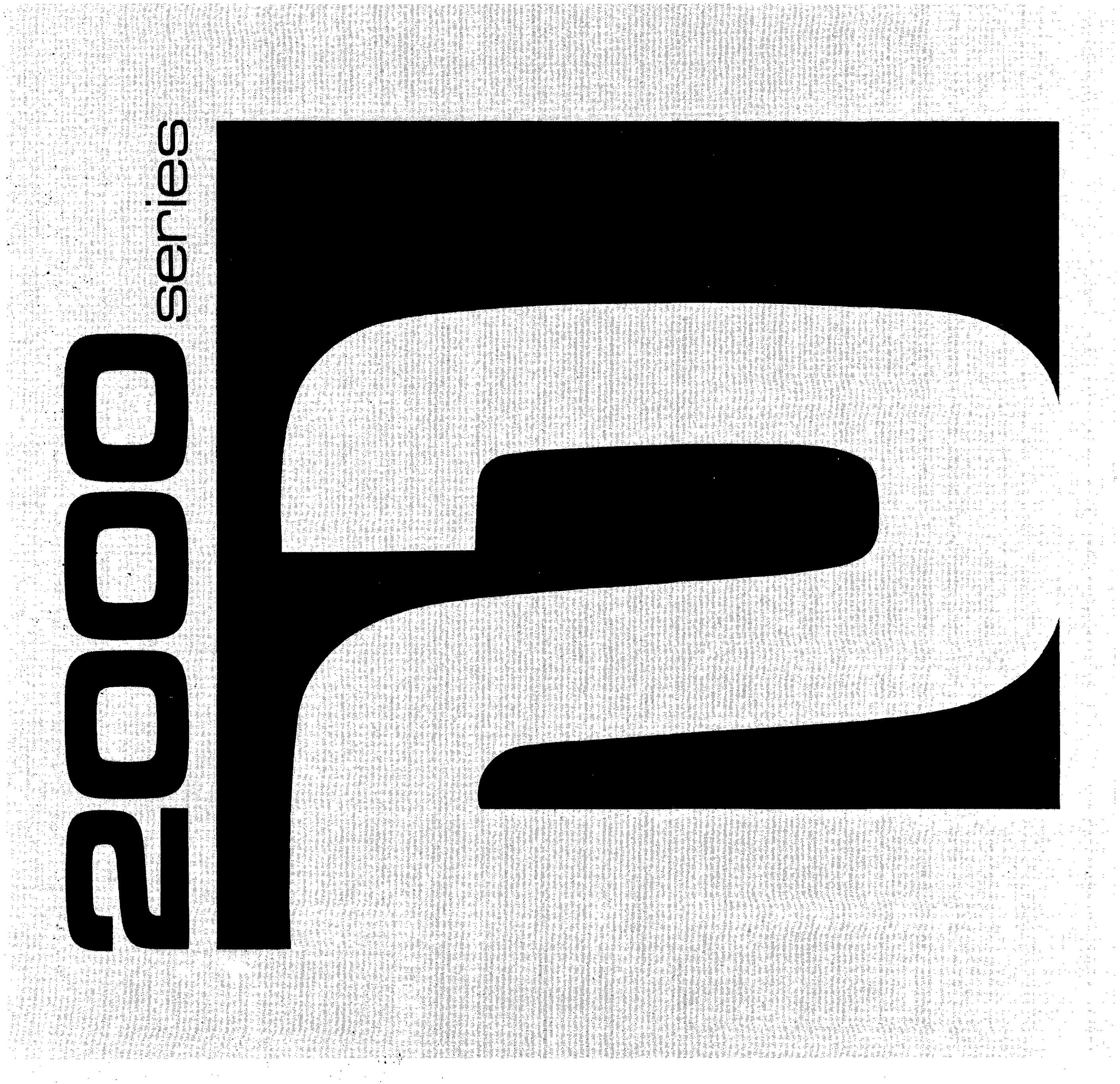
HEWLETT hp PACKARD

Assembler DOS-III

Reference Manual



Assembler For HP 2000 Computers

Reference Manual

HEWLETT PACKARD

HEWLETT-PACKARD COMPANY
11000 WOLFE ROAD, CUPERTINO, CALIFORNIA, 95014

PART NO. 24307-90014

Printed in U.S.A. 7/74

LIST OF EFFECTIVE PAGES

Pages	Effective Date
Title	July 1974
ii to v	July 1974
1-1 to 1-3	July 1974
2-1 to 2-7	July 1974
3-1 to 3-10	July 1974
4-1 to 4-18	July 1974
A-1 to A-3	July 1974
B-1 to B-8	July 1974
C-1 to C-2	July 1974
D-1 to D-3	July 1974
E-1 to E-2	July 1974
I-1 to I-3	July 1974

PREFACE

This manual describes the Assembler which is designed to operate under control of the HP 24307B DOS-III Disc Operating System. The Assembler permits the programmer to use all supported machine instructions for the HP 21MX Computer.

It is assumed that object programs produced by the Assembler will be executed on an HP 21MX Computer. However, the object program may be executed on other HP 2100-Series computers (2114, 2115, 2116, or 2100) if the following machine and pseudo instructions **are not used:**

- Word Processing (described in paragraph 3-5)
- Byte Processing (paragraph 3-6)
- Bit Processing (paragraph 3-7)
- Index Register Group (paragraph 3-11)
- DBL and DBR: Define Left Byte and Define Right Byte (paragraph 4-3)
- BYT: Define Octal Byte Constants (paragraph 4-4)
- MIC: Define User Instruction (paragraphs 4-8 through 4-14)

It is assumed that object programs produced by the Assembler will be loaded and executed under control of the HP 24307B DOS-III Disc Operating System. However, the object program may be loaded and executed under control of some other operating system with the following restrictions:

- ENT pseudo instructions with absolute or common symbols as operands **must not be used.**
- I/O select codes **must not** be defined via the ENT pseudo instruction.
- The DBL (Define Left Byte) and DBR (Define Right Byte) pseudo instructions **must not be used.**

When assembling programs to be run under control of the Basic Control System (BCS), the following restrictions also apply:

- Absolute operands greater than 77_8 are illegal in relocatable programs. However, such usage will **not** be diagnosed as errors by the loader; it will instead result in errors during execution of the object program.
- The ORB (Reset Base Page Origin) pseudo instruction is **not** available.

This manual is arranged in four sections with six appendices. Section I discusses the assembly process in general, program relocation, assembly options, and assembler input and output. Section II describes the source statement format. Section III describes all of the available machine instructions. Section IV describes all of the available assembler pseudo instructions. Appendix A describes the Hewlett-Packard character set. Appendix B summarizes all of the available machine and pseudo instructions (including instruction formats), arranged by instruction type. Appendix C presents a one-sentence definition of all available machine and pseudo instructions, arranged alphabetically by mnemonic. Appendix D presents a tabular summary of the binary format of all available machine instructions. Appendix E describes how to run an assembly under DOS-III. Appendix F describes all of the assembler error messages. For the programmer's convenience, a table of the powers of two is contained on the inside back cover.

CONTENTS

Section I Page

INTRODUCING THE ASSEMBLER

Section II Page

SOURCE STATEMENT FORMAT

Section III Page

MACHINE INSTRUCTIONS

Section IV Page

PSEUDO INSTRUCTIONS

Appendix A Page

HP CHARACTER SET

Appendix B Page

SUMMARY OF INSTRUCTIONS

CONTENTS (continued)

ILLUSTRATIONS

TABLES

INTRODUCING THE ASSEMBLER

SECTION I

The Assembler translates symbolic source language instructions into an object program for execution on the computer. The source language provides mnemonic machine operation codes, assembler-directing pseudo instructions, 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. When relocatable object programs and subprograms are subsequently executed, they are loaded and linked to one another by the relocating loader; absolute object programs are loaded by the Basic Binary Loader or the Basic Binary Disc Loader.

The Assembler can read the source input from paper tape, punched cards, magnetic tape, or from a DOS-III file (or files) in the User Area of the disc. The Assembler outputs the resultant object program on the standard punch output device and/or to the Job Binary Area of the disc in a format acceptable to the DOS-III Relocating Loader.

1-1. ASSEMBLY PROCESSING

The Assembler is a two pass system. 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 and (if requested) prints a symbol table listing on the standard list output device. It also checks for certain possible error conditions and prints error messages on the console device if necessary.

During pass two, the Assembler again examines each statement in the source program along with the symbol table and produces the binary object program. It outputs the object program to the standard punch output device and/or to the Job Binary Area of the disc. If requested, the Assembler also prints the object program listing on the standard list output device. Additional error messages may also be printed on the console device.

Note: If only one output device is available, and if both the punch output and list output are requested, the listing function is deferred and performed as a third pass.

If the source input is being read from a non-disc device, it is written on the disc at the start of pass 1; for pass 2, the source is then read from the disc. However, if there is not sufficient space available on the disc to do this, the source input will have to be read through the non-disc device at the start of pass 2. In such a case, the Assembler prints $END ASMB PASS on the console device at the end of pass 1. The operator responds by reloading the source input into the non-disc device and then entering :GO through the console device.

1-2. 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.

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 operand expressions, but not specifically defined, and symbols that are defined more than once are considered to be in error by the Assembler.

1-3. PROGRAM RELOCATION

Relocatable programs are 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 location of the common storage area is always established by the loader. References to the common area are common relocatable. If a program refers to the common area, the program must 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 2000_8.

1-4. PROGRAM LOCATION COUNTER

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

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.

1-5. ASSEMBLY OPTIONS

The control statement must be the first statement in the source program and it specifies the desired assembly options:

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

"ASMB," is in positions 1-5 of the statement. Following the comma are one or more parameters, in any order. The parameters are shown in Table 1-1. If output to the Job Binary Area is specified in the :PROG,ASMB directive (:PROG,ASMB, . . . ,99), the control statement may contain no parameters. This is the only instance in which the control statement may contain no parameters.

Since they contradict one another, F and X must never appear in the control statement for the same source program. If neither A nor R is specified, R is assumed. If T is omitted, the symbol table listing will **not** be printed. If B is omitted, the object program will **not** be outputted to the standard punch output device (it may, however, be outputted to the Job Binary Area of the disc if so specified in the :PROG,ASMB DOS-III directive).

The control statement can be altered at assembly time through the system console device. To do so, add 100_{10} to the input logical unit number in the :PROG,ASMB directive. In such a case, the Assembler responds by printing the following on the system console device:

```
ENTER NEW CONTROL STATEMENT
```

The user then enters the desired control statement through the system console device. The new control statement overrides the one contained in the source input. During Pass 1, the Assembler prints "CONSOLE!" followed by the new control statement on the list device. The change does **not** appear in the source program listing.

1-6. SOURCE PROGRAM

Following the control statement, 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 terminated by an end-of-statement or end-of-record mark if not on cards.

1-7. BINARY OUTPUT

The binary output is defined by the ASMB control statement. The binary 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.)

1-8. LIST OUTPUT

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

Columns	Content
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:

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

The Symbol Table listing at the end of Pass 1 has the following format:

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

Table 1-1. Control Statement Parameters

PARAMETER	MEANING
A	Absolute assembly. 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. Note that an absolute object program **cannot** be executed under DOS-III.
R	Relocatable assembly. The object program may be loaded anywhere in memory. All operands which refer to memory locations are automatically adjusted as the program is loaded. Operands referring to memory locations greater than 1777_8 must be relocatable expressions. Programs may contain external symbols and entry points, and may refer to common storage.
B	Binary output. An absolute or relocatable object program is to be outputted to the standard punch device.
L	List output. A program listing is to be printed on the standard list device.
T	Symbol table print. A listing of the symbol table is to be printed on the standard list output device.
N,Z	Selective assembly. Sections of the program are to be included or excluded at assembly time depending upon the option specified. See the descriptions of the IFN and IFZ pseudo instructions in Section IV of this manual.
C	Cross reference table print. All references to statement labels, external symbols, and user-defined opcodes are to be listed on the standard list output device after the end of the assembly.
F	Floating point instructions. The floating point machine instructions are to be used instead of the software simulation routines for the following floating point operations: FIX, FLT, FDV, FMP, FAD, and FSB.
X	No EAU hardware. Signifies that the object program will be executed on a machine which does **not** have the Extended Arithmetic Unit (EAU) hardware. This parameter prevents the use of the following EAU instructions: ASR, ASL, RRR, RRL, LSR, LSL, and SWP. In addition, it causes all occurrences of the MPY, DIV, DLD, and DST instructions to be substituted with a call to the appropriate subroutine in the floating point library.

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

Character	Relocation Base
Blank	Absolute
R	Program relocatable
C	Common relocatable
X	External symbol

At the end of each pass, the following is printed on both the console and list device:

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

The value **nnnn** indicates the number of errors.

If there are errors, the message PG xxx is printed on the list device immediately preceding the **nnnn ERRORS* message, where xxx is the page number where the final error was detected. The same message appears in the listing following each error and it points to the page number where the previous error was detected. The backwards pointer following the first error in the program is PG 000.

SOURCE STATEMENT FORMAT

SECTION II

A source language statement consists of a label, an operation code, an operand (or operands) 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 operators. An operand may also be a literal. Indicators may be appended to an operand to specify certain functions such as indirect addressing. The comments portion of the statement is optional.

2-1. STATEMENT CHARACTERISTICS

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

1. Label
2. Opcode
3. Operands
4. Comments

2-2. FIELD DELIMITERS

One or more spaces separate the fields of a statement. An end-of-statement or end-of-record mark terminates the statement. On paper tape this mark is a return, CR, and line feed, LF.† A single space as the first character of a source statement is the null indicator for the label field.

2-3. 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 Comments 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 and operands in a multi-operand field. They may also be used to enhance the appearance of the listing. Within a field they may be used freely when following +, -, ,, or (.

2-4. STATEMENT LENGTH

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

2-5. 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. It is terminated by space. A space in position one is the null field indicator for the label field; the statement is unlabeled.

