60361900

CONTROL DATA CORPORATION

MACRO ASSEMBLER REFERENCE MANUAL

CDC®

MASS STORAGE OPERATING SYSTEM

INTERACTIVE TERMINAL-ORIENTED SYSTEM

REVISION RECORD

REVISION	DESCRIPTION
A (4/72)	Original printing for Macro Assembler Version 3.0 and MSOS Version 4.0.
B (3/74)	General revision and update for compatibility with MSOS 4.1. Appendix D added.
C (5/75)	Revised to add a new Chapter 5 and the Glossary and incorporate PSR changes 1950 and 3249.
D (10/75)	Revised to reflect MSOS 4.3 changes.
E (8/76)	Revised to reflect MSOS 5 changes. Appendixes E and F added.
F (1/77)	Manual updated to reflect correct publication numbers for related manuals.
G (1/79)	Manual revised to reflect ITOS 2.0 release. The front cover, title page, and pages iii/iv and vi have been revised.

Publication No. 60361900

Additional copies of this manual may be obtained from the nearest Control Data Corporation sales office.

Printed in the United States of America

Address comments concerning this manual to:

Control Data Corporation
Publications and Graphics Division
4455 Eastgate Mall
La Jolla, California 92037

or use Comment Sheet in the back of this manual.

LIST OF EFFECTIVE PAGES

New features, as well as changes, deletions, and additions to information in this manual, are indicated by bars in the margins or by a dot near the page number if the entire page is affected. A bar by the page number indicates pagination rather than content has changed.

PAGE	REV
Cover	--
Title Page	--
ii	G
iii/iv	G
v	E
vi	G
vii thru ix	E
1-1 thru 1-3	E
1-4	B
1-5	A
1-6	A
1-7	D
1-8	A
2-1	E
2-2	E
2-3 thru 2-5	A
2-6 thru 2-8	B
2-9 thru 2-27	E
3-1	D
3-2	B
3-3	A
3-4	A
3-5	B
3-6	B
3-7	D
3-8	D
3-9 thru 3-12	B
3-13	C
3-14	B
3-15	B
3-16	A
3-17	D
3-18	E
4-1	E
4-2	B
4-3	D
4-4	D
4-5	A
4-6 thru 4-9	B
4-10	C
5-1	E
5-2	E
5-2.1	E
5-3 thru 5-6	C
5-7	E
5-8	E
5-9 thru 5-15	C
6-1	E
6-2	D
6-3 thru 6-6	C
Glossary-1	E
Glossary-2	E
A-1	E
A-2	A
A-3	B
A-4 thru A-10	E
B-1	E
C-1	E
C-2 thru C-4	A
D-1	B
D-2	B
E-1 thru E-3	E
F-1	E
F-2	E
Index-1 thru Index-4	E
Comment Sheet	G
Cover	--

PREFACE

The Macro Assembler for the CONTROL DATA® CYBER 18/1700 Computer Systems is a three-pass assembler that can convert source language input, including macro instructions, to relocatable output and generate list output. The source programs are written with symbolic machine, pseudo, and macro instructions.

Macro definitions may be defined by the user within the source program, or they may be placed on a separate macro library.

Input is from the standard input device, binary output is to the standard output device, and list output is to the standard list device.

The following describe functions occurring in each pass of the assembler.

Pass 1

Programmer-defined macros are processed, and appropriate tables are built. Whenever a macro instruction is encountered, the macro skeleton with actual parameters substituted is inserted into the source input on the mass storage device.

The source input is copied onto the mass storage device.

Sequence numbers of the input source images are checked.

Pass 2

Each source image on the mass storage device is read, and pass 2 errors are listed as they occur.

Conditional assembly pseudo instructions are processed.

Symbol and external tables are built.

Pass 3

Each image is read, and pass 3 errors are listed.

List and relocatable binary outputs are generated according to the input options.

TABLST

TABLST prints and punches the entry points and external images. The transfer image is punched.

An EOF image is output to the next load-and-go sector on mass storage.

A symbol table listing is given.

XREF

XREF creates and prints the cross-references lists.

This macro assembler operates under the Mass Storage Operating System (MSOS), Version 5, and the Interactive Terminal-Oriented System (ITOS), Version 2.

Refer to the Mass Storage Operating System (MSOS) 5 Reference Manual equipment configuration for the minimum hardware required by the Macro Assembler.

It is assumed that users of this manual are familiar with MSOS.

Following is a list of related publications.

Description	Publication No.
MSOS 5 Reference Manual	96769400
Mass Storage FORTRAN Version 3 A/B Reference Manual	60362000
1700 Computer System Codes	60163500
Small Computer Maintenance Monitor Reference Manual	39520200
MSOS 5 Instant	96769430
1700 MSOS 5 File Manager Version 1 Reference Manual	39520600
MSOS 5 Release Bulletin	96769440
MSOS 5 Installation Handbook	96769410
Small Computer Maintenance Monitor Instant	39521700
MSOS 5 Ordering Bulletin	96769490
Interactive Terminal-Oriented System (ITOS) Version 2 Reference Manual	96769240
Interactive Terminal-Oriented System (ITOS) Version 2 Installation Handbook	60475200

This product is intended for use only as described in this document. Control Data cannot be responsible for the proper functioning of undescribed features or undefined parameters.

CONTENTS

	PREFACE	v
CHAPTER 1	INSTRUCTION FORMAT	1-1
	1.1 Source Program	1-1
	1.2 Source Statement	1-1
	1.2.1 Location Field	1-2
	1.2.2 Remarks	1-2
	1.2.3 Instruction	1-2
	1.2.4 Address Field	1-2.1
	1.2.5 Comment Field	1-8
	1.2.6 Sequence Field	1-8
CHAPTER 2	MACHINE INSTRUCTIONS	2-1
	2.1 Storage Reference Instructions	2-1
	2.1.1 Address Modes	2-1
	2.1.2 Absolute Addressing	2-3
	2.1.3 Relative Addressing	2-4
	2.1.4 Constant Addressing	2-6
	2.1.5 Data Transmission Instructions	2-6
	2.1.6 Arithmetic Instructions	2-7
	2.1.7 Logical Instructions	2-8
	2.1.8 Jump Instructions	2-9
	2.1.9 Type 2 Storage Reference	2-10
	2.2 Register Reference Instructions	2-10.7
	2.3 Inter-Register Instructions	2-12
	2.3.1 Type 1 Inter-Register Instructions	2-12
	2.3.2 Type 2 Inter-Register Instructions	2-14
	2.4 Shift Instructions	2-14
	2.5 Skip Instructions	2-15
	2.5.1 Type 1 Skip Instructions	2-15
	2.5.2 Type 2 Skip Instructions	2-17
	2.6 Decrement and Repeat	2-18
	2.7 Field Reference Instructions	2-19
	2.8 Miscellaneous Instructions	2-20
	2.9 Negative Zero/Overflow Set	2-26
CHAPTER 3	PSEUDO INSTRUCTIONS	3-1
	3.1 Subprogram Linkage	3-1
	3.1.1 NAM	3-1
	3.1.2 END	3-1
	3.1.3 ENT	3-2
	3.1.4 EXT/EXT*	3-2
	3.2 Data Storage	3-4
	3.2.1 BSS	3-4
	3.2.2 BZS	3-4
	3.2.3 COM	3-5
	3.2.4 DAT	3-6

	3.3	Constant Declarations	3-7
		3.3.1 ADC/ADC*	3-7
		3.3.2 ALF	3-7
		3.3.3 NUM	3-9
		3.3.4 DEC	3-10
		3.3.5 VFD	3-11
	3.4	Assembler Control	3-13
		3.4.1 EQU	3-13
		3.4.2 ORG/ORG*	3-14
		3.4.3 IFA	3-15
		3.4.4 EIF	3-16
		3.4.5 OPT	3-17
		3.4.6 MON	3-17
	3.5	Listing Control	3-18
		3.5.1 NLS	3-18
		3.5.2 LST	3-18
		3.5.3 SPC	3-18
		3.5.4 EJT	3-18
CHAPTER 4	MACROS		4-1
	4.1	Macro Pseudo Instructions	4-1
		4.1.1 MAC	4-1
		4.1.2 EMC	4-2
		4.1.3 LOC	4-2
		4.1.4 IFC	4-2
	4.2	Macro Skeleton	4-3
	4.3	Macro Instruction	4-4
		4.3.1 Parameters	4-4
		4.3.2 Examples	4-6
CHAPTER 5	STANDARD MACRO LIBRARY		5-1
	5.1	Creating the Library	5-1
	5.2	Modifying the Library	5-2
	5.3	Programs in the Macro Library	5-2
		5.3.1 Formatting Macros	5-2
		5.3.2 File Manager Macros	5-3
		5.3.3 Monitor Request Macros	5-8
		5.3.4 Other Macros	5-14
CHAPTER 6	ASSEMBLER OUTPUT		6-1
	6.1	Control Options	6-1
		6.1.1 P Option	6-1
		6.1.2 X Option	6-1
		6.1.3 L Option	6-1
		6.1.4 C Option	6-1
		6.1.5 M Option	6-2
	6.2	Assembly Listing	6-2
		6.2.1 Error Listing	6-2
		6.2.2 Cross-Reference Listing	6-3
		6.2.3 Sample Program	6-4
		6.2.4 Sample Listing	6-4

GLOSSARY Glossary-1

APPENDIX A MNEMONIC INSTRUCTIONS CODES A-1

APPENDIX B PROGRAMMING CONSIDERATIONS B-1

APPENDIX C ASCII CODES C-1

APPENDIX D MACRO ASSEMBLER ERRORS D-1

APPENDIX E INSTRUCTION CODES E-1

APPENDIX F MACRO LIBRARY F-1

INDEX Index-1

TABLES

2-1 Type 2 Storage Addressing Relationships 2-10.4

INSTRUCTION FORMAT 1

1.1 SOURCE PROGRAM

The number of independent subprograms comprising a source program is limited only by available space. Each subprogram may be assembled independently, or several may be assembled as a group. The main subprogram of a group is the one to which initial control is given; it need not be the first subprogram. The last subprogram of a group must be followed by the MON pseudo instruction indicating the end of assembly and return to the operating system.

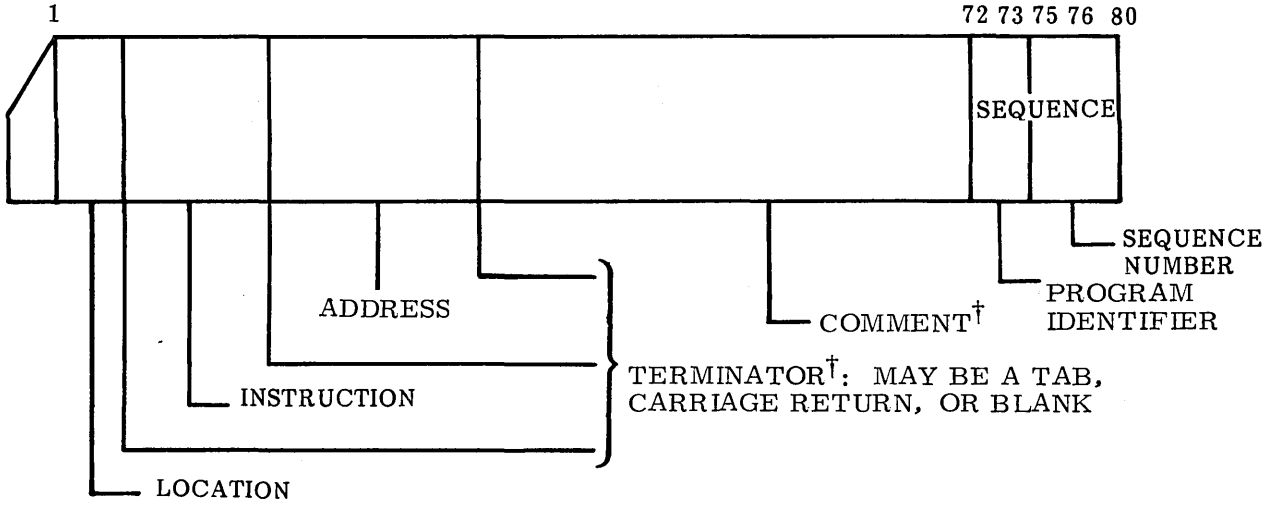
Communication between subprograms is effected by the subprogram linkage pseudo instructions and by the use of common and data storage.

At execution time, the entry point named in the END pseudo instruction specifies the entry point to which initial control passes. A jump to the dispatcher or an exit request (see the MSOS reference manual) signals return of control to the operating system upon job completion. EXIT or a jump to the dispatcher must be the last statement to be executed.

1.2 SOURCE STATEMENT

A source statement consists of location, instructions, address, remarks, and sequence fields (see figures 1-1 and 1-2). The first four fields may not exceed 72 characters; within that limitation they are free field. The sequence field is used when the source image is 80 characters; it is restricted to columns 73 through 80.

Each field is terminated by a tab ($B; paper tape only), carriage return (end of statement mark) or blanks. Any number of blanks may separate fields. A carriage return is always the end-of-statement mark on paper tape.



†Blanks are permitted in remarks without terminating the field. It can be terminated only by a carriage return or by reading column 72.

Figure 1-1. Normal Instruction, Free Field Format

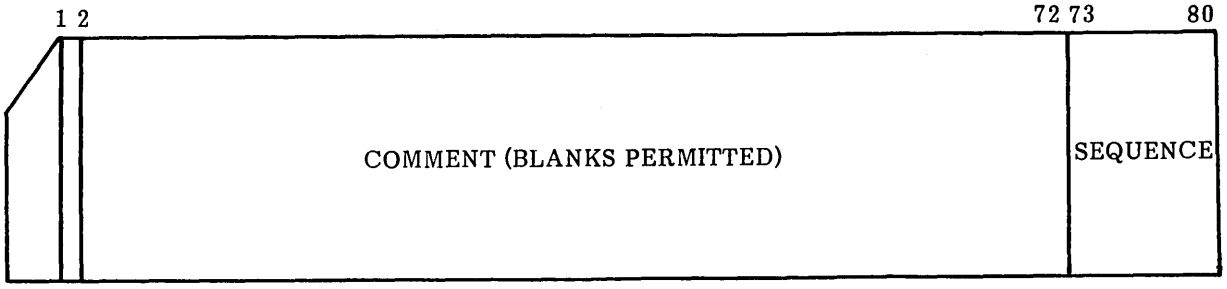


Figure 1-2. Comment

1.2.1 LOCATION FIELD

The location field of a source statement must begin in column 1.

This field is used to specify a labeled (label starting in column 1) or an unlabeled (blank or tab in column 1) statement.

The statement label is a symbolic name consisting of from one to six alphanumeric characters; the first must be alphabetic. Characters in excess of six are ignored. A two-character name makes the most efficient use of storage and assembly time.

Examples:

LOOP1	Legal
123456	Illegal; first character is numeric
P1	Legal
A123456	Legal; only A12345 is processed

1.2.2 REMARKS

An asterisk in column 1 of the location field specifies that the source statement is a remark. Comments, written in columns 2 through 72, are printed with the assembly list output but have no effect on the object program. An asterisk elsewhere in the location field is illegal. Remarks may also follow the address field of any instruction. There must be at least one blank separating the address field from the remarks.

1.2.3 INSTRUCTION

This field begins to the right of the location field and must be separated from it by at least one blank character or a tab. If the location field contains no label (blank or tab on column 1), the operation code may begin in column 2.

The operation code field contains the three-letter instruction codes for machine and pseudo instructions, or it contains macro instructions which may be up to six characters. Certain instructions may be followed by a one-character terminator.

The mnemonic instruction codes are listed in appendix A.

1.2.4 ADDRESS FIELD

The instruction field begins to the right of the operation code field, separated from it by at least one blank character or a tab. It is terminated by a blank or tab or by the 72nd character of the source statement. Exceptions are the macro instructions, which may have a continuation line, and the pseudo instruction ALF (section 3.3.2). The field may not necessarily contain an address (e.g., for some skip instructions it may contain a constant designating the number of words to be skipped).

This field contains an expression consisting of an operand or string of operands joined by arithmetic operators, or it may contain a series of operands separated by commas. An operand may be any of the following:

Symbolic name

Numeric constant

One of the special characters: * A Q M 0 I B

Symbolic Operand

A symbolic name used as an operand in the address field must be defined in one of the following ways.

Label in the location field of any machine instruction

Label in the location field of any macro instruction

Label in the location field of constant declaration pseudo instructions: ADC, ALF, NUM, DEC, VFD

Symbolic name in the address field of the pseudo instructions: EXT, COM, DAT, BSS, BZS, EQU

A defined symbolic name references a specific location in memory. It may be relocatable or absolute. A relocatable symbol refers to a location that may be relocated during loading.

Storage is divided into three areas: program, data, and common. These areas are defined at assembly time and the initial location of each is set to a relocation address of zero. The object code produced by the assembler contains addresses that are modified by a relocation factor to produce the actual address in memory.

A symbol is program relocatable if it references a location in the subprogram, data relocatable if it references a location in data storage, and common relocatable if it references a location in common storage. All other symbols are absolute. A symbol is made absolute by equating it to a number, an arithmetic expression, or another absolute symbol.

In all cases a symbolic label and a symbol defined by BSS or BZS take the relocation and value of the current location counter. The location counter of a program is originally program relocatable; however, its relocation may be changed by the ORG instruction.

An address expression that includes more than one operand must reference only one relocatable area. Terms of different relocation types must reduce to one relocatable area or to an absolute address. When the address mode of an instruction is made one-word relative by an asterisk terminator, the relocation type of the address expression must agree with the type of the current location counter.

A symbolic operand may be preceded by a plus or a minus sign. If preceded by a plus or no sign, the symbol refers to its associated value; if preceded by a minus, the symbol refers to the ones complement of its associated value. When an expression contains more than one symbol, the final sign of the expression is the algebraic sum of the operands.

Example:

RT=relocation type of current location counter: P=program relocatable, C=common relocatable, D=data relocatable, and A=absolute address.

```
RT      Label       Operation       Address

                    COM             COM1,COM2
                    DAT             DAT1,DAT2
                    EQU             D(1), E(3), G(E-D), H($1000), F(DAT1)
P                   BZS             A,B,C
P                   BZS             J,K(10)
```

The symbols D, E, G, and H are absolute; DAT1, DAT2, and F are data relocatable; COM1 and COM2 are common relocatable; A, B, C, J, and K are program relocatable.

```
P       START       ADC             0
P                   LDA*            START
P                   STA*            DAT1    (Error)
P                   STA*            COM1    (Error)
```

The errors resulted because the relocation types of the symbols in the address field do not match that of the location counter, and the one-word relative address mode was requested by an asterisk terminator.

```
RT      Label       Operation          Address

P                   LDA+               (Not an error)
```

Relocations need not match when mode is two-word absolute.

```
P                   LDA        START   (O.K., relocations match)
P                   LDA        COM1    (Not an error)
```

Assembler changes this instruction to two-word absolute because relocations do not match, but no error is indicated.

```
P                   LDA        COM2-DAT1+COM1-D+E-COM2+START-K+DAT2
```

This address expression results in a common relocation type; all other relocations cancel out (refer to address expressions).

```
                    ORG        DAT1
```

ORG changes the relocation of the location counter to data.

```
D                   LDA*       START   (Error)
D                   STA*       DAT2+9
                    ORG*
```

ORG* returns the location counter to original relocation.

```
P                   LDA*       START   (Not an error)
                    ORG        H
A                   LDA*       START   (Error)
A                   STA*       DAT1    (Error)
A                   LDA*       $1001
A                   STA-       B
                    ORG*
                    END
```

Numeric Operand

A numeric operand in the address field may be decimal or hexadecimal. A decimal number is represented by up to five decimal digits and must be within the range ±32767. A hexadecimal number is represented by a dollar sign and not more than four hexadecimal digits in the range ±7FFF. (Hexadecimal operands in the NUM pseudo instruction may be in the range ±FFFF.)

Numeric operands in the address field may be preceded by a plus or a minus sign. If a plus or no sign is specified, the binary equivalent of the number is the value used; a minus means the one's complement of the binary equivalent is the value.

A numeric operand has no relocation type; it is always absolute.

Address Expression

An address expression may be a single operand or a string of operands joined by the following arithmetic operators.

+	Addition
-	Subtraction
*	Multiplication
/	Division

Arithmetic operators may not follow each other without an intervening operand. Parentheses are not permitted for grouping terms.

The asterisk has an additional meaning as an operand. When it is used as the multiplication operator (refer to special characters), it must be immediately preceded by an operand which may be another asterisk. When the asterisk is used as an operator, only one of its associated operands may be relocatable.

The slash, used as the division operator, must be between two operands. The operand which follows may not be zero or relocatable.

An external name may be used in an address expression only as a single operand. Arithmetic operators preceding or following an external operand are illegal.

Example:

```
        NAM   EXAMPL
        COM   A,B
        EQU   C(1),D(5)
        EXT   G
        BZS   E(10),F
START   LDA   D-C/5+**2
        ADD   A-B/2
        ADD   E+5
        STA   G
        END
```

The first asterisk in the LDA instruction refers to the value of the current location counter.

The following instructions are illegal assuming the same pseudo instructions precede the START.

```
START   LDA   D-C**5+2      *5 has no intervening operator
        ADD   A-2/B         Division by relocatable operand
        ADD   E*F           Both operands are relocatable
        STA   G+5           An external must stand alone
```

The hierarchy for the evaluation of arithmetic expressions is:

/ or *	Evaluated first
+ or -	Evaluated next

Expressions containing operators at the same level are evaluated from left to right. The expression

A/B+C*D

is evaluated algrebraically as

A/B+(C)(D)

and not as any of the following:

$$\frac{(A)(D)}{B+C} \qquad \frac{A}{(B+C)(D)} \qquad \frac{A}{B+(C)(D)}$$

Parentheses may not be used for grouping operands. The algrebraic expression

(A-D)(B+C/E)

must be specified

A*B+A*C/E-D*B-D*C/E

The following expression is illegal.

(A-D)*(B+C/E)

Division in an address expression always yields a truncated result; thus, 11/3=3. The expression A*B/C may result in a value different from B/C*A. For example, if A=4, B=3, and C=2 then

A*B/C=4*3/2=6 but
B/C*A=3/2*4=4

All expressions are evaluated modulo $2^{15}-1$. An address expression consisting solely of numeric operands is absolute. If an expression contains symbolic operands, the final relocation for the expression is determined by the relocations of the symbolic operands. If the relocation of the operands is expressed by the following terms, the final relocation is the algebraic sum of the relocation terms.

±P Positive or negative program relocation

±C Positive or negative common relocation

±D Positive or negative data relocation

The relocation must reduce to one of the relocation terms or to zero. If zero, the location is absolute.

Example:

Source Statements			Relocation Formula
	COM	A,B	
	DAT	C,D	
	EQU	E(1),F(D)	
STRT	LDA	B+C-E*2-A-D	+C+D-C-D=0 (absolute)
	LDA	B+D-F+STRT-A-C	+C+D-D+P-C-D=P-D (illegal)
	LDA	B+D-E+STRT-A-C	+C+D+P-C-D=P (program)
	LDA	B-D-A	+C-D-C=-D (negative data)

Special Characters

Special characters may be used as operands in the address field of a source statement. Their definition may not be changed by the user. The three classes of special characters are storage, register, and index.

Class	Character	Referenced Location
Storage	*	Current location counter
	I	Location FF_{16}
Register	A	A register
	Q	Q register
	M	Mask register
	0	Destination registers
Index	Q	Index 1, Q register
	I	Index 2, location FF_{16}
	B	Index 1 plus index 2

Storage class characters (*, I) reference storage locations. The asterisk refers to the location of the current instruction. For a two word instruction, an asterisk references the location of the first word. Special character I refers to value FF_{16}. I is the only indexing character that may stand alone as an operand with storage reference instructions. It may not be redefined in a program. It may be used anywhere the value FF_{16} is used.

The register class characters (A, Q, M, and 0) are used only with inter-register transfer instructions. They refer to the A, Q, and M (mask) registers. Character 0 sets the destination registers to zero (section 2.5).

Examples:

Instruction	Function
SET A,Q,M	Set A, Q, and mask registers to ones
TRA Q	Transfer contents of A register to Q register
LAM M	Transfer logical product of A and mask register to mask register

Index class characters (Q, I, and B) are used in conjunction with an address expression to refer to the index registers. Any one character may follow an address expression; it is separated from the expression by a comma with no intervening blank. Indexing may be used only with storage reference instructions.

Q	Contents of Q register are added to contents of the expression to form the actual address
I	Contents of location FF_{16} are added to contents of address expression to form the actual address
B	Contents of Q register are added to address expression and this sum is added to contents of FF_{16} to produce the actual address

Examples:

Address Field		Function
LOC1,B	Legal	Contents of registers Q and FF_{16} and the contents of LOC1 are added to produce the actual address
,,I	Illegal	Character following first comma is assumed to be index character
TAG2,Q,I	Illegal	Only one index notation allowed
Q	Illegal	Unless Q has been previously defined as a location symbol or is being used with the inter-register transfer instruction, it must follow a location symbol
TAG3,I	Legal	Contents of FF_{16} and TAG3 are added to produce the actual address

1.2.5 COMMENT FIELD

The address field is followed by the comment field which is used for remarks. Remarks do not affect the object code, but are printed as part of the list output. The comment field terminates at column 72, or with a carriage return (paper tape). Blanks are permitted in the comment field.

1.2.6 SEQUENCE FIELD

When the input image is 80 characters, columns 73 through 80 are available for sequencing; 73 through 75 may be used for program identification, 76 through 80 for a sequence number.

Sequence numbers are checked for errors only if the input image is 80 characters. Each sequence number must be greater than or equal to the previous sequence number. The value of a character in the sequence number is in ASCII code except that a blank is treated as zero.

MACHINE INSTRUCTIONS 2

Machine instructions represented by a three-letter mnemonic code are divided into six basic classes and six additional classes of enhanced instructions.

BASIC CLASSES

Group A storage reference	Shift
Group B storage reference	Skip
Register reference	Inter-register transfer

ENHANCED INSTRUCTION CLASSES

Type 2 storage reference	Type 2 skip
Field reference	Type 2 inter-register transfer
Decrement and repeat	Miscellaneous

Storage reference instructions result in one or two machine words, depending on modification. Other machine instructions result in one machine word.

Appendix A lists the machine instructions in the order in which they are discussed in this chapter and also defines groups A and B storage reference instructions.

