PROGRAMMERS REFERENCE MANUAL	MS-7110
SPD 10/25 MANUALS	
SPD 10/25 INTELLIGENT TERMINAL SYSTEM DESCRIPTION	MS-7199
SPD 10/25 MULTI STATION DISPLAY SYSTEM DESCRIPTION	MS-7190
SPD 10/25 INTELLIGENT TERMINAL SYSTEM PROGRAMMERS REF.	MS-7217
SPD 20/20 MANUALS	
SPD 20/20 OPERATORS MANUAL	MS-7190
SPD 20/20 MULTI STATION DISPLAY PROGRAMMERS REFERENCE	MS-7144
SPD 20/20 MULTI STATION DISPLAY SYSTEM DESCRIPTION	MS-7165
SPD 20/20 MULTI-STATION DISPLAY SYSTEM OPERATORS MAN.	MS-7190
SPD 320/325	
SPD 320/325 VIDEO TERMINAL SYSTEM DESCRIPTION	MS-7158
SPD 320 VIDEO TERMINAL SYSTEM IBM 3270 COMP. "PLUS" B.	MS-7172
COMMUNICATIONS MANUALS	
COMMUNICATION CONTROLLER REFERENCE MANUAL	MS-7152
INCOTERM DATA COMMUNICATIONS MANUAL	CS-015
CONTROLLERS	
REMOTE LOAD CONTROLLER REFERENCE MANUAL	MS-7121
CYCLIC CHECK CONTROLLER REFERENCE MANUAL (with adden.)	MS-7122
COMMUNICATION CONTROLLER REFERENCE MANUAL	MS-7152
CYCLIC CHECK	
CYCLIC CHECK CONTROLLER REFERENCE MANUAL (with adden)	MS-7122
DATA ENTRY	
INCOFORM SOURCE DATA ENTRY SYSTEM DESCRIPTION MANUAL	MS-7205
INCOFORM SOURCE DATA ENTRY SYSTEM DESCRIPTION BROCHURE	MS-
INCOFORM SOURCE DATA ENTRY SYSTEM OPERATORS MANUAL	MS-7208
INCOFORM FORMS GENERATION OPERATORS MANUAL	MS-7209
PERIPHERAL EQUIPMENT MANUALS	
PRINTER	
SPD P-100 PRINTER REFERENCE MANUAL	MS-7123
SPD P-165/SPD P-165A PRINTERS REFERENCE MANUAL	MS-7146
Punch	
SPD PRP-45/200 PRINTING READER PUNCH OPERATORS MANUAL	MS-7154
SPD PRP-45/200 PRINTING READER PUNCH PRODUCT BULLETIN	MS-7204
TAPE	
MAGNETIC TAPE UNITS REFERENCE MANUAL	MS-7153
SPD-MT TAPE UNITS PRODUCT BULLETIN	MS-7162
SPD-T TAPE CASSETTE PRODUCT BULLETIN	MS-7201

RELOCATABLE ASSEMBLY SYSTEM

FOR THE SPD COMPUTERS

Revised May 2, 1975

Further revised June 1, 1975
(to correspond to V6.05)

Further revised June 17, 1975
(to correspond to RASSEMBL V6.07
and later versions)

Dr. Dewar

June 24, 1975

TABLE OF CONTENTS

INTRODUCTION

This manual describes a system for relocatable assembly for the SPD series of computers. It is based on the existing assembly language as described in the SPD Assembly Language Reference Manual, Order No. MS-7215.0, and a familiarity with this manual is assumed.

Basically, this system permits sections of programs or individual subroutines to be preassembled and stored in relocatable libraries for later inclusion in a final absolute assembly.

The systems as described here is implemented in release 6 of SPD/DOS.

1. RELOCATABLE ASSEMBLIES

A relocatable assembly translates an appropriate source file into a single relocatable file containing one or more modules which can subsequently be individually included in an absolute assembly by means of an IN pseudo-operation.

