96836500

CONTROL DATA CORPORATION

CYBER CROSS SYSTEM VERSION 1 MACRO ASSEMBLER REFERENCE MANUAL

CONTROL DATA®
CYBER 170 SERIES
CYBER 70 SERIES MODELS 72, 73, 74
6000 SERIES COMPUTER SYSTEMS
CYBER 18 COMPUTER SYSTEMS
255X HOST COMMUNICATIONS PROCESSORS

REVISION RECORD

REVISION	DESCRIPTION
A	Manual released.
(12/75)	
B	Manual revised to reflect NOS/BE 1.1 changes and name change of manual from CCP Support Software to
(4/76)	CYBER Cross System.

Publication No.
96836500

Printed in the United States of America

Address comments concerning this manual to:

Control Data Corporation
Publications and Graphics Division
4455 Eastgate Mall
La Jolla, California 92037
or use Comment Sheet in the back of this manual.

LIST OF EFFECTIVE PAGES

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

Page	Revision	Feature
Cover	—	
Title Page	—	
Revision Record	B	
iii	B	
iv	B	
v	B	
vii	B	
viii	B	
1-1	B	
2-1	B	
2-2	B	
2-3	B	
2-4	B	
2-5	B	
2-6	B	
2-7	B	
3-1	B	
3-2	B	
3-3	B	
3-4	B	
3-5	B	
3-6	B	
3-7	B	
3-8	B	
3-9	B	
3-10	B	
3-11	B	
3-12	B	
3-13	B	
3-14	B	
3-15	B	
3-16	B	
3-17	B	
3-18	B	
3-19	B	
3-20	B	
3-21	B	
3-22	B	
3-23	B	
3-24	B	
3-25	B	
3-26	B	
3-27	B	
3-28	B	
3-29	B	

Page	Revision	Feature
3-30	B	
3-31	B	
3-32	B	
3-33	B	
3-34	B	
3-35	B	
3-36	B	
4-1	B	
4-2	B	
4-3	B	
4-4	B	
4-5	B	
4-6	B	
4-7	B	
4-8	B	
4-9	B	
4-10	B	
4-11	B	
4-12	B	
4-13	B	
4-14	B	
A-1	B	
A-2	B	
B-1	B	
B-2	B	
B-3	B	
C-1	B	
D-1	B	
E-1	B	
F-1	B	
F-2	B	
F-3	B	
F-4	B	
F-5	B	
F-6	B	
F-7	B	
F-8	B	
F-9	B	
F-10	B	
F-11	B	
F-12	B	
F-13	B	
F-14	B	
G-1	B	
G-2	B	

LIST OF EFFECTIVE PAGES (CONTINUED)

Page	Revision	Feature
G-3	B	
Index-1	B	
Index-2	B	
Index-3	B	
Comment Sheet	B	
Envelope	—	
Back Cover	—	

Page	Revision	Feature

PREFACE

The CYBER Macro Assembler is a component of the CONTROL DATA® CYBER Cross System. The Macro Assembler, referred to as CLASS (Compass Like Assembler), operates under control of the CYBER 170/70/6000 NOS or NOS/BE operating systems. CLASS is intended to convert source language input including macro instructions into relocatable binary output and generate list output. A separate version of the Macro Assembler is available for the CYBER 18 computer series.

This manual describes the general operation of the assembler and provides the necessary instructions for preparing programs for assembly. No attempt is made here to provide a programmers guide and, therefore, examples are limited. It is assumed that the reader is already familiar with the operation of the CYBER 18 computer.

Additional information can be found in the following publications:

Description	Publication No.
CYBER Cross System Version 1 Reference Manual	96836000
CYBER Cross System Version 1 Micro Assembler Reference Manual	96836400
CYBER Cross System Version 1 Link Editor and Library Maintenance Programs Reference Manual	60471200
NOS/BE 1 Reference Manual	60493800
NOS 1 Reference Manual, Volume 1	60435400
NOS 1 Reference Manual, Volume 2	60445300
1700 MSOS 4 Macro Assembler Reference Manual	60361900
MSOS Version 4 Macro Assembler Reference Manual	60361500
1700 Computer Reference Manual	60153100
CYBERNET/KRONOS 2.1 Batch and Remote Batch Reference Manual	80400600

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

CONTENTS

TABLES

INTRODUCTION 1

The CLASS Macro Assembler, hereafter referred to as CLASS†, is a two-pass assembler that executes on a CYBER 170/70/6000 computer. It can convert source language input including macro instructions to relocatable binary output and generate list output. The source programs are written with symbolic machine, pseudo, and macro instructions. Enhanced instructions are assembled by means of assembly text consisting of macros residing on a separate file that accompanies the assembler. Additional macros may be defined by the user in the source program.

During the first pass each card (one instruction per card) is processed in sequence, programmer-defined macros are processed into macro skeletons, skeletons with actual parameters substituted are inserted at macro calls, a symbol table is built, conditional assemblies are evaluated and processed, and the resulting source cards are written to a scratch file.

During the second and final pass, each source image is read in sequence from the scratch file and processed, errors are flagged as they occur, the actual binary relocatable output is generated for each instruction, the source image and binary image are listed if not suppressed, and a cross-reference is generated for the listing.

Three versions of the binary output are produced:

- File B is used to link programs in the CYBER Cross System
- File P8 is used to punch cards for loading with the MSOS 4 Loader
- File RB is used to write to magnetic tape for the MSOS 4 Loader.

† CLASS, a Compass-Like ASSembler

LANGUAGE STRUCTURE 2

2.1 CONTROL OF THE ASSEMBLER

Parameters on the CLASS call card are used to specify control options to the assembler. (See Appendix B for a detailed explanation of these parameters.) The following is a list of the features available:

- The starting columns for the location, operation code, address, and comment fields can be designated for listing output.
- All BZS/BSS blocks and their names, addresses, and lengths can be listed on the banner page (the first page of the listing).
- List control cards such as LST, NLS, SPC, and EJT can be printed.
- The printing of comment cards (i.e., * in column 1) can be suppressed.
- EJT cards can be processed as page ejects. Normally, they are processed as four spaces. (See Section 3.2.5.)
- If the IFA condition is false, printing of code between IFA and EIF can be suppressed.
- All machine code on multiword instructions (such as LRQ, NUM, DEC, etc.) can be printed.
- A full cross-reference map of symbols, providing the page number/line number where they are referenced, can be printed.
- A short reference map giving only the symbols and their value can be printed.
- Column tabs can be set for the tidy feature.
- Macro code may be expanded (not recommended if Type 2 instructions are used).
- A macro cross-reference table which contains the macro name, the number of formal parameters and locals, and the page and line number where they are referenced, can be printed.
- An assembly text of predefined symbols, macro skeletons, and macro name table can be generated and utilized.

2.2 SOURCE PROGRAM INPUT STRUCTURE

2.2.1 SOURCE PROGRAM

A source program consists of one or more subprograms. Each subprogram is a set of source statements preceded by a NAM card and followed by an END card. 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 does not have to be the first subprogram.

Communications between subprograms is accomplished by the subprogram linkage pseudo instructions (e.g., EXT, ENT) and by the use of common and data storage, which are established by the COM and DAT pseudo instructions.

2.2.2 SOURCE STATEMENT

A source statement consists of the location, instruction, operation code, address, comment, and sequence fields, respectively. The first five fields must not exceed 72 characters; within that limitation they are free field except that each field must be separated by at least one space. 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, depending on the input device. Any number of blanks may separate fields.

2.2.2.1 LOCATION FIELD

The location field must begin in column 1. If used, this field specifies a labeled statement. This statement is a symbolic name consisting of one to six alphanumeric characters; the first must be alphabetic. Characters in excess of six are ignored.

2.2.2.2 INSTRUCTION FIELD (OPERATION CODE FIELD, OPCODE)

This field begins to the right of the location field and must be separated from it by at least one blank (or a tab). If the location field contains no label, the operation code may begin in column 2. The operation code field contains a three-character instruction code or pseudo instruction code; or it contains a macro-instruction code which may be up to six characters. Certain instructions (storage reference, etc.) may be followed by one of the one-character OPCODE terminators Δ, *, +, or -. There are two main groups of machine instructions: Type 1 (CYBER 18 instructions) and Type 2 enhanced instructions).

2.2.2.3 ADDRESS FIELD

The address field begins to the right of the operation code field, separated from it by at least one blank character or tab. It is terminated by a blank or tab, or by the 72nd character of the source statement. Exceptions are the macro instructions that may have a continuation line and the pseudo instruction ALF.

This field contains an expression consisting of:

1. One or more operands connected by the operators +, -, *, or /
2. One of the register designators A, B†, I, M†, Q, 1, 2, 3, 4
3. Both of the above, separated by commas

An operand is either a numeric constant or a symbol used (defined) as:

- The label in the location field of any machine or macro instruction
- The label in the location field of one of the pseudo instructions: ADC, ALF, NUM, DEC, or VFD
- A symbolic name in the address field of one of the pseudo instructions: EXT, COM, DAT, BSS, BZS, EQU, FLD, or EXF

Such a symbol references a specific location in memory.

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 ±32,767. 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 0 through +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 indicates that the ones complement of the binary equivalent is the value.

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

†Type 1 instructions only

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

The asterisk has meaning both as an operator (multiplication) and as an operand (the current value of the P counter). 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.

The slash, used as the division operator, must be between two operands. The operand that follows may not be 0 or relocatable.

Example:

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

The 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.
```

An external name (location in another subprogram referenced by this subprogram) may be used in an address expression as a single operand only. Arithmetic operators preceding or following an external operand are illegal.

The hierarchy for evaluating 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 algebraically 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 algebraic expression

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

must be written as

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 0 or one of the relocation terms. If 0, the location is absolute.

Example:

	Source Statements		Relocation Formula
	NAM	EXAMPLE 3	
	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)
	END		

INDEXING AND REFERENCE REGISTERS

The special characters A, B, I, M, Q, 1, 2, 3, and 4 are used to specify registers which may be used as reference or index registers within instructions. The set of legal reference registers and index registers differs between Type 1 and Type 2 instructions:

	Reference Registers	Index Registers
Type 1	A, I, M, Q	B, I, Q
Type 2	A, I, Q, 1, 2, 3, 4	A, I, Q, 1, 2, 3, 4

Indexing may be used with storage reference instructions only. Only one index specifier may follow any address expression; it is separated from the expression by a comma with no intervening blanks. The meanings of these special characters used as indices are specified below:

Q	The contents of the Q register are added to the contents of the expression to form the actual address.
I	The contents of location $FF are added to the contents of the address expression to form the actual address.
B	The contents of the Q register are added to the address expression. This sum is added to the contents of $FF to produce the actual address.
A	The contents of the A register are added to the contents of the expression to form the actual address.
1	The contents of the 1 register are added to the contents of the expression to form the actual address.
2	The contents of the 2 register are added to the contents of the expression to form the actual address.
3	The contents of the 3 register are added to the contents of the expression to form the actual address.
4	The contents of the 4 register are added to the contents of the expression to form the actual address.

Examples:

Address Field		Function
LOC1, B	Legal	The contents of registers Q and $FF and the address of LOC1 are added to produce the actual address.
,,I	Illegal	The character following the first comma is assumed to be the index character.
TAG2, Q, I	Illegal	Only one index notation is allowed.
Q	Illegal	Unless Q has been previously defined as a location symbol or is being used with the interregister transfer instruction, it must follow a location symbol.

Address Field		Function
TAG3, I	Legal	The contents of $FF and TAG3 are added to produce the actual address.
TAG2, 4	Legal	The address of TAG2 and register 4 are added to produce the actual address.

Certain instructions use the special characters to reference registers, for example:

Instruction	Address Field	Function
SET	A, Q, M	Set the A, Q, and mask registers to 1s.
TRA	Q	Transfer the contents of the A register to the Q register.
LAM	M	Transfer the logical product of the A and the mask register to the mask register.
XF2	Q	Transfer the contents of register 2 to register Q.

2.2.2.4 COMMENT FIELD

The address field is followed by the comment field. Comments do not affect program execution; however, the comments in the NAM card are copied into the object text NAM block. The comment field terminates at column 72 or with a carriage return (paper tape). Blanks are permitted in the comment field.

2.2.2.5 SEQUENCE FIELD

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

SYMBOLIC MACHINE INSTRUCTIONS 3

3.1 GENERAL

The instructions are divided into classes according to the kind of function performed. These instruction classes are:

- Storage Reference
- Field Reference
- Register Reference
- Shift
- Skip
- Inter-Register Transfer
- Decrement and Repeat
- Miscellaneous

Instructions are referred to as Type 1 or Type 2. Type 1 instructions are original CDC CYBER 18 instructions; Type 2 instructions are added or enhanced instructions. A complete list of the instruction set with the relocatable binary machine code format and general definitions is given in Appendix F. These definitions are intended for quick reference.

3.1.1 STORAGE REFERENCE INSTRUCTIONS

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 0. The object code produced by the assembler contains addresses which the loader modifies by a relocation factor at load time to produce the actual address in memory. Actual location of these three areas in core memory is controlled by the loader.

A symbol is program relocatable if it references a location in the subprogram; it is data relocatable if it references a location in data storage; and it is 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 of different relocation types must reduce to one relocation type 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.

Type 1 storage reference instructions are divided into groups A and B. These storage reference instructions use storage addresses as operands or as operand addresses. Group B includes jump and store instructions and may not use the constant mode of addressing.

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.

Type 2 storage reference instructions allow absolute, relative, and, in certain cases, constant addressing modes. Constant addressing is valid only in storage reference instructions that transmit information to a register. Special characters designate the mode of addressing, the number of words for the instruction, and indirect addressing.

Character	Description
*	An asterisk as the last character of the operation code specifies relative addressing in a one-word instruction (two words for Type 2).
-	A minus as the last character of the operation code specifies absolute addressing in a one-word instruction (two words for Type 2).
+	A plus as the last character of the operation code specifies absolute addressing in a two-word instruction (three words for Type 2).
=	An equal sign as the first character in the address field preceding a constant indicates constant addressing; the instruction is always two words (three words for Type 2).
( )	Parentheses enclosing the address expression indicate indirect addressing for Type 1 and Type 2.
Δ	If no character is specified as a terminator to the operation code, multiword relative addressing is assumed with the following exceptions: • If a constant is specified, the constant mode is assumed. • If the relocation type of the address expression differs from the relocation type of the location counter, two-word absolute addressing is assumed (Type 1 only). • If a nonrelative external is referenced, absolute addressing is assumed (Type 1 only).

Example:

The following are the relocation types (RT) of the current location counter:

P	Program relocatable
C	Common relocatable
D	Data relocatable
A	Absolute address

RT	Label	Operation	Address
		NAM	EXAMPLE2
C		COM	COM1, COM2
D		DAT	DAT1, DAT2
A		EQU	D(1), E(3), G(E-D), H($1000)
P		BZS	A, B, C
P		BZS	J, K(10)
P	START	ADC	0
P		LDA*	START
P		STA*	DAT1 (Error)
P		STA*	COM1 (Error)

Both errors resulted because the relocation types of the symbols in the address field did not match those in the location counter and the short relative address mode was requested by an asterisk terminator.

RT	Label	Operation	Address	Comments
P		LDA+	DAT1 (Not an error)	Relocations do not have to match when the mode is long absolute.
P		LDA	START (OK, relocations match)	
P		LDA	COM1 (Not an error)	The assembler changes this instruction to long absolute because the relocations do not match, but no error is indicated.
P		LDA	START-K+DAT2-DAT1+COM2	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 the original relocation.
P		LDA*	START (Not an error)	
		ORG	H	
A		LDA*	START (Error)	

RT	Label	Operation	Address
A		STA*	DAT1 (Error)
A		LDA*	$1001
A		STA-	B (Error)
		ORG*	
		END	EXAMPLE2

The machine language formats resulting from a storage reference instruction are given in the following section.

3.1.1.1 STORAGE REFERENCE, ABSOLUTE ADDRESSING (+, —)

The value of the address expression of an absolute short storage reference instruction must be non-relocatable. The evaluated result is stored in the last eight bits of the machine language instruction. If this value is greater than 255, it is flagged as an error. If these last eight bits are 0, a long absolute instruction is assumed when the instruction is executed. No error is flagged.

Type 1

If the address expression is enclosed in parentheses, indirect addressing is indicated and bit 10 of the first word is set.

Type 2

If the address expression is enclosed in parentheses, indirect addressing is indicated and bit 6 of the first word is set.

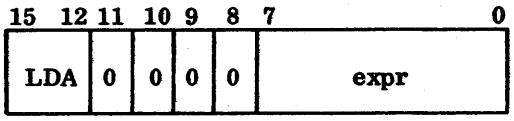
Examples:

Absolute Short Direct -

Instruction:

LDA- expr[†]

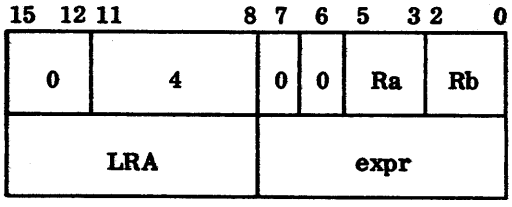
Machine Word:



Instruction:

LRA- expr

Machine Word:



[†]expr is an address expression.

Type 1

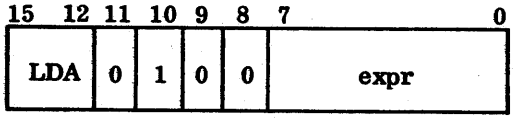
Type 2

Absolute Short Indirect† -, ( )

Instruction:

LDA- (expr)

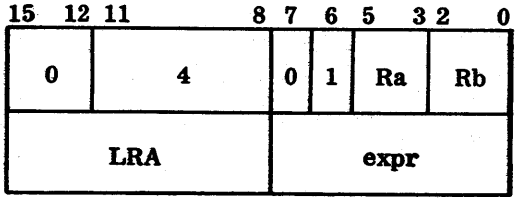
Machine Word:



Instruction:

LRA- (expr)

Machine Word:

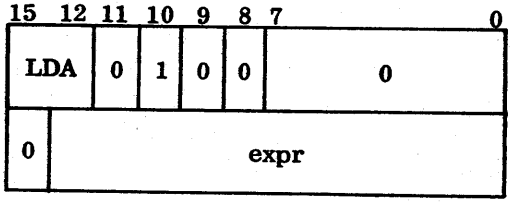


Absolute Long Direct +

Instruction:

LDA+ expr

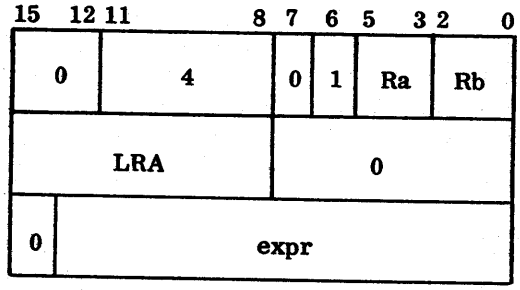
Machine Word:



Instruction:

LRA+ expr

Machine Word:



†In indirect addressing the storage location referenced is not the address expression location, but the contents of the address expression location.

Type 1

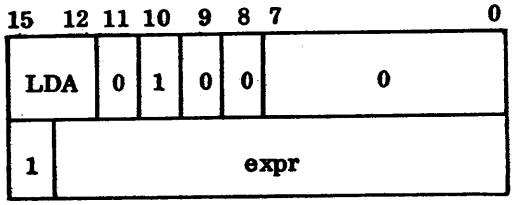
Type 2

Absolute Long Indirect +, ( )

Instruction:

LDA+ (expr)

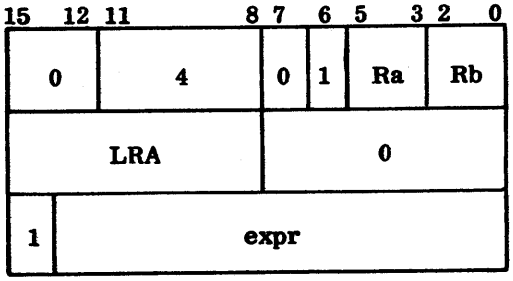
Machine Word:



Instruction:

LRA+ (expr)

Machine Word:



3.1.1.2 STORAGE REFERENCE, RELATIVE ADDRESSING (Δ, *)

When short 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 ±$7F, 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 0, a long instruction is assumed regardless of the operation code terminator. No error message is printed for this condition.

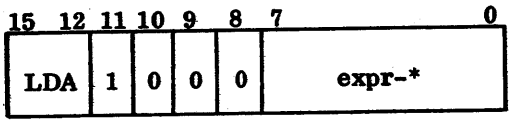
Examples:

Relative Short Direct *

Instruction:

LDA* expr

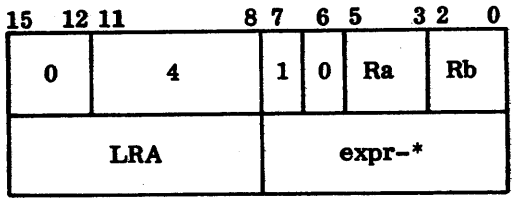
Machine Word:



Instruction:

LRA* expr

Machine Word:



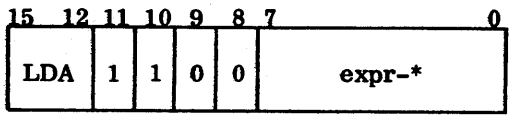
Type 1 **Type 2**

Relative Short Indirect *, ( )

Instruction:

LDA* (expr)

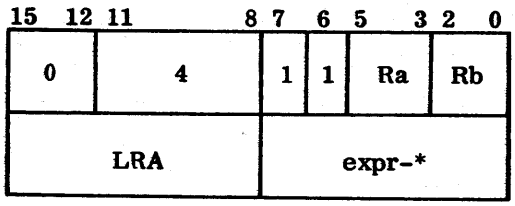
Machine Word:



Instruction:

LRA* (expr)

Machine Word:



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

When a relative long instruction is specified, the value of the current location counter is subtracted (using 16-bit ones complement arithmetic) from the value of the address expression to obtain the 16-bit second word. For Type 1, 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 long absolute instruction. If the address expression is an external reference, the instruction is absolute or relative depending on the definition of the external.

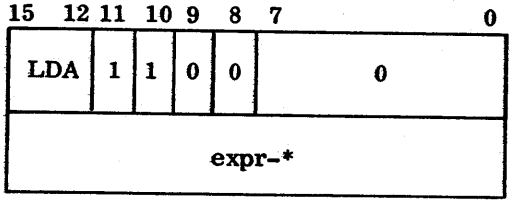
Examples:

Relative Long Direct Δ

Instruction:

LDA expr

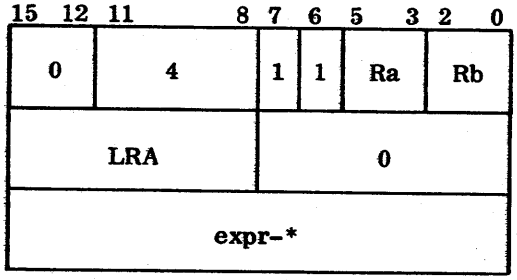
Machine Word:



Instruction:

LRA expr

Machine Word:



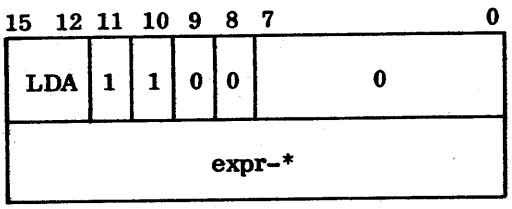
Type 1 Type 2

Relative Long Indirect Δ, ( )

Instruction:

LDA (expr)

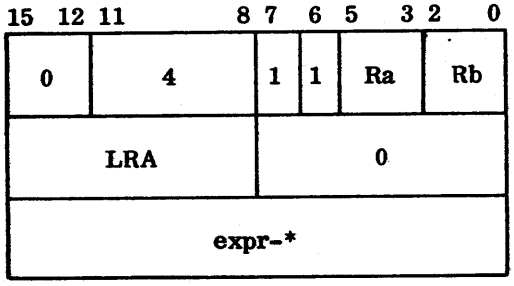
Machine Word:



Instruction:

LRA (expr)

Machine Word:



3.1.1.3 STORAGE REFERENCE, CONSTANT ADDRESSING

Constant addressing may only be used for certain 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	Two alphanumeric characters (Type 1 only)
N	±ddddd	A five-digit decimal number with or without a leading sign
N	±$hhhh	A four-digit hexadecimal number preceded by $, with or without a sign
X	e	An address expression evaluated modulo $2^{15}-1$
X	(e)	An address expression evaluated modulo $2^{15}-1$, with bit 15 set (Type 1 only)

Examples:

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

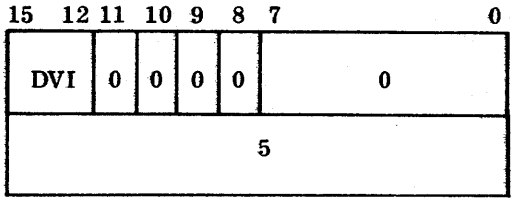
An instruction containing a constant in the address field results in two machine words (three words for Type 2).

Example:

Instruction:

```
DVI     =N5
```

Machine Words:



3.1.1.4 FIELD REFERENCE

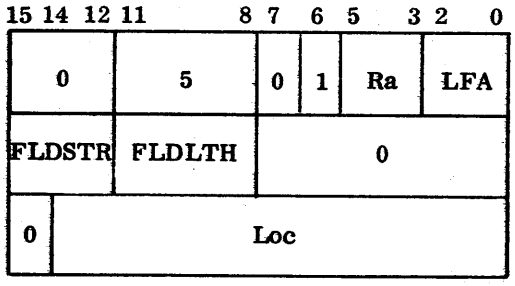
The field reference instructions follow precisely the same address mode conventions as the general storage reference instructions. Field start (FLDSTR) and field length (FLDLTH) may not be externals, although a field can be defined in its entirety as an external field.

Example:

Instruction:

```
LFA+    Loc, 7, 2
```

Machine Word:



Where: **FLDSTR is set to 7.**

FLDLTH is set to 1 (one less than the actual field length, 2).

3.1.1.5 REGISTER INSTRUCTIONS (TYPE 1 ONLY)

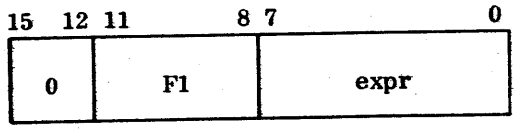
Register instructions (Type 1 only) result in one machine word; an eight-bit operation code field and an eight bit Δ field. The first four bits of the operation code are set to 0; the next four bits contain the unique identifier F1 for each register instruction. The expression in the address field of the instruction is evaluated modulo $2^{15}-1$ and truncated to fit in the eight-bit Δ field of the machine word. The value of the expression must be absolute.

Example:

Instruction:

```
ENA       expr
```

Machine Word:



3.1.1.6 SHIFT INSTRUCTIONS (TYPE 1 ONLY)

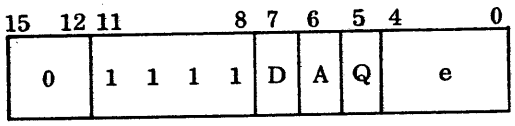
The shift instructions (Type 1 only) result in one machine word containing an 11-bit operation code and a five-bit shift count. The first four bits of the operation code are set to 0; the next four bits contain the unique identifier F1; the remaining three bits identify the direction of the shift and the registers used. F1 is 1111 for shift instructions. The expression in the address field of the instruction is evaluated modulo $2^{15}-1$ and becomes the shift count. It is truncated to five bits without a sign and placed in bits 4 to 0 of the machine word. This expression must be absolute.

Example:

Instruction:

```
LLS       e
```

Machine Word:



Where:	D is	0	Right shift.
		1	Left shift.
	A is	0	A register is ignored.
		1	A register is used in the shift.
	Q is	0	Q register is ignored.
		1	Q register is used in the shift.

3.1.1.7 SKIP INSTRUCTIONS

Skip instructions result in one machine word: a 12-bit operation code and a four-bit unsigned skip count. The first four bits of the operation code field are set to 0; the next four bits contain the skip instruction code 0001 or 0000; 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 $2^{15}-1$.

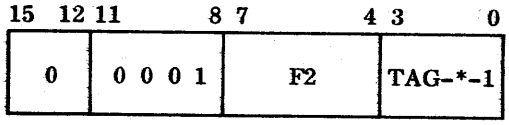
This expression may be absolute or relocatable for Type 1 instructions, but must be absolute for Type 2 instructions. If absolute, the value of the expression is the skip count. If relocatable, the value of the skip count is obtained by subtracting (16-bit ones complement arithmetic) the value of the 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 results.† If the expression is relocatable, the relocation type of the expression must match the relocation type of the location counter or an error results.

Examples:

Address Expression Relocatable Instruction:

```
SAZ       TAG      (TAG program relocatable)
```

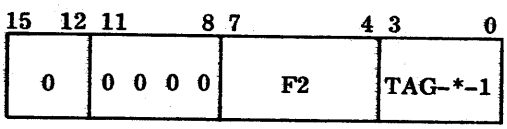
Machine Word:



Address Expression Absolute Instruction:

```
S3M       TAG-*-1  (TAG program relocatable)
```

Machine Word:



†An error message is printed for Type 1 instructions.

3.1.1.8 INTER-REGISTER TRANSFER

Inter-register transfer instructions result in one machine word, a 13-bit operation code, and a three-bit field containing the code for the destination register specified in the address field. The first four bits of the operation code are set to 0 for all inter-register transfer instructions; the next four bits are set to 1000 or 0111; the five remaining bits identify the transfer specified by the instruction code. The last three bits of the machine word are generated from the address field.

The register field bits are set as follows:

Type 1:

Bit	Description
2 = 1	Destination is the A register
2 = 0	A register is ignored
1 = 1	Destination is the Q register
1 = 0	Q register is ignored
0 = 1	Destination is the mask register
0 = 0	Mask register is ignored

Type 2:

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

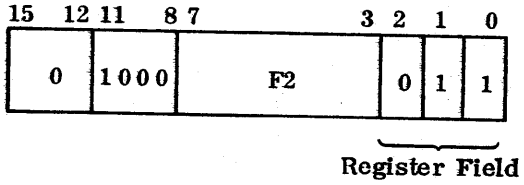
For Type 1 instructions, when 0 is specified in the address field, all three destination register bits are set to 0 indicating no destination for the result. If the instruction is AAM, AAQ, or AAB, the add takes place, no register is destroyed, and the result of the add may be tested for overflow. If any other instruction contains a 0 in the address field, no operation takes place.

Examples:

Type 1 Instruction:

```
TRA        Q, M
```

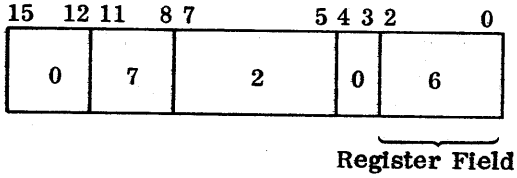
Machine Word:



Type 2 Instruction:

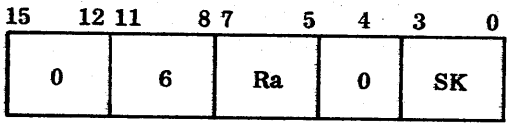
```
XF2        A
```

Machine Word:



3.1.1.9 DECREMENT AND REPEAT (TYPE 2 ONLY)

The decrement and repeat instructions (Type 2 only) result in one machine word. The skip count SK must be absolute, positive, and less than 16.

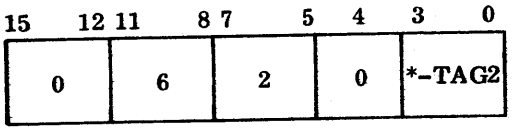


Example:

Instruction:

D2P *-TAG2

Machine Code:



3.1.1.10 MISCELLANEOUS INSTRUCTIONS

The miscellaneous instructions are defined in Appendix F. Most miscellaneous instructions generate one machine word.

3.2 PSEUDO INSTRUCTIONS

Pseudo instructions control the assembler, provide subprogram linkage, control output listing, reserve storage, and convert data. Pseudo instructions may be placed anywhere in a source language subprogram. However, NAM must be the first statement of a subprogram and END must be the last statement.

3.2.1 SUBPROGRAM LINKAGE

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

NAM

Nam identifies a source language subprogram and must be the first statement of the subprogram.

The format is:

NAM s

Where: s is an optional symbolic name of the subprogram that is printed as part of the assembly list output.

END

END must be the last statement of a source language subprogram. The format is:

```
END     s
```

Where: s is 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
```

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

ENT

The ENT instruction specifies the symbolic names of entry points that may be referenced from other subprograms, and identifies these names for the loader. The format is:

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

Where: s_i are 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 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; ETNX not defined)
         .       .
         .       .
         .       .
        END     ENT1
```

EXT/EXT*

The EXT instruction specifies the symbolic names of entry points in external subprograms that may be referenced from this subprogram, and identifies these names for the loader. The format is:

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

Where: s_i are entry points in the address field of EXT and must be symbols defined in the subprograms they reference. s_i must not refer to symbols in the same subprogram.

Example:

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

The EXT* instruction is the same as EXT except that later references to s_i must be absolute under EXT, and references to s_i must be relative if EXT* is used. 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.

ENF

The ENF pseudo instruction specifies the symbolic names of fields that may be referenced from other subprograms, and identifies these fields for the loader. The format is:

ENF $f_1, f_2, \ldots, f_n$

Where: f_i are field names that have been defined in the subprogram containing the ENF pseudo instruction.

EXF/EXF*

The EXF pseudo instruction specifies the entry fields in other subprograms that may be referenced from this subprogram. EXF* is the same as EXF, except that references to the EXF* externals must be relative. The format is:

EXF $f_1, f_2, \ldots, f_n$

Where: f_i are external fields that may be referenced by this subprogram.

3.2.2 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.

BSS

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

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

Where: s_i is a symbolic name that defines the first location of the named segment.

omitted	A segment is assigned with the length e, but no name is assigned to the segment.

e_i is an expression that must be previously defined. It corresponds to the symbolic name that defines the length of the segment in words. Segments are assigned contiguously to form one block of data starting at location s_1. It can be assigned by an EQU instruction. 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.

BZS

This statement functions in the same way as BSS, except that the specified storage locations are set to 0. The format is:

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

Where: s_i is a symbolic name that defines the first location of the named segment.

omitted	A segment is assigned with the length e, but no name is assigned to the segment.

e_i is an expression that must be previously defined. It corresponds to the symbolic name that defines the length of the segment in words. Segments are assigned contiguously to form one block of data starting at location s_1. It can be assigned by an EQU instruction. 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.

Example:

```
        NAM
NAM3    LDA     XYZ1
        BSS     NAM4(3)          Set up a three-word block with NAM4 as the
                                 first word.
        BZS     NAM5(5)          Set up a five-word block of 0s with NAM5 as
                                 the first word.
        BSS     NAM1,NAM2(9)     Set up a one-word block, NAM1. Set up a nine-
                                 word block with NAM2 as the first word.
        BSS     NAM3             Illegal; NAM3 has already been declared.
        BSS     NAM6,(4)         Set up a one-word block, NAM6. Set up a four-
                                 word block for an unnamed segment.
        BSS     NAM7             Set up a one-word block, NAM7.
        EQU     NAM8(4),NAM9(2)  LOC2
        BZS     NAM10(NAM8-NAM9) Set up a two-word block of 0s with NAM10 as
                                 the first word.
        BSS     NAM8(NAM10-1)    Illegal; NAM8 has already been declared.
        BSS     LOC1(0),LOC2     Assign the same word to LOC1 and LOC2.
         .
         .
         .
        END
```

COM

The COM instruction names and defines segments in a block of storage that are common to more than one subprogram. The format is:

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

Where: s_1 is a symbolic name that defines the first location of the named segment.

omitted	A segment is assigned with the length e, but no name is assigned to the segment.

e_i is an expression that must be previously defined. It corresponds to the symbolic name that defines the length of the segment in words. Segments are assigned contiguously to form one block of data starting at location s_1. It can be assigned by an EQU instruction. 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.

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
```

DAT

The DAT instruction reserves area for common storage that is assigned within the program area and which may be preset with data or instructions by using the ORG pseudo instruction. The format is:

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

Where: s_i is a symbolic name that defines the first location of the named segment.

omitted — A segment is assigned with the length e, but no name is assigned to the segment.

e_i is an expression that must be previously defined. It corresponds to the symbolic name that defines the length of the segment in words. Segments are assigned contiguously to form one block of data starting at location s_1. It can be assigned by an EQU instruction. 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.

3.2.3 CONSTANT DECLARATIONS

These pseudo instructions introduce constant values into the instruction sequence.

ADC/ADC*

An ADC/ADC* instruction evaluates the address expressions. The resultant address constants are stored in consecutive locations within the instruction sequence. The format is:

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

Where: s is a symbolic name in the location that is assigned to the first constant in the address field.

e_i is an expression that corresponds to the symbolic name that 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. Indirect addressing is specified by parentheses.

omitted	The length is assumed to be one computer word.

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 location counter is subtracted from the value of the evaluated expression (16-bit ones complement arithmetic) and the result is the 16-bit address constant.

Indirect addressing cannot be specified in the ADC* statement.

ALF

The ALF instruction puts the message in ASCII format. The format is:

s ALF n, message

Where: s is a symbolic name in the location that is assigned to the first constant in the address field.

n is an unsigned integer; it specifies the number of words to be stored. 2n equals the number of characters.

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.
a noninteger character	Signals the end of the message
a special 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 and stored as 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 (hexadecimal) and counted as one character in determining the value of n, when n is an integer character count. A colon is a eight- and five-keypunch code with the ASCII value of $3A.

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

NUM

The NUM instruction defines numeric constants. The format is:

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

Where: s is a symbolic name in the location that is assigned to the first constant in the address field.

k_i are specified integer constants stored into consecutive locations in the instruction sequence. Each constant may be a decimal integer within the range ±32,767, or a hexadecimal integer preceded by a $ within the range ±7FFF. The constant is assumed to be positive. When the sign is minus, the ones complement of the number is used.

Example:

The source statements,

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

are translated into the following machine words.

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

DEC

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

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

Where: s is a symbolic name in the location that is assigned to the first constant in the address field.

k_i are specified integer constants stored into consecutive locations in the instruction sequence. They are signed decimal integers followed by a decimal and/or binary scaling factor. The decimal scaling factor consists of a D followed by a signed or unsigned decimal integer. The binary scaling factor is a 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 the following machine words:

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

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. However, all the data to be stored in a memory location must be specified in a single VFD. A symbolic name in the location field is assigned to the first word of data. The format is:

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

Where: s is a symbolic name that defines the first location of the named segment.

m is the mode of the data.

N The data is a numeric constant 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 ±32,767 and a hexadecimal integer within the range ±7FFF.

A v is a string of characters 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.

X v is an expression 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 $2^{15}-1$. If the final value is absolute and n exceeds the size required, the value is right-justified in the field. If it is 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 ones 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, but the value used in the 16-bit calculation of this symbol is $FFFE. Numeric operands used in a 16-bit expression may be 16 bits in magnitude.

n is the number of bits to be allocated.

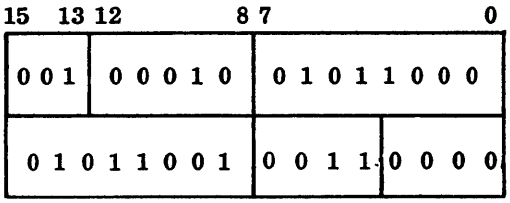
v is the value of the data.

Examples:

1. The source language statements,

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

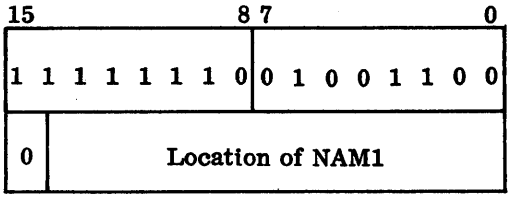
result in the following machine words:



2. The source language statements,

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

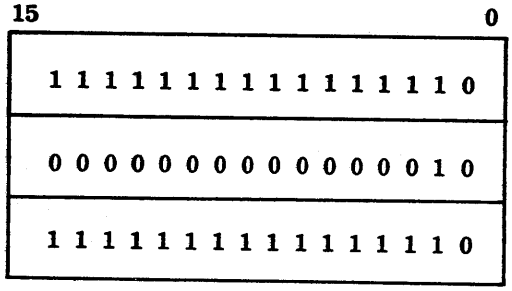
result in the following words:



3. The source language statements,

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

result in the following machine words:



3.2.4 ASSEMBLER COMMUNICATION

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

EQU

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

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

Where: s_i is a symbolic name that is equated to the value of e_i.

e_i are symbolic operands that have been previously defined and which are 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 0.

FLD

The FLD pseudo instruction defines a field to the assembler. Fields defined using this pseudo instruction may be referenced by a simple name which has all the attributes of the field. These fields may also be declared external or entry fields. The format is:

```
FLD       N(W,S,L)
```

Where: N is the name of the field; follows the same rules as the location names.

W is the word in which the field is contained.

S is the start bit, the leftmost bit in the field. $15 \geq S \geq 0$

L is the length of the field (number of bits). $1 \leq L \leq 16$

ORG/ORG*

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

```
ORG       e
```

Where: e_i is an expression that is evaluated modulo $2^{15}-1$. 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.

The ORG* 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)
```

IFA

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

s IFA e_1,c,e_2

Where: s is a symbolic name in the location field. It is used as an identifying tag only; it is not defined as a location symbol within the program. If specified, the first two characters of the identifier s must match the first two 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 statement.

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

c allows the code to be assembled if a specified condition exists between e_1 and e_2. 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	$e_1 < e_2$

EIF

The EIF instruction signals the termination of an IFA or IFC instruction (Section 3.3.1) 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
```

Where: s is the symbolic name in the address field that establishes the correspondence between an IFA or IFC and an EIF instruction. The first two characters of s must be the same as the first two 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+9. Both OP1 and OP2 are skipped if the value of NAM1 is less than the value of NAM2+8.

3.2.5 LISTING CONTROL

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

NLS

The NLS instruction inhibits list output. The format is:

```
NLS
```

Normally, list output is enabled until an NLS occurs. It remains inhibited until an LST instruction or the end of the program occurs.

LST

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

```
LST
```

SPC

The SPC instruction controls line spacing on the list output unit. No spaces are output if the SPC card is encountered at the top of a listing page. The format is:

```
SPC         e
```

Where: e is the number of lines to be skipped. The expression is evaluated modulo $2^{15}-1$ and must be absolute.

EJT

CLASS processes an EJT pseudo instruction as four spaces instead of a page eject. However, if the EJT is encountered more than three-quarters of the way down a listing page, the page will be ejected. No spacing or ejecting occurs if the EJT is encountered at the top of a listing page. There is an option available to process all EJT cards as page ejects. The format is:

```
EJT
```

ERR

The ERR pseudo instruction is provided for programmer-defined errors. It is intended to be used within the coding lines of an IFA or IFC sequence to identify certain erroneous conditions which are set up by the programmer. It is also used in the assembly text for enhanced instructions. The format is:

```
ERR
```

The following is a source code sequence example:

```
TAG         IFA         SYMB, EQ, 0
            ERR
            EIF         TAG
```

In this example, if SYMB is equated to 0, an error results and will be flagged with PD on the line where ERR is assembled.

NOREF

The NOREF pseudo instruction is provided in CLASS to enable the programmer to specify any symbols that he does not want included in the complete reference map at the end of the listing (see Section 4.2.5). The address field contains the symbol names to be excluded, separated by commas. The format is:

```
        NOREF
```

The following is the source language input:

```
        NOREF     SYMB1, SYMB2, SYMB3, ...
```

3.3 MACROS

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 CLASS Assembler includes two types of macros:

Programmer-defined	Macros declared and defined by MAC pseudo instructions. Each macro may be defined anywhere in the program prior to the first reference to it. Comment cards may be placed anywhere in the macro definition.
Library	Definitions contained on the system library that may be called from any subprogram.

If an error is encountered by CLASS in a macro definition, it is flagged and CLASS continues processing the definition. However, whenever a macro is called whose definition was erroneous, no attempt is made to substitute the skeleton for the call. The programmer is merely notified that there was an error in the definition. Also, symbols beginning with the characters O0 and Q99 may not be used as macro names as these are reserved for the assembler. The mnemonics IFK, IFR, and PCO are reserved for the assembler. CLASS limits nesting to 10 calls per macro, only because deeper nesting usually is caused by an error loop. If deeper nesting is required, CLASS must be reassembled (see Appendix E).

3.3.1 MACRO PSEUDO INSTRUCTIONS

These pseudo instructions are only used within a macro definition.

MAC

The MAC instruction is required and names a macro and its formal parameters. The location field contains the name used to call the defined macro. It may be any name not used by a machine or pseudo instruction. The format is:

```
s           MAC       P1, P2, ..., Pn
```

Where: s is a symbolic name in the location field, which is assigned to the first word of the generated code.

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

- They must be symbolic of one or two 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.