2.1 STORAGE REFERENCE INSTRUCTIONS

Group A and B storage reference instructions use storage addresses as operands or as operand addresses. Group B instructions include jump instructions and may not use the constant mode of addressing. Type 2 storage reference instructions use the enhanced two or three-word format. These may be 8- or 16-bit address.

2.1.1 ADDRESS MODES

Group A storage reference instructions allow three modes of addressing: absolute, relative, and constant. Group B does not allow the use of the constant mode, but is otherwise the same as group A.

Special characters designate the mode of addressing, the number of words for the instruction, and indirect addressing.

Character	Description
*	Asterisk as the last character of operation code specifies relative addressing in a one-word instruction
-	Minus as the last character of operation code specifies absolute addressing in a one-word instruction
+	Plus as the last character of operation code specifies absolute addressing in a two-word instruction

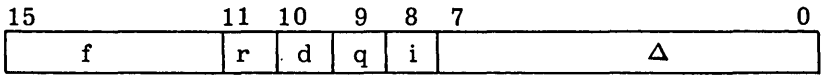
Character	Description
=	Equal sign as the first character in address field preceding a constant indicates constant addressing; the instruction is always two words
()	Parentheses enclosing the address expression indicate indirect addressing

If no character is specified as a terminator to the operation code, two-word relative addressing is assumed with the following exceptions.

1. If a constant is specified, the constant mode is assumed.
2. If the relocation type of the address expression differs from the relocation type of the location counter, two-word absolute addressing is assumed.
3. If a nonrelative external is referenced, absolute addressing is assumed.

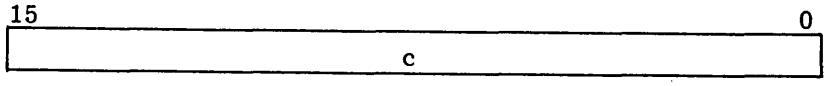
The machine language format resulting from a storage reference instruction is illustrated as follows.

First word:

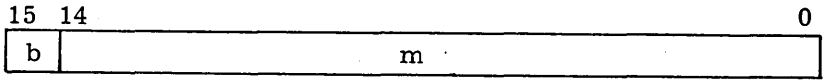


- f 4-bit operation code is defined below
- r Specifies relative addressing
- d Specifies indirect addressing
- q Index register 1 flag; specifies adding contents of Q register to address
- i Index register 2 flag; specifies adding contents of storage register FF_{16} to address
- Δ 8-bit field; may be relative or absolute address for one-word instructions. When zero, indicates two-word instruction.

Second word (when used):



- c 16-bit field for constant addressing or relative address. When it contains relative address, bit 15 is the sign.



- b Indirect address bit
- m Memory address

Address expressions are evaluated modulo $2^{15}-1$.

2.1.2 ABSOLUTE ADDRESSING

The value of the address expression of a one-word absolute instruction must be non-relocatable. The evaluated result is stored in 8 bits of the machine word. If this value is greater than 256, it is flagged as an error. If the 8-bit Δ field is zero, two machine words are assumed regardless of the operation code terminator; no error message is printed. If the address expression is enclosed in parentheses for indirect addressing, bit 10 of the first word is set to 1.

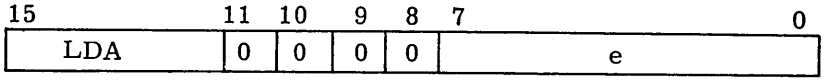
Examples:

One Word, Direct

Instruction:

LDA- e

Machine Word:

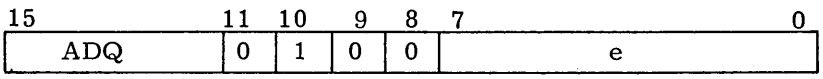


One Word, Indirect

Instruction:

ADQ- (e)

Machine Word:



The value of the address expression of a two-word absolute instruction is stored in the least significant bits of the second word. If the expression is enclosed in parentheses for indirect addressing, bit 15 of the second word is set to 1. The indirect address bit 10 in the first word is always set to 1 when two-word absolute addressing is specified whether the address expression is specified as indirect or direct. This indicates that the address expression is in the second word. The 8-bit Δ field of the first word is set to zero for two-word instructions.

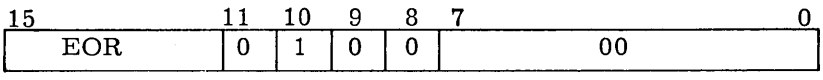
Examples:

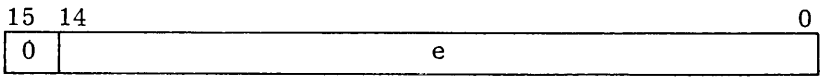
Two Word, Direct

Instruction:

EOR+ e

Machine Words:



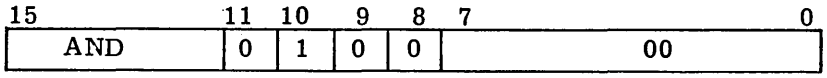


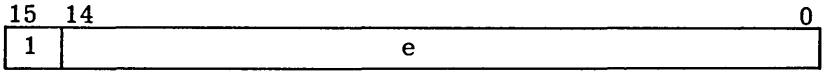
Two Word, Indirect

Instruction:

AND+ (e)

Machine Words:





2.1.3 RELATIVE ADDRESSING

When one-word relative addressing is specified, the value of the current location counter is subtracted (16-bit ones complement arithmetic) from the evaluated address expression. The result is placed in the 8-bit Δ field. If the value of the result is outside the range $\pm 7F_{16}$, an error condition is flagged. An error condition is also flagged if the relocation type of the address expression differs from that of the location counter. If the 8-bit Δ field is zero, two words are assumed regardless of the operation code terminator. No error message is printed for this condition.

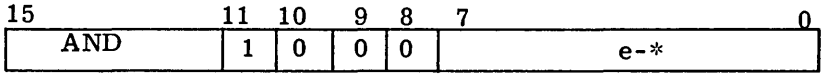
Examples:

One Word, Direct

Instruction:

AND* e

Machine Word:

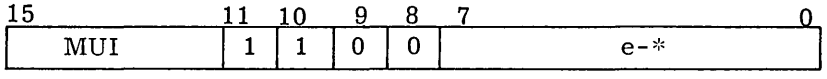


One Word, Indirect

Instruction:

MUI* (e)

Machine Word:



In the expression e-* the asterisk indicates the value of the current location counter.

When a two-word instruction is specified, the value of the current location counter plus one is subtracted (using 16-bit 1's complement arithmetic) from the value of the address expression to obtain the 16-bit second word. If the relocation type of the address expression differs from that of the location counter and the address does not reference an external, the assembler forces a two-word absolute instruction. If the address expression is an external reference, the instruction is absolute or relative depending on the definition of the external.

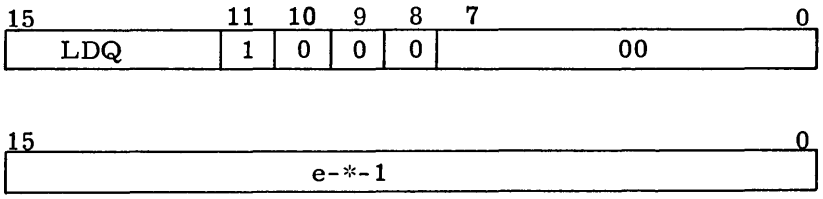
Examples:

Two Word, Direct

Instruction:

LDQ e

Machine Words:

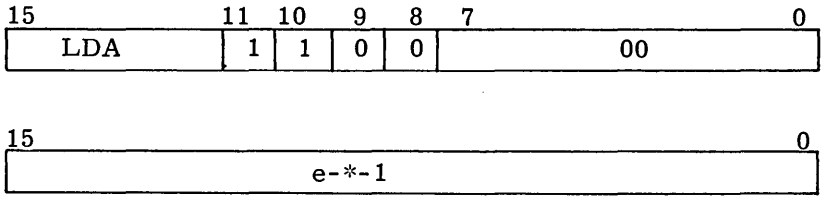


Two Word, Indirect

Instruction:

LDA (e)

Machine Words:



In the expression, e-*-1, the asterisk indicates the value of the current location counter.

2.1.4 CONSTANT ADDRESSING

Constant addressing may be used only for Group A storage reference instructions. Constants in the address field are preceded by an equal sign and a one-letter code. A constant may be one of the following:

Code	Type	Meaning
A	aa	2 alphanumeric characters
N	±ddddd	5-digit decimal number with or without a leading sign
N	±$hhhh	4-digit hexadecimal number preceded by $, with or without a sign
X	e	Address expression evaluated modulo $2^{15}-1$
X	(e)	Address expression evaluated modulo $2^{15}-1$, with bit 15 set

Examples:

DVI	=N$1000	(Hexadecimal constant)
ADD	=N-12345	(Decimal constant)
LDA	=AXY	(ASCII constant)
AND	=XTAG1+5	(Address expression constant)

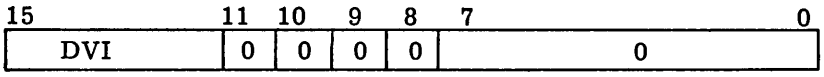
An instruction containing a constant in the address field results in two machine words.

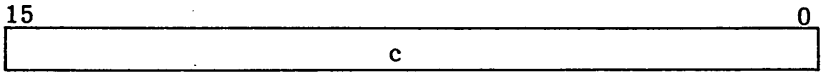
Example:

Instruction:

DVI	=nc	(n is the code, c is the constant)

Machine Words:





2.1.5 DATA TRANSMISSION INSTRUCTIONS

STQ (F=4) Store Q. Store the contents of the Q register in the storage location specified by the effective address. The contents of Q are not altered.

STA (F=6) Store A. Store the contents of the A register in the storage location specified by the effective address. The contents of A are not altered.

SPA (F = 7) Store A, Parity to A. Store the contents of the A register in the storage location specified by the effective address. Clear A if the number of 1 bits in A is odd. Set A equal to 0001_{16} if the number of 1 bits in A is even. The contents of A are not altered if the write into storage is aborted because of parity error or protect fault.

LDA (F = C) Load A. Load the A register with the contents of the storage location specified by the effective address. The contents of the storage location are not altered.

LDQ (F = E) Load Q. Load the Q register with the contents of the storage location specified by the effective address. The contents of the storage location are not altered.

2.1.6 ARITHMETIC INSTRUCTIONS

All the following arithmetic operations use one's complement arithmetic.

MUI (F = 2) Multiply Integer. Multiply the contents of the storage location, specified by the effective address, by the contents of the A register. The 32-bit product replaces the contents of Q and A, the most significant bits of the product in the Q register.

DVI (F = 3) Divide Integer. Divide the combined contents of the Q and A registers by the contents of the effective address. The Q register contains the most significant bits before dividing. If a 16-bit dividend is loaded into A, the sign bit of A must be extended throughout Q. The quotient is in the A register and the remainder is in the Q register at the end of the divide operation.

The OVERFLOW indicator is set if the magnitude of the quotient is greater than the capacity of the A register. Once set, the OVERFLOW indicator remains set until a Skip On Overflow (SOV) or Skip On No Overflow (SNO) instruction is executed.

ADD (F = 8) Add to A. Add the contents of the storage location, specified by the effective address, to the contents of the A register.

The OVERFLOW indicator is set if the magnitude of the sum is greater than the capacity of the A register. Once set, the OVERFLOW indicator remains set until a Skip On Overflow (SOV) or Skip On No Overflow (SNO) instruction is executed.

SUB (F = 9) Subtract From A. Subtract the contents of the storage location, specified by the effective address, from the contents of the A register. Operation on overflow is the same as for an Add to A instruction.

RAO (F = D) Replace Add One in Storage. Add one to the contents of the storage location specified by the effective address. The contents of A are not altered. Operation on overflow is the same as for an Add to A instruction.

ADQ (F = F) Add to Q. Add the contents of the storage location, specified by the effective address, to the contents of the Q register. Operation on overflow is the same as for an Add to A instruction.

2.1.7 LOGICAL INSTRUCTIONS

The AND (AND with A) instruction achieves its results by forming a logical product. A logical product is a bit-by-bit multiplication of two binary numbers according to the following rules:

0 x 0 = 0	1 x 0 = 0
0 x 1 = 0	1 x 1 = 1

Example:

```
  0011  Operand A
x 0101  Operand B
  0001  Logical Product
```

A logical product is used, in many cases, to select only specific portions of an operand for use in some operation. For example, if only a specific portion of an operand in storage is to be entered into the A register, the operand is subjected to a mask in A. This mask is composed of a predetermined pattern of 0s and 1s. Executing the AND instruction causes the operand to retain its original contents only in those bits which have 1s in the mask in A.

The EOR (Exclusive OR with A) instruction achieves its result by forming an exclusive OR. Executing the EOR instruction causes the operand to complement its original contents only in those bits which have 1s in the mask in A. An exclusive OR is a bit-by-bit logical subtraction of two binary numbers according to the following rules:

Exclusive OR

A	B	A ⊻ B
1	1	0
1	0	1
0	1	1
0	0	0

Example:

```
  0011  Operand A
x 0101  Operand B
  0110  Exclusive OR
```

AND (F = A) AND with A. Form the logical product, bit-by-bit, of the contents of the storage location specified by the effective address and the contents of the A register. The result replaces the contents of A. The contents of storage are not altered.

EOR (F = B) Exclusive OR with A. Form the logical difference (exclusive OR), bit-by-bit, of the contents of the storage location specified by the effective address and the contents of the A register. The result replaces the contents of A. The contents of storage are not altered.

2.1.8 JUMP INSTRUCTIONS

A Jump (JMP) instruction causes a current program sequence to terminate and initiates a new sequence at a different location in storage. The program address register, P, provides continuity between program instructions and always contains the storage location of the current instruction in the program.

When a Jump instruction occurs, P is cleared and a new address is entered.† In the Jump instruction, the effective address specifies the beginning address of the new program sequence. The word at the effective address is read from storage and interpreted as the first instruction of the new sequence.

A Return Jump (RTJ) instruction enables the computer to leave the main program, jump to some subprogram, execute the subprogram, and return to the main program via another instruction. The Return Jump provides the computer with the necessary information to enable returning to the main program. Figure 2-1 shows how a Return Jump instruction can be used.

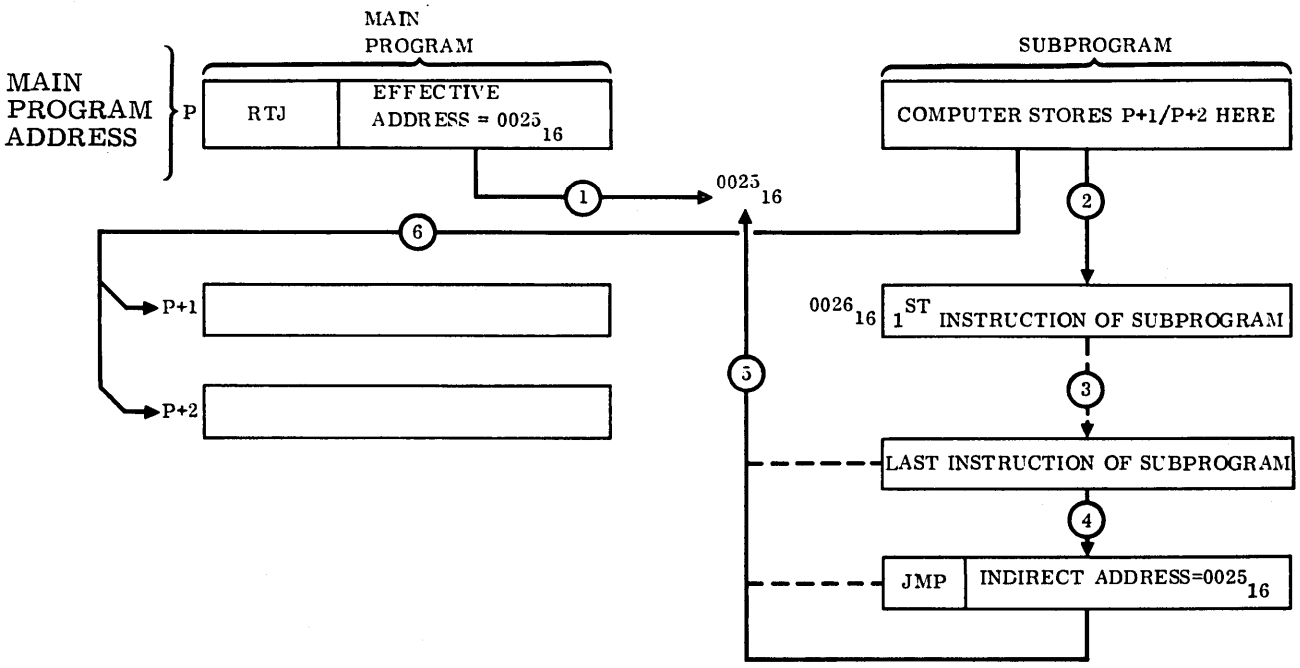


Figure 2-1. Program Using Return Jump Instruction

†Jumps or return jumps from unprotected to protected storage cause a fault, but the address that is saved in the trap location is the destination address (i.e., the address of the next sequential main program instruction).

An RTJ instruction is executed at main program address P. The computer jumps to effective address 0025_{16} and stores P+1 or P+2 (depending on the address mode of RTJ) at this location. Then the program address counter P is set to 0026_{16} and the computer starts executing the subprogram. At the end of the subprogram, the computer executes a jump instruction (JMP) with indirect addressing. This causes the computer to jump to the address specified by the subprogram address 0025_{16} (P+1 or P+2 of the main program). Now main program execution continues at P+1 or P+2.

JMP (F = 1) Jump. Jump to the address specified by the effective addresses. This effectively replaces the contents of program address counter P with the effective address specified in the JMP instruction.

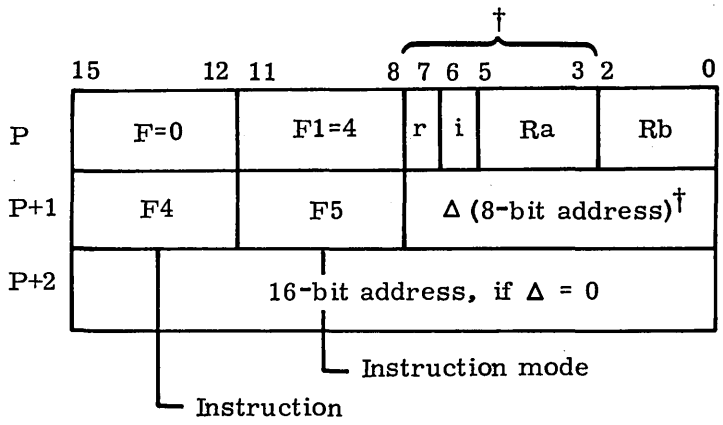
RTJ (F = 5) Return Jump. Replace the contents of the storage location specified by the effective address with the address of the next consecutive instruction. The address stored in the effective address is P+1 or P+2, depending on the addressing mode of RTJ. The contents of P are then replaced with the effective address plus one.

2.1.9 TYPE 2 STORAGE REFERENCE

NOTE

Instruction formats for enhancements to the instruction repertoire are upward-compatible with the existing 1704/14/84 computer

FORMAT:



Type 2 storage reference instructions are identified by F field = 0, F1 = 4, and the r, i, Ra, and Rb fields are not all zero. (If these fields are all zero, the instruction is an EIN.) This instruction is made up of two words if $\Delta \neq 0$ or three words otherwise. The type 2 storage reference instructions contain four parts: instruction field (F4), instruction mode field (F5), addressing mode fields (delta, r, i, and Ra), and the Rb register. From these, two operands (A and B) are specified for executing the instruction.

The F4 determines the instruction (e.g., add, subtract, etc.); F5 determines the instruction mode.

---

† Addressing mode field

F5	Mode
0	word processing, register destination
1	word processing, memory destination
2	character processing, register destination
3	character processing, memory destination

F5 is not used for subroutine jumps and subroutine exit. The register/memory destination bit of F5 is not used for compare instructions.

The addressing mode requires four fields:

Field	Addressing Mode
Δ	Eight or 16-bit address. Δ ≠0, 8-bit; Δ =0, 16-bit with the address in word P+2
r	Relative address
i	Indirect address
Ra	Index register

Type 1 storage instructions allow indexing by one or two registers (I and/or Q); type 2 allows indexing by any one of seven registers (1, 2, 3, 4, Q, A, or I).

The addressing mode fields determine the effective address for operand A; the Rb register and the instruction mode field (F5) determine the address for operand B. For character addressing, the effective address (operand A and the Rb register) are combined to generate the actual character effective address. Operand B is always the A register for character addressing.

CAUTION

For character addressing, a selection of absolute (r=0), no indirect (i=0), no index register (Ra=0), and no character register (Rb=0) results in an EIN instruction.

Unspecified combinations of F4, F5, and Rb are reserved for future expansion.

Addresses are defined below:

- Instruction address: The address of the instruction being executed, also called P
- Indirect address: A storage address that contains an address rather than an operand. There is no multi-level indirect addressing for type 2 storage reference instructions.
- Base address: The operand address after all indirect addressing has been accomplished but before modification by an index register. The base address is the effective address if no indexing is specified.
- Effective address: The final address of the operand
- Indexing: If specified, the contents of the Ra register are added to the base address to form the effective address. Indexing occurs after indirect addressing has been completed.

The computer uses the 16-bit ones complement adder during indexing operations. Consequently, index register contents are treated as signed quantities (bit 15 is the sign bit).

- Registers: The Ra and Rb registers are defined as follows:

Register	Value
None	0
1	1
2	2
3	3
4	4
Q	5
A	6
I	7

Type 2 storage reference instructions have eight types of addressing modes:

- Eight-bit absolute — (r=0, i=0, $\Delta \neq 0$)

  The base address equals delta. The sign bit of delta is not extended. The contents of index register Ra, when specified, are added to the base address to form the effective address.

- Eight-bit absolute indirect — (r=0, i=1, $\Delta \neq 0$)

  The eight-bit value of delta is an indirect address. The sign bit of delta is not extended. The contents of this address in low core (addresses 0001 to 00FF) are the base address. The contents of index register Ra, when specified, are added to the base address to form the effective address.

- Eight-bit relative — (r=1, i=0, $\Delta \neq 0$)

  The base address is equal to the instruction address plus one, P+1, plus the value of delta with sign extended. The contents of index register Ra, when specified, are added to the base address to form the effective address.

  If no indexing takes place, the addresses that can be referenced in the eight-bit relative mode are restricted to the program area. Delta is eight bits long; thus the computer references a location between $P-7E_{16}$ and $P+80_{16}$ inclusive.

- Eight-bit relative indirect — (r=1, i=1, $\Delta \neq 0$)

  The address of the second word of the instruction, P+1, plus the value of delta with sign extended is an indirect address. The contents of this address are the base address. The contents of index register Ra, when specified, are added to the base address to form the effective address.

- Absolute constant — (r=0, i=0, $\Delta = 0$)

  The address of the third word of the instruction P+2, is the base address. The contents of the index register Ra, when specified, are added to the base address to form the effective address. Thus, when Ra is not specified, the contents of P+2 are the value of the operand.

  Note that there is no immediate operand condition (i.e., indexing is specified and the instruction is read-operand type) as there is for type 1 storage reference addressing.

- Sixteen-bit absolute — (Storage) (r=0, i=1, $\Delta = 0$)

  The base address equals the contents of P+2 plus P+2. The contents of index register Ra, when specified, are added to the base address to form the effective address.

- Sixteen-bit relative — (r=1, i=0, Δ=0)

  The base address equals the contents of P+2 plus P+2. The contents of index register Ra, when specified, are added to the base address to form the effective address.

- Sixteen-bit relative indirect — (r=1, i=1, Δ=0)

  The address of the third word of the instruction, P+2, plus the contents of the third word of the instruction are an indirect address. The contents of this address are the base address. The contents of index register Ra, when specified, are added to the base address to form the effective address.

Table 2-1 summarizes the eight addressing modes. Δ, r, i, and Ra are specified as is the effective address and the address of the next instruction.

SJE
(F4=5, F5=0, Rb=0)

Subroutine/Jump Exit. The contents of P are replaced with the effective address. This instruction can be used as a jump or subroutine exit. For example, if Δ=1, and Ra has been set up by a previous subroutine jump, control is returned following that subroutine jump. Subroutine jumps save the address of the last word of the instruction, rather than the next instruction so that the subroutine jump exit may be a two-word instruction (Δ is nonzero) rather than three.

For example, the following program makes a subroutine jump at location 1000. The A register contains 1002 upon entry to subroutine SUB. Upon SUB's completion, it exits to location 1003.

```
1000    0446              SJA+      SUB
1001    5000
1002    2000
1003                        .
                            .
                            .
2000    ...     SUB       ...       ...
                            .
                            .
                            .
2020    0430              SJE-      1, A
2021    5001
```

CAUTION

Since Rb=0, a selection of absolute (r=0), no indirect (i=0), and no index register (Ra=0) result in an EIN instruction.

NOTE

In the following instructions, Rb has the numeric value corresponding to the alphanumeric R: 1 → 1, 2 → 2, 3 → 3, 4 → 4, 5 → Q, 6 → A, 7 → I.

SJr
(F4=5, F5=0, Rb→r)

Subroutine Jump. Loads the r register with the address of the last word of this instruction (i.e., P+1 for delta not zero; P+2 for delta zero). The contents of P are then replaced with the effective address.

