CONTROL DATA
CORPORATION

CONTROL DATA®
1700 COMPUTER SYSTEMS

1700 MSOS 4 MACRO ASSEMBLER REFERENCE MANUAL

REVISION RECORD

REVISION	DESCRIPTION
A	Original printing for Macro Assembler Version 3.0 and MSOS Version 4.0.
(4/72)	
B	General revision and update for compatibility with MSOS 4.1. Appendix D added.
(3/74)	

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
Small Computer Development Division
4455 Eastgate Mall
La Jolla, California 92037

or use Comment Sheet in the back of this manual.

PREFACE

The Macro Assembler for the CONTROL DATA® 1700 Computer System is a 3-pass assembler which 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 output 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.

Refer to the 1700 Mass Storage Operating System (MSOS) 4 Reference Manual equipment configuration for the minimum hardware required by the Macro Assembler.

It is assumed that users of this manual are familiar with the 1700 MSOS.

Following is a list of 1700 MSOS manuals.

Description	Pub. No.
1700 MSOS 4 Reference Manual	60361500
1700 MSOS 4 Mass Storage FORTRAN Version 3 A/B Reference Manual	60362000
1700 MSOS 4 Computer System Codes	60163500
1700 MSOS 4 Macro Assembler General Information	39519800
1700 MSOS 4 Mass Storage FORTRAN General Information	39519900
1700 MSOS 4 Small Computer Maintenance Monitor Reference Manual	39520200
1700 MSOS 4 Instant	39520500
1700 MSOS 4 File Manager Version 1 Reference Manual	39520600
1700 MSOS 4 Release Bulletin	39520800
1700 MSOS 4 Installation Handbook	39520900
1700 MSOS 4 Small Computer Maintenance Monitor Instant	39521700
1700 MSOS 4 Ordering Bulletin	39521900
1700 MSOS 4 General Information	39522400

CONTENTS

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 (1700 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. 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.

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 1 to 6 alphanumeric characters; the first must be alphabetic. Characters in excess of six are ignored. A 2-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 1 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 6 characters. Certain instructions may be followed by a 1-character terminator.

The mnemonic instruction codes listed in Appendix A are described in the 1700 Computer Reference Manual.

1.2.4 ADDRESS FIELD

The address field begins to the right of the operation code field, separated from it by at least 1 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).

