

HP ASSEMBLER Programmer's Reference Manual



11000 Wolfe Road
Cupertino, California 95014

First Edition, Feb. 1968
Revised, April 1970
Revised, June 1971

Copyright, 1971, by
HEWLETT-PACKARD COMPANY
Cupertino, California
Printed in the U.S.A.

Third Edition

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system (e.g., in memory, disc or core) or be transmitted by any means, electronic, mechanical, photocopy, recording or otherwise, without prior written permission from the publisher.

Printed in the U.S.A.

HP Computer Museum
www.hpmuseum.net

For research and education purposes only.

PREFACE

This publication is the reference manual for the Hewlett-Packard Assembly Language for the 2100 family of computers. Since Hewlett-Packard provides assemblers with all of its operating systems, this manual covers only the specifications of assembly language, not operating procedures for the assemblers. The user should refer to the appropriate system manual or operator's guide listed below:

SOFTWARE OPERATING PROCEDURES

DISC OPERATING SYSTEM (02116-91748)

MOVING-HEAD DISC OPERATING SYSTEM (02116-91779)

MAGNETIC TAPE SYSTEM (02116-91752)

In addition, the Formatter and other relocatable subroutines that can be called by relocatable assembly language programs are described in full in the RELOCATABLE SUBROUTINES manual (02116-91780). Interaction between relocatable programs and operating systems is described in:

BASIC CONTROL SYSTEM (02116-9017)

MOVING-HEAD DISC OPERATING SYSTEM

DISC OPERATING SYSTEM

MAGNETIC TAPE SYSTEM

Interaction between absolute programs and SIO drivers is described in an appendix to this book.

NEW AND CHANGED INFORMATION

All known errors in this manual have been corrected.
In addition, operating procedures have been eliminated and are now contained in the manuals listed on the previous page.

CONTENTS

iii	PREFACE
iv	NEW AND CHANGED INFORMATION
v	CONTENTS
vi	INTRODUCTION
1-1	SECTION I
	GENERAL DESCRIPTION
1-1	Assembly Processing
1-1	Symbolic Addressing
1-3	Program Relocation
1-3	Program Location Counters
1-4	Assembly Options
2-1	SECTION II
	INSTRUCTION FORMAT
2-1	Statement Characteristics
2-4	Label Field
2-5	Opcode Field
2-6	Operand Field
2-16	Comments Field
3-1	SECTION III
	MACHINE INSTRUCTIONS
3-1	Memory Reference
3-4	Register Reference
3-8	Input/Output, Overflow, and Halt
3-11	Extended Arithmetic Unit
4-1	SECTION IV
	PSEUDO INSTRUCTIONS
4-1	Assembler Control
4-8	Object Program Linkage
4-11	Address and Symbol Definition
4-17	Constant Definition

CONTENTS

SECTION IV (cont.)

PSEUDO INSTRUCTIONS

- 4-23 Storage Allocation
- 4-23 Assembly Listing Control
- 4-26 Arithmetic Subroutine Calls

5-1 SECTION V

ASSEMBLER INPUT AND OUTPUT

- 5-1 Control Statement
- 5-2 Source Program
- 5-3 Binary Output
- 5-3 List Output

APPENDICES

- A-1 HP Character Set
- B-1 Summary of Instructions
- C-1 Alphabetical List of Instructions
- D-1 Sample Problems
- E-1 System Input/Output Subroutines
- F-1 Consolidated Coding Sheet

INDEX

INTRODUCTION

The Assembler and the Extended Assembler translate symbolic source language instructions into an object program for execution on the computer. The source language provides mnemonic machine operation codes, assembler directing pseudo codes, and symbolic addressing. The assembled program may be absolute or relocatable.

The source program may be assembled as a complete entity or it may be subdivided into several relocatable subprograms (or a main program and several subroutines), each of which may be assembled separately. The relocating loader loads the program and Links the subprograms as required. The Basic Binary Loader or Basic Binary Disc Loader loads absolute programs.

Input for the Assembler is prepared on paper tape or cards; the Assembler punches the binary program on paper tape in a format acceptable to the loader.

SECTION I

GENERAL DESCRIPTION

ASSEMBLY PROCESSING

The Assembler is a two pass system, or, if both punch and list output are requested, a three pass system on a minimum configuration. A pass is defined as a processing cycle of the source program input.

In the first pass, the Assembler creates a symbol table from the names used in the source statements. It also checks for certain possible error conditions and generates diagnostic messages if necessary.

During pass two, the Assembler again examines each statement in the source program along with the symbol table and produces the binary program and a program listing. Additional diagnostic messages may also be produced.

If only the output device is available and if both the binary output and the list output are requested, the listing function is deferred and performed as pass three.

When using the Assembler with a mass storage device the source program is written on the device during the first pass; the second pass of the source is read from the mass storage.

SYMBOLIC ADDRESSING

Symbols may be used for referring to machine instructions, data, constants, and certain other pseudo operations. A symbol represents the address for a computer word in memory. A symbol is defined when it is used as a label for a location in the program, a name of a common storage segment, the label of a data storage area or constant, the label of an absolute or relocatable value, or a location external to the program.

GENERAL DESCRIPTION

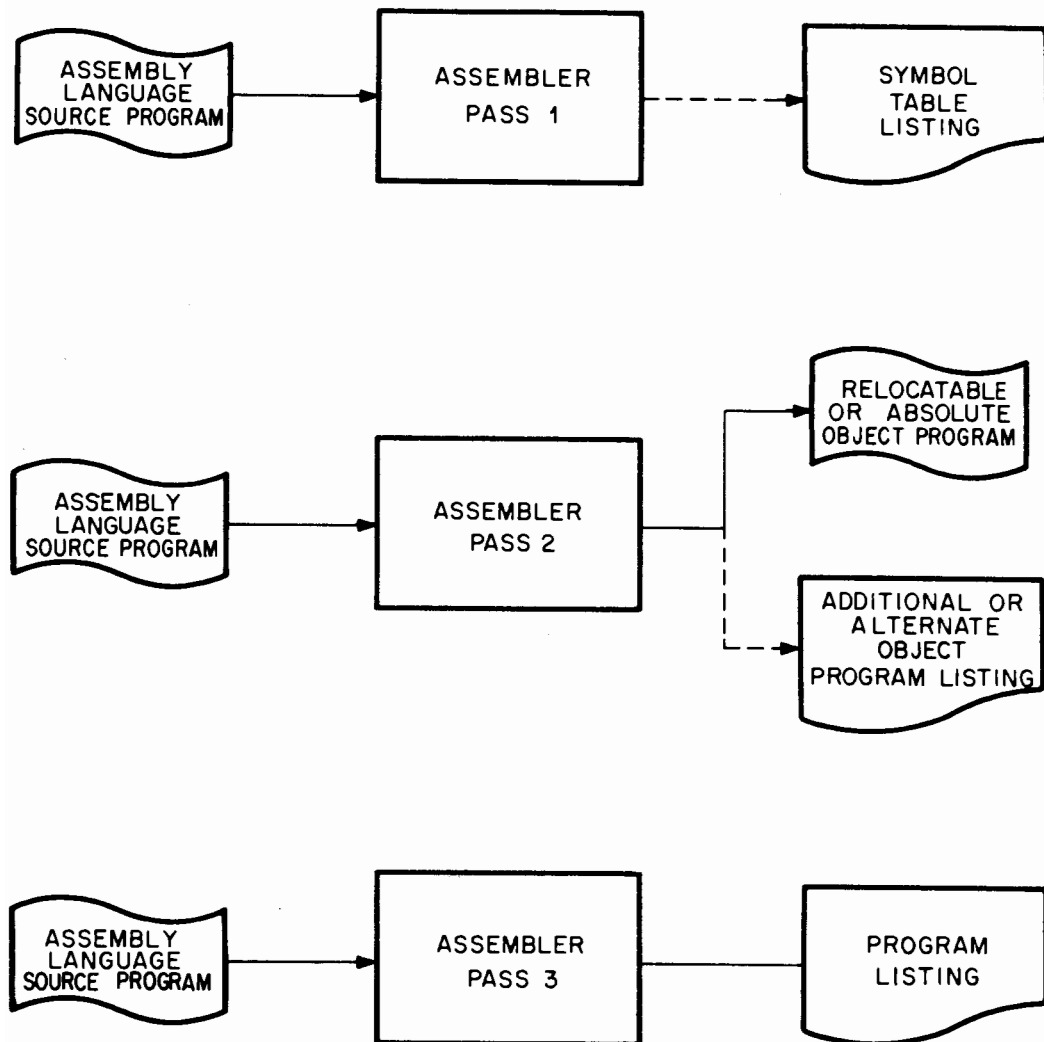


Figure 1-1. HP Assembler Processing

GENERAL DESCRIPTION

Through use of simple arithmetic operators, symbols may be combined with other symbols or numbers to form an expression which may identify a location other than that specifically named by a symbol. Symbols appearing in operands and expressions, but not specifically defined, and symbols that are defined more than once are considered to be in error by the Assembler.

PROGRAM RELOCATION

Relocatable programs may be relocated in core by the relocating loader; the location of the program origin and all subsequent instructions is determined at the time the program is loaded.

A relocatable program is assembled assuming a starting location of zero. All other instructions and data areas are assembled relative to this zero base. When the program is loaded, the relocatable operands are adjusted to correspond with the actual locations assigned by the loader.

The starting locations of the common storage area and the base page portion of the program are always established by the loader. References to the common area are common relocatable. References to the base page portion of the program are base page relocatable. If a program refers to the common area or makes use of the base page via the ORB pseudo instruction, the program must also be relocatable.

If a program is to be relocatable, all subprograms comprising the program must be relocatable; all memory reference operands must be relocatable expressions or literals, or have an absolute value of less than 100_8 .

PROGRAM LOCATION COUNTERS

The Assembler maintains a counter, called the program location counter, that assigns consecutive memory addresses to source statements.

GENERAL DESCRIPTION

The initial value of the program location counter is established according to the use of either the NAM or ORG pseudo operation at the start of the program. The NAM operation causes the program location counter to be set to zero for a relocatable program; the ORG operation specifies the absolute starting location for an absolute program.

Through use of the ORB pseudo operation a relocatable program may specify that certain operations or data areas be allocated to the base page. If so, a separate counter, called the base page location counter, is used in assigning these locations.

ASSEMBLY OPTIONS

Parameters specified with the first statement, the control statement, define the output to be produced by the Assembler:†

Absolute - The addresses generated by the Assembler are to be interpreted as absolute locations in memory. The program is a complete entity; external symbols, common storage references, and entry points are not permitted.

Relocatable - The program may be located anywhere in memory. All operands which refer to memory locations are adjusted as the program is loaded. Operands, other than those referring to the first 64 locations, must be relocatable expressions. Subprograms may contain external symbols and entry points, and may refer to common storage.

Binary output - An absolute or relocatable program is to be punched on paper tape.

List output - A program listing is produced either during pass two or pass three.

†See Section V for complete details.

GENERAL DESCRIPTION

Table print - List the symbol table at the end of the first pass.

Selective assembly - Sections of the program may be included or excluded at assembly time depending on the option used.

SECTION II

INSTRUCTION FORMAT

A source language statement consists of a label, an operation code, an operand, and comments. The label is used when needed as a reference by other statements. The operation code may be a mnemonic machine operation or an assembly directing pseudo code. An operand may be an expression consisting of an alphanumeric symbol, a number, a special character, or any of these combined by arithmetic operations. (For the Extended Assembler, an operand may also be a literal.) Indicators may be appended to the operand to specify certain functions such as indirect addressing. The comments portion of the statement is optional.

STATEMENT CHARACTERISTICS

The fields of the source statement appear in the following order:

Label
Opcode
Operand
Comments

Field Delimiters

One or more spaces separate the fields of a statement. An end-of-statement mark terminates the entire statement. On paper tape this mark is a return, (CR), and line feed, (LF).† A single space following the end-of-statement mark from the previous source statement is the null field indicator of the label field.

†A circled symbol (e.g., (CR)) represents an ASCII code or Teleprinter key.

HEWLETT-PACKARD ASSEMBLER CODING FORM

A0255

2-2

INSTRUCTION FORMAT

Character Set

The characters that may appear in a statement are as follows:

- A through Z
- 0 through 9
- . (period)
- * (asterisk)
- + (plus)
- (minus)
- , (comma)
- = (equals)
- () (parentheses)
- (space)

Any other ASCII characters may appear in the Remarks field. (See Appendix A.)

The letters A through Z, the numbers 0 through 9, and the period may be used in an alphanumeric symbol. In the first position in the Label field, an asterisk indicates a comment; in the Operand field, it represents the value of the program location counter for the current instruction. The plus and minus are used as operators in arithmetic address expressions. The comma separates several operation codes, or an expression and an indicator in the Operand field. An equals sign indicates a literal value. The parentheses are used only in the COM pseudo instruction.

Spaces separate fields of a statement. They may also be used to establish the format of the output list. Within a field they may be used freely when following +, -, ,, or (.

STATEMENT LENGTH

A statement may contain up to 80 characters including blanks, but excluding the end-of-statement mark. Fields beginning in characters 73 - 80 are not processed by the Assembler.

INSTRUCTION FORMAT

LABEL FIELD

The Label field identifies the statement and may be used as a reference point by other statements in the program.

The field starts in position one of the statement; the first position following an end-of-statement mark for the preceding statement. It is terminated by a space. A space in position one is the null field indicator for the label field; the statement is unlabeled.

Label Symbol

A label must be symbolic. It may have one to five characters consisting of A through Z, 0 through 9, and the period. The first character must be alphabetic or a period. A label of more than five characters could be entered on the source language tape, but the Assembler flags this condition as an error and truncates the label from the right to five characters.

Examples:

1	5	10	15	20	25	30	35	40	45	50
Label	Operation	Operand							Comments	
^†	LDA				NO LABEL					
.ABCD					VALID LABEL					
.1234					VALID LABEL					
A.123					VALID LABEL					
.					VALID LABEL					
1.AB					ILLEGAL LABEL - FIRST CHARACTER					
ABC123					NUMERIC.					
					ILLEGAL LABEL - TRUNCATED TO					
					ABC12.					
A*BC					ILLEGAL LABEL - ASTERISK NOT					
					ALLOWED IN LABEL.					
^ABC†					NO LABEL - THE ASSEMBLER ATTEMPTS					
					TO INTERPRET ABC AS AN OPERATION					
					CODE.					

†The caret symbol, ^ , indicates the presence of a space.

INSTRUCTION FORMAT

Each label must be unique within the program; two or more statements may not have the same symbolic name. Names which appear in the Operand field of an EXT or COM pseudo instruction may not also be used as statement labels in the same subprogram.

Examples:

1	5	10	15	20	25	30	35	40	45	50
		COM	ACOM(20),	BC(30)						
LB		EQU	160		VALID	LABEL				
		EXT	XL1, XL2							
START		LDA	LB		VALID	LABEL				
N25					VALID	LABEL				
XL2					ILLEGAL LABEL -	USED IN EXT.				
BC					ILLEGAL LABEL -	USED IN COM.				
N25					ILLEGAL LABEL -	PREVIOUSLY				
					DEFINED.					

Asterisk

An asterisk in position one indicates that the entire statement is a comment. Positions 2 through 80 are available; however, positions 1 through 68 only are printed as part of the assembly listing on the 2752A Teleprinter. An asterisk within the Label field is illegal in any position other than one.

OPCODE FIELD

The operation code defines an operation to be performed by the computer or the Assembler. The Opcode field follows the Label field and is separated from it by at least one space. If there is no label, the operation code may

INSTRUCTION FORMAT

begin anywhere after position one. The Opcode field is terminated by a space immediately following an operation code. Operation codes are organized in the following categories:

Machine operation codes

- Memory Reference
- Register Reference
- Input/Output, Overflow, and Halt
- Extended Arithmetic Unit

Pseudo operation codes

- Assembler control
- Object program linkage
- Address and symbol definition
- Constant definition
- Storage allocation
- Arithmetic subroutine calls
- Assembly Listing Control (Extended Assembler)

Operation codes are discussed in detail in Sections III and IV.

OPERAND FIELD

The meaning and format of the Operand field depend on the type of operation code used in the source statement. The field follows the Opcode field and is separated from it by at least one space. It is terminated by a space except when the space follows, + - (or, if there are no comments, by an end-of-statement mark.

The Operand field may contain an expression consisting of one of the following:

- Single symbolic term

- Single numeric term

- Asterisk

- Combination of symbolic terms, numeric terms, and the asterisk jointed by the arithmetic operators + and -.

INSTRUCTION FORMAT

An expression may be followed by a comma and an indicator.

Programs being assembled by the Extended Assembler may also contain a literal value in the Operand field.

Symbolic Terms

A symbolic term may be one to five characters consisting of A through Z, 0 through 9, and the period. The first character must be alphabetic or a period.

Examples:

1	5	10	15	20	25	30	35	40	45	50
Label		Operation		Operand		Comments				
		LDA	A1234		VALID	IF	DEFINED			
		ADA	B.1		VALID	IF	DEFINED			
		JMP	ENTRY		VALID	IF	DEFINED			
		STA	1ABC		ILLEGAL	OPERAND	FIRST	CHARACTER		
					NUMERIC.					
		STB	ABCDEF		ILLEGAL	OPERAND	MORE	THAN	FIVE	
					CHARACTERS.					

A symbol used in the Operand field must be a symbol that is defined elsewhere in the program in one of the following ways:

As a label in the Label field of a machine operation

As a label in the Label field of a BSS, ASC, DEC, DEX, OCT, DEF, ABS, EQU or REP pseudo operation

As a name in the Operand field of a COM or EXT pseudo operation

As a label in the Label field of an arithmetic subroutine pseudo operation

INSTRUCTION FORMAT

The value of a symbol is absolute or relocatable depending on the assembly option selected by the user. The Assembler assigns a value to a symbol as it appears in one of the above fields of a statement. If a program is to be loaded in absolute form, the values assigned by the assembler remain fixed. If the program is to be relocated, the actual value of a symbol is established on loading. A symbol may also be made absolute through use of the EQU pseudo instruction.

A symbolic term may be preceded by a plus or minus sign. If preceded by a plus or no sign, the symbol refers to its associated value. If preceded by a minus sign, the symbol refers to the two's complement of its associated value. A single negative symbolic operand may be used only with the ABS pseudo operation.

Numeric Terms

A numeric term may be decimal or octal. A decimal number is represented by one to five digits within the range 0 to 32767. An octal number is represented by one to six octal digits followed by the letter B; (0 to 177777B).

If a numeric term is preceded by a plus or no sign, the binary equivalent of the number is used in the object code. If preceded by a minus sign, the two's complement of the binary equivalent is used. A negative numeric operand may be used only with the DEX, DEC, OCT, and ABS pseudo operations.

In an absolute program, the maximum value of a numeric operand depends on the type of machine or pseudo instruction. In a relocatable program, the value of a numeric operand may not exceed 77B. Numeric operands are absolute. Their value is not altered by the assembler or the loader.

INSTRUCTION FORMAT

Asterisk

An asterisk in the Operand field refers to the value in the program location counter (or base page location counter) at the time the source program statement is encountered. The asterisk is considered a relocatable term in a relocatable program.

Expression Operators

The asterisk, symbols, and numbers may be joined by the arithmetic operators + and - to form arithmetic address expressions. The Assembler evaluates an expression and produces an absolute or relocatable value in the object code.

Examples:

1	Label	5	Operation	10	Operand	15	20	25	30	35	40	Comments	45	50
			LDA	SYM+6				ADD 6 TO THE VALUE OF SYM						
			ADA	SYM-3				SUBTRACT 3 FROM THE VALUE OF SYM						
			.											
			.											
			.											
			JMP	*+5				ADD 5 TO THE CONTENTS OF THE						
			.					PROGRAM LOCATION COUNTER.						
			.											
			.											
			STB	-A+C-4				ADD - VALUE OF A, THE VALUE OF C						
			.					AND SUBTRACT 4.						
			.											
			.											
			STA	XTA-*				SUBTRACT VALUE OF PROGRAM						
								LOCATION COUNTER FROM VALUE OF						
								XTA.						

INSTRUCTION FORMAT

Evaluation of Expressions

An expression consisting of a single operand has the value of that operand. An expression consisting of more than one operand is reduced to a single value. In expressions containing more than one operator, evaluation of the expression proceeds from left to right. The algebraic expression $A-(B-C+5)$ must be represented in the Operand field as $A-B+C-5$. Parentheses are not permitted in operand expressions for the grouping of operands.

The range of values that may result from an operand expression depends on the type of operation. The Assembler evaluates expressions as follows:†

Pseudo Operations	modulo $2^{15}-1$
Memory Reference	modulo $2^{10}-1$
Input/Output	$2^6 - 1$ (maximum value)

Expression Terms

The terms of an expression are the numbers and the symbols appearing in it. Decimal and octal integers, and symbols defined as being absolute in an EQU pseudo operation are absolute terms. The asterisk and all symbols that are defined in the program are relocatable or absolute depending on the type of assembly. Symbols that are defined as external may appear only as single term expressions.

Within a relocatable program, terms may be program relocatable, base page relocatable, or common relocatable. A symbol that names an area of common storage is a common relocatable term. A symbol that is allocated to the base page is a base page relocatable term. A symbol that is defined in any

†The evaluation of expressions by the Assembler is compatible with the addressing capability of the hardware instructions (e.g., up to 32K words through Indirect Addressing). The user must take care not to create addresses which exceed the memory size of the particular configuration.

INSTRUCTION FORMAT

other statement is a program relocatable term. Within one expression all relocatable terms must be base page relocatable, program relocatable, or common relocatable; the three types may not be mixed.

Absolute and Relocatable Expressions

An expression is absolute if its value is unaffected by program relocation. An expression is relocatable if its value changes according to the location into which the program is loaded. In an absolute program, all expressions are absolute. In a relocatable program, an expression may be base page relocatable, program relocatable, common relocatable, or absolute (if less than 100_8) depending on the definition of the terms composing it.

ABSOLUTE EXPRESSIONS

An absolute expression may be any arithmetic combination of absolute terms. It may also contain relocatable terms alone, or in combination with absolute terms. If relocatable terms do appear, there must be an even number of them; they must be of the same type; and they must be paired by sign (a negative term for each positive term). The paired terms do not have to be contiguous in the expression. The pairing of terms by type cancels the effect of relocation; the value represented by the pair remains constant.

An absolute expression reduces to a single absolute value. The value of an absolute multiterm expression may be negative only for ABS pseudo operations. A single numeric term also may be negative in an OCT, DEX, or DEC pseudo instruction. In a relocatable program the value of an absolute expression must be less than 100_8 for instructions that reference memory locations (Memory Reference, DEF, Arithmetic subroutine calls).