TABLE 2-1. TYPE 2 STORAGE ADDRESSING RELATIONSHIPS

Addressing Modes	Delta	r	i	Ra	Effective Address (EA)	Address of Next Instruction
8-Bit Absolute	≠0	0	0	0	Δ	P+2
		0	0	1	Δ+(1)	P+2
		0	0	2	Δ+(2)	P+2
		0	0	3	Δ+(3)	P+2
		0	0	4	Δ+(4)	P+2
		0	0	5	Δ+(Q)	P+2
		0	0	6	Δ+(A)	P+2
		0	0	7	Δ+(I)	P+2
8-Bit Absolute Indirect	≠0	0	1	0	(Δ)	P+2
		0	1	1	(Δ)+(1)	P+2
		0	1	2	(Δ)+(2)	P+2
		0	1	3	(Δ)+(3)	P+2
		0	1	4	(Δ)+(4)	P+2
		0	1	5	(Δ)+(Q)	P+2
		0	1	6	(Δ)+(A)	P+2
		0	1	7	(Δ)+(I)	P+2
8-Bit Relative†	≠0	1	0	0	P+1+Δ	P+2
		1	0	1	P+1+Δ+(1)	P+2
		1	0	2	P+1+Δ+(2)	P+2
		1	0	3	P+1+Δ+(3)	P+2
		1	0	4	P+1+Δ+(4)	P+2
		1	0	5	P+1+Δ+(Q)	P+2
		1	0	6	P+1+Δ+(A)	P+2
		1	0	7	P+1+Δ+(I)	P+2
8-Bit Relative Indirect†	≠0	1	1	0	(P+1+Δ)	P+2
		1	1	1	(P+1+Δ)+(1)	P+2
		1	1	2	(P+1+Δ)+(2)	P+2
		1	1	3	(P+1+Δ)+(3)	P+2
		1	1	4	(P+1+Δ)+(4)	P+2
		1	1	5	(P+1+Δ)+(Q)	P+2
		1	1	6	(P+1+Δ)+(A)	P+2
		1	1	7	(P+1+Δ)+(I)	P+2
Absolute Constant	=0	0	0	0	P+2	P+3
		0	0	1	P+2+(1)	P+3
		0	0	2	P+2+(2)	P+3
		0	0	3	P+2+(3)	P+3
		0	0	4	P+2+(4)	P+3
		0	0	5	P+2+(Q)	P+3
		0	0	6	P+2+(A)	P+3
		0	0	7	P+2+(I)	P+3

NOTE: ( ) denotes contents of.

†For these addressing modes, delta is sign extended.

TABLE 2-1. TYPE 2 STORAGE ADDRESSING RELATIONSHIPS (Continued)

Addressing Modes	Delta	r	i	Ra	Effective Address (EA)	Address of Next Instruction
16-Bit Absolute (Storage)	=0	0	1	0	(P+2)	P+3
		0	1	1	(P+2)+(1)	P+3
		0	1	2	(P+2)+(2)	P+3
		0	1	3	(P+2)+(3)	P+3
		0	1	4	(P+2)+(4)	P+3
		0	1	5	(P+2)+(Q)	P+3
		0	1	6	(P+2)+(A)	P+3
		0	1	7	(P+2)+(I)	P+3
16-Bit Relative	=0	1	0	0	P+2+(P+2)	P+3
		1	0	1	P+2+(P+2)+(1)	P+3
		1	0	2	P+2+(P+2)+(2)	P+3
		1	0	3	P+2+(P+2)+(3)	P+3
		1	0	4	P+2+(P+2)+(4)	P+3
		1	0	5	P+2+(P+2)+(Q)	P+3
		1	0	6	P+2+(P+2)+(A)	P+3
		1	0	7	P+2+(P+2)+(I)	P+3
16-Bit Relative Indirect	=0	1	1	0	(P+2+(P+2))	P+3
		1	1	1	(P+2+(P+2))+(1)	P+3
		1	1	2	(P+2+(P+2))+(2)	P+3
		1	1	3	(P+2+(P+2))+(3)	P+3
		1	1	4	(P+2+(P+2))+(4)	P+3
		1	1	5	(P+2+(P+2))+(Q)	P+3
		1	1	6	(P+2+(P+2))+(A)	P+3
		1	1	7	(P+2+(P+2))+(I)	P+3

ARr
(F4=8, F5=0, Rb→r)

Add Register. Adds (using one's complement arithmetic) the contents of the storage location specified by the effective address to the contents of the r register. Operation on overflow is the same as for the ADD instruction. The contents of storage are not altered.

SBr
(F4=9, F5=0, Rb→r)

Subtract Register. Subtracts (using one's complement arithmetic) the contents of the storage location specified by the effective address from the contents of the r register. Operation on overflow is the same as for the ADD instruction. The contents of storage are not altered.

ANr
(F4=A, F5=0, Rb→r)

AND Register. Forms the logical product (AND), bit by bit, of the contents of the storage location specified by the effective address and the contents of the r register. The result places the contents of the r register. The contents of storage are not altered.

AMr
(F4=A, F5=1, Rb→r)

AND Memory. Forms the logical product (AND), bit by bit, of the contents of the storage location specified by the effective address and the contents of the r register. The result replaces the contents of the storage location specified by the effective address. The original contents of the storage location (specified by the effective address) replace the contents of the A register. The contents of the r register are not altered unless r is the A register. Memory is locked until completion of the instruction. This instruction is useful for memory-to-memory communication between CPUs.

LRr (F4=C, F5=0, Rb→r)	Load Register. Loads the r register with the contents of the storage location specified by the effective address. The contents of storage are not altered.
SRr (F4=C, F5=1, Rb→r)	Store Register. Stores the contents of the r register in the storage location specified by the effective address. The contents of the r register are not altered.
LCA (F4=C, F5=2, Rb= character flag+index)	Load Character to A. Loads bits 0 through 7 of the A register with a character from the storage location specified by the sum of the effective address and bits 1 through 15 of the Rb register. Bit 0=0 of the Rb register specifies the left character (bits 8 through 15) of the storage location; bit 0=1 specifies the right character (bits 0 through 7). Bits 8 through 15 of the A register are cleared to zero. The contents of storage are not altered.
SCA (F4=C, F5=3, Rb= character flag+index)	Store Character from A. Stores bits 0 through 7 of the A register into a character of the storage location specified by the sum of the effective address and bits 1 through 15 of the Rb register. Bit 0=0 of the Rb register specifies the left character (bits 8 through 15) of the storage location; bit 0=1 specifies the right character (bits 0 through 7). The contents of the A register and the other character of storage are not altered.
ORr (F4=D, F5=0, Rb→r)	OR Register. Forms the logical sum (inclusive OR), bit by bit, of the contents of the storage location specified by the sum of the effective address and the contents of the r register. The result replaces the original contents of the r register. The contents of storage are not altered.
OMr (F4=D, F5=1, Rb→r)	OR Memory. Forms the logical sum (inclusive OR), bit by bit, of the contents of the storage location specified by the effective address and the contents of the r register. The result replaces the contents of the storage location specified by the effective address. The original contents of the storage location (specified by the effective address) replace the contents of the A register. The contents of the r register are not altered unless r is the A register. Memory is locked until completion of the instruction. This instruction is useful for memory-to-memory communication between CPUs.
CrE (F4=E, F5=0, Rb→r)	Compare Register Equal. If the contents of the r register and the contents of the storage location specified by the effective address are equal, this instruction skips one location; otherwise, the next instruction is executed. The contents of the r register and storage are not altered.
CCE (F4=E, F5=2, Rb= character flag+index)	Compare Character Equal. If bits 0 through 7 of the A register and the character of the storage location specified by the sum of the effective address and bits 1 through 15 of the Rb register are equal, the instruction skips one location; otherwise the instruction executes the next instruction. Bit 0=0 of the Rb register specifies the left character (bits 8 through 15) of the storage location; bit 0=1 specifies the right character (bits 0 through 7). The contents of the A register and storage are not altered.

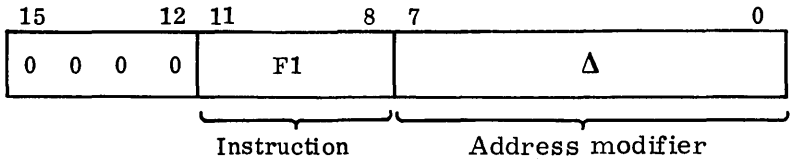
CAUTION

Each compare instruction assumes that a one-word instruction follows it.

2.2 REGISTER REFERENCE INSTRUCTIONS

Register reference instructions use the address mode field for the operation code. Register reference instructions are identified when the upper four bits (15 through 12) of an instruction are 0s.

Format:



SLS (F1 = 0) Selective Stop. Stops the computer if this instruction is executed when the selective stop switch is on. On restart, the computer executes the instruction at P + 1. This becomes a pass instruction when the selective stop switch is off.

INP (F1 = 2) Input to A. Reads one word from an external device into the A register. The word in the Q register selects the sending device. If the device sends a reply, the next instruction comes from P + 1. If the device sends a reject, the next instruction comes from P + 1 + Δ, where delta is an eight-bit signed number. If an internal reject occurs, the next instruction comes from P + Δ .

OUT (F1 = 3) Output from A. Outputs one word from the A register to an external device. The word in the Q register selects the receiving device. If the device sends a reply, the next instruction comes from P + 1. If the device sends a reject, the next instruction comes from P + 1 + Δ, where delta is an eight-bit signed number. If an internal reject occurs, the next instruction comes from P + Δ .

INA (F1 = 9) Increase A. Replaces the contents of A with the sum of the initial contents of A and delta, where delta is treated as a signed number with the sign extended into the upper eight bits. Operation on overflow is the same as for an add-to-A instruction.

ENA (F1 = A) Enter A. Replaces the contents of the A register with the eight-bit delta, sign extended.

NOP (F1 = B) No Operation. This is a pass instruction (no operation is performed).

ENQ (F1 = C) Enter Q. Replaces the contents of the Q register with the eight-bit delta, sign extended

INQ (F1 = D) Increase Q. Replaces the contents of the Q register with the sum of the initial contents of Q and delta, where delta is treated as a signed number with the sign extended into the upper eight bits. Operation on overflow is the same as for an add-to-A instruction.

The following instructions (F1 equals 4, 5, 6, 7, or E) are legal only if the program protect switch is off or if the instructions themselves are protected. If an instruction is illegal, it becomes a selective stop and an interrupt on the program protect fault is possible (if selected).

- Protect switch on — Selective stop unless the instruction is protected
- Protect switch off — Normal instruction execution (no program protection)

EIN (F1 = 4) Enable Interrupt. Activates the interrupt system after one instruction following EIN has been executed. The interrupt system must be active and the appropriate mask bit set for an interrupt to be recognized.

IIN (F1 = 5) Inhibit Interrupt. Deactivates the interrupt system. If interrupt occurs during execution of this instruction, the interrupt is not recognized until one instruction after the next EIN instruction is executed.

SPB (F1 = 6) Set Program Protect Bit. Sets the program protect bit in the address specified by the Q register.

CPB (F1 = 7) Clear Program Protect Bit. Clears the program protect bit in the address specified by the Q register.

EXI (F1 = E) Exit Interrupt State. This instruction is used to exit from any interrupt state. Delta defines the interrupt state from which the exit is taken. At the time an interrupt occurs, the value of P is stored in the interrupt trap location assigned to that particular interrupt state. This value is called the return address since it enables return to the next unexecuted instruction after interrupt processing. The EXI instruction automatically reads the address containing the return address, then jumps to the return address. In addition, if the computer is in 32K mode, this instruction also sets the OVERFLOW indicator to the state of bit 15 in the return address. This bit records the state of the OVERFLOW indicator when the interrupt occurred. In 65K mode, this instruction does not reset the OVERFLOW indicator.

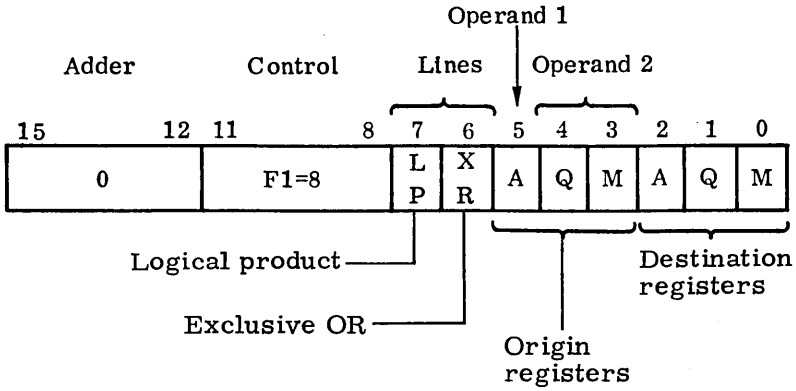
2.3 INTER- REGISTER INSTRUCTIONS

There are two types of inter-register instructions: basic (type 1) and enhanced (type 2).

2.3.1 TYPE 1 INTER-REGISTER INSTRUCTIONS

These instructions cause data from certain combinations of two origin registers to be sent through the adder to any combination of destination registers. Various operations, selected by the adder control lines, are performed on the data as it passes through the adder.

Format:



If bit 0 in an inter-register instruction is set (M is the destination register) and the instruction is not protected, it is a program protect violation and becomes a nonprotected selective stop instruction. The program protect fault bit is set and interrupt occurs.

The origin registers are considered as operands. There are two kinds.

- Operand 1 may be:
  - FFFF (bit 5 is 0)
  - The contents of A (bit 5 is 1)
- Operand 2 may be:
  - FFFF (bit 4 is 0 and bit 3 is 0)
  - The contents of M (bit 4 is 0 and bit 3 is 1)
  - The contents of Q (bit 4 is 1 and bit 3 is 0)
  - The inclusive OR, bit-by-bit, of the contents of Q and M (bit 4 is 1 and bit 3 is 1)

The following operations are possible.

- Exclusive OR (LP = 0 and XR = 1) — The data placed in the destination registers is the exclusive OR, bit-by-bit, of operands 1 and 2.
- Logical product (LP = 1 and XR = 0) — The data placed in the destination registers is the logical product, bit-by-bit, of operands 1 and 2.
- Complement logical product (LP = 1 and XR = 1) — The data placed in the destination registers is the complement of the logical product, bit-by-bit, of operands 1 and 2.

- Arithmetic sum (LP = 0 and XR = 0) — The data placed in the destination registers is the arithmetic sum of operands 1 and 2. The OVERFLOW indicator operates the same for an add-to-A instruction.

Inter-Register Mnemonics

SET (F1 = 8, bits 7 through 3 = 10000)	Set to Ones
CLR (F1 = 8, bits 7 through 3 = 01000)	Clear to Zero
TRA (F1 = 8, bits 7 through 3 = 10100)	Transfer A†
TRM (F1 = 8, bits 7 through 3 = 10001)	Transfer M †
TRQ (F1 = 8, bits 7 through 3 = 10010)	Transfer Q†
TRB (F1 = 8, bits 7 through 3 = 10011)	Transfer Q + M†
TCA (F1 = 8, bits 7 through 3 = 01100)	Transfer Complement A†
TCM (F1 = 8, bits 7 through 3 = 01001)	Transfer Complement M†
TCQ (F1 = 8, bits 7 through 3 = 01010)	Transfer Complement Q†
TCB (F1 = 8, bits 7 through 3 = 01011)	Transfer Complement Q + M†
AAM (F1 = 8, bits 7 through 3 = 00101)	Transfer Arithmetic Sum A, M
AAQ (F1 = 8, bits 7 through 3 = 00110)	Transfer Arithmetic Sum A, Q
AAB (F1 = 8, bits 7 through 3 = 00111)	Transfer Arithmetic Sum A, Q + M
EAM (F1 = 8, bits 7 through 3 = 01101)	Transfer Exclusive OR A, M
EAQ (F1 = 8, bits 7 through 3 = 01110)	Transfer Exclusive OR A, Q
EAB (F1 = 8, bits 7 through 3 = 01111)	Transfer Exclusive OR A, Q + M
LAM (F1 = 8, bits 7 through 3 = 10101)	Transfer Logical Product A, M
LAQ (F1 = 8, bits 7 through 3 = 10110)	Transfer Logical Product A, Q
LAB (F1 = 8, bits 7 through 3 = 10111)	Transfer Logical Product A, Q + M
CAM (F1 = 8, bits 7 through 3 = 11101)	Transfer Complement Logical Product A, M
CAQ (F1 = 8, bits 7 through 3 = 11110)	Transfer Complement Logical Product A, Q
CAB (F1 = 8, bits 7 through 3 = 11111)	Transfer Complement Logical Product A, Q + M

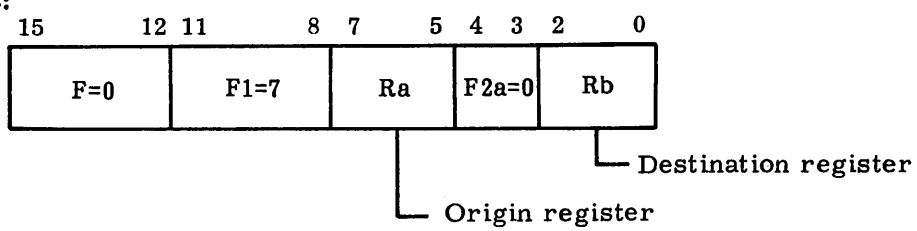
NOTE

The + implies an inclusive OR.

†The use of bit 7 is optional; it may be a 1 or a 0. The assembler uses bit 7 = 0.

2.3.2 TYPE 2 INTER-REGISTER INSTRUCTIONS

Format:



Type 2 inter-register instructions are identified when the F field is zero, the F1 field is equal to seven, and the F2a, Ra, and Rb fields are not all zero. (If these fields are all zero, the instruction is CPB).

Type 2 inter-register instructions contain three parts: an operation field (F2a) and two register fields (Ra and Rb). The F2a field determines the operation (i.e., transfer). The Ra and Rb fields specify two operands.

XFr R
(F2a=0, Ra→r, Rb→R) — Transfer Register. Transfers the contents of the r register to the R register.

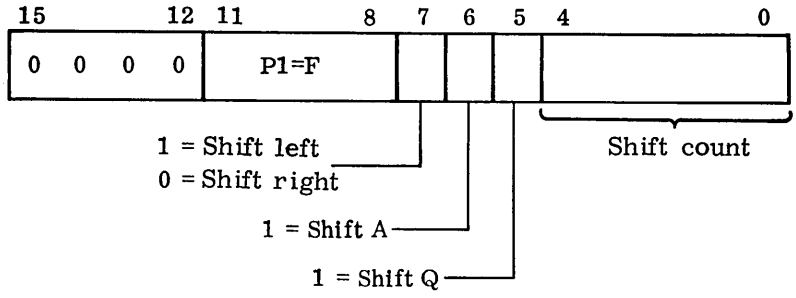
NOTE

Ra→r and Rb→R means register 1→1, 2→2, 3→3, 4→4, 5→Q, 6→A, and 7→I.

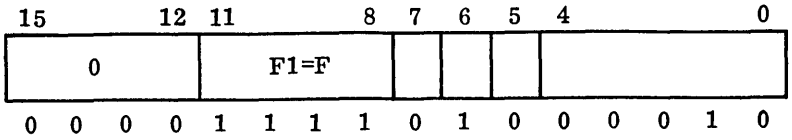
2.4 SHIFT INSTRUCTIONS

The shift instructions shift A, Q, or A/Q left or right the number of places specified by the five-bit shift count. Right shifts are end-off with sign extension in the upper bits. Left shifts are end-around. The maximum long-right or long-left shift is $1F_{16}$ places.

Format:



Example: Shift A right two places — 0F42.



Shift Mnemonics

ARS (F1 = F, bits 7 through 5 = 010)	A Right Shift
QRS (F1 = F, bits 7 through 5 = 001)	Q Right Shift
LRS (F1 = F, bits 7 through 5 = 011)	Long Right Shift (QA)
ALS (F1 = F, bits 7 through 5 = 110)	A Left Shift
QLS (F1 = F, bits 7 through 5 = 101)	Q Left Shift
LLS (F1 = F, bits 7 through 5 = 111)	Long Left Shift (QA)

2.5 SKIP INSTRUCTIONS

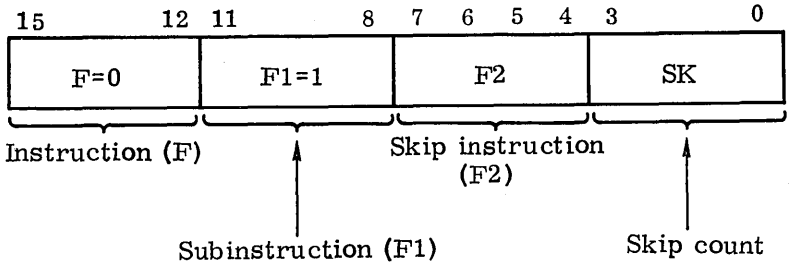
There are two types of skip instructions: basic or type 1 and enhanced or type 2.

2.5.1 TYPE 1 SKIP INSTRUCTIONS

Skip instructions result in one machine word: a 12-bit operation code and a four-bit unsigned skip count. The first four bits of the operation code field are set to zero, the next four bits contain the skip instruction code 0001, and the last four bits contain a unique identifier, F2, for each skip instruction. The expression in the address field of the instruction is evaluated modulo $2^{15}-1$.

This expression may be absolute or relocatable. If absolute, the value of the expression is the skip count. If relocatable, the value of the skip count is obtained by subtracting (16-bit one's complement arithmetic) the value of the current location counter plus one from the expression. The skip count is then placed in the last four bits of the machine word. The final value of the skip count must not exceed four bits or an error message is printed. If the expression is relocatable, the relocation type of the expression must match the relocation type of the location counter or an error results.

Format:



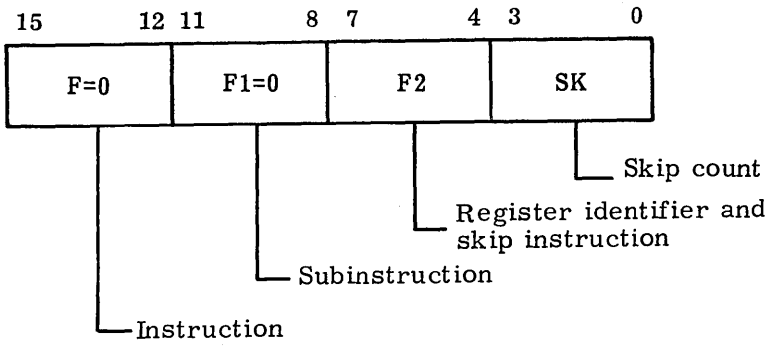
When the skip condition is met, the skip count plus one is added to P to obtain the address of the next instruction (e.g., when the skip count is zero, go to P+1). When

the skip condition is not met, the address of the next instruction is P + 1 (skip count ignored). The skip count does not have a sign bit.

SAZ (F2 = 0)	Skip if A is positive zero (all bits are 0)
SAN (F2 = 1)	Skip if A is not positive zero (not all bits are 0)
SAP (F2 = 2)	Skip if A is positive (bit 15 is 0)
SAM (F2 = 3)	Skip is A is negative (bit 15 is 1)
SQZ (F2 = 4)	Skip if Q is positive zero (all bits are 0)
SQN (F2 = 5)	Skip if Q is not positive zero (not all bits are 0)
SQP (F2 = 6)	Skip if Q is positive (bit 15 is 0)
SQM (F2 = 7)	Skip if Q is negative (bit 15 is 1)
SWS (F2 = 8)	Skip if selective skip switch is set
SWN (F2 = 9)	Skip if selective skip switch is not set
SOV (F2 = A)	Skip on overflow. This instruction skips if an overflow condition is sensed. This instruction clears the OVERFLOW indicator.
SNO (F2 = B)	Skip on no overflow. This instruction skips if an overflow condition is not present. This instruction clears the OVERFLOW indicator.
SPE (F2 = C)	Skip on storage parity error. This instruction skips if a storage parity error occurred; it clears the Storage Parity Error Interrupt signal and the PARITY FAULT indicator.
SNP (F2 = D)	Skip on no storage parity error.
SPF (F2 = E)	Skip on program protect fault. The program protect fault is set by: • A nonprotected instruction attempting to write into an address that is protected • An attempt to execute a protected instruction immediately following a nonprotected instruction, unless an interrupt caused the instruction sequence • Execution of any nonprotected instruction affecting interrupt mask or enables The program protect fault is cleared when it is sensed by the SPF instruction. The program protect fault cannot be set if the program protect system is disabled.
SNF (F2 = F)	Skip on no program protect fault.

2.5.2 TYPE 2 SKIP INSTRUCTIONS

Format:



Type 2 skip instructions are identified when the F and F1 fields are both zero and the F2 and SK fields are not both zero.

CAUTION

If these fields are both zero, the instruction is an SLS.

Type 2 skip instructions contain two parts: operation field (F2) and skip count (SK). The F2 field determines the operation (i.e., skip on register 1, 2, 3, or 4 if zero, nonzero, positive, or negative). A, Q, and I cannot be used to determine the skip condition. The skip count specifies how many locations to skip if the skip condition is met.

When the skip condition is met, the skip count plus one is added to the P register to obtain the address of the next instruction (e.g., when the skip count is one, go to P+2). When the skip condition is not met, the address of the next instruction is P+1 (skip count ignored). The skip count does not have a sign bit.

NOTE

In the following four skip instructions, F2 is a combined register designation (i.e., upper two bits are register with 0_2=4, 1_2=1, 10_2=2, 11_2=3) and instruction value (i.e., lower two bits with 0_2=skip if 1, 1_2=skip if not 0, 10_2 = skip if positive, and 11_2=skip if negative).

SrZ SK (F2=0, 4, 8, or C corresponds to r=4, 1, 2, or 3)

Skips if the r register is a positive zero (all bits are zero).

SrN SK (F2=1, 5, 9, or D corresponds to r=4, 1, 2, or 3)

Skips if the r register is not a positive zero (not all bits are zero).

SrP SK (F2=2,6,A, or E corresponds to r=4,1,2, or 3)

Skips if the r register is positive (bit 15 is zero).

SrM SK (F2=3,7,B, or F corresponds to r=4,1,2, or 3)

Skips if the r register is negative (bit 15 is a one).

2.6 DECREMENT AND REPEAT

These enhanced instructions are specified when the F field is zero, the F1 field is equal to six, bit 4 is zero, and the Ra and SK fields are both not zero.

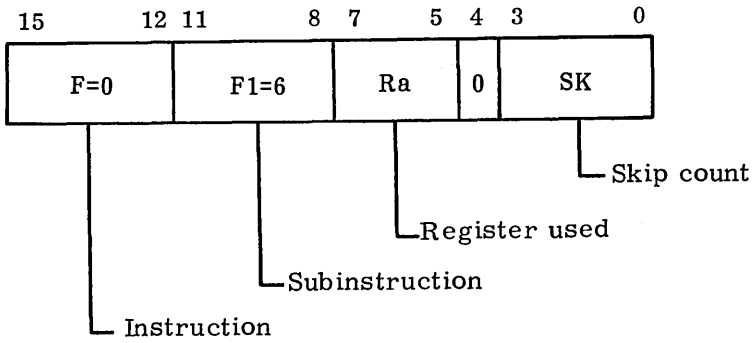
CAUTION

If both the Ra and SK fields are zero, the instruction is an SPB.