This field contains an address 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 which 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 which 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), I(DAT1)
P                   BZS             A,B,C
P                   BZS             J,K(10)
```

The symbols D, E, G, and H are absolute; DAT1, DAT2, and I 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 word instruction, an asterisk references the location of the first word. Special character I refers to location FF_{16}. I is the only indexing character that may stand alone as an operand with storage reference instructions. It may not be defined as a location symbol in a program.

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 classes.

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

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

The function of each machine instruction is discussed in detail in the 1700 Computer Reference Manual. Appendix A lists the machine instructions in the order in which they are discussed in this chapter.

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.

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.

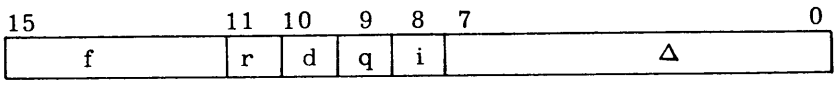
Character	Description
*	Asterisk as the last character of operation code specifies relative addressing in a one-word instruction
-	Minus as the last character of operation code specifies absolute addressing in a one-word instruction
+	Plus as the last character of operation code specifies absolute addressing in a two-word instruction
=	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 described in 1700 Computer Reference Manual

r Specifies relative addressing

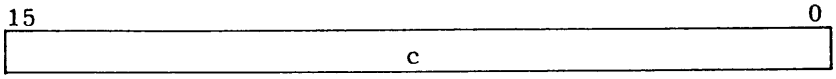
d Specifies indirect addressing

q Index register 1 flag; specifies adding contents of Q register to address

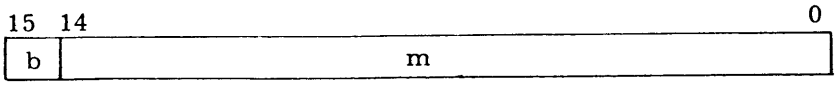
i Index register 2 flag; specifies adding contents of storage register FF_{16} to address

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.

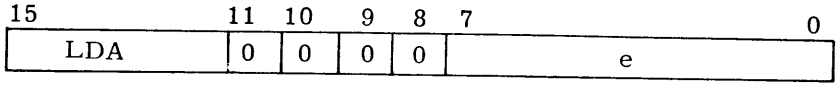
Examples:

One Word, Direct

Instruction:

LDA- e

Machine Word:

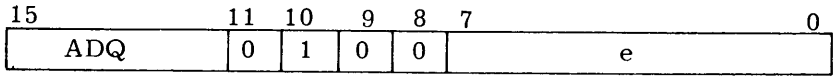


One Word, Indirect

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.

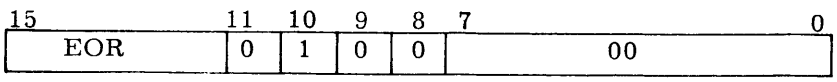
Examples:

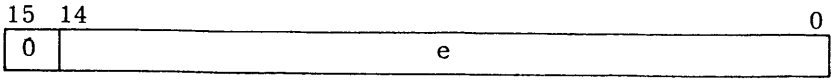
Two Word, Direct

Instruction:

EOR+ e

Machine Words:



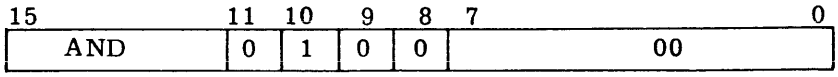


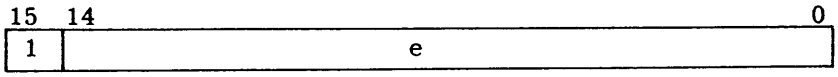
Two Word, Indirect

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.

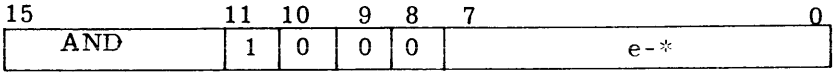
Examples:

One Word, Direct

Instruction:

AND* e

Machine Word:

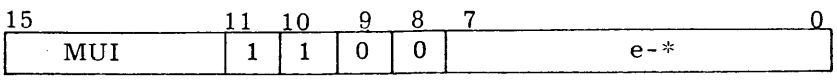


One Word, Indirect

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.

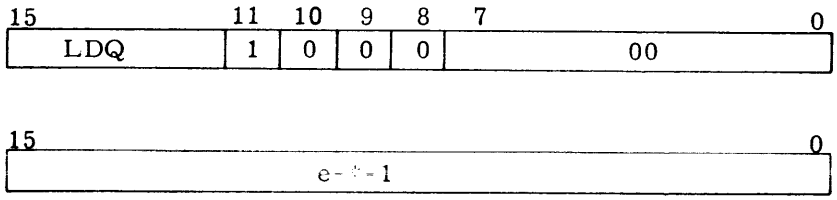
Examples:

Two Word, Direct

Instruction:

```
LDQ    e
```

Machine Words:

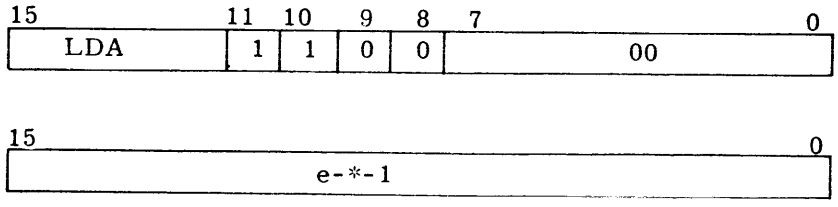


Two Word, Indirect

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)

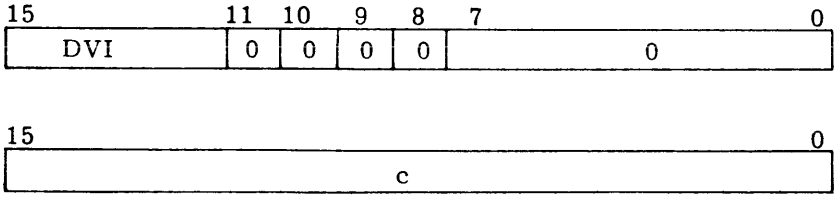
An instruction containing a constant in the address field results in two machine words.

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 $\forecolon$ 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.

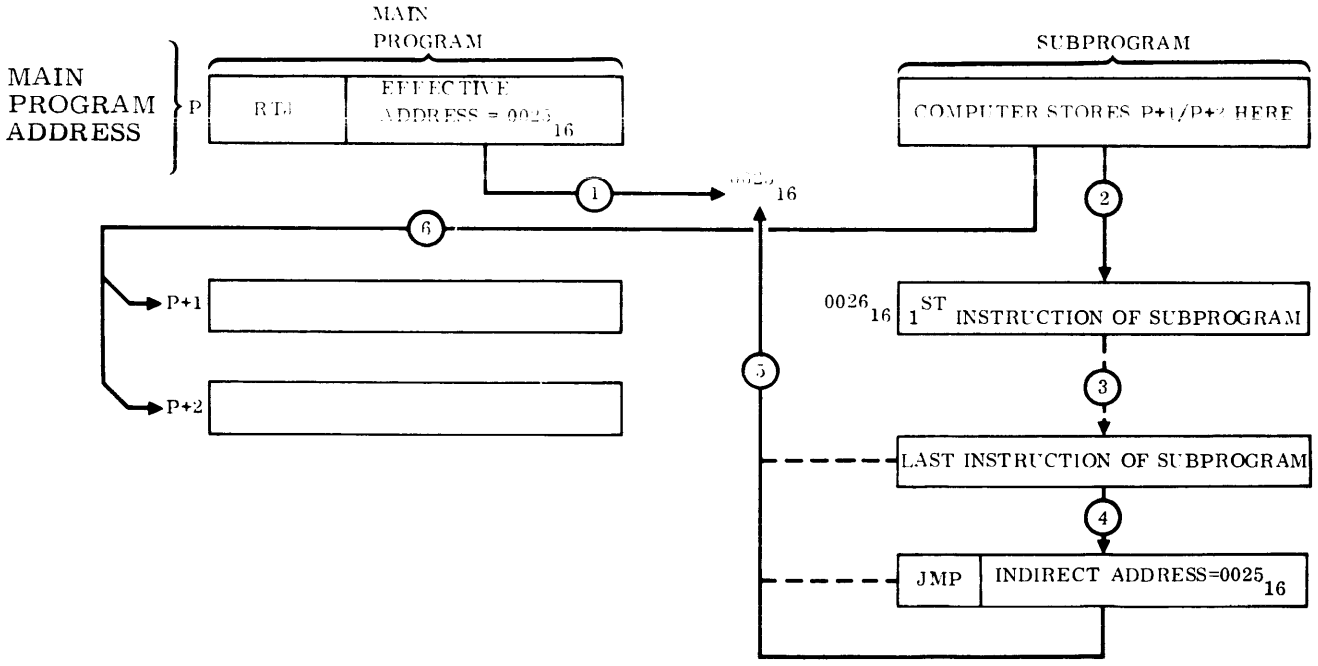


Figure 2-1. Program Using Return Jump Instruction

*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).

A Return Jump 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.

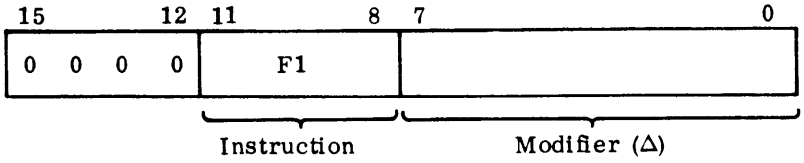
JMP (F = 1) Jump. Jump to the address specified by the effective address. This effectively replaces the contents of program address counter P with the effective address specified in the Jump instruction.

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.

2.2 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. Refer to the 1700 Computer Reference Manual.

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. Refer to the 1700 Computer Reference Manual.

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). Compares to Selective Stop instruction with the STOP switch off.

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 Q 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 (refer to the 1700 Computer Reference Manual). If an instruction is illegal, it becomes a selective stop and an interrupt on program protect fault is possible (if selected).

- Protect switch on — Selective stop unless 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 Q.

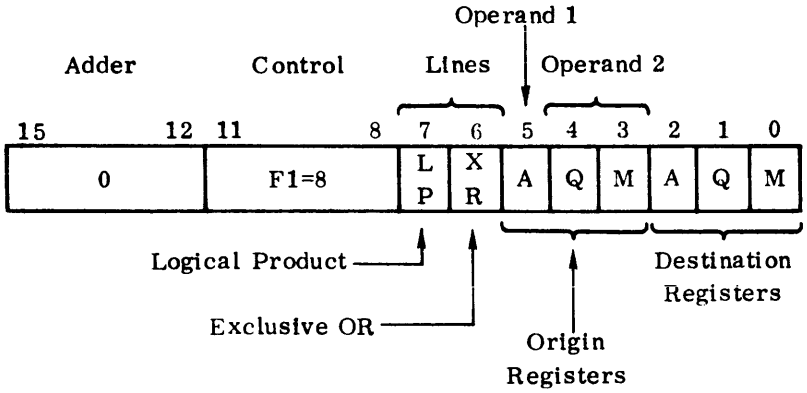
CPB (F1 = 7) Clear Program Protect Bit. Clears the program protect bit in the address specified by Q.

EXI (F1 = E) Exit Interrupt State. This instruction must be used to exit from any interrupt state. Delta defines the interrupt state from which the exit is taken (see 1700 Computer Reference Manual). 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 as 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. Refer to the 1700 Computer Reference Manual for an explanation of the overflow condition in 65K mode.

2.3 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 if 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 non-protected Selective Stop instruction. The Program Protect Fault bit is set and interrupt occurs. See the 1700 Computer Reference Manual for additional information.