The form of the source file is:

```
<global definitions>
<name-1> MOD
<source-code for name-1 module>
<name-2> MOD
<source-code for name-2 module>
        .
        .
<name-n> MOD
<source-code for name-n module>
        END
```

< name-1 >, < name-2 >,... < name-n > are the names of the modules to be assembled. There must be at least one and no more than 250 modules in one assembly. The names obey the normal rules for assembler names.

The <global defintions> section, which is optional, contains EQU and SET statements which define symbols which can be referenced in any of the modules. Since no forward references can occur and since the symbols $ and $L cannot be referenced at this stage, the values defined are all absolute quantities known at assembly time. No data generating instructions or BSS statements may be included in the < global defintions > section.

The END statement in a relocatable assembly may not have a label or an operand and is only used to indicate the end of the last module.

FORM OF MODULE TEXT

The module text is essentially similar to a program written in the standard SPD assembly language with the following exceptions:

SIZE may not be used. Size and address checks as well as literal allocations, are performed in the final absolute assembly.

ORG, XORG may not be used. A relocatable module consists of a sequential, contiguous section of code (except as described below under TOP).

SEG, ESEG may not be used. Segmentation, if any, is definied in the final absolute assembly.

BOOT may not be used.

CNFG may be used, but acts differently. See following section.

OBJ, NOBJ may not be used.

Use of CNFG

CNFG may be used but acts only to determine the allowed set of instruction mnemonics and the use of indexing. The first CNFG in a module must come before any code is generated and acts to determine the CNFG setting of the module. A CNFG 0 module may be included anywhere in the final absolute assembly. A specifically CNFG'ed module may be included only in a region of the absolute assembly with matching CNFG.

Subsequent CNFG lines change the set of allowed mnemonics but do not affect the CNFG setting of the module. The default of CNFG 0 is reset at the start of each module, so each module must contain at least one CNFG statement if this default is unsuitable.

Relocatable Addresses

The location counter printed represents a relocatable value and always starts at X'000'. When a module is INcluded in an absolute assembly, these relocatable values are added to the initial load and location counter values to obtain resulting absolute addresses.

Absolute References

Absolute references to top sectors should be done on the basis of top sector starting at X'7E00'. The symbols $X and $C have the value X'7FFE' regardless of the CNFG.

TOP Sector Section

It is optionally allowed to end the module with a TOP sector section consisting of a TOP pseudo-operation (no label or operand) followed by BSS or code generating operations. The data assembled following a TOP pseudo-operation will be allocated in the literal pool in the final assembly in a contiguous manner. The storage locations assigned to each module's TOP area will be unique and not overlap another TOP area or any allocated literals.

The TOP pseudo-operation allows a module to specify top sector work areas and data.

External Symbol Definitions

Symbols which are defined within a module whose definitions are to be made available to other modules and code in the final absolute assembly must be listed using the DEF pseudo-operation.

DEF < symbol >

< symbol > may have an absolute, relocatable or TOP value. The value obtained in the absolute assembly in the latter two cases is the absolute value corresponding to the actual memory allocation for the module.

All DEF pseudo-operations must occur before any BSS or code generating operations.

The module name itself is automatically included in the DEF list with value relocatable zero.

External Symbol References

Symbols which are defined (via DEF) in other modules and symbols defined in the final absolute assembly must be listed using the XTN pseudo-operation.

XTN <symbol>

All XTN pseudo-operations must occur before any BSS or code generating instructions.

Symbol Definitions

Symbols may be defined using the label field as usual. Except in the case of SET or EQU with absolute operands, the values of such symbols are said to be relocatable or TOP values depending on where they appear.

The symbols defined within each MODule are completely distinct. Thus the same name may be used in different ways in different modules. Note however, that if two modules give the same symbol name on REF lines, an M flag will result if both modules are INcluded in the same final assembly.