2-6. LABEL SYMBOL

A label may have one to five characters consisting of A through Z, 0 through 9, and the period.

Note: The Assembler allows the use of certain other characters in the Label field. However, they are reserved for use in Hewlett-Packard programs.

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

```
      LDA                      NO LABEL
.ABCD                          VALID LABEL
.1234                          VALID LABEL
A.123                          VALID LABEL
.                              VALID LABEL
1.AB                           ILLEGAL LABEL - FIRST CHARACTER
                               NUMERIC.
ABC123                         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.
```

Figure 2-1. Sample Labels

```
      COM ACOM(20),BC(30),ABC
LB    EQU 160                  VALID LABEL
      ENT ABC
      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.
ABC                            VALID LABEL
```

Figure 2-2. Label Usage Examples

```
      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.
```

Figure 2-3. Symbolic Term Examples

The first character must be alphabetic or a period. A label of more than five characters could be entered on the source statement, but the Assembler flags this condition as an error and truncates the label from the right to five characters. Some examples are shown in Figure 2-1.

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. However, names appearing in a COM pseudo instruction may be defined as entry points in an ENT psuedo instruction. Some examples are shown in Figure 2-2.

2-7. 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. An asterisk within a label is illegal in any position.

2-8. 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 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
- Define User Opcodes

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

2-9. 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. If more than one operand is required, they are separated from one another by at least one space.

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 joined by the arithmetic operators + and -.

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

Programs may also contain a literal value in the Operand field.

2-10. 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. Some examples are shown in Figure 2-3.

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 or a user-defined instruction
- As a label in the Label field of a BSS, ASC, DEC, DEX, OCT, DEF, BYT, ABS, EQU, DBL, DBR 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

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 be assigned an absolute value 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.

2-11. 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, BYT 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 1777_8. Numeric operands are absolute. Their value is not altered by the assembler or the loader.

2-12. ASTERISK

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

2-13. 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. Some examples are shown in Figure 2-4.

```
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.
```

Figure 2-4. Expression Operator Examples

2-14. 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.

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)

2-15. 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 and may not be indirect.

Within a relocatable program, terms may be program relocatable or common relocatable. A symbol that names an area of common storage is a common relocatable term. A symbol that is defined in any statement other than COM or EQU is a relocatable term. Within one expression all relocatable terms must be program relocatable or common relocatable; the two types may not be mixed.

2-16. 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 program relocatable, common relocatable, or absolute (if less than 2000_8) depending on the definition of the terms composing it.

2-17. ABSOLUTE EXPRESSIONS. An absolute expression may be any arithmetic combination of absolute terms. It may contain relocatable terms alone, or in combination with absolute terms. If relocatable terms 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 a pair remains constant.

An absolute expression reduces to a single absolute value. The value of an absolute multi-term expression may be negative only for ABS pseudo operations. A single numeric term also may be negative in an OCT, DEX, BYT, or DEC pseudo instruction. In a relocatable program the value of an absolute expression must be less than 2000_8 for instructions that reference memory locations (Memory Reference, DEF, Arithmetic subroutine calls, etc.).

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

$A - C_1 + C_2$	$A - P_1 + P_2$	$C_1 - C_2 + A$
$A + A$	$P_1 - P_2$	$-C_1 + C_2 + A$
$* - P_1$	$-P_1 + P_2$	$-A - P_1 + P_2$

The asterisk is program relocatable.

2-18. 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 an 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.

†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.

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

$P_1 - A$	$C_1 - A$	$P_1 - P_2 + *$
$P_1 - P_2 + P_3$	$C_1 - C_2 + C_3$	$C_1 + A$
$* + A$	$* - P_1 + P_2$	$* - A$
$P_2 + A$	$A + C_1$	$- A - P_1 + P_2 + P_3$
$P_1 - *$	$C_1 - C_2 + C_3 - A$	$A + *$
		$- C_1 + C_2 + C_3$

2-19. LITERALS

Literal values may be specified as operands in relocatable programs. The 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.

If the same literal is used in more than one instruction or if different literals have the same value (e.g., =B100 and =D64), only one value is generated, and all instructions using these literals refer to the same location.

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

ADA	CPA	MBT	may use =D, =B, =A, =L
ADB	CPB	MDB	
ADX	DIV	MPY	
ADY	IOR	MVW	
AND	LDA	SBS	
CBS	LDB	TBS	
CBT	LDX	XOR	
CMW	LDY		
DLD	FDV	FSB	may use =F
FMP	FAD		

Examples are as follows:

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 absolute value resulting from the expression.
FMP	=F39.75	Contents of A- and B-Registers multiplied by floating point constant 39.75.

2-20. INDIRECT ADDRESSING

The HP computers provide an indirect addressing capability for memory reference instructions. The operand portion of an indirect instruction contains the address of another location. The 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 address 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. An example is shown in Figure 2-5.

†See CONSTANT DEFINITION, Section IV.

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 a program link location (established by the loader in either the base page or the current page*) and then stores an indirect address in the particular program link location. If the program link location is in the base page, all references to the same operand from other pages will be via the same link location.

2-21. 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. Some examples are shown in Figure 2-6.

2-22. 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-record mark, the end-of-statement mark, (CR) (LF), or the 80th character of a statement terminates the field. The statement length should not exceed 52 characters, the width of the source language portion of the listing. However, 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. Any characters beyond the above limits will not appear on the listing.

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. This limitation applies also to multi-operand instructions.

```
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
```

Figure 2-5. Indirect Addressing Example

```
       STC I07,C      CLEAR FLAG I07 AFTER CONTROL
                      BIT IS SET
       OTB I05,C      CLEAR FLAG I05 AFTER MOVE
```

Figure 2-6. Clear Flag Examples

*Refer to the description of the :PROG,LOADR directive in *HP 24307B DOS-III Disc Operating System (24307-90006).*

MACHINE INSTRUCTIONS

SECTION III

The 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

3-1. MEMORY REFERENCE

The memory reference instructions perform arithmetic, logical, jump, word manipulation, byte manipulation, and bit manipulation operations on the contents of memory locations and the registers. An instruction may directly address the 2048_{10} words of the current and base pages. If required, indirect addressing may be used to refer to all $32{,}768_{10}$ 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 or absolute expressions; however, absolute expressions must be less than 2000_8 in value. If the program is to be assembled in absolute form, the Operand field may contain any expression which is consistent with the location of the program. Literals may not be used in absolute programs. Absolute programs must be complete entities; they may not refer to external subroutines or to common storage.

3-2. 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 skipped.

3-3. 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 =L type.

label	ADA	$\left\{\begin{matrix}\text{m [,I]}\\ \text{lit}\end{matrix}\right\}$	comments

Add the contents of m to A.

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

Add the contents of m to B.

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

Load A with the contents of m.

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

Load B with the contents of m.

label	STA	m [,I]	comments

Store contents of A in m.

label	STB	m [,I]	comments

Store contents of B in m.

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

3-4. LOGICAL OPERATIONS

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

label	AND	m [,I] lit	comments

The logical product ("AND") of the contents of m and the contents of A are placed in A.

label	XOR	m [,I] lit	comments

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

label	IOR	m [,I] lit	comments

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

label	CPA	m [,I] lit	comments

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

label	CPB	m [,I] lit	comments

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

3-5. WORD PROCESSING (21MX ONLY)

The word processing instructions allow the user to move a series of data words from one array in memory to another or to compare (word-by-word) the contents of two arrays in memory.

label	MVW	literal m [,I]	comments

Move words. The A-register contains the starting (lowest) word address of the source array. The B-register contains the starting (lowest) word address of the destination array. The number of words to be moved is specified by **literal** or by the value contained in **m** [,I]. The specified number of words are moved from the source array into the destination array. As each word is moved, the A- and B-registers are incremented by one. The source array is not altered.

label	CMW	literal m [,I]	comments

Compare words. The A-register contains the starting (lowest) word address of array #1. The B-register contains the starting (lowest) word address of array #2. The number of word comparisons to be performed is specified by **literal** or by the value contained in **m** [,I]. The two arrays are compared word-by-word beginning at the specified addresses. The operation is finished when an inequality is detected or when the specified number of word comparisons have been performed. When the operation is finished, the A-register contains the word address of the last word in array #1 which was compared; the B-register contains the starting address of array #2 incremented by the "count" parameter (**literal** or the value in **m** [,I]). If the two arrays are equal, execution proceeds at the next sequential source language instruction (P+3). If array #1 is "less than" array #2, execution proceeds at instruction P+4. If array #1 is "greater than" array #2, execution proceeds at instruction P+5. The two arrays are not altered.

3-6. BYTE PROCESSING (21MX ONLY)

The byte processing instructions allow the user to copy a data byte from memory into the A- or B-register, copy a data byte from the A- or B-register into memory, copy a series of data bytes from one array in memory to another, compare (byte-by-byte) the contents of two arrays in memory, or scan an array in memory for particular data bytes.

A **byte address** is defined as two times the word address of the memory location containing the particular data byte. If the byte location is the low order half of the memory location (bits 0-7), bit 0 of the byte address is set; if the byte location is the high order half of the memory location (bits 8-15), bit 0 of the byte address is clear.

label	LBT	comments

Load byte. The B-register contains the byte address of the byte to be loaded. The specified byte is copied from memory into bits 0-7 of the A-register (bits 8-15 of the A-register are set to zeros). The B-register is then incremented by one. The memory location is not altered.

label	SBT	comments

Store byte. The B-register contains the byte address into which the byte is to be stored. Bits 0-7 of the A-register are copied into the specified memory byte location (bits 8-15 of the A-register are ignored). The B-register is then incremented by one. The A-register is not altered.

label	MBT	{ literal m [,I] }	comments

Move bytes. The A-register contains the starting (lowest) byte address of the source array. The B-register contains the starting (lowest) byte address of the destination array. The number of bytes to be moved is specified by **literal** or by the value contained in **m** [,I]. The specified number of bytes are moved from the source array into the destination array. As each byte is moved, the A- and B-registers are incremented by one. The source array is not altered.

label	CBT	{ literal m [,I] }	comments

Compare bytes. The A-register contains the starting (lowest) byte address of array #1. The B-register contains the starting (lowest) byte address of array #2. The number of byte comparisons to be performed is specified by **literal** or by the value contained in **m** [,I]. The two arrays are compared byte-by-byte beginning at the specified addresses. The operation is finished when an inequality is detected or when the specified number of byte comparisons have been performed. When the operation is finished, the A-register contains the byte address of the last byte in array #1 which was compared; the B-register contains the starting byte address of array #2 incremented by the "count" parameter (**literal** or the value in **m** [,I]). If the two arrays are equal, execution proceeds at the next sequential source language instruction (P+3). If array #1 is "less than" array # 2, execution proceeds at instruction P+4. If array #1 is "greater than" array #2, execution proceeds at instruction P+5. The two arrays are not altered.

label	SFB	comments

Scan for byte. The A-register contains a test byte in bits 0-7 and a termination byte in bits 8-15. The B-register contains the starting (lowest) byte address of the array to be scanned. The array is compared byte-by-byte against both the test and termination bytes starting at the specified address. The operation is finished when a positive comparison is detected or when the end of memory is reached. If the test byte is detected, execution proceeds at the next sequential source language instruction (P+1) and the B-register contains the address of the test byte in the array. If the termination byte is detected, execution proceeds at instruction P+2 and the B-register contains the address plus one of the termination byte in the array. If the end of memory is detected, execution proceeds at instruction P+2 and the B-register contains all zeros. The A-register and the array are not altered.

3-7. BIT PROCESSING (21MX ONLY)

The bit processing instructions allow the user to selectively test, set, or clear bits in a memory location according to the contents of a mask. In the descriptions below, **addrl** and **addr2** may be operand expressions.

label	TBS	{ literal addrl[,I] }	addr2[,I]	comments

Test bits. **literal** is a test mask, **addrl**[,I] is the address of a memory location containing a test mask, and **addr2**[,I] is the address of a memory location containing the bits to be tested. The bits in **addr2**[,I] which correspond to the "1" bits in the mask are tested. All other bits in **addr2**[,I] are ignored. If all the applicable bits in **addr2**[,I] are set, execution proceeds at the next sequential source language instruction (P+3). If any of the applicable bits in **addr2**[,I] are clear, execution proceeds at instruction P+4.

label	SBS	{ literal addrl[,I] }	addr2[,I]	comments

Set bits. **literal** is a mask, **addrl**[,I] is the address of a memory location containing a mask, and **addr2**[,I] is the address of a memory location containing the bits to be set. The bits in **addr2**[,I] which correspond to the "1" bits in the mask are set. All other bits in **addr2**[,I] are not affected. Functionally, the SBS instruction is a "logical OR" operation.

label	CBS	{ literal addrl[,I] }	addr2[,I]	comments

Clear bits. **literal** is a mask, **addrl**[,I] is the address of a memory location containing a mask, and **addr2**[,I] is the address of a memory location containing the bits to be cleared. The bits in **addr2**[,I] which correspond to the "1" bits in the mask are cleared. All other bits in **addr2**[,I] are not affected.

3-8. REGISTER REFERENCE

The register reference instructions include a shift-rotate group, an alter-skip group, an index register group, and NOP (no operation). For the shift-rotate and alter-skip groups, the instruction mnemonics within each group may be combined into a single source statement to cause multiple operations to be executed during one memory cycle. In such cases, successive mnemonics within a single source statement are separated from one another by a comma.

3-9. 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:

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

$$\text{label}\ \left[\left\{\begin{matrix}\text{BLS}\\ \text{BRS}\\ \text{RBL}\\ \text{RBR}\\ \text{BLR}\\ \text{BLF}\\ \text{ERB}\\ \text{ELB}\end{matrix}\right\}\right]\ \text{[,CLE] [,SLB]}\ \left[,\left\{\begin{matrix}\text{BLS}\\ \text{BRS}\\ \text{RBL}\\ \text{RBR}\\ \text{BLR}\\ \text{BLF}\\ \text{ERB}\\ \text{ELB}\end{matrix}\right\}\right]\ \text{comments}$$

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.