INSTRUCTION FORMAT

Examples:

If P_1 and P_2 are program relocatable terms; B_1 and B_2 , base page relocatable; C_1 and C_2 , common relocatable; and A , an absolute term; then the following are absolute terms:

$$A - C_1 + C_2$$

$$A + A$$

$$* - P_1$$

$$B_1 - *$$

$$A - P_1 + P_2$$

$$P_1 - P_2$$

$$B_1 - B_2 - A$$

$$-P_1 + P_2$$

$$C_1 - C_2 + A$$

$$B_1 - B_2$$

$$-C_1 + C_2 + A$$

$$-A - P_1 + P_2$$

The asterisk is base page relocatable or program relocatable depending on the location of the instruction.

RELOCATABLE EXPRESSIONS

A relocatable expression is one whose value is changed by the loader. All relocatable expressions must have a positive value.

A relocatable expression may contain any odd number of relocatable terms, alone, or in combination with absolute terms. All relocatable terms must be of the same type. Terms must be paired by sign with the odd term being positive.

A relocatable expression reduces to a single positive relocatable term, adjusted by the values represented by the absolute terms and paired relocatable terms associated with it.

Examples:

If P_1 , P_2 , and P_3 are program relocatable terms; B_1 , B_2 , and B_3 base page relocatable; C_1 , C_2 and C_3 , common relocatable; and A , an absolute term; then the following are relocatable terms:

INSTRUCTION FORMAT

$P_1 - A$	$C_1 - A$	$B_1 + A$
$P_1 - P_2 + P_3$	$C_1 - C_2 + C_3$	$C_1 + A$
$* + A$	$* - P_1 + P_2$	$* - A$
$A + B_1$	$A + C_1$	$-A - P_1 + P_2 + P_3$
$B_1 - B_2 + B_3 - A$	$C_1 - C_2 + C_3 - A$	$A + *$
$* + P_1 - *$	$P_1 - P_2 + *$	$-C_1 + C_2 + C_3$

Literals

Actual literal values may be specified as operands in relocatable programs to be assembled by the Extended Assembler. The Extended Assembler converts the literal to its binary value, assigns an address to it, and substitutes this address as the operand. Locations assigned to literals are those immediately following the last location used by the program.

A literal is specified by using an equal sign and a one-character identifier defining the type of literal. The actual literal value is specified immediately following this identifier; no spaces may intervene.

The identifiers are:

- =D a decimal integer, in the range -32767 to 32767, including zero.†
- =F a floating point number; any positive or negative real number in the range 10^{-38} to 10^{38} , including zero.†
- =B an octal integer, one to six digits, $b_1b_2b_3b_4b_5b_6$, where b_1 may be 0 or 1, and b_2-b_7 may be 0 to 7.†
- =A two ASCII characters.†
- =L an expression which, when evaluated, will result in an absolute value. All symbols appearing in the expression must be previously defined.

† See CONSTANT DEFINITION, Section 4.

INSTRUCTION FORMAT

If the same literal is used in more than one instruction, only one value is generated, and all instructions using this literal refer to the same location.

Literals may be specified only in the following memory reference instructions and pseudo instructions:

ADA	ADB	AND	MPY	} may use =D, =B, =A, =L
LDA	LDB	XOR	DIV	
CPA	CPB	IOR		
DLD	FAD	} may use =F		
FMP	FSB			
FDV				

Examples:

LDA	=D7980	A-Register is loaded with the binary equivalent of 7980 ₁₀ .
IOR	=B777	Inclusive OR is performed with contents of A-Register and 777 ₈ .
LDA	=ANO	A-Register is loaded with binary representation of ASCII characters NO.
LDB	=LZETZ-ZOOM+68	B-Register is loaded with the value resulting from the absolute expression.
FMP	=F39.75	Contents of A- and B-Registers multiplied by floating point constant 39.75.

Indirect Addressing

The HP computers provide an indirect addressing capability for memory reference instructions. The operand portion of an indirect instruction contains an address of another location rather than an actual operand. The

INSTRUCTION FORMAT

secondary location may be the operand or it may be indirect also and give yet another location, and so forth. The chaining ceases when a location is encountered that does not contain an indirect address. Indirect addressing provides a simplified method of address modifications as well as allowing access to any location in core.

The Assembler allows specification of indirect addressing by appending a comma and the letter I to any memory reference operand other than one referring to an external symbol. The actual operand of the instruction may be given in a DEF pseudo operation; this pseudo operation may also be used to indicate further levels of indirect addressing.

Examples:

1	Label	5	Operation	10	Operand	15	20	25	30	35	40	Comments	45	50
AB			LDA	SAM,I				EACH	TIME	THE	ISZ	IS	EXECUTED,	
AC			ADA	SAM,I				THE	EFFECTIVE	OPERAND	OF	AB	AND	
AD			ISZ	SAM				AC	CHANGE	ACCORDINGLY.				
			.											
			.											
			.											
SAM			DEF	ROGER										

A relocatable assembly language program, however, may be designed without concern for the pages in which it will be stored; indirect addressing is not required in the source language. When the program is being loaded, the loader provides indirect addressing whenever it detects an operand which does not fall in the current page or the base page. The loader substitutes a reference to the base page and then stores an indirect address in this referenced location. References to the same operand from other pages will be linked through the same location in the base page.

INSTRUCTION FORMAT

Base Page Addressing

The computer provides a capability which allows the memory reference instructions to address either the current page or the base page. The Assembler or the loader adjusts all instructions in which the operands refer to the base page; specific notation defining an operand as a base page reference is not required in the source program.

Clear Flag Indicator

The majority of the input/output instructions can alter the status of the input/output interrupt flag after execution or after the particular test is performed. In source language, this function is selected by appending a comma and a letter C to the Operand field.

Examples:

1	5	10	15	20	25	30	35	40	45	50
Label	Operation		Operand			Comments				
		STC	IO7,C		CLEAR	FLAG	IO7	AFTER	CONTROL	
					BIT	IS	SET			
		OTB	IO5,C		CLEAR	FLAG	IO5	AFTER	MOVE	

COMMENTS FIELD

The Comments field allows the user to transcribe notes on the program that will be listed with source language coding on the output produced by the Assembler. The field follows the Operand field and is separated from it by at least one space. The end-of-statement mark, (CR) (LF), or the 80th character in the entire statement terminates the field. If the listing is to be produced on the 2752A Teleprinter, the total statement length, excluding

INSTRUCTION FORMAT

the end-of-statement mark, should not exceed 52 characters, the width of the source language portion of the listing. Statements consisting solely of comments may contain up to 68 characters including the asterisk in the first position. On the list output, statements consisting entirely of comments begin in position 5 rather than 21 as with other source statements.

If there is no operand present, the Comments field should be omitted in the NAM and END pseudo operations and in the input/output statements, SOC, SOS, and HLT. If a comment is used, the Assembler attempts to interpret it as an operand.

SECTION III

MACHINE INSTRUCTIONS

The HP Assembler language machine instruction codes take the form of three-letter mnemonics. Each source statement corresponds to a machine operation in the object program produced by the Assembler.

Notation used in representing source language instruction is as follows:

label	Optional statement label
m	Memory location -- an expression
I	Indirect addressing indicator
sc	Select code -- an expression
C	Clear interrupt flag indicator
comments	Optional comments
[]	Brackets defining a field or portion of a field that is optional
{ }	Brackets indicating that one of the set may be selected.
lit	literal

MEMORY REFERENCE

Memory reference instructions perform arithmetic, logical and jump operations on the contents of the locations in core and the registers. An instruction may directly address the 2048 words of the current and base pages. If required, indirect addressing may be utilized to refer to all 32,768 words of memory. Expressions in the Operand field are evaluated modulo 2^{10} .

If the program is to be assembled in relocatable form, the Operand field may contain relocatable expressions or absolute expressions which are less than 100_8 in value. If the program is to be absolute, the operands may be any expressions consistent with the location of the program. Literals may not be used in an absolute program. Absolute programs must be complete entities; they may not refer to external subroutines or common storage.

MACHINE INSTRUCTIONS

Jump and Increment-Skip

Jump and Increment-Skip instructions may alter the normal sequence of program execution.

label	JMP	m [,I]	comments
-------	-----	--------	----------

Jump to m. Jump indirect inhibits interrupt until the transfer of control is complete.

label	JSB	m [,I]	comments
-------	-----	--------	----------

Jump to subroutine. The address for label+1 is placed into the location represented by m and control transfers to m+1. On completion of the subroutine, control may be returned to the normal sequence by performing a JMP m, I.

label	ISZ	m [,I]	comments
-------	-----	--------	----------

Increment, then skip if zero. ISZ adds 1 to the contents of m. If m then equals zero, the next instruction in memory is bypassed.

Add, Load and Store

Add, Load, and Store instructions transmit and alter the contents of memory and of the A- and B-Registers. A literal, indicated by "lit", may be either =D, =B, =A, or =I type.

label	ADA	$\begin{Bmatrix} m [,I] \\ lit \end{Bmatrix}$	comments
-------	-----	---	----------

Add the contents of m to A.

MACHINE INSTRUCTIONS

label	ADB	$\begin{matrix} m \text{ } [,I] \\ \{ \text{lit} \} \end{matrix}$	comments
-------	-----	---	----------

Add the contents of m to B.

label	LDA	$\begin{matrix} m \text{ } [,I] \\ \{ \text{lit} \} \end{matrix}$	comments
-------	-----	---	----------

Load A from m.

label	LDB	$\begin{matrix} m \text{ } [,I] \\ \{ \text{lit} \} \end{matrix}$	comments
-------	-----	---	----------

Load B from m.

label	STA	$m \text{ } [,I]$	comments
-------	-----	-------------------	----------

Store contents of A in m.

label	STB	$m \text{ } [,I]$	comments
-------	-----	-------------------	----------

Store contents of B in m.

In each instruction, the contents of the sending location is unchanged after execution.

Logical Operations

The logical instructions allow bit manipulation and the comparison of two computer words.

label	AND	$\begin{matrix} m \text{ } [,I] \\ \{ \text{lit} \} \end{matrix}$	comments
-------	-----	---	----------

The logical product of the contents of m and the contents of A are placed in A.

MACHINE INSTRUCTIONS

label	XOR	$\begin{matrix} m & [,I] \\ \{ & \text{lit} \} \end{matrix}$	comments
-------	-----	--	----------

The modulo-two sum (exclusive "or") of the bits in m and the bits in A is placed in A.

label	IOR	$\begin{matrix} m & [,I] \\ \{ & \text{lit} \} \end{matrix}$	comments
-------	-----	--	----------

The logical sum (inclusive "or") of the bits in m and the bits in A is placed in A.

label	CPA	$\begin{matrix} m & [,I] \\ \{ & \text{lit} \} \end{matrix}$	comments
-------	-----	--	----------

Compare the contents of m with the contents of A. If they differ, skip the next instruction; otherwise, continue.

label	CPB	$\begin{matrix} m & [,I] \\ \{ & \text{lit} \} \end{matrix}$	comments
-------	-----	--	----------

Compare the contents of m with the contents of B. If they differ, skip the next instruction; otherwise, continue.

REGISTER REFERENCE

The register reference instructions include a shift-rotate group, an alter-skip group, and NOP (no-operation). With the exception of NOP, they have the capability of causing several actions to take place during one memory cycle. Multiple operations within a statement are separated by a comma.

MACHINE INSTRUCTIONS

Shift-Rotate Group

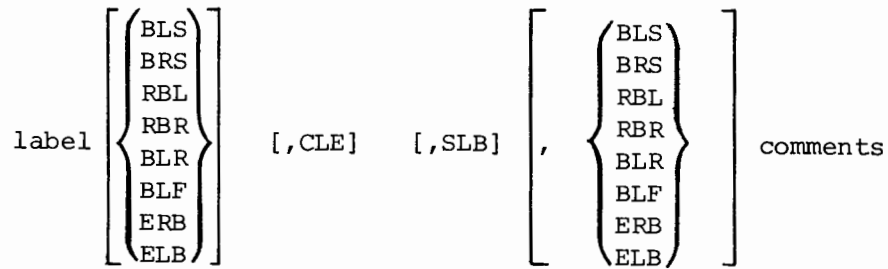
This group contains 19 basic instructions that can be combined to produce more than 500 different single cycle operations.

CLE	Clear E to zero
ALS	Shift A left one bit, zero to least significant bit. Sign unaltered
BLS	Shift B left one bit, zero to least significant bit. Sign unaltered
ARS	Shift A right one bit, extend sign; sign unaltered.
BRS	Shift B right one bit, extend sign; sign unaltered.
RAL	Rotate A left one bit
RBL	Rotate B left one bit
RAR	Rotate A right one bit
RBR	Rotate B right one bit
ALR	Shift A left one bit, clear sign, zero to least significant bit
BLR	Shift B left one bit, clear sign, zero to least significant bit
ERA	Rotate E and A right one bit
ERB	Rotate E and B right one bit
ELA	Rotate E and A left one bit
ELB	Rotate E and B left one bit
ALF	Rotate A left four bits
BLF	Rotate B left four bits
SLA	Skip next instruction if least significant bit in A is zero
SLB	Skip next instruction if least significant bit in B is zero

These instructions may be combined as follows:

$$\text{label} \left[\begin{array}{c} \left(\begin{array}{c} \text{ALS} \\ \text{ARS} \\ \text{RAL} \\ \text{RAR} \\ \text{ALR} \\ \text{ALF} \\ \text{ERA} \\ \text{ELA} \end{array} \right) \end{array} \right] [\text{,CLE}] [\text{,SLA}] \left[\begin{array}{c} \left(\begin{array}{c} \text{ALS} \\ \text{ARS} \\ \text{RAL} \\ \text{RAR} \\ \text{ALR} \\ \text{ALF} \\ \text{ERA} \\ \text{ELA} \end{array} \right) \end{array} \right] \text{comments}$$

MACHINE INSTRUCTIONS



CLE, SLA, or SLB appearing alone or in any valid combination with each other are assumed to be a shift-rotate machine instruction.

The shift-rotate instructions must be given in the order shown. At least one and up to four are included in one statement. Instructions referring to the A-register may not be combined in the same statement with those referring to the B-register.

No-Operation Instruction

When a no-operation is encountered in a program, no action takes place; the computer goes on to the next instruction. A full memory cycle is used in executing a no-operation instruction.



A subroutine to be entered by a JSB instruction should have a NOP as the first statement. The return address can be stored in the location occupied by the NOP during execution of the program. A NOP statement causes the Assembler to generate a word of zeros.

Alter-Skip Group

The alter-skip group contains 19 basic instructions that can be combined to produce more than 700 different single cycle operations.

CLA	Clear the A-Register
CLB	Clear the B-Register

MACHINE INSTRUCTIONS

CMA	Complement the A-Register
CMB	Complement the B-Register
CCA	Clear, then complement the A-Register (set to ones)
CCB	Clear, then complement the B-Register (set to ones)
CLE	Clear the E-Register
CME	Complement the E-Register
CCE	Clear, then complement the E-Register
SEZ	Skip next instruction if E is zero
SSA	Skip if sign of A is positive (0)
SSB	Skip if sign of B is positive (0)
INA	Increment A by one
INB	Increment B by one
SZA	Skip if contents of A equals zero
SZB	Skip if contents of B equals zero
SLA	Skip if least significant bit of A is zero
SLB	Skip if least significant bit of B is zero
RSS	Reverse the sense of the skip instructions. If no skip instructions precede in the statement, skip the next instruction

These instructions may be combined as follows:

label	$\left[\begin{Bmatrix} \text{CLA} \\ \text{CMA} \\ \text{CCA} \end{Bmatrix} \right]$	[,SEZ]	$\left[\begin{Bmatrix} \text{CLE} \\ \text{CME} \\ \text{CCE} \end{Bmatrix} \right]$	[,SSA] [,SLA] [,INA] [,SZA] [,RSS]	comments
label	$\left[\begin{Bmatrix} \text{CLB} \\ \text{CMB} \\ \text{CCB} \end{Bmatrix} \right]$	[,SEZ]	$\left[\begin{Bmatrix} \text{CLE} \\ \text{CME} \\ \text{CCE} \end{Bmatrix} \right]$	[,SSB] [,SLB] [,INB] [,SZB] [,RSS]	comments

The alter-skip instructions must be given in the order shown. At least one and up to eight are included in one statement. Instructions referring to the A-register may not be combined in the same statement with those referring to the B-register. When two or more skip functions are combined in a single operation, a skip occurs if any one of the conditions exists. If a word with RSS also includes both SSA and SLA (or SSB and SLB), a skip occurs only when sign and least significant bit are both set (1).

MACHINE INSTRUCTIONS

INPUT/OUTPUT, OVERFLOW, AND HALT

The input/output instructions allow the user to transfer data to and from an external device via a buffer, to enable or disable external interrupt, or to check the status of I/O devices and operations. A subset of these instructions permits checking for an arithmetic overflow condition.

Input/output instructions require the designation of a select code, *sc*, which indicates one of 64 input/output channels or functions. Each channel consists of a connect/disconnect control bit, a flag bit, and a buffer of up to 16 bits. The setting of the control bit indicates that a device associated with the channel is operable. The flag bit is set automatically when transmission between the device and the buffer is completed. Instructions are also available to test or clear the flag bit for the particular channel. If the interrupt system is enabled, setting of the flag causes program interrupt to occur; control transfers to the interrupt location related to the channel.

Expressions used to represent select codes (channel numbers) must have a value of less than 2^6 . The value specifies the device or operation referenced. Instructions which transfer data between the A or B register and a buffer, access the Switch register when *sc* = 1. The character C appended to such an instruction clears the overflow bit after the transfer from the switch register is complete.

Input/Output

Prior to any input/output data transmission, the control bit is set. The instruction which enables the device may also transfer data between the device and the buffer.

label	STC	sc [,C]	comments
-------	-----	---------	----------

Set I/O control bit for channel specified by *sc*. STC transfers or enables transfer of an element of data from an input device to the buffer or to an

MACHINE INSTRUCTIONS

output device from the buffer. The exact function of the STC depends on the device; for the 2752A Teleprinter, an STC enables transfer of a series of bits. If $sc = 1$, this statement is treated as NOP. The C option clears the flag bit for the channel.

label	CLC	sc [,C]	comments
-------	-----	---------	----------

Clear I/O control bit for channel specified by sc . When the control bit is cleared, interrupt on the channel is disabled, although the flag may still be set by the device. If $sc = 0$, control bits for all channels are cleared to zero; all devices are disconnected. If $sc = 1$, this statement is treated as NOP.

label	LIA	sc [,C]	comments
-------	-----	---------	----------

Load into A the contents of the I/O buffer indicated by sc .

label	LIB	sc [,C]	comments
-------	-----	---------	----------

Load into B the contents of the I/O buffer indicated by sc .

label	MIA	sc [,C]	comments
-------	-----	---------	----------

Merge (inclusive "or") the contents of the I/O buffer indicated by sc into A.

label	MIB	sc [,C]	comments
-------	-----	---------	----------

Merge (inclusive "or") the contents of the I/O buffer indicated by sc into B.

label	OTA	sc [,C]	comments
-------	-----	---------	----------

Output the contents of A to the I/O buffer indicated by sc .

MACHINE INSTRUCTIONS

label	OTB	sc[,C]	comments
-------	-----	--------	----------

Output the contents of B to the I/O buffer indicated by sc.

label	STF	sc	comments
-------	-----	----	----------

Sets the flag bit of the channel indicated by sc. If sc = 0, STF enables the interrupt system. A sc code of 1 causes the overflow bit to be set.

label	CLF	sc	comments
-------	-----	----	----------

Clear the flag bit to zero for the channel indicated by sc. If sc = 0, CLF disables the interrupt system. If sc = 1, the overflow bit is cleared to zero.

label	SFC	sc	comments
-------	-----	----	----------

Skip the next instruction if the flag bit for channel sc is clear. If sc = 1, the overflow bit is tested.

label	SFS	sc	comments
-------	-----	----	----------

Skip the next instruction if the flag bit for channel sc is set. If sc = 1, the overflow is tested.

Overflow

In addition to the use of a select code of 1, the overflow bit may be accessed by the following instructions:

label	CLO	comments
-------	-----	----------

Clear the overflow bit.

MACHINE INSTRUCTIONS

label	STO	comments
-------	-----	----------

Set overflow bit.

label	SOC	[C]	comments
-------	-----	-----	----------

Skip the next instruction if the overflow bit is clear. The C option clears the bit after the test is performed.

label	SOS	[C]	comments
-------	-----	-----	----------

Skip the next instruction if the overflow bit is set. The C option clears the bit after the test is performed.

The C option is identified by the sequence "space C space" following either "SOC" or "SOS". Anything else is treated as a comment.

Halt

label	HLT	{ [sc [,C]] [c] }	comments
-------	-----	-----------------------	----------

Halt the computer. The machine instruction word is displayed in the T-Register. If the C option is used, the flag bit associated with channel sc is cleared.

If neither the select code nor the C option is used, the comments portion must be omitted.

EXTENDED ARITHMETIC UNIT

Ten instructions may be used with the EAU version of the Assembler or Extended Assembler to increase the computer's overall efficiency. The computer must include the Extended Arithmetic Unit option to obtain the resulting increase in available core storage and decrease in program run time.

MACHINE INSTRUCTIONS

label	MPY	$\{m[,I] \}$ lit	comments
-------	-----	----------------------	----------

The MPY instruction multiplies the contents of the A-Register by the contents of m. The product is stored in registers B and A. B contains the sign of the product and the 15 most significant bits; A contains the least significant bits.

label	DIV	$\{m[,I] \}$ lit	comments
-------	-----	----------------------	----------

The DIV instruction divides the contents of registers B and A by the contents of m. The quotient is stored in A and the remainder in B. Initially B contains the sign and the 15 most significant bits of the dividend; A contains the least significant bits.

label	DLD	$\{m[,I] \}$ lit	comments
-------	-----	----------------------	----------

The DLD instruction loads the contents of locations m and m + 1 into registers A and B, respectively.

label	DST	m[,I]	comments
-------	-----	--------	----------

The DST instruction stores the contents of registers A and B in locations m and m + 1, respectively.

MPY, DIV, DLD, DST results in two machine words: a word for the instruction code and one for the operand.

The above four instructions are available without the Extended Arithmetic Unit option as software subroutines.[†] As a part of the Extended Arithmetic option, they require less core storage and can be executed in less time.

[†] See ARITHMETIC SUBROUTINE CALLS, Section 4.

MACHINE INSTRUCTIONS

The following seven instructions can be used only on machines with the Extended Arithmetic Unit. These shift-rotate instructions provide the capability to shift or rotate the B- and A-Registers n number of bit positions, where $1 \leq n \leq 16$.

label	ASR	n	comments
-------	-----	---	----------

The ASR instruction arithmetically shifts the B- and A-Registers right n bits. The sign bit (bit 15 of B) is extended.

label	ASL	n	comments
-------	-----	---	----------

The ASL instruction arithmetically shifts the B- and A-Register left n bits. Zeroes are placed in the least significant bits. The sign bit (bit 15 of B) is unaltered. The overflow bit is set if bit 14 differs from bit 15 before each shift; otherwise, exit with overflow bit cleared.

label	RRR	n	comments
-------	-----	---	----------

The RRR instruction rotates the B- and A-Registers right n bits.

label	RRL	n	comments
-------	-----	---	----------

The RRL instruction rotates the B- and A-Registers left n bits.

label	LSR	n	comments
-------	-----	---	----------

The LSR instruction logically shifts the B- and A-Registers right n bits. Zeroes are placed in the most significant bits.

label	LSL	n	comments
-------	-----	---	----------

The LSL instruction logically shifts the B- and A-Registers left n bits. Place zeroes into the least significant bits.