There are, therfore, three types of symbols which can be referenced.

(a) Global symbols. Defined in the <global definitions> section. These values are always absolute.

(b) Local symbols. Defined within the current module. These values may be absolute, relocatable or TOP values depending on the corresponding definitions.

(c) External symbols. Listed in XTN lines in the current module. The values of such symbols are not known till the final absolute assembly.

Expression Formation Rules

The use of relocatable, TOP and external symbols in expressions is limited as follows:

External symbols (those defined in XTN statements) may only appear on their own and may not be used with any expression operator. They may not occur except as addresses including literals, immediate data operands, or I/O functions and channel codes.

The +(addition) operator allows a relocatable or TOP value to be added to an absolute value (but not vice versa) giving a relocatable or TOP result respectively.

The -(subtraction) operator allows an absolute value to be subtracted from a relocatable or TOP value (but not vice versa) giving a result value of the same type. It also allows two values of the same type (both relocatable or both TOP) to be subtracted to give an absolute value.

No other operators permit the use of relocatable or TOP values.

The only places where expressions with relocatable or TOP values may be used are the following:

Address operands for instructions,
DAC, ADDR or WORD operands,
EQU or SET operands (predefinition required),
Literal operands.

The special symbol $ may be used and has as value the relocatable or TOP value of the current location counter contents.

The special symbols $C and $X have as value absolute X'7FFE' regardsless of the current CNFG setting.

The special symbol $L may not be used in relocatable assemblies. The special symbols $B and $D may be used with their usual meanings.

The IN Pseudo-Operation

The pseudo-operation IN may be used to copy a module from a previously assembled library into the library being created. The format is:

IN <filename>,<modulename>

<filename> is the name of the relocatable library file containing the module. No unit number is given; the currently selected unit is searched first and then the other unit to find the file.

<modulename> is the name of the module to be included. As shown, this operand is required.

IN can occur in place of a MOD pseudo-operation and its text. It cannot occur in the middle of a module.

Error Flags

The use of a pseudo-operation not permitted in a relocatable assembly will generate an O flag.

The use of a DEF or XTN after a BSS or code generating instruction causes a Q flag to be posted.

An R (relocation error) flag for violation of the rules on expression formation given above.

The A, E, and X flags are never posted in a relocatable assembly, these errors will be detected during the final absolute assembly.

THE RASSEMBL UTILITY

RASSEMBL is a new SPD/DOS utility program which performs a relocatable assembly. The use of RASSEMBL is similar to that of ASSEMBLE except that the input is in the format described in this section and the output is the corresponding relocatable file.

The options and call are the same as for ASSEMBLE except for the program name.

The error codes are identical except that RA is used instead of AS and all references to "object" file are changed to "relocatable" file.

Format of Listing

The listing format is similar to that generated by ASSEMBLE except as follows:

The location counter listed is the appropriate relocatable or TOP value. Both sections start with the counter value at X'0000'.

The * for desectorizing does not appear since desectorizing is left to the absolute assembly. Also, no literal table or literal cross-reference table is printed.

The module number is printed as two hexadeciamal digits (00 = global definitions section) in the same place as the segment number of an absolute assembly.

Short form instructions list as follows:

mm-llll =[@]00.[=] vvvvt

mm	is the module number
llll	is the current (relocatable or top) assembly counter
@	is present only for an indexed instruction
00	is the opcode as two hexadecimal digits with all addressing bits off

.	is replaced by * if indirect addressing is specified
=	is present only for a literal reference
vvvv	is operand value
t	indicates the operand type

blank	absolute
R	relocatable
T	TOP section
X	external

in the case of external, the value vvvv listed is the serial number of the external reference as listed on the corresponding XTN line.

Long form jumps list as follows:

mm- *lll* :wwww [*] vvvvt

mm	as above
lll	
vvvv	
t	
wwww	first word of instruction
	present only if the jump is indirectly addressed

The values listed for WORD, DAC instructions and for EQU, SET and DEF operands have the type appended as described for one word instructions.