3-10. 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
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 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).

3-11. INDEX REGISTER GROUP (21MX ONLY)

This group contains 32 instructions which perform various operations involving the use of index registers X and Y. An instruction may directly address the 2048_{10} words of the current and base pages. If required, indirect addressing may be used (except where noted otherwise) to refer to all $32,768_{10}$ words of memory. Expressions in the Operand field are evaluated modulo 2^{10}.

label	CAX	comments

Copy A to X. The contents of the A-register are copied into the X-register. The A-register is not altered.

label	CBX	comments

Copy B to X. The contents of the B-register are copied into the X-register. The B-register is not altered.

label	CAY	comments

Copy A to Y. The contents of the A-register are copied into the Y-register. The A-register is not altered.

label	CBY	comments

Copy B to Y. The contents of the B-register are copied into the Y-register. The B-register is not altered.

label	CXA	comments

Copy X to A. The contents of the X-register are copied into the A-register. The X-register is not altered.

label	CXB	comments

Copy X to B. The contents of the X-register are copied into the B-register. The X-register is not altered.

label	CYA	comments

Copy Y to A. The contents of the Y-register are copied into the A-register. The Y-register is not altered.

label	CYB	comments

Copy Y to B. The contents of the Y-register are copied into the B-register. The Y-register is not altered.

label	XAX	comments

Exchange A and X. The contents of the A-register are copied into the X-register and the contents of the X-register are copied into the A-register.

label	XBX	comments

Exchange B and X. The contents of the B-register are copied into the X-register and the contents of the X-register are copied into the B-register.

label	XAY	comments

Exchange A and Y. The contents of the A-register are copied into the Y-register and the contents of the Y-register are copied into the A-register.

label	XBY	comments

Exchange B and Y. The contents of the B-register are copied into the Y-register and the contents of the Y-register are copied into the B-register.

label	ISX	comments

Increment X and skip if zero. The contents of the X-register are incremented by one and then tested. If the new value in X is zero, the next sequential instruction (P+1) is skipped and execution proceeds at instruction P+2; if the new value in X is non-zero, execution proceeds at instruction P+1.

label	ISY	comments

Increment Y and skip if zero. The contents of the Y-register are incremented by one and then tested. If the new value in Y is zero, the next sequential instruction (P+1) is skipped and execution proceeds at instruction P+2; if the new value in Y is non-zero, execution proceeds at instruction P+1.

label	DSX	comments

Decrement X and skip if zero. The contents of the X-register are decremented by one and then tested. If the new value in X is zero, the next sequential instruction (P+1) is skipped and execution proceeds at instruction P+2; if the new value in X is non-zero, execution proceeds at instruction P+1.

label	DSY	comments

Decrement Y and skip if zero. The contents of the Y-register are decremented by one and then tested. If the new value in Y is zero, the next sequential instruction (P+1) is skipped and execution proceeds at instruction P+2; if the new value in Y is non-zero, execution proceeds at instruction P+1.

label	LDX	{ m [,I] literal }	comments

Load X from memory. The contents of the specified memory location are copied into the X-register. Indirect addressing may be used. The memory location is not altered.

label	LDY	{ m [,I] literal }	comments

Load Y from memory. The contents of the specified memory location are copied into the Y-register. Indirect addressing may be used. The memory location is not altered.

label	STX	m [,I]	comments

Store X into memory. The contents of the X-register are copied into the specified memory location. Indirect addressing may be used. The X-register is not altered.

label	STY	m [,I]	comments

Store Y into memory. The contents of the Y-register are copied into the specified memory location. Indirect addressing may be used. The Y-register is not altered.

label	LAX	m [,I]	comments

Load A from memory indexed by X. The contents of the specified memory location are copied into the A-register. Indirect addressing may be used. The address of the memory location is computed by adding the contents of the X-register to **m** or to **m**,I. Note that indirect addressing (if specified) is performed first and then the address is indexed. The X-register and the memory location are not altered.

label	LBX	m [,I]	comments

Load B from memory indexed by X. The contents of the specified memory location are copied into the B-register. Indirect addressing may be used. The address of the memory location is computed by adding the contents of the X-register to **m** or to **m**,I. Note that indirect addressing (if specified) is performed first and then the address is indexed. The X-register and the memory location are not altered.

label	LAY	m [,I]	comments

Load A from memory indexed by Y. The contents of the specified memory location are copied into the A-register. Indirect addressing may be used. The address of the memory location is computed by adding the contents of the Y-register to **m** or to **m**,I. Note that indirect addressing (if specified) is performed first and then the address is indexed. The Y-register and the memory location are not altered.

label	LBY	m [,I]	comments

Load B from memory indexed by Y. The contents of the specified memory location are copied into the B-register. Indirect addressing may be used. The address of the memory location is computed by adding the contents of of the Y-register to **m** or to **m**,I. Note that indirect addressing (if specified) is performed first and then the address is indexed. The Y-register and the memory location are not altered.

label	SAX	m [,I]	comments

Store A into memory indexed by X. The contents of the A-register are copied into the specified memory location. Indirect addressing may be used. The address of the memory location is computed by adding the contents of the X-register to **m** or to **m**,I. Note that indirect addressing (if specified) is performed first and then the address is indexed. The A-register and the X-register are not altered.

label	SBX	m [,I]	comments

Store B into memory indexed by X. The contents of the B-register are copied into the specified memory location. Indirect addressing may be used. The address of the memory location is computed by adding the contents of the X-register to **m** or to **m**,I. Note that indirect addressing (if specified) is performed first and then the address is indexed. The B-register and the X-register are not altered.

label	SAY	m [,I]	comments

Store A into memory indexed by Y. The contents of the A-register are copied into the specified memory location. Indirect addressing may be used. The address of the memory location is computed by adding the contents of the Y-register to **m** or to **m**,I. Note that indirect addressing (if specified) is performed first and then the address is indexed. The A-register and the Y-register are not altered.

label	SBY	m [,I]	comments

Store B into memory indexed by Y. The contents of the B-register are copied into the specified memory location. Indirect addressing may be used. The address of the memory location is computed by adding the contents of the Y-register to **m** or to **m**,I. Note that indirect addressing (if specified) is performed first and then the address is indexed. The B-register and the Y-register are not altered.

label	ADX	m [,I]	comments

Add memory to X. The contents of the specified memory location are algebraically added to the contents of the X-register. Indirect addressing may be used. The memory location is not altered.

label	ADY	m [,I]	comments

Add memory to Y. The contents of the specified memory location are algebraically added to the contents of the Y-register. Indirect addressing may be used. The memory location is not altered.

label	JLY	m [,I]	comments

Jump and load Y. Control transfers unconditionally to the specified memory location and the address P+2 is loaded into the Y-register. Indirect addressing may be used. This instruction is used for calling subroutines. The subroutines use the Y-register to access parameters and to return control (by way of the JPY instruction) to the calling program.

label	JPY	m	comments

Jump indexed by Y. Control transfers unconditionally to the specified memory location. Indirect addressing may **not** be used. The address of the memory location is computed by adding the contents of the Y-register to **m**. This instruction is used for returning control from subroutines to the calling program (assuming that they were entered by way of JLY instructions).

3-12. 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.

label	NOP	comments

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 zero.

3-13. INPUT/OUTPUT, OVERFLOW, AND HALT

The input/output instructions allow the user to transfer 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_{10} 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.

Note: When Memory Protect is enabled, execution of all I/O instructions except those which reference the switch register (select code 01) or the overflow bit **is prohibited.**

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 select code (sc) may be a label which was previously defined as an external symbol by an EXT pseudo-instruction. In such a case, the entry point referred to by the EXT pseudo-instruction must be an absolute value less than 64_{10} (any other value will change the instruction). The character C appended to such an instruction clears the overflow bit after the transfer from the switch register is complete.

3-14. INPUT/OUTPUT

Prior to any input/output data transmission, the control bit must be set. The following instruction accomplishes this and 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 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 statemment 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.

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.

3-15. 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.

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.

3-16. 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.

3-17. EXTENDED ARITHMETIC UNIT (EAU)

If the computer on which the object program is to be run contains an EAU, this group of instructions may be used to increase the computer's overall efficiency.

The user specifies whether or not an EAU will be available via a parameter in the control statement (see paragraph 1-3). If an EAU will **not** be available, the instructions ASR, ASL, RRR, RRL, LSR, LSL, and SWP **cannot** be used in the source program (they will be flagged as errors) and the instructions MPY, DIV, DLD, and DST will result in calls to arithmetic subroutines (see paragraph 4-7).

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 following seven instructions provide the capability to shift or rotate the B- and A-Registers n number of bit positions to the right or left, where 1≤n≤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.

	SWP		

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.

3-18. FLOATING POINT

The instructions in this group are used for performing arithmetic operations on floating point operands. The user specifies whether or not floating point machine instructions are available via a parameter in the control statement (see paragraph 1-3). If the floating point machine instructions are **not** available, the instructions in this group result in calls to arithmetic subroutines (see paragraph 4-7). The Operand field may contain any relocatable expression or absolute expression resulting in a value of less than 2000_8.

label	FMP	m [,I] =Fn	comments

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] =Fn	comments

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] =FN	comments

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] =Fn	comments

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	FIX	comments

Convert the floating-point number contained in the A- and B-registers to a fixed-point number. The result is returned in the A-register. After the operation is completed, the contents of the B-register are meaningless.

label	FLT	comments

Convert the fixed-point number contained in the A-register to a floating-point number. The result is returned in the A- and B-registers.

PSEUDO INSTRUCTIONS

SECTION IV

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

4-1. 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, DOS-III and BCS loaders.

	NAM	name[,type][,link]	comments

NAM defines the name and type of a relocation program. **name** is the program's name. **type** defines the program type as follows:

3 = main program

4 = disc-resident driver

5 = program segment

6 = library routine

7 = subroutine

link specifies base page or current page linking by the loader. **link** = 0 specifies current page linking by the relocating loader. **link** ≠ 0 specifies base page linking. If **link** is not specified the Assembler sets it ≠ 0; i.e., base page linking is the default condition for the program.

If **type** equals any other number, the Assembler and DOS-III DSGEN will accept it but the DOS-III Relocating Loader will not. **name** must not be the same as an existing DOS-III file name. 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. A relocatable program must begin with a NAM statement.†

	ORG	m	comments

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

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 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	comment

ORR resets the program location counter to the value existing when ORG instruction was encountered. An example is shown in Figure 4-1.

More than one ORG 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 of the string occurred. An example is shown in Figure 4-2.

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

†The Control Statement, the HED instruction, and comments may appear prior to the NAM or ORG statements.

```
      NAM RSET       SET PLC TO VALUE OF ZERO, ASSIGN
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.
```

Figure 4-1. ORR Example (with Single ORG)

```
      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
```

Figure 4-2. ORR Example (with Multiple ORGs)

The IFN and IFZ pseudo instructions cause the inclusion of instructions in a program provided that either an "N" or "Z", respectively, is specified as a parameter for the ASMB control statement.† The IFN or IFZ instruction precedes the set of statements that are to be included. The pseudo instruction XIF serves as a terminator. If XIF is omitted, END acts as a terminator to both the set of statements and the assembly.

```
        | IFN | comments
           .
           .
           .
          XIF
```

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.

```
        | IFZ | comments
           .
           .
           .
          XIF
```

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 assembled. Some examples are shown in Figures 4-3 and 4-4.

In Figure 4-3, the 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.

In Figure 4-4, the 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.

```
      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
```

Figure 4-3. IFN/XIF and IFZ/XIF Example

```
      NAM WAGE
       .
       .
       .
      JSB HOUR
      MPY TIME1
      IFZ
      JSB OVTIM
      MPY TIME2
       .
       .
       .
TIME1 DEC 40
TIME2 BSS 1
      END
```

Figure 4-4. IFZ/XIF Example

†See "Assembly Options" in Section I of this manual.

The REP pseudo instruction 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 should not be part of the instruction to be repeated; it would result in a doubly defined symbol error.

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
         ADA    DATA    A-Register.
         ADA    DATA
```

Example:

```
FILL     REP    100B
         NOP
```

The example above loads 100_8 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 sóurce 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 when executing the program under BCS because the loader transfers control via a JSB. The DOS-III Relocating Loader transfers control via a JMP.

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.

4-2. 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. In the case of the COM pseudo instruction, the first space not preceded by a comma or a left parenthesis terminates the entire field.

	COM	name$_1$ [(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. An example is shown in Figure 4-5.

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

```
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
                      ADD+1.
```

Organization of common block:

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

Figure 4-5. COM Examples

The segment names that appear in the COM statements can be used in the Operand fields of DEF, ABS, EQU, ENT 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 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 an EXT statement in the same subprogram. Labels defined as absolute by EQU statements or defined by COM statements may be declared as entry points.

	EXT	name_1 [,name_2,...,name_n]	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, certain I/O statements or EQU or DEF pseudo instructions. An external symbol must appear alone; it may not be in a multiple term expression or be specified as indirect. References to external locations are processed by the 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. Examples of the use of EXT and ENT are shown in Figure 4-6.

```
PROGA  NOP
       LDA  SAMD       SAMD AND SAND ARE REFERENCED IN
        .              PROGA, BUT ARE ACTUALLY
        .              LOCATIONS IN PROGB.
        .
       JMP  SAND
       EXT  SAMD,SAND
       ENT  PROGA
       END
        .
        .
PROGB  NOP
        .
        .
SAMD   OCT  767
SAND   STA  SAMD
        .
        .
       ENT  SAMD,SAND
        .
        .
       JSB  PROGA
        .
        .
       EXT  PROGA
        .
        .
       END
```

Figure 4-6. ENT/EXT Examples

4-3. ADDRESS AND SYMBOL DEFINITION

The pseudo operations in this group assign a value, a word address, or a byte address 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 2000_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. Some examples are shown in Figure 4-7.

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.** Figure 4-8 illustrates what **not** to do. 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 link to the base page rather than the value of TBL. Assuming that the loader assigns the absolute locations shown in Figure 4-9, the ISZ will index the contents of location 2000_8 which is a LDA 700,I, and change it to LDA 701,I. Now we will use whatever happens to be in 701 rather than the link we intended to use which is in 700. We change the **link** instead of its **contents**.