MACHINE INSTRUCTIONS



Exchange the contents of the A- and B-Registers. The contents of the A-Register are shifted into the B-Register and the contents of the B-Register are shifted into the A-Register.

SECTION IV

PSEUDO INSTRUCTIONS

The pseudo instructions control the Assembler, establish program relocatability, and define program linkage as well as specify various types of constants, blocks of memory, and labels used in the program. With the Extended Assembler, pseudo instructions also control listing output.

ASSEMBLER CONTROL

The Assembler control pseudo instructions establish and alter the contents of the base page and program location counters, and terminate assembly processing. Labels may be used but they are ignored by the Assembler. NAM records produced by the Assemblers are accepted by the DOS, DOS-M and BCS loaders.

	NAM	[name]	comments
--	-----	---------	----------

NAM defines the name of a relocatable program. A relocatable program must begin with a NAM statement.[†] A relocatable program is assembled assuming a starting location of zero (i.e., zero relative). The name may be a symbol of one to five alphanumeric characters the first of which must be alphabetic or a period. The program name is printed on the list output. The name is optional and if omitted, the comments must be omitted also.

	ORG	m	comments
--	-----	---	----------

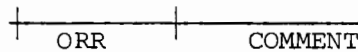
The ORG statement defines the origin of an absolute program, or the origin of subsequent sections of absolute or relocatable programs.

[†] The Control Statement, the HED instruction, and comments may appear prior to the NAM or ORG statements. If the Control Statement (ASMB,...) does not appear on tape preceding the program, it must be entered from the teleprinter.

PSEUDO INSTRUCTIONS

An absolute program must begin with an ORG statement.[†] The operand m, must be a decimal or octal integer specifying the initial setting of the program location counter.

ORG statements may be used elsewhere in the program to define starting addresses for portions of the object code. For absolute programs the Operand field, m, may be any expression. For relocatable programs, m, must be a program relocatable expression; it may not be base page or common relocatable or absolute. An expression is evaluated modulo 2^{15} . Symbols must be previously defined. All instructions following an ORG are assembled at consecutive addresses starting with the value of the operand.



ORR resets the program location counter to the value existing when an ORG or ORB instruction was encountered.

Example:

1	5	10	15	20	25	30	35	40	45	50
		NAM	RSET			SET	PLC	TO	VALUE	OF
FIRST	ADA					RSET	AS	NAME	OF	PROGRAM.
	.									
	.									
	.									
	ADA	CTRL				ASSUME	PLC	AT	FIRST+2280.	
	ORG	FIRST+2926				SAVE	PLC	VALUE	OF	FIRST+2280
	.					AND	SET	PLC	TO	FIRST+2926.
	.									
	.									
	JMP	EVEN+1				ASSUME	PLC	AT	FIRST+3004	
	ORR					RESET	PLC	TO	FIRST+2280.	

[†] The Control Statement, the HED instruction, and comments may appear prior to the NAM or ORG statements. If the Control Statement (ASMB,...) does not appear on tape preceding the program, it must be entered from the teleprinter.

PSEUDO INSTRUCTIONS

More than one ORG or ORB statement may occur before an ORR is used. If so, when the ORR is encountered, the program location counter is reset to the value it contained when the first ORG or ORB of the string occurred.

Example:

1	5	10	15	20	25	30	35	40	45	50
		NAM	RSET		SET	PLC	TO	ZERO		
FIRST	ADA									
	.									
	.									
	.									
	LDA	WYZ		ASSUME	PLC	AT	FIRST+2250			
	ORG	FIRST+2500		SET	PLC	TO	FIRST+2500			
	.									
	.									
	.									
	LDB	ERA		ASSUME	PLC	AT	FIRST+2750			
	ORG	FIRST+2900		SET	PLC	TO	FIRST+2900			
	.									
	.									
	.									
	CLE			ASSUME	PLC	AT	FIRST+2920			
	ORR			RESET	PLC	TO	FIRST+2250			

If a second ORR appears before an intervening ORG or ORB, the second ORR is ignored.

ORR cannot be used to reset the location counter for locations in the base page that are governed by the ORB statement.

	ORB	comments
--	-----	----------

ORB defines the portion of a relocatable program that must be assigned to the base page by the Assembler. The Label field (if given) is ignored, and the statement requires no operand. All statements that follow the ORB statement are assigned contiguous locations in the base page. Assignment to the base page terminates when the Assembler detects an ORG, ORR, or END statement.

PSEUDO INSTRUCTIONS

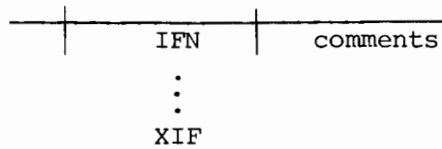
When more than one ORB is used in a program, each ORB causes the Assembler to resume assigning base page locations at the address following the last assigned base page location.

An ORB statement in an absolute program has no significance and is flagged as an error.

Example:

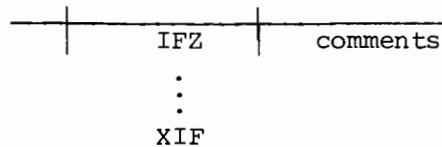
1	Label	5	Operation	10	Operand	15	20	25	30	35	40	Comments	45	50
			NAM	PROG				ASSIGN ZERO AS RELATIVE STARTING						
			.					LOCATION FOR PROGRAM PROG.						
			.											
			.											
			ORB					ASSIGN ALL FOLLOWING STATEMENTS						
								TO BASE PAGE.						
IAREA		BSS	100											
			.											
			.											
			ORB					RESUME ASSIGNMENT AT NEXT						
			.					AVAILABLE LOCATION IN BASE PAGE.						
			.											
			.											
			ORB					RESUME ASSIGNMENT AT NEXT						
			.					AVAILABLE LOCATION IN BASE PAGE.						
			.											
			ORB					RESUME ASSIGNMENT AT NEXT						
			.					AVAILABLE LOCATION IN BASE PAGE.						
			.											
			ORB					RESUME ASSIGNMENT AT NEXT						
			.					AVAILABLE LOCATION IN BASE PAGE.						
			.											
			ORB					RESUME ASSIGNMENT AT NEXT						
			.					AVAILABLE LOCATION IN BASE PAGE.						
			.											
			ORB					RESUME ASSIGNMENT AT NEXT						
			.					AVAILABLE LOCATION IN BASE PAGE.						
			.											
			ORB					RESUME ASSIGNMENT AT NEXT						
			.					AVAILABLE LOCATION IN BASE PAGE.						
			.											
			ORB					RESUME ASSIGNMENT AT NEXT						
			.					AVAILABLE LOCATION IN BASE PAGE.						
			.											
			ORB					RESUME ASSIGNMENT AT NEXT						
			.					AVAILABLE LOCATION IN BASE PAGE.						
			.											
			ORB					RESUME ASSIGNMENT AT NEXT						
			.					AVAILABLE LOCATION IN BASE PAGE.						
			.											
			ORB					RESUME ASSIGNMENT AT NEXT						
			.					AVAILABLE LOCATION IN BASE PAGE.						
			.											
			ORB					RESUME ASSIGNMENT AT NEXT						
			.					AVAILABLE LOCATION IN BASE PAGE.						
			.											
			ORB					RESUME ASSIGNMENT AT NEXT						
			.					AVAILABLE LOCATION IN BASE PAGE.						
			.											
			ORB					RESUME ASSIGNMENT AT NEXT						
			.					AVAILABLE LOCATION IN BASE PAGE.						
			.											
			ORB					RESUME ASSIGNMENT AT NEXT						
			.					AVAILABLE LOCATION IN BASE PAGE.						
			.											
			ORB					RESUME ASSIGNMENT AT NEXT						
			.					AVAILABLE LOCATION IN BASE PAGE.						
			.											
			ORB					RESUME ASSIGNMENT AT NEXT						
			.					AVAILABLE LOCATION IN BASE PAGE.						
			.											
			ORB					RESUME ASSIGNMENT AT NEXT						
			.					AVAILABLE LOCATION IN BASE PAGE.						
			.											
			ORB					RESUME ASSIGNMENT AT NEXT						
			.					AVAILABLE LOCATION IN BASE PAGE.						
			.											
			ORB					RESUME ASSIGNMENT AT NEXT						
			.					AVAILABLE LOCATION IN BASE PAGE.						
			.											
			ORB					RESUME ASSIGNMENT AT NEXT						
			.					AVAILABLE LOCATION IN BASE PAGE.						
			.											
			ORB					RESUME ASSIGNMENT AT NEXT						
			.					AVAILABLE LOCATION IN BASE PAGE.						
			.											
			ORB					RESUME ASSIGNMENT AT NEXT						
			.					AVAILABLE LOCATION IN BASE PAGE.						
			.											
			ORB					RESUME ASSIGNMENT AT NEXT						
			.					AVAILABLE LOCATION IN BASE PAGE.						
			.											
			ORB					RESUME ASSIGNMENT AT NEXT						
			.					AVAILABLE LOCATION IN BASE PAGE.						
			.											
			ORB					RESUME ASSIGNMENT AT NEXT						
			.					AVAILABLE LOCATION IN BASE PAGE.						
			.											
			ORB					RESUME ASSIGNMENT AT NEXT						
			.					AVAILABLE LOCATION IN BASE PAGE.						
			.											
			ORB					RESUME ASSIGNMENT AT NEXT						
			.					AVAILABLE LOCATION IN BASE PAGE.						
			.											
			ORB					RESUME ASSIGNMENT AT NEXT						
			.					AVAILABLE LOCATION IN BASE PAGE.						
			.											
			ORB					RESUME ASSIGNMENT AT NEXT						
			.					AVAILABLE LOCATION IN BASE PAGE.						
			.											
			ORB					RESUME ASSIGNMENT AT NEXT						
			.					AVAILABLE LOCATION IN BASE PAGE.						
			.											
			ORB					RESUME ASSIGNMENT AT NEXT						
			.					AVAILABLE LOCATION IN BASE PAGE.						
			.											
			ORB					RESUME ASSIGNMENT AT NEXT						
			.					AVAILABLE LOCATION IN BASE PAGE.						
			.											
			ORB					RESUME ASSIGNMENT AT NEXT						
			.					AVAILABLE LOCATION IN BASE PAGE.						
			.											
			ORB					RESUME ASSIGNMENT AT NEXT						
			.					AVAILABLE LOCATION IN BASE PAGE.						
			.											
			ORB					RESUME ASSIGNMENT AT NEXT						
			.					AVAILABLE LOCATION IN BASE PAGE.						
			.											
			ORB					RESUME ASSIGNMENT AT NEXT						
			.					AVAILABLE LOCATION IN BASE PAGE.						
			.											
			ORB					RESUME ASSIGNMENT AT NEXT						
			.					AVAILABLE LOCATION IN BASE PAGE.						
			.											
			ORB					RESUME ASSIGNMENT AT NEXT						
			.					AVAILABLE LOCATION IN BASE PAGE.						
			.											
			ORB					RESUME ASSIGNMENT AT NEXT						
			.					AVAILABLE LOCATION IN BASE PAGE.						
			.											
			ORB					RESUME ASSIGNMENT AT NEXT						
			.					AVAILABLE LOCATION IN BASE PAGE.						
			.											
			ORB					RESUME ASSIGNMENT AT NEXT						
			.					AVAILABLE LOCATION IN BASE PAGE.						
			.											
			ORB					RESUME ASSIGNMENT AT NEXT						
			.					AVAILABLE LOCATION IN BASE PAGE.						
			.											
			ORB					RESUME ASSIGNMENT AT NEXT						
			.					AVAILABLE LOCATION IN BASE PAGE.						
			.											
			ORB					RESUME ASSIGNMENT AT NEXT						
			.					AVAILABLE LOCATION IN BASE PAGE.						
			.											
			ORB					RESUME ASSIGNMENT AT NEXT						
			.					AVAILABLE LOCATION IN BASE PAGE.						
			.											
			ORB					RESUME ASSIGNMENT AT NEXT						
			.					AVAILABLE LOCATION IN BASE PAGE.						
			.											
			ORB					RESUME ASSIGNMENT AT NEXT						
			.					AVAILABLE LOCATION IN BASE PAGE.						
			.											
			ORB					RESUME ASSIGNMENT AT NEXT						
			.					AVAILABLE LOCATION IN BASE PAGE.						
			.											
			ORB					RESUME ASSIGNMENT AT NEXT						
			.					AVAILABLE LOCATION IN BASE PAGE.						
			.											
			ORB					RESUME ASSIGNMENT AT NEXT						
			.					AVAILABLE LOCATION IN BASE PAGE.						
			.											
			ORB					RESUME ASSIGNMENT AT NEXT						
			.					AVAILABLE LOCATION IN BASE PAGE.						
			.											
			ORB					RESUME ASSIGNMENT AT NEXT						
			.					AVAILABLE LOCATION IN BASE PAGE.						
			.											
			ORB					RESUME ASSIGNMENT AT NEXT</						

PSEUDO INSTRUCTIONS



All source language statements appearing between the IFN and the XIF pseudo instructions are included in the program if the character "N" is specified on the ASMB control statement.

All source language statements appearing between the IFZ and the XIF pseudo instructions are included in the program if the character "Z" is specified on the ASMB control statement.



When the particular letter is not included on the control statement, the related set of statements appears on the Assembler output listing but is not assembled.

Any number of IFN-XIF and IFZ-XIF sets may appear in a program, however, they may not overlap. An IFZ or IFN intervening between an IFZ or IFN and the XIF terminator results in a diagnostic being issued during compilation; the second pseudo instruction is ignored.

Both IFN-XIF and IFZ-XIF pseudo instructions may be used in the program; however, only one type will be selected in a single assembly. Therefore, if both characters "N" and "Z" appear in the control statement, the character which is listed last will determine the set of coding that is to be included in the program.

PSEUDO INSTRUCTIONS

Example:

1	5	10	15	20	25	30	35	40	45	50
Label	Operation	Operand							Comments	
	NAM	TRAVL								
	.									
	.									
	.									
	IFZ									
	LDA	CAR								
	CMA	SZA								
	JMP	NO. GO								
	LDA	MILES								
	DIV	SPEED								
	STA	GAS								
	XIF									
	.									
	.									
	.									
	IFN									
	LDA	PLANE								
	CMA	SZA								
	JMP	NO. GO								
	LDA	TIME								
	CPA	COST								
	XIF									
NO. GO	HLT	77								
	.									
	.									
	.									
	END									

Program TRAVL will perform computations involving either or neither CAR or PLANE considerations depending on the presence or absence of Z or N parameters in the Control Statement.

PSEUDO INSTRUCTIONS

Example:

1	Label	5	Operation	10	Operand	15	20	25	30	35	40	Comments	45	50
			NAM	WAGE										
			.											
			.											
			.											
			JSB	HOUR										
			MPY	TIME1										
			IFZ											
			JSB	OVTIM										
			MPY	TIME2										
			.											
			.											
			.											
	TIME1		DEC	40										
	TIME2		BSS	1										
			END											

Program WAGES computes a weekly wage value. Overtime consideration will be included in the program if "Z" is included in the parameters of the Control Statement.

The REP pseudo instruction, available in the Extended Assembler only, causes the repetition of the statement immediately following it a specified number of times.

label	REP	n	comments
-------	-----	---	----------

The statement following the REP in the source program is repeated n times. The n may be any absolute expression. Comment lines (indicated by an asterisk in character position 1) are not repeated by REP. If a comment follows a REP instruction, the comment is ignored and the instruction following the comment is repeated.

A label specified in the REP pseudo instruction is assigned to the first repetition of the statement. A label cannot be part of the instruction to be repeated; it would result in a doubly defined symbol error.

PSEUDO INSTRUCTIONS

Example:

```
          CLA
TRIPL    REP      3
          ADA      DATA
```

The above source code would generate the following:

```
          CLA                Clear the A-Register; the content of DATA is
TRIPL    ADA      DATA      tripled and stored in the A-Register.
          ADA      DATA
          ADA      DATA
```

Example:

```
FILL     REP      100B
          NOP
```

The example above loads 100₈ memory locations with the NOP instruction. The first location is labeled FILL.

Example:

```
          REP 2
          MPY DATA
```

The above source code would generate the following:

```
          MPY DATA
          MPY DATA
```

	END	[m]	comments

This statement terminates the program; it marks the physical end of the source language statements. The Operand field, m, may contain a name appearing as a statement label in the current program or it may be blank. If a name is entered, it identifies the location to which the loader transfers control after a relocatable program is loaded. A NOP should be stored at that location; the loader transfers control via a JSB.

PSEUDO INSTRUCTIONS

If the Operand field is blank, the Comments field must be blank also, otherwise, the Assembler attempts to interpret the first five characters of the comments as the transfer address symbol.

The Label field of the END statement is ignored.

OBJECT PROGRAM LINKAGE

Linking pseudo instructions provides a means for communication between a main program and its subroutines or among several subprograms that are to be run as a single program. These instructions may be used only in a relocatable program.

The Label field of this class is ignored in all cases. The Operand field is usually divided into many subfields, separated by commas. The first space not preceded by a comma or a left parenthesis terminates the entire field.

	COM	name, [size ₁]] [,name ₂ [size ₂]], ..., name _n [(size _n)]	comments
--	-----	--	----------

COM reserves a block of storage locations that may be used in common by several subprograms. Each name identifies a segment of the block for the subprogram in which the COM statement appears. The sizes are the number of words allotted to the related segments. The size is specified as an octal or decimal integer. If the size is omitted, it is assumed to be one.

Any number of COM statements may appear in a subprogram. Storage locations are assigned contiguously; the length of the block is equal to the sum of the lengths of all segments named in all COM statements in the subprogram.

To refer to the common block, other subprograms must also include a COM statement. The segment names and sizes may be the same or they may differ. Regardless of the names and sizes specified in the separate subprograms, there is only one common block for the combined set. It has the same relative origin; the content of the nth word of common storage is the same for all subprograms.

PSEUDO INSTRUCTIONS

1	5	10	15	20	25	30	35	40	45	50	
Label	Operation	Operand									Comments
PROG1	COM	ADDR1(5),ADDR2(10),ADDR3(10)									
	.										
	.										
	LDA	ADDR2+1			PICK UP SECOND WORD OF SEGMENT						
	.			ADDR2+1							
	.										
	END										
	.										
	.										
PROG2	COM	AAA(2),AAB(2),AAC,AAD(20)									
	.										
	.										
	LDA	AAD+1			PICK UP SECOND WORD OF SEGMENT						
				AAD+1.							

Organization of common block:

<u>PROG1</u> <u>name</u>	<u>PROG2</u> <u>name</u>	<u>Common</u> <u>Block</u>
ADDR1	AAA	(location 1)
		(location 2)
	AAB	(location 3)
		(location 4)
ADDR2	AAC	(location 5)
	AAD	(location 6)
		(location 7)
		(location 8)
ADDR3		(location 9)
		(location 10)
		(location 11)
		(location 12)
		(location 13)
		(location 14)
		(location 15)
		(location 16)
		(location 17)
		(location 18)
		(location 19)
		(location 20)
		(location 21)
		(location 22)
		(location 23)
		(location 24)
		(location 25)

PSEUDO INSTRUCTIONS

The LDA instructions in the two subprograms each refer to the same location in common storage, location 7.

The segment names that appear in the COM statements can be used in the Operand fields of DEF, ABS, EQU, or any memory reference statement; they may not be used as labels elsewhere in the program.

The loader establishes the origin of the common block; the origin cannot be set by the ORG or ORB pseudo instruction. All references to the common area are relocatable.

Two or more subprograms may declare common blocks that differ in size. The subprogram that defines the largest block must be the first submitted for loading.

	ENT	name ₁ [,name ₂ ,...,name _n]	comments
--	-----	--	----------

ENT defines entry points to the program or subprogram. Each name is a symbol that is assigned as a label for some machine operation in the program. Entry points allow another subprogram to refer to this subprogram. All entry points must be defined in the program.

Symbols appearing in an ENT statement may not also appear in EXT or COM statements in the same subprogram.

	EXT	name ₁ [,name ₂ ,...,namen]	comments
--	-----	---	----------

This instruction designates labels in other subprograms that are referenced in this subprogram. The symbols must be defined as entry points by the other subprograms.

The symbols defined in the EXT statement may appear in memory reference statements, the EQU or DEF pseudo instructions. An external symbol must appear

PSEUDO INSTRUCTIONS

alone; it may not be in a multiple term expression or be specified as indirect. References to external locations are processed by the BCS loader as indirect addresses linked through the base page.

Symbols appearing in EXT statements may not also appear in ENT or COM statements in the same subprogram. The label field is ignored.

Example:

1	Label	5	Operation	10	Operand	15	20	25	30	35	40	Comments	45	50
	PROGA		NOP											
			LDA		SAMD									
			.											
			.											
			.											
			JMP		SAMD									
			EXT		SAMD, SAND									
			ENT		PROGA									
			END											
			.											
			.											
	PROGB		NOP											
			.											
			.											
	SAMD		OCT		767									
	SAND		STA		SAMD									
			.											
			.											
			ENT		SAMD, SAND									
			.											
			.											
			JSB		PROGA									
			.											
			.											
			EXT		PROGA									
			.											
			.											
			END											

PSEUDO INSTRUCTIONS

ADDRESS AND SYMBOL DEFINITION

The pseudo operations in this group assign a value or a word location to a symbol which is used as an operand elsewhere in the program.

label	DEF	m[,I]	comments
-------	-----	-------	----------

The address definition statement generates one word of memory as a 15-bit address which may be used as the object of an indirect address found elsewhere in the source program. The symbol appearing in the label is that which is referenced; it appears in the Operand field of a Memory Reference instruction.

The operand field of the DEF statement may be any positive expression in an absolute program; in a relocatable program it may be a relocatable expression or an absolute expression with a value of less than 100_8 . Symbols that do appear in the Operand field may appear as operands of EXT or COM statements, in the same subprogram and as entry points in other subprograms.

The expression in the Operand field may itself be indirect and make reference to another DEF statement elsewhere in the source program.

Example:

1	5	10	15	20	25	30	35	40	45	50
		NAM	PROGN			ZERO-RELATIVE	START	OF	PROGRAM.	
		EXT	SINE, SQRT							
		COM	SCMA(20), SCMB(50)							
		.								
		.								
		JSB	SINE			EXECUTE	SINE	ROUTINE		
		.								
		.								
		LDA	XCMA, I			PICK	UP	COMMON	WORD	INDIRECTLY.
		.								
		.								
XCMA		DEF	SCMA			SCMA	IS	A	15-BIT	ADDRESS.
		.								
		.								
		JSB	XSQ, I			GET	SQUARE	ROOT	USING	TWO-LEVEL
XSQ		DEF	XSQR, I			INDIRECT	ADDRESSING.			
		.								
		.								
XSQR		DEF	SQRT			SQRT	IS	A	15-BIT	ADDRESS.
		END	PROGN							

PSEUDO INSTRUCTIONS

The DEF statement provides the necessary flexibility to perform address arithmetic in programs which are to be assembled in relocatable form. Relocatable programs should not modify the operand of a memory reference instruction.