The value listed for DAC is preceded by * if indirect addressing is specified.

The cross reference table is sorted first by module number then by symbol name. It includes the module number, value and type of each symbol.

2. ABSOLUTE ASSEMBLIES

Absolute assemblies are performed with the ASSEMBLE utility. In fact, a normal assembly, as performed under previous releases of DOS, is still possible and is simply a special case of an absolute assembly in which no relocatable modules are included.

The enhancements to ASSEMBLE allow previously assembled relocatable modules to be included in an assembly. The effect is similar to including the source statements of the original module.

IN PSEUDO-OPERATION

The IN pseudo-operation allows a single specified module to be included:

IN < filename > , < modulename >

<filename> is the DOS filename of the relocatable library containing the module to be included. No unit is given. ASSEMBLE searches all units (currently selected unit first) to find this file.

<modulename> is the name of the module to be included. If this operand is omitted, then the first or only module is included.

The instructions and data in the relocatable section of the module are assembled with the location counter values adjusted according to the starting value at the time the IN was encountered after any required work alignment. In the case where the IN appears in the range of an XORG, the bias, if any, between the load and location counters is maintained.

The TOP section in the module, if any, is assembled in the literal pool region as specified by the second argument of SIZE. The literal pool area is allocated as follows:

First the TOP sector regions of INcluded modules are allocated at successively lower addresses, then the literals and links are allocated below these areas. Note that XORG has no effect in the TOP section data.

The location counter and load counter values following completion of the INclude are those obtained after assembling the last relocatable data (and are not affected by the presence of a TOP section).

The CNFG mode of the assembly is unaffected by an IN statement, regardless of the occurrence of CNFG statements in the included module. It is the programmer's responsibility to avoid INcluding modules containing instructions inappropriate to the program.

LIN PSEUDO-OPERATION

The LIN pseudo-operation controls the listing of data assembled by IN pseudo-operations.

LIN <expr>

<expr> is an expression whose value is 0 or 1. If the value is 0, then the assembled data is not listed unless an error flag is posted. If the value is 1, then the assembled data is listed. Any symbols appearing in <expr> must be predefined.

The default LIN mode if no LIN pseudo-operation appears in the source program or until the first LIN appears is LIN 0.

SIZE PSEUDO-OPERATION

The second argument to SIZE specifies the origin for TOP areas of INcluded modules as well as for literals and links.

In order to properly allocate TOP areas, the value of this second parameter must be known during pass one. For simplicity and consistency, a rule has been introduced requiring all symbols in the operand expressions of SIZE to be predefined.

CNFG PSEUDO-OPERATION

The initial CNFG must precede the SIZE statement, if present. This is necessary for proper allocation of TOP areas.

ERROR FLAGS

If an XTN symbol reference in an INcluded module is undefined, then a U flag will be posted on the IN line and on all generated instructions referencing this symbol. The cross-reference table will identify the undefined symbol.

If a DEF symbol (or the module name) is multiply defined (i.e., defined explicity in the absolute assembly or DEFined in some other module), then an M flag is posted in the IN line and on all generated instructions referencing the symbol. The cross-reference table will identify the multiply defined symbol.

If a generated operand address is odd when it should be even, the generated instruction is flagged with an E flag.

Generated instructions which are flagged are printed regardless of the LIN mode. By subtracting the location counter value on the IN line from the location counter value for the flagged line, the relocatable address within the original module text can be calculated.

A new flag, Y, is posted if the CNFG setting of a module does not match the current CNFG.

DOS OPTIONS

The operation of ASSEMBLE is unchanged except that two letter options have been added to control LIN mode:

H Enforce LIN 0 mode, regardless of LIN statements in the source program.

G Enforce LIN 1 mode, regardless of LIN statements in the source program.