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
```

LOC

The LOC instruction is optional and allows the use of the same symbols in macros and programs to avoid doubly defined symbols. Symbols 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 instructions. A symbol in the location field of the LOC instruction is ignored. The format is:

```
LOC     s1,s2,...,sn
```

Where: s_i are local symbols in the address field that must conform to the following rules:

- They must be symbolic names of one or two 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.

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

Where: s is an identifying tag in the location field which is 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 is a string of one to six 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 is a 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 3.2.4) is encountered.

3.3.2 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. 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. Format 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.

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 p_2, p_4, and p_6 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)	If e is specified, s must be specified
COM	s(e)	
DAT	s(e)	
IFA	e_1,c,e_2	c must be specified
IFC	a_1,c,a_2	

The 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.

Nesting Macros

Calls to either library macros or programmer-defined macros can be nested in a macro definition. Recursive calls to nested macros are allowed to the 10th level. Further recursive calls are ignored.

Local location symbols are unique to each level of nesting.

Example:

```
XYZ     MAC     p1,p2,p3,p4,p5
        LOC     A
        LDA     'p1'
        'p2'    'p3'
        S'p4'A  'A'-*-1
        JMP'.'  'p5'
'A'     ENA     1
        EMC
```

Macro skeleton

3.3.3 MACRO INSTRUCTIONS

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 $p_1, p_2, \ldots, p_n$

Where: s is a symbolic name in the location field that is assigned to the first word of the generated code.

N is a 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 Δ, +, -, or *.

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

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 must be separated from the next by a comma.
- The list must be 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 73rd 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.

3.3.4 ASSEMBLY TEXT GENERATION AND USE

CLASS has the capability of creating or calling an assembly text of predefined symbols and macro definitions. This text may be generated or called by use of special parameters on the CLASS call card. (See Appendix B for a complete explanation of call card parameters.)

In order to generate assembly text, M=FNAME should be specified on the CLASS call card. The source program to be assembled should contain only the symbols and macro definitions to be included in the text. Each macro definition should begin with a MAC pseudo operation and end with an EMC pseudo operation. Symbols should be defined with an EQU statement.

After assembling the program, CLASS will build a file containing the predefined symbols with their values, the macro skeleton images, and a macro name table. The file name will be the name specified by the M parameter. The text may be saved for future use by making it a system file or by making it a user file.

The assembly text can be utilized by specifying F=FNAME or G=FNAME on the CLASS call card. The text must have already been generated and reside either in the system or on a user-supplied file. The G parameter is used if the text is a user file.

CLASS reads in the file and adds the predefined symbols and macros to its tables prior to assembly. If macros are redefined during assembly, the newest definition will be used. If symbols are redefined, an error will be generated.

3.4 CLASS LIMITATIONS

3.4.1 WARNINGS

CLASS allows the following storage reference case to be assembled without error:

```
LDA+      ($C5)
```

which is assembled as:

```
C400
80C5
```

This would cause different results in 32K and 65K machines.

CLASS requires long relative references to relative externals and long absolute references to absolute externals when the enhanced instructions are used.

User-defined macros may have any six character names except IFK, IFR, PCO, or any Type 2 machine instruction name; and macro names may not begin with the characters O0.

The use of SPC 0 causes the assembler to lose the line count for the current page.

3.4.2 SPECIAL CHARACTERS

CLASS uses several special characters in its processing to flag various conditions. They may not be used on the input source cards or unpredictable errors may result. The special characters that cannot be used are:

Character	6000 Display Code	Hollerith Punch
¬	76	12-8-6
↑	70	11-8-5
↓	71	11-8-6
^	72	12-0

Because there is a discrepancy between the codes for a colon on the CYBER 18 computers and on the CYBER 170/70/6000 computers, CLASS accepts as a colon an 8-5 punch, as specified in the CYBER 18 Macro Assembler reference manual, or an 8-2 punch as specified in the CYBER 170/70/6000 manuals. Since the colon has special meaning in the ALF pseudo instruction, it is not advisable to attempt to insert the colon in core using the ALF pseudo instruction.

CLASS expects an 8-4 punch for an apostrophe from the 026 card punch, but this is printed as a quote.

In general, special characters should be used with caution because of the discrepancies between the various codes and keypunches.

OUTPUT 4

There are two types of output from the CLASS assembler:

- Relocatable Binary Output
- List Output

4.1 RELOCATABLE BINARY OUTPUT

CLASS produces relocatable binary output in three formats: P8 for punching up to 80-column cards, B for the CYBER Cross System, and RB for magnetic tape input to the MSOS Loader. Files P8 and B have the necessary identifiers to be recognizable to the MSOS peripheral device drivers; i.e., 7/9 punch checksum and word count. File P8 is written one program per record and file B is written one block (BZS, RBD, etc.) per record. File RB has no checksums and is written one block per record. It should be written on a stranger (S-format) tape if the tape is to be loaded by the MSOS Loader.

The assembler specifies relocatable binary blocks by the type of indicator field in bits 15 through 13 of the first word of the block. The following block types are defined:

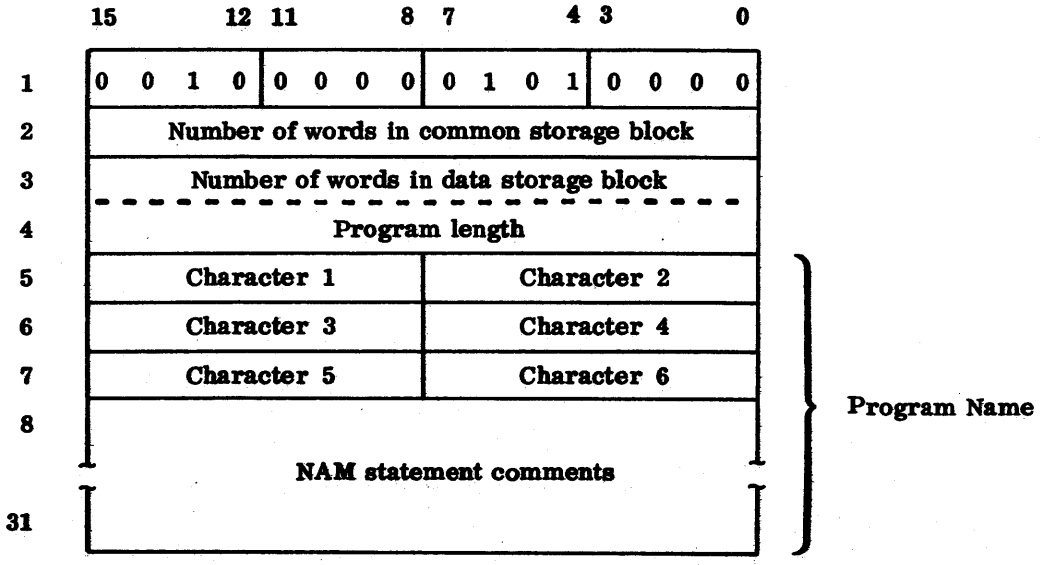
Type	Indicator	Description
NAM	001	Name block
RBD	010	Command sequence block
BZS	011	Zero storage block
ENT	100	Entry point block
EXT	101	External name block
ENF	000	Entry field block
EXF	111	External field block
XFR	110	Transfer address block

Output begins with a NAM block and terminates with an XFR block. The EXT and EXF blocks follow the RBD blocks. The RBD, BZS, ENT, and ENF blocks may come in any order.

The following is the format for the eight block types.

NAM Block

The NAM block contains a word count for common and data storage, the program length, and the name of the program.



RBD Block

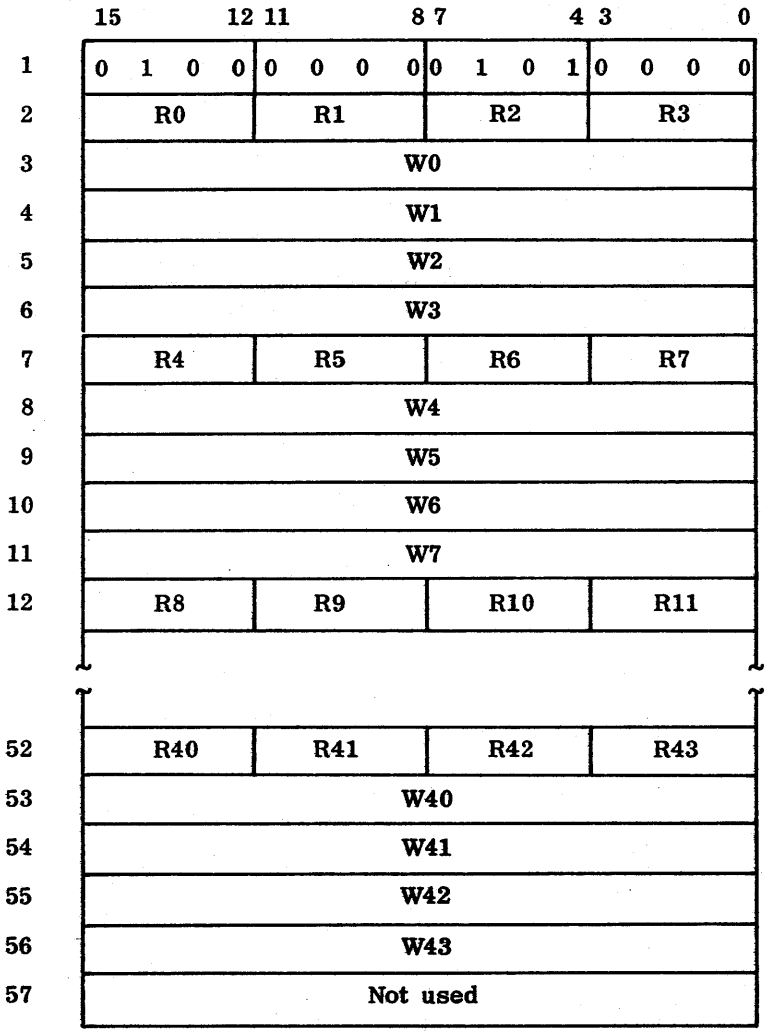
An RBD block contains a portion of the actual command sequence data of the program.

Words 2 through 57 contain the relocation bytes and words for the command sequence input. Each relocation byte is a four-bit indicator that identifies a word of the command sequence input as an absolute 15-bit address or as a 15-bit address relative to some relocation base. The relocation base for a word is determined by the particular combination of bit settings within the relocation byte.

The following are the relocation bytes in RBD blocks:

0000	Absolute (no relocation)
0001	Positive program relocation
0101	Negative program relocation
0010	Positive common storage relocation
0110	Negative common storage relocation
0011	Positive data storage relocation
0111	Negative data storage relocation

The core image of the RBD block is:



Where: Wn is the nth word of the input block (n = 1 to 43).

Rn is the relocation byte of the nth word.

W0 is the origin address of the input block

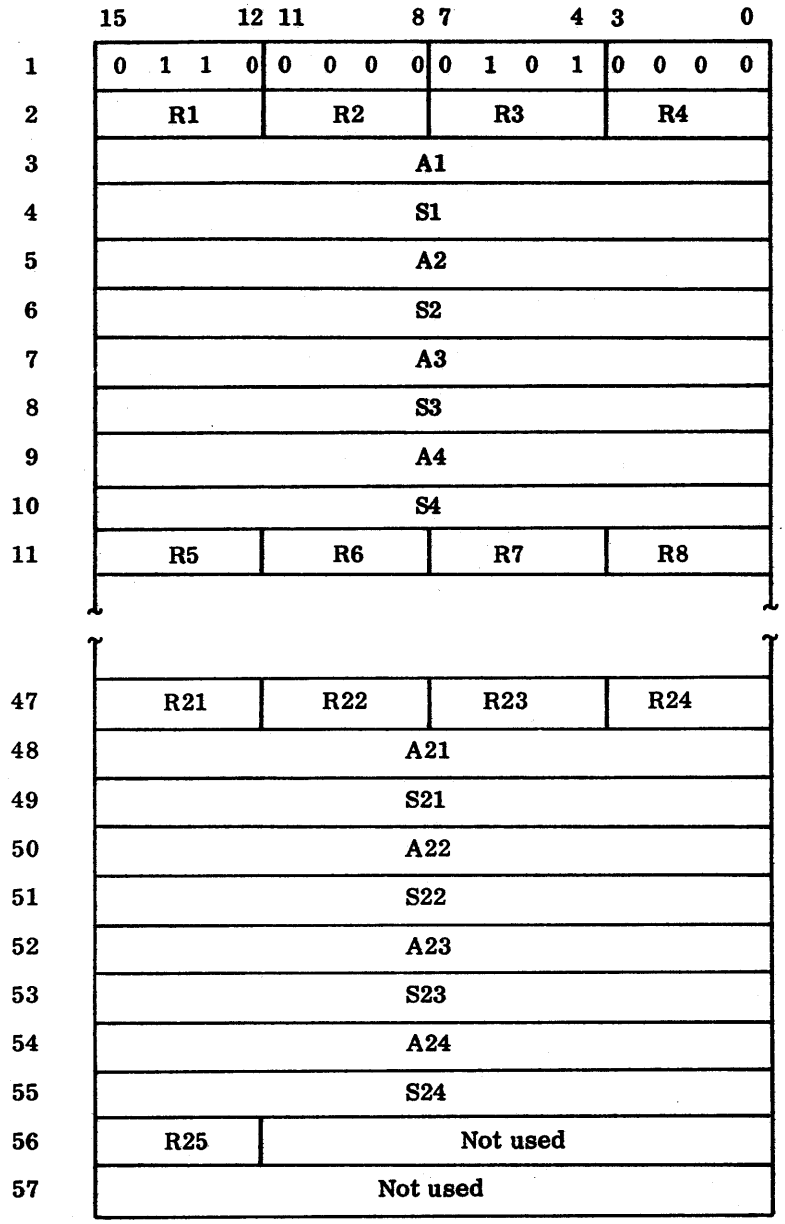
R0 is the relocation byte for w0.

There is one relocation byte for every word in the command sequence output and a maximum of 45 entries in the RBD block. The first word is the address relative to the start of the program where the loader begins storing command sequence data. The relocation byte for the first word address (storage address) of an RBD block may be 0000, 0001, or 0011. If the field contains a number larger than 0011, 0011 is assumed. Zero is the leading bit for all but the last relocation byte; 1 is the leading bit for the last relocation byte.

BZS Block

A BZS block contains relocation bytes, the starting address, and block sizes for areas of core to be cleared to 0s when the program is loaded.

The core image of the BZS block is:



Where: A is the starting address.

S is the size of the area reserved by BZS.

R is the relocation of the starting address.

An is the starting address of the nth entry.

Sn is the size of the BZS reservation for the nth entry.

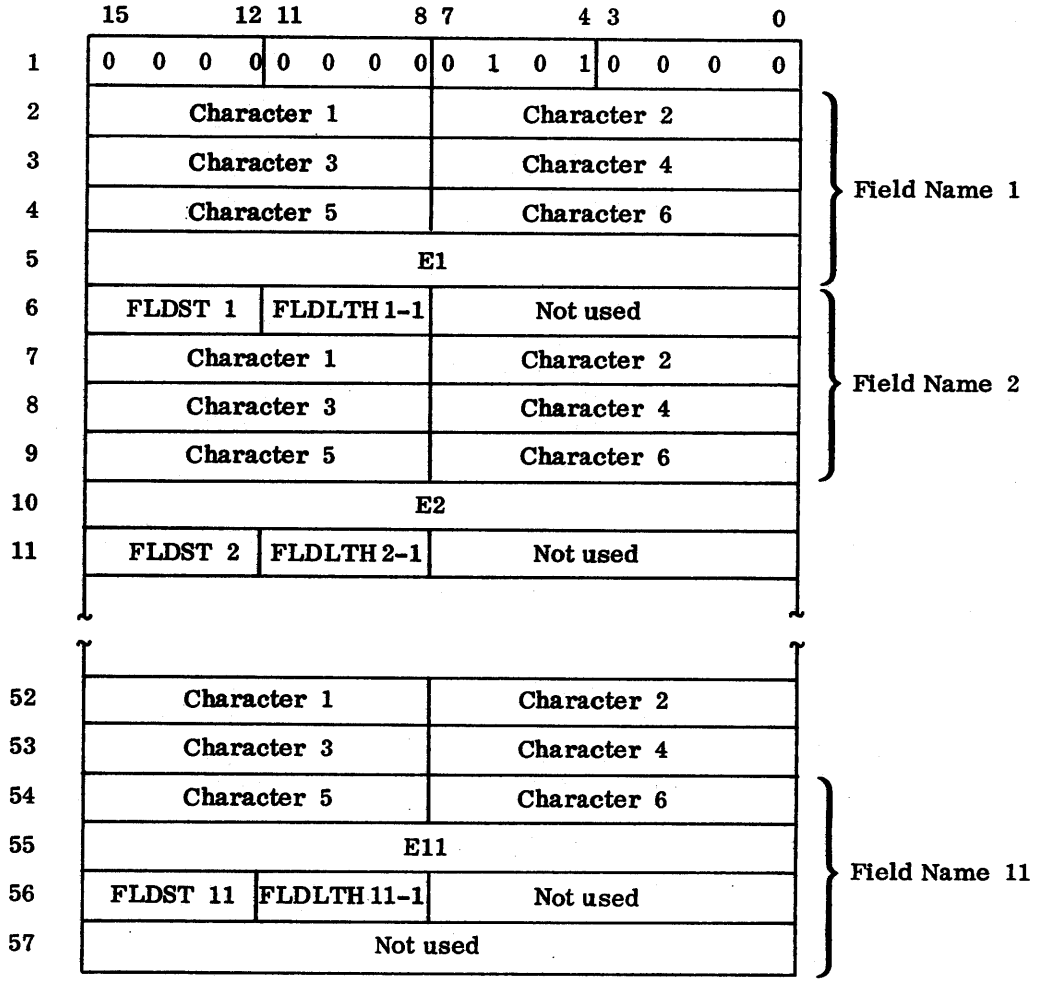
Rn is the relocation byte of the nth entry.

The relocation bytes for a starting address may be 0000, 0001, or 0011.

ENF Block

Up to 11 entry fields may be specified in an ENF block. The end of data in this block is identified by 0s. If the sign bit of a word containing the entry point address is 0, the address is program relocatable. If the sign bit is 1, the address is absolute and in ones complement. Data begins in word 2.

The core image of the ENF block is:



Where: **Name n** is a six-character name of the nth entry in the block.

En is the entry address of the nth field name. En is negative (ones complement) if absolute and positive if relative.

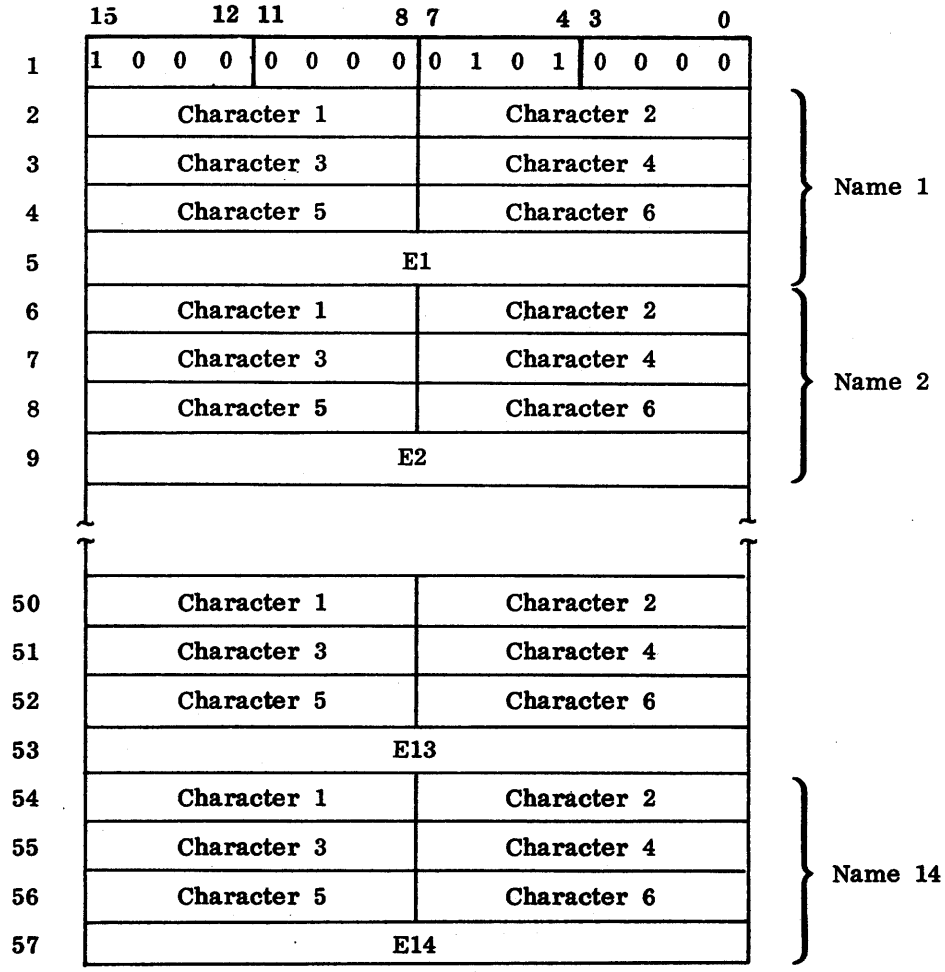
FLDST n is the leftmost bit of the nth field. $0 \leq$ FLDST n ≤ 15

FLDLTH n is the length of the nth field. $1 \leq$ FLDLTH n ≤ 16

ENT Block

Up to 14 entry point names and addresses may be included in an ENT block. The end of data in this block is identified by 0s. If the sign bit of a word containing the entry point address is 0, the address is program relocatable. If the sign bit of the word is 1, the address is absolute and in ones complement. Data begins in word 2 and extends to word 57.

The core image of the ENT block is:



Where: Name n is a six-character name of the nth entry in the block.

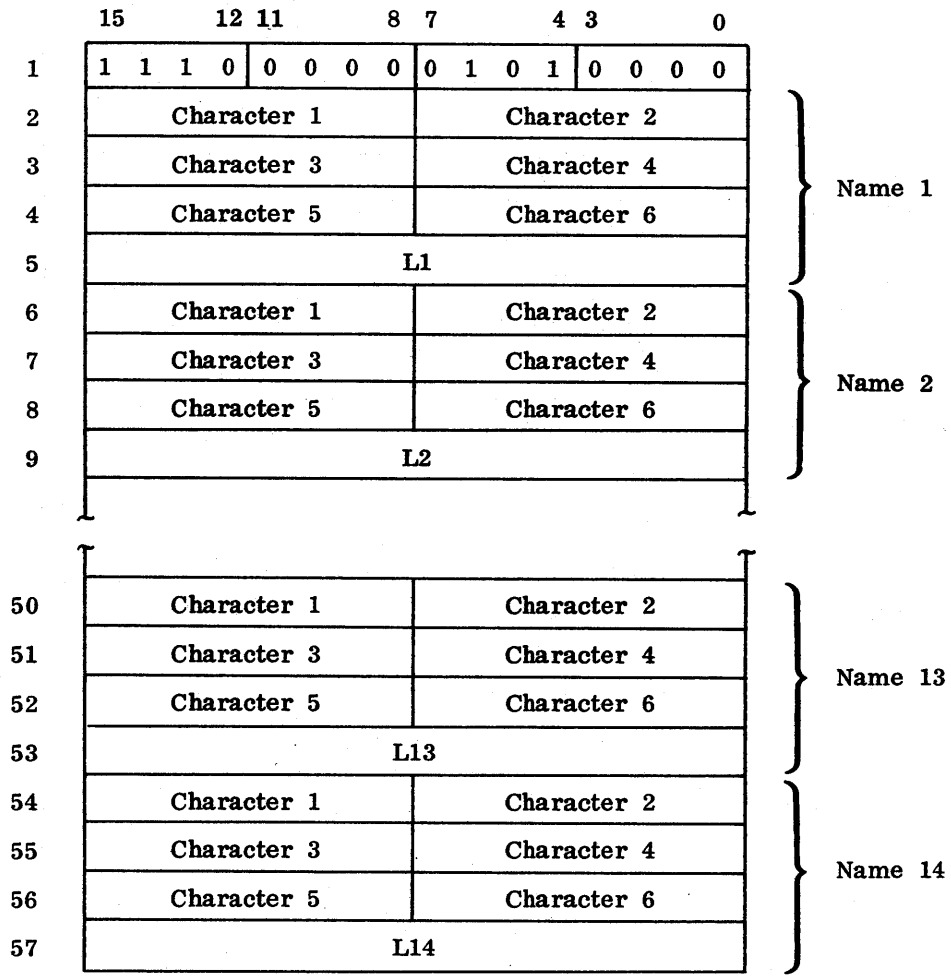
En is the entry point address of the nth name. En is negative (ones complement) if absolute and positive if program relocatable.

When processing an ENT block, the loader records the entry point name in its table. The entry point address is adjusted for relocation (either program or absolute), then it is recorded in the table of entry points. This procedure is repeated until the end of input is reached (a name equal to 0).

For each name, the loader determines if an entry point has been previously recorded in the table. If so, a duplicate entry error has occurred. The same entry point name may not be used by two programs to occupy memory space at the same time. Only the first occurrence of an entry point name is valid; others are illegal and are not loaded.

EXF Block

Up to 14 external fields and link addresses may be included in an EXF block. The core image of the EXF block is:



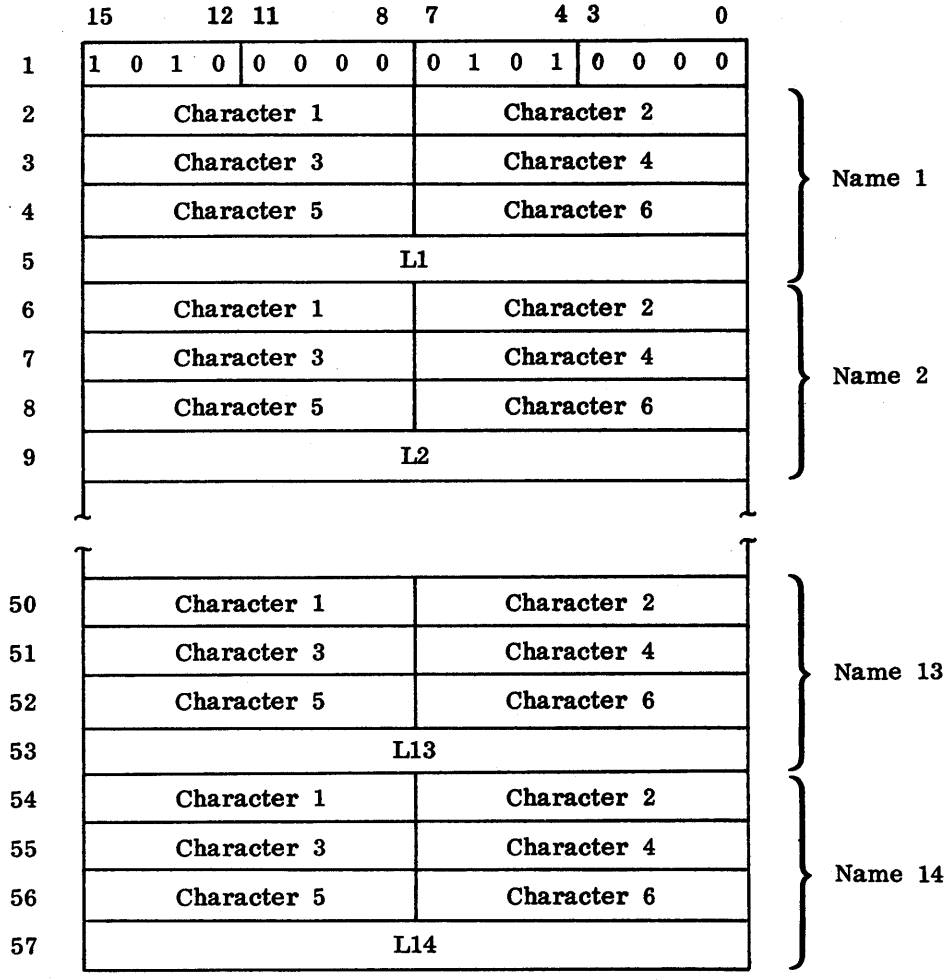
Where: Name n is a six-character name of the nth entry in the block.

Ln is the link address of the nth name. LN is negative (ones complement) if absolute and positive if relative.

The end of the EXF block is indicated by 0s. If the sign bit of the word containing the link address is 0, the address is program relocatable. If the sign bit is 1, the address is absolute and in ones complement. The format of the data in the block is the same for EXF as for ENT information. Relative external fields are indicated by setting the leftmost bit of the word containing character 1 of the field name.

EXT Block

Up to 14 external names and link addresses may be included in an EXT block. The core image of the EXT block is:



Where: Name n is a six-character name of the nth entry in the block.

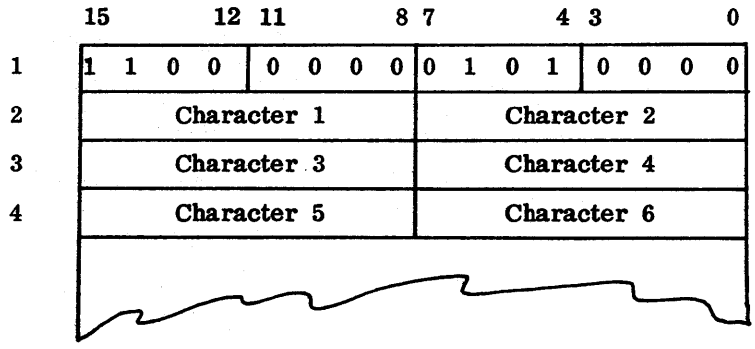
Ln is the link address of the nth name. Ln is negative (ones complement) if absolute and positive if relative.

The end of the EXT block is indicated by 0s. If the sign bit of the word containing the link address is 0, the address is program relocatable. If the sign bit is 1, the address is absolute and in ones complement. The format of the data in the block is the same for EXT as for ENT information. Relative externals are indicated by setting the leftmost bit of the word containing character 1 of the name.

XFR Block

The XFR block contains a transfer address (in words 2 to 4), which is six ASCII characters in length, including trailing spaces. The transfer address must be an entry point in the program being loaded or in another program loaded during the same load operation.

The core image of the XFR block is:



4.2 LIST OUTPUT

4.2.1 LIST OPTIONS

CLASS contains list options that can be requested on the CLASS call card. These options provide added information in a highly usable form for the programmer or simply make the listing more readable. To request one or several of the options, the parameter LO = should be specified, followed by the letter(s) corresponding to the option(s) desired.

Example:

```
LO = BMRTX
```

Note that no commas separate the option letters.

The following options are available:

Identifier	Option
B	List the BSS/BZS blocks on the banner page.
C	List the program list control cards (SPC, EJT, etc.).
D	Suppress the comment cards.
E	Process EJT as a page eject.
I	List the code skipped by the IFA pseudo option.
L	List the macro cross-reference table.
M	List all entries on multiword entries.
R	List the full reference map.
S	List the abbreviated reference map.
T	Tidy the list file (convert free form to set columns).
X	Expand the macro code into macro calls.

If more than seven options are desired, a second LO must be specified. If no LO option is specified, options B, M, R, and T are selected. If LO is specified, only the options specified are selected.

4.2.2 BANNER PAGE

CLASS outputs a banner page at the front of the assembly listing for every program. The information on this page consists of the following:

- A title line, giving the name of the program, CLASS version level, date, time it was executed, and page number
- A subtitle, STORAGE ALLOCATION
- The address and length of the program
- The address of the END card
- All entry points with their addresses, listed alphabetically
- All external symbols, listed alphabetically

If LO option B is selected, BSS/BZS blocks will also be listed with their type (local, common, or data), address, and length.

4.2.3 MAIN PROGRAM LISTING

The assembly list output by CLASS consists of 34 columns of descriptive information related to the source statement, followed by a maximum of 80 columns listing the source statement. This is illustrated in Table 4-1.

Each page is headed by a title line that gives the name of the program, CLASS version level, date, time of execution and page number.

The last page of the listing contains the amount of storage used, the number of source statements, a symbol count, the number of references, the time used for assembly, and an error count in decimal if errors are present.

4.2.4 ERROR SUMMARY

If assembly errors are encountered by CLASS during the assembly of the program, an error summary is listed following the assembly listing. The information given includes the type of error found and the page and line number where it was encountered. All errors of the same type are grouped together in the order of their occurrence. Error types that are included in this summary are:

- **Doubly defined symbol**
- **Undefined symbol**
- **Illegal expression**
- **Illegal operation code**
- **Illegal relocation**
- **Numeric operand overflow**
- **Macro definition error**
- **Macro instruction error**
- **Illegal symbol name**
- **Programmer-defined error**

Example:

	(Page No./Line No.)			
Illegal operation code	2/05	3/04	4/04	6/21
Illegal relocation	3/1	5/17		

Table 4-1. Listing Page Format

COLUMN	CONTENTS
1-2	The line number; given only on lines that are multiples of five; i.e., lines 5, 10, 15, etc.
3-4	Two spaces
5-6	The error mnemonic (see Appendix D for mnemonics); if no error is on this line, it is left blank.
7	A space
8	The relocation designator for the location: P Program relocation D Data relocation C Common relocation
9-12	Location in hexadecimal
13-17	Spaces
18-21	The first machine word in hexadecimal
22	Space
23	The relocation designator if the word is a one-word instruction: 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
24	Space
25-28	The second machine word if this is a two-word instruction; otherwise, it is blank.
29	Space
30	The relocation designator if this is a two-word instruction. Uses the same code as the one-word relocation designator.
31-34	Spaces
35-114	Input source statements: If the tidy option is selected on the CLASS call card (see Appendix B), the source statements will be put in the columns specified. If the T option of LO parameter is specified (see Section 4.2.1), the fields will be put in the following columns of the listing: Starting Column — Field 35 — Location 45 — Operation Code 52 — Address 64 — Comments 108 — Sequence Number

4.2.5 COMPLETE REFERENCE MAP

To obtain a complete reference map, specify R in the LO option list. The following information is given in the complete reference map:

1. Symbol names (in alphabetical order)
2. Symbol values
3. If the symbol is in the common or data storage area external to the program or is a system symbol (such as the I register), it is flagged by the following:

   /common/
   DATA
   EXTERNAL
   -SYSTEM-
   ABSOLUTE

4. The page number/line number in chronological order, for all references to the symbol. An identifying letter is printed following the page/line number references that have special meanings. These are the following:

Letter	Meaning
L	The location of the symbol
B	Where it is defined as a BSS or BZS block
E	Where it is defined as an entry point
X	Where it is defined as an external
Q	Where it is equated
S	Where it is referenced as the address of a store instruction (Type 1)
F	Where it is defined as a field

5. If the symbol is undefined or doubly defined, it will be preceded with a U or a D, respectively.

4.2.6 SHORT REFERENCE MAP

To obtain an abbreviated reference map, specify S in the LO option list. The following information is given in the short reference map:

1. Symbol names (in alphabetical order)
2. Symbol values
3. A letter indicating the type of relocation:

P	Program
D	Data
C	Common
S	System

Example:

```
ASSEMBLY OF TEMPTY
ABBREVIATED REFERENCE MAP.
I     I         OOFF/   S
S     SAVE      0005/   P
```

4.2.7 MACRO CROSS-REFERENCE MAP

To obtain the macro cross-reference map, specify L in the LO option list. The following information is given in the macro cross-reference map:

1. The macro names (in alphabetical order)
2. The number of formal parameters
3. The number of local parameters
4. The page number/line number where the macro is referenced (called)

If there has been an error on the macro definition, an M error code will be printed before the macro name.

Examples:

```
      MACRO1      0004P     0001L     2/21      3/10
M     MACRO2      0002P     0000L     1/15      5/3       5/17
```

INSTALLATION OF CLASS A

CLASS must be assembled under COMPASS and requires KRONTXT as an assembly text. The CLASS release materials consist of a program library tape of CLASS, KRONTXT, associated common decks, and assembly text.

The following simplified installation procedure is provided as an example of CLASS installation under NOS/BE 1.1. It is assumed that the program libraries have been made available as permanent disk files.

1. Installation of CLASS as permanent files:

```
JOB,CM60000.
ATTACH(OLDPL,CLASSPL)
UPDATE(Q)
RETURN(OLDPL)
ATTACH(OLDPL,KRONTXTPL)
UPDATE(Q,C=TEXT)
REQUEST(KRONTXT,*PF)
COMPASS(I=TEXT,B=KRONTXT,L=0)
REQUEST(CLASS,*PF)
COMPASS(I,G=KRONTXT,B=CLASS)
RETURN(OLDPL,TEXT,COMPILE)
ATTACH(OLDPL,MACROPL)
UPDATE(Q)
REQUEST(SMAC17,*PF)
CLASS(I,M=SMAC17)
CATALOG(KRONTXT,KRONTEXT)
CATALOG(CLASS,CLASSBIN)
CATALOG(SMAC17,MACBIN)
7/8/9
*COMPILE CLASS
7/8/9
*COMPILE KRONTXT
7/8/9
*COMPILE MACROS
6/7/8/9
```

2. **Installation of CLASS as system-resident files:**

```
JOB,CM60000.
ATTACH(OLDPL,CLASSPL)
UPDATE(Q)
RETURN(OLDPL)
ATTACH(OLDPL,KRONTXTPL)
UPDATE(Q,C=TEXT)
COMPASS(I=TEXT,B=KRONTXT,L=0)
COMPASS(I,G=KRONTXT,B=CLASS)
RETURN(OLDPL,TEXT,COMPILE)
ATTACH(OLDPL,MACROPL)
UPDATE(Q)
CLASS(I,M=SMAC17)
REWIND(CLASS,SMAC17)
EDITLIB(SYSTEM,ERROR=3)
7/8/9
*COMPILE CLASS
7/8/9
*COMPILE KRONTXT
7/8/9
*COMPILE MACROS
7/8/9
READY(SYSTEM)
LIBRARY(NUCLEUS,OLD)
REPLACE(*,KRONTXT)
REPLACE(*,CLASS)
REPLACE(*,SMAC17)
FINISH.
COMPLETE.
ENDRUN.
6/7/8/9
```

CLASS CONTROL CARD PARAMETERS B

The control card used to call CLASS has the following format:

CLASS(p_1,p_2,...,p_n)

The parameters (p_1,p_2, etc.) that may be specified on the call card are:

I	Input file name.
L	Output file name.
B, P8, RB	Assign binary file names.
T	Tidy tab columns (6-digit decimal number).
LO	List options.
NR	Do not rewind input file before assembly.
M	Build assembly text.
F	Call assembly text from system file.
G	Call assembly text from user file.

These parameters may be in any order and must be in one of the following forms:

Omitted	Default
p_i	The parameter is set to the alternate default option.
p_i=x	Parameter p_i is set to the option indicated by x.

The following is a more detailed description of the parameters:

Input File Name:

Blank	Input on file INPUT
I	Input on file COMPILE
I=FNAME	Input on file FNAME

Output File Name:

Blank	**Full list on file OUTPUT**
L	**Full list on file OUTPUT**
L=FNAME	**Full list on file FNAME**
L=0	**No list output**

Assign Binary File Name:

Blank	**Binary output on file LGO.**	**Punch output on file PUNCH80, tape output on file RBIN.**
B	**Binary output on file LGO.**	
B=FNAME	**Binary output on file FNAME.**	
B=0	**No binary output.**	
P8=FNAME	**Punch output on file FNAME.**	
RB=FNAME	**Tape output on file FNAME.**	

Tidy Tab Columns:

Blank	**Tabs are set at columns 11, 18, 31.**
T	**Tabs are set at columns 11, 18, 31.**
T=nnnnnn	**Three 2-digit decimal tabs are set at the columns specified. Each location symbol begins in column 1.**

List Options:

Blank	Options B, M, R, and T are selected.
LO	Options B, M, R, and T are selected.
LO=xxxxxx	Where xxxxxx may be any combination of the following:

B	List the BSS/BZS blocks on the banner page.
C	List the program control cards (SPC, EJT, etc.).
D	Suppress the comment cards.
E	Process EJT as eject.
I	List the code skipped by the IFA pseudo operation.
L	List the macro cross-reference.
M	List all entries on multiword instructions (i.e., DEC, ALF, etc.).
R	List the full reference map.
S	List the abbreviated reference map.
T	Tidy the listing columns (11, 18, 31 are default).
X	Expand the macro code.

If more than seven list options are desired, a second LO must be specified.

If LO=xxxxxx is specified, only the options specified are selected.

Do Not Rewind:

Blank	Rewind input file.
NR	Do not rewind input file.

Build Assembly Text:

Blank	No assembly text will be generated.
M=FNAME	Assembly text is generated on file FNAME.

Call Assembly Text:

Blank	No assembly text is called.
F	Assembly text is called from system file SMAC17.
F=FNAME	Assembly text is called from system file FNAME.
G	Assembly text is called from the user file SMAC17.
G=FNAME	Assembly text is called from the user file FNAME.

In order to call an assembly text, it must have been built prior to this run and must reside on the file called. A maximum of one assembly text file is allowed.

CONTROL CARDS FOR JOB RUN C

The following are sample deck structures for assembling CYBER 18-17 source code using CLASS.

Examples:

1. Using CLASS installed as permanent files:

```
JOB,CM70000.
ATTACH(CLASS,CLASSBIN,ID=xxxx)
ATTACH(SMAC17,MACBIN,ID=xxxx)
CLASS(G)
7/8/9
(SOURCE CODE)
6/7/8/9
```

2. Using CLASS installed as system-resident files:

```
JOB,CM70000.
ASSEM(F)
7/8/9
(SOURCE CODE)
6/7/8/9
```

ERROR MESSAGES D

D.1 ASSEMBLY ERROR MESSAGES

When an error occurs during assembly, it is flagged with a two-character error mnemonic in columns 5-6 on the listing. If more than one error occurs in the same line of code, the most recent error encountered is the one that is flagged. However, all errors are reflected in the total error count.

Error Mnemonic	Meaning
DS	Doubly defined symbol
UD	Undefined symbol
EX	Illegal expression
OP	Illegal operation code
RL	Illegal relocation
OV	Numeric operand overflow
IS	Illegal symbol name
PD	Programmer-defined error

D.2 MACRO ERROR DIAGNOSTIC MESSAGES

When an error is encountered in a macro definition or call, a two-character error mnemonic appears on the line following the erroneous code along with a diagnostic message.

Error Mnemonic	Diagnostic Message
MD	*/MAC DEF ERROR NO NAME
MD	*/MAC DEF ERROR BAD PARAM
MD	*/MAC DEF ERROR NO OPCODE
MD	*/MAC DEF ERROR BUFFER OVE
MD	*/MAC DEF ERROR BAD TERM

Error Mnemonic	Diagnostic Message
MC	*/MACRO CALL ERROR NEST GT 10
MC	*/MACRO CALL ERROR BUFFER OV
MC	*/MACRO CALL ERROR PARAM ERR
MC	*/MACRO CALL ERROR BAD CONT
MC	*/MACRO CALL ERROR BAD DEF

D.3 MISSING END CARD

If no END card is encountered by CLASS at the end of the program, the following message is output at the end of the listing:

/// END CARD MISSING ///

D.4 DAYFILE ERROR MESSAGES

If CLASS encounters a control card error or insufficient field length has been assigned for the job, an error message is output to the dayfile and the job is aborted. The following messages are output to the dayfile, indicating the erroneous condition:

CONTROL CARD ERROR
NO INPUT FILE SPECIFIED
FILE NAME CONFLICT
INVALID TAB COLUMNS (for tidy feature)
INVALID LIST OPTIONS
INSUFFICIENT FL FOR CLASS
MEMORY OVERFLOW IN PASS 1
MEMORY OVERFLOW IN PASS 2

The following messages output to the dayfile indicate erroneous conditions that exist but which do not cause the job to abort:

MACTXT OVERFLOW
Q99 ILLEGAL TO CLASS
SYSTEM ERROR IN PASS 1 ASSEMBLY (if this message occurs, there is probably a bug in CLASS)
NN ERRORS IN xxxxxx (where xxxxxx is the program name)

ASSEMBLY MODIFICATIONS TO CLASS E

CLASS presently limits nesting to 10 calls per macro by use of a counter. If this maximum is to be changed, modify the following card:

```
MREF    EQU    10
```

The maximum range is 1 to 31.

If the buffer length is exceeded in producing a macro skeleton, modify the following card:

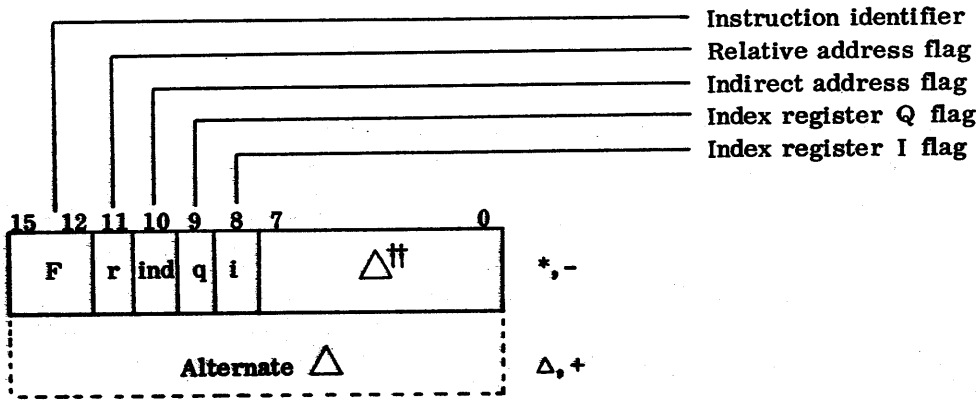
```
SKELBL  EQU    1080
```

F.1 STORAGE REFERENCE INSTRUCTIONS

The Type 1 source format is:

[Location]† OPCODE Address [, Index] [Comments]

The machine code format is:



Where: * and - terminators cause a one-word instruction.

△ and + terminators cause a multiword instruction.

The storage reference instructions and their meanings are:

Operation Code	Description	Contents of the F Field	Definition
ADD	Add A	8	Add the contents of EA††† to the contents of the A register.
ADQ	Add Q	F	Add the contents of EA to the contents of the Q register.

†Brackets indicate an optional field.

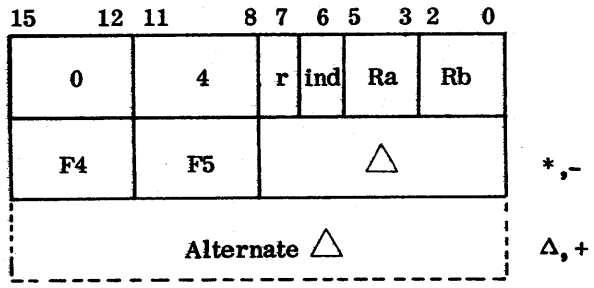
††The small △ denotes a blank, the larger △ denotes a machine code field.

†††EA is the effective address storage location; i.e., the final result of evaluating the address expression and the index register.

Operation Code	Description	Contents of the F Field	Definition
AND	AND with A	A	AND the contents of EA to the contents of the A register.
DVI	Divide integer	3	Divide the concatenated contents of the QA registers by the contents of EA. Put the quotient in the A register and the remainder in the Q register.
EOR	Exclusive OR with A	B	Exclusive OR the contents of EA with the contents of the A register.
JMP	Jump	1	Jump to EA.
LDA	Load A	C	Load the A register with the contents of EA.
LDQ	Load Q	E	Load the Q register with the contents of EA.
MUI	Multiply integer	2	Multiply the contents of EA with the contents of the A register. Put the 32-bit product in the QA registers.
RAO	Replace Add 1 in storage	D	Increment the contents of EA by 1. The contents of the A register are not altered.
RTJ	Return jump	5	Transfer the address of the next instruction to EA. Jump to EA + 1.
SPA	Store A, parity to A	7	Store the contents of the A register in EA. If there is an odd number of bits in the A register, set it to 0. If there is an even number of bits, set it to 1.
STA	Store A	6	Put the contents of the A register in EA. The contents of the A register are not altered.
STQ	Store Q	4	Put the contents of the Q register in EA. The contents of the Q register are not altered.
SUB	Subtract	9	Subtract the contents of EA from the contents of the A register. Put the result in the A register. The contents of EA are not changed.

Arithmetic operations are ones complement arithmetic. The OVERFLOW indicator is set if the result of the arithmetic operation is greater than the capacity of the destination register or storage location. It remains set until a skip on overflow instruction is executed.

The Type 2 machine code format is:



Operation Code	Description	Contents of the F4 Fields	F5	Rb	Definition
AMA	AND to memory, A	A	1	6	Form the logical product bit-by-bit of the contents of EA with the contents of the named register. Put the result in EA; the contents of the register are not altered, except for the A register, which receives the original contents of EA.
AMI	AND to memory, I	A	1	7	
AMQ	AND to memory, Q	A	1	5	
AM1	AND to memory, 1	A	1	1	
AM2	AND to memory, 2	A	1	2	
AM3	AND to memory, 3	A	1	3	
AM4	AND to memory, 4	A	1	4	
ANA	AND to A	A	0	6	Form the logical product bit-by-bit of the contents of EA with the contents of the named register. Put the result in the register; the contents of EA are not altered.
ANI	AND to I	A	0	0	
ANQ	AND to Q	A	0	5	
AN1	AND to 1	A	0	1	
AN2	AND to 2	A	0	2	
AN3	AND to 3	A	0	3	
AN4	AND to 4	A	0	4	
ARA	Add to A	8	0	6	Add the contents of EA to the contents of the named register. Put the result in the register; the contents of EA are not altered. Overflow as in ADD.
ARI	Add to I	8	0	7	
AR1	Add to 1	8	0	1	
AR2	Add to 2	8	0	2	
AR3	Add to 3	8	0	3	
AR4	Add to 4	8	0	4	
ARQ	Add to Q	8	0	5	
CAE	Compare A equal	E	0	6	Compare the contents of the named register to the contents of EA bit-by-bit. If equal, skip one location; otherwise, execute the next (one-word) instruction. The contents of EA and the named register are not altered.
CIE	Compare I equal	E	0	7	
CQE	Compare Q equal	E	0	5	
C1E	Compare 1 equal	E	0	1	
C2E	Compare 2 equal	E	0	2	
C3E	Compare 3 equal	E	0	3	
C4E	Compare 4 equal	E	0	4	

Operation Code	Description	Contents of the F4 Fields	F5	Rb	Definition
LRA	Load register A	C	0	6	Load the named register with the contents of EA. The contents of EA are not altered.
LRI	Load register I	C	0	7	
LRQ	Load register Q	C	0	5	
LR1	Load register 1	C	0	1	
LR2	Load register 2	C	0	2	
LR3	Load register 3	C	0	3	
LR4	Load register 4	C	0	4	
OMA	OR to memory, A	D	1	6	Form the logical sum (inclusive OR) bit-by-bit of the contents of EA with the contents of the named register. Put the result in EA; the contents of the register are not altered except for the A register, which receives the original contents of EA.
OMI	OR to memory, I	D	1	7	
OMQ	OR to memory, Q	D	1	5	
OM1	OR to memory, 1	D	1	1	
OM2	OR to memory, 2	D	1	2	
OM3	OR to memory, 3	D	1	3	
OM4	OR to memory, 4	D	1	4	
ORA	OR to A	D	0	6	Form the logical sum bit-by-bit of the contents of EA with the contents of the named register. Put the result in the register. The contents of EA are not altered.
ORI	OR to I	D	0	7	
ORQ	OR to Q	D	0	5	
OR1	OR to 1	D	0	1	
OR2	OR to 2	D	0	2	
OR3	OR to 3	D	0	3	
OR4	OR to 4	D	0	4	
SBA	Subtract to A	9	0	6	Subtract the contents of EA from the named register. Put the result in the register; the contents of EA are not altered. Overflow as in ADD.
SBI	Subtract to I	9	0	7	
SBQ	Subtract to Q	9	0	5	
SB1	Subtract to 1	9	0	1	
SB2	Subtract to 2	9	0	2	
SB3	Subtract to 3	9	0	3	
SB4	Subtract to 4	9	0	4	
SJA	Subroutine jump, A	5	0	6	Put the address of the last word of this instruction in the named register. Jump to EA. SJE does the jump to EA only.
SJE	Subroutine jump exit	5	0	0	
SJI	Subroutine jump, I	5	0	7	
SJQ	Subroutine jump, Q	5	0	5	
SJ1	Subroutine jump, 1	5	0	1	
SJ2	Subroutine jump, 2	5	0	2	
SJ3	Subroutine jump, 3	5	0	3	
SJ4	Subroutine jump, 4	5	0	4	

Operation Code	Description	Contents of the F4 Fields	F5	Rb	Definition
SRA	Store register A	C	1	6	Store the contents of the named register in EA. The contents of the register are not altered.
SRI	Store register I	C	1	7	
SRQ	Store register Q	C	1	5	
SR1	Store register 1	C	1	1	
SR2	Store register 2	C	1	2	
SR3	Store register 3	C	1	3	
SR4	Store register 4	C	1	4	
CCE	Compare character equal	E	2	†	Compare the contents of bits 0 to 7 of the A register with the specified character in the sum of EA + the contents of bits 1 to 15 in register Rb. Bit 0 = 0 in Rb specifies a left character; otherwise, a right character is implied. If equal bit-by-bit, skip one location; otherwise execute the next instruction.
LCA	Load character to A	C	2	†	Load to bits 0 to 7 of the A register character specified in the same manner as in CCE.
SCA	Store character from A	C	3	†	Store the contents of bits 0 to 7 of the register in the bits specified (same procedure as CCE).

F.2 FIELD REFERENCE INSTRUCTIONS

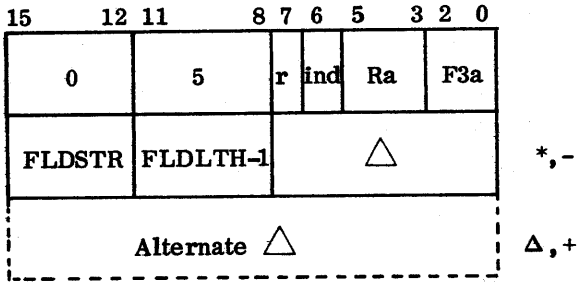
The Type 2 (only) source format is:

[Location] OPCODE Address, FLDSTR, FLDLTH,[Index] [Comments]

or

[Location] OPCODE Fieldname,[Index] [Comments]

The machine code format is:



†As specified

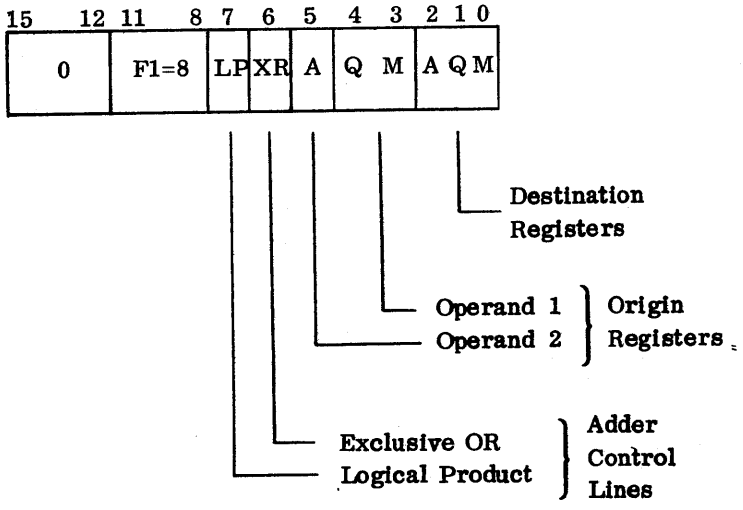
Operation Code	Description	Contents of the F3a Field	Definition
CLF	Clear field	6	Clear the bits in the specified field to all 0s.
LFA	Load field to A	4	Load the specified field in the A register right-justified, with leading 0s. The contents of EA are not altered.
SEF	Set field	7	Set bits in the specified field to all 1s.
SFA	Store field from A	5	Store the appropriate number of right-justified bits from the A register to the specified field. The contents of the A register are not altered.
SFN	Skip on field no zero	3	If all bits of the specified field are not 0, skip one location; otherwise, execute the next instruction. The field contents are not altered.
SFZ	Skip on field zero	2	If all bits of the specified field are 0, skip one location; otherwise, execute the next instruction. The field contents are not altered.

F.3 INTERREGISTER INSTRUCTIONS

The Type 1 source format is:

[Location] OPCODE Register [Comments]

The machine code format is:



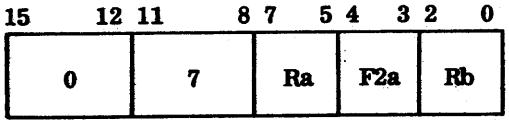
Operation Code	Description	Contents of the F1 Field
AAB	Transfer the arithmetic sum of A, Q, +M	8
AAM	Transfer the arithmetic sum of A, M	8
AAQ	Transfer the arithmetic sum of A, Q	8
CAB	Transfer the complement of the logical product of A, Q, +M	8
CAM	Transfer the complement of the logical product of A, M	8
CAQ	Transfer the complement of the logical product of A, Q	8
CLR	Clear to 0.	8
EAB	Transfer the exclusive OR, A, Q, +M	8
EAM	Transfer the exclusive OR, A, M	8
EAQ	Transfer the exclusive OR, A, Q	8
LAB	Transfer the logical product of A, Q, +M	8
LAM	Transfer the logical product of A, M	8
LAQ	Transfer the logical product of A, Q	8
SET	Set to 1s.	8
TCA	Transfer the complement of A	8
TCB	Transfer the complement of Q+M	8
TCM	Transfer the complement of M	8
TCQ	Transfer the complement of Q	8
TRA	Transfer the complement of A	8
TRB	Transfer the complement of Q+M	8
TRM	Transfer the complement of M	8
TRQ	Transfer the complement of Q	8

The instructions transfer data from one or two origin registers through the adder to any combination of A, Q, M destination registers. The OVERFLOW indicator is set when overflow actually occurs.

The Type 2 source format is:

[Location] OPCODE Register [Comments]

The machine code format is:



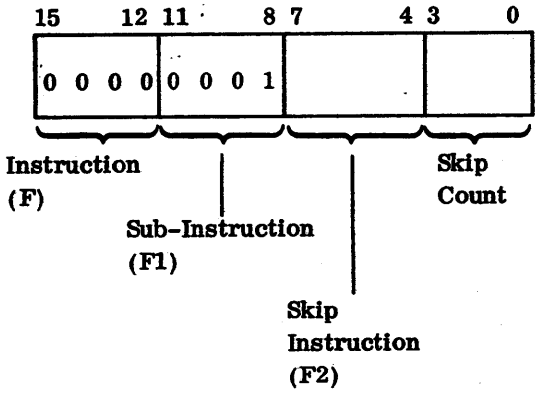
Operation Code	Description	Contents of the F2a Fields	Ra	Rb	Definition
XFA	Transfer A	0	6	*	These instructions transfer the contents of the named register to a specified destination register.
XFI	Transfer I	0	7	*	
XFQ	Transfer Q	0	5	*	
XF1	Transfer 1	0	1	*	
XF2	Transfer 2	0	2	*	
XF3	Transfer 3	0	3	*	
XF4	Transfer 4	0	4	*	

F.4 SKIP INSTRUCTIONS

The Type 1 source format is:

[Location] OPCODE SK [Comments]

The machine code format is:



*Destination register

Operation Code	Definition	Contents of the F2 Field	Definition
SAM	Skip if A minus	3	Skip if bit 15 of the register is 1.
SAN	Skip if A nonzero	1	Skip if all of the bits in the A register are not 0.
SAP	Skip if A $\geq$ 0	2	Skip if bit 15 of the A register is 0.
SAZ	Skip if A zero	0	Skip if all of the bits in the A register are 0.
SNF	Skip if no protect fault	F	Skip if the PROTECT FAULT indicator is 0.
SNO	Skip if no overflow	B	Skip if the OVERFLOW indicator is 0.
SNP	Skip if no parity error	D	Skip if the STORAGE PARITY indicator is 0.
SOV	Skip if overflow*	A	Skip if the OVERFLOW indicator is 1.
SPE	Skip if parity error*	C	Skip if the STORAGE PARITY indicator is 1.
SPF	Skip if protect fault*	E	Skip if the PROGRAM PROTECT indicator is 1.
SQM	Skip if Q minus	7	Skip if bit 15 of the Q register is 1.
SQN	Skip if Q nonzero	5	Skip if all bits in the A register are not 0.
SQP	Skip if Q $\geq$ 0	6	Skip if bit 15 of the Q register is 0.
SQZ	Skip if Q zero	4	Skip if all bits of the Q register are 0.
SWN	Skip if switch not set	9	Skip if the SELECTIVE SKIP switch is 0.
SWS	Skip if switch set	8	Skip if the SELECTIVE SKIP switch is 1.

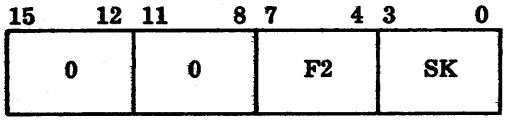
When the skip condition is met, the address of the next instruction to be executed is the location of the skip instruction plus the skip count, plus one.

The Type 2 source format is:

```
Location    OPCODE   SK
```

Where: SK is the skip count.

The machine code format is:



*Clears the appropriate indicator.

Operation Code	Description	Contents of the F2 Field	Definition
S1M	Skip if 1 minus	7	Skip if bit 15 of register 1 is 1.
S1N	Skip if 1 nonzero	5	Skip if all bits of register 1 are not 0.
S1P	Skip if 1 $\geq$ 0	6	Skip if bit 15 of register 1 is 0.
S1Z	Skip if 1 zero	4	Skip if all bits of register 1 are 0.
S2M	Skip if 2 minus	B	Skip if bit 15 of register 2 is 1.
S2N	Skip if 2 nonzero	9	Skip if all bits of register 2 are not 0.
S2P	Skip if 2 $\geq$ 0	A	Skip if bit 15 of register 2 is 0.
S2Z	Skip if 2 zero	8	Skip if all bits of register 2 are 0.
S3M	Skip if 3 minus	F	Skip if bit 15 of register 3 is 0.
S3N	Skip if 3 nonzero	D	Skip if not all bits of register 3 are 0.
S3P	Skip if 3 $\geq$ 0	E	Skip if bit 15 of register 3 is 0.
S3Z	Skip if 3 zero	C	Skip if all bits of register 3 are 0.
S4M	Skip if 4 minus	3	Skip if bit 15 of register 4 is 0.
S4N	Skip if 4 nonzero	1	Skip if all bits of register 4 are not 0.
S4P	Skip if 4 $\geq$ 0	2	Skip if bit 15 of register 4 is 0.
S4Z	Skip if 4 zero	0	Skip if all bits of register 4 are 0.

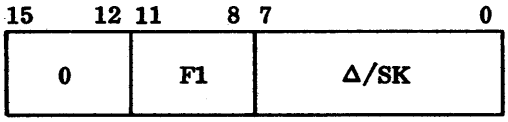
CLASS requires an absolute expression for Type 2 skip instructions.

F.5 REGISTER REFERENCE/SHIFT INSTRUCTIONS

The Type 1 (only) source format is:

Location OPCODE [Address] [Comments]

The machine code format is:

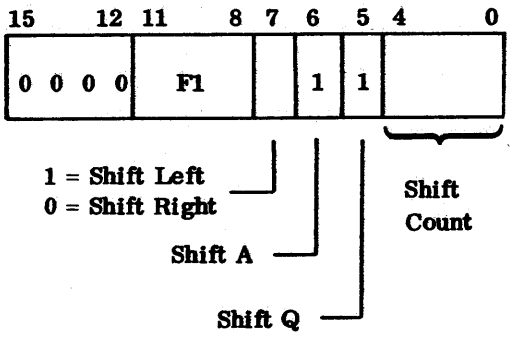


Operation Code	Description	Contents of the F1 Field	Definition
CPB	Clear program protect	7	Clear the protect bit in the address specified in the Q register.
EIN	Enable interrupt	4	Activate the interrupt system after one instruction following EIN has executed.
ENA	Enter A	A	Replace the contents of the A register with an 8-bit delta, sign extended.
ENQ	Enter Q	C	Replace the contents of the Q register with an 8-bit delta, sign extended.
EXI	Exit interrupt state	E	Exit from the interrupt state specified in delta. Automatically resets the OVERFLOW indicator, activates the interrupt system and jumps to the return address in the trap region.
IIN	Inhibit interrupt	5	Deactivate the interrupt system.
INA	Increase A	9	Increase the contents of the A register by a signed delta quantity. The appropriate overflow is generated.
INP	Input to A	2	Read one word to the A register from an I/O device specified in the Q register*.
INQ	Increase Q	D	Increase the contents of the Q register by a signed delta quantity. An appropriate overflow is generated.
NOP	No operation	B	Self-explanatory
OUT	Output from A	3	Output one word from the A register to an I/O device specified in the Q register*.
SLS	Selective step	0	Stop the computer if the SELECTIVE STOP switch is on. On restart, the next location is executed.
SPB	Set program protect	6	Set the program protect bit in the address specified by the Q register.

*For INP and OUT the next instruction is:

- The location of the I/O instruction plus one if the device sends a reply
- The location of the I/O instructions plus one, plus the signed delta if the device sends a reject
- The location of the I/O instruction plus the signed delta if an internal reject occurs

The machine code format is:



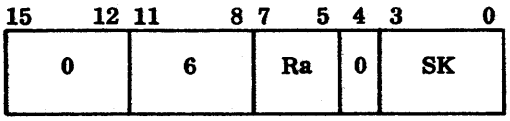
Operation Code	Description	Contents of the F1 Field	Definition
ALS	A left shift	F	Right shifts are end-off with sign extension in the upper bits; left shifts are end-around. Long shifts are for juxtaposed QA registers. The maximum long shift is 31_{10} places.
ARS	A right shift	F	
LLS	Long left shift	F	
LRS	Long right shift	F	
QLS	Q left shift	F	
QRS	Q right shift	F	

F.6 DECREMENT REGISTER AND SKIP REPEAT INSTRUCTIONS

The Type 2 (only) source format is:

[Location] OPCODE Skip count [Comments]

The machine code format is:



Operation Code	Description	Contents of the RA Field	Definition
DAP	Decrement & repeat if A ≥ 0	6	If bit 15 of the named register is 0, decrement the register contents by 1 and repeat (jump backwards the number of locations specified in skip count SK (unsigned)). The skip count must be specified as a positive absolute integer expression in the source code.
DIP	Decrement & repeat if I ≥ 0	7	
DQP	Decrement & repeat if Q ≥ 0	5	
D1P	Decrement & repeat if 1 ≥ 0	1	
D2P	Decrement & repeat if 2 ≥ 0	2	
D3P	Decrement & repeat if 3 ≥ 0	3	
D4P	Decrement & repeat if 4 ≥ 0	4	

F.7 MISCELLANEOUS INSTRUCTIONS

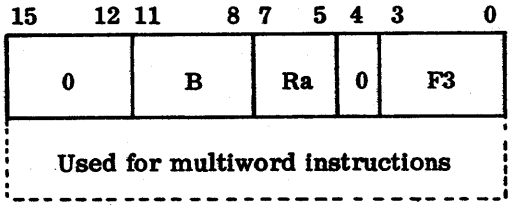
The Type 2 (only) source format is:

[Location] OPCODE [Register]

or

[Location] OPCODE Address

The machine code format is:



Operation Code	Description	Contents of the Ra	F3 Field	Definition
ASC	Accumulator scale	0	A	Shift the A register left end-around until bits 14 and 15 are different. The number of places shifted is placed in register 1 on completion. There is no shift if A is 0000 or FFFF.
CBP	Clear breakpoint interrupt	0	7	Clear the macro breakpoint interrupt.
DMI	Define micro interrupt	0	6	Define the use of one of the 12 available micro-interrupts.
EMS	Execute micro sequence	†	2	Transfer machine control to the upper micro-instruction at the address given in the specified register.
GPE	Generate character parity even	0	8	Set or clear bit 7 of the A register to cause parity of bits 0 to 7 to be even; other bits of the A register are not altered.
GPO	Generate character parity odd	0	9	Set or clear bit 7 of the A register to cause parity of bits 0 to 7 to be odd; other bits of the A register are not altered.
LLB	Load lower unprotected bounds register	†	1	Load the lower unprotected bounds register from the register specified.
LMM	Load micro memory	0	1	Load a block to micro memory, starting at the 1700 memory address in register 2 and the micro-memory address in register 1. The contents of Q is the number of 32-bit words to be loaded.
LRG	Load registers††	0	2	Load registers 1, 2, 3, 4, Q, A, I, M, and the overflow respectively starting at the address specified.
LUB	Load upper unprotected bounds register	†	0	Load the upper unprotected bounds register from the register specified.
SIO	Set/Sample input or output	0	4	Input or output the contents of the A register to an M05 peripheral device as specified by bits in the Q register.
SPS	Sample port/status	0	5	Replace the contents of the A register with the coded status of the M05 peripheral device specified by bits in the Q register. Clear the interrupt generated by M05.
SRG	Store registers††	0	3	Store registers 1, 2, 3, 4, Q, A, I, M, and the overflow respectively starting at the address specified.

†As specified in the instruction register field

††A two-word instruction that requires a + OPCODE terminator for external addresses.

ASCII CONVERSION TABLES G

The 1963 American Standard Code for Information Interchange (ASCII) is used by CLASS. ASCII code uses eight bits: bit 8, which is always zero, is omitted in the table below. Bits 1 through 4 contain the low-order four bits of code for the character in that row. Bits 5 through 7 contain the high-order three bits of the code for the character in that column. The code is given in ascending sequence.

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	
DC_3	001 0011	13	
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
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	

8-BIT ASCII CODES	171x-1 TTY ARRAY	171x-2 TTY ARRAY	026 PUNCHES	029 PUNCHES	6-BIT EXT. BCD MAG TAPE	8-BIT ASCII CODES	171x-1 TTY ARRAY	171x-2 TTY ARRAY	026 PUNCHES	029 PUNCHES	6-BIT EXT. BCD MAG TAPE
20_{16}	Space	Space	No Punch	No Punch	20_8	40_{16}	@	@	0-8-7	8-4	37_8
21†	!	!	11-8-2	12-8-7	52	41	A	A	12-1	12-1	61
22	"	"	8-7	8-7	17	42	B	B	12-2	12-2	62
23†	#	#	12-8-7	8-3	77	43	C	C	12-3	12-3	63
24	$	$	11-8-3	11-8-3	53	44	D	D	12-4	12-4	64
25†	%	%	0-8-5	0-8-4	35	45	E	E	12-5	12-5	65
26†	&	&	8-2	12	00 (35)††	46	F	F	12-6	12-6	66
27†	'	'	8-4	8-5	14	47	G	G	12-7	12-7	67
28†	(	(	0-8-4	12-8-5	34	48	H	H	12-8	12-8	70
29†	)	)	12-8-4	11-8-5	74	49	I	I	12-9	12-9	71
2A	*	*	11-8-4	11-8-4	54	4A	J	J	11-1	11-1	41
2B†	+	+	12	12-8-6	60	4B	K	K	11-2	11-2	42
2C	,	,	0-8-3	0-8-3	33	4C	L	L	11-3	11-3	43
2D	-	-	11	11	40	4D	M	M	11-4	11-4	44
2E	.	.	12-8-3	12-8-3	73	4E	N	N	11-5	11-5	45
2F	/	/	0-1	0-1	21	4F	O	O	11-6	11-6	46
30	0	0	0	0	12	50	P	P	11-7	11-7	47
31	1	1	1	1	01	51	Q	Q	11-8	11-8	50
32	2	2	2	2	02	52	R	R	11-9	11-9	51
33	3	3	3	3	03	53	S	S	0-2	0-2	22
34	4	4	4	4	04	54	T	T	0-3	0-3	23
35	5	5	5	5	05	55	U	U	0-4	0-4	24
36	6	6	6	6	06	56	V	V	0-5	0-5	25
37	7	7	7	7	07	57	W	W	0-6	0-6	26
38	8	8	8	8	10	58	X	X	0-7	0-7	27
39	9	9	9	9	11	59	Y	Y	0-8	0-8	30
3A	:	:	8-5	8-2	15	5A	Z	Z	0-9	0-9	31
3B	;	;	11-8-6	11-8-6	56	5B†	[	[	12-8-5	12-8-2	75
3C†	<	<	12-8-6	12-8-4	76	5C†	\	\	0-8-2	0-8-2	36
3D†	=	=	8-3	8-6	13	5D†	]	]	11-8-5	11-8-2	55
3E†	>	>	8-6	0-8-6	16	5E	↑	^	11-8-7	11-8-7	57
3F†	?	?	12-8-2	0-8-7	72	5F†	←	-	0-8-6	0-8-5	32

† Refer to note 2 below.
†† Refer to note 4 below.

NOTES

1. The 171x-2 TTY array is the ASCII 68, 64 character subset. This array is the same as used on the 171x-3 devices which receive from a 1774.
2. To operate in 026 punched card mode, ASCII 63 options are selected. To operate in 029 punched card mode, ASCII 68 options are selected. These options are assembly-time options for each driver affected.
3. The CDC Standard 1.10.003 is supported by an assembly option. For CDC ASCII mode of operation, the card punches 12-8-2 and 12-0 are stored internally as 7B. The card punches 11-8-2 and 11-0 are stored internally as 7D. For line printer operations, the internal codes 7B and 7D are converted to 5B and 5D to allow printing the hardware compatible graphic characters [(left bracket) and] (right bracket).
4. Since 173x magnetic tape controllers do not provide any code conversion, BCD code 00 is illegal and causes a noise record or BCD code 35 is substituted for the illegal 00 code to prevent tape errors.

   On tape write operations the ASCII codes 25_{16} (%) and 26_{16} (&) are written as BCD 35_8.

   On tape read operations the BCD code 35_8 is always translated to an ASCII $25 (%).

SCOPE 3.4
STANDARD CHARACTER SETS

CDC Graphic	ASCII Graphic Subset	Display Code	Hollerith Punch (026)	External BCD Code	ASCII Punch (029)	ASCII Code	CDC Graphic	ASCII Graphic Subset	Display Code	Hollerith Punch (026)	External BCD Code	ASCII Punch (029)	ASCII Code
:†	:	00†	8-2	00	8-2	3A	6	6	41	6	06	6	36
A	A	01	12-1	61	12-1	41	7	7	42	7	07	7	37
B	B	02	12-2	62	12-2	42	8	8	43	8	10	8	38
C	C	03	12-3	63	12-3	43	9	9	44	9	11	9	39
D	D	04	12-4	64	12-4	44	+	+	45	12	60	12-8-6	2B
E	E	05	12-5	65	12-5	45	-	-	46	11	40	11	2D
F	F	06	12-6	66	12-6	46	*	*	47	11-8-4	54	11-8-4	2A
G	G	07	12-7	67	12-7	47	/	/	50	0-1	21	0-1	2F
H	H	10	12-8	70	12-8	48	(	(	51	0-8-4	34	12-8-5	28
I	I	11	12-9	71	12-9	49	)	)	52	12-8-4	74	11-8-5	29
J	J	12	11-1	41	11-1	4A	$	$	53	11-8-3	53	11-8-3	24
K	K	13	11-2	42	11-2	4B	=	=	54	8-3	13	8-6	3D
L	L	14	11-3	43	11-3	4C	blank	blank	55	no punch	20	no punch	20
M	M	15	11-4	44	11-4	4D	, (comma)	, (comma)	56	0-8-3	33	0-8-3	2C
N	N	16	11-5	45	11-5	4E	. (period)	. (period)	57	12-8-3	73	12-8-3	2E
O	O	17	11-6	46	11-6	4F	≡	#	60	0-8-6	36	8-3	23
P	P	20	11-7	47	11-7	50	[	[	61	8-7	17	12-8-2	5B
Q	Q	21	11-8	50	11-8	51	]	]	62	0-8-2	32	11-8-2	5D
R	R	22	11-9	51	11-9	52	%††	%	63	8-6	16	0-8-4	25
S	S	23	0-2	22	0-2	53	≠	" (quote)	64	8-4	14	8-7	22
T	T	24	0-3	23	0-3	54	→	_ (underline)	65	0-8-5	35	0-8-5	5F
U	U	25	0-4	24	0-4	55	∨	!	66	11-0 or 11-8-2†††	52	12-8-7 or 11-0†††	21
V	V	26	0-5	25	0-5	56							
W	W	27	0-6	26	0-6	57	∧	&	67	0-8-7	37	12	26
X	X	30	0-7	27	0-7	58	↑	' (apostrophe)	70	11-8-5	55	8-5	27
Y	Y	31	0-8	30	0-8	59	↓	?	71	11-8-6	56	0-8-7	3F
Z	Z	32	0-9	31	0-9	5A	<	<	72	12-0 or 12-8-2†††	72	12-8-4 or 12-0†††	3C
0	0	33	0	12	0	30							
1	1	34	1	01	1	31	>	>	73	11-8-7	57	0-8-6	3E
2	2	35	2	02	2	32	≤	@	74	8-5	15	8-4	40
3	3	36	3	03	3	33	≥	\	75	12-8-5	75	0-8-2	5C
4	4	37	4	04	4	34	≠	^ (circumflex)	76	12-8-6	76	11-8-7	5E
5	5	40	5	05	5	35	; (semicolon)	; (semicolon)	77	12-8-7	77	11-8-6	3B

†Twelve or more zero bits at the end of a 60-bit word are interpreted as end-of-line mark rather than two colons. End-of-line mark is converted to external BCD 1632.

††In installations using the CDC 63-graphic set, display code 00 has no associated graphic or Hollerith code; display code 63 is the colon (8-2 punch).

†††The alternate Hollerith (026) and ASCII (029) punches are accepted for input only.

INDEX

COMMENT SHEET

CONTROL DATA CORPORATION

TITLE: CONTROL DATA CYBER Cross System Version 1
Macro Assembler Reference Manual

PUBLICATION NO. 96836500 **REVISION** B

This form is not intended to be used as an order blank. Control Data Corporation solicits your comments about this manual with a view to improving its usefulness in later editions.

Applications for which you use this manual.

Do you find it adequate for your purpose?

What improvements to this manual do you recommend to better serve your purpose?

Note specific errors discovered (please include page number reference).

General comments:

FROM NAME:_____ POSITION:_____

COMPANY NAME:_____

ADDRESS:_____

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

FOLD ON DOTTED LINES AND TAPE

CUT ON THIS LINE

TAPE

TAPE

FOLD

FOLD

FIRST CLASS
PERMIT NO. 8241

MINNEAPOLIS, MINN.

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

POSTAGE WILL BE PAID BY

CONTROL DATA CORPORATION
Publications and Graphics Division
215 Moffett Park Drive
Sunnyvale, California 94086

CUT ON THIS LINE

FOLD

FOLD

TAPE

TAPE

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

LITHO IN U.S.A.

CONTROL DATA CORPORATION