In the example below, if TBL and LDTBL are in different pages, the Loader processes TBL as an indirect address linked through the base page. The ISZ erroneously increments the loader-provided reference to the base page rather than the value of TBL.

Example:

Label			Operation			Operand			Comments		
1	5	10	15	20	25	30	35	40	45	50	
LDTBL		LDA	TBL								
		.									
		.									
		.									
		ISZ	LDTBL								
		.									
		.									
		.									
TBL		BSS	100								

Assuming the loader might assign absolute locations comparable to the following octal values:

<u>Page</u>	<u>Loc</u>	<u>Opcode</u>	<u>Reference</u>
(0)	(700)	DEF ⋮	400
(1)	(200)	LDA ⋮	(0) 700 (I)
(1)	(300)	ISZ ⋮	(1) 200
(2)	(0)		(TBL)

PSEUDO INSTRUCTIONS

It can be seen that the ISZ instruction would increment the quantity 700 rather than the address of the table (4000_8).

The following assures correct address modification during program execution.

Example:

1	5	10	15	20	25	30	35	40	45	50
Label		Operation		Operand		Comments				
ITBL	DEF	TBL								
LDTBL	LDA	ITBL,I								
	.									
	.									
	.									
	ISZ	ITBL								
	.									
	.									
	.									
TBL	BSS	100								

This sequence might be stored by the loader as:

<u>Page</u>	<u>Loc</u>	<u>Opcode</u>	<u>Reference</u>
(1)	(200)	DEF	4000
(1)	(201)	LDA	200 (I)
		⋮	
(1)	(300)	ISZ	(1) (200)
		⋮	
(2)	(0)		(TBL)

PSEUDO INSTRUCTIONS

The value of 4000 is incremented; each execution of LDA will access successive locations in the table.

label	ABS	m	comments
-------	-----	---	----------

ABS defines a 16-bit absolute value to be stored at the location represented by the label. The Operand field, m, may be any absolute expression; a single symbol must be defined as absolute elsewhere in the program.

Example:

Label	Operation	Operand	Comments
AB	EQU	35	ASSIGNS THE VALUE OF 35 TO THE SYMBOL AB
M35	ABS	-AB	M35 CONTAINS -35.
P35	ABS	AB	P35 CONTAINS 35.
P70	ABS	AB+AB	P70 CONTAINS 70.
P30	ABS	AB-5	P30 CONTAINS 30.

label	EQU	m	comments
-------	-----	---	----------

The EQU pseudo operation assigns to a symbol a value other than the one normally assigned by the program location counter. The symbol in the Label field is assigned the value represented by the Operand field. The Operand field may contain any expression. The value of the operand may be common, base page or program relocatable as well as absolute, but it may not be negative. Symbols appearing in the operand must be previously defined in the source program.

The EQU instruction may be used to symbolically equate two locations in memory, or it may be used to give a value to a symbol. The EQU statement does not result in a machine instruction.

PSEUDO INSTRUCTIONS

Examples:

1	Label	5	Operation	10	Operand	15	20	25	30	35	40	Comments	45	50
			NAM	FAM										
			.											
			.											
			.											
	J3		DEF											
			.											
			.											
			.											
			LDA	J3										
			ADA	ONE										
			STA	J3+1										
	JFOUR		EQU	J3+1										
			.											
			.											
			.											
	MWH		AND	JFOUR										
			.											

Examples:

1	Label	5	Operation	10	Operand	15	20	25	30	35	40	Comments	45	50
			NAM	STOTB										
			.											
			.											
			.											
			COM	TABLA(10)										
			.											
			.											
			.											
	TABL B		EQU	TABLA+5										
			.											
			.											
			.											
			LDA	TABL B+1										
			.											
			.											
			.											
			.											
			NAM	REG										
			.											
			.											
			.											
	A		EQU	0										
	B		EQU	1										
			.											
			.											
			LDA	B										
			.											

PSEUDO INSTRUCTIONS

CONSTANT DEFINITION

The pseudo instructions in this class enter a string of one or more constant values into consecutive words of the object program. The statements may be named by labels so that other program statements can refer to the fields generated by them.

label	ASC	n, <2n characters>	comments
-------	-----	--------------------	----------

ASC generates a string of 2n alphanumeric characters in ASCII code into n consecutive words.[†] One character is right justified in each eight bits; the most significant bit is zero. n may be any expression resulting in an unsigned decimal value in the range 1 through 28. Symbols used in an expression must be previously defined. Anything in the Operand field following 2n characters is treated as comments. If less than 2n characters are detected before the end-of-statement mark, the remaining characters are assumed to be spaces, and are stored as such. The label represents the address of the first two characters.

Example:

Label	Operation	Operand	Comments
1	5	10	15
TTYP	ASC	3,ABCDE	

causes the following:

ALPHABETIC									
	15	14		8	7	6		0	
TTYP			A					B	
			C					D	
			E					A	

EQUIVALENT IN OCTAL NOTATION									
	15	14		8	7	6		0	
TTYP			1	0	1			1	0
			1	0	3			1	0
			1	0	5			0	4

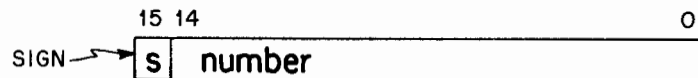
[†] To enter the code for the ASCII symbols which perform some action (e.g., CR and LF), the OCT pseudo instruction must be used.

PSEUDO INSTRUCTIONS

label	DEC	$d_1[.d_2, \dots, d_n]$	comments
-------	-----	-------------------------	----------

DEC records a string of decimal constants into consecutive words. The constants may be either integer or real (floating point), and positive or negative. If no sign is specified, positive is assumed. The decimal number is converted to its binary equivalent by the Assembler. The label, if given, serves as the address of the first word occupied by the constant.

A decimal integer must be in the range of 0 to $2^{15} - 1$; it may assume positive, negative, or zero values. It is converted into one binary word and appears as follows:



Example:

1	Label	5	Operation	10	Operand	15	20	25	30	35	40	Comments	45	50
1	INT		DEC	50	, +328, -300									

causes the following (octal representation)

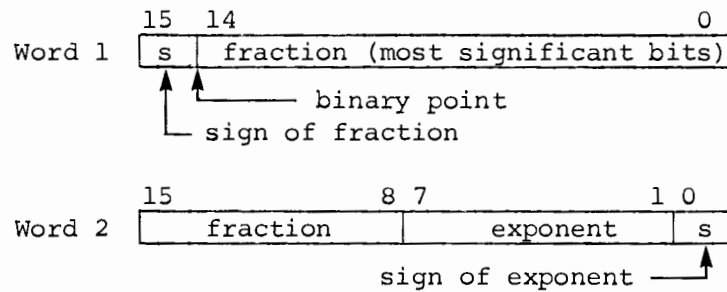
	15	14				0
INT	0	0	0	0	6	2
	0	0	0	5	1	0
	1	7	7	3	2	4

A floating point number has two components, a fraction and an exponent. The exponent specifies the power of 10 by which the fraction is multiplied. The fraction is a signed or unsigned number which may be written with or without a decimal point. The exponent is indicated by the letter E and follows a signed or unsigned decimal integer. The floating point number may have any of the following formats:

$\pm n.n$ $\pm n.$ $\pm n.nE\pm e$ $\pm .nE\pm e$ $\pm n.E\pm e$ $\pm nE\pm e$

PSEUDO INSTRUCTIONS

The number is converted to binary, normalized (leading bits differ), and stored in two computer words. If either the fraction or the exponent is negative, that part is stored in two's complement form.



The floating point number is made up of a 7-bit exponent with sign and a 23-bit fraction with sign. The number must be in the approximate range of 10^{-38} and zero.

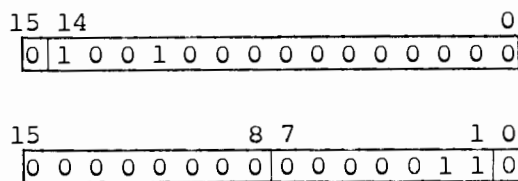
Examples:

1	Label	5	Operation	10	Operand	15	20	25	30	35	40	Comments	45	50
			DEC		.45E1									
			DEC		45.00E-1									
			DEC		4500E-3									
			DEC		4.5									

are all equivalent to

$$.45 \times 10^1$$

and are stored in normalized form as:



PSEUDO INSTRUCTIONS

Label	Operation	Operand	Comments
	DEC	-.695,400E-4	

are stored as:

1	0	1	0	0	1	1	0	0	0	0	1	0	1	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

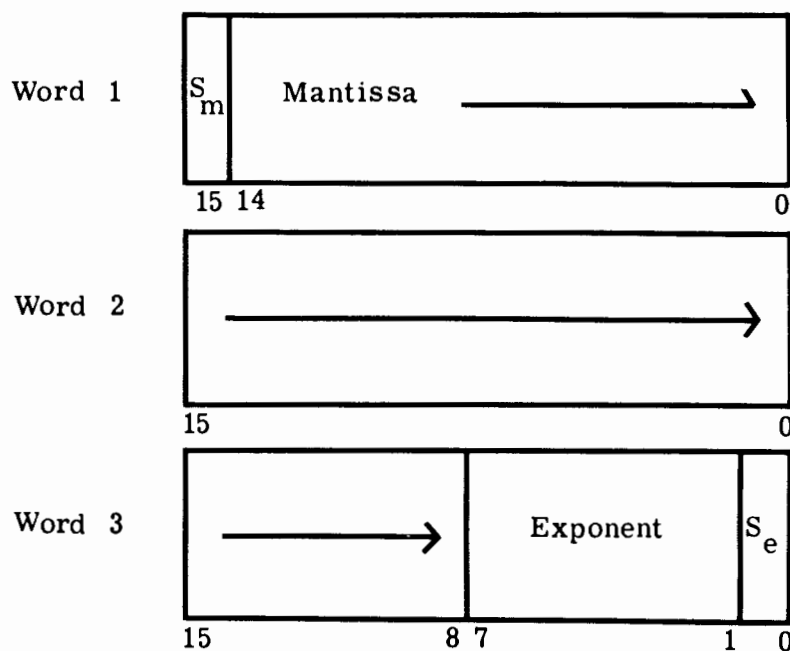
0	0	1	1	1	0	1	1	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

0	1	0	1	0	0	0	1	1	1	1	0	1	0	1	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

1	0	0	0	0	1	0	1	1	1	1	1	0	0	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

label	DEX	d ₁ [, d ₂ , ... , d _n]	comments
-------	-----	---	----------

DEX, for the Extended Assembler, records a string of extended precision decimal constants into consecutive words within a program. Each such extended precision constant occupies three words as shown below:



PSEUDO INSTRUCTIONS

Legend: S_m = Sign of the mantissa (fraction)

S_e = Sign of the Exponent

NOTE: A value is entered only if normalizing of the mantissa is needed.

An extended precision floating point number is made up of a 39-bit mantissa (fraction) and sign and a 7-bit exponent and sign. The exponent and sign will be zero if the mantissa does not have to be normalized.

This is the only form used for DEX. All values, whether they be floating point, integer, fraction, or integer and fraction, will be stored in three words as just described. This storage format is basically an extension of that used for DEC, as previously described:

Examples:

DEX 12,-.45

are stored as:

WORD 1	WORD 2	WORD 3
0110000000000000	0000000000000000	0000000000001000
WORD 1	WORD 2	WORD 3
1000110011001100	1100110011001100	1001101111111111

label	OCT	$o_1 [, o_2, \dots, o_n]$	comments
-------	-----	---------------------------	----------

OCT stores one or more octal constants in consecutive words of the object program. Each constant consists of one to six octal digits (0 to 177777). If no sign is given, the sign is assumed to be positive. If the sign is negative, the two's complement of the binary equivalent is stored. The constants are separated by commas; the last constant is terminated by a space. If less than six digits are indicated for a constant, the data is right justified in the word. A label, if used, acts as the address of the first

PSEUDO INSTRUCTIONS

constant in the string. The letter B must not be used after the constant in the Operand field; it is significant only when defining an octal term in an instruction other than OCT.

Examples:

1	5	10	15	20	25	30	35	40	45	50
		OCT	+0							
		OCT	-2							
NUM		OCT	177,20405,-36							
		OCT	51,77777,-1,10101							
		OCT	107642,177077							
		OCT	1976					ILLEGAL: CONTAINS		
		OCT	-177777					DIGIT 9		
		OCT	177B					ILLEGAL: CONTAINS		
								CHARACTER B		

The previous statements are stored as follows:

		15	14		0
	0	0	0	0	0
	1	7	7	7	6
NUM	0	0	0	1	7
	0	2	0	4	0
	1	7	7	7	4
	0	0	0	0	5
	0	7	7	7	7
	1	7	7	7	7
	0	1	0	1	0
	1	0	7	6	4
	1	7	7	0	7
	X	X	X	X	X
	0	0	0	0	0
	X	X	X	X	X

THE RESULT OF ATTEMPTING TO DEFINE AN ILLEGAL CONSTANT IS UNPREDICTABLE

PSEUDO INSTRUCTIONS

STORAGE ALLOCATION

The storage allocation statement reserves a block of memory for data or for a work area.

label	BSS	m	comments
-------	-----	---	----------

The BSS pseudo operation advances the program or base page location counter according to the value of the operand. The Operand field may contain any expression that results in a positive integer. Symbols, if used, must be previously defined in the program. The label, if given, is the name assigned to the storage area and represents the address of the first word. The initial content of the area set aside by the statement is unaltered by the loader.

ASSEMBLY LISTING CONTROL

Assembly listing control pseudo instructions allow the user to control the assembly listing Output during pass 2 or 3 of the assembly process. These pseudo instructions may be used only when the source program is translated by the Extended Assembler provided for 8K or larger machines (8,192-word memory or larger).

UNL	comments
-----	----------

Output is suppressed from the assembly listing, beginning with the UNL pseudo instruction and continuing for all instructions and comments until either an LST or END pseudo instruction is encountered. Diagnostic messages for errors encountered by the Assembler will be printed, however. The source statement sequence numbers (printed in columns 1-4 of the source program listing) are incremented for the instructions skipped.

PSEUDO INSTRUCTIONS

	LST	comments
--	-----	----------

The LST pseudo instruction causes the source program listing, terminated by a UNL, to be resumed.

A UNL following a UNL, a LST following a LST, and a LST not preceded by a UNL are not considered errors by the Assembler.

	SUP	comments
--	-----	----------

The SUP pseudo instruction suppresses the output of additional code lines from the source program listing. Certain pseudo instructions, because they result in using subroutines, generate more than one line of coding. These additional code lines are suppressed by a SUP instruction until a UNS or the END pseudo instruction is encountered. SUP will suppress additional code lines in the following pseudo instructions:

ASC	DIV	FAD	FSB
OCT	DLD	FDV	MPY
DEC	DST	FMP	

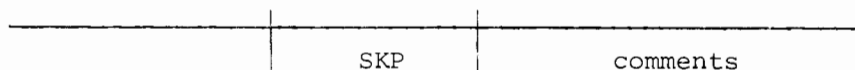
The SUP pseudo instruction may also be used to suppress the listing of literals at the end of the source program listing.

	UNS	comments
--	-----	----------

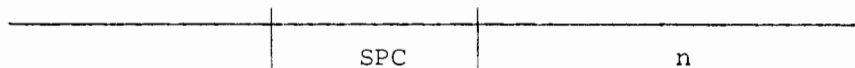
The UNS pseudo instruction causes the printing of additional coding lines, terminated by a SUP, to be resumed.

PSEUDO INSTRUCTIONS

A SUP preceded by another SUP, UNS preceded by UNS, or UNS not preceded by a SUP are not considered errors by the Assembler.



The SKP pseudo instruction causes the source program listing to be skipped to the top of the next page. The SKP instruction is not listed, but the source statement sequence number is incremented for the SKP.



The SPC pseudo instruction causes the source program listing to be skipped a specified number of lines. The list output is skipped n lines, or to the bottom of the page, whichever occurs first. The n may be any absolute expression. The SPC instruction is not listed but the source statement sequence number is incremented for the SPC.



The HED pseudo instruction allows the programmer to specify a heading to be printed at the top of each page of the source program listing.

The heading, m, a string of up to 56 ASCII characters, is printed at the top of each page of the source program listing following the occurrence of the HED pseudo instruction. If HED is encountered before the NAM or ORG at the beginning of a program, the heading will be used on the first page of the source program listing. A HED instruction placed elsewhere in the program causes a skip to the top of the next page.

The heading specified in the HED pseudo instruction will be used on every page until it is changed by a succeeding HED instruction.

PSEUDO INSTRUCTIONS

The source statement containing the HED will not be listed, but source statement sequence number will be incremented.

ARITHMETIC SUBROUTINE CALLS

The members of this group of pseudo instructions request the Assembler to generate calls to arithmetic subroutines* external to the source program. These pseudo instructions may be used in relocatable programs only. The Operand field may contain any relocatable expression or an absolute expression resulting in a value of less than 100_8 .

label	MPY	m[,I]	comments
		=Dn or =Bn	

Multiply the contents of the A-register by the contents of m or the quantity defined by the literal and store the product in registers B and A. B contains the sign of the product and the 15 most significant bits; A contains the least significant bits.

label	DIV	m[,I]	comments
		=Dn or =Bn	

Divide the contents of registers B and A by the contents of m or the quantity defined by the literal. Store the quotient in A and the remainder in B. Initially B contains the sign and the 15 most significant bits of the dividend; A contains the least significant bits.

label	FMP	m[,I]	comments
		=Fn	

* Not intended for use with DEX formatted numbers. For such numbers JSB's to double precision subroutines must be used. See RELOCATABLE SUBROUTINES Manual.

PSEUDO INSTRUCTIONS

Multiply the two-word floating point quantity in registers A and B by the two-word floating point quantity in locations m and m+1 or the quantity defined by the literal. Store the two-word floating point product in registers A and B.

label	FDV	m[,I]	comments
=Fn			

Divide the two-word floating point quantity in registers A and B by the two-word floating point quantity in locations m and m+1 or the quantity defined by the literal. Store the two-word floating point quotient in A and B.

label	FAD	m[,I]	comments
=Fn			

Add the two-word floating point quantity in registers A and B to the two-word floating point quantity in locations m and m+1 or the quantity defined by the literal. Store the two-word floating point sum in A and B.

label	FSB	m[,I]	comments
=Fn			

Subtract the two-word floating point quantity in m and m+1 or the quantity defined by the literal from the two-word floating point quantity in registers A and B and store the difference in A and B.

label	DLD	m[,I]	comments
=Fn			

Load the contents of locations m and m+1 or the quantity defined by the literal into registers A and B respectively.

PSEUDO INSTRUCTIONS

label	DST	m[,I]	comments
-------	-----	-------	----------

Store the contents of registers A and B in locations m and m+1 respectively.

Each use of a statement from this group generates two words of instructions. Symbolically, they could be represented as follows:

```
JSB    <.arithmetic pseudo operation>
DEF    m [,I]
```

An EXT <.arithmetic pseudo operation> is implied preceding the JSB operation.

In the above operations, the overflow bit is set when one of the following conditions occurs:

- Integer overflow
- Floating point overflow or underflow
- Division by zero.

Execution of any of the subroutines alters the contents of the E-Register.

SECTION V

ASSEMBLER INPUT AND OUTPUT

The Assembler accepts as input a paper tape containing a control statement and a source language program. A relocatable source language program may be divided into several subroutines; the designation of these elements is optional. The output produced by the Assembler may include a punched paper tape containing the object program, an object program listing, and diagnostic messages.

CONTROL STATEMENT

The control statement specifies the output to be produced:

$$\text{ASMB}, p_1, p_2, \dots, p_n$$

"ASMB," is entered in positions 1-5. Following the comma are one or more parameters, in any order, which define the output to be produced. The control statement must be terminated by an end-of-statement mark, CR LF .

The parameters may be any legal combination of the following starting in position 6:

- A Absolute: The addresses generated by the Assembler are to be interpreted as absolute locations in memory. The program is a complete entity. It may not include NAM, ORB, COM, ENT, EXT, arithmetic pseudo operation statements or literals. The binary output format is that specified for the Basic Binary loader.

ASSEMBLER INPUT AND OUTPUT

- R Relocatable: The program may be located anywhere in memory. Instruction operands are adjusted as necessary. The binary output format is that specified for the Relocating loader.
- B Binary output: A program is to be punched according to one of the above parameters.
- L List output: A program listing is to be produced either during pass two or pass three (if binary output selected) according to one of the above parameters.
- T Table print: List the symbol table at the end of the first pass. For the Extended Assembler: List the symbol table in alphabetic order in three sections: section 1 for one-character symbols, section 2 for two- and three-character symbols, and section 3 for four- and five-character symbols.
- N Include sets of instructions following the IFN pseudo instruction.
- Z Include sets of instructions following the IFZ pseudo instruction.

Either A or R must be specified in addition to any combination of B, L, or T.

If a programmer wishes to assemble Pass 1 of a source program to check for errors, he can specify only an A or R to be the sole parameter of the Assembler Control Statement, executing only Pass 1. (This produces Pass 1 error messages without listing the program or providing an object tape). Extended Assembler only.

ASSEMBLER INPUT AND OUTPUT

The Assembler control statement must specifically request pass 2 operations (list or punch) in order for pass 2 to be executed. Lack of pass 2 option information causes processing only of pass 1 errors. If a C option is also provided, an automatic cross-reference symbol table is done after pass 1 when operating in the MTS environment.

The control statement may be on the same tape as the source program, or on a separate tape; or it may be entered via the teleprinter keyboard.

SOURCE PROGRAM

The first statement of the program (other than remarks or a HED statement) must be a NAM statement for a relocatable program or an ORG statement for indicating the origin of an absolute program. The last statement must be an END statement and may contain a transfer address for the start of a relocatable program. Each statement is followed by an end-of-statement mark.

BINARY OUTPUT

The punch output is defined by the ASMB control statement. The punch output includes the instructions translated from the source program. It does not include system subroutines referenced within the source program (arithmetic subroutine calls, .IOC., .DIO., .ENTR, etc.)

ASSEMBLER INPUT AND OUTPUT

LIST OUTPUT

Fields of the object program are listed in the following print columns.

<u>Columns</u>	<u>Content</u>
1-4	Source statement sequence number generated by the Assembler
5-6	Blank
7-11	Location (octal)
12	Blank
13-18	Object code word in octal
19	Relocation or external symbol indicator
20	Blank
21-72	First 52 characters of source statement.