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

- Operand 1 may be:

  - FFFF (bit 5 is 0) or
  - The contents of A (bit 5 is 1)

- Operand 2 may be:

  - FFFF (bit 4 is 0 and bit 3 is 0) or
  - The contents of M (bit 4 is 0 and bit 3 is 1) or
  - The contents of Q (bit 4 is 1 and bit 3 is 0) or
  - 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 register(s) is the exclusive OR, bit-by-bit, of operand 1 and operand 2.

- Logical Product (LP = 1 and XR = 0) — The data placed in the destination register(s) is the logical product, bit-by-bit, of operand 1 and operand 2.

- Complement Logical Product (LP = 1 and XR = 1) — The data placed in the destination register(s) is the complement of the logical product, bit-by-bit, of operand 1 and operand 2.

- Arithmetic Sum (LP = 0 and XR = 0) — The data placed in the destination register(s) is the arithmetic sum of operand 1 and operand 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

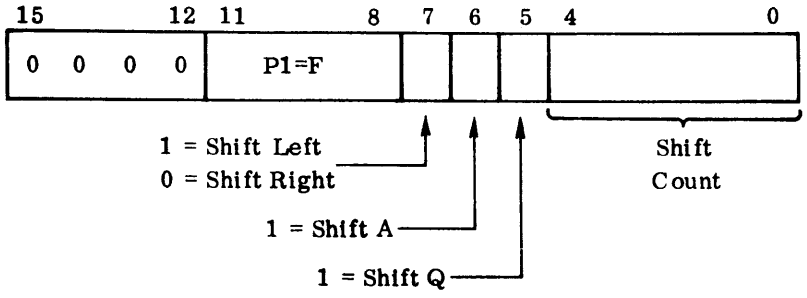
Note: The "+" implies an inclusive OR.

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

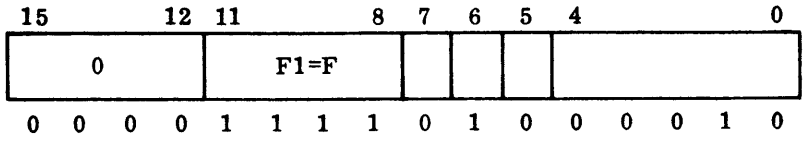
2.4 SHIFT INSTRUCTIONS

The Shift instructions shift A, Q, or QA 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 places.

Format:



Example: Shift A right two places — 0F42.



SHIFT MNEMONICS

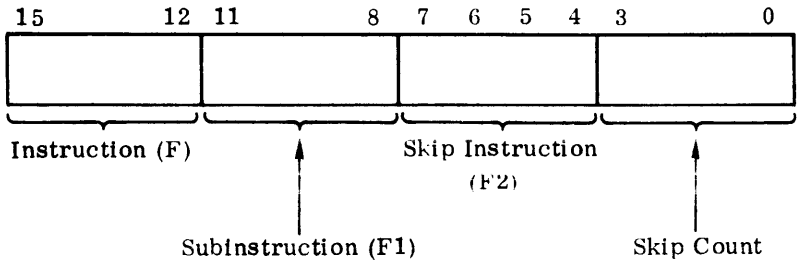
ARS (F1 = F, bits 7 through 5 = 010)	A Right Shift
QRS (F1 = F, bits 7 through 5 = 001)	Q Right Shift
LRS (F1 = F, bits 7 through 5 = 011)	Long Right Shift (QA)
ALS (F1 = F, bits 7 through 5 = 110)	A Left Shift
QLS (F1 = F, bits 7 through 5 = 101)	Q Left Shift
LLS (F1 = F, bits 7 through 5 = 111)	Long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

2.5 SKIP INSTRUCTIONS

Skip instructions result in one machine word: a 12-bit operation code and a 4-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 instructions 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. (Refer to the 1700 Computer Reference Manual.)
SNF (Fs = F)	Skip on No Program Protect Fault.

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

- Addition (-0) + (-0) = (-0)
- Subtraction (-0) - (+0) = (-0)

- Multiplication

  (+0) x (-N) = (-0)

  (-N) x (+0) = (-0)

  (-0) x (+N) = (-0)

  (+N) x (-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
```