3. RELOCATABLE FILES IN SPD/DOS

SPD/DOS Release 6 supports relocatable files as a fourth file type using the type letter R.

COPY

COPY has an R option to allow relocatable files to be copied. The form of relocatable files on external media is as for data files except that the label has an R instead of D.

DCOPY

DCOPY has an R option to allow relocatable files to be copied.

ERASE

ERASE has an R option to allow relocatable files to be erased.

LIST

LIST has an R option to allow relocatable files to be listed.

PACK

PACK has an R option allowing retention of relocatable files.

RENAME

RENAME has an R option to allow relocatable files to be renamed.

VERIFY

VERIFY has an R option to allow relocatable file labels to be verified.

SYSTEM SUBROUTINES

Relocatable files may be opened or created by using D&OPEN or D&CREA with an R (X'52') in the FDB&TYP of the FDB. They can then be read or written using D&READ or D&WRIT.

4. FORMAT OF RELOCATABLE FILES ON DISKETTE

A relocatable file is basically a sequential string of bytes which fills consecutive sectors of the file at any non-zero interlace factor. The standard value of this interlace factor as generated by the RASSEMBL utility will be chosen for efficiency, probably in the region of 9. No assumption should be made about this value on reading a relocatable file, the value should be obtained from the directory entry as usual.

MODULE DIRECTORY

The first part of the file is the module directory consisting of a series of entries in the form:

1-8 bytes	Module name. MSB of last character set on.
1 byte	Relative track of start of module (i.e., track number of start of module preamble minus track number of start of file).
1 byte	Sector (0-31) of start of module preamble.
1 byte	Offset (0-127) of start of module preamble within sector.

As indicated, the length of each entry is variable from 4-11 bytes, the entries spilling over sector boundaries as required.

The end of the module directory is indicated by a dummy entry with a name of X'FF'. This dummy entry correctly indicates the first unused byte location in the file. The preamble for the first module immediately follows this dummy entry, with X'FF' fill bytes optionally intervening.

MODULE PREAMBLE

The preamble is a contiguous string of bytes of variable length starting at the location indicated by the associated module directory entry. It has the following format:

Bytes 0-1 Highest (i.e., first unused) relocatable origin in module. This is used to determine the location and load counter values to be set following an IN pseudo-operation.

Bytes 2-3 Highest (i.e., first unused) TOP origin in module. X'0000' if no TOP pseudo-operation appears in module.

Byte 4 CNFG setting of module.

External Symbol Dictionary

A contiguous series of bytes containing one entry for each EXT or DEF pseudo-operation in the module source text, immediately following the five header bytes described above.

Either type of entry starts with the characters of the name, the MS bit of the last character being set on. In the case of DEF entries, a WS (see section on word specifications) follows which specifies the defined value of the symbol. An XTN entry complete with the name (an examination of the codes involved will indicate that this arrangement causes no ambiquities).

The order of DEF entries has no effect since the only function of these entries is to make the appropriate entries in the symbol table of the absolute assembly.

The order of the XTN entries is significant. The first entry is numbered X'0001' and subsequent entries are assigned successively higher numbers. References in the module text to external symbols use these numbers.

The DEF entry corresponding to the module name is present in the preamble (and indicates an associated value of relocatable X'0000').

End of Module Preamble

The end of the external symbol dictionary and hence of the preamble is marked by a single byte with value X'FF'. The module text follows immediately, or after X'FF' fill bytes.

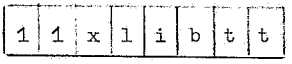
MODULE TEXT

The text for each module is a contiguous series of bytes in a special format designed to minimize the space required for relocatable libraries.

First we define some special sequences which are used throughout.