Lines consisting entirely of comments (i.e., * in column 1) are printed as follows:

<u>Columns</u>	<u>Content</u>
1-4	Source statement sequence number
5-72	Up to 68 characters of comments

A Symbol Table listing has the following format:

<u>Columns</u>	<u>Content</u>
1-5	Symbol
6	Blank
7	Relocation of external symbol indicator
8	Blank
9-14	Value of the symbol

ASSEMBLER INPUT AND OUTPUT

The characters that designate an external symbol or type of relocation for the Operand field or the symbol are as follows:

<u>Character</u>	<u>Relocation Base</u>
Blank	Absolute
R	Program relocatable
B	Base page relocatable
C	Common relocatable
X	External symbol

At the end of each pass, the following is printed:

```
**NO ERRORS*  
or  
**nnnn ERRORS*
```

The value nnnn indicates the number of errors.

APPENDIX A

HP CHARACTER SET

ASCII CHARACTER FORMAT

b ₇		0	0	0	0	1	1	1	1	
b ₆		0	0	1	1	0	0	1	1	
b ₅		0	1	0	1	0	1	0	1	
b ₄										
b ₃										
b ₂										
b ₁										
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	1	0	0	0	0	0
0	0	0	1	0	0	0	0	0	0	0
0	0	0	1	1	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	0
0	1	0	0	1	0	0	0	0	0	0
0	1	0	1	0	0	0	0	0	0	0
0	1	0	1	1	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	0	0	0
1	0	0	1	0	0	0	0	0	0	0
1	0	0	1	1	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0	0	0
1	1	0	0	0	1	0	0	0	0	0
1	1	0	0	1	0	0	0	0	0	0
1	1	0	0	1	1	0	0	0	0	0
1	1	0	1	0	0	0	0	0	0	0
1	1	0	1	0	1	0	0	0	0	0
1	1	0	1	1	0	0	0	0	0	0
1	1	0	1	1	1	0	0	0	0	0
1	1	1	0	0	0	0	0	0	0	0
1	1	1	0	0	1	0	0	0	0	0
1	1	1	0	1	0	0	0	0	0	0
1	1	1	0	1	1	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0
1	1	1	1	0	1	0	0	0	0	0
1	1	1	1	0	1	1	0	0	0	0
1	1	1	1	1	0	0	0	0	0	0
1	1	1	1	1	0	1	0	0	0	0
1	1	1	1	1	1	0	0	0	0	0
1	1	1	1	1	1	1	0	0	0	0
1	1	1	1	1	1	1	1	0	0	0
1	1	1	1	1	1	1	1	1	0	0
1	1	1	1	1	1	1	1	1	1	0
1	1	1	1	1	1	1	1	1	1	1

Standard 7-bit set code positional order and notation are shown below with b₇ the high-order and b₁ the low-order, bit position.

Example: The code for "R" is:

b ₇	b ₆	b ₅	b ₄	b ₃	b ₂	b ₁
1	0	1	0	0	1	0

HP CHARACTER SET

LEGEND

NULL	Null/Idle	DC ₁ -DC ₃	Device Control
SOM	Start of message	DC ₄ (Stop)	Device control (stop)
EOA	End of address		
EOM	End of message	ERR	Error
EOT	End of transmission	SYNC	Synchronous idle
WRU	"Who are you?"	LEM	Logical end of media
RU	"Are you...?"	S ₀ -S ₇	Separator (information)
BELL	Audible signal	b	Word separator (space, normally non-printing)
FE ₀	Format effector	<	Less than
HT	Horizontal tabulation	>	Greater than
SK	Skip (punched card)	↑	Up arrow (Exponentiation)
LF	Line feed	←	Left arrow (Implies/ Replaced by)
V _{TAB}	Vertical tabulation	\	Reverse slant
FF	Form feed	ACK	Acknowledge
CR	Crrriage return	①	Unassigned control
SO	Shift out	ESC	Escape
SI	Shift in	DEL	Delete/Idle
DC ₀	Device control reserved for data link escape		

HP CHARACTER SET

BINARY CODED DECIMAL FORMAT

Kennedy 1406/1506 ASCII-BCD Conversion

<u>Symbol</u>	<u>BCD (octal code)</u>	<u>ASCII Equivalent (octal code)</u>	<u>Symbol</u>	<u>BCD (octal code)</u>	<u>ASCII Equivalent (octal code)</u>
(Space)	20	040	A	61	101
!	52	041	B	62	102
#	13	043	C	63	103
\$	53	044	D	64	104
%	34	045	E	65	105
&	60	046	F	66	106
'	14	047	G	67	107
(34	050	H	70	110
)	74	051	I	71	111
*	54	052	J	41	112
+	60	053	K	42	113
,	33	054	L	43	114
-	40	055	M	44	115
.	73	056	N	45	116
/	21	057	O	46	117
0	12	060	P	47	120
1	01	061	Q	50	121
2	02	062	R	51	122
3	03	063	S	22	123
4	04	064	T	23	124
5	05	065	U	24	125
6	06	066	V	25	126
7	07	067	W	26	127
8	10	070	X	27	130
9	11	071	Y	30	131
:	15	072	Z	31	132
;	56	073	[75	133
<	76	074	\	36	134
=	13	075]	55	135
>	16	076			
?	72	077			
@	14	100			

Other symbols which may be represented in ASCII are converted to spaces in BCD (20)

HP CHARACTER SET

HP 2020A/B ASCII-BCD Conversion

<u>Symbol</u>	<u>ASCII (Octal code)</u>	<u>BCD (Octal code)</u>	<u>Symbol</u>	<u>ASCII (Octal code)</u>	<u>BCD (Octal code)</u>
(Space)	40	20	A	101	61
!	41	52	B	102	62
"	42	37	C	103	63
#	43	13	D	104	64
\$	44	53	E	105	65
%	45	34	F	106	66
&	46	60 †	G	107	67
'	47	36	H	110	70
(50	75	I	111	71
)	51	55	J	112	41
*	52	54	K	113	42
+	53	60	L	114	43
,	54	33	M	115	44
-	55	40	N	116	45
.	56	73	O	117	46
/	57	21	P	120	47
			Q	121	50
			R	122	51
0	60	12	S	123	22
1	61	01	T	124	23
2	62	02	U	125	24
3	63	03	V	126	25
4	64	04	W	127	26
5	65	05	X	130	27
6	66	06	Y	131	30
7	67	07	Z	132	31
8	70	10	[133	75 †
9	71	11]	135	55 †
			↑	136	77 †
			←	137	32
:	72	15			
;	73	56			
<	74	76			
=	75	35			
>	76	16			
?	77	72			
@	100	14			

† BCD code of 60 always converted to ASCII code 53 (+).

† BCD code of 75 always converted to ASCII code 50 (()) and
BCD code of 55 always converted to ASCII code 51 ()).

APPENDIX B

ASSEMBLER INSTRUCTIONS

<u>Symbols</u>	<u>Meaning</u>
label	Symbolic label, 1-5 alphanumeric characters and periods
m	Memory location represented by an expression
I	Indirect addressing indicator
C	Clear flag indicator
(m,m+1)	Two-word floating point value in m and m+1
comments	Optional comments
[]	Optional portion of field
{ }	One of set may be selected
P	Program Counter
()	Contents of location
^	Logical product
∨	Exclusive "or"
∨	Inclusive "or"
A	A-register
B	B-register
E	E-register
A _n	Bit n of A-register
B _n	Bit n of B-register
b	Bit positions in B- and A-register
<u>(A/B)</u>	Complement of contents of register A or B
(AB)	Two-word floating point value in register A and B
sc	Channel select code represented by an expression
d	Decimal constant
o	Octal constant
r	Repeat count
n	Integer constant
lit	Literal value

INSTRUCTIONS

MACHINE INSTRUCTIONS

MEMORY REFERENCE

Jump and Increment-Skip

ISZ	m [,I]	(m) + 1 → m: then if (m) = 0, execute P + 2 otherwise execute P + 1
JMP	m [,I]	Jump to m; m → P
JSB	m [,I]	Jump subroutine to m: P + 1 → m; m + 1 → P

Add, Load and Store

ADA	$\left\{ \begin{matrix} m [,I] \\ lit \end{matrix} \right\}$	(m) + (A) → A
ADB	$\left\{ \begin{matrix} m [,I] \\ lit \end{matrix} \right\}$	(m) + (B) → B
LDA	$\left\{ \begin{matrix} m [,I] \\ lit \end{matrix} \right\}$	(m) → A
LDB	$\left\{ \begin{matrix} m [,I] \\ lit \end{matrix} \right\}$	(m) → B
STA	m [,I]	(A) → m
STB	m [,I]	(B) → m

Logical

AND	$\left\{ \begin{matrix} m [,I] \\ lit \end{matrix} \right\}$	(m) (A) → A
XOR	$\left\{ \begin{matrix} m [,I] \\ lit \end{matrix} \right\}$	(m) (A) → A
IOR	$\left\{ \begin{matrix} m [,I] \\ lit \end{matrix} \right\}$	(m) (A) → A
CPA	$\left\{ \begin{matrix} m [,I] \\ lit \end{matrix} \right\}$	If (m) ≠ (A), execute P + 2, otherwise execute P + 1
CPB	$\left\{ \begin{matrix} m [,I] \\ lit \end{matrix} \right\}$	If (m) ≠ (B), execute P + 2, otherwise execute

INSTRUCTIONS

MACHINE INSTRUCTIONS (cont.)

REGISTER REFERENCE

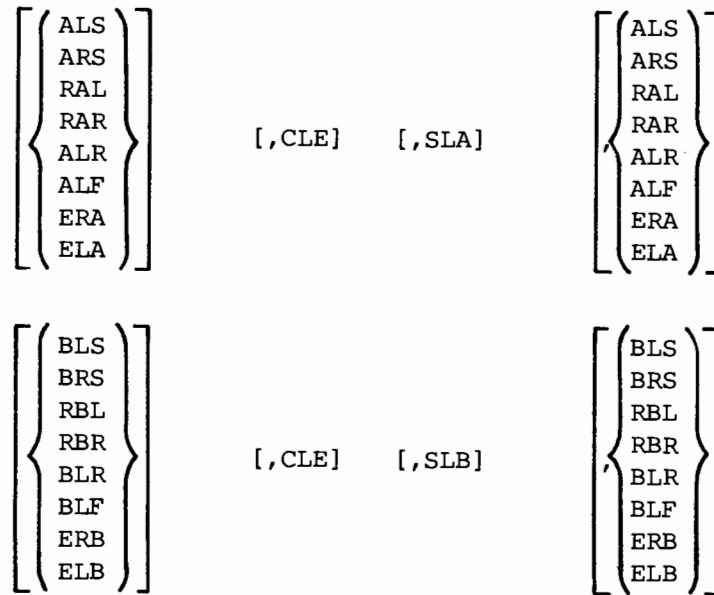
Shift-Rotate

CLE	$0 \rightarrow E$
ALS	Shift (A) left one bit, $0 \rightarrow A_0$, A_{15} unaltered
BLS	Shift (B) left one bit, $0 \rightarrow B_0$, B_{15} unaltered
ARS	Shift (A) right one bit, $(A_{15}) \rightarrow A_{14}$
BRS	Shift (B) right one bit, $(B_{15}) \rightarrow B_{14}$
RAL	Rotate (A) left one bit
RBL	Rotate (B) left one bit
RAR	Rotate (A) right one bit
RBR	Rotate (B) right one bit
ALR	Shift (A) left one bit, $0 \rightarrow A_{15}$
BLR	Shift (B) left one bit, $0 \rightarrow B_{15}$
ERA	Rotate E and A right one bit
ERB	Rotate E and B right one bit
ELA	Rotate E and A left one bit
ELB	Rotate E and B left one bit
ALF	Rotate A left four bits
BLF	Rotate B left four bits
SLA	If $(A_0) = 0$, execute $P + 2$, otherwise execute $P + 1$
SLB	If $(B_0) = 0$, execute $P + 2$, otherwise execute $P + 1$

INSTRUCTIONS

MACHINE INSTRUCTIONS (cont)

Shift-Rotate instructions can be combined as follows:



No-operation

NOP Execute P + 1

Alter-Skip

CLA	0's → A
CLB	0's → B
CMA	$\overline{(A)} \rightarrow A$
CMB	$\overline{(B)} \rightarrow B$
CCA	1's → A
CCB	1's → B
CLE	0 → E
CME	$\overline{(E)} \rightarrow E$
CCE	1 → E
SEZ	If (E) = 0, execute P + 2, otherwise execute P + 1
SSA	If (A ₁₅) = 0, execute P + 2, otherwise execute P + 1
SSB	If (B ₁₅) = 0, execute P + 2, otherwise execute P + 1

INSTRUCTIONS

MACHINE INSTRUCTIONS (cont)

(Alter-Skip (cont)

INA	(A) + 1 → A
INB	(B) + 1 → B
SZA	If (A) = 0, execute P + 2, otherwise execute P + 1
SZB	If (B) = 0, execute P + 2, otherwise execute P + 1
SLA	If (A ₀) = 0, execute P + 2, otherwise execute P + 1
SLB	If (B ₀) = 0, execute P + 2, otherwise execute P + 1
RSS	Reverse sense of skip instructions. If no skip instructions precede, execute P + 2

Alter-Skip instructions can be combined as follows:

$$\left[\begin{array}{c} \{ \text{CLA} \\ \text{CMA} \\ \text{CCA} \} \end{array} \right] [,SEZ] \left[\begin{array}{c} \{ \text{CLE} \\ \text{CME} \\ \text{CCE} \} \end{array} \right] [,SSA] [,SLA] [,INA] [,SZA] [,RSS]$$

$$\left[\begin{array}{c} \{ \text{CLB} \\ \text{CMB} \\ \text{CCB} \} \end{array} \right] [,SEZ] \left[\begin{array}{c} \{ \text{CLE} \\ \text{CME} \\ \text{CCE} \} \end{array} \right] [,SSB] [,SLB] [,INB] [,SZB] [,RSS]$$

INPUT/OUTPUT, OVERFLOW, and HALT

Input/Output

STC	sc	[,C]	Set control bit _{sc} , enable transfer of one element of data between device _{sc} and buffer _{sc}
CLC	sc	[,C]	Clear control bit _{sc} . If sc = 0 clear all control bits.
LIA	sc	[,C]	(buffer _{sc}) → A
LIB	sc	[,C]	(buffer _{sc}) → B
MIA	sc	[,C]	(buffer _{sc}) (A) → A
MIB	sc	[,C]	(buffer _{sc}) (B) → B
OTA	sc	[,C]	(A) → buffer _{sc}
OTB	sc	[,C]	(B) → buffer _{sc}

INSTRUCTIONS

MACHINE INSTRUCTIONS (cont)

Input/Output (cont)

STF	sc	Set flag bit sc . If $\text{sc} = 0$, enable interrupt system. $\text{sc} \stackrel{\text{sc}}{=} 1$ sets overflow bit.
CLF	sc	Clear flag bit sc . If $\text{sc} = 0$, disable interrupt system. If $\text{sc} \stackrel{\text{sc}}{=} 1$, clear overflow bit.
SFC	sc	If (flag bit sc) = 0, execute $P + 2$, otherwise execute $P + 1$. If $\text{sc} = 1$, test overflow bit.
SFS	sc	If (flag bit sc) = 1, execute $P + 2$, otherwise execute $P + 1$. If $\text{sc} = 1$, test overflow bit.

Overflow

CLO		$0 \rightarrow$ overflow bit
STO		$1 \rightarrow$ overflow bit
SOC	[C]	If (overflow bit) = 0, execute $P + 2$, otherwise execute $P + 1$
SOS	[C]	If (overflow bit) = 0, execute $P + 2$, otherwise execute $P + 1$

HALT

HLT	[sc [C]]	Halt computer
-----	----------	---------------

EXTENDED ARITHMETIC UNIT (requires EAU version of Assembler or Extender Assembler)

MPY	$m[,I]$ lit	$(A) \times (m) \rightarrow (B_{+msb} \text{ and } A_{ sb})$
DIV	$m[,I]$ lit	$(B_{+msb} \text{ and } A_{ sb}) / (m) \rightarrow A$, remainder $\rightarrow B$
DLD	$m[,I]$ lit	$(m) \text{ and } (m + 1) \rightarrow A \text{ and } B$
DST	$m[,I]$ lit	$(A) \text{ and } (B) \rightarrow m \text{ and } m + 1$
ASR	b	Arithmetically shift (BA) right b bits, B_{15} extended
ASL	b	Arithmetically shift (BA) left b bits, B_{15} unaltered, 0's to $A_{ sb}$

INSTRUCTIONS

MACHINE INSTRUCTIONS (cont)

EXTENDED ARITHMETIC UNIT (cont)

RRR	b	Rotate (BA) right b bits
RRL	b	Rotate (BA) left b bits
LSR	b	Logically shift (BA) right b bits, 0's to B _{msb}
LSL	b	Logically shift (BA) left b bits, 0's to A _{sb}

PSEUDO INSTRUCTIONS

ASSEMBLER CONTROL

NAM	[name]	Specifies relocatable program and its name.
ORG	m	Gives absolute program origin or origin for a segment of relocatable or absolute program.
ORR		Reset main program location counter at value existing when first ORG or ORB of a string was encountered.
ORB		Defines base page portion of relocatable program.
END	[m]	Terminates source language program. Produces transfer to program starting location, m, if given.
REP <statement>	r	Repeat immediately following statement r times.
IFN <statements> XIF		Include statements in program if control statement contains N.
IFZ <statements> XIF		Include statements in program if control statement contains Z.

INSTRUCTIONS

PSEUDO INSTRUCTIONS (cont)

OBJECT PROGRAM LINKAGE

COM	$name_1[(size_1)][,name_2[(size_2)],...,name_n[(size_n)]]$	Reserves a block of common storage locations. $name_1$ identifies segments of block, each of length size.
ENT	$name_1[,name_2,...,name_n]$	Defines entry points, $name_1$, that may be referred to by other programs.
EXT	$name_1[,name_2,...,name_n]$	Defines external locations, $name_1$, which are labels of other programs, referenced by this program.

ADDRESS AND SYMBOL DEFINITION

label	DEF	$m[,I]$	Generates a 15-bit address which may be referenced indirectly through the label.
label	ABS	m	Defines a 16-bit absolute value to be referenced by the label.
label	EQU	m	Equates the value, m , to the label.

INSTRUCTIONS

PSEUDO INSTRUCTIONS (cont)

CONSTANT DEFINITION

ASC n , <2n characters>	Generates a string of 2n ASCII characters.
DEC d_1 [d_2, \dots, d_n]	Records a string of decimal constants of the form: Integer: $\pm n$ Floating point: $\pm n.n$, $\pm n.$, $\pm .n$, $\pm nE\pm e$, $\pm n.nE\pm e$, $\pm n.E\pm e$, $\pm .nE\pm e$
DEX d_1 [d_2, \dots, d_n]	Records a string of extended precision decimals constants of the form Floating point: $\pm n$, $\pm n.m$, $\pm n.$, $\pm .n$, $\pm nE\pm e$, $\pm n.nE\pm e$, $\pm n.E\pm e$, $\pm .nE\pm e$
OCT o_1 [o_2, \dots, o_n]	Records a string of octal constants of the form: ± 000000

STORAGE ALLOCATION

BSS	m	Reserves a storage area of length, m .
-----	-----	--

INSTRUCTIONS

PSEUDO INSTRUCTIONS (cont)

ARITHMETIC SUBROUTINE CALLS REQUESTS*

MPY†	$\left\{ \begin{matrix} m[,I] \\ \text{lit} \end{matrix} \right\}$	$(A) \times (m) \rightarrow (B_{+msb} \text{ and } A_{ sb})$
DIV†	$\left\{ \begin{matrix} m[,I] \\ \text{lit} \end{matrix} \right\}$	$(B_{+msb} \text{ and } A_{ sb}) / (m) \rightarrow A, \text{ remainder} \rightarrow B$
FMP	$\left\{ \begin{matrix} m[,I] \\ \text{lit} \end{matrix} \right\}$	$(AB) \times (m, m + 1) \rightarrow AB$
FDV	$\left\{ \begin{matrix} m[,I] \\ \text{lit} \end{matrix} \right\}$	$(AB) / (m, m + 1) \rightarrow AB$
FAD	$\left\{ \begin{matrix} m[,I] \\ \text{lit} \end{matrix} \right\}$	$(m, m + 1) + (AB) \rightarrow AB$
FSB	$\left\{ \begin{matrix} m[,I] \\ \text{lit} \end{matrix} \right\}$	$(AB) - (m, m + 1) \rightarrow AB$
DLD†	$\left\{ \begin{matrix} m[,I] \\ \text{lit} \end{matrix} \right\}$	$(m) \text{ and } (m + 1) \rightarrow A \text{ and } B$
DST†	$m[,I]$	$(A) \text{ and } (B) \rightarrow m \text{ and } m + 1$

†For configurations including Extended Arithmetic Unit, these mnemonics generate hardware instructions when the EAU version of the Assembler or Extended Assembler is used.

*Not intended for use with DEX formatted numbers. For such numbers, JSB Machine Instructions must be used.

INSTRUCTIONS

PSEUDO INSTRUCTIONS (cont)

ASSEMBLY LISTING CONTROL

UNL		Suppress assembly listing output.
LST		Resume assembly listing output.
SKP		Skip listing to top of next page.
SPC	n	Skip n lines on listing.
SUP		Suppress listing of extended code lines (e.g., as produced by subroutine calls).
UNS		Resume listing of extended code lines.
HED	<heading>	Print <heading> at top of each page, where <heading> is up to 56 ASCII characters.