```
LDTBL  LDA TBL
        .
        .
        .
       ISZ LDTBL
        .
        .
        .
TBL    BSS 100
```

Figure 4-8. Example of Incorrect Address Modification

```
      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
```

Figure 4-7. DEF Examples

INSTRUCTION			PAGE	ABSOLUTE LOCATION OF CODE	OPCODE	OPERAND (PAGE) & LOCATION	
(Loader-assigned indirect link on base page)			(0)	(700)	DEF		4000
					• •		
LDTBL	LDA	TBL	(1)	(2000)	LDA	(0)	700 (I)
					• •		
	ISZ	LDTBL	(1)	(3000)	ISZ	(1)	2000
					• •		
TBL	BSS	100	(2)	(4000)	BSS		

Figure 4-9. Loader-Assigned Locations for Figure 4-8

The example shown in Figure 4-10 assures correct address modification during program execution. Assume that the sequence shown in Figure 4-10 is assigned (by the loader) the absolute locations shown in Figure 4-11. The LDA 2000,I picks up the **contents** of the location pointed to by ITBL (location 4000_8). The ISZ 2000 indexes the pointer DEF 4000 to point to 4001. The next LDA will reference location 4001, DEF TBL+1. This is what we intend.

```
ITBL   DEF TBL
LDTBL  LDA ITBL,I
        .
        .
        .
       ISZ ITBL
        .
        .
        .
TBL    BSS 100
```

Figure 4-10. Example of Correct Address Modification

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. Examples are shown in Figure 4-12.

INSTRUCTION			PAGE	ABSOLUTE LOCATION OF CODE	OPCODE	OPERAND (PAGE) & LOCATION	
ITBL	DEF	TBL	(1)	(2000)	DEF		4000
	LDA	ITBL,I	(1)	(2001)	LDA	(1)	2000,I
					• •		
	ISZ	ITBL	(1)	(3000)	ISZ	(1)	2000
					• •		
	TBL	BSS 100	(2)	4000	BSS		

Figure 4-11. Loader-Assigned Locations for Figure 4-10

```
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.
```

Figure 4-12. ABS Examples

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 should 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. Some examples are shown in Figures 4-13 and 4-14.

label	DBL	m	comments

label	DBR	m	comments

Define Left Byte and Define Right Byte (21MX only). The DBL and DBR pseudo instructions each generate one word of memory which contains a 16-bit byte address. For DBL, the byte address being defined is the left half (bits 8-15) of word location **m**; for DBR, it is the right half (bits 0-7). Indirect addressing may **not** be used. A **byte address** is defined as two times the word address of the memory location containing the particular byte. If the byte location is the left half of the memory location (bits 8-15), bit 0 of the byte address is clear; if the byte location is the right half of the memory location (bits 0-7), bit 0 of the byte address is set. In an absolute program, **m** may be any positive expression. In a relocatable program, **m** may be any absolute expression with a value less than 100 or any relocatable expression. The generated word may be referenced (via **label**) in the Operand field of LDA and LDB instructions elsewhere in the source program for the purpose of loading byte addresses into the A- and B-registers.

CAUTION

Care must be taken when using the **label** of a DBL or DBR pseudo instruction as an indirect address elsewhere in the source program. The programmer must keep track of whether he is using word addresses or byte addresses.

```
       NAM FAM
        .
        .
        .
J3     DEF
        .
        .
        .
       LDA J3
       ADA ONE
       STA J3+1
JFOUR  EQU J3+1      THE SYMBOLS JFOUR AND J3+1 BOTH IDENTIFY
        .            THE SAME LOCATION. THE "AND" OPERATION
        .            IS PERFORMED ON THIS LOCATION.
        .
MWH    AND JFOUR
        .
```

Figure 4-13. EQU Example

```
          NAM STOTB
           .
           .
           .
          COM TABLA(10) DEFINES A 10 WORD TABLE, TABLA.
           .
           .
           .
TABLB     EQU TABLA+5     NAMES WORDS 6 THROUGH 10 OF
           .              TABLA AS TABLB.
           .
           .
           .
          LDA TABLB+1     LOADS CONTENTS OF 7TH WORD
           .              COMMON INTO A. THE STATEMENT LDA
           .              TABLA+6 WOULD PERFORM THE SAME
           .              OPERATION
           .
          NAM REG
           .
           .
           .
A         EQU 0           DEFINES SYMBOL A AS 0 (LOCATION
B         EQU 1           OF A-REGISTER), AND SYMBOL B AS
           .              1 (LOCATION OF B-REGISTER).
           .
          LDA B           LOADS CONTENTS OF B-REGISTER
                          INTO A-REGISTER.
```

Figure 4-14. EQU Examples

Examples:

```
BYT1     DBL   WORD1
BYT2     DBR   WORD1
                .
                .
                .
WORD1    NOP
```

If WORD1 has the relocatable address 2002_8, then BYT1 will contain the relocatable value 4004_8 and BYT2 will contain the relocatable value 4005_8.

4-4. 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 prefiously 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. An example is shown in Figure 4-15.

label	DEC	d_1 [.d_2,...,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

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

```
TTYP   ASC 3,ABCDE
```

causes the following:

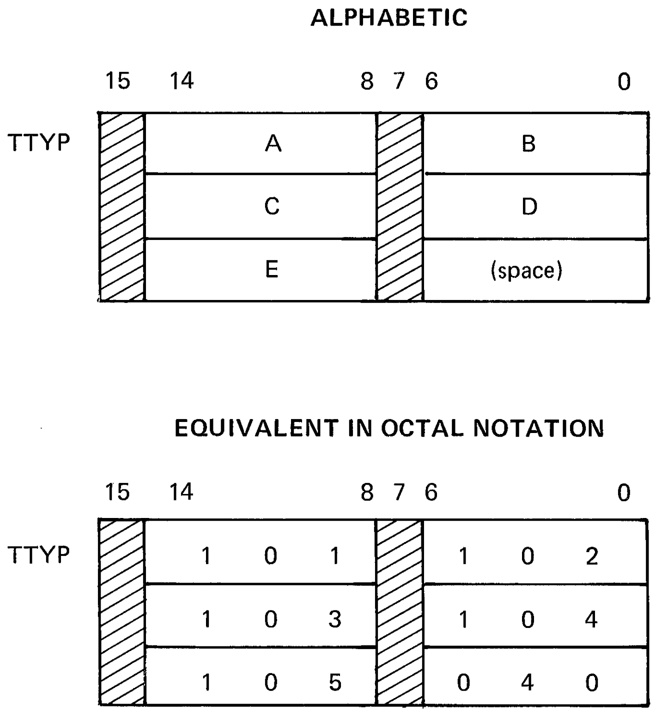
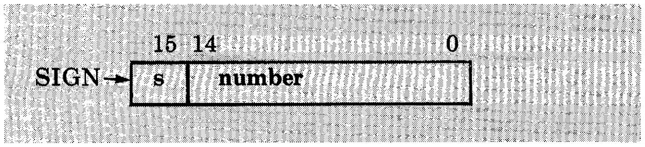


Figure 4-15. ASC Example

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}; it may assume positive, negative, or zero values. It is converted into one binary word and appears as follows:

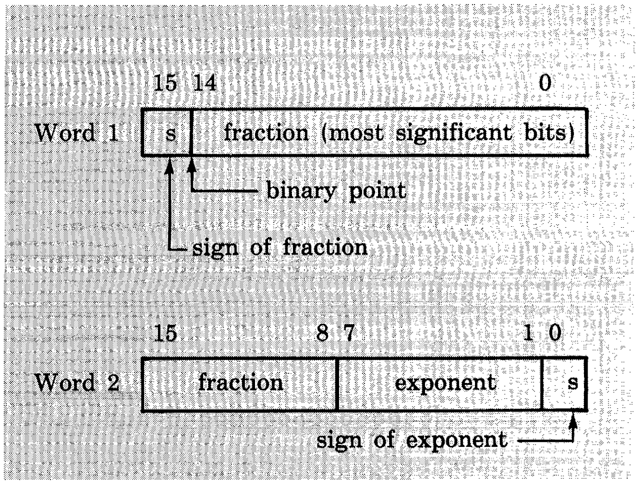


Some examples are shown in Figure 4-16.

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:

±n.n ±n. ±n.nE±e ±.nE±e ±n.E+e ±nE±e

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.



```
INT     DEC 50,+328,-300,+32768,-32768
```

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
	1	0	0	0	0	0
	1	0	0	0	0	0

Note: The values ±2^{15} (±32768) are both converted to 100000_8.

Figure 4-16. DEC Examples (Integer)

```
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:

15	14 … 0
0	1 0 0 1 0 0 0 0 0 0 0 0 0 0 0

15 … 8	7 … 1	0
0 0 0 0 0 0 0 0	0 0 0 0 0 1 1	0

Figure 4-17. DEC Examples (Floating Point)

```
DEC -.695,400E-4
```

are stored as:

1	0 1 0 0 1 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

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 1 0 0	1

Figure 4-18. DEC Examples (Floating Point)

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 are shown in Figures 4-17 and 4-18.

label	DEX	d_1 [$,d_2,...,d_n$]	comments

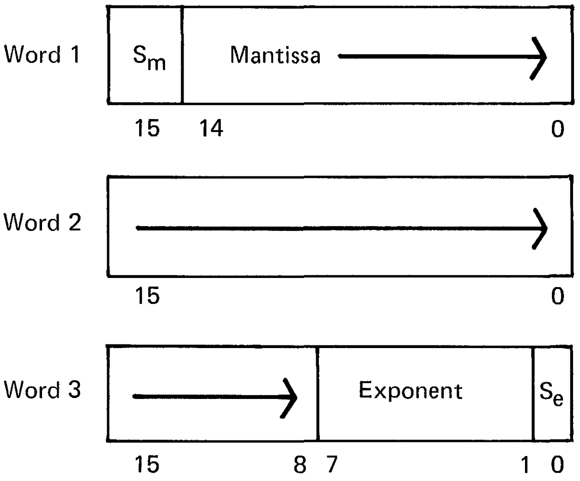
DEX records a string of extended precision decimal constants into consecutive words within a program. Each such extended precision constant occupies three words as shown in Figure 4-19.

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. Some examples are shown in Figure 4-20.

label	OCT	o_1 [$,o_2,...,o_n$]	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 number is right justified in the word. A label, if used, acts as the address of the first 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. Some examples are shown in Figure 4-21.



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.

Figure 4-19. DEX Memory Format

DEX 12,-.45

are stored as:

WORD 1	WORD 2	WORD 3
0110000000000000	0000000000000000	0000000000001000

WORD 1	WORD 2	WORD 3
1000110011001100	1100110011001100	1001101111111111

Figure 4-20. DEX Examples

```
        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 above statements are stored as follows:

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

The result of attempting to define an illegal constant is unpredictable

Figure 4-21. OCT Examples

label	BYT	$b_1,b_2, \ldots b_n$	comments

Define Octal Byte Constants (21MX only). The BYT pseudo instruction generates octal constants in consecutive byte locations of memory. Each constant in the Operand field ($b_1,b_2, \ldots b_n$) consists of one to three octal digits, must be within the range 0 through 377, and may be preceded by a plus (+) or minus (-) sign. If a constant is not signed, it is assumed to be positive. If a constant is negative, the two's complement of the binary equivalent (truncated to eight bits) is stored. If the Operand field contains an odd number of constants, bits 0-7 of the final word generated will be clear (zeros). Since the constants are assumed to be octal, the letter "B" must not be used. Some examples are shown in Figure 4-22.

4-5. 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.

4-6. ASSEMBLY LISTING CONTROL

Assembly listing control pseudo instructions allow the user to control the assembly listing Output during pass 2 of the assembly process.

	UNL	comments

List 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.

	LST	comments

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

A UNL following a UNL, an LST following an LST, and an 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 machine and pseudo instructions generate more than one line of coding. These additional code lines are

```
ALF    BYT 50,377,-10,2,-312
```

causes the following (octal representation):

	15	14				0
ALF	0	2	4	3	7	7
	1	7	4	0	0	2
	0	3	3	0	0	0

Figure 4-22. BYT Examples

suppressed by an SUP instruction until a UNS or the END pseudo instruction is encountered. SUP will suppress additional code lines in the following machine and pseudo instructions:

ADX
ADY
ASC
BYT
CBS
CBT
CMW
DEC
DIV
DJP
DJS
DLD
DST
FAD
FDV
FMP
FSB
JLY
JPY
LAX
LAY
LBX
LBY
LDX
LDY
MBT
MCA
MCB
MDB
MLA
MLB
MPY
MSA
MSB
MVW
OCT
SAX
SAY
SBS
SBX
SBY
SJP
SJS
STX
STY
TBS
UJP
UJS
XMM
XMS

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 an SUP, to be resumed.

An SUP preceded by another SUP, UNS preceded by UNS, or UNS not preceded by an SUP are not considered errors by the Assembler.

```
        SKP     comments
```

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.

```
        SPC     n
```

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.