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

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 $s_1, s_2, \ldots, s_n$

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 $s_1, s_2, \ldots, s_n$

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.

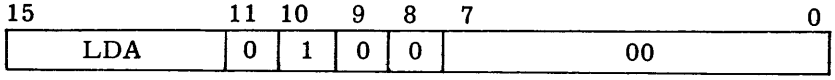
Example 1:

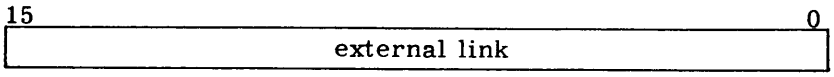
```
        NAM
        EXT     ENT1,ENT2       (Legal)
ENT3    LDA     XYZ
        COM     ENT5
        EXT     ENT3            (Illegal; ENT3 is same subprogram)
        EXT     ENT4            (Legal)
        EXT     ENT5            (Illegal; ENT5 in common storage)
        EXT     ENT1            (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* $s_1, s_2, \ldots, s_n$

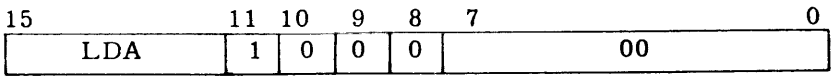
The plus terminator cannot be used with an operation code when the address references a relative external entry point. It is also illegal to enclose an external in parentheses in the address field of an ADC instruction (section 3.3.1).

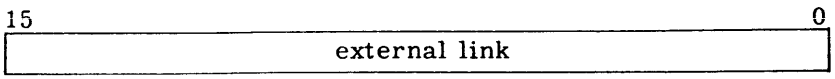
Examples:

```
 .
 .
 .
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 will be set if the expression is enclosed in parentheses (indicating an indirect reference). 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.