Word Specification (WS)

A WS is a specification of a word value or address operand consisting of a descriptor byte followed by a one or two byte value. The descriptor byte has the following form:



x Normally set to 1. Set to 0 only for an indexed one word instruction operand.

l Normally set to 0. Set to 1 only for a literal reference in a one word instruction (value given is literal operand).

i Normally set to 0. Set to 1 if the WS represents an address operand for an indirectly addressed instruction or pseudo-operation.

b Set to 1 if the following value is two bytes. Set to 0 if the following value is one byte (i.e., MS byte of value is X'00').

t t Type of value

0 0	Absolute value
0 1	Relocatable value
1 0	TOP value
1 1	External symbol reference (in this case the "value" is the external symbol number).

Byte Specification (BS)

A BS specifies a byte value, as used, for example, in an immediate class instruction. The following possibilities exist:

(a) For an external reference, a BS has the same form as a WS. Note that the descriptor byte has one of the two values X'E3' or X'E7'.

(b) Absolute byte values greater than or equal to X'E0' are represented by X'FF' followed by the byte value.

(c) Absolute byte values less than X'E0', the BS is simply the byte value in question.

One-Word (Word or Byte Class) Instructions

One byte containing the opcode followed by a WS giving the operand address. The opcode byte has all addressing bits off.

Immediate Class Instructions

Opcode byte + BS giving the byte operand.

Jump on Condition Instructions

Two bytes giving the opcode and second byte of the instruction followed by a WS giving the jump operand.

An SKP instruction is represented as the word value X'8800' using the special code X'F5' (see section on special codes).

Compare and Jump Instructions

One byte giving the opcode followed by BS giving the immediate byte operand followed by WS giving the jump address operand.

Generic Instructions

Two bytes of the instruction.

Input-Output Instructions

The opcode byte is generated first followed by a BS specifying the function code and a BS specifying the channel. In the case where both function and channel are absolute (non-external), and the function code is non-zero, these two BS's may be (but are not required to be) combined into a single BS specifying the second byte of the instruction. In the cases of JFACK and JTACK, a WS follows specifying the jump address.

Special Codes

X'F1'	BSS 1 (used for word alignment).
X'F2' a a	BSS aa where aa is a two byte absolute value giving the count.
X'F3' BS	Generate byte specified by BS once.
X'F4' a a BS	Generate byte specified by BS number of times indicated by two byte absolute value aa.
X'F5' WS	Generate word specified by WS once.
X'F6' a a WS	Generate word specified by WS number of times indicated by two byte absolute value aa.
X'F7'	Start of TOP sector section.
X'01' text X'01'	Generate text (0 or more bytes from TEXT, TXT8, LTX8, LTXT pseudo-operation). Note that the opcode value X'01' never appears so no confusion arises.

End of Module Text

The module text is terminated by a single byte X'FF'. The preamble for the next module, if any, follows. The bytes following the last byte of the last module text are unreferenced and undefined.

VÄXJÖ DATA SYSTEM AB

Distributör för Incoterm i Sverige, Norge, Finland och Danmark

VDS

VÄXJÖ
Adress: Box 3034, Smedjegatan 37, 350 03 VÄXJÖ
Telefon: 0470/10070
Telex: 52 138

HELSINGBORG
Adress: Landskronavägen 23, 252 32 HELSINGBORG
Telefon: 042/14 94 30

MALMÖ
Adress: Södergatan 12, 211 34 MALMÖ
Telefon: 040/724 50

STOCKHOLM
Adress: Skeppargatan 8, 114 52 STOCKHOLM
Telefon: 08/14 22 35

CORONADATA AB
Adress: Box 5143, Åvägen 18, 402 23 GÖTEBORG
Telefon: 031/20 03 80

CORONADATA A/S
Adress: Park Alle 296, DK - 2600 GLOSTRUP
Telefon: (02) 45 88 22

CORONADATA OY
Adress: Notstigen 4 C, SF - 00330 HELSINGFORS 33
Telefon: 0 - 48 87 22

CORONADATA A/S
Adress: Torggatan 7, OSLO 1
Telefon: 2 - 33 42 60

LANPRODUKTER, Växjö 1976