```
        HED     m(heading)
```

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 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.

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

4-7. ARITHMETIC SUBROUTINE CALLS

If an X appears in the control statement for the source program, the Assembler generates calls to arithmetic subroutines external to the source program for the following instructions: MPY, DIV, DLD, and DST. The instruction formats and functions are as described in paragraph 3-17 of Section III in this manual.

If an F **does not** appear in the control statement for the source program, the Assembler generates calls to arithmetic subroutines external to the source program for the following instructions: FMP, FDV, FAD, FSB, FIX, and FLT. The instruction formats and functions are as described in paragraph 3-18 of Section III in this manual.

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 alter the contents of the E-Register.

4-8. DEFINE USER INSTRUCTION (21MX ONLY)

	MIC	opcode,fcode,pnum	comments

This pseudo instruction provides the user the capability of defining his own instructions. **opcode** is a three-character alphabetic mnemonic, **fcode** is an instruction code, and **pnum** declares how many (0-7) parameter addresses are to be associated with the newly-defined instruction. Both **fcode** and **pnum** may be expressions which generate an absolute result. A user-defined instruction must not appear in the source program prior to the MIC pseudo instruction which defines it. When the user-defined mnemonic is used later in the source program, the specified number of parameter addresses (**pnum**) are supplied in the Operand field of the user-defined instruction separated from one another by spaces. The parameter addresses may be any addressable values, relocatable and/or indirect.

Note: All three operands (**opcode, fcode,** and **pnum**) must be supplied in the MIC pseudo instruction in order for the specified instruction to be defined. If **pnum** is zero, it must be expressly declared as such (**not** omitted).

4-9. "JUMP TO MICROPROGRAM"

The MIC pseudo instruction is primarily intended to facilitate the passing of control from an assembly language program to a user's microprogram residing in Read-Only-Memory (ROM) or Writable Control Store (WCS). Ordinarily, to do this the user must include an OCT 101**xxx** or OCT 105**xxx** statement (where **xxx** is 140 through 737) at the point in the source program where the jump is to occur. If parameters are to be passed, they are usually defined as constants (via OCT or DEF statements) immediately following the OCT 105**xxx** statement. With the MIC pseudo instruction, the user can define a source language instruction which passes control and a series of parameter addresses to a microprogram. If it is desired to pass additional parameters to a microprogram beyond those pointed to by the user-defined instruction, they must be defined as constants (via OCT or DEF statements) immediately following each use of the user-defined instruction.

4-10. EXAMPLE

Assume that the first two parameters to be passed from the assembly language program to the user's microprogram reside in the memory locations PARM1 and PARM2 and that the third parameter resides in the memory location pointed to by ADR,I. Also assume that the octal code for transferring control to the particular microprogram is 105240_8.

The following statement defines a source language instruction which passes control and three parameter addresses to the microprogram:

```
 MIC  ABC,105240B,3
```

Whenever it is desired to pass control from the assembly language program to the microprogram, the following user-defined instruction may be used in the source program:

```
 ABC  PARAM1 PARAM2 ADR,I
```

4-11. COMBINING MULTIPLE MNEMONICS

Another use of the MIC pseudo instruction is to assign a single mnemonic to a multiple instruction (shift-rotate or alter-skip) statement.

4-12. EXAMPLE

Instead of using the source statement:

```
 ALR,CLE,SLA,RAL
```

the user may define a single mnemonic as follows:

```
 MIC  XYZ,01472B,0
```

where 01472B is the octal instruction code for the four-mnemonic statement shown above. Whenever XYZ is subsequently used as an instruction mnemonic in the source program, it is the equivalent of using the source statement:

```
 ALR,CLE,SLA,RAL
```

4-13. DEFINING CONSTANTS

The MIC pseudo instruction may also be used for defining constants (**opcode** = mnemonic, **fcode** = constant, and **pnum** = 0). Whenever the defined mnemonic is used as an instruction mnemonic in the source program the

Assembler automatically replaces it with the specified constant.

4-14. EXAMPLE

The following statement defines the constant 10_{10} and assigns it the mnemonic TEN:

```
MIC  TEN,10,0
```

Whenever TEN appears as an instruction mnemonic later in the source program, the value 10_{10} is automatically inserted in that location by the Assembler.

HP CHARACTER SET

APPENDIX A

A-1. ASCII CHARACTER FORMAT

b7					0	0	0	0	1	1	1	1
b6					0	0	1	1	0	0	1	1
b5					0	1	0	1	0	1	0	1
BITS b4 ↓	b3 ↓	b2 ↓	b1 ↓	COLUMN → ROW ↓	0	1	2	3	4	5	6	7
0	0	0	0	0	NUL	DLE	SP	0	@	P	`	p
0	0	0	1	1	SOH	DC1	!	1	A	Q	a	q
0	0	1	0	2	STX	DC2	"	2	B	R	b	r
0	0	1	1	3	ETX	DC3	#	3	C	S	c	s
0	1	0	0	4	EOT	DC4	$	4	D	T	d	t
0	1	0	1	5	ENQ	NAK	%	5	E	U	e	u
0	1	1	0	6	ACK	SYN	&	6	F	V	f	v
0	1	1	1	7	BEL	ETB	'	7	G	W	g	w
1	0	0	0	8	BS	CAN	(	8	H	X	h	x
1	0	0	1	9	HT	EM	)	9	I	Y	i	y
1	0	1	0	10	LF	SUB	*	:	J	Z	j	z
1	0	1	1	11	VT	ESC	+	;	K	[	k	{
1	1	0	0	12	FF	FS	,	<	L	\	l	¦
1	1	0	1	13	CR	GS	-	=	M	]	m	}
1	1	1	0	14	SO	RS	.	>	N	↑	n	~
1	1	1	1	15	SI	US	/	?	O	←	o	DEL

Figure A-1. ASCII Characters and Binary Codes

Standard 7-bit set code positional order and notation are shown below with b_7 the high-order and b_1 the low-order, bit position.

Example: The code for “R” is: $\overset{b_7}{1}\ \overset{b_6}{0}\ \overset{b_5}{1}\ \overset{b_4}{0}\ \overset{b_3}{0}\ \overset{b_2}{1}\ \overset{b_1}{0}$

Table A-1. Legend for Figure A-1

NUL	Null	DC1	Device Control 1
SOH	Start of Heading	DC2	Device Control 2
STX	Start of Text	DC3	Device Control 3
ETX	End of Text	DC4	Device Control 4
EOT	End of Transmission	NAK	Negative Acknowledgement
ENQ	Enquiry	SYN	Synchronous Idle
ACK	Positive Acknowledgement	ETB	End of Transmission Block
BEL	Bell (Audible signal)	CAN	Cancel
BS	Backspace	EM	End of Medium
HT	Horizontal Tabulation	SUB	Substitute
LF	Line Feed	ESC	Escape
VT	Vertical Tabulation	FS	File Separator
FF	Form Feed	GS	Group Separator
CR	Carriage Return	RS	Record Separator
SO	Shift Out	US	Unit Separator
SI	Shift In	SP	Space
DLE	Data Link Escape	DEL	Delete

A-2. BINARY CODED DECIMAL (BCD) FORMAT

Table A-2. HP 7970B BCD-ASCII Conversion

SYMBOL	BCD (OCTAL CODE)	ASCII EQUIVALENT (OCTAL CODE)	SYMBOL	BCD (OCTAL CODE)	ASCII EQUIVALENT (OCTAL CODE)
(space)	20	040	@	14	100
!	52	041	A	61	101
"	37	042	B	62	102
#	13	043	C	63	103
$	53	044	D	64	104
%	57	045	E	65	105
&	†	046	F	66	106
'	35	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
=	17	075	]	55	135
>	16	076	↑	77	136
?	72	077	←	32	137

†The ASCII code 046 is converted to the BCD code for a space (20) when writing data onto a 7-track tape.

APPENDIX B

SUMMARY OF INSTRUCTIONS

Symbols	Meaning
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
$\wedge$	Logical product
$\veebar$	Exclusive "or"
$\vee$	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
$(\overline{A/B})$	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
msb	Most significant bits
lsb	Least significant bits

B-1. MACHINE INSTRUCTIONS

B-2. MEMORY REFERENCE

B-3. Jump and Increment-Skip

ISZ	m [,I]	(m) +→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

B-4. 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

B-5. Logical

AND	$\left\{\begin{matrix} m\ [,I] \\ lit \end{matrix}\right\}$	$(m) \wedge (A) \rightarrow A$
XOR	$\left\{\begin{matrix} m\ [,I] \\ lit \end{matrix}\right\}$	$(m) \veebar (A) \rightarrow A$
IOR	$\left\{\begin{matrix} m\ [,I] \\ lit \end{matrix}\right\}$	$(m) \vee (A) \rightarrow A$
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 P + 1

B-6. Word Processing

MVW	$\left\{\begin{matrix} m\ [,I] \\ lit \end{matrix}\right\}$	Move (m) words from array (A)→array (B)
CMW	$\left\{\begin{matrix} m\ [,I] \\ lit \end{matrix}\right\}$	Compare (m) words of array (A) against (m) words of array (B); if the two arrays are equal, execute P + 3, if array (A) is less than array (B), execute P + 4, if array (A) is greater than array (B), execute P + 5

B-7. Byte Processing

LBT		B contains a 16-bit byte address; ((B))→A_{0-7}; o's to A_{8-15}
SBT		B contains a 16-bit byte address; (A_{0-7})→(B)
MBT	$\left\{\begin{matrix} \text{m [,I]} \\ \text{lit} \end{matrix}\right\}$	A and B contain 16-bit byte addresses; move (m) bytes from array (A)→array (B)
CBT	$\left\{\begin{matrix} \text{m [,I]} \\ \text{lit} \end{matrix}\right\}$	A and B contain 16-bit byte addresses; compare (m) bytes of array (A) against (m) bytes of array (B); if the two arrays are equal, execute P + 3; if array (A) is less than array (B), execute P + 4; if array (A) is greater than array (B), execute P + 5
SFB		A_{0-7} contain the test byte, A_{8-15} contain the termination byte, and B contains a 16-bit byte address; scan array (B); if test byte found, execute P + 1, B contains address of test byte; if termination byte found, execute P + 2, B contains address of termination byte; if neither is found, execute P + 2, B contains zero

B-8. Bit Processing

TBS	$\left\{\begin{matrix} \text{m [,I]} \\ \text{lit} \end{matrix}\right\}$	n [,I]	Compare all "set" bits in (m) against corresponding bits in (n); if all bits tested are set, execute P + 3; if any of the bits tested are clear, execute P + 4
SBS	$\left\{\begin{matrix} \text{m [,I]} \\ \text{lit} \end{matrix}\right\}$	n [,I]	Set all bits in (n) which correspond to "set" bits in (m)
CBS	$\left\{\begin{matrix} \text{m [,I]} \\ \text{lit} \end{matrix}\right\}$	n [,I]	Clear all bits in (n) which correspond to "set" bits in (m)

B-9. REGISTER REFERENCE

B-10. Shift-Rotate

CLE	0→E
ALS	Shift (A) left one bit, 0→A_0, A_{15} unaltered
BLS	Shift (B) left one bit, 0→B_0, B_{15} unaltered
ARS	Shift (A) right one bit, (A_{15})→A_{14}
BRS	Shift (B) right one bit, (B_{15})→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→A_{15}
BLR	Shift (B) left one bit, 0→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

Shift-Rotate instructions can be combined as follows:

$$\left[\left\{\begin{matrix}\text{ALS}\\\text{ARS}\\\text{RAL}\\\text{RAR}\\\text{ALR}\\\text{ALF}\\\text{ERA}\\\text{ELA}\end{matrix}\right\}\right]\quad[\text{,CLE}]\quad[\text{,SLA}]\quad\left[,\left\{\begin{matrix}\text{ALS}\\\text{ARS}\\\text{RAL}\\\text{RAR}\\\text{ALR}\\\text{ALF}\\\text{ERA}\\\text{ELA}\end{matrix}\right\}\right]$$

$$\left[\left\{\begin{matrix}\text{BLS}\\\text{BRS}\\\text{RBL}\\\text{RBR}\\\text{BLR}\\\text{BLF}\\\text{ERB}\\\text{ELB}\end{matrix}\right\}\right]\quad[\text{,CLE}]\quad[\text{,SLB}]\quad\left[,\left\{\begin{matrix}\text{BLS}\\\text{BRS}\\\text{RBL}\\\text{RBR}\\\text{BLR}\\\text{BLF}\\\text{ERB}\\\text{ELB}\end{matrix}\right\}\right]$$

B-11. No-Operation

NOP	Execute P + 1

B-12. 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
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:

$$\left[\left\{\begin{matrix}\text{CLA}\\ \text{CMA}\\ \text{CCA}\end{matrix}\right\}\right] \quad [\text{,SEZ}] \quad \left[,\left\{\begin{matrix}\text{CLE}\\ \text{CME}\\ \text{CCE}\end{matrix}\right\}\right] \quad [\text{,SSA}]\ [\text{,SLA}]\ [\text{,INA}]\ [\text{,SZA}]\ [\text{,RRS}]$$

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

B-13. Index Register

CAX		(A)→X
CBX		(B)→X
CAY		(A)→Y
CBY		(B)→Y
CXA		(X)→A
CXB		(X)→B
CYA		(Y)→A
CYB		(Y)→B
XAX		(A)→X and (X)→A
XBX		(B)→X and (X)→B
XAY		(A)→Y and (Y)→A
XBY		(B)→Y and (Y)→B
ISX		(X) + 1→X, then test new (X); if (X) = 0, execute P + 2, otherwise execute P + 1
ISY		(Y) + 1→Y, then test new (Y); if (Y) = 0, execute P + 2, otherwise execute P + 1
DSX		(X) - 1→X, then test new (X); if (X) = 0, execute P + 2, otherwise execute P + 1
DSY		(Y) - 1→Y, then test new (Y); if (Y) = 0, execute P + 2, otherwise execute P + 1
LDX	m [,I] lit	(m)→X
LDY	m [,I] lit	(m)→Y
STX	m [,I]	(X)→m
STY	m [,I]	(Y)→m
LAX	m [,I]	(m + (X))→A
LBX	m [,I]	(m + (X))→B
LAY	m [,I]	(m + (Y))→A
LBY	m [,I]	(m + (Y))→B
SAX	m [,I]	(A)→m + (X)
SBX	m [,I]	(B)→m + (X)
SAY	m [,I]	(A)→m + (Y)
SBY	m [,I]	(B)→m + (Y)
ADX	m [,I]	(m) + (X)→X
ADY	m [,I]	(m) + (Y)→Y
JLY	m [,I]	Jump to m; P + 2→Y
JPY	m	Jump to m + (Y)

B-14. INPUT/OUTPUT, OVERFLOW, AND HALT

B-15. 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$_{sc}$
STF	sc	Set flag bit$_{sc}$. If sc = 0, enable interrupt system. sc = 1 sets overflow bit.
CLF	sc	Clear flag bit$_{sc}$. If sc = 0, disable interrupt system. If sc = 1, clear overflow bit.
SFC	sc	If (flag bit$_{sc}$) = 0, execute P + 2, otherwise execute P + 1. If sc = 1, test overflow bit.
SFS	sc	If (flag bit$_{sc}$) = 1, execute P + 2, otherwise execute P + 1. If sc = 1, test overflow bit.

B-16. Overflow

CLO		0→overflow bit
STO		1→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

B-17. Halt

HLT	[sc [,C]]	Halt computer

B-18. EXTENDED ARITHMETIC UNIT

MPY	$\left\{\begin{matrix} \text{m [,I]} \\ \text{lit} \end{matrix}\right\}$	(A) x (m)→(B$_{\pm msb}$ and A$_{lsb}$)
DIV	$\left\{\begin{matrix} \text{m [,I]} \\ \text{lit} \end{matrix}\right\}$	(B$_{\pm msb}$ and A$_{lsb}$)/(m)→A, remainder→B
DLD	$\left\{\begin{matrix} \text{m [,I]} \\ \text{lit} \end{matrix}\right\}$	(m) and (m + 1)→A and B
DST	$\left\{\begin{matrix} \text{m [,I]} \\ \text{lit} \end{matrix}\right\}$	(A) and (B)→m 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$_{lsb}$

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, o's to A_{lsb}

B-19. FLOATING POINT

FMP	$\left\{\begin{matrix} m\ [,I] \\ lit \end{matrix}\right\}$	(AB) x (m, m + 1) → AB
FDV	$\left\{\begin{matrix} m\ [,I] \\ lit \end{matrix}\right\}$	(AB)/(m, m + 1) → AB
FAD	$\left\{\begin{matrix} m\ [,I] \\ lit \end{matrix}\right\}$	(m, m + 1) + (AB) → AB
FSB	$\left\{\begin{matrix} m\ [,I] \\ lit \end{matrix}\right\}$	(AB) - (m, m + 1) → AB
FIX		(AB) converted from floating-point to fixed-point; result → A
FLT		(A) converted from fixed-point to floating-point; result → AB

B-20. PSEUDO INSTRUCTIONS

B-21. 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.
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.

B-22. 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.

B-23. 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.

label DBL m Defines a 16-bit byte address (left half, bits 8-15, of word location **m**) to be referenced by the label.

label DBR m Defines a 16-bit byte address to be referenced by the label. The byte address is for the right half (bits 0-7) of word location m.

B-24. CONSTANT DEFINITION

ASC n, <2n characters> Generates a string of 2n ASCII characters.

DEC d$_1$ [,d$_2$,...,d$_n$] Records a string of decimal constants of the form:

Integer: ±n
Floating point: ±n.n, ±n., ±.n, ±nE±e, ±n.nE±e, ±n.E±e, ±.nE±e

DEX d$_1$ [,d$_2$,...,d$_n$] Records a string of extended precision decimals constants of the form

Floating point: ±n, ±n.m, ±n., ±.n,
±nE±e, ±n.nE±e, ±n.E±e, ±.nE±e

OCT o$_1$ [,o$_2$,...,o$_n$] Records a string of octal constants of the form: ±000000

BYT b [,b ,...,b$_n$] Records a string of octal byte constants of the form: ±nnn (where **nnn** is 0 through 377_8).

B-25. STORAGE ALLOCATION

BSS m Reserves a storage area of length, m.

B-26. 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.

B-27. DEFINE USER INSTRUCTION

MIC opcode,fcode,pnum Defines a source language instruction. **opcode** = three-character alphabetic mnemonic, **fcode** = instruction code, and **pnum** declares how many parameter addresses are to be associated with the newly-defined instruction.

ALPHABETIC LIST OF INSTRUCTIONS

APPENDIX C

ABS	Define absolute value
ADA	Add to A
ADB	Add to B
ADX	Add memory to X
ADY	Add memory to Y
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
BYT	Defines octal byte constants
CAX	Copy A to X
CAY	Copy A to Y
CBS	Clear bits
CBT	Compare bytes
CBX	Copy B to X
CBY	Copy B to Y
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
CMW	Compare words
COM	Reserve block of common storage
CPA	Compare to A, skip if unequal
CPB	Compare to B, skip if unequal
CXA	Copy X to A
CXB	Copy X to B
CYA	Copy Y to A
CYB	Copy Y to B
DBL	Defines left byte (bits 8-15) address
DBR	Defines right byte (bits 0-7) address
DEC	Defines decimal constants
DEF	Defines address
DEX	Defines extended precision constants
DIV	Divide
DLD	Double load
DST	Double store
DSX	Decrement X and skip if zero
DSY	Decrement Y and skip if zero
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 appears in Control Statement, assemble ensuing instructions
INA	Increment A by 1
INB	Increment B by 1
IOR	Inclusive "or" to A

ISX	Increment X and skip if zero
ISY	Increment Y and skip if zero
ISZ	Increment, then skip if zero
JLY	Jump and load Y
JMP	Jump
JPY	Jump indexed by Y
JSB	Jump to subroutine
LAX	Load A from memory indexed by X
LAY	Load A from memory indexed by Y
LBT	Load byte
LBX	Load B from memory indexed by X
LBY	Load B from memory indexed by Y
LDA	Load into A
LDB	Load into B
LDX	Load X from memory
LDY	Load Y from memory
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)
MBT	Move bytes
MIA	Merge (or) into A from I/O channel
MIB	Merge (or) into B from I/O channel
MIC	Defines jump to user microcode
MPY	Multiply
MVW	Move words
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
SAX	Store A into memory indexed by X
SAY	Store A into memory indexed by Y
SBS	Set bits
SBT	Store byte
SBX	Store B into memory indexed by X
SBY	Store B into memory indexed by Y
SEZ	Skip if E = 0
SFB	Scan for byte
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)
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
STX	Store X into memory
STY	Store Y into memory
SUP	Suppress list output of additional code lines
SWP	Switch the (A) and (B)
SZA	Skip if A = 0
SZB	Skip if B = 0
TBS	Test bits
UNL	Suppress list output
UNS	Resume list output of additional code lines
XAX	Exchange A and X
XAY	Exchange A and Y
XBX	Exchange B and X
XBY	Exchange B and Y
XIF	Terminate an IFN or IFZ group of instructions
XOR	Exclusive "or" to A

CONSOLIDATED CODING SHEETS

APPENDIX D

Table D-1 presents the binary codes for the base set instructions while Table D-2 presents those for the extended instruction group.

Table D-1. Base Set Instruction Codes in Binary

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	
				A/B	0	D/E	*RS	001			D/E		*RS	001	
				A/B	0	D/E	R*L	010			D/E		R*L	010	
				A/B	0	D/E	R*R	011			D/E		R*R	011	
				A/B	0	D/E	*LR	100			D/E		*LR	100	
				A/B	0	D/E	ER*	101			D/E		ER*	101	
				A/B	0	D/E	EL*	110			D/E		EL*	110	
				A/B	0	D/E	*LF	111			D/E		*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*	IN*	SZ*	RSS
				A/B		CM*	10	CME	10						
				A/B		CC*	11	CCE	11						
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	IOG	000			1	H/C	HLT	000		← Select Code →					
					1	0	STF	001							
					1	1	CLF	001							
					1	0	SFC	010							
					1	0	SFS	011							
				A/B	1	H/C	MI*	100							
				A/B	1	H/C	LI*	101							
				A/B	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	EAG	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			← number of bits →	
				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, according to bit 11.
D/I, A/B, Z/C, D/E, H/C coded: 0/1.
**Second word is Memory Address.

†CLE: Only this bit is required.
‡SL*: Only this bit and bit 11 (A/B as applicable) are required.

Table D-2. Extended Instruction Group Codes in Binary

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SAX/SAY/SBX/SBY	1	0	0	0	A/B	0	1	1	1	1	1	0	X/Y	0	0	0
CAX/CAY/CBX/CBY	1	0	0	0	A/B	0	1	1	1	1	1	0	X/Y	0	0	1
LAX/LAY/LBX/LBY	1	0	0	0	A/B	0	1	1	1	1	1	0	X/Y	0	1	0
STX/STY	1	0	0	0	1	0	1	1	1	1	1	0	X/Y	0	1	1
CXA/CYA/CXB/CYB	1	0	0	0	A/B	0	1	1	1	1	1	0	X/Y	1	0	0
LDX/LDY	1	0	0	0	1	0	1	1	1	1	1	0	X/Y	1	0	1
ADX/ADY	1	0	0	0	1	0	1	1	1	1	1	0	X/Y	1	1	0
XAX/XAY/XBX/XBY	1	0	0	0	A/B	0	1	1	1	1	1	0	X/Y	1	1	1
ISX/ISY/DSX/DSY	1	0	0	0	1	0	1	1	1	1	1	1	X/Y	0	0	I/D
JUMP INSTRUCTIONS	1	0	0	0	1	0	1	1	1	1	1	1	▨	0	1	0
BYTE INSTRUCTIONS	1	0	0	0	1	0	1	1	1	1	1	1	0	▨		
BIT INSTRUCTIONS	1	0	0	0	1	0	1	1	1	1	1	1	1	▨		
WORD INSTRUCTIONS	1	0	0	0	1	0	1	1	1	1	1	1	1	1	1	▨

JUMP INSTRUCTIONS:

JLY	=	0
JPY	=	1

BYTE INSTRUCTIONS:

LBT	=	0	1	1
SBT	=	1	0	0
MBT	=	1	0	1
CBT	=	1	1	0
SFB	=	1	1	1

BIT INSTRUCTIONS:

SBS	=	0	1	1
CBS	=	1	0	0
TBS	=	1	0	1

WORD INSTRUCTIONS:

CMW	=	0
MVW	=	1

RUNNING ASSEMBLIES UNDER DOS-III

APPENDIX

E

The Assembler, a segmented program that executes in the main-memory User Program Area, operates under control of DOS-III. The Assembler consists of a main program (ASMB) and six segments (ASMBD, ASMB1, ASMB2, ASMB3, ASMB4, ASMB5), and resides on the disc. The main program is read into main memory when called by a PROG directive.

Source programs, accepted from either an input device or a user source file (or files) on the disc, are translated into absolute or relocatable object programs; absolute code is punched in binary records, suitable for execution only outside of DOS-III. ASMB can store relocatable code in the Job Binary Area of the disc for on-line execution, as well as punch it on paper tape.

A source program passes through the input device only once, unless there is insufficient disc storage space. In the latter case, DOS-III informs the user that two passes are required.

E-1. ASSEMBLER I/O

The Assembly Language I/O EXEC calls should specify the proper logical unit numbers for the DOS-III configuration.

When preparing input for the batch device, the programmer must remember to never put a colon (:) in column one of the source statement. DOS-III aborts the current program if a directive (signified by : in column one) occurs during data input.

If the memory protect hardware option is present (and enabled), it protects the resident supervisor from alteration. It interrupts the execution of a user program under these conditions:

- Any operation that would modify the protected area or jump into it.
- Any I/O instruction, except those referencing the switch register or overflow register.
- The halt instruction.

Memory protect gives control to DOS-III when an interrupt occurs, and DOS-III checks whether it was an EXEC call. If not, the user program is aborted.

E-2. ASSEMBLER OPERATION

The DOS-III Assembler is initiated with a PROG directive. However, before entering the PROG directive, the operator must place the source program in the input device. If the source program is on the disc, the operator must first specify the file with a JFILE directive, and set parameter p_1 = 2 in the PROG directive. The PROG directive for Assembler should take the following form:

```
:PROG,ASMB[,P1,P2,P3,P4,99]
```

where:

P_1
logical unit number of input device (default is 5; set to 2 for source file input indicated by a JFILE directive)

P_2
logical unit number of list dedvice (default is 6)

P_3
logical unit number of punch device (default is 4)

P_4
lines/page on the source listing (default is 56)

99
the job binary parameter. If present, the object program is stored in the Job Binary Area for later loading. Any requested punch output still occurs. (The 99 may occur anywhere in the parameter list, but terminates the list.)

All parameters are optional. However, parameters P_1 through P_4 are positional (if present they must appear in the order shown above). If any of the parameters P_1 through P_4 are omitted, the associated (trailing) comma may also be omitted and the default value is assumed. If the 99 is omitted, the binary output is **not** placed in the Job Binary Area.

E-3. MESSAGES DURING ASSEMBLY

When the end of a source tape is encountered, the following is output on the system console:

```
I/O ERR ET EQT #n
```

EQT #n is unavailable until the operator declares it up:

```
:UP,n
:GO
```

Compilation continues after the :GO. More than one source tape can be compiled into one program by loading the next tape before giving the :GO.

The following message on the system console signifies the end of assembly:

```
$END ASMB
```

If another pass of the source program is required, this message is output at the end of pass one.

```
$END ASMB PASS
```

The operator must replace the program in the input device and enter:

```
:GO
```

If an error is found in the Assembler control statement, the following message is output on the system console:

```
$END ASMB CS
```

and the current assembly stops.

If an end-of-file condition on source input occurs before an END statement is found, the console signals:

```
$END ASMB XEND
```

and the current assembly stops.

If source input from logical unit 2 (disc) is requested, but no file has been declared, the system console signals:

```
$END ASMB NPRG
```

and the current assembly stops.

If the Job Binary Area, where binary code is stored by a 99 parameter, overflows, assembly continues but the following message is output on the system console:

```
JBIN OVF
```

However, no further binary code is stored in the Job Binary Area.

The next message is printed on a separate line just above each error diagnostic printed in the program listing during pass 1.