Decrement and repeat instructions contain two parts: register field (Ra) and skip count (SK). The register field specifies which register is to be decremented by one and checked for the skip condition. The skip count specifies how many locations to repeat (go backwards) if the skip condition is met.

When the skip condition is met, the skip count is subtracted from the P register to obtain the address of the next instruction (e.g., when the skip count is one, go to P-1). When the skip condition is not met, the address of the next instruction is P+1. The skip count does not have a sign bit.

Format:



Note that Ra corresponds to r in the instruction mnemonic with 1→1, 2→2, 3→3, 4→4, 5→Q, 6→A, and 7→I.

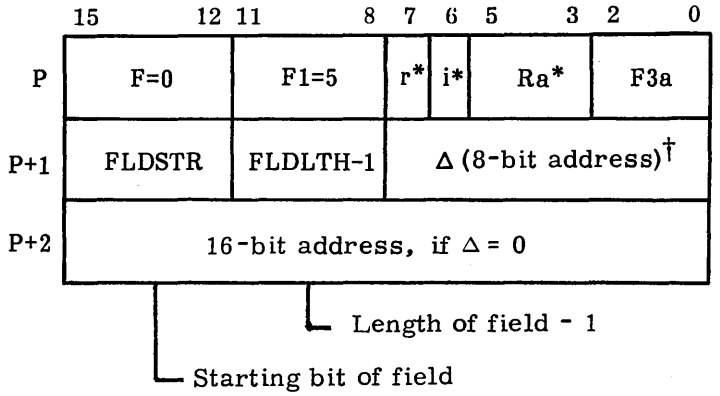
DrP SK (Ra=1,2,3,4,5,6, or 7) (r=1,2,3,4,Q,A, or I)

Decrement and Repeat if Positive. Decrements by one the contents of the r register. Operation on overflow is the same as for the ADD instruction. Repeat (go backwards) SK locations if the contents of the r register are positive (bit 15 is zero); otherwise, execute the next instruction.

2.7 FIELD REFERENCE INSTRUCTIONS

The enhanced instructions provide the ability to reference fields within a word rather than the whole word itself.

Format:



Field reference instructions are identified when the F field is zero, the F1 field is equal to five, and the r, i, Ra, and F3a fields are not all zero.

CAUTION

If r, i, Ra, and F3a fields are all zero, the instruction is an IIN.

Field reference instructions contain four parts: operation field (F3a), addressing mode fields (delta, r, i, and Ra), FLDSTR, and FLDLTH-1 fields. The F3a field determines the operation (e.g., load, store, etc.). The addressing mode fields are defined exactly as the type 2 storage reference instructions.

FLDSTR defines the starting bit of the field: FLDSTR=0 means the field starts at bit 0; FLDSTR=15 means the field starts at bit 15. FLDLTH-1 defines the length of the field minus one; FLDLTH-1=0 means the field is one bit long; FLDLTH-1=15 means the field is 16 bits long. Note that is FLDLTH-1=0, the field reference instructions become bit reference instructions.

A field starts at the bit specified by FLDSTR and includes FLDLTH contiguous bits to the right of that bit. No field may cross a word boundary (i.e., FLDSTR-FLDLTH-1 must be greater than or equal to zero); e.g., if FLDSTR=0, the field length must be one bit long (FLDLTH-1=0).

SFZ (F3a=2) — Skip if Field Zero. If the contents of the specified field of the storage location specified by the effective address are zero (all bits are zero), the instruction skips one location; otherwise, it executes the next instruction.

---

†Addressing mode

SFN (F3a=3) — Skip if Field Not Zero. If the contents of the specified field of the storage location specified by the effective address is nonzero (not all bits are zero), the instruction skips one location; otherwise, it executes the next instruction.

CAUTION

Each skip field instruction assumes that a one-word instruction follows it.

LFA (F3a=4) — Load Field. Loads the A register, right-justified, with the contents of the specified field of the storage location specified by the effective address. All other bits of the A register are cleared. The contents of storage are not altered.

SFA (F3a=5) — Store Field. Stores the contents of the field from the A register, right-justified, into the specified field of the storage location specified by the effective address. All other bits of storage are unchanged. Memory is locked until completion of the instruction. The contents of the A register are not altered.

CLF (F3a=6) — Clear Field. Clears (sets all bits to 0) the specified field of the storage location specified by the effective address. All other bits of storage are unchanged. Memory is locked until completion of the instruction.

SEF (F3a=7) — Set Field. Sets to all ones the specified field of the storage location specified by the effective address. All other bits of storage are unchanged. Memory is locked until completion of the instruction.

2.8 MISCELLANEOUS INSTRUCTIONS

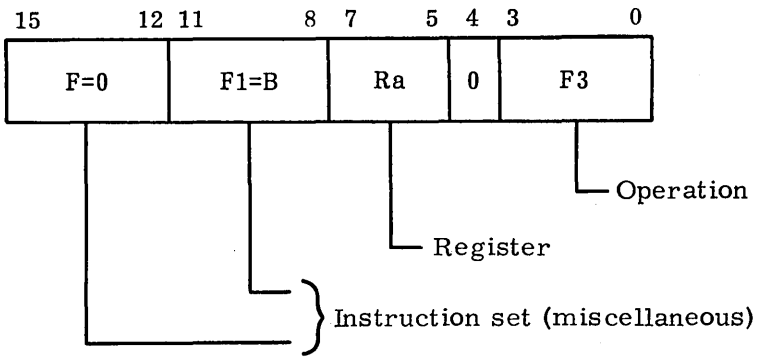
This is a set of enhanced instructions. These instructions are identified when the F field is zero, the F1 field is equal to a decimal eleven (hexadecimal B), bit 4 is zero, and the Ra and F3 fields are not both zero.

CAUTION

If both Ra and F3 are zero, the code specifies an NOP instruction.

All of the miscellaneous instructions are privileged instructions; i.e., they cannot be executed by an unprotected program and will cause a program protect violation instead.

Format:



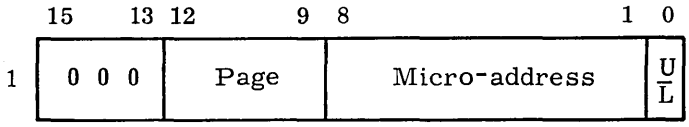
Miscellaneous instructions contain two parts: operation field (F3) and register field (Ra).

If Ra is nonzero, the F3 operation can select up to 16 miscellaneous instructions with register Ra used to specify an operand. If Ra is zero, the F3 operation field can select up to 15 more miscellaneous instructions without any explicit operand specified.

LMM
(F3=1, Ra=0)

Load Micro Memory. Loads a block of 32-bit micro-memory words into read/write micro memory from the 16-bit CYBER 18/1700 storage memory.

Initially, the Q register contains the number of 32-bit micro-memory words to be transferred. (If Q equals 0, no words are transferred.) Register 1 contains the starting address of micro memory:



a. Bits 13 through 15 must be zero.

b. Bits 9 through 12 specify the micro page.

c. Bits 1 through 8 specify the micro-memory address.

d. Bit 0 specifies the upper (0) or lower (1) micro instruction.

Register 2 contains the starting address of storage memory.

This instruction is interruptible after storing each 32-bit micro-memory word. Registers 1, 2, and Q are incremented/decremented to allow the restarting of the instruction after any interruption. Therefore, upon completion of the instruction, these registers do not contain their original values, but the following:

Q ⟵ 0
R1 ⟵ (R1)i + (Q)i
R2 ⟵ (R2)i + 2*(Q)i

Where i denotes the initial value before execution.

LRG
(F3=2, Ra=0)

Load Registers. Registers 1, 2, 3, 4, Q, A, I, and M and the OVERFLOW indicator are loaded with the contents of nine storage locations beginning at a storage location specified by the contents of the contents of the next location, P+1, as follows:

(((P+1))+1) ⟶ 1
(((P+1))+2) ⟶ 2
(((P+1))+3) ⟶ 3
(((P+1))+4) ⟶ 4
(((P+1))+5) ⟶ Q
(((P+1))+6) ⟶ A
(((P+1))+7) ⟶ I
(((P+1))+8) ⟶ M
Bit 15 of (((P+1))+9) ⟶ OVERFLOW

The contents of the storage location specified by the contents of the next location are then decremented by a decimal ten; that is,

((P+1))-$A ⟶ (P+1)

Note that any data stored in location ((P+1)) or bits 0 through 14 of location (((P+1))+9) may be extracted before the execution of the LRG instruction via the address (specified by the contents of the contents of the next location).

The contents of the nine storage locations are not altered, and the next instruction is executed at location P+2 (i.e., the LRG instruction is a two-word instruction).

SRG
(F3=3, Ra=0)

Store Registers. The contents of the storage location specified by the contents of the next location, P+1, is first incremented by a decimal ten; that is:

((P+1))+$A → (P+1)

Then, the contents of registers 1, 2, 3, 4, Q, A, I, and M and the OVERFLOW indicator are stored in nine storage locations beginning at a storage location specified by the contents of the incremented address as follows:

(1) → ((P+1))+1
(2) → ((P+1))+2
(3) → ((P+1))+3
(4) → ((P+1))+4
(Q) → ((P+1))+5
(A) → ((P+1))+6
(I) → ((P+1))+7
(M) → ((P+1))+8
(OVERFLOW) → bit 15 of ((P+1))+9

After the storing is completed, the OVERFLOW indicator is cleared.

Note that location ((P+1)) and bits 0 through 14 of location ((P+1))+9 can be used to store a program address, priority level, parameter address, or other data after the execution of the SRG instruction via the address (specified by the contents of the contents of the next location).

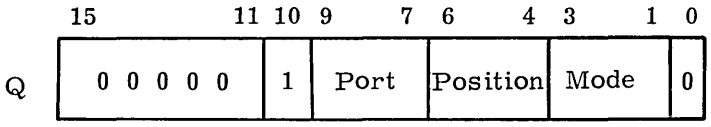
The contents of the registers are not altered, and the next instruction is executed at location P+2 (i.e., the SRG instruction is a two-word instruction).

SIO
(F3=4, Ra=0)

Set/Sample Output or Input. For output, one word from the A register is output to an external device. The word in the Q register selects the receiving device.

For input, one word from an external device is input to the A register. The word in the Q register selects the sending device.

This instruction permits tranmission to/from peripheral devices. The Q register is defined as follows:



a. Bits 11 through 15 and bit 0 must be zero.

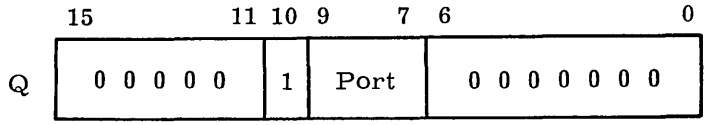
b. Bits 7 through 10 contain the port number of the device with bit 10 always a one. Port numbers are analogous to the A/Q I/O equipment numbers and thus cannot conflict with them.

c. Bits 4 through 6 contain the device's position on the port. These bits may also be mode bits if some or all of the position bits are not required.

d. Bits 1 through 3 contain the mode in which the device is to operate. Bit 3 is always the set/sample condition bit: if it is 1, one data word is output; if zero, one data word is input.

SPS (F3=5, Ra=0)

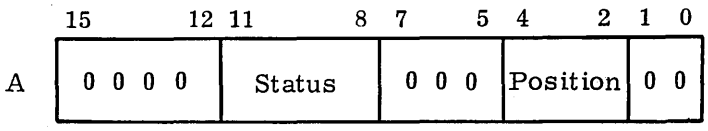
Sample Position/Status. Inputs to the A register the position and the status of an I/O device that has caused a macro interrupt. The word in the Q register selects the device. This instruction also provides for clearing the macro interrupt generated by the I/O controller. The Q register is defined as follows:



a. Bits 0 through 6 and 11 through 15 must be zero.

b. Bits 7 through 10 contain the port number of the I/O device with bit 10 always a one. Port numbers are analogous to the A/Q I/O equipment numbers and thus cannot conflict with them.

Upon completion of the instruction, the A register contains the following:



a. Bits 0 through 1, 5 through 7, and 12 through 15 are zero.

b. Bits 2 through 4 contain the device's position.

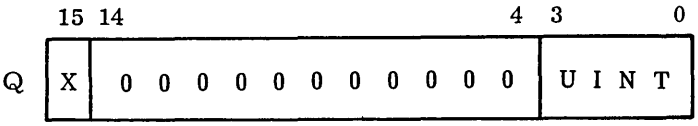
c. Bits 8 through 11 contain the least significant four bits of the data input lines. If these four bits of status information are insufficient, the device's controller may provide for the transfer of additional status bits using the SIO instruction.

DMI (F3=6, Ra=0)

Define Micro Interrupt. Defines the use of one of the 12 available micro interrupts. (The use of micro interrupts 12 through 15 is restricted to internal use.)

This instruction allows enabling/disabling of a micro interrupt and defining it for auto-data transfer (ADT) or for a special usage.

The Q register selects and enables/disables the micro interrupt and is defined as follows:

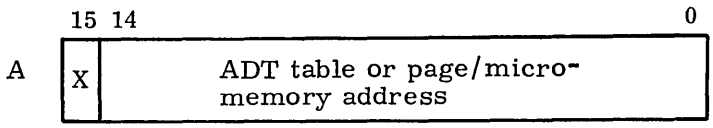


a. Bit 15 enables or disables the micro interrupt: If bit 15 is a one, the micro interrupt is enabled; if it is zero, the micro interrupt is disabled. (If the micro interrupt is disabled, the A register is not utilized.) Initially, following a master clear, all 12 micro interrupts are disabled.

b. Bits 4 through 14 must be zero.

c. Bits 0 through 3 contain the micro interrupt number (0 through 13). Note that 12 through 15 are not available for use, and if used, the instruction is treated as a non-operation.

The A register defines the micro interrupt for auto-data transfer or for a special usage and is defined as follows:



a. If bit 15 is a zero, auto-data transfer is selected and bits 0 through 14 contain the auto-data transfer table address. The address must be within the lower 32K of main memory.

There are four possible types of auto-data transfer tables for a particular micro interrupt.

1. Single A/Q device
2. Multi-A/Q devices
3. Clock device
4. Single or multi-devices

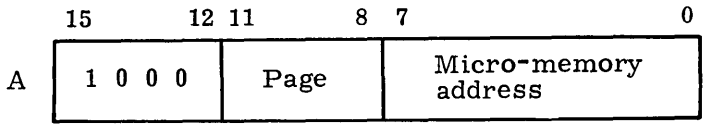
b. If bit 15 is a one, the special usage is selected and a jump is made to the upper micro instruction of the page/micro memory in bits 0 through 14. It is assumed that a section of micro memory has been previously loaded and that it must process the micro interrupt properly and return control to the current macro instruction address (P) by jumping to the lower micro instruction of micro-memory address $3E_{16}$ in micro-page zero.

Registers P, A, and Q and all of file 2 must not be altered, and return must be within 12.5 microseconds.

CAUTION

The micro function, SUB-, must not be used.

Bits 0 through 15 of the A register are defined as follows for this special usage:



a. Bit 15 must be one.

b. Bits 12 through 14 must be zero.

c. Bits 8 through 11 specify the micro page.

d. Bits 0 through 7 specify the micro-memory address.

CAUTION

Extreme caution should be exercised in the utilization of this option, since it provides an escape from CYBER 18/1700 emulation.

CBP (F3=7, Ra=0)

Clear Breakpoint Interrupt. Clears the macro breakpoint interrupt. This interrupt occurs when the following conditions are true:

a. The macro breakpoint (reference and/or storage) is externally selected.

b. The macro breakpoint interrupt option is externally selected.

c. The CPU recognizes a breakpoint condition and generates a macro breakpoint interrupt.

The macro programmer than has the responsibility to clear (CBP instruction) and process the interrupt.

GPE (F3=8, Ra=0)

Generate Character Parity Even. Sets or clears bit 7 of the A register to cause the parity of bits 0 through 7 to be even. The rest of the contents of the A register are not altered.

GPO (F3=9, Ra=0)

Generate Character Parity Odd. Sets or clears bit 7 of the A register to cause the parity of bits 0 through 7 to be odd. The rest of the contents of the A register are not altered.

ASC (F3=A, Ra=0)

Scale Accumulator. The A register is shifted left (end-around) until bits 14 and 15 of the A register are different. Upon completion of the instruction, register 1 contains the number of places that the A register was shifted. (This number may range from zero through 14_{10}, inclusive.) If the A register is plus zero (0000_{16} or $FFFF_{16}$), no shifting is done and register 1 contains minus zero ($FFFF_{16}$).

LUB R (F3=0, Ra→R)

Load Upper Unprotected Bounds. Loads the upper unprotected bounds register from the contents of the R register.

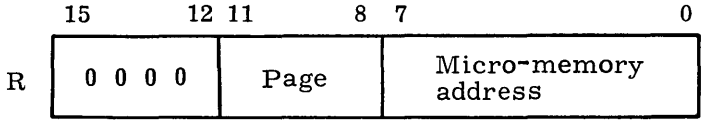
LLB R (F3=1, Ra→R)

Load Lower Unprotected Bounds. Loads the lower unprotected bounds register from the contents of the R register.

EMS R (F3=2, Ra→R)

Execute Micro Sequence. Transfers machine control to the upper micro instruction of the page/micro-memory address in bits 0 through 15 of the R register. It is assumed that a section of micro memory has been previously loaded and that it should return control to the next macro-instruction address (P+1) by jumping to the lower micro instruction of micro-memory address $3E_{16}$ in micro-page zero.

Registers P, A, and Q and all of file 2 should not be altered, and return must be within 12.5 microseconds (or the micro sequence must be interruptable). Bits 0 through 15 of the R register are defined as follows:



a. Bits 12 through 15 must be zero.

b. Bits 8 through 11 specify the micro page.

c. Bits 0 through 7 specify the micro-memory address.

CAUTION

Extreme caution should be exercised in the utilization of this instruction, since it provides an escape from CYBER 18/1700 emulation.

2.9 NEGATIVE ZERO/OVERFLOW SET

Negative zero and/or overflow set can be caused by two characteristics of the computer:

- The computer has a one's complement subtractive adder.
- Multiplication and division are done with positive numbers only. Therefore, a sign correction occurs, if required, before and after the multiplication or division symbols.

Arithmetic operations that produce a negative zero result and/or set overflow in the computer are:

- Addition: $(-0) + (-0) = (-0)$
- Subtraction: $(-0) - (+0) = (-0)$
- Multiplication:
  $(+0) \times (-N) = (-0)$
  $(-N) \times (+0) = (-0)$
  $(-0) \times (+N) = (-0)$
  $(+N) \times (-0) = (-0)$
- Division

  Where: $N \neq 0$
  R = Remainder

  $\frac{(+0)}{(-N)} = (-0),\ R = (+0)$

  $\frac{(-0)}{(+N)} = (-0),\ R = (-0)$

  $\frac{(-0)}{(-N)} = (+0),\ R = (-0)$

  $\frac{(+N)}{(+0)} = (-0),\ R = (+N)$ overflow set

  $\frac{(-N)}{(-0)} = (-0),\ R = (-N)$ overflow set

  $\frac{(-2N)}{(+N)} = (-2),\ R = (-0)$

$\frac{(-2N)}{(-N)} = (+2),\ R = (-0)$

$\frac{(+0)}{(+0)} = (-0),\ R = (+0)$ overflow set

$\frac{(+0)}{(-0)} = (+0),\ R = (+0)$ overflow set

$\frac{(-0)}{(+0)} = (+0),\ R = (-0)$ overflow set

$\frac{(-0)}{(-0)} = (-0),\ R = (-0)$ overflow set

PSEUDO INSTRUCTIONS 3

Pseudo instructions control the assembler, provide subprogram linkage, control output listing, reserve storage, convert data, and so on.

Pseudo instructions may be placed anywhere in a source language subprogram. However, OPT or NAM must be the first statement of a subprogram and MON or END must be the last statement.

3.1 SUBPROGRAM LINKAGE

These instructions identify and link subprograms; a symbolic name in the location field is ignored.

3.1.1 NAM

NAM identifies a source language subprogram and must be the first statement of the subprogram. Only the assembler control pseudo instruction OPT (section 3.4.5) may precede it.

The format is

```
NAM   s   C
```

s An optional symbolic name of the subprogram which is printed as part of the assembly list output.

C A comment starting in column 27 up through column 72 (up to 46 characters may be printed) printed by the loader when the program is loaded and also printed by some other programs.

3.1.2 END

END must be the last statement of a source language subprogram. If END terminates a subprogram assembled separately or the last subprogram of a group, MON follows END. Otherwise END is followed by NAM or OPT.

The format is

```
END   s
```

s An optional symbolic name of an entry point to the first subprogram to be executed. If specified, s must be defined as an entry point in the subprogram to which control passes. This entry point may be in the same subprogram as the END statement or in a subprogram loaded at the same time.

Example:

```
END   START
```

START is the location of the first statement to be executed.

3.1.3 ENT

The ENT instruction lists the symbolic names of entry points which may be referenced from other subprograms.

The format is

```
ENT    s1,s2,...,sn
```

s_i Entry points listed in the address field of ENT and must be defined in the subprogram containing the ENT instruction. s_i must not refer to to a location outside the subprogram, common storage, or data storage.

Example:

```
        NAM    PROG1
        ENT    ENT1,ENT2      (Legal)
ENT1    LDA    XYZ1
ENT2    STA    XYZ2
         :       :
        ENT    ENTX           (Illegal; ENTX not defined)
         :       :
        END    ENT1
```

3.1.4 EXT/EXT*

The EXT instruction lists the symbolic names of entry points in external subprograms which may be referenced from this subprogram.

The format is

```
EXT    s1,s2,...,sn
```

s_i Entry points in the address field of EXT, which must be symbols defined in the subprograms they reference. s_i must not refer to symbols in the same subprogram.

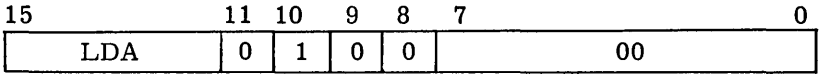
Example 1:

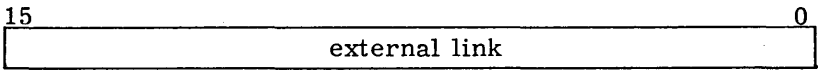
```
        NAM
        EXT    ENT1,ENT2      (Legal)
ENT3    LDA    XYZ
        COM    ENT5
        EXT    ENT3           (Illegal; ENT3 is same subprogram)
        EXT    ENT4           (Legal)
        EXT    ENT5           (Illegal; ENT5 in common storage)
        EXT    ENT1           (Legal;defined in same way as above)
         .       .
         .       .
         .       .
        END
```

Example 2:

```
EXT     ENT1,ENT2
 .
 .
 .
LDA     ENT1
```

This reference to ENT1 results in the following two machine words.





External link is a pointer to the location of ENT1 used by the loader at load time.

The EXT* instruction is the same as EXT except that s_i are absolute locations in EXT and references to s_i are made relative in EXT*.

The format is

```
EXT*    s1,s2,...,sn
```

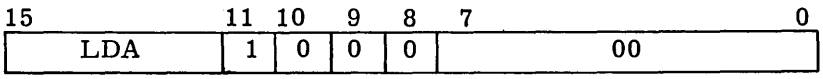
The plus terminator cannot be used with an operation code when the address references a relative external entry point. It is also illegal to enclose an external in parentheses in the address field of an ADC instruction (section 3.3.1).

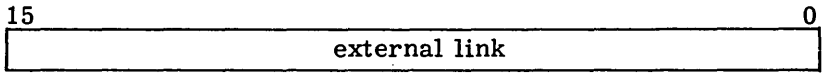
Examples:

```
 .
 .
 .
EXT*    NAME1,NAME2,NAME3
LDA     NAME1
LDA+    NAME1                   (Illegal)
 .
 .
LDA     (NAME2)
ADC     (NAME3)                 (Illegal)

EXT*    NAME1,NAME2
 .
 .
LDA     NAME1
```

This reference to NAME1 results in the following two machine words.





External link is a pointer to the location of NAME1 used by the loader at load time.

3.2 DATA STORAGE

The following instructions allocate data storage. BSS and BZS assign storage local to the subprogram in which they appear. COM and DAT assign data common to any number of subprograms. Symbolic names in the location fields of data storage instructions are ignored.

3.2.1 BSS

The BSS instruction assigns symbolic names to segments of storage within the instruction sequence of the subprogram.

The format is

BSS $s_1(e_1), s_2(e_2), \ldots, s_n(e_n)$

s_i	name	Symbolic name which defines the first location of the named segment.
	omitted	When omitted from a subfield, a segment is assigned with the length e but no name is assigned to the segment.
e_i	expression	Corresponding expressions of the symbolic name which defines the length of the segment in words. Segments are assigned contiguously to form one block of data starting at location s_1. The size of the block is equal to the sum of the sizes of the segments. e_i are evaluated modulo $2^{15}-1$ and must be absolute.
	0	The associated symbolic name is assigned to the next segment which in effect assigns two names to that segment.
	omitted	The length is assumed to be one computer word.
	symbolic name	Must be previously defined; can be assigned by an EQU instruction.

3.2.2 BZS

This statement functions in the same way as the BSS, except that the specified storage locations are set to zero.

The format is

BZS $s_1(e_1), s_2(e_2), \ldots, s_n(e_n)$

Example:

```
        NAM
NAM3    LDA     XYZ1
        BSS     NAM4(3)             (Assign 3 words to NAM4)
        BZS     NAM5(5)             (Assign 5 words, set to zero,
                                    to NAM5)
        BSS     NAM1,NAM2(9)        (Assign 1 word to NAM1; assign
                                    9 words to NAM2)
        BSS     NAM3                (Illegal; NAM3 already assigned)
        BSS     NAM6,(4)            (Assign 1 word to NAM6, assign
                                    4 words to unnamed segment)
        BSS     NAM7                (Assign 1 word to NAM7)
        EQU     NAM8(4),NAM9(2)

        BZS     NAM10(NAM8-NAM9)    (Assign 2 words, set to zero, to
                                    NAM10)
        BSS     NAM8(NAM10-1)       (Illegal; NAM8 already assigned)
        BSS     LOC1(0),LOC2        (Assign the same word to LOC1 and
                                    LOC2)
         .
         .
         .
        END
```

3.2.3 COM

The COM instruction names and defines segments in a block of storage common to more than one subprogram.