If n is an integer, 2n characters of the message are stored. 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: 0 0 1 | 12: 0 0 0 1 0 | 7: 0 1 0 1 1 0 0 0 | 0
2	15: 0 1 0 1 1 0 0 1 | 7: 0 0 1 1 | 3: 0 0 0 0 | 0

2. Source language statements

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

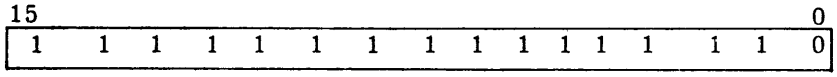
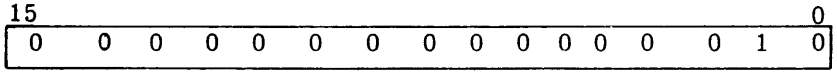
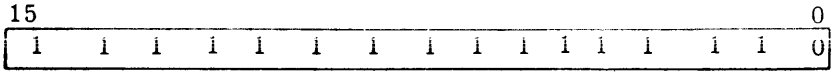
result in machine words

Word	Contents
1	15: 1 1 1 1 1 1 1 1 | 7: 0 1 0 0 1 1 0 0 | 0
2	15: 0 | 14: loc of NAM1 | 0

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	The expression is assumed to be zero.

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     e1,c,e2
```

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
```

When OPT appears, the assembler requests input of control options 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	Execute output on mass storage device
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.

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 1700 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.

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 which 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 remark 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 list 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
```

4.4 MACRO 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 complete set of macro definitions to be input to the library is terminated by the characters ENDMAC starting in column 1 of the source image.

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 system initializer or library editor of the 1700 MSOS.

The library editor of the 1700 operating system 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.

Refer to the 1700 MSOS Reference Manual for further information.

ASSEMBLER OUTPUT 5

5.1 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.

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.

5.1.1 P OPTION

Relocatable binary output is selected by the P option. The format is described in the 1700 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.

5.1.2 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 1700 MSOS Reference Manual.

5.1.3 L OPTION

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

With the OPT pseudo instruction (section 3.4.5) any or all of the preceding options may be omitted. OPT also provides options for listing macro skeletons and abandoning assembly.

5.1.4 C OPTION

The C option produces a cross-reference list which is printed at the end of the assembly list (as described in section 5.2.2).

5.2 ASSEMBLY LISTING

The assembly list, output to standard list output device, consists of 18 columns 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.

5.2.1 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.

5.2.2 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.

5.2.3 SAMPLE PROGRAM

The following source program results in the assembly listing in section 5.2.4.

```
          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
```

5.2.4 SAMPLE LISTING

The following assembly listing is output from the assembly of the source program in section 5.2.3.

```
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  P000E 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
```

MNEMONIC INSTRUCTION CODES A

MACHINE INSTRUCTIONS

There are six classes of 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.

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 1700 MSOS Version 4 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 will never be 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 1700 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
⑴	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.

INDEX

COMMENT SHEET

MANUAL TITLE Control Data 1700 Computer System 1700 MSOS 4
Macro Assembler Reference Manual

PUBLICATION NO. 60361900 REVISION B

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 references and fill in publication revision level as shown by the last entry on the Record of Revision page at the front of the manual. Customer engineers are urged to use the TAR.

CUT ALONG LINE

AA3419 REV. 11/69 PRINTED IN U.S.A.

NO POSTAGE STAMP NECESSARY IF MAILED IN U. S. A.

FOLD ON DOTTED LINES AND STAPLE

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
SMALL COMPUTER DEVELOPMENT DIVISION
4455 EASTGATE MALL
LA JOLLA, CALIFORNIA 92037

ATTN: PUBLICATIONS DEPARTMENT

CUT ALONG LINE

FOLD

STAPLE

STAPLE

CORPORATE HEADQUARTERS, 8100 34th AVE. SO., MINNEAPOLIS, MINN. 55440
SALES OFFICES AND SERVICE CENTERS IN MAJOR CITIES THROUGHOUT THE WORLD

Litho in U.S.A.