```
# nnn
```

nnn is the "tape" number on which the error (reported on the next line of the listing) occurred. A program may consist of more than one tape. The tape counter starts with one and increments by one whenever an end-of-tape condition occurs (paper tape) or a blank card is encountered. When the counter increments, the numbering of source statements starts over at one.

Each error diagnostic printed in the program listing during pass 2 of the assembly is associated with a different message (printed on a separate line just above each diagnostic):

```
PG ppp
```

ppp is the page number (in the listing) of the **previous** error diagnostic. PG 000 is associated with the first error found in the program.

APPENDIX F

ASSEMBLER ERROR MESSAGES

Errors detected in the source program are indicated by a 1- or 2-letter mnemonic followed by the sequence number and the first 62 characters of the statement in error. The messages are printed on the list output device during the passes indicated. A message specifying the number of errors detected is printed on the system console device at the end of each pass.

Error listings produced during Pass 1 are preceded by a number which identifies the source input file where the error was found. Pass 2 error messages are preceded by a reference to the previous page of the listing where an error message was written. The first error will refer to page "0". The error count at the end of Pass 2 is preceded by the page number in the listing where the final error was encountered.

Error Code	Pass	Description
CS	1	Control statement error: a) The control statement contained a parameter other than the legal set. b) Both A and R were specified. c) There was no output parameter (B, T, or L) and the Job Binary parameter was not specified.
DD	1	Doubly defined symbol: A name defined in the symbol table appears more than once as: a) A label of a machine instruction. b) A label of one of the pseudo operations: BSS DBL BYT DBR ASC EQU DEC ABS DEF OCT DEX Arithmetic subroutine call c) A name in the Operand field of a COM or EXT statement. d) A label in an instruction following a REP pseudo operation. e) Any combination of the above. An arithmetic subroutine call symbol appears in a program both as a pseudo instruction and as a label.
EN	1	The symbol specified in an ENT statement has already been defined in an EXT statement.
EN UNDEF <symbol>	2	The entry point specified in an ENT statement does not appear in the label field of a machine or BSS instruction. The entry point has been defined in the Operand field of an EXT statement (or has been equated to an absolute value of zero — this is not an error, but is noted).

Error Code **Pass** **Description**

IF 1 An IFZ or an IFN follows either an IFZ or an IFN without an intervening XIF. The second pseudo instruction is ignored.

IL 1 Illegal instruction:

a) Instruction mnemonic cannot be used with type of assembly requested in control statement. The following are illegal in an absolute assembly:

NAM EXT
ENT COM
Arithmetic subroutine calls

b) The ASMB statement has an R parameter, and NAM has been detected after the first valid Opcode.

1 or 2 Illegal character: A numeric term used in the Operand field contains an illegal character (e.g. an octal constant contains other than +, -, or 0-7). This code may also appear following an M error for missing operands.

M 1 or 2 Illegal operand:

a) Operand is missing for an Opcode requiring one.

b) Operands are optional and omitted but comments are included for:

END
HLT

c) Operand is an external symbol or an indirect address for:

DBL
DBR

d) An absolute expression in one of the following instructions from a relocatable program is greater than 1777_8.

Instructions referencing memory locations

DEF, DBL, and DBR
Arithmetic subroutine calls

e) A negative operand is used with an Opcode other than ABS, DEX, DEC, OCT, and BYT.

f) A character other than I follows a comma with operands which can be indirect.

g) Operand is an indirect address when used with JPY.

h) Using a literal as the second operand in the following instructions:

TBS
SBS
CBS

i) A character other than C follows a comma in certain I/O instructions.

j) A relocatable expression in the Operand field of one of the following:

ABS ASR RRL
REP ASL LSR
SPC RRR LSL

k) An illegal operator appears in an Operand field (e.g. + or - as the last character).

l) An ORG statement appearing in a relocatable program includes an expression that is common relocatable or absolute.

m) A relocatable expression contains a mixture of program and common relocatable terms.

n) An external symbol appears in an operand expression or is specified as indirect.

Error Code	Pass	Description
		o) The literal, literal code, or type of literal is illegal for the operation code used (e.g., STA = B7). p) An integer expression in one of the following instructions does not meet the condition $1 \leq n \leq 16$. The integer is evaluated modulo 2^4. ASR RRR LSR ASL RRL LSL q) The value of an 'L' type literal is relocatable.
NO	1 or 2	No origin definition: The first statement in the assembly containing a valid opcode following the ASMB control statement (and remarks and/or HED, if present) is neither an ORG nor a NAM statement. If absolute, the program is assembled starting at 2000; if relocatable, the program is assembled starting at zero.
OP	1 or 2	Illegal Opcode preceding first valid Opcode. The statement being processed does not contain an asterisk in position one. The statement is asumed to contain an illegal Opcode; it is treated as a remarks statement. Illegal Opcode: A mnemonic appears in the Opcode field which is not valid. A word is generated in the object program.
OV	1 or 2	Numeric operand overflow: The numberic value of a term or expression has overflowed its limit: $1 \geq N \geq 16$ Shift-Rotate Set 2^6-1 Input/Output, Overflow, Halt 2^{10}-1 Memory Reference (in absolute assembly) 2^{15} Data generated by DEC or DEX 2^{15}-1 DEF and ABS operands and expressions concerned with program location counter. 2^{16}-1 OCT
SO		There are more symbols defined in the program than the symbol table can handle.
SY	1 or 2	Illegal Symbol: A Label field contains an illegal character or is greater than 5 characters. A label with illegal characters may result in an erroneous assembly if not corrected. A long label is truncated on the right to 5 characters. Illegal Symbol: A symbolic term in the Operand field is greater than five characters; the symbol is truncated on the right to 5 characters. Too many control statements: The source file contains more than one control statement. The Assembler assumes that the second control statement is a label, since it begins in column 1. Thus, the commas are considered as illegal characters and the "label" is too long. The binary object program is not affected by this error. The first control statement processed is the one used by the Assembler.
UN	1 or 2	Undefined Symbol: a) A symbolic term in an Operand field is not defined in the Label field of an instruction or is not defined in the Operand field of a COM or EXT statement. b) A symbol appearing in the Operand field of one of the following pseudo operations was not defined previously in the source program: BSS ASC EQU ORG END

$2^{\pm n}$ IN DECIMAL

2^n	n	2^{-n}
1	0	1.0
2	1	0.5
4	2	0.25
8	3	0.125
16	4	0.0625
32	5	0.03125
64	6	0.01562 5
128	7	0.00781 25
256	8	0.00390 625
512	9	0.00195 3125
1 024	10	0.00097 65625
2 048	11	0.00048 82812 5
4 096	12	0.00024 41406 25
8 192	13	0.00012 20703 125
16 384	14	0.00006 10351 5625
32 768	15	0.00003 05175 78125
65 536	16	0.00001 52587 89062 5
131 072	17	0.00000 76293 94531 25
262 144	18	0.00000 38146 97265 625
524 288	19	0.00000 19073 48632 8125
1 048 576	20	0.00000 09536 74316 40625
2 097 152	21	0.00000 04768 37158 20312 5
4 194 304	22	0.00000 02384 18579 10156 25
8 388 608	23	0.00000 01192 09289 55078 125
16 777 216	24	0.00000 00596 04644 77539 0625
33 554 432	25	0.00000 00298 02322 38769 53125
67 108 864	26	0.00000 00149 01161 19384 76562 5
134 217 728	27	0.00000 00074 50580 59692 38281 25
268 435 456	28	0.00000 00037 25290 29846 19140 625
536 870 912	29	0.00000 00018 62645 14923 09570 3125
1 073 741 824	30	0.00000 00009 31322 57461 54785 15625
2 147 483 648	31	0.00000 00004 65661 28730 77392 57812 5
4 294 967 296	32	0.00000 00002 32830 64365 38696 28906 25

INDEX

HEWLETT PACKARD

SALES & SERVICE OFFICES

AFRICA, ASIA, AUSTRALIA

AMERICAN SAMOA
Calculators Only
Oceanic Systems Inc.
P.O. Box 777
Pago Pago Bayfront Road
Pago Pago 96799
Tel: 633-5513
Cable: OCEANIC-Pago Pago

ANGOLA
Telectra
Empresa Técnica de
Equipamentos
Eléctricos, S.A.R.L.
R. Barbosa Rodrigues, 42-I°DT.°
Caixa Postal, 6487
Luanda
Tel: 35515/6
Cable: TELECTRA Luanda

AUSTRALIA
Hewlett-Packard Australia
Pty. Ltd.
31-41 Joseph Street
Blackburn, Victoria 3130
P.O. Box 36
Doncaster East, Victoria 3109
Tel: 89-6351
Telex: 31-024
Cable: HEWPARD Melbourne

Hewlett-Packard Australia
Pty. Ltd.
31 Bridge Street
Pymble
New South Wales, 2073
Tel: 449-6566
Telex: 21561
Cable: HEWPARD Sydney

Hewlett-Packard Australia
Pty. Ltd.
153 Greenhill Road
Parkside, 5063, S.A.
Tel: 272-5911
Telex: 82536 ADEL
Cable: HEWPARD ADELAIDE

Hewlett-Packard Australia
Pty. Ltd.
141 Stirling Highway
Nedlands, W.A. 6009
Tel: 86-5455
Telex: 93859 PERTH
Cable: HEWPARD PERTH

Hewlett-Packard Australia
Pty. Ltd.
121 Wollongong Street
Fyshwick, A.C.T. 2609
Tel: 95-3733
Telex: 62650 Canberra
Cable: HEWPARD CANBERRA

Hewlett Packard Australia
Pty. Ltd.
5th Floor
Teachers Union Building
495-499 Boundary Street
Spring Hill, 4000 Queensland
Tel: 29-1544
Telex: 42133 BRISBANE

GUAM
Medical/Pocket Calculators Only
Guam Medical Supply, Inc.
Jay Ease Building, Room 210
P.O. Box 8947
Tamuning 96911
Tel: 646-4513
Cable: EARMED Guam

HONG KONG
Schmidt & Co.(Hong Kong) Ltd.
P.O. Box 297
Connalight Centre
39th Floor
Connaught Road, Central
Hong Kong
Tel: H-255291-5
Telex: 74766 SCHMC HX
Cable: SCHMIDTCO Hong Kong

INDIA
Blue Star Ltd.
Kasturi Buildings
Jamshedji Tata Rd.
Bombay 400 020
Tel: 29 50 21
Telex: 2156
Cable: BLUEFROST

Blue Star Ltd.
Sahas
414/2 Vir Savarkar Marg
Prabhadevi
Bombay 400 025
Tel: 45 78 87
Telex: 4093
Cable: FROSTBLUE

Blue Star Ltd.
Band Box House
Prabhadevi
Bombay 400 025
Tel: 45 73 01
Telex: 3751
Cable: BLUESTAR

Blue Star Ltd.
14/40 Civil Lines
Kanpur 208 001
Tel: 6 88 82
Telex: 292
Cable: BLUESTAR

Blue Star Ltd.
7 Hare Street
P.O. Box 506
Calcutta 700 001
Tel: 23-0131
Telex: 7655
Cable: BLUESTAR

Blue Star Ltd.
7th & 8th Floor
Bhandari House
91 Nehru Place
New Delhi 110024
Tel: 634770 & 635166
Telex: 2463