APPENDIX C

ALPHABETIC LIST OF INSTRUCTIONS

ABS	Define absolute value
ADA	Add to A
ADB	Add to B
ALF	Rotate A left 4
ALR	Shift A left 1, clear sign
ALS	Shift A left 1
AND	"And" to A
ARS	Shift A right 1, sign carry
ASC	Generate ASCII characters
ASL	Arithmetic long shift left
ASR	Arithmetic long shift right
BLF	Rotate B left 4
BLR	Shift B left 1, clear sign
BLS	Shift B left 1
BRS	Shift B right 1, carry sign
BSS	Reserve block of storage starting at symbol
CCA	Clear and complement A (1's)
CCB	Clear and complement B (1's)
CCE	Clear and complement E (set E = 1)
CLA	Clear A
CLB	Clear B
CLC	Clear I/O control bit
CLE	Clear E
CLF	Clear I/O flag
CLO	Clear overflow bit
CMA	Complement A
CMB	Complement B
CME	Complement E
COM	Reserve block of common storage
CPA	Compare to A, skip if unequal
CPB	Compare to B, skip if unequal

INSTRUCTIONS

ALPHABETIC LIST OF INSTRUCTIONS (cont)

DEC	Defines decimal constants
DEF	Defines address
DEX	Defines extended precision constants
DIV	Divide
DLD	Double load
DST	Double store
ELA	Rotate E and A left 1
ELB	Rotate E and B left 1
END	Terminate program
ENT	Entry point
ERA	Rotate E and A right 1
ERB	Rotate E and B right 1
EQU	Equate symbol
EXT	External reference
FAD	Floating add
FDV	Floating divide
FMP	Floating multiply
FSB	Floating subtract
HED	Print heading at top of each page
HLT	Halt
IFN	When N appears in Control Statement, assemble ensuing instructions
IFZ	When Z appears in Control Statement, assemble ensuing instructions
INA	Increment A by 1
INB	Increment B by 1
IOR	Inclusive "or" to A
ISZ	Increment, then skip if zero
JMP	Jump
JSB	Jump to subroutine
LDA	Load into A
LDB	Load into B

INSTRUCTIONS

ALPHABETIC LIST OF INSTRUCTIONS (cont)

LIA	Load into A from I/O channel
LIB	Load into B from I/O channel
LSL	Logical long shift left
LSR	Logical long shift right
LST	Resume list output (follows a UNL)
MIA	Merge (or) into A from I/O channel
MIB	Merge (or) into B from I/O channel
MPY	Multiply
NAM	Names relocatable program
NOP	No operation
OCT	Defines octal constant
ORB	Establish origin in base page
ORG	Establish program origin
ORR	Reset program location counter
OTA	Output from A to I/O channel
OTB	Output from B to I/O channel
RAL	Rotate A left 1
RAR	Rotate A right 1
RRL	Rotate B left 1
RBR	Rotate B right 1
REP	Repeat next statement
RRL	Rotate A and B left
RRR	Rotate A and B right
RSS	Reverse skip sense
SEZ	Skip if E = 0
SFC	Skip if I/O flag = 0 (clear)
SFS	Skip if I/O flag = 1 (set)
SKP	Skip to top of next page
SLA	Skip if LSB of A = 0
SLB	Skip if LSB of B = 0
SOC	Skip if overflow bit = 0 (clear)
SOS	Skip if overflow bit = 1 (set)

INSTRUCTIONS

ALPHABETIC LIST OF INSTRUCTIONS (cont)

SPC	Space n lines
SSA	Skip if sign A = 0
SSB	Skip if sign B = 0
STA	Store A
STB	Store B
STC	Set I/O control bit
STF	Set I/O flag
STO	Set overflow bit
SUP	Suppress list output of additional code lines
SWP	Switch the (A) and (B)
SZA	Skip if A = 0
SZB	Skip if B = 0
UNL	Suppress list output
UNS	Resume list output of additional code lines
XIF	Terminate an IFN or IFZ group of instructions
XOR	Exclusive "or" to A

APPENDIX D

SAMPLE PROGRAM

Following are two sample problems, the second of which implements several options of the Extended Assembler.

PARTS FILE UPDATE

A master file of parts is updated by a parts usage list to produce a new master parts file. A report, consisting of the parts used and their cost, is also produced.

The master file and the parts usage file contain four word records. Each record of the cost report is eleven words long.

The organization of the files is as follows:

Parts Master Files (PRTSM)

Identification	Quantity	Cost/ Item
----------------	----------	---------------

Identification field of the Parts Master Files exists in ASCII although the entire record is read and written in binary.

Parts Usage File (PRTSU)

Identification	Quantity
----------------	----------

The parts usage file has been recorded in ASCII.

Parts Cost Report (PRTSC)

Identification		Quantity used		Cost for Quantity
----------------	--	---------------	--	----------------------

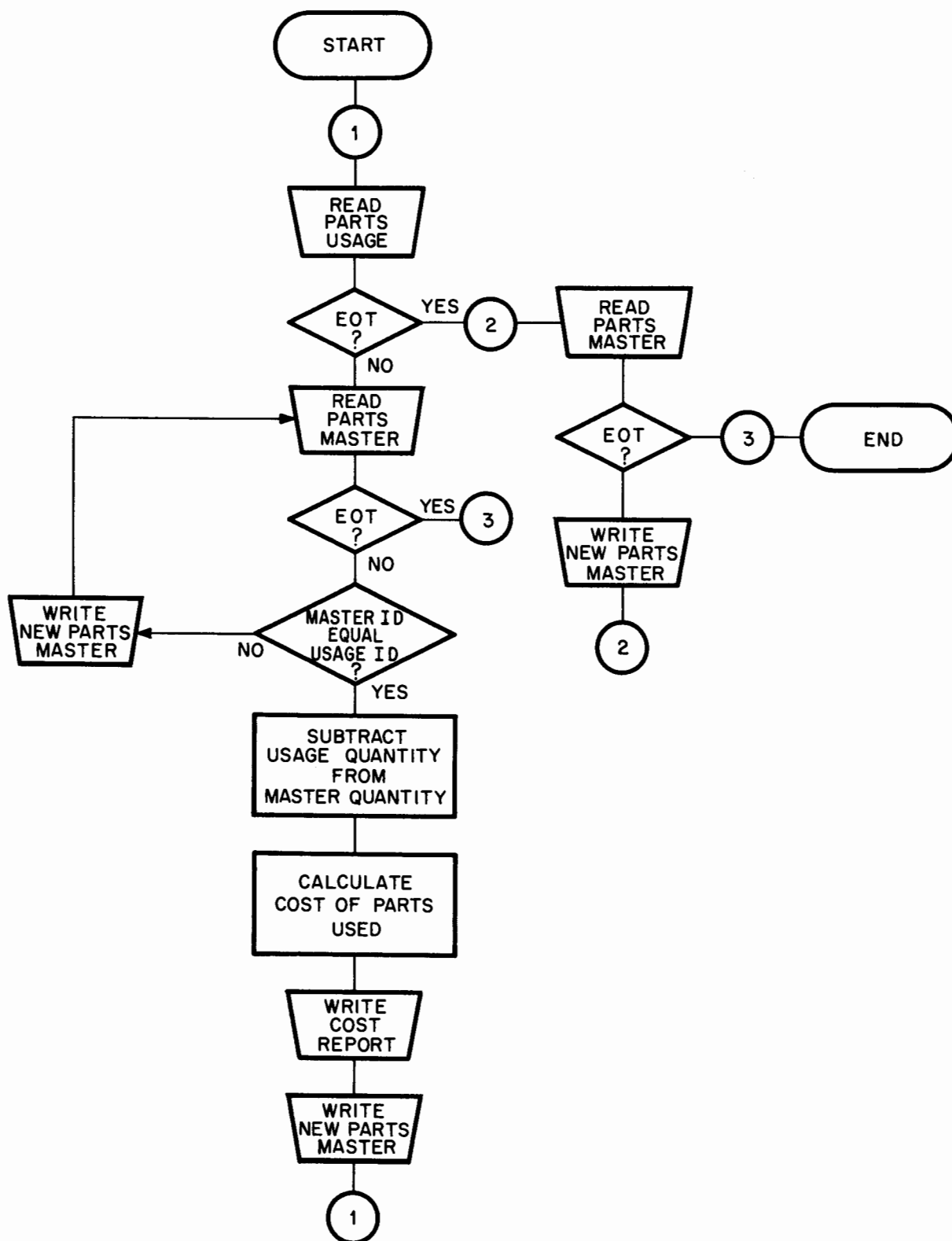
The Parts Cost Report is recorded in ASCII with spacing and editing for printing.

SAMPLE PROGRAMS

SAMPLE PROGRAMS (cont)

The sample program reads and writes the files, adjusts the new stock levels, and calculates the cost. External subprograms perform the binary-to-decimal and decimal-to-binary conversions and handle unrecoverable input/output errors, invalid data conditions, and normal program termination. Input/output operations are performed using the Basic Control System input/output subroutine, .IOC.

SAMPLE PROGRAMS



SAMPLE PROGRAM
GENERAL FLOW CHART

SAMPLE PROGRAMS

SAMPLE ASSEMBLER SYMBOL TABLE OUTPUT

PAGE 0001

	0001	ASMB,R,B,L,T
START	R 000000	
PRTSM	B 000000	
PRTSU	B 000004	
PRTSC	B 000010	
EOTS1	B 000023	
EOTS2	B 000024	
MTEMP	B 000025	
UTEMP	B 000026	
SWTMP	B 000027	
SPACS	B 000031	
DLRSG	B 000033	
A	000000	
B	000001	
.IOC.	X 000001	
BCONV	X 000002	
DCONV	X 000003	
ABORT	X 000004	
HALT	X 000005	
DTOBI	C 000000	
DTOBO	C 000002	
BTODI	C 000003	
BTOD0	C 000005	
OPEN	R 000002	
SPCFL	R 000003	
DLD	X 000006	
DST	X 000007	
READU	R 000013	
CKSTU	R 000020	
RJCTU	R 000035	
EOTU	R 000040	
MSGU	R 000051	
READM	R 000063	
CKSTM	R 000070	
RJCTM	R 000105	
EOTM	R 000110	
MSGM	R 000117	
HLTSW	R 000137	
COMPR	R 000140	
PROCM	R 000157	
PROCC	R 000165	
MPY	X 000010	
CONVM	R 000213	
CONU1	R 000224	
CONU2	R 000235	
CONVC	R 000246	
WRITC	R 000261	
CKSTC	R 000266	
RJCTC	R 000276	
WRITN	R 000301	
CKSTN	R 000306	
RJCTN	R 000316	
** NO ERRORS*		

SAMPLE ASSEMBLER LIST OUTPUT

PAGE 0002

0001	00000		NAM UPDTE	
0002	00000	000000	START NOP	
0003	00001	026002R	JMP OPEN	
0004	00000		ORB	ASSIGN STORAGE & CONSTANTS TO BP
0005	00000	000000	PRTSM BSS 4	MASTER PARTS FILE - BINARY.
0006	00004	000000	PRTSU BSS 4	PARTS USAGE LIST - ASCII.
0007	00010	000000	PRTSC BSS 11	PARTS COST REPORT - ASCII.
0008	00023	026063R	EOTS1 JMP READM	
0009	00024	026301R	EOTS2 JMP WRITN	
0010	00025	000000	MTEMP BSS 1	
0011	00026	000000	UTEMP BSS 1	
0012	00027	000000	SWTMP BSS 2	
0013	00031	020040	SPACS ASC 2,	
	00032	020040		
0014	00033	020044	DLRSG ASC 1, \$	
0015	00000		A EQU 0	
0016	00001		B EQU 1	
0017			EXT .IOC.	PERFORM I/O OPERATIONS USING BCS
0018*				I/O CONTROL ROUTINE.
0019			EXT BCONV	ENTRY POINT FOR DECIMAL(ASCII)
0020*				TO BINARY CONVERSION SUBPROGRAM.
0021			EXT DCONV	ENTRY POINT FOR BINARY TO
0022*				DECIMAL(ASCII) CONVERSION SUB-
0023*				PROGRAM.
0024			EXT ABORT	ENTRY POINT FOR SUBPROGRAM WHICH
0025*				HANDLES UNRECOVERABLE I/O ERRORS
0026*				OR INVALID DATA.
0027			EXT HALT	END OF PROGRAM SUBROUTINE.
0028			COM DTOBI(2),DT0B0,BTODI(2),BTOD0(2)	
0029*				COMMON STORAGE LOCATIONS USED TO
0030*				PASS DATA BETWEEN MAIN PROGRAM
0031*				AND CONVERSION SUBPROGRAMS.
0032	00002		ORR	RESETS PLC AFTER USE OF ORB AT
0033*				BEGINNING OF PROGRAM.
0034	00002	000000	OPEN NOP	
0035	00003	016006X	SPCFL DLD SPACS	STORES EDITING CHARACTERS IN
	00004	000031B		
0036	00005	016007X	DST PRTSC+2	OUTPUT AREA FOR PARTS COST
	00006	000012B		
0037	00007	016007X	DST PRTSC+6	REPORT.
	00010	000016B		
0038	00011	060033B	LDA DLRSG	
0039	00012	070020B	STA PRTSC+8	
0040	00013	016001X	READU JSB .IOC.	READ ONE RECORD FROM USAGE LIST
0041	00014	010001	OCT 10001	LOCATED ON STANDARD UNIT 1
0042	00015	026035R	JMP RJCTU	(TELEPRINTER INPUT). PRTSU IS
0043	00016	000004B	DEF PRTSU	ADDRESS OF STORAGE AREA; AREA IS
0044	00017	000004	DEC 4	4 WORDS LONG.
0045	00020	016001X	CKSTU JSB .IOC.	CHECK STATUS OF UNIT 1.
0046	00021	040001	OCT 40001	
0047	00022	002020	SSA	
0048	00023	026020R	JMP CKSTU	IF BUSY, LOOP UNTIL FREE.
0049	00024	001200	RAL	
0050	00025	002020	SSA	
0051	00026	026030R	JMP **2	
0052	00027	026063R	JMP READM	IF COMPLETE, TRANSFER TO SECTION
0053*				WHICH READS MASTER FILE RECORD.

SAMPLE ASSEMBLER LIST OUTPUT

PAGE 0003

0054	00030	001727	ALF,ALF	TEST END OF TAPE STATUS BIT
0055	00031	001200	RAL	(ORIGINAL BIT 05).
0056	00032	002020	SSA	
0057	00033	026040R	JMP EOTU	IF SET, GO TO EOT PROCEDURE.
0058	00034	026004X	JMP ABORT	IF NOT SET, SOME ERROR CONDITION
0059*				(UNRECOVERABLE) EXISTS.
0060	00035	006020	RJCTU SSB	CHECK CAUSE OF REJECT. IF UNIT
0061	00036	026013R	JMP READU	BUSY LOOP UNTIL FREE. ANY OTHER
0062	00037	026004X	JMP ABORT	CAUSE IS UNRECOVERABLE ERROR.
0063	00040	060023B	EOTU LDA EOTS1	IF END OF USAGE FILE, ALTER
0064	00041	072002R	STA OPEN	PROGRAM SEQUENCE TO BYPASS
0065	00042	060024B	LDA EOTS2	SECTIONS THAT READ AND PROCESS
0066	00043	072140R	STA COMPR	USAGE FILE. PRINT MESSAGE ON
0067	00044	016001X	JSB .IOC.	TELEPRINTER INDICATING EOT.
0068	00045	020002	OCT 20002	
0069	00046	026044R	JMP EOTU+4	
0070	00047	000051R	DEF MSGU	
0071	00050	000011	DEC 9	
0072	00051	042516	MSGU ASC 9,END OF USAGE FILE	
	00052	042040		
	00053	047506		
	00054	020125		
	00055	051501		
	00056	043505		
	00057	020106		
	00060	044514		
	00061	042440		
0073	00062	026063R	JMP READM	
0074	00063	016001X	READM JSB .IOC.	READ A RECORD FROM MASTER PARTS
0075	00064	010105	OCT 10105	FILE ON STANDARD UNIT 05(PUNCHED
0076	00065	026105R	JMP RJCTM	TAPE READER). PRTSM IS ADDRESS
0077	00066	000000B	DEF PRTSM	OF STORAGE AREA; AREA IS 4 WORDS
0078	00067	000004	DEC 4	LONG. RECORD IS IN BINARY FORMAT
0079	00070	016001X	CKSTM JSB .IOC.	CHECK STATUS OF UNIT 5.
0080	00071	040005	OCT 40005	
0081	00072	002020	SSA	
0082	00073	026070R	JMP CKSTM	IF BUSY, LOOP UNTIL FREE.
0083	00074	001200	RAL	
0084	00075	002020	SSA	
0085	00076	026100R	JMP **2	
0086	00077	026140R	JMP COMPR	IF COMPLETE, TRANSFER TO EITHER
0087	00100	001727	ALF,ALF	PROCESSING OR WRITE OUTPUT
0088	00101	001200	RAL	DEPENDING ON SETTING OF COMPR.
0089	00102	002020	SSA	TEST FOR END OF TAPE.
0090	00103	026110R	JMP EOTM	IF END, GO TO EOT PROCEDURE.
0091	00104	026004X	JMP ABORT	IF NOT, AN UNRECOVERABLE ERROR
0092*				EXISTS.
0093	00105	006020	RJCTM SSB	CHECK CONTENTS OF B FOR CAUSE OF
0094	00106	026063R	JMP READM	REJECT. IF UNIT BUSY, LOOP UNTIL
0095	00107	026004X	JMP ABORT	FREE, OTHERWISE I/O ERROR EXISTS
0096	00110	062137R	EOTM LDA HLTSW	ALTER PROGRAM SEQUENCE TO HALT
0097	00111	072315R	STA CKSTN+7	EXECUTION AFTER LAST RECORD IS
0098	00112	016001X	JSB .IOC.	WRITTEN PRINT MESSAGE
0099	00113	020002	OCT 20002	INDICATING END OF MASTER INPUT.
0100	00114	026112R	JMP EOTM+2	
0101	00115	000117R	DEF MSGM	
0102	00116	000017	DEC 15	
0103	00117	042516	MSGM ASC 15,END OF MASTER PARTS FILE INPUT	

SAMPLE ASSEMBLER LIST OUTPUT

PAGE 0004

00120	042040		
00121	047506		
00122	020115		
00123	040523		
00124	052105		
00125	051040		
00126	050101		
00127	051124		
00130	051440		
00131	043111		
00132	046105		
00133	020111		
00134	047120		
00135	052524		
0104	00136	026140R	JMP COMPR
0105	00137	026005X	HLTSW JMP HALT
0106	00140	000000	COMPR NOP
0107	00141	016224R	JSB CONUI
0108	00142	016213R	JSB CONVM
0109	00143	060026B	LDA UTEMP
0110	00144	064025B	LDB MTEMP
0111	00145	050001	CPA B
0112	00146	026157R	JMP PROCM
0113	00147	007004	CMB,INB
0114	00150	040001	ADA B
0115	00151	002020	SSA
0116	00152	026004X	JMP ABORT
0117	00153	062156R	LDA **3
0118	00154	072315R	STA CKSTN+7
0119	00155	026301R	JMP WRITN
0120	00156	026063R	JMP READM
0121	00157	016235R	PROCM JSB CONU2
0122	00160	060002B	LDA PRTSM+2
0123	00161	064027B	LDB UTEMP+1
0124	00162	007004	CMB,INB
0125	00163	040001	ADA B
0126	00164	070002B	STA PRTSM+2
0127	00165	016006X	PROCC DLD PRTSU
	00166	000004B	
0128	00167	016007X	DST PRTSC
	00170	000010B	
0129	00171	016006X	DLD PRTSU+2
	00172	000006B	
0130	00173	016007X	DST PRTSC+4
	00174	000014B	
0131	00175	060003B	LDA PRTSM+3
0132	00176	016010X	MPY UTEMP+1
	00177	000027B	
0133	00200	070030B	STA SWTMP+1
0134	00201	074027B	STB SWTMP
0135	00202	016246R	JSB CONV
0136	00203	016006X	DLD SWTMP
	00204	000027B	
0137	00205	016007X	DST PRTSC+9
	00206	000021B	
0138	00207	062212R	LDA **3
0139	00210	072315R	STA CKSTN+7
0140	00211	026261R	JMP WRITC

END OF PROGRAM SUBROUTINE.

CONVERT ID NUMBER FIELDS OF MASTER AND USAGE FILES TO BIN. LOAD THESE FIELDS FROM TEMPORARY STORAGE.

COMPARE

IF EQUAL, JUMP TO PROCESSING

IF ID NUMBER OF MASTER GREATER THAN ID NUMBER OF USAGE, DATA IN USAGE FILE ERRONEOUS. TERMINATE RUN.

IF ID MASTER LESS THAN ID USAGE, ALTER SEQUENCE: READ NEXT MASTER RECORD IMMEDIATELY AFTER WRITING CURRENT MASTER RECORD.

CONVERT QUANTITY FIELD OF USAGE FILE TO BINARY AND SUBTRACT FROM QUANTITY FIELD OF MASTER AND STORE RESULT.

STORE ID OF PARTS USED IN REPORT

FILE STORAGE AREA.

STORE QUANTITY OF PARTS USED IN

REPORT FILE STORAGE AREA.

COMPUTE COST OF PARTS USED.

CONVERT RESULT TO DECIMAL

STORE IN REPORT FILE AREA.

ALTER SEQUENCE: READ NEXT USAGE RECORD AFTER WRITING CURRENT MASTER RECORD.