The format is

COM $s_1(e_1), s_2(e_2), \ldots, s_n(e_n)$

s_i	name	Symbolic name which defines the first location of the named segment.
	omitted	When omitted from a subfield, a segment is assigned with the length e but no name is assigned to the segment.
e_i	expression	Corresponding expressions of the symbolic name which defines the length of the segment in words. Segments are assigned contiguously to form one block of data starting at location s_1. The size of the block is equal to the sum of the sizes of the segments. e_i are evaluated modulo $2^{15}-1$ and must be absolute.
	0	The associated symbolic name is assigned to the next segment which in effect assigns two names to that segment.
	omitted	The length is assumed to be one computer word.
	symbolic name	Must be previously defined; can be assigned by an EQU instruction.

If a program includes more than one COM statement, they define consecutive segments of common storage in the order of their appearance. The area used by common storage is assigned by the loader at load time to locations outside the program area. Data in common storage cannot be preset by the ORG pseudo instruction.

Example:

```
        NAM
        COM     NAM4
NAM3    STA     XYZ1
        COM     NAM7($1EF),NAM8
        EQU     NAM1(6),NAM2(2)
        COM     NAM5(NAM1-NAM2)
        COM     NAM6(NAM3)              (Illegal)
          .
          .
          .
        END
```

3.2.4 DAT

The DAT instruction reserves area for common storage which is assigned within the program area and may be preset with data or instructions by using the ORG pseudo instruction (section 3.4.2).

The format is

DAT $s_1(e_1),s_2(e_2),\ldots,s_n(e_n)$

s_i	name	Symbolic name which defines the first location of the named segment.
	omitted	When omitted from a subfield, a segment is assigned with the length e but no name is assigned to the segment.
e_i	expression	Corresponding expressions of the symbolic name which define the length of the segment in words. Segments are assigned contiguously to form one block of data starting at location s_1. The size of the block is equal to the sum of the sizes of the segments. e_i are evaluated modulo $2^{15}-1$ and must be absolute.
	0	The associated symbolic name is assigned to the next segment which in effect assigns two names to that segment.
	omitted	The length is assumed to be one computer word.
	symbolic name	Must be previously defined; can be assigned by an EQU instruction.

3.3 CONSTANT DECLARATIONS

These pseudo instructions introduce constant values into the instruction sequence.

3.3.1 ADC/ADC*

The ADC/ADC* instruction evaluates numerical constants or address expressions and inserts the results in line. When ADC is followed by an asterisk, the evaluated address expressions are made relative to the current location counter. The relocation type of the expression must be the same as that of the location counter. The value of the locations counter is subtracted from the value of the evaluated expression (16-bit one's complement arithmetic) and the result is the 16-bit address constant.

The format is

s ADC $e_1, e_2, (e_3), \ldots, e_n$

s — Symbolic name in the location field which is assigned to the first constant in the address field.

e_i — Numerical constant or address expression to be evaluated. The result is evaluated modulo $2^{15}-1$. Bit 15 is set if the expression is enclosed in parentheses. The results corresponding to $e_1, e_2, \ldots, e_n$ are stored in consecutive storage locations.

Note: Indirect addressing cannot be specified in the ADC* statement.

3.3.2 ALF

The ALF instruction translates a message into ASCII format.

The format is

s ALF n, message

s — Symbolic name in the location field which is assigned to the first constant in the address field.

n — Unsigned integer specifying the number of words to be stored; 2n equals the number of characters. Excess characters are treated as a remark. (The ALF statement, including the message, will not be processed beyond the 72nd character of the source image.) If the message is less than 2n characters, the unused portion of the specified area is blank filled.

Noninteger character which signals the end of the message. When n is a special terminating character, the storage of the message terminates the first time this character is encountered in the message if it occurs before the 72nd character. If the character just prior to n is the first character of a word, a blank is placed in the second character to complete the word.

A character message is stored into consecutive locations in the instruction sequence. The message is converted to ASCII characters (Appendix C) and stored two 8-bit characters per word.

The following typewriter control characters may be input with the ALF statement.

Code	Meaning	Hexadecimal Value
:R	Carriage return	D
:T	Horizontal tab	9
:L	Line feed	A
:B	Bell	7
:F	Top of form	C
:V	Vertical tab	B

These codes are converted to a single output character with the corresponding hexadecimal value and are counted as one character in determining the value of n, when n is an integer character count. A colon is an 8 to 5 key punch code with the ASCII value of $3A_{16}$.

A symbolic name in the location field is assigned to the first word to the message.

Example:

The following source language statements

```
        ALF     4,EXAMPLE1
NAM1    ALF     .,EXAMPLE2
        ALF     6,EXMP3:TEXMP4:R
NAM2    ALF     4,EXMP5
```

are translated into machine words.

Location	Character Left	Right
	E	X
	A	M
	P	L
	E	1
NAM1	E	X
	A	M
	P	L
	E	2
	Δ	Δ
	⋮	⋮
	Δ	Δ
	E	X
	M	P
	3	tab
	E	X
	M	P
	4	carriage return
NAM2	E	X
	M	P
	5	Δ
	Δ	Δ

In this example Δ is a blank. Three dots indicate blanks fill in the words between EXAMPLE2 and EXMP3. This is because the special terminating character, ., does not occur in the message before the 72nd character. If, in the example, n is in column 13, then 25 words of blanks are used to fill the words between EXAMPLE2 and EXMP3.

3.3.3 NUM

The NUM instruction defines numeric constants.

The format is

s NUM $k_1, k_2, \ldots k_n$

s	Symbolic name in the location which is assigned to the first constant in the address field.
k_i	Specified integer constants stored into consecutive locations in the instruction sequence. Each constant may be a decimal integer within the range ±32767, or a hexadecimal integer preceded by a $ within the range ±7FFF. The constant may be signed; if it is not signed, the constant is assumed to be positive. When the sign is minus, the one's complement of the number is used.

Examples:

The following source language statements

```
        NUM     1,2,3,$A
NAM1    NUM     +14,-10,-$13B,$7FF
```

are translated into machine words.

Location	Contents	Location	Contents
	0001	NAM1	000E
	0002		FFF5
	0003		FEC4
	000A		07FF

3.3.4 DEC

The DEC instruction converts decimal constants into fixed-point binary.

The format is

s DEC $k_1, k_2, \ldots, k_n$

s — Symbolic name in the location which is assigned to the first constant in the address field.

k_i — Specified integer constants stored into consecutive locations in the instruction sequence. It is a signed decimal integer followed by a decimal and/or binary scaling factor. The decimal scaling factor consists of the letter D followed by a signed or unsigned decimal integer. The binary scaling factor is the letter B followed by one or two signed or unsigned decimal digits. The form of a constant in the address field may be

fDdBb

which is equivalent to the algebraic expression

$f \cdot 10^d \cdot 2^b$

The fixed-point binary number resulting from the conversion must have a magnitude less than 2^{15}. If the result of scaling is greater than $2^{15}-1$, an error diagnostic is printed.

A symbolic name in the location field is assigned to the location of the first constant.

Example:

The source language statements

```
        DEC     35D-1B6
NAM1    DEC     -35B6
        DEC     32760B-4
NAM2    DEC     32761D-5B15,+625D-2B3
NAM3    DEC     10D3
```

are converted to machine words.

Location	Contents of Bits 15 through 0
	0000000011100000
NAM1	1111011100111111
	0000011111111111
NAM2	0010100111101111
	0000000000110010
NAM3	0010011100010000

3.3.5 VFD

The VFD (variable field definition) instruction assigns data to consecutive locations in the instruction sequence without regard for computer words. Data is stored in bit strings rather than word units; it may be numeric constants, ASCII characters, or expressions. A symbolic name in the location field is assigned to the first word of data.

The format is

s VFD $m_1n_1/v_1, m_2n_2/v_2, \ldots, m_nn_n/v_n$

s name — Symbolic name which defines the first location of the named segment.

m_i — Specifies the mode of the data.

N — When the value of the data is a numeric constant, the mode is specified as N and the number of bits must not be greater than 16. If n is larger than necessary, the value is right justified in the field and the sign extended in the remaining high order bits. If n is less than is required, the value is truncated and the least significant bits are stored. The value, v, is a decimal integer or a hexadecimal integer preceded by a dollar sign. Integers may be signed or unsigned; if the sign is omitted, the number is assumed to be positive. A decimal number must be within the range ±32767 and a hexadecimal integer within the range ±7FFF.

A — When v is string of characters, m must be A and n must be a multiple of 8. The number of characters in the string should be equal to n/8 including embedded blanks. The last character must be followed by a blank or a comma. The characters are converted to ASCII code and stored as in the ALF instruction (section 3.3.2).

X — When v is an expression, m must be X and n must be less than or equal to 16. If n is less than 16, the final value of the expression may be relocatable or absolute. It is evaluated modulo $2^{15}-1 = 7FFF_{16}$. If the final value is absolute and n exceeds the size required, the value is right justified in the field. If absolute and n is less than the required size, the value is truncated and the least significant bits are stored in the field. If the final value is relocatable, n must equal 15 and the expression must be positioned so that it will be stored right justified at bit position 0 of the computer word.

If n equals 16, the expression must be absolute; it is evaluated using 16-bit one's complement arithmetic. If a symbol is used in a 16-bit expression, bit 14 of the value of the symbol is extended to bit 15 and therefore the calculation of the value of the symbol is accurate only to $2^{14}-1$. For example, if the symbol A is equated to the value -1, the value of A in the symbol table is $7FFE_{16}$ but the value used in the 16-bit calculation of this symbol is $FFFE_{16}$. Numeric operands used in a 16-bit expression may be 16 bits in magnitude.

n_i Number of bits to be allocated

v_i Value of the data

Examples:

1. Source language statements

```
NAM
VFD     N3/1,X5/6-4,A16/XY,X4/NAM1-NAM2
BSS     NAM2(3),NAM1
  :
  :
END
```

result in machine words

Word	Contents

1

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	0	0	0	1	0	0	1	0	1	1	0	0	0

2

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	1	0	0	1	0	0	1	1	0	0	0	0

2. Source language statements

```
NAM
VFD     N8/-1,A8/L,N1/0,X15/NAM1
BSS     NAM1
  :
  :
END
```

result in machine words

Word	Contents

1

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	1	1	1	0	0	1	0	0	1	1	0	0

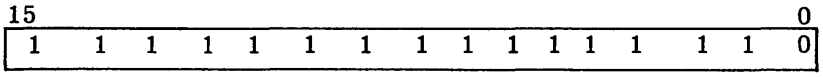
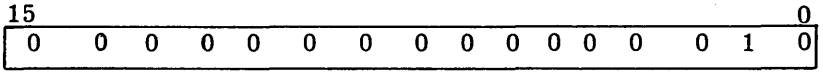
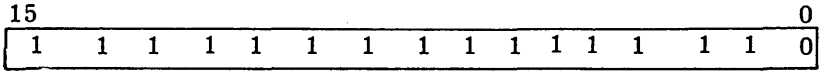
2

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	loc of NAM1														

3. Source language statements

```
NAM
EQU     A(-1),B(2)
VFD     X16/A,X16/B,X16/$7FFF*2
 .
 .
END
```

result in machine words

Word	Contents
1	
2	
3	

3.4 ASSEMBLER CONTROL

The assembly process is controlled or modified by these pseudo instructions. A symbolic name in the location field is ignored except where specifically noted.

3.4.1 EQU

The EQU instruction equates each symbolic name to the expression value.

The format is

EQU $s_1(e_1), s_2(e_2), \ldots, s_n(e_n)$

s_i	name	Symbolic name s_i is equated to the value e_i.
e_i	expression	Any symbolic operand used in the expression must be previously defined and not external to the subprogram in which the EQU statement appears. e_i are evaluated modulo 2^{15}-1 and must be absolute.
	omitted	An expression error is generated.

Example:

```
          NAM     EXAMPL
PICKUP    LDA     XYZ1
NAM6      ADD     XYZ2
          EQU     NAM3($4F),NAM4(-39)
          EQU     NAM7(NAM6-1)
          EQU     NAM8(STORE)              (Illegal)
STORE     STA     XYZ3
          EQU     NAM9(STORE)              (Legal)
            .
            .
            .
          END
```

3.4.2 ORG/ORG*

The ORG statement specifies an address expression to which the current location counter is set.

The format is

```
ORG    e
```

e	expression	The expression, e, is evaluated modulo $2^{15}-1$ and the location counter is set to the resultant value. The value of the expression may be program or data relocatable, or absolute; if relocatable, it must be positive. Any symbolic operands in the expression must have been previously defined.

The instructions following an ORG statement are assembled into consecutive locations beginning at the location of the evaluated address expression, e. This sequence may be changed by another ORG, or terminated by an ORG* statement. Within the range of a data relocatable ORG any reference to an external symbol is illegal.

```
ORG*
```

This instruction is used to return to the normal instruction sequence previously interrupted by an ORG. More than one ORG may be specified without an intervening ORG*; however, when an ORG* does occur, the location counter is reset to the value it had prior to the first ORG.

Example:

```
          .
          .
          BSS     ORG1(10),ORG2,ORG3(5)
NAM1      ENA     0
          .
          .
          .
NAM2      JMP*    NAM3
          ORG     NAM1
          (sequence of code beginning at NAM1)
          ORG*
          (resume sequence of code at NAM2+1)
NAM3      JMP*    NAM4
          ORG     ORG1
          (sequence of code beginning at ORG1)
          ORG     ORG3
          (sequence of code beginning at ORG3)
          ORG*
          (resume sequence of code at NAM3+1)
          .
          .
          END
```

3.4.3 IFA

The IFA instruction assembles a set of coding lines only if a specified condition is true.

The format is

s IFA e_1, c, e_2

s — The symbolic name in the location field is used as an identifying tag only; it is not defined as a location symbol within the program. If specified, the first 2 characters of the identifier, s, must match the first 2 characters of the symbolic name in the address field of the corresponding EIF. If s is blank in an IFA statement, it must also be blank in the corresponding EIF.

e_i — The expressions e_1 and e_2 are evaluated modulo $2^{15}-1$ and must result in an absolute value. Any symbolic name in either expression must have been previously defined.

c — If the conditions specified by c exist between e_1 and e_2, the code is assembled; if the condition does not exist, the code following the IFA statement is skipped until a corresponding EIF statement is encountered.

The following conditions may be specified by c.

Condition	Meaning
EQ	$e_1 = e_2$
NE	$e_1 \neq e_2$
GT	$e_1 > e_2$
LT	$e_1 < e_2$

3.4.4 EIF

The EIF instruction signals the termination of an IFA or IFC instruction (section 4.1.4) when coding lines are skipped as a result of an untrue condition. When the condition in the IFA or IFC is true, EIF is ignored.

The format is

```
EIF     s
```

s The symbolic name, s, in the address field establishes the correspondence between an IFA or IFC and an EIF instruction. The first 2 characters of s must be the same as the first 2 characters in the location field of the corresponding IFA or IFC. An EIF with a blank address field terminates an unlabeled IFA or IFC.

Example:

```
          NAM
            .
            .
LOC1      BSS     A(20),B(10),C(2)
          EQU     NAM1(10),NAM4(B),NAM2(2)
NAM3      IFA     NAM1,EQ,NAM2+8
OP1       SAZ     1
          EIF     NAM3
          IFA     NAM1,GT,NAM2+8
OP2       SAZ     2
          EIF
            .
            .
            .
          END
```

OP1 is assembled and OP2 is skipped if the value of NAM1 equals the value of NAM2+8; OP1 is skipped and OP2 is assembled if the value of NAM1 is greater than the value of NAM2+8; both OP1 and OP2 are skipped if the value of NAM1 is less than the value of NAM2+8.

3.4.5 OPT

The OPT pseudo instruction signals the input of control options to the assembler.

The format is

```
OPT    P
```

where P is one or more of the control options listed below.

When only OPT appears, the assembler requests input of control options from the comment device by typing:

```
OPTIONS
```

The following control options are entered in any order on the teletypewriter. Imbedded spaces and illegal characters are ignored. A carriage return signals the end of control options input.

Option	Meaning
L	List output on standard list device
P	Punch output on standard punch device
X	Place object output on mass storage device (scratch area)
M	List called macro skeletons
A	Abandon all remaining assemblies and return control to operating system
Ilu	Input from unit lu. Reads instructions until the END statement is encountered, then returns to the standard input device; lu may be any ASCII or BCD input device.
C	List cross references at end of assembly listing.

OPT is not a part of the source language program. It is used strictly for control of the assembler and has no code associated with it.

OPT may precede any NAM instruction in any subprogram. If the first statement encountered is not OPT, standard options are assumed until END is encountered. If OPT is encountered between the first statement of a program and the END statement, a diagnostic is issued. The standard options are L, P, X, and C. Chapter 5 describes output resulting from the standard options.

3.4.6 MON

The MON instruction returns control to the operating system after the last subprogram has been assembled.

The format is

```
MON
```

MON may be used only after the END statement. The location and address fields are ignored. This statement is part of the source language program and is used strictly for control of the assembler; no code is associated with it.

3.5 LISTING CONTROL

The following pseudo instructions control the printing of assembly output. The location and address fields are ignored unless specified.

3.5.1 NLS

The NLS instruction inhibits list output.

The format is

```
NLS
```

Normally list output is enabled initially until an NLS occurs and then remains inhibited until an LST instruction or the end of the program occurs.

3.5.2 LST

The LST instruction initiates list output after an NLS has inhibited it.

The format is

```
LST
```

3.5.3 SPC

The SPC instruction controls line spacing on the list output unit.

The format is

```
SPC    e
```

e Number of lines to be skipped; the expression is evaluated modulo $2^{15}-1$ and must be absolute. After the expression is evaluated, if space count exceeds 60 lines, one line is skipped.

3.5.4 EJT

The EJT instruction causes page ejection during printing of the list output.

The format is

```
EJT
```

MACROS 4

An often used set of instructions may be grouped together to form a macro. Once a macro is defined, it may be used as a pseudo instruction. The macro assembler includes two types of macros.

Programmer defined	Macros which must be declared by MAC pseudo instructions immediately following the NAM image. Comment cards may, however, be placed anywhere in the macro definition.
Library	Definitions contained on the system library and may be called from any subprogram.

4.1 MACRO PSEUDO INSTRUCTIONS

These pseudo instructions are used only within a macro definition.

4.1.1 MAC

The MAC instruction is required and names a macro and lists its formal parameters. The location field contains the name used to call the defined macro. It may be any name which is not a machine or pseudo instruction. It is not necessary that all parameters be used within the macro.

The format is

s MAC $P_1, P_2, \ldots, P_n$

s — A symbolic name in the location field is assigned to the first word of the generated code.

p_i — Symbolic names that are local to the macro definition and may be used anywhere else in the program without ambiguity. The formal parameters must conform to the following rules.

They must be symbolic names of 1 or 2 characters.

The parameter list must not extend beyond the 72nd character of the line containing MAC.

The parameter list must terminate with a blank or the 72nd character of the line.

Each parameter in the list is separated from the next by a comma.

4.1.2 EMC

The EMC instruction is required and signals the end of a macro definition. A symbolic name in the location or address field is ignored. EMC is always the last instruction in a macro definition.

The format is

```
EMC
```

4.1.3 LOC

The LOC instruction is optional and allows the use of the same symbols in macros and programs to avoid doubly defined symbols. Symbols, other than formal parameters, that are local to the macro being defined are listed in this instruction. Local symbols have meaning only in the macro in which they are listed by LOC, thus allowing the same symbols to be used elsewhere in the program without ambiguity.

The LOC instruction must immediately follow the MAC instruction. A symbol in the location field of the LOC instruction is ignored.

The format is

LOC $s_1, s_2, \ldots, s_n$

s_i Local symbols in the address field which must conform to the following rules.

They must be symbolic names of 1 or 2 characters.

The list cannot extend beyond the 72nd character of the line containing the LOC instruction.

The list terminates with a blank or the 72nd character of the line.

Each symbol in the list is separated from the next by a comma.

No local symbol in the list may be the same as a formal parameter specified for the macro.

No more than 256 local symbols can be used in one program.

4.1.4 IFC

The IFC instruction is optional and allows a set of instructions within a macro definition to be assembled only if a specified condition is true. This instruction is meaningful only within the range of a MAC pseudo instruction.

The format is

s IFC a_1, c, a_2

s — The symbol in the location field is an identifying tag used to establish correspondence with the terminating EIF. An EIF terminates an IFC when the first two characters of the symbol in the address field of EIF are the same as the location symbol of the IFC, or when both symbols are blank and it is the first EIF encountered.

a_i — Must be a string of from 1 to 6 characters or a formal parameter specified in the MAC statement. The character string should not contain commas, blanks, or apostrophes. Two character strings are equal when they contain the same characters in the same position and are of the same length. Characters in excess of six are ignored.

c — Specified condition

Condition	Meaning
EQ	$a_1 = a_2$
NE	$a_1 \neq a_2$

If the condition specified exists between a_1 and a_2, the code is assembled; if not, the code following the IFC is skipped until a corresponding EIF pseudo instruction (section 3.4.4) is encountered.

Source language examples of macro definitions and instructions are given in section 4.3.2.

4.2 MACRO SKELETON

A macro skeleton is the set of instructions within a macro definition that is the prototype of the operations to be performed when the macro is called.

The instructions may be any machine or pseudo instruction except MAC, LOC, EMC, NAM, END, or MON. A macro skeleton may also contain macro instructions calling other macros. A macro skeleton may not contain a macro instruction calling itself. Formal parameters, enclosed in apostrophes, may appear anywhere in the instruction format of a prototype instruction. Local symbols defined by a LOC statement may be used anywhere in the macro skeleton; they also must be enclosed in apostrophes. The only legal use of the apostrophe in a macro definition is to enclose formal parameters or local symbols. Formal parameters that extend past the 72nd character into the sequence field are ignored. Formal parameters in a comment statement signaled by an * in column 1 are also ignored.

In addition to the formal parameters specified in the MAC pseudo instruction, a special formal parameter (a period enclosed in apostrophes) may be used in the macro skeleton. It is replaced by the instruction terminator of the calling macro instruction when a terminator is specified.

Let A, B, C, . . . be distinct arbitrary macro skeletons. A may contain a macro instruction calling B, B a macro skeleton calling C, etc. Up to ten such successive macro calls are allowed by the assembler. Further successive calls are ignored.

Example:

```
XYZ     MAC     P1,P2,P3,P4,P5
        LOC     A
        LDA     'P1'        }
        'P2'    'P3'        }
        S'P4'Z  'A'-*-1     }  Macro skeleton
        JMP'.'  'P5'        }
'A'     ENA     1           }
        EMC
```

4.3 MACRO INSTRUCTION

With a macro instruction, the code generated from the named macro is inserted in the instruction sequence beginning at the location of the macro instruction.

The format is

s N $P_1, P_2, \ldots, P_n$

s A symbolic name in the location field is assigned to the first word of the generated code.

N Symbolic name of the macro in the operation code field. It is the name specified in the location field of the MAC statement of the macro definition it calls. The macro name may be followed by one of the special terminators +, -, or *.

p_i Symbolic names which are local to the macro definition and may be used anywhere else in the program without ambiguity.

4.3.1 PARAMETERS

Actual

The actual parameters must be listed in the same order as the formal parameters in the MAC statement. The list of actual parameters must conform to the following rules.

Each parameter in the list is separated from the next by a comma.

The list is terminated with a blank or the 72nd character unless the 72nd character is a comma.

The list may be continued onto the next line; if so, the last parameter on the first line is terminated by a comma and a blank or the 72nd character.

The continuation line must contain the macro name in the operation code field. A symbolic name in the location field is ignored.

An actual parameter containing embedded blanks or commas must be enclosed by apostrophes.

The internal buffer for storage of actual parameters is 96 words long; this allows approximately three continuation lines. If the buffer overflows, an error message is given.

Example:

The macro defined in the previous example as XYZ could be called by the following macro instruction.

```
TAG1    XYZ*    SYMB1,STA,'SYMB2,I',
        XYZ*    Q,LABEL1                (Continuation line)
```

This macro instruction would generate the following code starting at location TAG1.

```
TAG1    LDA     SYMB1
        STA     SYMB2,I
        SQZ     [nn-*-1
        JMP*    LABEL1
[nn     ENA     1
```

NOTE

[nn is a unique identifier assigned at assembly time.

Null

Actual parameters may be omitted from a macro instruction. An omitted (null) parameter in the middle of the list is indicated by its terminating comma only. Parameters at the end of the list may be omitted with no indication.

Example:

```
XYZ     MAC     P1,P2,P3,P4,P5,P6
```

The macro instruction with P2, P4, and P6 omitted in the actual parameter list would be:

```
XYZ             MUI,,SYMB5,,3
```

Empty fields are allowed in all machine and pseudo instructions with the following exceptions.

```
ALF     n,message       (n must be specified)
EQU     s(e)    }
COM     s(e)    }       (If e is specified, s
DAT     s(e)    }        must be specified)
IFA     e1,c,e2 }
IFC     a1,c,a2 }       (c must be specified)
```

Actual parameters to be inserted into the value of a VFD instruction using mode A must agree with the number of characters specified. A null actual parameter can cause an error in the generated code unless the VFD allows for null parameters.

Example:

```
X       MAC     P,Q,R
        VFD     A8/'P',A8/'Q',A8/'R'
```

For the macro defined, the calling macro instruction must specify each actual parameter as 1 character long. If an actual parameter is more than 1 character, an error message is given. However, if an actual parameter is omitted, a code is generated and an error results.

```
X       A,,B                    (Q is omitted)
VFD     A8/A,A8/,A8/B           (Code generated)
```

If actual parameters might be omitted, the VFD instruction in the macro skeleton should include empty subfields for each character.

Example:

The macro definition should be written:

```
X       MAC     P,Q,R
        VFD     A8/'P',,A8/'Q',,A8/'R',
```

A calling sequence with no actual parameters generates the following code and no error results.

```
        VFD     A8/,,A8/,,A8/,
```

4.3.2 EXAMPLES

The following examples show macro definitions and the code generated by macro instructions calling the defined macros.

1. Macro Definition

```
XYZ     MAC     P1,P2,P3,P4,P5,P6
        LDQ     =N'P5','P6'
'P4'    LDA     'P3'
        'P1'    'P2'
        ADD     SYMB1
        IFA     'P5',NE,0
I1      IFC     'P1',EQ,MUI
        STA     SYMB3
        LDA     SYMB2
        EIF
        EIF     I1
        EMC
```

a. Macro Instruction

```
CALL1   XYZ     MUI,'SYMB4,I',SYMB5,HERE,3,I
```

Generated Code

```
CALL1   LDQ     =N3,I
HERE    LDA     SYMB5
        MUI     SYMB4,I
        ADD     SYMB1
        IFA     3,NE,0          (Condition satisfied)
I1      IFC     MUI,EQ,MUI      (Condition satisfied)
        STA     SYMB3           (Assembled)
        LDA     SYMB2           (Assembled)
        EIF
        EIF     I1
```

b. Macro Instruction

```
CALL2   XYZ     DVI,SYMB7,'SYMB8,I',THERE,2
```

Generated Code

```
CALL2   LDQ     =N2,
THERE   LDA     SYMB8,I
        DVI     SYMB7
        ADD     SYMB1
        IFA     2,NE,0          (Condition satisfied)
I1      IFC     DVI,EQ,MUI      (Condition not satisfied)
        STA     SYMB3           (Not assembled)
        LDA     SYMB2           (Not assembled)
        EIF
        EIF     I1
```

2. Macro Definition

```
A       MAC     P1,P2,P3,P4
I1      IFC     *,EQ,'P1'
        LDA     'P2'
        EIF     I1
I2      IFC     *,NE,'P1'
        LDA     'P3'
        EIF     I2
        STA     'P4'
        EMC
```

a. Macro Instruction

```
        A       *,NAM1,NAM2,NAM3
```

Generated Code

```
I1      IFC     *,EQ,*          (Condition satisfied)
        LDA     NAM1            (Assembled)
        EIF     I1
I2      IFC     *,NE,*          (Condition not satisfied)
        LDA     NAM2
        EIF     I2
        STA     NAM3
```

3. Macro Definition

```
JAN     MAC     SY
        IFC     *,EQ,'.'
        SAZ     1
        EIF
        IFC     *,NE,'.'
        SAZ     2
        EIF
        JMP'.'  'SY'
        EMC
```

a. Macro Instruction

```
                JAN*        SYMB1
```

Generated Code

```
                IFC         *,EQ,*      (Condition satisfied)
                SAZ         1           (Assembled)
                EIF                     (Ignored)
                IFC         *,NE,*      (Condition not satisfied)
                SAZ         2           (Not assembled)
                EIF                     (Skip terminated)
                JMP*        SYMB1
```

b. Macro Instruction

```
                JAN         SYMB2
```

Generated Code

```
                IFC         *,EQ,       (Condition not satisfied)
                SAZ         1           (Not assembled)
                EIF                     (Skip terminated)
                IFC         *,NE,       (Condition satisfied)
                SAZ         2           (Assembled)
                EIF                     (Ignored)
                JMP         SYMB2
```

4. Macro Definition

```
IFEXMP  MAC     P1
Z       IFC     *,EQ,'P1'
        NUM     2
        EIF     Z
Y       IFC     *,NE,'P1'
X       IFC     0,EQ,'P1'
        NUM     1
        EIF     X
Y       IFC     0,NE,'P1'
        NUM     0
        EIF     Y
        EMC
```

a. Macro Instruction

```
                IFEXMP      *
```

Generated Code

```
Z       IFC     *,EQ,*          (Condition satisfied)
        NUM     2               (Assembled)
        EIF     Z
Y       IFC     *,NE,*          (Condition not satisfied)
X       IFC     0,EQ,*          (Not assembled)
        NUM     1               (Not assembled)
        EIF     X               (Not assembled)
Y       IFC     0,NE,*          (Not assembled)
        NUM     0               (Not assembled)
        EIF     Y               (Skip terminated)
```

b. Macro Instruction

```
        IFEXMP  0
```

Generated Code

```
Z       IFC     *,EQ,0          (Condition not satisfied)
        NUM     2               (Not assembled)
        EIF     Z               (Skip terminated)
Y       IFC     *,NE,0          (Condition satisfied)
X       IFC     0,EQ,0          (Condition satisfied)
        NUM     1               (Assembled)
        EIF     X
Y       IFC     0,NE,0          (Condition not satisfied)
        NUM     0               (Not assembled)
        EIF     Y               (Skip terminated)
```

c. Macro Instruction

```
        IFEXMP
```

Generated Code

```
Z       IFC     *,EQ,           (Condition not satisfied)
        NUM     2               (Not assembled)
        EIF     Z               (Skip terminated)
Y       IFC     *,NE,           (Condition satisfied)
X       IFC     0,EQ,           (Condition not satisfied)
        NUM     1               (Not assembled)
        EIF     X               (Skip terminated)
Y       IFC     0,NE,           (Condition satisfied)
        NUM     0               (Assembled)
        EIF     Y
```

5. Macro Definitions

```
DEPTH1  MAC     A
        DEPTH2  'A',PARAM1
        EMC
DEPTH2  MAC     A,B
        DEPTH3  'A',PARAM2
        EMC
DEPTH3  MAC     C,D
        LDA     'C'
        STA     'D'
        EMC
```

Macro Instruction

```
            DEPTH1      SYMB1
```

Generated Code

```
            DEPTH2      SYMB1,PARAM1
            DEPTH3      SYMB1,PARAM2
            LDA         SYMB1
            STA         PARAM2
```

6. Macro Definition

```
B           MAC         A,B,C,D,E,F,G,H,I,J,K
            LOC         LO
            ALF         'A','B'Δ ERROR
            VFD         'C'/'D',A16/'E',,,A32/TEST
            IFC         'G',EQ,SKIP
            LDA         'H'
            EIF
'I'         INA         'J'
            'K'         1
            SAN         'LO'
            ENA         -1
'LO'        STA         'F'
            EMC
```

Macro Instruction

```
            B           4,1,N4,-1,XY,'TEMP,I',SKIP,'TEMP,I',
            B           NAM2,10,NOP
```

Generated Code

```
            ALF         4,1 Δ ERROR
            VFD         N4/-1,A16/XY,,,A32/TEST
            IFC         SKIP,EQ,SKIP
            LDA         TEMP,I
            EIF
NAM2        INA         10
            NOP         1
            SAN         [nn
            ENA         -1
[nn         STA         TEMP,I
```

STANDARD MACRO LIBRARY 5

5.1 CREATING THE LIBRARY

LIBMAC is released as a separate library macro preparation routine. Input to this routine is in the form of a set of macro definitions, each starting with a MAC statement and ending with an EMC statement. The definitions for the macros may be obtained from a COSY tape. All the macros are contained in one deck on the COSY tape with the deckname MACROS. No extra modifications are needed to use the source code obtained from the COSY tape. LIBMAC must appear in column 1, following the set of macro definitions.

The procedure to execute LIBMAC is:

```
*JOB
J
*K, Ilu, Plu
J
*LIBMAC
```

Where: I assigns the logical unit of input.

P assigns the logical unit output.

The library macro preparation routine outputs two files on the standard I/O device for binary output. One contains a macro directory; the other contains the macro skeletons. The routine checks for errors and prints an error message along with the line containing the error.

Binary output is in two sections: the macro skeleton file and the macro directory file. After the skeleton file has been output, the message MACSKL END is output on the typewriter and a carriage return must be typed to start output of the macro directory.

The output files are placed on the program library in two permanent files using the MSOS System Initializer or library editor. The library editor is used to put the macros on the program library.

The control statement

```
*N, MACROS, , , B
```

places the macro directory file on the program library.

The control statement

```
*N, MACSKL, , , B
```

places the macro skeletons on the program library.

The following error codes are output by the macro library generator (LIBMAC). The format is

LIBMAC ERROR nn ...

Where nn is one of the following codes:

Code	Meaning
01	No MAC definition card
02	Address modifier on MAC card
03	Label field missing or incorrect
04	Illegal terminator after macro name
05	More than two characters in a MAC or LOC definition card
06	Invalid special character on MAC or LOC card
07	Duplicate parameter names on MAC and/or LOC card
08	Invalid special character in a parameter string on a MAC or LOC card
09	Address modifier on LOC card
0A	No terminating apostrophe on macro skeleton record
0B	Parameter name on macro skeleton record not previously defined on MAC or LOC card
0C	Internal buffer exceeded; skeleton record too long
0D	Macro definitions exceeded limit (currently 320 definitions allowed)
0E	More than 65K or skeleton file defined

The line printed following the error code is the line in error. All errors are fatal.

5.2 MODIFYING THE LIBRARY

All modifications to the macro library require a new macro library to be generated. Macro definitions may be added or removed from the old macro source deck. The new macro library may be created using *LIBMAC.

5.3 PROGRAMS IN THE MACRO LIBRARY

The macros described in this section can be found in the macro library of a standard system. Additional macros may be added according to the user's needs. The macro assembler recognizes the macros in the library and converts them to their appropriate calling sequences.

5.3.1 FORMATTING MACROS

The formatting macros allow the programmer to transfer information from one area of storage to another while changing the format or type class of the information. To change the type of a single variable, the programmer may use a HEXASC, HEXDEC, ASCII, DECHEX, or FLOATG macro. Variable lists may be formatted using the ENCODE and DECODE macros. (Refer to the MS FORTRAN Version 3A/B reference manual for further details.)

HEXASC and HEXDEC Macros

The HEXASC macro converts a hexadecimal integer to ASCII characters. HEXDEC converts a hexadecimal integer to a decimal integer in ASCII characters. The macro calling sequence is:

```
HEXASC  a,b (absolute)
   or
HEXASC* a,b (relative)
HEXDEC  a,b (absolute)
   or
HEXDEC* a,b (relative)
```

Where: a is the address of the variable.

b is the address of the buffer (two words for HEXASC, three words for HEXDEC).

ASCII and DECHEX Macros

The ASCII macro converts two words of ASCII characters in BUFFER to a hexadecimal integer. DECHEX converts three words of a decimal integer in ASCII characters in BUFFER to a hexadecimal integer. The macro calling sequence is:

```
ASCII  a,b     (absolute)
   or
ASCII* a,b     (relative)
DECHEX  a,b (absolute)
   or
DECHEX* a,b (relative)
```

Where: a is the buffer address (two words for ASCII and three words for DECHEX).

b is the address of the variable.

FLOATG Macro

FLOATG converts a two-word floating-point variable into its floating-point representation with its exponent in ASCII characters: ±.xxxxxxE±ee. The macro calling sequence is:

```
FLOATG   a,b  (absolute)
     or
FLOATG*  a,b  (relative)
```

Where: a is the address of a floating-point variable.

b is the address of a buffer (six words).

ENCODE AND DECODE MACROS

The DECODE macro transmits n consecutive ASCII characters according to FORMAT into locations starting with the first word in BUFFER to the variable list as n machine-language elements. ENCODE transmits n machine-language elements of the variable list according to FORMAT into locations starting with the first word in BUFFER. Up to 150 ASCII characters (one line) are stored in consecutive locations for output. The macro calling sequence is:

```
ENCODE   a,b,c,d,e,   (absolute)
         or
ENCODE*  a,b,c,d,e    (relative)
DECODE   a,b,c,d,e    (absolute)
         or
DECODE*  a,b,c,d,e    (relative)
```

Where: a is the address of a buffer.

b is the address of the format.

c is the number of words to be transferred.

d is the address of the variable list.

e is the address of an error routine. If it is blank, no test for error conditions is made.

5.3.2 FILE MANAGER MACROS

The following File Manager macros provide the programmer with a convenient method for performing all the File Manager functions:

FLDF
DEFFIL and DEFIDX
LOKFIL and UNLFIL
RELFIL

STOSEQ, STOIDX, and STODIR
RTVSEQ and RTVDIR
RTVIDX and RTVIDO
STATFL

These macros allow the user to create and maintain sequential or indexed files. When a file is no longer needed, its space may be released for other tasks through the use of a File Manager macro. These macros also provide an easy method for obtaining the request indicator word which specifies errors that occurred on the last file request. (Refer to the File Manager Reference Manual for further details.)

FLDF Macro

FLDF defines the parameters of a file so that they do not have to be in the calling sequence of the other macros. The macro calling sequence is:

```
FLDF  filnum, maxrl, lu, numekv, keylth, filcom, reclth
```

Where: filnum is the file number; it contains a positive integer specifying the file.

maxrl is the maximum record length; to be used for determining the effective maximum record length and file record block length. It contains a positive integer.

lu is the logical unit; it contains a positive integer specifying where the file's records are to be stored.

numekv is the number of expected key values; it contains a positive integer estimating the number of records with different key values to be stored in the file.

keylth is the key length word, with the indexed options:

Bits 0 through 5	Length of the key	
6 through 12	Reserved	
13	1	FIFO linking (bit 15 must be set). If this bit is not set and bit 15 is set, LIFO linking is implied.
14	1	Indexed-ordered file
15	1	Indexed-linked file

filcom is the file combination with the remove option; bits 0 through 14 contain a nonzero number (if the file is or is to be locked) specifying the combination (which is or is to be) used to lock the file; bit 15 set to 1 indicates that the record is to be removed from the file.

reclth is the record buffer length; it contains a positive integer specifying the length of the record buffer.

DEFFIL and DEFIDX Macros

The DEFFIL macro defines a file; DEFIDX further defines a file as being indexed.

The macro calling sequence is:

```
DEFFIL   filnum   (absolute)
    or
DEFFIL*  filnum   (relative)
DEFIDX   filnum   (absolute)
    or
DEFIDX*  filnum   (relative)
```

Where: filnum is the file number.

LOKFIL and UNLFIL Macros

The LOKFIL macro locks the file; UNLFIL unlocks the file. The macro calling sequence is:

```
LOKFIL   filnum   (absolute)
    or
LOKFIL*  filnum   (relative)
UNLFIL   filnum   (absolute)
    or
UNLFIL*  filnum   (relative)
```

Where: filnum is the file number.

RELFIL Macro

The RELFIL macro releases the file so that space previously used by the file can be reused. The macro calling sequence is:

```
RELFIL   filnum   (absolute)
    or
RELFIL*  filnum   (relative)
```

Where: filnum is the file number.

STOSEQ, STOIDX, and STODIR Macros

There are three File Manager macros that are used to store a record:

- STOSEQ – Stores a record sequentially into a file
- STOIDX – Stores a record using an index into a file
- STODIR – Stores directly into a file

The macro calling sequence is:

```
STOSEQ    filnum, recbuf, reclth   (absolute)
    or
STOSEQ*   filnum, recbuf, reclth   (relative)
STOIDX    filnum, keyval, recbuf   (absolute)
    or
STOIDX*   filnum, keyval, recbuf   (relative)
STODIR    filnum, recbuf           (absolute)
    or
STODIR*   filnum, recbuf           (relative)
```

Where: filnum is the file number; it contains a positive integer identifying the file into which a record is to be stored.

recbuf is the record buffer; it is an array of reclth words containing the record to be stored.

reclth is the record length. It contains a positive integer specifying the length of the record. If the record is the first to be stored into an indexed-linked file with FIFO linking, reclth becomes the fixed record length for all subsequent stores. If it is not the first store into the file, reclth must be less than or equal to the length of the first record. (Even though reclth may be less than the length of the first record, the fixed length will be used in storing all subsequent records.)

keyval is the key value; it is an array of keylth words containing the key value of the record.

RTVSEQ and RTVDIR Macros

The RTVSEQ macro is used to retrieve a record sequentially from a file; RTVDIR is used to retrieve a record directly from a file. The macro calling sequence is:

```
RTVSEQ    filnum, recbuf     (absolute)
    or
RTVSEQ*   filnum, recbuf     (relative)
RTVDIR    filnum, recbuf     (absolute)
    or
RTVDIR*   filnum, recbuf     (relative)
```

Where: filnum is the file number; it contains a positive integer identifying the file from which the record is to be retrieved.

recbuf is the record buffer; it is a nonpreset array of reclth words, where the File Manager transfers the retrieved record.

RTVIDX and RTVIDO Macros

The RTVIDX macro is used to retrieve a record using an index from a file. RTVIDO is used to retrieve a record using an ordered index from a file. The macro calling sequence is:

```
RTVIDX   filnum, keyval, recbuf     (absolute)
         or
RTVIDX*  filnum, keyval, recbuf     (relative)
RTVIDO   filnum, keyval, recbuf     (absolute)
         or
RTVIDO*  filnum, keyval, recbuf     (relative)
```

Where: filnum is the file number; it contains a positive integer identifying the file from which a record is to be retrieved.

keyval is the key value; it contains an integer equal to the lowest numeric key value desired; otherwise, it contains a positive integer specifying the numeric key value of the desired record.

recbuf is the record buffer; it is a nonpreset array of reclth words, where the file manager transfers the retrieved record.

STATFL Macro

STATFL provides the user with an easy method of getting the request indicator word, masking specified error conditions, and giving control to a specified error routine if errors are present. A file status of 0 implies that no errors occurred on the last file request.

The macro calling sequence is:

```
STATFL   fn, mk, bd
STATFL   fn
STATFL   fn, mk
STATFL   fn, mk, bd
```

Where: fn is the file number.

mk is the mask that is used to form the logical product with the request indicator. (If mk is left blank, only the status is placed in the A register.) The terminator, such as the dash (-) in the fourth example, determines the addressing mode used on the AND instruction and may be a -, +, *, or blank.

bd is the program label where control is given if the logical product of mk and the request indicator is nonzero. If bd is left blank, no code is generated to test the request indicator status. In this case the logical product of the request indicator and the mask is left in the A register at the end of the macro and may be tested by the user.

5.3.3 MONITOR REQUEST MACROS

The following monitor request macros provide the programmer with a convenient method for making requests to the monitor:

READ, FREAD, WRITE, and FWRITE
INDIR
TIMER
SCHDLE
MOTION
SPACE
RELEAS
DISCHD
ENSCHD
TIMPT1
PTNCOR
SYSCHD
CORE
LOADER
GTFILE
STATUS
EXIT

With these macros, the program may instruct the monitor to read, write, load, schedule programs, allocate, and release space and motion. Special MOTION request macros are also included. Each macro performs one MOTION request. Refer to the MSOS reference manual for further details.

READ, FREAD, WRITE, and FWRITE Macros

READ/WRITE instructions transfer data between the specified input/output device and core. The word count specified in the request determines the end of the transfer.

FREAD/FWRITE requests read/write records in a specific format for each device.

The macro calling sequence is:

READ, FREAD, WRITE, FWRITE } lu, c, s, n, m, rp, cp, a, x, d

Where:
- lu is the logical unit.
- c is the completion address.
- s is the starting address.
- n is the number of words to transfer.
- m is the mode.
- rp is the request priority.
- cp is the completion priority.
- a is the absolute/indirect indicator for the logical unit.
- x is the relative/indicator (affects parameters C, S, and N).
- d is the Part 1 request indicator (absolute parameter addresses).

INDIR Macros

The INDIR macro allows indirect execution of any other request, as determined by the parameter list referenced by p.

The macro calling sequence is:

```
INDIR p, i
```

Where: p is the address of the first word of the parameter list of any other request; p must not be enclosed in parentheses.

i is the indicator for the request used.

TIMER Macro

The TIMER macro is a delayed SCHDLE request. Through the user of TIMER, a SCHDLE request is made after a specified time delay. The macro calling sequence is:

```
TIMER c, p, x, t, u, d
```

Where: c is the completion address to be executed.

p is the priority level of the program.

x is the relative/indirect indicator.

SCHDLE Macro

Programs are queued on a priority basis through the use of the SCHDLE macro. A program requested by SCHDLE is executed only when it is the oldest waiting task with the highest priority. The macro calling sequence is:

```
SCHDLE c, p, x, d
```

Where: c is the address to be executed.

p is the priority level of the program; for unprotected programs, p is 1. Completion routines requested by unprotected programs are not executed until the scheduled routine exits. If two programs are of equal priority, the one in progress is continued.

x is the relative/indirect indicator.

d is the Part 1 request indicator (absolute request parameters).

MOTION Macro

The MOTION macro request is used to control motion and end-of-file processing. The macro calling sequence is:

```
MOTION lu, c, p1, p2, p3, dy, rp, cp, a, x, d, m
```

Where: lu is the logical unit.

c is the completion address.

p_1, p_2, p_3 are the motion control parameters. Up to three motion commands may be defined in a MOTION request; they are executed in the sequence p_1, p_2, p_3. The first command with a value of zero terminates the request.

dy is the density parameter.

rp is the request priority.

cp is the completion priority.

a is the absolute/indirect indicator for the logical unit.

x is only related to the completion address.

d is set to 0 All parameters are processed as described.

1 A Part 1 request is indicated (c is a 16-bit absolute address and must not equal R for the a parameter).

m is the mode.

The following macros can also be used for MOTION requests; each macro can perform only one MOTION request.

BSR*	lu, a, n, c, p	Motion code 1
EOF*	lu, a, n, c, p	Motion code 2
REW*	lu, a, n, c, p	Motion code 3
UNL*	lu, a, n, c, p	Motion code 4
ADF*	lu, a, n, c, p	Motion code 5
BSF*	lu, a, n, c, p	Motion code 6
ADR*	lu, a, n, c, p	Motion code 7
MOT	lu, a, n, c, p, m	Used by each of the above macros to execute the motion

Where: * specifies a relative completion address. If left blank, there is absolute completion (The macro computes the relative address constant.)

lu is the logical unit number of the device.

a is the absolute/indirect/relative indicator for the logical unit.

n is the number of iterations. If blank, 1 is assumed (not to exceed 4,095).

c is the completion address. If the macro call terminator is an *, completion is relative (only the label name is required). If the macro call terminator is a blank, the completion is absolute. If c is left blank, there is no completion.

p is the priority level; defines both the request and completion priority. If left blank, the priority is 0.

m is the motion code.

All parameters are optional and may be left blank, with the exception of lu.

SPACE Macro

SPACE is used by protected programs to allocate space in core. To operate mass storage resident programs, the SPACE request processor must be used. The macro calling sequence is:

```
SPACE n, c, rp, cp, x, d
```

Where:
- n is the number of words necessary.
- c is the completion address to which control is transferred when core space is allocated.
- rp is the request priority (with respect to other SPACE requests). If space is not available, requests are threaded together so that the oldest (highest priority) is filled first when space becomes available. This priority is also used as the index to the table LVLSTR to determine the starting address of of allocatable core for the request priority. This has the effect of providing larger areas of core to SPACE requests with a higher priority level.
- cp is the completion priority; the level at which the completion address is entered.
- x is the relative/indirect indicator.

RELEAS Macro

The RELEAS macro is used to return to the system storage acquired by a SPACE macro. The macro calling sequence is:

```
RELEAS s, t, x, d
```

Where:
- s is the starting address of the block to be released. If this address is not the same as the address returned from a SPACE request, core space is not released; however, an error does not occur.
- t is the exit indicator.
- x is the relative/indirect indicator.
- d is the Part 1 request indicator.

DISCHD Macro

With the DISCHD macro, the scheduling of specific system directory programs can be disabled for a period of time. The macro calling sequence is:

```
DISCHD c
```

Where: c is the index to the system directory.

ENSCHD Macro

The ENSCHD macro enables the scheduling of system directory programs after they have been disabled by a DISCHD macro. The macro calling sequence is:

```
ENSCHD c
```

Where: c is the index to the system library.

TIMPT1 Macro

The Part 1 TIMER macro must be used for scheduling system directory programs that are loaded in Part 1. This request may also be used for Part 0 programs. The macro calling sequence is:

```
TIMPT1 c, p, x, t, u
```

Where:
- c is the index to the system directory.
- p is the priority level of the program.
- x has no meaning.
- t is the time delay.
- u is the units of delay. This parameter determines the units in which the time delay is measured.

PTNCOR Macro

The PTNCOR macro is used to allocate a block of partitioned core. The macro calling sequence is:

```
PTNCOR n, c, p, rp, cp, x, d
```

Where:
- n is the number of words in block to be allocated.
- c is the completion address.
- p is the starting partition number; the number of the first partition in the block to be allocated (partitions are numbered zero through fifteen).
- rp is the request priority. This priority governs where this request will be threaded on partition p's thread if more than one request is on the thread.
- cp is the completion priority; the level at which the completion address is entered.
- x is the relative/indirect indicator.
- d is the Part 1 request indicator.

SYSCHD Macro

Protected programs, which run in Part 1, must use this request to schedule a system directory program. It may also be used for Part 0 programs. The macro calling sequence is:

```
SYSCHD c, p
```

Where: c is the index to the system directory. The entry referred to by the index specifies the program.

p is the priority level of the program.

CORE Macro

This macro is used to set or determine the bounds of available unprotected core (that portion of unprotected core not occupied by a program or data for a job). If the A and Q registers are 0 when the request is made, the current upper bound is returned in A and the lower bound in Q. To set the bounds, the request is made with the upper bounds in A and the lower bounds in Q. Both values must be in unprotected core and the upper value must be greater than the lower. Illegal values result in job termination. Each new request replaces the parameters from the previous request. At the beginning of a load, the entire unprotected area is made available again.

The macro calling sequence is:

```
CORE
```

LOADER Macro

The LOADER macro is available to unprotected programs at level zero only. It is used to execute the mass storage resident relocatable binary loader. The A register contains the input logical unit if a relocatable binary program is being loaded. The Q register contains the type of loading operation. Parameters must be in the A and Q registers at the time the request is made.

The macro calling sequence is:

```
LOADER
```

GTFILE Macro

The GTFILE macro is used to access permanent files in the program library. The macro calling sequence is:

GTFILE c, i, s, w_1, w_2, x, rp, cp, d

Where: c is the completion address.

i is the address increment; a positive increment that is added to the address of the first word of the parameter list to form the address of the first of a three-word block containing the ASCII name of the file.

s is the starting address of the block into which the file, or portion of the file, is to be retransferred.

x is the relative/indirect indicator.

w_1, w_2 are the first and last words, if only part of the file is required. These parameters must be blank if the entire file is to be used.

rp is the priority of the mass storage requests needed to complete a GTFILE request; it is always 0 for unprotected requests.

cp is the priority of the completion address, the level at which the completion address is to be executed; it is always 1 for unprotected requests.

d is the Part 1 request indicator.

STATUS Macro

The STATUS macro is used to determine the status of an input/output device by accessing information from the physical device table for the specified logical unit. The macro calling sequence is:

```
STATUS lu, 0, a
```

Where: lu is the logical unit; an ordinal in the logical equipment tables modified by parameter a.

0 is the third word of the calling sequence; it must always be 0.

a is the absolute/indirect indicator.