SAMPLE ASSEMBLER LIST OUTPUT

PAGE 0005

0141	00212	026013R		JMP READU	
0142	00213	000000	CONVM	NOP	
0143	00214	016006X		DLD PRTSM	STORE ID FIELDS IN COMMON
	00215	000000B			
0144	00216	016007X		DST DTOBI	LOCATIONS TO BE PROCESSED BY
	00217	000000C			
0145	00220	016002X		JSB BCONV	CONVERSION SUBPROGRAM. ON
0146	00221	062002C		LDA DTOBO	COMPLETION, STORE RESULTS IN
0147	00222	070025B		STA MTEMP	LOCATIONS USED BY PROCESSING
0148	00223	126213R		JMP CONVM,I	SECTIONS. CONVM APPLIES TO ID OF
0149	00224	000000	CONU1	NOP	MASTER PARTS FILE; CONU1, TO ID
0150	00225	016006X		DLD PRTSU	OF USAGE; CONU2, TO QUANTITY OF
	00226	000004B			
0151	00227	016007X		DST DTOBI	USAGE; AND CONVC, TO COST OF
	00230	000000C			
0152	00231	016002X		JSB BCONV	PARTS(THIS IS A BINARY TO
0153	00232	062002C		LDA DTOBO	DECIMAL CONVERSION).
0154	00233	070026B		STA UTEMP	
0155	00234	126224R		JMP CONU1,I	
0156	00235	000000	CONU2	NOP	
0157	00236	016006X		DLD PRTSU+2	
	00237	000006B			
0158	00240	016007X		DST DTOBI	
	00241	000000C			
0159	00242	016002X		JSB BCONV	
0160	00243	062002C		LDA DTOBO	
0161	00244	070027B		STA UTEMP+1	
0162	00245	126235R		JMP CONU2,I	
0163	00246	000000	CONVC	NOP	
0164	00247	016006X		DLD SWTMP	
	00250	000027B			
0165	00251	016007X		DST BTODI	
	00252	000003C			
0166	00253	016003X		JSB DCONV	
0167	00254	016006X		DLD BTOD0	
	00255	000005C			
0168	00256	016007X		DST SWTMP	
	00257	000027B			
0169	00260	126246R		JMP CONVC,I	
0170	00261	016001X	WRITC	JSB .IOC.	WRITE ONE RECORD OF PARTS COST
0171	00262	020102		OCT 20102	REPORT ON STANDARD UNIT 2
0172	00263	026276R		JMP RJCTC	(TELEPRINTER OUTPUT). PRTSC IS
0173	00264	000010B		DEF PRJTC	ADDRESS IN STORAGE AREA; AREA IS
0174	00265	000013		DEC 11	11 WORDS LONG. RECORD IS IN ASCI
0175	00266	016001X	CKSTC	JSB .IOC.	CHECK STATUS OF UNIT 2.
0176	00267	040002		OCT 40002	
0177	00270	002020		SSA	
0178	00271	026266R		JMP CKSTC	IF BUSY, LOOP UNTIL FREE.
0179	00272	001200		RAL	
0180	00273	002020		SSA	
0181	00274	026004X		JMP ABORT	TERMINATE IF ANY I/O ERROR.
0182	00275	026301R		JMP WRITN	IF COMPLETE, TRANSFER TO WRITN.
0183	00276	006020	RJCTC	SSB	IF BUSY, LOOP UNTIL FREE.
0184	00277	026261R		JMP WRITC	TERMINATE ON ANY OTHER REJECT
0185	00300	026004X		JMP ABORT	CONDITION.
0186	00301	016001X	WRITN	JSB .IOC.	WRITE ONE RECORD (BINARY) OF
0187	00302	020104		OCT 20104	NEW MASTER PARTS LIST ON UNIT 4
0188	00303	026316R		JMP RJCTN	(TAPE PUNCH). PRTSM (INPUT AREA)

SAMPLE ASSEMBLER LIST OUTPUT

PAGE 0006

0189	00304	000000B	DEF	PRTSM	IS ALSO USED AS OUTPUT AREA.
0190	00305	000004	DEC	4	
0191	00306	016001X	CKSTN	JSB .IOC.	CHECK STATUS OF UNIT 4.
0192	00307	040004	OCT	40004	
0193	00310	002020	SSA		
0194	00311	026306R	JMP	CKSTN	IF BUSY, LOOP UNTIL FREE.
0195	00312	001200	RAL		
0196	00313	002020	SSA		
0197	00314	026004X	JMP	ABORT	
0198	00315	026013R	JMP	READU	
0199	00316	006020	RJCTN	SSB	IF BUSY, LOOP UNTIL FREE, OTHER-
0200	00317	026301R	JMP	WRITN	WISE TERMINATE.
0201	00320	026004X	JMP	ABORT	
0202			END	START	
** NO ERRORS*					

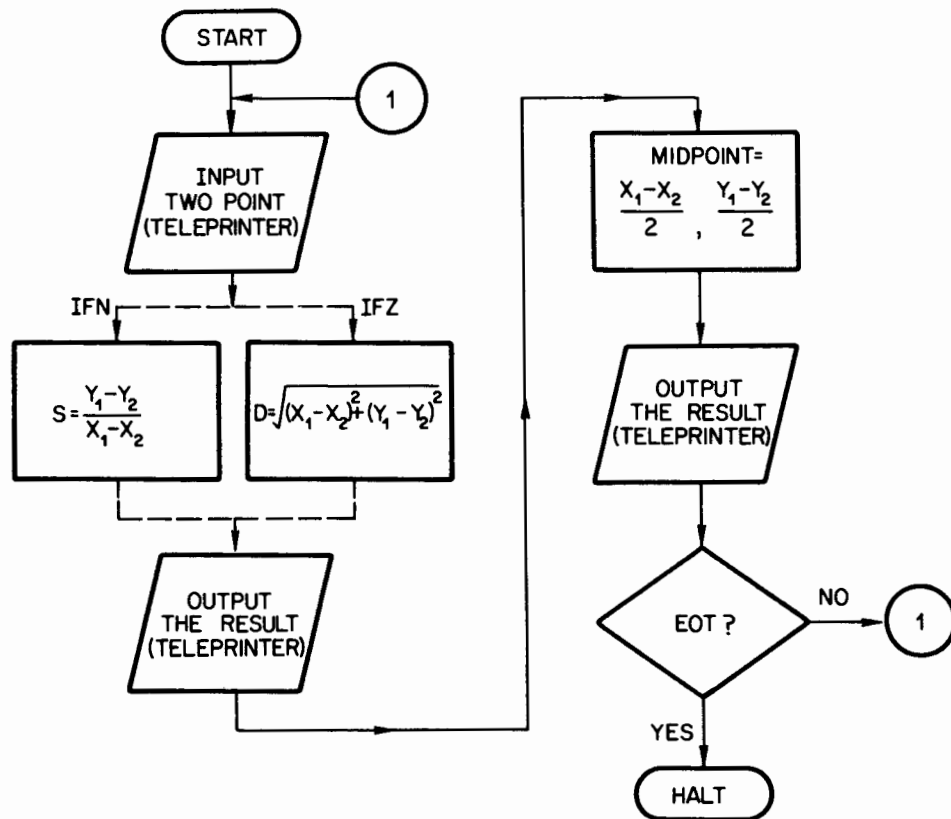
SAMPLE PROGRAMS

CALCULATING DISTANCE

Program "Line" will either calculate the distance between two points or find the slope of the line connecting the points; then the point equidistant from each point (the mid-point) is calculated.

Data is input using the formatter library routine four n-digit real numbers at a time. The first quantity is the X coordinate of the first point; the second quantity is the Y coordinate of the first point; the third and fourth quantities are the X and Y coordinates of the second point.

The result is output to the teleprinter by the formatter library routine; each quantity cannot be more than an eight-digit real number.



GENERAL FLOW CHART

SAMPLE PROGRAMS

Below is the source program as it is typed up on the teleprinter. After it are the assembler listings. The first listing results from including the Z option in the control statement. In the second listing the N option has been included in the control statement.

NOTE: When the complete data tape has been read and the tape reader encounters 10 blank feed frames, an EQT message is typed on the teleprinter and the computer halts. Thus no halt instruction is needed in the program.

```
      HED LINE FORMULI:  DISTANCE, SLOPE, MID-POINT
*   PROGRAM LINE WILL EITHER CALCULATE THE DISTANCE BETWEEN
*   TWO POINTS OR FIND THE SLOPE OF THE LINE CONNECTING
*   THE POINTS; THEN THE POINT EQUIDISTANT FROM EACH
*   POINT (THE MID-POINT) IS CALCULATED.
*   DATA IS INPUT USING THE FORMATTER LIBRARY ROUTINE
*   FOUR N-DIGIT REAL NUMBERS AT A TIME.  THE FIRST
*   QUANTITY IS THE X COORDINATE OF THE FIRST POINT; THE
*   SECOND QUANTITY IS THE Y COORDINATE OF THE FIRST POINT;
*   THE THIRD AND FOURTH QUANTITIES ARE THE X AND Y COORDINATES
*   OF THE SECOND POINT.
*   THE RESULT IS OUTPUT TO THE TELEPRINTER BY THE
*   FORMATTER LIBRARY ROUTINE; EACH QUANTITY CANNOT BE MORE
*   THAN AN EIGHT DIGIT REAL NUMBER.
      NAM LINE
START NOP
      JMP INPUT
      EXT .IOC.,FLOAT,IFIX,SQRT
      EXT .DIO.,.IOI.,.DTA.,.RAR.
      EXT .IOR.,.IAR.
.DATA DEF DATA
.PRIN DEF PRINT
DATA BSS 4
FMT  ASC 3,(F8.3)
FMT2  ASC 8,(F8.3,"",F8.3/)
FMT3  ASC 3,(4I2)
      SKP
*   INPUT THE FIRST TWO POINTS; FOUR DATA WORDS
INPUT NOP
      LDA =B5
      CLB,INB
      JSB .DIO.
      DEF FMT3
      DEF **4
      LDA =B4
      LDB .DATA
      JSB .IAR.
      SPC 3
```

SAMPLE PROGRAMS

* THE DISTANCE BETWEEN THE TWO POINTS:

```
IFZ
LDA DATA+2
CMA, INA
ADA DATA
SPC 1
JMP **5
PRINT REP 4
NOP
SPC 1
STA PRINT
SUP
MPY PRINT
STA PRINT
SPC 1
LDA DATA+3
CMA, INA
ADA DATA+1
STA PRINT+1
MPY PRINT+1
ADA PRINT
SPC 1
JSB FLOAT
JSB SQRT
DST PRINT
XIF
SPC 3
```

* FIND THE SLOPE OF THE LINE

```
IFN
LDA DATA+2
CMA, INA
ADA DATA
JMP **5
PRINT REP 4
NOP
STA PRINT
SPC 1
LDA DATA+3
CMA, INA
ADA DATA+1
CLB
DIV PRINT
DST PRINT
XIF
SPC 3
```

* OUTPUT THE RESULT

```
LDA =B2
CLB
JSB .D10.
DEF FMT
DEF **4
DLD PRINT
JSB .I0R.
JSB .DTA.
SPC 3
```

SAMPLE PROGRAMS

* FIND THE MID-POINT OF THE LINE SEGMENT:

```
LDA DATA
ADA DATA+2
CLB
JSB FLOAT
FMP =F.5
DST PRINT
SPC 1
LDA DATA+1
ADA DATA+3
CLB
JSB FLOAT
FMP =F.5
DST PRINT+2
SPC 1
UNL
LDA =B2
CLB
JSB .DIO.
DEF FMT2
DEF **5
LDA =B2
LDB .PRIN
JSB .RAR.
JSB .DTA.
LST
SPC 3
UNS
JMP INPUT
END START
```

SAMPLE PROGRAMS

PAGE 0001

```
0001                                ASMB,R,L,T,Z
START R 000000
.IOC. X 000001
FLOAT X 000002
IFIX X 000003
SQRT X 000004
.DIO. X 000005
.IOI. X 000006
.DTA. X 000007
.RAR. X 000010
.IOR. X 000011
.IAR. X 000012
.DATA R 000002
.PRIN R 000003
DATA R 000004
FMT R 000010
FMT2 R 000013
FMT3 R 000023
INPUT R 000026
PRINT R 000043
.MPY X 000013
.DST X 000014
.DLD X 000015
.FMP X 000016
** NO ERRORS*
```

SAMPLE PROGRAMS

PAGE 0002 #01 LINE FORMULI: DISTANCE, SLOPE, MID-POINT

```

0002* PROGRAM LINE WILL EITHER CALCULATE THE DISTANCE BETWEEN
0003* TWO POINTS OR FIND THE SLOPE OF THE LINE CONNECTING
0004* THE POINTS; THEN THE POINT EQUIDISTANT FROM EACH
0005* POINT (THE MID-POINT) IS CALCULATED.
0006* DATA IS INPUT USING THE FORMATTER LIBRARY ROUTINE
0007* FOUR N-DIGIT REAL NUMBERS AT A TIME. THE FIRST
0008* QUANTITY IS THE X COORDINATE OF THE FIRST POINT;THE
0009* SECOND QUANTITY IS THE Y COORDINATE OF THE FIRST POINT;
0010* THE THIRD AND FOURTH QUANTITIES ARE THE X AND Y COORDINATES
0011* OF THE SECOND POINT.
0012* THE RESULT IS OUTPUT TO THE TELEPRINTER BY THE
0013* FORMATTER LIBRARY ROUTINE; EACH QUANTITY CANNOT BE MORE
0014* THAN AN EIGHT DIGIT REAL NUMBER.
0015 00000 NAM LINE
0016 00000 000000 START NOP
0017 00001 026026R JMP INPUT
0018 EXT .IOC.,FLOAT,IFIX,SQRT
0019 EXT .DIO.,.IOI.,.DTA.,.RAR.
0020 EXT .IOR.,.IAR.
0021 00002 000004R .DATA DEF DATA
0022 00003 000043R .PRIN DEF PRINT
0023 00004 000000 DATA BSS 4
0024 00010 024106 FMT ASC 3,(F8.3)
      00011 034056
      00012 031451
0025 00013 024106 FMT2 ASC 8,(F8.3,"",F8.3/)
      00014 034056
      00015 031454
      00016 021054
      00017 021054
      00020 043070
      00021 027063
      00022 027451
0026 00023 024064 FMT3 ASC 3,(4I2)
      00024 044462
      00025 024440

```

SAMPLE PROGRAMS

PAGE 0003 #01 LINE FORMULI: DISTANCE, SLOPE, MID-POINT

0028* INPUT THE FIRST TWO POINTS; FOUR DATA WORDS

```
0029 00026 000000 INPUT NOP
0030 00027 062131R LDA =B5
0031 00030 006404 CLB,INB
0032 00031 016005X JSB .D10.
0033 00032 000023R DEF FMT3
0034 00033 000037R DEF **4
0035 00034 062132R LDA =B4
0036 00035 066002R LDB .DATA
0037 00036 016012X JSB .IAR.
```

0039* THE DISTANCE BETWEEN THE TWO POINTS:

```
0040 IFZ
0041 00037 062006R LDA DATA+2
0042 00040 003004 CMA,INA
0043 00041 042004R ADA DATA
```

```
0045 00042 026047R JMP **5
```

```
0046 PRINT REP 4
```

```
0047 00043 000000 NOP
```

```
0047 00044 000000 NOP
```

```
0047 00045 000000 NOP
```

```
0047 00046 000000 NOP
```

```
0049 00047 072043R STA PRINT
```

```
0050 SUP
```

```
0051 00050 016013X MPY PRINT
```

```
0052 00052 072043R STA PRINT
```

```
0054 00053 062007R LDA DATA+3
```

```
0055 00054 003004 CMA,INA
```

```
0056 00055 042005R ADA DATA+1
```

```
0057 00056 072044R STA PRINT+1
```

```
0058 00057 016013X MPY PRINT+1
```

```
0059 00061 042043R ADA PRINT
```

```
0061 00062 016002X JSB FLOAT
```

```
0062 00063 016004X JSB SQRT
```

```
0063 00064 016014X DST PRINT
```

```
0064 XIF
```

0066* FIND THE SLOPE OF THE LINE

```
0067 IFN
```

```
0068 LDA DATA+2
```

```
0069 CMA,INA
```

```
0070 ADA DATA
```

```
0071 JMP **5
```

```
0072 PRINT REP 4
```

```
0073 NOP
```

```
0074 STA PRINT
```

```
0075 SPC 1
```

```
0076 LDA DATA+3
```

```
0077 CMA,INA
```

```
0078 ADA DATA+1
```

SAMPLE PROGRAMS

PAGE 0004 #01 LINE FORMULI: DISTANCE, SLOPE, MID-POINT

```
0079          CLB
0080          DIV PRINT
0081          DST PRINT
0082          XIF
```

```
0084*  OUTPUT THE RESULT
0085  00066 062133R      LDA =B2
0086  00067 006400      CLB
0087  00070 016005X      JSB .DIO.
0088  00071 000010R      DEF FMT
0089  00072 000076R      DEF **4
0090  00073 016015X      DLD PRINT
0091  00075 016011X      JSB .IOR.
0092  00076 016007X      JSB .DTA.
```

```
0094*  FIND THE MID-POINT OF THE LINE SEGMENT:
0095  00077 062004R      LDA DATA
0096  00100 042006R      ADA DATA+2
0097  00101 006400      CLB
0098  00102 016002X      JSB FLOAT
0099  00103 016016X      FMP =F.5
0100  00105 016014X      DST PRINT
```

```
0102  00107 062005R      LDA DATA+1
0103  00110 042007R      ADA DATA+3
0104  00111 006400      CLB
0105  00112 016002X      JSB FLOAT
0106  00113 016016X      FMP =F.5
0107  00115 016014X      DST PRINT+2
```

```
0119          LST
```

```
0121          UNS
0122  00130 026026R      JMP INPUT
      00131 000005
      00132 000004
      00133 000002
      00134 040000
      00135 000000
```

```
0123          END START
** NO ERRORS*
```


SAMPLE PROGRAMS

PAGE 0001

```
0001                                ASMB,R,L,T,N
START R 000000
.IOC. X 000001
FLOAT X 000002
IFIX X 000003
SQRT X 000004
.DIO. X 000005
.IOI. X 000006
.DTA. X 000007
.RAR. X 000010
.IOR. X 000011
.IAR. X 000012
.DATA R 000002
.PRIN R 000003
DATA R 000004
FMT R 000010
FMT2 R 000013
FMT3 R 000023
INPUT R 000026
PRINT R 000043
.DIV X 000013
.DST X 000014
.DLD X 000015
.FMP X 000016
** NO ERRORS*
```

SAMPLE PROGRAMS

PAGE 0002 #01 LINE FORMULI: DISTANCE, SLOPE, MID-POINT

```

0002* PROGRAM LINE WILL EITHER CALCULATE THE DISTANCE BETWEEN
0003* TWO POINTS OR FIND THE SLOPE OF THE LINE CONNECTING
0004* THE POINTS; THEN THE POINT EQUIDISTANT FROM EACH
0005* POINT (THE MID-POINT) IS CALCULATED.
0006* DATA IS INPUT USING THE FORMATTER LIBRARY ROUTINE
0007* FOUR N-DIGIT REAL NUMBERS AT A TIME. THE FIRST
0008* QUANTITY IS THE X COORDINATE OF THE FIRST POINT;THE
0009* SECOND QUANTITY IS THE Y COORDINATE OF THE FIRST POINT;
0010* THE THIRD AND FOURTH QUANTITIES ARE THE X AND Y COORDINATES
0011* OF THE SECOND POINT.
0012* THE RESULT IS OUTPUT TO THE TELEPRINTER BY THE
0013* FORMATTER LIBRARY ROUTINE; EACH QUANTITY CANNOT BE MORE
0014* THAN AN EIGHT DIGIT REAL NUMBER.
0015 00000 NAM LINE
0016 00000 000000 START NOP
0017 00001 026026R JMP INPUT
0018 EXT .IOC.,FLOAT,IFIX,SQRT
0019 EXT .DIO.,.IOI.,.DTA.,.RAR.
0020 EXT .IOR.,.IAR.
0021 00002 000004R .DATA DEF DATA
0022 00003 000043R .PRIN DEF PRINT
0023 00004 000000 DATA BSS 4
0024 00010 024106 FMT ASC 3,(F8.3)
      00011 034056
      00012 031451
0025 00013 024106 FMT2 ASC 8,(F8.3,"","F8.3/)
      00014 034056
      00015 031454
      00016 021054
      00017 021054
      00020 043070
      00021 027063
      00022 027451
0026 00023 024064 FMT3 ASC 3,(4I2)
      00024 044462
      00025 024440

```

SAMPLE PROGRAMS

PAGE 0003 #01 LINE FORMULI: DISTANCE, SLOPE, MID-POINT

0028* INPUT THE FIRST TWO POINTS; FOUR DATA WORDS

```
0029 00026 000000 INPUT NOP
0030 00027 062123R LDA =B5
0031 00030 006404 CLB,INB
0032 00031 016005X JSB .DIO.
0033 00032 000023R DEF FMT3
0034 00033 000037R DEF **4
0035 00034 062124R LDA =B4
0036 00035 066002R LDB .DATA
0037 00036 016012X JSB .IAR.
```

0039* THE DISTANCE BETWEEN THE TWO POINTS:

```
0040 IFZ
0041 LDA DATA+2
0042 CMA,INA
0043 ADA DATA
0044 SPC 1
0045 JMP **5
0046 PRINT REP 4
0047 NOP
0048 SPC 1
0049 STA PRINT
0050 SUP
0051 MPY PRINT
0052 STA PRINT
0053 SPC 1
0054 LDA DATA+3
0055 CMA,INA
0056 ADA DATA+1
0057 STA PRINT+1
0058 MPY PRINT+1
0059 ADA PRINT
0060 SPC 1
0061 JSB FLOAT
0062 JSB SQRT
0063 DST PRINT
0064 XIF
```

0066* FIND THE SLOPE OF THE LINE

```
0067 IFN
0068 00037 062006R LDA DATA+2
0069 00040 003004 CMA,INA
0070 00041 042004R ADA DATA
0071 00042 026047R JMP **5
0072 PRINT REP 4
0073 00043 000000 NOP
0073 00044 000000 NOP
0073 00045 000000 NOP
0073 00046 000000 NOP
0074 00047 072043R STA PRINT

0076 00050 062007R LDA DATA+3
0077 00051 003004 CMA,INA
0078 00052 042005R ADA DATA+1
```

SAMPLE PROGRAMS

PAGE 0004 #01 LINE FORMULI: DISTANCE, SLOPE, MID-POINT

```
0079 00053 006400      CLB
0080 00054 016013X     DIV PRINT
      00055 000043R
0081 00056 016014X     DST PRINT
      00057 000043R
0082                                XIF
```

0084* OUTPUT THE RESULT

```
0085 00060 062125R     LDA =B2
0086 00061 006400      CLB
0087 00062 016005X     JSB .DIO.
0088 00063 000010R     DEF FMT
0089 00064 000070R     DEF **4
0090 00065 016015X     DLD PRINT
      00066 000043R
0091 00067 016011X     JSB .IOR.
0092 00070 016007X     JSB .DTA.
```

0094* FIND THE MID-POINT OF THE LINE SEGMENT:

```
0095 00071 062004R     LDA DATA
0096 00072 042006R     ADA DATA+2
0097 00073 006400      CLB
0098 00074 016002X     JSB FLOAT
0099 00075 016016X     FMP =F.5
      00076 000126R
0100 00077 016014X     DST PRINT
      00100 000043R
```

```
0102 00101 062005R     LDA DATA+1
0103 00102 042007R     ADA DATA+3
0104 00103 006400      CLB
0105 00104 016002X     JSB FLOAT
0106 00105 016016X     FMP =F.5
      00106 000126R
0107 00107 016014X     DST PRINT+2
      00110 000045R
```

```
0119                                LST
```

```
0121                                UNS
0122 00122 026026R     JMP INPUT
      00123 000005
      00124 000004
      00125 000002
      00126 040000
      00127 000000
```

```
0123                                END START
```

** NO ERRORS*

APPENDIX E

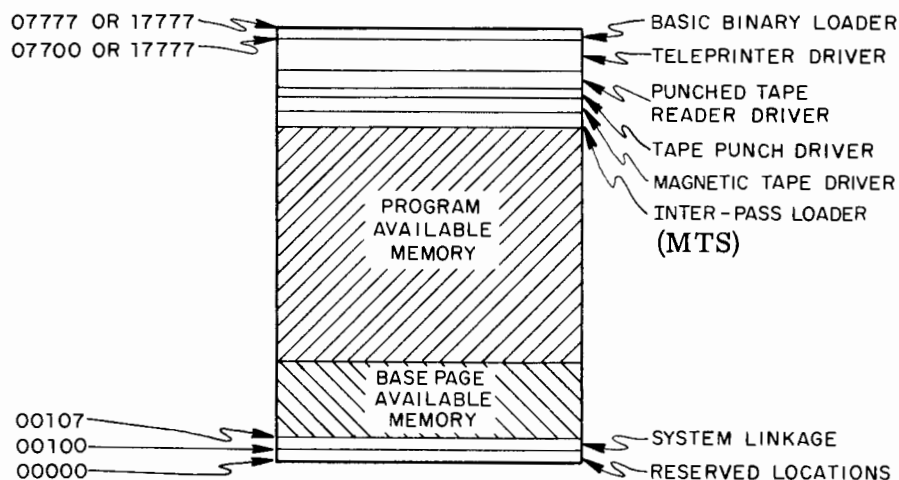
SYSTEM INPUT/OUTPUT SUBROUTINES

The System Input/Output (SIO) subroutines may be used to perform basic input/output operations for programs in absolute form.†

MEMORY ALLOCATION

These drivers are stored in high memory immediately preceding the Basic Binary Loader. The Teleprinter driver must be loaded first; it is stored in the highest portion of this area. The drivers for the Punched Tape Reader (or Marked Card Reader), the Tape Punch, and the Magnetic Tape Unit may then be loaded. The sequence of loading must fall within this order, depending on your equipment configuration: Line Printer Driver, Punched Tape Reader Driver (or Marked Card Reader), Tape Punch Driver, Magnetic Tape Driver, and if needed, the MTS Boot.