EXIT Macro

This macro is used by unprotected programs to signal completion of a job or an interrupt routine. When computation if completed, the request notifies the operating system. The macro calling sequence is:

```
EXIT
```

5.3.4 OTHER MACROS

Each of the following macros perform a frequently used function:

VOLA
VOLR
CLOCK
DISP
BUFFER

These macros provide the programmer with a convenient method for allocating and releasing volatile storage, obtaining a value for the real-time clock, exiting to the Dispatcher, and creating a physical device table for the software buffer.

VOLA Macro

VOLA allocates volatile storage. The macro calling sequence is:

```
VOLA  a,b
```

Where: a is the number of words requested.
b is the return address.

VOLR Macro

VOLR releases volatile storage. The macro calling sequence is:

```
VOLR  a,b
```

Where: a is the storage location of the return address.
b is the increment added to the return address.

CLOCK Macro

CLOCK picks up the value for the real-time clock from low core. The macro calling sequence is:

```
CLOCK  a
```

Where: a is the storage address of the clock value. If a is blank, the value will be in the A register only.

DISP Macro

DISP causes an exit to the Dispatcher. The macro calling sequence is:

```
DISP
```

BUFFER Macro

BUFFER creates a physical device table for the software buffer. The macro calling sequence is:

```
BUFFER  a,b,c,d,e,f
```

Where: a is the least significant bits of the start of the buffer.
b is the least significant bits of the end of the buffer.
c is the most significant bits of the buffer.
d is the output logical unit.
e is the mass memory logical unit.
f is the character buffer size.

ASSEMBLER OUTPUT 6

6.1 CONTROL OPTIONS

Four standard options determine the type of output from the assembler. All four are automatically selected if no OPT statement is encountered before the first NAM.

Standard Option	Meaning
P	Relocatable binary output on standard output unit
X	Load and go; execute output loaded on a mass storage device.
L	List output on standard list unit.
C	List cross-references at end of assembly listing
Nonstandard Option	
M	List expansion of macro code

6.1.1 P OPTION

Relocatable binary output is selected by the P option. The format is described in the MSOS reference manual.

The standard output binary device is used for relocatable binary information. If the binary output device is magnetic tape, the final relocatable program terminates with an EOL record, *T. If the binary output device is paper tape, a blank trailer terminates each assembly.

6.1.2 X OPTION

If the X option is selected, relocatable binary output is placed on the mass storage unit for subsequent loading and execution as described in the MSOS reference manual.

6.1.3 L OPTION

The L option results in an assembly listing described as follows.

6.1.4 C OPTION

The C option produces a cross-reference list that is printed at the end of the assembly list.

With the OPT pseudo instruction, any or all of the preceding options may be omitted. OPT also provides options for listing macro skeletons and abandoning assembly.

6.1.5 M OPTION

The M option produces an expansion of all operand addresses and comment information contained in any selected macro call used in a program.

6.2 ASSEMBLY LISTING

The assembly list, output to standard list output device, consists of 20 columns (including spacing before printing) of information related to the source statement, followed by a maximum of 80 columns listing the source statement.

Each page has a header containing the program name, page number, and date.

Column	Contents
1 through 4	Card number; truncated from 5 to 4 decimal digits
5	Space
6	Relocation designator for location P Program relocation D Data relocation
7 through 10	Location in hexadecimal
11	Space
12 through 15	Machine word in hexadecimal
16 through 17	Relocation designator for word P Program relocation -P Negative program relocation C Common relocation -C Negative common relocation D Data relocation -D Negative data relocation X External blank Absolute
18	Space
19 through 98	Input source statement

Following the assembly list, the lengths of the program, common, and data are given in hexadecimal and decimal values,

PGM = 0155(341) COM = 2BE(702) DAT = 0000(0)

The data length includes those areas reserved by DAT pseudo instructions.

6.2.1 ERROR LISTING

A list of errors occurring in passes 1 and 2 precedes the program listing on the standard list I/O unit. If the L option is selected, errors in pass 3 precede the source line on the list output. A decimal error count is printed at the end of each subprogram. If L is not selected, error messages are output on the standard comment unit.

Format for pass 1 and 2 error messages:

Column	Contents
1 and 2	**
3 through 6	4-digit line number
6 and 7	**
8 and 9	2-character error code
10 through 19	**********

Format for pass 3 error messages:

Column	Contents
1 through 6	******
7 and 8	2-character error code
9 through 18	**********

The error codes and their meanings are given in Appendix D.

6.2.2 CROSS—REFERENCE LISTING

Cross-references are listed at the end of an assembly listing if the option C was specified by the user. Cross-references will also be listed if no OPT statement was found, since the C option is a default option.

The cross-references are divided into four functional parts:

1. Equivalences
2. Symbols
3. Externals
4. Symbols in alphabetical order

If cross-references are to be listed and there is not enough core to process all four parts of the cross-references listing, then the assembler attempts to sort the symbol table alphabetically. If there is not enough core to sort the symbol table alphabetically, the symbol table is dumped.

The equivalences, symbols, and externals are listed according to the line number at which they are defined. In addition to the definition line number, the value or address and the line numbers of all references to that symbol are given. The list of symbols in alphabetical order includes all the symbols in the program. The number following each symbol is the corresponding definition line.

6.2.3 SAMPLE PROGRAM

The following source program results in the assembly listing in section 6.2.4.

```
          NAM       TEST2 ERS MACRO EXAMPLEX
XYZ       MAC       P1,P2,P3,P4,P5,P6
          LDQ       =N'P5','P6'
'P4'      LDA       'P3'
          'P1'      'P2'
          ADD       SYMB1
          IFA       'P5',NE,0
I1        IFC       'P1',EQ,MUI
          STA       SYMB3
          LDA       SYMB2
          EIF
          EIF       I1
          EMC
MACRO     MAC       P1,P2,P3,P4,P5,P6
          LOC       A
          LDA       'P1'
          'P2'      'P3'
          S'P4'Z    'A'-*-1
          JMP'P5'   'P6'
'A'       ENA       1
          EMC
          MACRO     SYMB1,STA,'SYMB2,I',
          MACRO     Q,*,LABEL1
SYMB1     ADC       0
SYMB2     ADC       0
          XYZ       MUI,,SYMB5,,3
SYMB5     ADC       0
CALL1     XYZ       MUI,'SYMB4,I',SYMB5,HERE,3,I
SYMB4     ADC       0
CALL2     XYZ       DVI,SYMB7,'SYMB8,I',THERE,2
SYMB8     ADC       0
SYMB7     ADC       0
          END
```

6.2.4 SAMPLE LISTING

The following assembly listing is output from the assembly of the source program in section 6.2.3.

```
0001                NAM       TEST2 ERS MACRO EXAMPLEX
0002      XYZ       MAC       P1,P2,P3,P4,P5,P6
0003                LDQ       =N'P5','P6'
0004      'P4'      LDA       'P3'
0005                'P1'      'P2'
0006                ADD       SYMB1
0007                IFA       'P5',NE,0
```

```
0008                        I1          IFC     'P1',EQ,MUI
0009                                    STA     SYMB3
0010                                    LDA     SYMB2
0011                                    EIF
0012                                    EIF     I1
0013                                    EMC
0014                        MACRO       MAC     P1,P2,P3,P4,P5,P6
0015                                    LOC     A
0016                                    LDA     'P1'
0017                                    'P2'    'P3'
0018                                    S'P4'Z  'A'-*-1
0019                                    JMP'P5' 'P6'
0020                        'A'         ENA     1
0021                                    EMC
0022                                    MACRO SYMB1,STA,'SYMB2,I',
0023                                    MACRO  Q,*,LABEL1
0023  P0000 C800
      P0001 0006
0023  P0002 6900
      P0003 0005
0023  P0004 0141
******UD**********
******RL**********
0023  P0005 1000
0023  P0006 0A01
0024  P0007 0000        SYMB1   ADC     0
0025  P0008 0000        SYMB2   ADC     0
0026                          XYZ     MUI,,SYMB5,,3
0026  P0009 E000
      P000A 0003
0026  P000B C800
      P000C 0009
0026  P000D 2400
      P000E 0000
0026  P000F 8800
      P0010 FFE6
******J**********
0026  P0011 6400
      P0012 0000
0026  P0013 C800
      P0014 FFE3
0027  P0015 0000        SYMB5   ADC     0
0028                    CALL1   XYZ     MUI,'SYMB4,I',SYMB5,HERE,3,I
0028  P0016 E100
      P0017 0003
0028  P0018 C800
      P0019 FFE8
0028  P001A 2900
      P001B 0007
0028  P001C 8800
      P001D FFE9
******J**********
0028  P001E 6400
      P001F 0000
0028  P0020 C800
      P0021 FFE6
0029  P0022 0000        SYMB4   ADC     0
```

```
0030                CALL2     XYZ       DVI,SYMB7,'SYMB8,I',THERE,2
0030   P0023 F000
       P0024 0002
0030   P0025 C900
       P0026 0005
0030   P0027 3800
       P0028 0004
0030   P0029 8800
       P002A FFDC
0031   P002B 0000   SYMB8     ADC       0
0032   P002C 0000   SYMB7     ADC       0
0033                          END
```

GLOSSARY

Absolute address	An address that is permanently assigned by the machine designer to a storage location
Address field	Contains an address expression consisting of one or more operands joined by arithmetic operators
ASCII	American National Standard Code for Information Interchange. The standard code, using a coded character set consisting of seven-bit coded characters (eight bits including parity check), used for information interchange amount data processing systems, communication systems, and associated equipment. The ASCII set consists of control characters and graphic characters.
Assembler	A program that prepares an object language program from a symbolic (source) language program by substituting machine instructions for symbolic instructions and by generating absolute or relocatable addresses for symbolic addresses.
Character mode	The eight-bit data mode. Characters are usually ASCII, seven bits, right-justified in the eight-bit field.
Clear	The process of forcing all bits to zero
Comment field	Printed after the address field of any instruction. The comments have no effect on the object program.
Control options	Determines assembler outputs
Cross reference listing	An output of the macro assembler with four parts: equivalences, symbols, externals, and alphabetized symbols
Dispatcher	The module within the MSOS Monitor that determines the next program to execute
Enhanced machine instructions	The set of additional machine instructions added to the original machine instructions by the micro-processor computer family (CYBER 18/1700). Six new instruction types are available: type 2 storage reference, type 2 skip, type 2 inter-register, field reference, decrement and repeat, miscellaneous.
Fx	Field designators for machine instructions. There are seven types: F and F1 through F6.
Free field	Fields of varying length as opposed to fixed fields that always have the same number of bits in each specific field type. For MSOS source language, free fields must be ended with a field terminator: blank, carriage, return, or tab.
Hexadecimal	Pertaining to the number representation system having the base 16

Indirect address	An address that specifies a storage location that contains either a direct address or another indirect address.
Index register	A register whose contents may be used to modify addresses or for other program-specified purposes. Macro assembler uses A, Q, I, and B for normal indexing and A, Q, I, and 1 through 4 for enhanced instruction indexing. B is a pseudo register consisting of Q+I.
Interrupt mask	A mask that defines the interrupt lines for the interrupts that are allowed
Library	An organized collection of standard, checked-out programs, routines, and subroutines
Location field	Used to specify a labeled or unlabeled statement
Logical product	A bit-by-bit multiplication of two binary numbers according to a specific set of rules
Machine instruction	An instruction the computer can recognize and execute
Macro instruction	An instruction is a source language that is equivalent to a specified sequence of machine instructions
Macro skeleton	The set of instructions within a macro definition that is the prototype of the operation to be performed when the macro is called
Monitor	The program that exercises overall CPU control under MSOS. When initialization is completed and the system is ready for on-line operation, control passes to the monitor, which then executes programs according to prescribed or operator-directed instructions.
Monitor macros	Macro statements held in the macro library (see appendix F).
One's complement	The base-minus-one complement of a numeral whose radix is two
Operand	That which is operated upon. An operand is usually identified by an address part of an instruction.
Options	See section 6.
Program relocatable	Relocation determined by the location of the program load
Pseudo instruction	Instructions that require translation prior to execution
Relative address	The number that specifies the difference between the absolute address and the base address
Sequence field	Used when the source image is 80 characters (columns 73 through 80)
Set	Sets all bits to 1s
SK	Skip count
Source language	A language that is an input to a given translation process

MNEMONIC INSTRUCTION CODES A

BASIC MACHINE INSTRUCTIONS

There are six classes of basic (nonenhanced) machine instruction codes.

Storage reference, Group A
Storage reference, Group B
Register
Shift
Skip
Interregister transfer

Storage Reference Instructions

	Operation Code	Definition
Group A	LDA	Load A register
	LDQ	Load Q register
	ADD	Add to the A register
	SUB	Subtract from A register
	ADQ	Add to Q register
	AND	Perform logical AND with A register
	EOR	Perform logical exclusive OR with A register
	MUI	Multiply integer with A register
	DVI	Divide integer into A register
Group B	STA	Store A register
	STQ	Store Q register
	JMP	Unconditional jump
	RTJ	Return jump
	RAO	Replace add one in storage
	SPA	Store A register, return parity to A register

Register Instructions

Operation Code	Definition
SLS	Selective stop
INP	Input to A register
OUT	Output from A register
ENA	Enter A register
ENQ	Enter Q register
INA	Increase A register
INQ	Increase Q register
NOP	No operation
EIN	Enable interrupt
IIN	Inhibit interrupt
EXI	Exit interrupt state
SPB	Set program protect bit
CPB	Clear program protect bit

Shift Instructions

ARS	A right shift
QRS	Q right shift
LRS	Long right shift (Q and A combined)
ALS	A left shift
QLS	Q left shift
LLS	Long left shift (Q and A combined)

Skip Instructions

SAZ	Skip if A=0
SAN	Skip if A≠0
SAP	Skip if A is positive
SAM	Skip if A is negative
SQZ	Skip if Q=0
SQN	Skip if Q≠0
SQP	Skip if Q is positive
SQM	Skip if Q is negative
SWS	Skip if switch is set
SWN	Skip if switch is not set
SOV	Skip on overflow

Operation Code	Definition
SNO	Skip on no overflow
SPE	Skip on storage parity error
SNP	Skip on no storage parity error
SPF	Skip on program protect fault
SNF	Skip on no program protect fault

Inter-Register Transfer Instructions

SET	Set specified register to ones
CLR	Clear specified register to zeros
TRA	Transfer A to specified register
TRM	Transfer M to specified register
TRQ	Transfer Q to specified register
TRB	Transfer both (Q+M) to specified register
TCA	Transfer complement of A to specified register
TCM	Transfer complement of M to specified register
TCQ	Transfer complement of Q to specified register
TCB	Transfer complement of both (Q+M) to specified register
AAM	Transfer arithmetic sum of A and M to specified register
AAQ	Transfer arithmetic sum of A and Q to specified register
AAB	Transfer arithmetic sum of A and both (Q+M) to specified register
EAM	Transfer exclusive or of A and M to specified register
EAQ	Transfer exclusive or of A and Q to specified register
EAB	Transfer exclusive or of A and both (Q+M) to specified register
LAM	Transfer logical product of A and M to specified register
LAQ	Transfer logical product of A and Q to specified register
LAB	Transfer logical product of A and both (Q+M) to specified register
CAM	Transfer complement of logical product of A and M to specified register
CAQ	Transfer complement of logical product of A and Q to specified register
CAB	Transfer complement of logical product of A and both (Q+M) to specified register

Note: + indicates an inclusive OR.

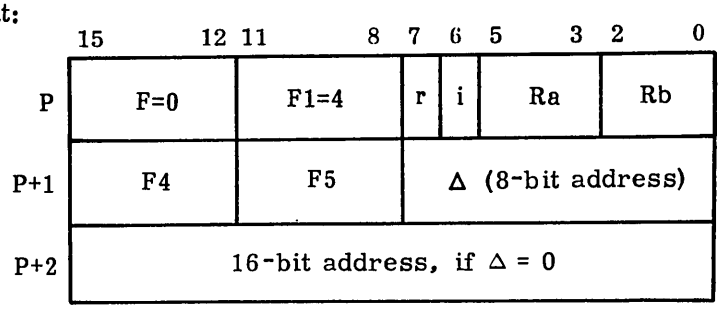
ENHANCED MACHINE INSTRUCTIONS

There are six classes of enhanced machine instruction codes:

- Type 2 storage reference
- Field reference
- Type 2 skip
- Decrement and repeat
- Type 2 inter-register reference
- Miscellaneous

Type 2 Storage Reference Instructions

Format:



SJE	Subroutine Jump Exit	P ◄— EA†
SJ1	Subroutine Jump	R1◄— Address of next instruction - 1, P ◄— EA
SJ2		R2◄— Address of next instruction - 1, P ◄— EA
SJ3		R3◄— Address of next instruction - 1, P ◄— EA
SJ4		R4◄— Address of next instruction - 1, P ◄— EA
SJQ		Q◄— Address of next instruction - 1, P ◄— EA
SJA		A◄— Address of next instruction - 1, P ◄— EA
SJI		I◄— Address of next instruction - 1, P ◄— EA
AR1	Add memory to register	R1◄— (R1) + (EA)
AR2		R2◄— (R2) + (EA)
AR3		R3◄— (R3) + (EA)
AR4		R4◄— (R4) + (EA)
ARQ		Q◄— (Q) + (EA)
ARA		A◄— (A) + (EA)
ARI		I◄— (I) + (EA)
SB1	Subtract memory from register	R1◄— (R1) - (EA)
SB2		R2◄— (R2) - (EA)
SB3		R3◄— (R3) - (EA)
SB4		R4◄— (R4) - (EA)
SBQ		Q◄— (Q) - (EA)
SBA		A◄— (A) - (EA)
SBI		I◄— (I) - (EA)

†EA is the effective address; registers are 1 through 4 (labeled as R1 through R4), Q, A, and I.

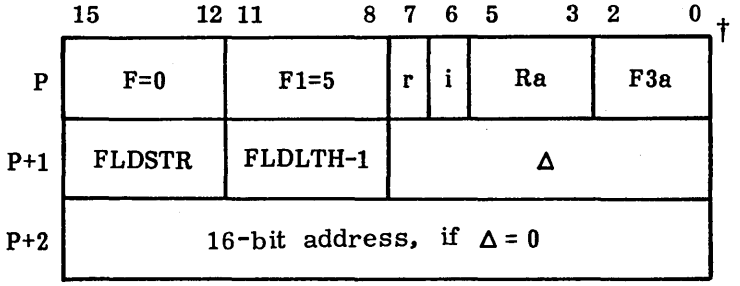
AN1 AN2 AN3 AN4 ANQ ANA ANI	AND memory to register	R1 ← (R1) • (EA) R2 ← (R2) • (EA) R3 ← (R3) • (EA) R4 ← (R4) • (EA) Q ← (Q) • (EA) A ← (A) • (EA) I ← (I) • (EA)
AM1 AM2 AM3 AM4 AMQ AMA AMI	AND register to memory	EA ← (EA)i • (R1), A ← (EA)i EA ← (EA)i • (R2), A ← (EA)i EA ← (EA)i • (R3), A ← (EA)i EA ← (EA)i • (R4), A ← (EA)i EA ← (EA)i • (Q), A ← (EA)i EA ← (EA)i • (A), A ← (EA)i EA ← (EA)i • (I), A ← (EA)i
LR1 LR2 LR3 LR4 LRQ LRA LRI	Load memory to register	R1 ← (EA) R2 ← (EA) R3 ← (EA) R4 ← (EA) Q ← (EA) A ← (EA) I ← (EA)
SR1 SR2 SR3 SR4 SRQ SRA SRI	Store register to memory	EA ← (R1) EA ← (R2) EA ← (R3) EA ← (R4) EA ← (Q) EA ← (A) EA ← (I)
LCA	Load character to A register	LS8† of A ← CHR† (EA), MS8† of A ← 0
SCA	Store character from A register to memory	CHR(EA) ← LS8 of A
OR1 OR2 OR3 OR4 ORQ ORA ORI	Inclusive OR memory to register	R1 ← (R1) v (EA) R2 ← (R2) v (EA) R3 ← (R3) v (EA) R4 ← (R4) v (EA) Q ← (Q) v (EA) A ← (A) v (EA) I ← (I) v (EA)
OM1 OM2 OM3 OM4 OMQ OMA OMI	Inclusive OR register to memory	R1 ← (EA)i v (R1), A ← (EA)i† R2 ← (EA)i v (R2), A ← (EA)i R3 ← (EA)i v (R3), A ← (EA)i R4 ← (EA)i v (R4), A ← (EA)i Q ← (EA)i v (Q), A ← (EA)i A ← (EA)i v (A), A ← (EA)i I ← (EA)i v (I), A ← (EA)i

†CHR is an eight-bit character.
MS8 is the most significant eight bits of the register.
LS8 is the least significant eight bits of the register.
(EA)i is the initial contents of EA prior to execution.

C1E	Compare register to memory equal	IF (R1).EQ.(EA), skip one location
C2E		IF (R2).EQ.(EA), skip one location
C3E		IF (R3).EQ.(EA), skip one location
C4E		IF (R4).EQ.(EA), skip one location
CQE		IF (Q).EQ.(EA), skip one location
CAE		IF (A).EQ.(EA), skip one location
CIE		IF (I).EQ.(EA), skip one location
CCE	Compare character from A register to memory equal	IF (LS8 † of A).EQ.(CHR of EA), skip one location

FIELD REFERENCE INSTRUCTIONS

Format:



SFZ	Skip if field zero	IF FLD(EA).EQ.0, skip one location†
SFN	Skip if field nonzero	IF FLD(EA).NE.0, skip one location
LFA	Load field to A	A ← 0, FLD(A) ← FLD(EA)
SFA	Store field from A	FLD(EA) ← FLD(A)
CLF	Clear field to zeros	FLD(EA) ← 0
SEF	Set field to ones	FLD(EA) ← 1

NOTE

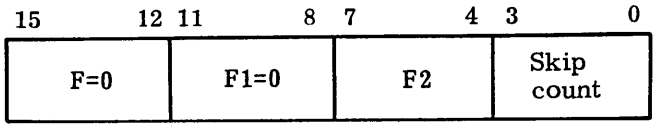
The following four instruction types use a single word as do the basic class instructions for F=0. However, the Δ field for addressing is replaced by counter and register fields. The register fields hold an address where it is needed.

†FLDSTR denotes the field start bit.
FLDLTH denotes the field length.
( ) denotes contents of.
EA denotes the effective address.
FLD denotes the field addressing.

Basic		Enhanced	
$01(F1)(SK)	Type 1 skip	Nothing comparable	
$00 - Δ	SLS	$00(F1)(SK)	Type 2 skips
$06 - Δ	SPB	$06(Ra/0(SK)	Decrement/repeat
$07 - Δ	CPB	$07(Ra/0)(0/Rb)	Type 2 inter-register
$0B00	NOP	$0B(Ra/0)(F3)	Miscellaneous

Type 2 Skip Instructions

Format:

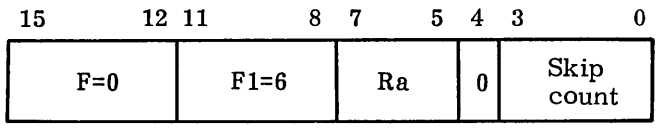


F2+SK≠0 (if 0, it is an SLS instruction)

S4Z	Skip if register zero	IF (R4).EQ.0, skip SK+1 locations
S1Z		IF (R1).EQ.0, skip SK+1 locations
S2Z		IF (R2).EQ.0, skip SK+1 locations
S3Z		IF (R3).EQ.0, skip SK+1 locations
S4N	Skip is register nonzero	IF (R4).NE.0, skip SK+1 locations
S1N		IF (R1).NE.0, skip SK+1 locations
S2N		IF (R2).NE.0, skip SK+1 locations
S3N		IF (R3).NE.0, skip SK+1 locations
S4P	Skip if register positive	IF (R4).GE.0, skip SK+1 locations
S1P		IF (R1).GE.0, skip SK+1 locations
S2P		IF (R2).GE.0, skip SK+1 locations
S3P		IF (R3).GE.0, skip SK+1 locations
S4M	Skip if register negative	IF (R4).LT.0, skip SK+1 locations
S1M		IF (R1).LT.0, skip SK+1 locations
S2M		IF (R2).LT.0, skip SK+1 locations
S3M		IF (R3).LT.0, skip SK+1 locations

Decrement and Repeat Instructions

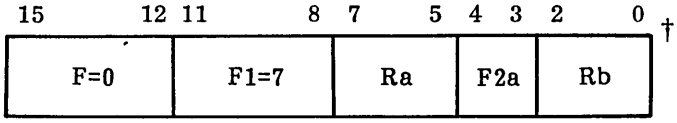
Format:



D1P	Decrement and repeat if positive	R1 ←— (R1)-1, IF (R1).GE.0, go back SK locations
D2P		R2 ←— (R2)-1, IF (R2).GE.0, go back SK locations
D3P		R3 ←— (R3)-1, IF (R3).GE.0, go back SK locations
D4P		R4 ←— (R4)-1, IF (R4).GE.0, go back SK locations
DQP		Q ←— (Q)-1, IF (Q).GE.0, go back SK locations
DAP		A ←— (A)-1, IF (A).GE.0, go back SK locations
DIP		I ←— (I)-1, IF (I).GE.0, go back SK locations

Type 2 Inter-Register Instructions

Format:

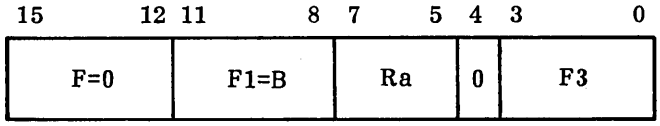


†

		Rb	Ra
XF1	Transfer register to register	R ←	(R1), where R=1,2,3,4,Q,A, or I
XF2		R ←	(R2), where R=1,2,3,4,Q,A, or I
XF3		R ←	(R3), where R=1,2,3,4,Q,A, or I
XF4		R ←	(R4), where R=1,2,3,4,Q,A, or I
XFQ		R ←	(Q), where R=1,2,3,4,Q,A, or I
XFA		R ←	(A), where R=1,2,3,4,Q,A, or I
XFI		R ←	(I), where R=1,2,3,4,Q,A, or I

Miscellaneous Instructions

Format:



LMM	Load micro memory	Ra=0
LRG	Load registers	
SRG	Store registers	
SIO	Set/sample input or output	
SPS	Sample port/status	
DMI	Define micro interrupt	
CBP	Clear breakpoint interrupt	
GPE	Generate character parity even	
GPO	Generate character parity odd	
ASC	Scale accumulator	
LUB	Load upper unprotected bounds	RA registers 1 through 7
LLB	Load lower unprotected bounds	
EMS	Execute micro sequence	

† F2a=0
R a values: 1-7

PSEUDO INSTRUCTIONS

There are six classes of pseudo instructions.

- Subprogram linkage
- Data storage
- Constant declaration
- Assembler control
- Listing control
- Macro definition

Subprogram Linkage

Operation Code	Definition
NAM	Identify source language subprogram
END	End source language subprogram
ENT	Designate internal entry point names
EXT	Designate external entry point names
EXT*	Designate relative external entry point names

Data Storage

BSS	Define a block of storage starting at symbol
BZS	Define a block of zero storage
COM	Define a block of common storage
DAT	Define a block of data storage

Constant Declarations

ADC	Store address constants
ADC*	Store relative address constants
ALF	Store an alphanumeric message
NUM	Store numeric constants
DEC	Convert and store decimal constants in fixed point format
VFD	Variable field definition and storage

Assembler Control

Operation Code	Definition
EQU	Equate symbols to addresses
ORG	Defines origin for assembly of instructions following ORG
ORG*	Terminate ORG
IFA	If condition is true, assemble following instructions
EIF	Terminate IFA (or IFC macro pseudo instruction)
OPT	Signal input of control options
MON	Return control to operating system

Listing Control

NLS	Inhibit list output
LST	Resume list output after NLS
SPC	Space lines on list output
EJT	Eject page on list output

Macro Definition

MAC	Specify name of macro
EMC	End macro definition
LOC	Define local symbolic labels
IFC	If condition is true, assemble following instructions in macro

PROGRAMMING CONSIDERATIONS B

CODING TECHNIQUES

The following limitations should be observed when coding programs to run under MSOS in 65K mode.

All 16 bits of an address word are needed in order to address all of available core. This means that bit 15 can no longer be used to indicate the conditions it can be used for in a 32K mode system.

Multilevel indirect addressing cannot be used in 65K mode which signifies that instructions of the following form can no longer be used.

```
ADC     (TAG)
LDA+    (TAG)
```

If relative addresses are generated, the following instruction is allowed.

```
LDA     (TAG)
```

The instruction

```
ADC     (TAG)
```

is allowed in 65K mode if there are no storage instructions that make indirect reference to this location and the program containing this expression is never loaded into part 1 of a 65K system.

ASCII CODES C

The 1963 Control Data Subset of ASCII (CDC-STD 1.10.003, Revision C) is used by the macro assembler. ASCII code uses eight bits; the eighth bit, which is always zero, is omitted in the following table.

ASCII Symbol	Bit Configuration	Hexadecimal Number	Meaning
NULL	000 0000	0	Null/idle
SOM	000 0001	1	Start of message
EOA	000 0010	2	End of address
EOM	000 0011	3	End of message
EOT	000 0100	4	End of transmission
WRU	000 0101	5	Who are you
RU	000 0110	6	Are you
BELL	000 0111	7	Audible signal
FE_0	000 1000	8	Format effector
HT/SK	000 1001	9	Horizontal tab/skip (punched card)
LF	000 1010	A	Line feed
V_{TAB}	000 1011	B	Vertical tabulation
FF	000 1100	C	Form feed
CR	000 1101	D	Carriage return
SO	000 1110	E	Shift out
SI	000 1111	F	Shift in
DC_0	001 0000	10	Device control/data link escape
DC_1	001 0001	11	Device controls
DC_2	001 0010	12	Device controls
DC_3	001 0011	13	Device controls
DC_4 (STOP)	001 0100	14	Device control/stop
ERR	001 0101	15	Error
SYNC	001 0110	16	Synchronous idle
LEM	001 0111	17	Logical end of media

ASCII Symbol	Bit Configuration	Hexadecimal Number	Meaning
S_0	001 1000	18	Information separators
S_1	001 1001	19	
S_2	001 1010	1A	
S_3	001 1011	1B	
S_4	001 1100	1C	
S_5	001 1101	1D	
S_6	001 1110	1E	
S_7	001 1111	1F	
Δ	010 0000	20	Word separator (space)
!	010 0001	21	Exclamation point
"	010 0010	22	Quotation mark
#	010 0011	23	Number
$	010 0100	24	Dollar sign (hexadecimal)
%	010 0101	25	Percent
&	010 0110	26	Ampersand
' (APOS)	010 0111	27	Apostrophe
(	010 1000	28	Left parenthesis
)	010 1001	29	Right parenthesis
*	010 1010	2A	Asterisk
+	010 1011	2B	Plus
, (Comma)	010 1100	2C	Comma
-	010 1101	2D	Minus
.	010 1110	2E	Decimal point or period
/	010 1111	2F	Slash
0	011 0000	30	Numbers
1	011 0001	31	
2	011 0010	32	
3	011 0011	33	
4	011 0100	34	
5	011 0101	35	
6	011 0110	36	
7	011 0111	37	
8	011 1000	38	
9	011 1001	39	
:	011 1010	3A	Colon
;	011 1011	3B	Semi-colon

ASCII Symbol	Bit Configuration	Hexadecimal Number	Meaning
<	011 1100	3C	Less than
=	011 1101	3D	Equals
>	011 1110	3E	Greater than
?	011 1111	3F	Question mark
@	100 0000	40	Each
A	100 0001	41	
B	100 0010	42	
C	100 0011	43	
D	100 0100	44	
E	100 0101	45	
F	100 0110	46	
G	100 0111	47	
H	100 1000	48	
I	100 1001	49	
J	100 1010	4A	
K	100 1011	4B	
L	100 1100	4C	
M	100 1101	4D	Letters
N	100 1110	4E	
O	100 1111	4F	
P	101 0000	50	
Q	101 0001	51	
R	101 0010	52	
S	101 0011	53	
T	101 0100	54	
U	101 0101	55	
V	101 0110	56	
W	101 0111	57	
X	101 1000	58	
Y	101 1001	59	
Z	101 1010	5A	
[	101 1011	5B	Left bracket
\	101 1100	5C	Reverse slant
]	101 1101	5D	Right bracket

ASCII Symbol	Bit Configuration	Hexadecimal Number	Meaning
↑	101 1110	5E	Up arrow (exponentiation)
←	101 1111	5F	Left arrow (replaced by)
ACK	111 1100	7C	Acknowledge
⑦	111 1101	7D	Unassigned control
ESC	111 1110	7E	Escape
DEL	111 1111	7F	Delete/idle

The numbers between 5F and 7C have no ASCII code assigned to them.

MACRO ASSEMBLER ERRORS D

MESSAGE	SIGNIFICANCE
xxxxyy**********	Format for pass 1 and 2 error messages Where: xxxx is a 4-digit line number. yy is a 2-character error code (explained below).
******yy**********	Format for pass 3 error messages. If the L option is selected, errors in pass 3 precede the source line on the list output. If L is not selected, error messages are output on the standard comment unit.
ABS BASE ERR	Assembler was loaded at a different location from where it was absolutized.
**DS	Double defined symbol; a name in: • The location field of a machine instruction or an ALF, NUM, or ADC pseudo instruction; or • The address field of an EQU, COM, DATA, EXT, BSS or a BZS pseudo instruction.
**EX	Illegal expression, either: • No forward referencing of some symbolic operands; or • No relocation of certain expression values; or • A violation of relocation; or • Illegal register reference; or • A symbol other than Q, 1, or B is specified.
INPUT ERROR	An error was returned by driver when doing a Read.
**LB	Numeric or symbolic label contains illegal character. The label is ignored.
MASS STORAGE OVERFLOW	Not enough room for input image on mass storage.
**MC	Macro call error, • Illegal parameter list • No continuation card where one was indicated.

MESSAGE	SIGNIFICANCE
**MD	Macro definition error.
**MO	Overflow of load-and-go area; affects only X option.
**NN	Missing or misplaced NAM statement.
**OP	Illegal operation code, either: • Illegal symbol in operation code field; or • Illegal operation code terminator.
**OV	Numeric constant or operand value is greater than allowed.
**PP	Error in previous pass of compilation assembly. See output page immediately preceding first page of listing for pass 1 or pass 2 error message.
**RL	Illegal relocation, either: • Violation of relocation; or • Violation of a rule for instructions that requires the expression value to either be absolute or have no forward referencing of symbolic operands.
**SQ	Sequence error — Tags instructions with sequence numbers that are out of order. This is not fatal and is not counted in the number of errors reported at the bottom of the symbol table.
**UD	An undefined symbol in an address expression.

INSTRUCTION CODES E

Most pseudo instructions may be placed anywhere in a source language subprogram. Exceptions: The first statement of a subprogram must be OPT or NAM; the last statement must be MON or END.

Subprogram Linkage

NAM s	s is the symbolic subprogram name
END s	Ends subprogram; s is the subprogram to be entered at this point.
ENT $s_1, s_2, \ldots$	Entry points in this subprogram used by other subprograms
EXT $s_1, s_2, \ldots$	Entry points in other subprograms used by this subprogram (absolute location)
EXT* $s_1, s_2, \ldots$	Same as EXT except locations are relative

Data Storage

BSS $s_1(e_1), s_2(e_2), \ldots$	Allocates local subprogram data storage; s_i is the symbolic name of the data block, e_i is the symbolic name of the block length in words.
BZS $s_1(e_1), s_2(e_2), \ldots$	Same as BSS except data block is zeroed at assignment time.
COM $s_1(e_1), s_2(e_2), \ldots$	Allocates block of common storage; s_i and e_i are defined as in BSS.
DAT $s_1(e_1), s_2(e_2), \ldots$	Allocates common block within the program area; s_i and e_i are defined as in BSS. Block words may be preset using ORG instruction.

Constant Declarations

s ADC $e_1, e_2, (e_3), \ldots, e_n$	Set address to constant expression; s is name in location field, e_i are constants or address expressions, (e_i) sets bit 15; results are started in consecutive locations.
s ADC* $e_1, \ldots$	Same as ADC, except relative addressing is used.
s ALF n, message	Translates message to ASCII; s is name in location field, n is number of words in message (2n is number of characters). Number of characters is limited to the number of characters available in the comment field of the source statement; i.e., 72-(location + instruction + address + 2).

s NUM ...k_i,...	Defines numeric constants; s is the name in the location field, k_i is the specified integer with $-7FFF_{16} \leq k \leq 7FFF_{16}$; results are stored in consecutive locations.
s DEC ...k_i,...	Converts decimal constants to fixed point binary; s and k are defined as in NUM instruction.
s VFD ...$m_i n_i / v_i$,...	Variable field definition. Packs data as bit strings into consecutive locations (computer word boundaries are ignored). • s is the name in the location field • m_i specifies data mode (N = numeric constant in range $\pm 7FFF_{16}$, A = character in eight-bit bytes, X = expression) • n is the number of bits • v is the value of data

Assembler Control

EQU ...$s_i(e_i)$,...	Equates symbolic name to expression value; s_i is the symbolic name, e_i is the expression.
ORG e	Sets location counter to address generated by expression e (absolute location).
ORG* e	Same as ORG but address is relative.
s IFA e_i, c, e_2	Assembles set of coding lines only if specified condition occurs; s is name in location field, e_1 and e_2 are expressions to be compared, c is condition (EQ is =, NE is ≠, GT is >, LT is <).
EIF s	Terminates IFA or IFC statement if condition fails and coding lines are skipped; s is the name in the location field.
OPT	Control options will be input to assembler
MON	Returns control to the operating system

Listing Control

NLS	Inhibits list output
LST	Initiates list output after NLS inhibits it.
SPC e	Controls line spacing; e is number of lines to be skipped.
EJT	Eject page

Macro

s MAC P_1, P_2...	Names the macro statement and lists its formal parameters; s is symbolic name of macro, P_i is symbolic name for a macro parameter.
EMC	Ends the macro definition

LOC $s_1, s_2, \ldots$	Localizes parameters in macro so same symbol may be used in macro and in programs; s_i corresponds to P_i in the MAC statement. LOC must immediately follow MAC statement.
s IFC e_1, c, e_2	Allows condition internal to macro. Same as IFA in program except only EQ or NE are allowed.

MACRO LIBRARY F

Formatting Macros

HEXASC/HEXASC*	Converts hexadecimal to ASCII
HEXDEC/HEXDEC*	Converts hexadecimal to decimal
ASCII/ASCII*	Converts two ASCII characters to a hexadecimal integer
DECHEX/DECHEX*	Converts three words of ASCII numbers to a hexadecimal integer
FLOATG/FLOATG*	Converts two-word FP variable to FP with exponent in ASCII
ENCODE/ENCODE*	Codes buffer in the format given (ASCII)
DECODE/DECODE*	Decodes ASCII from buffer into variable list in machine code

File Manager Macros

FLDF	Defines parameters of file
DEFFIL	Defines file
DEFIDX	Additional definitions of the file
LOKFIL	Locks the file
UNLFIL	Unlocks the file
RELFIL	Releases file
STOSEQ	Stores record sequentially in file
STOIDX	Stores record by index
STODIR	Stores directly into a file
RTVSEQ	Retrieve sequential record
RTVDIR	Retrieve record directly from file
RTVIDX	Retrieve record by index
RTVIDO	Retrieve record using ordered index
STATFL	Finds status file and executes error routine if necessary

Monitor Request Macros

READ	Read unformatted data (word mode) from I/O device
FREAD	Read formatted data (sector mode) from I/O device
WRITE	Write unformatted data (word mode) to I/O device
FWRITE	Write formatted data (sector mode) to I/O device
INDIR	Allows indirect execution of another request
TIMER	Delays scheduling of request
SCHDLE	Queues requests for execution
MOTION	Direct motion on an I/O device (e.g., seek on disk)
SPACE	Allocates space in core for protected programs
RELEAS	Releases core space allocated by SPACE request
DISCHD	Disables specified System Directory program
ENSCHD	Enables System Directory program disabled by DISCHD instruction
TIMPT1	Schedules System Directory programs in parts 0 and 1 of the CPU
PTNCOR	Allocates block of partitioned core
SYSCHD	Schedules System Directory programs in part 0 or 1 for protected programs
CORE	Sets or determines bounds of unprotected core
LOADER	Executes MS resident relocatable binary loader for unprotected programs executing at level zero
GTFILE	Accesses permanent files in the program library
STATUS	Determines status of I/O device from checking physical device table
EXIT	Signals job completion by an unprotected program

Miscellaneous Macros

VOLA	Allocates volatile storage
VOLR	Releases volatile storage
CLOCK	Finds real time
DISP	Causes current program to exit to dispatcher
BUFFER	Creates a physical device table for the software buffer

INDEX

Absolute addressing 2-1, 3
ADC/ADC* 3-7; A-9
ADD arithmetic instruction 2-7
Address expression 1-3, 5
Address field 1-2
Address modes 2-1, 10.1
    Absolute 2-1, 3
    Constant 2-1, 6
    Relative 2-1, 4
ADQ arithmetic instruction 2-7
ALF 3-7; A-9
AMr 2-10.5
AND instruction 2-8
ANr 2-10.5
Arithmetic instructions 2-7
    ADD 2-7
    ADQ 2-7
    DVI 2-7
    MUI 2-7
    RAO 2-7
    SUB 2-7
Arithmetic operators 1-5
Arithmetic sum 2-13
ARr 2-10.5
ASCII codes Appendix C
ASCII formatting macro 5-2
Assembler Control 3-13; A-10
    EIF 3-16
    EQU 3-13
    IFA 3-15
    MON 3-17
    OPT 3-17
    ORG/ORG* 3-14
Assembler output 6-1
Assembler passes
    Pass 1 v, 6-2
    Pass 2 v, 6-2
    Pass 3 v, 6-2
Assembly listing 6-2
    Cross-reference 6-3
    Error 6-2
    Sample assembly 6-4
    Sample source program 6-4
Asterisk 1-2, 5

BUFFER macro 5-15
BSS 3-4; A-9
BZS 3-4; A-9

C option 6-1
CCE 2-10.6
CLOCK macro 5-15
COM 3-5
Comment field 1-8
Common storage 1-3
Complement logical product 2-12
Constant addressing 2-1, 6
Constant declarations 3-7; A-9
    ADC/ADC* 3-7
    ALF 3-7
    DEC 3-10
    NUM 3-9
    VFD 3-11
Control options 6-1
    C 6-1
    L 6-1
    P 6-1
    X 6-1
CORE monitor request macro 5-13
CPB register reference instruction 2-11
CrE 2-10.6
Cross-reference listing 6-3

DAT 3-6; A-9
Data storage 1-3; 3-4; A-9
    BSS 3-4
    BZS 3-4
    COM 3-5
    DAT 3-6
Data transmission instructions 2-6
    LDA 2-7
    LDQ 2-7
    SPA 2-7
    STA 2-6
    STQ 2-6
DEC 3-10; A-9
DECHEX formatting macro 5-2
DECODE formatting macro 5-3
Decrement and repeat instructions 2-1, 18
DEFFIL file manager macro 5-4
DEFIDX file manager macro 5-4
Diagnostics (see MSOS Diagnostic Handbook)
DISCHD monitor request macro 5-11
DISP macro 5-15
DVI arithmetic instruction 2-7

EIF 3-16; A-10
EIN register reference instruction 2-11
EJT 3-18; A-10
EMC 4-2; A-10
ENA register reference instruction 2-11
ENCODE formatting macro 5-3
END subprogram linkage 3-1; A-9
Enhanced instructions 2-1
ENQ register reference instructions 2-11
ENSCHD monitor request macro 5-12
ENT subprogram linkage 3-2; A-9
EOR 2-8
Error codes 5-1
Error listing 6-2
EQU 3-13; A-10
Evaluation hierarchy 1-6
Exclusive OR 2-8, 12
EXI register reference instruction 2-11
EXIT monitor request macro 5-14
EXT/EXT* 3-2; A-9
External name 1-5

Field reference instructions 2-1, 19
File manager macros 5-3
    DEFFIL 5-4
    DEFIDX 5-4
    FLDF 5-4
    LOKFIL 5-5
    RELFIL 5-5
    RTVDIR 5-6
    RTVIDO 5-7
    RTVIDX 5-7
    RTVSEQ 5-6
    STATFL 5-7
    STODIR 5-5
    STOIDX 5-5
    STOSEQ 5-5
    UNFIL 5-5
FLDF file manager macro 5-4
FLOATG formatting macro 5-3
Formatting macros 5-2
    ASCII 5-2
    DECHEX 5-2
    DECODE 5-3
    ENCODE 5-3
    FLOATG 5-3
    HEXASC 5-2
    HEXDEC 5-2
FREAD monitor request macro 5-8
FWRITE monitor request macro 5-8

Group A storage reference instructions 2-1
Group B storage reference instructions 2-1
GTFILE monitor request macro 5-13

HEXASC formatting macro 5-2
HEXDEC formatting macro 5-2

IFA 3-15; A-10
IFC 4-2; A-10
IIN register reference instruction 2-11
INA register reference instruction 2-11
Index characters 1-7, 8
INDIR monitor request macro 5-9
INP register reference instruction 2-10.7
INQ register reference instruction 2-11
Instruction format 1-1
    Source program 1-1
    Source statement 1-1
        Address field 1-2
        Comment field 1-8
        Instruction 1-2
        Location field 1-1
        Sequence field 1-8
Inter-register instructions 2-12
Inter-register mnemonics 2-13
Inter-register transfer instructions A-3

Jump instructions 2-9

L option 6-1
LCA 2-10.6
LIBMAC 5-1
Listing control 3-18; A-10
    EJT 3-18
    LST 3-18
    NLS 3-18
    SPC 3-18
LDA data transmission instruction 2-6
LDQ data transmission instruction 2-6
LOADER monitor request macro 5-13
LOC 4-2; A-10
Location field 1-1
Logical instructions 2-8
Logical product 2-12
LOKFIL file manager macro 5-5
LRr 2-10.6
LST 3-18; A-10

MAC 4-1; A-10
Machine instructions 2-1; A-1
    Inter-register 2-12; A-3
    Register reference 2-10; A-2
    Shift 2-14; A-2
    Skip 2-15; A-2
    Storage reference 2-1; A-1

Macro assembler errors Appendix D
Macro definition instructions A-10
Macro instructions 4-4
    Actual parameters 4-4
    Examples 4-6
    Null parameters 4-5
Macro library 5-1
    Creating 5-1
    Modifying 5-2
    Programs 5-2
Macro pseudo instructions 4-1; A-9
Macro skeleton 4-3
Macros 4-1
    EMC 4-2
    IFC 4-2
    LOC 4-2
    MAC 4-1
Miscellaenous instructions 2-1,20
Mnemonic instruction codes Appendix A
MON 1-1; 3-17; A-10
Monitor request macros 5-8
    CORE 5-13
    DISCHD 5-11
    ENSCHD 5-12
    EXIT 5-14
    FREAD 5-8
    FWRITE 5-8
    GTFILE 5-13
    INDIR 5-9
    LOADER 5-13
    MOTION 5-9
    PTNCOR 5-12
    READ 5-8
    RELEAS 5-11
    SCHDLE 5-9
    SPACE 5-11
    STATUS 5-14
    SYSCHD 5-12
    TIMER 5-9
    TIMPT1 5-12
    WRITE 5-8
MOTION monitor request macro 5-9
MUI arithmetic instruction 2-7

NAM subprogram linkage 3-1; A-9
Negative overflow/zero set 2-26
NLS 3-18; A-10
NOP register reference instruction 2-11
NUM 3-9; A-9
Numeric operand 1-4

OMr 2-10.6
Operand 1-2.1
Operation code field 1-2
OPT 3-17; A-10
ORG/ORG* 3-14; A-10
ORr 2-10.6
OUT register reference instruction 2-10.7

P option 6-1
Parenthesis 1-6
Parameters 4-4
    Actual 4-4
    Null 4-5
Programming considerations Appendix B
Program storage 1-3
Pseudo instructions 3-1; A-9
    ADC/ADC* 3-7
    ALF 3-7
    BSS 3-4
    BZS 3-4
    COM 3-5
    DAT 3-6
    DEC 3-10
    EIF 3-16
    EJT 3-18
    END 3-1
    ENT 3-2
    EQU 3-13
    EXT/EXT* 3-2
    IFA 3-15
    LST 3-18
    MON 3-17
    NAM 3-1
    NLS 3-18
    NUM 3-9
    OPT 3-17
    ORG/ORG* 3-14
    SPC 3-18
    VFD 3-11
PTNCOR monitor request macro 5-12

RAO arithmetic instruction 2-7
READ monitor request macro 5-8
Register reference instructions 2-1, 10
    CPB 2-11
    EIN 2-11
    ENA 2-11
    ENQ 2-11
    EXI 2-11
    IIN 2-11
    INA 2-11
    INP 2-10.7
    INQ 2-11
    NOP 2-11
    OUT 2-10.7
    SLS 2-10.7
    SPB 2-11

Relative addressing 2-1, 4
RELEAS monitor request macro 5-11
RELFIL file manager macro 5-5
Remarks 1-2
Return jump 2-9
RTVDIR file manager macro 5-6
RTVIDO file manager macro 5-7
RTVIDX file manager macro 5-7
RTVSEQ file manager macro 5-6

Sample listing 6-4
Sample program 6-4
SBr 2-10.5
SCA 2-10.6
SCHDLE monitor request macro 5-9
Sequence field 1-8
Shift instructions 2-14; A-2
SJE 2-10.3
SJr 2-10.3
Skip instructions 2-15; A-2
Slash 1-5
SLS register reference instruction 2-10.7
Source program 1-1
Source statement 1-1
SPA data transmission instruction 2-7
SPACE monitor request macro 5-11
SPB register reference instruction 2-11
SPC 3-18; A-10
Special characters 1-7
  Index 1-8
  Register 1-7
  Storage 1-7
SRr 2-10.6
STA data transmission instruction 2-6
Statement label 1-1
STATFL file manager macro 5-7
STATUS monitor request macro 5-14
STODIR file manager macro 5-5
STOIDX file manager macro 5-5
Storage characters 3-4
Storage reference instructions 2-1; A-1
  Absolute addressing 2-3
  Address modes 2-1
  Arithmetic 2-7
  Constant addressing 2-6
  Data transmission 2-6
  Jump 2-9
  Logical 2-8
  Machine language format 2-2
  Relative addressing 2-4
STOSEQ file manager macro 5-5
STQ data transmission instruction 2-6
SUB arithmetic instruction 2-7
Subprogram linkage 3-1; A-9
  END 3-1
  ENT 3-2
  EXT/EXT* 3-2
  NAM 3-1
Symbolic name 1-2, 3
Symbolic operand 1-2.1
SYSCHD monitor request macro 5-12

TABLST v
TIMER monitor request macro 5-9
TIMPT1 monitor request macro 5-12
Type 1 inter-register instructions 2-12
Type 1 skip instructions 2-15
Type 2 inter-register instructions 2-1, 14
Type 2 skip instructions 2-1
Type 2 storage addressing relationships 2-10.4
Type 2 storage reference instructions 2-1, 10

UNFIL file manager macro 5-5

VFD 3-11; A-9
VOLA 5-15
VOLR 5-15

WRITE monitor request macro 5-8

X option 6-1
XREF vi

COMMENT SHEET

MANUAL TITLE Macro Assembler Reference Manual

PUBLICATION NO. 60361900 REVISION G

FROM NAME:

BUSINESS
ADDRESS:

COMMENTS: This form is not intended to be used as an order blank. Your evaluation of this manual will be welcomed by Control Data Corporation. Any errors, suggested additions or deletions, or general comments may be made below. Please include page number.

CUT ALONG LINE

STAPLE

STAPLE

FOLD

FIRST CLASS
PERMIT NO. 333

LA JOLLA, CA.

BUSINESS REPLY MAIL
NO POSTAGE STAMP NECESSARY IF MAILED IN U.S.A.

POSTAGE WILL BE PAID BY

CONTROL DATA CORPORATION
PUBLICATIONS AND GRAPHICS DIVISION
4455 EASTGATE MALL
LA JOLLA, CALIFORNIA 92037

CUT ALONG LINE

FOLD

STAPLE

STAPLE

CORPORATE HEADQUARTERS, P.O. BOX 0, MINNEAPOLIS, MINN. 55440
SALES OFFICES AND SERVICE CENTERS IN MAJOR CITIES THROUGHOUT THE WORLD

LITHO IN U.S.A.

CONTROL DATA CORPORATION