The drivers are accessed through 15-bit absolute addresses which are stored in the System Linkage area starting at location 101₈. The allocation of memory is as follows:



†The SIO subroutines are designed for use with FORTRAN, Assembler, Symbolic Editor, etc.; however, they may be used with any absolute object program.

SYSTEM INPUT/OUTPUT SUBROUTINES

OPERATION AND CALLING SEQUENCE: PAPER TAPE DEVICES

All data transmission is accomplished without interrupt control, and therefore, operations are not buffered by the drivers. Control is not returned to the calling program until an operation is completed. Data is transferred to and from buffer storage areas specified in the user program.

The general form of the paper tape input/output calling sequence is:

```
LDA <buffer length> (words or characters)
LDB <buffer address>
JSB 10fB,I (f is Input/Output function)
<normal return>
```

Register Contents

When the JSB is performed, the A-Register must contain the length of the buffer storage area and the B-Register, the address of the buffer. Control returns to the location following the JSB. After an input request is completed, the A-Register contains a positive integer indicating the number of characters or words transmitted, or zeros, if an end-of-tape condition occurred.

The digit supplied for f in the JSB instruction determines the paper tape input/output function to be performed. The value of the operand address is the location in the system linkage that contains the absolute address of the driver entry point. The following are available:

- 101 Input
- 102 List Output
- 103 Punch Output
- 104 Keyboard Input-ASCII data is read from teleprinter and printed as it is received.

SYSTEM INPUT/OUTPUT SUBROUTINES

If the Teleprinter driver alone is loaded, these locations point to entry points of this driver. If Punched Tape Reader and Tape Punch drivers are in memory, location 101 points to the Punched Tape Reader driver and location 103, to the Tape Punch driver. If the latter are to be used, they must be loaded after the Teleprinter driver.

OPERATION AND CALLING SEQUENCE: MAGNETIC TAPE DRIVER

As with the Paper Tape SIO drivers, all data transmission is accomplished without interrupt control. Control is not returned to the calling program until an operation is completed. (Rewind and Rewind/Standby are the only exceptions to this. In these cases return is made as soon as the command is accepted.)

The general form of the calling sequence is:

```
LDA <buffer length> or <file count>
LDB <buffer address> or <record count>
JSB 107B,I
OCT <command code>
<EOF/EOT/SOT return>
<error return>
<normal return>
```

NOTE: Location 107₈ must contain the address of the magnetic tape driver.

Register Contents

Before initiating read or write operations, the A-Register must contain the buffer length. This will be a positive integer if length is defined in characters and a negative integer if length is defined in words. The

SYSTEM INPUT/OUTPUT SUBROUTINES

B-Register must contain the buffer address.

Before initiating tape positioning operations, the A-Register must contain the number of files that are to be spaced. A positive integer indicates forward spacing; a negative integer indicates backward spacing. The B-Register contains the number of records that are to be spaced. A positive integer indicates forward spacing; a negative integer indicates backward spacing. The positioning may be defined in terms of any combination of forward or backward spacing of files and records (e.g., space forward two files then backspace three records). If files only or records only are to be spaced, the contents of the other register should be zeros.

The registers are not used when entering the subroutine to perform one of the following operations:

Write end-of-file	Rewind/Standby
Write file gap	Status
Rewind	

Linkage Address

107B is the system linkage word that contains the absolute address of the entry point for the Magnetic Tape driver.

On return from a read operation, the A-Register contains a positive value indicating the number of words or characters transmitted.

On return from all operations except Rewind and Rewind/Standby, the B-Register contains status of the operation. (See Status.)

SYSTEM INPUT/OUTPUT SUBROUTINES

MAGNETIC TAPE OPERATIONS

The magnetic tape driver will perform the following operations. The pertinent operation is specified by the command code which appears after the OCT in the calling sequence.

<u>Operation</u>	<u>Command Code</u>
Read	0
Write	1
Write End-of-File	2
Rewind (Auto mode)	3
Position	4
Rewind/Standby (Local mode)	5
Gap	6
Status	7

Read

One tape record is read into the buffer. The number of characters or words read is stored in the A-Register. The value will be equal to the buffer length except when the data on tape is less than the length of the buffer. One tape record is read to transfer the number of characters specified into the buffer. The number of characters in that record (not the number transferred) will be stored in the A-Register. If the tape record exceeds the buffer length, the data will be read into the buffer until the buffer is filled, the remainder of the record will be skipped. If the length of an input buffer is an odd number of characters, a read operation will result in the overlaying of the character following the last character of the buffer; the subroutine actually transmits full words only.

Three attempts are made to read the record before returning control to the parity error address.

SYSTEM INPUT/OUTPUT SUBROUTINES

If an EOT condition exists at the time of entry, the command will be ignored and control will be returned to the EOT/EOF address.

If the buffer length specified is 0, control will return to the normal address without any tape movement.

The input buffer storage area can be as large or as small as needed. The number of characters in the tape record will be stored in the A-Register.

Write

The contents of the buffer is written on tape preceded by the record length. Since a minimum of 7 tape characters (12 on 3030) may be written, short records are padded.

If the end-of-tape is detected during the write operation, the normal return is used. The next write operation, however, results in a return of control of the EOF/EOT location; no data is written. If an EOT condition exists at the time of entry, the command will be ignored and control will be returned to the EOT/EOF address.

Write End-of-File

A standard EOF character (17_8 for 2020, 23_8 for 3030) is written on tape. Control return to the normal location with the EOF status on the B-Register. No gap is written.

If the end of tape was reached on a previous write command, control returns to the EOF/EOT location; the character is written.

SYSTEM INPUT/OUTPUT SUBROUTINES

Rewind

This command initiates a rewind operation and then immediately returns control to the normal location.

The calling sequence for a Rewind operation consists of:

```
JSB      107B,I
OCT      3
<normal return>
```

The user need not test status on the rewind operation before issuing the next call.

Position

This is the general command to move the tape. Both file and record operations may be defined in the same operation. Either may be specified for forward or backward spacing. At the completion of the operation the tape will be positioned ready for reading or writing.

An attempt to space beyond the end-of-tape or start-of-tape will terminate the positioning operation and return control to the EOF/EOT/SOT location.

Rewind/Standby

This causes the tape to be positioned at load point and switches the device to local status. Control returns to the normal location immediately after the operation is initiated.

SYSTEM INPUT/OUTPUT SUBROUTINES

The calling sequence for a Rewind/Standby operation consists of:

```
JSB      107B,I
OCT      5
<normal return>
```

An attempt to issue another call on this device results in a halt (102044). The device must be switched to AUTO before the program can continue.

Gap

This command causes a three-inch gap to be written on the tape.

If the end-of-tape was reached on a previous write command, control returns to the EOF/EOT location; the gap is not written.

Status

This command returns certain status bits in the B-Register. The driver performs a clear command whenever it is entered and as a result the only bits that are valid indicators are:

```
Start-of-Tape
End-of-Tape
Write Not Enabled
```

All other commands (except Rewind and Rewind/Standby) provide valid status replies on return to the program.

SYSTEM INPUT/OUTPUT SUBROUTINES

The status reply consists only of bits 8-0 and has the following significance:

<u>Bits 8-0</u>	<u>Condition</u>
1xxxxxxx	Local - The device is in local status
xlxxxxxx	EOF - An End-of-File character (17_8 for 7 track, 23_8 for 9) has been detected while reading, forward spacing, or backspacing.
xxlxxxxx	SOT - The Start-of-Tape marker is under the photo sense head.
xxxlxxxx	EOT - The End-of-Tape reflective marker is sensed while the tape is moving forward. The bit remains set until a rewind command is given.
xxxxlxxx	Timing - A character was lost.
xxxxxlxxx	Reject - a) Tape motion is required and the unit is busy. b) Backward tape motion is required and the tape is at load point. c) A write command is given and the tape reel does not have a write enable ring.
xxxxxxlxx	Write not enabled - Tape reel does not have write enable ring or tape unit is rewinding.
xxxxxxxlx	Parity error - A vertical or longitudinal parity error occurred during reading or writing. (Parity is not checked during forward or backward spacing operations.)
xxxxxxx1	Busy - The tape is in motion or the device is in local status.

SYSTEM INPUT/OUTPUT SUBROUTINES

Following is a table summarizing the tape commands:

Operation	Command	Call		Return	
	Code	A	B	A	B
Read	0	Buffer Length	Buffer Address	Buffer or Record Length	Status
Write	1	Buffer Length	Buffer Address	Buffer Length	Status
Write EOF	2	-	-	-	Status
Rewind (Auto mode)	3	-	-	-	-
Position	4	Number of Files, Direction	Number of Records, Direction	-	Status
Rewind/Standby (Local mode)	5	-	-	-	-
Gap	6	-	-	-	Status
Status	7	-	-	-	Status

Additional Linkage Addresses

Other locations in the system linkage area contain the following:

100₈ Used by the standard software system to store a JMP to the transfer address.

105₈ First word address of available memory.

106₈ Last word address of available memory.

SYSTEM INPUT/OUTPUT SUBROUTINES

The latter two locations may be accessed by an absolute program. The user may store the first word of available memory in 105 by performing the following:

```
ORG 105B
ABS <last location of user program +1>
```

The last word of available memory is established by the drivers; it is the location immediately preceding the first location used by the last driver loaded.

BUFFER STORAGE AREA

The Buffer Address is the location of the first word of data to be written on an output device or the first word of a block reserved for storage of data read from an input device. The length of the buffer area is specified in the A-Register in terms of ASCII input or output characters or binary output words. For binary input, the length of the buffer is the length of the record which is specified in the first character of the record. ASCII and binary input record lengths are given as positive integers. The length of a binary output record is specified as the two's complement of the number of words in the record.

In addition to describing the buffer area in the calling sequence (for first word of binary input record), the area must also be specifically defined in the program, for example with a BSS instruction.

RECORD FORMATS

ASCII Records (Paper Tape)

An ASCII record is a group of characters terminated by an end-of-record mark which consists of a carriage return, (CR), and a line feed, (LF).

SYSTEM INPUT/OUTPUT SUBROUTINES

For an input operation, the length of the record transmitted to the buffer is the number of characters designated in the A-Register, or less if an end-of-record mark is encountered before the character count is exhausted. The codes for (CR) and (LF) are not transmitted to the buffer. An end-of-record mark preceding the first data character is ignored.

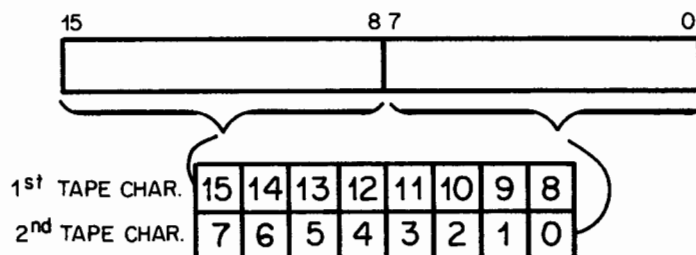
For an output operation, the length of the record is determined by the number of characters designated in the request. An end-of-record mark is supplied at the end of each output operation by the driver.

If a RUB OUT code followed by a (CR) (LF) is encountered on input from the teleprinter or punched tape reader, the current record is ignored (deleted) and the next record transmitted.†

If less than ten feed frames (all zeros) are encountered before the first data character from the punched tape reader, they are ignored. Ten feed frames are interpreted as an end-of-tape condition.

Binary Records (Paper Tape)

A binary record is transmitted exactly as it appears in memory or on 8-level paper tape. Each computer word is translated into two tape "characters" (and vice versa) as follows:



† RUB OUT which appears on the teleprinter keyboard is synonymous with the ASCII symbol DEL.

SYSTEM INPUT/OUTPUT SUBROUTINES

For an output operation, the record length is the number of words designated by the value in the A-Register (the value is the two's complement of the number of words). For input operations, the first word of the record contains a positive integer in bits 15-8 specifying the length (in words) of the record including the first word.

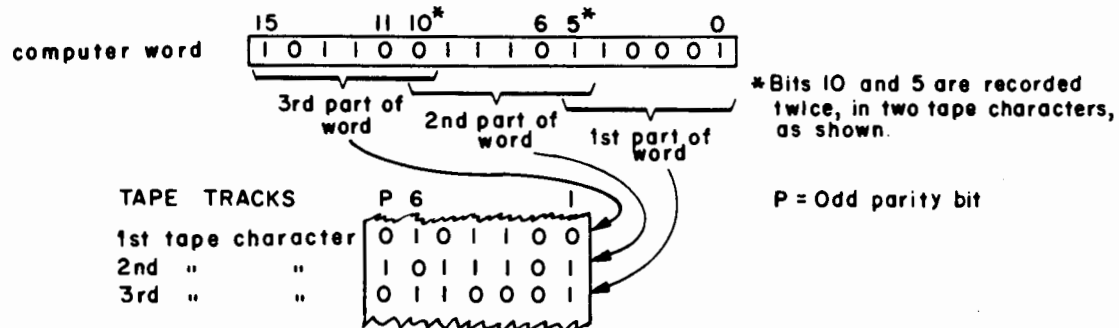
On input operations if less than ten feed frames precede the first data character, they are ignored; ten feedframes are interpreted as an end-of-tape condition. On output, the driver writes four feed frames to serve as a physical record separator.

Binary Records (Magnetic Tape)

The Magnetic Tape subroutine reads and writes binary (odd parity) records only. A record count is supplied by the driver as the first word of the record. This allows automatic padding of short records to the minimum record length with automatic removal of the padded portion of the record on read.

2020 7-LEVEL TAPE

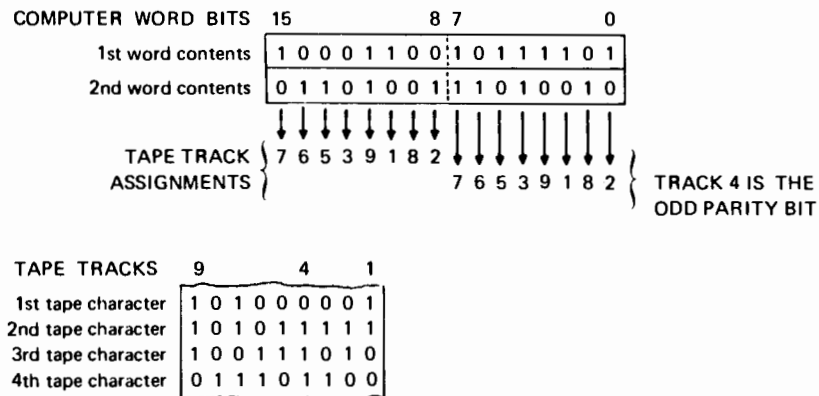
Each computer word is translated into three tape "characters" (and vice versa) as follows:



SYSTEM INPUT/OUTPUT SUBROUTINES

3030 9-LEVEL TAPE

Each computer word is translated into two tape "characters" by repositioning the bits in the following scheme:



OPERATION AND CALLING SEQUENCE: MARK SENSE CARD READER

The SIO Mark Sense Card Reader Driver overlays the Punched Tape Reader Driver exactly, therefore, only one or the other of these two drivers may be used in any one SIO System configuration. Further, the driver has no binary read capability; if this ability is needed, the BCS Mark Sense Card Reader Driver will have to be used.

All data transmission is accomplished without interrupt control. Execution control is not returned to the calling program until a complete card has been read.

The general form of the calling sequence is:

```

LDA <character count> (positive)
LDB <buffer address>
JSB <101B,I>
<normal return>
    
```

SYSTEM INPUT/OUTPUT SUBROUTINES

Register Contents

Before the JSB is executed, the A-Register must contain the character count (the buffer length) and the B-Register must contain the buffer address. Control returns to the location following the JSB; then the A-Register will contain the number of characters transmitted not including trailing blanks, or, if a transmission error was detected, it will contain all zeroes.

APPENDIX F

CONSOLIDATED CODING SHEET

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
D/I	AND	001	0	Z/C	← Memory Address →										
D/I	XOR	010	0	Z/C											
D/I	IOR	011	0	Z/C											
D/I	JSB	001	1	Z/C											
D/I	JMP	010	1	Z/C											
D/I	ISZ	011	1	Z/C											
D/I	AD*	100	A/B	Z/C											
D/I	CP*	101	A/B	Z/C											
D/I	LD*	110	A/B	Z/C											
D/I	ST*	111	A/B	Z/C											
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	SRG	000	A/B	0	D/E	*LS	000			CLE	D/E	SL*	*LS	000	
						*RS	001						*RS	001	
						R*L	010						R*L	010	
						R*R	011						R*R	011	
						*LR	100						*LR	100	
						ER*	101						ER*	101	
						EL*	110						EL*	110	
						*LF	111						*LF	111	
			NOP	000			000				000			000	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	ASG	000	A/B	1		CL*	01			CLE	01		SEZ	SS*	SL*
						CM*	10			CME	10			IN*	SZ*
						CC*	11			CCE	11			RSS	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	IOG	000	A/B	1	H/C	HLT	000			← Select Code →					
				1	0	STF	001								
				1	1	CLF	001								
				1	0	SFC	010								
				1	0	SFS	011								
				1	H/C	MI*	100								
				1	H/C	LI*	101								
				1	H/C	OT*	110								
			0	1	H/C	STC	111								
			1	1	H/C	CLC	111								
				1	0	STO	001				000				001
				1	1	CLO	001				000				001
				1	H/C	SOC	010				000				001
				1	H/C	SOS	011				000				001
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	EAU	000	MPY**	000	010						000			000	
			DIV**	000	100						000			000	
			DLD**	100	010						000			000	
			DST**	100	100						000			000	
			ASR	001	000					0	1				
			ASL	000	000					0	1				
			LSR	001	000					1	0				
			LSL	000	000					1	0				
			RRR	001	001					0	0				
			RRL	000	001					0	0				

Notes: * = A or B.
D/I, A/B, Z/C, D/E, H/C coded: 0/1.
**Second word is Memory Address.

INDEX

A

ABS.....2-7,2-8,2-11,4-16
 Absolute expressions.....2-11
 ASC.....2-7,4-18,4-25
 ADA, ADB.....2-14,3-2
 Address definition.....4-13
 Address expressions.....2-9
 Addressing.....1-1
 ALF.....3-5
 Alphabetic list of
 instructions.....C-1
 ALR.....3-5
 Alter-skip instructions.....3-6
 ALS.....3-5
 AND.....2-14,3-3
 Arithmetic subroutine calls.....4-27
 ARS.....3-5
 ALS.....3-13
 ASR.....3-13
 Assembler Control.....4-1
 Asterick.....2-5,2-9

B

Base page.....2-16
 BCD/ASCII.....A-3
 Binary Coded Decimal Format.....A-3
 Binary output.....5-3
 BLF.....3-5
 BLR.....3-5
 BLS.....3-5
 BRS.....3-5
 BSS.....2-7,4-24

C

CLA, CCB.....3-7
 CCE.....3-7
 Character set.....2-3,A-1
 CLA, CLB.....3-6
 CLC.....3-9
 Clear flag.....2-16
 CLF.....3-10
 CLE.....3-5,3-7
 CLO.....3-10
 CMA, CMB.....3-6
 CME.....3-7
 CPA, CPB.....2-14,3-4
 Coding form.....2-2
 COM.....2-5,2-7,4-9,5-1
 Comments.....2-16
 Control statement.....5-1
 Consolidated coding sheet.....F-1
 Constant definition.....4-18

D

DEC.....2-7,2-8,2-11,4-19,4-25
 DEF.....2-7,2-11,2-15,4-13
 Delimiters.....2-1
 DEX.....2-7,2-8,2-11,4-21
 DIV.....2-14,3-12,4-25,4-27
 DLD.....2-14,3-12,4-25,4-28
 DST.....3-12,4-25,4-28

E

ELA.....3-5
 ELB.....3-5
 END.....2-17,4-3,4-4,4-8
 ENT.....4-11,5-1
 ERA.....3-5

E

ERB.....3-5
EQU.....2-7,2-8,4-16
EXT.....2-5,2-7,4-11,5-1
Extended arithmetic
instructions.....3-11

F

FAD.....2-14,4-25,4-28
FDV.....2-14,4-25,4-28
FMP.....2-14,4-25,4-27
FSB.....2-14,4-25,4-28

H

HED.....4-2,4-26
HLT.....2-17,3-11

I

IFN.....4-4
IFZ.....4-4
INA, INB.....3-7
Indirect addressing.....2-14
Input/output instructions.....3-8
Instructions.....2-1
IOR.....2-14,3-14
ISZ.....3-2

J

JMP.....3-2
JSB.....3-2

L

Labels.....2-4
LDA, LDB.....2-14,3-3
LIA, LIB.....3-9
Listing control.....4-24
LIST output.....5-4
Literals.....2-13
Logical operations.....3-3
LSL.....3-13
LSR.....3-13
LST.....4-25

M

Memory reference instructions.....3-1
MIA, MIB.....3-9
MPY.....2-14,3-12,4-25,4-27

N

NAM.....1-4,2-17,4-1,5-1,5-3
No-operation instruction.....3-6
NOP.....3-4,3-6,4-8
Numeric terms.....2-8

O

Object program linkage.....4-9
OCT.....2-7,2-8,2-11,4-22,4-25
Opcodes.....2-5
Operands.....2-6
Operators.....2-9
Options.....1-4
ORB.....1-4,4-3,5-1
ORG.....1-4,4-1,5-3
ORR.....4-2
OTA, OIB.....3-9
Overflow instructions.....3-10

P

Passes.....1-1
Program location counter.....1-3
Pseudo-instructions.....4-1

R

RAL.....3-5
RAR.....3-5
RBL.....3-5
RBR.....3-5
Register reference
instructions.....3-4
Relocatable expressions.....2-12
Relocation.....1-3
REP.....2-7,4-7
RRL.....3-13
RRR.....3-13
RSS.....3-7

S

Sample program.....D-1
SEZ.....3-7
SFC.....3-10
SFS.....3-10
Shift-rotate instructions.....3-5
SIO drivers.....E-1
SKP.....4-26
SLA, SLB.....3-5, 3-7
SOC.....2-17, 3-11
SOS.....2-17, 3-11
Source program.....5-3
SPC.....4-26
SSA, SSB.....3-7
STA, STB.....3-3
Statements.....2-1
Statement length.....2-3
STC.....3-8
STF.....3-10
STO.....3-11
Storage allocation.....4-24
Summary.....B-1
SUP.....4-25
SWP.....3-14
Symbols.....2-7
Symbol definition.....4-13
SZA, SZB.....3-7

U

UNL.....4-24
UNS.....4-25

X

XIF.....4-4
XOR.....2-14, 3-4