Cable: BLUESTAR

Blue Star Ltd.
Blue Star House
11/11A Magarath Road
Bangalore 560 025
Tel: 55668
Telex: 430
Cable: BLUESTAR

Blue Star Ltd.
Meeakshi Mandiran
xxx/1678 Mahatma Gandhi Rd.
Cochin 682 016 Kerala
Tel: 32069, 32161, 32282
Telex: 046-514
Cable: BLUESTAR

Blue Star Ltd.
1-1-117/1
Sarojini Devi Road
Secunderabad 500 003
Tel: 70126, 70127
Cable: BLUEFROST
Telex: 459

Blue Star Ltd.
2/34 Kodambakkan High Road
Madras 600034
Tel: 82056
Telex: 041-379
Cable: BLUESTAR

Blue Star Ltd.
Nathraj Mansions
2nd Floor Bistupur
Jamshedpur 831 001
Tel: 7383
Cable: BLUESTAR
Telex: 240

INDONESIA
BERCA Indonesia P.T.
P.O. Box 496
1st Floor JL, Cikini Raya 61
Jakarta
Tel: 56038, 40369, 49886
Telex: 42895
Cable: BERCACON

BERCA Indonesia P.T.
63 JL. Raya Gubeng
Surabaya
Tel: 44309

ISRAEL
Electronics & Engineering Div.
of Motorola Israel Ltd.
16, Kremenetski Street
P.O. Box 25016
Tel-Aviv
Tel: 03-389 73
Telex: 33569
Cable: BASTEL Tel-Aviv

JAPAN
Yokogawa-Hewlett-Packard Ltd.
Ohashi Building
1-59-1 Yoyogi
Shibuya-ku, **Tokyo**
Tel: 03-370-2281/92
Telex: 232-2024YHP
Cable: YHPMARKET TOK 23-724

Yokogawa-Hewlett-Packard Ltd.
Nissei Ibaraki Building
2-8 Kasuga 2-chrome, Ibaraki-shi
Osaka,567
Tel: (0726) 23-1641
Telex: 5332-385 YHP OSAKA

Yokogawa-Hewlett-Packard Ltd.
Nakamo Building
24 Kami Sasajima-cho
Nakamura-ku, **Nagoya** , 450
Tel: (052) 571-5171

Yokogawa-Hewlett-Packard Ltd.
Tanigawa Building
2-24-1 Tsuruya-choo
Kanagawa-ku
Yokohama, 221
Tel: 045-312-1252
Telex: 382-3204 YHP YOK

Yokogawa-Hewlett-Packard Ltd.
Mito Mitsui Building
105, 1-chrome, San-no-maru
Mito, Ibaragi 310
Tel:: 0292-25-7470

Yokogawa-Hewlett-Packard Ltd.
Inoue Building
1348-3, Asahi-cho, 1-chome
Atsugi, Kanagawa 243
Tel: 0462-24-0452

Yokogawa-Hewlett-Packard Ltd.
Inoue Building
1348-3, Asahi-cho, 1-chome
Atsugi, Kanagawa 243
Tel: 0462-24-0452

Yokogawa-Hewlett-Packard Ltd.
Kimura Building
3rd Floor 20
2-chome, Tsukuba
Kumagaya, Saitama 360
Tel: 0485-24-6563

KENYA
Technical Engineering Services
(E.A.)Ltd.,
P.O. Box 18311
Nairobi
Tel: 557726/556762
Cable: PROTON

Medical Only
International Aeradio(E.A.)Ltd.,
P.O. Box 19012
Nairobi Airport
Nairobi
Tel: 336055/56
Telex: 22201/22301

KOREA
Samsung Electronics Co., Ltd.
20th Fl. Dongbang Bldg. 250,
2-KA , C.P.O. Box 2775
Taepyung-Ro, Chung-Ku
Seoul
Tel: (24) 2410-9
Telex: 22575

MALAYSIA
Teknik Mutu Sdn. Bhd.
2 Lorong 13/6A
Section 13
Petaling Jaya,**Selangor**
Tel: Kuala Lumpur-54994
Telex: MA 37605

Protel Engineering
P.O. Box I9I7
Lot 259, Satok Road
Kuching, **Sarawak**
Tel: 2400

MOZAMBIQUE
A.N. Goncalves, Lta.
162, 1° Apt. 14 Av. D. Luis
Caixa Postal 107
Lourenco Marques
Tel: 27091, 27114
Telex: 6-203 Negon Mo

NEW ZEALAND
Hewlett-Packard (N.Z.) Ltd.
4-12 Cruickshank Street
Kilbirnie, Wellington 3
Mailing Address: Hewlett-Packard
(N.Z.) Ltd.
P.O. Box 9443
Courtney Place
Wellington
Tel: 877-199
Telex: NZ 3839
Cable: HEWPACK Wellington

Hewlett-Packard (N.Z.) Ltd.
Pakuranga Professional Centre
267 Pakuranga Highway
Box 51092
Pakuranga
Tel: 569-651
Telex: NZ 3839
Cable: HEWPACK,Auckland

Analytical/Medical Only
Medical Supplies N.Z. Ltd.
Scientific Division
79 Carlton Gore Rd., Newmarket
P.O. Box 1234
Auckland
Tel: 75-289
Telex: 2958 MEDISUP
Cable: DENTAL Auckland

Analytical/Medical Only
Medical Supplies N.Z. Ltd.
P.O. Box 1994
147-161 Tory St.
Wellington
Tel: 850-799
Telex: 3858
Cable: DENTAL, Wellington

Analytical/Medical Only
Medical Supplies N.Z. Ltd.
P.O. Box 309
239 Stanmore Road
Christchurch
Tel: 892-019
Cable: DENTAL, Christchurch

Analytical/Medical Only
Medical Supplies N.Z. Ltd.
303 Great King Street
P.O. Box 233
Dunedin
Tel: 88-817
Cable: DENTAL, Dunedin

NIGERIA
The Electronics
Instrumentations Ltd.
N6B/770 Oyo Road
Oluseun House
P.M.B. 5402
Ibadan
Tel: 61577
Telex: 31231 TEIL Nigeria
Cable: THETEIL Ibadan

The Electronics Instrumentations Ltd.
144 Agege Motor Road, Mushin
P.O. Box 6645
Lagos
Cable: THETEIL Lagos

PAKISTAN
Mushko & Company, Ltd.
Oosman Chambers
Abdullah Haroon Road
Karachi-3
Tel: 511027, 512927
Telex: KR894
Cable: COOPERATOR Karachi

Mushko & Company, Ltd.
38B, Satellite Town
Rawalpindi
Tel: 41924
Cable: FEMUS Rawalpindi

PHILIPPINES
The Online Advanced Systems
Corporation
Filcapital Bldg.
11th Floor, Ayala Ave.
Makati, **Rizal**
Tel: 85-34-91, 85-35-81
Telex: 3274 ONLINE

RHODESIA
Field Technical Sales
45 Kelvin Road North
P.O. Box 3458
Salisbury
Tel: 705231 (5 lines)
Telex: RH 4I22

SINGAPORE
Hewlett-Packard Singapore
(Pte.) Ltd.
Blk. 2, 6th Floor, Jalan
Bukit Merah
Redhill Industrial Estate
Alexandra P.O. Box 58,
Singapore 3
Tel: 633022
Telex: HPSG RS 21486
Cable: HEWPACK, Singapore

SOUTH AFRICA
Hewlett-Packard South Africa
(Pty.), Ltd.
Private Bag Wendywood
Sandton, Transvaal 2144
Hewlett-Packard House
Daphne Street, Wendywood,
Sandton, Transvaal 2144
Tel: 802-104016
Telex: SA43-4782JH
Cable: HEWPACK JOHANNESBURG

Hewlett-Packard South Africa
(Pty.), Ltd.
P.O. Box 120
Howard Place, Cape Province, 7450
Pine Park Center, Forest Drive,
Pinelands, Cape Province, 7405
Tel: 53-7955 thu 9
Telex: 57-0006

Hewlett-Packard South Africa
(Pty.), Ltd.
P.O. Box 37099
Overport, Durban 4067
641 Ridge Road, Durban
Durban, 4001
Tel: 88-7478/9
Telex: 6-7954
Cable: HEWPACK

TAIWAN
Hewlett-Packard Far East Ltd.,
Taiwan Branch
39 Chung Shiao West Road
Sec. 1, 7th Floor
Taipei
Tel: 3819160-4 (5 Lines)
Telex: 21824 HEWPACK
Cable: HEWPACK TAIPEI

Hewlett-Packard Far East Ltd.
Taiwan Branch
68-2, Chung Cheng 3rd. Road
Kaohsiung
Tel: (07) 242318-Kaohsiung

Analytical Only
San Kwang Instruments Co., Ltd.,
No. 20, yung Sui Road
Taipei, 100
Tel: 3715I7I-4 (5 lines)
Telex: 22894 SANKWANG
Cable: SANKWANG TAIPEI

TANZANIA
Medical Only
International Aeradio (E.A.), Ltd.
P.O. Box 861
Daressalaam
Tel: 21251 Ext. 265
Telex: 41030

THAILAND
UNIMESA Co., Ltd.
Elcom Research Building
Bangjak Sukumvit Ave.
Bangkok
Tel: 932387, 930338
Cable: UNIMESA Bangkok

UGANDA
Medical Only
International Aeradio(E.A.), Ltd.,
P.O. Box 2577
Kampala
Tel: 54388
Cable: INTAERIO Kampala

ZAMBIA
R.J. Tilbury (Zambia) Ltd.
P.O. Box 2792
Lusaka
Tel: 73793
Cable: ARJAYTEE, Lusaka

OTHER AREAS NOT LISTED, CONTACT:
Hewlett-Packard Intercontinental
3200 Hillview Ave.
Palo Alto, California 94304
Tel: (415) 493-1501
TWX: 910-373-1267
Cable: HEWPACK Palo Alto
Telex: 034-8300, 034-8493

CANADA

ALBERTA
Hewlett-Packard (Canada) Ltd.
11620A - 168 Street
EdmontonT5M 3T9
Tel: (403) 452-3670
TWX: 610-831-2431 EDTH

Hewlett-Packard (Canada) Ltd.
915-42 Ave S.E. Suite 102
Calgary T2G 1Z1
Tel: (403) 287-1672
Twx; 6I0-82I-6I4I

BRITISH COLUMBIA
Hewlett-Packard (Canada) Ltd.
837 E. Cordova Street
Vancouver V6A 3R2
Tel: (604) 254-0531
TWX: 610-922-5059 VCR

MANITOBA
Hewlett-Packard (Canada) Ltd.
513 Century St.
St. James
Winnipeg R3H OL8
Tel: (204) 786-7581
TWX: 610-671-3531

NOVA SCOTIA
Hewlett-Packard (Canada) Ltd.
800 Windmill Road
P.O. Box 9331
Dartmouth B2Y 3Z6
Tel: (902) 469-7820
TWX: 6I0-27I-4482 HFX

ONTARIO
Hewlett-Packard (Canada) Ltd.
1785 Woodward Dr.
Ottawa K2C OP9
Tel: (613) 225-6530
TWX: 610-562-8968

Hewlett-Packard (Canada) Ltd.
6877 Goreway Drive
Mississauga L4V 1M8
Tel: (416) 678-9430
TWX: 610-492-4246

QUEBEC
Hewlett-Packard (Canada) Ltd.
275 Hymus Blvd.
Pointe Claire H9R 1G7
Tel: (514) 697-4232
TWX: 610-422-3022
TLX: 05-821521 HPCL

FOR CANADIAN AREAS NOT LISTED:
Contact Hewlett-Packard (Canada)
Ltd. in Mississauga.

CENTRAL AND SOUTH AMERICA

ARGENTINA
Hewlett-Packard Argentina
S.A.
Av. Leandro N. Alem 822 - 12°
1001**Buenos Aires**
Tel: 31-6063,4,5,6 and 7
Telex: Public Booth N° 9
Cable: HEWPACK ARG

BOLIVIA
Stambuk & Mark (Bolivia) Ltda.
Av. Mariscal, Santa Cruz 1342
La Paz
Tel: 40626, 53163, 52421
Telex: 3560014
Cable: BUKMAR

BRAZIL
Hewlett-Packard do Brasil
I.E.C. Ltda.
Avenida Rio Negro, 980
Alphaville
06400 Barueria **Sao Paulo**
Tel: 429-2148/9;429-2118/9

Hewlett-Packard do Brasil
I.E.C. Ltda.
Rua Padre Chagas, 32
90000-**Pórto Alegre**-RS
Tel: (0512) 22-2998, 22-5621
Cable: HEWPACk porto Alegre

Hewlett-Packard do Brasil
I.E.C. Ltda.
Rua Siqueira Campos, 53, 4°
andar-Copacabana
20000-**Rio de Janeiro**-GB
Tel: 257-80-94-DDD (021)
Telex: 39I-212-I905 HEWP-BR
Cable: HEWPACK
Rio de Janeiro

CHILE
Calcagni y Metcalfe Ltda.
Alameda 580-Of. 807
Casilla 2118
Santiago, 1
Tel: 398613
Telex: 3520001 CALMET
Cable: CALMET Santiago

Medical Only
General Machinery Co., Ltda.
Paraguay 494
Casilla 13910
Santiago
Tel: 31123, 31124
Cable: GEMCO Santiago

COLOMBIA
Instrumentación
Henrik A. Langebaek & Kier S.A.
Carrera 7 No. 48-75
Apartado Aéreo 6287
Bogotá, I D.E.
Tel: 69-88-77
Cable: AARIS Bogotá
Telex: 044-400

COSTA RICA
Cientifica Costarricense S.A.
Calle Central, Avenidas 1 y 3
Apartado 10159
San José
Tel: 21-86-13
Cable: GALGUR San José

ECUADOR
Medical Only
A.F. Viscaino Compañía Ltda.
Av. Rio Amazonas No. 239
P.O. Box 2925
Quito
Tel: 242-150,247-033/034
Cable: Astor Quito

Calculators Only
Computadoras y Equipos
Electrónicos
P.O. Box 2695
990 Toledo (y Cordero)
Quito
Tel: 525-982
Telex: 02-2113 Sagita Ed
Cable: Sagita-Quito

EL SALVADOR
Instrumentacion y Procesamiento
Electronico de el Salvador
Bulevar de los Heroes II-48
San Salvador
Tel: 252787

GUATEMALA
IPESA
Avenida La Reforma 3-48,
Zona 9
Guatemala City
Tel: 63627, 64786
Telex: 4192 Teletro.Gu

MEXICO
Hewlett-Packard Mexicana,
S.A. de C.V.
Torres Adalid No. 21, 11° Piso
Col. del Valle
Mexico 12, D.F.
Tel: (905) 543-42-32
Telex: 017-74-507

Hewlett-Packard Mexicana,
S.A. de C.V.
Ave. Constitución No. 2184
Monterrey, N.L.
Tel: 48-71-32, 48-71-84
Telex: 038-843

NICARAGUA
Roberto Terán G.
Apartado Postal 689
Edificio Terán
Managua
Tel: 25114, 23412,23454
Cable: ROTERAN Managua

PANAMA
Electrónico Balboa, S.A.
P.O. Box 4929
Calle Samuel Lewis
Cuidad de Panama
Tel: 64-2700
Telex: 3431103 Curunda,
Canal Zone
Cable: ELECTRON Panama

PARAGUAY
Z.J. Melamed S.R.L.
División: Aparatos y Equipos
Médicos
División: Aparatos y Equipos
Científicos y de Investigación
P.O. Box 676
Chile-482, Edificio Victoria
Asunción
Tel: 4-5069, 4-6272
Cable: RAMEL

PERU
Compañía Electro Médica S.A.
Los Flamencos 145
San Isidro Casilla 1030
Lima 1
Tel: 41-4325
Cable: ELMED Lima

PUERTO RICO
Hewlett-Packard Inter-Americas
Puerto Rico Branch Office
Calle 272, Urb. Country Club
Carolina 00639
Tel: (809) 762-7355/7455/7655
Telex: HPIC-PR 3450514

URUGUAY
Pablo Ferrando S.A.
Comercial e Industrial
Avenida Italia 2877
Casilla de Correo 370
Montevideo
Tel: 40-3102
Cable: RADIUM Montevideo

VENEZUELA
Hewlett-Packard de Venezuela
C.A.
Apartado 50933, Caracas I05
Edificio Segre
Tercera Transversal
Los Ruices Norte
Caracas 107
Tel: 35-0I-07, 35-00-84,
35-00-65, 35-00-3I
Telex: 25146 HEWPACK
Cable: HEWPACK Caracas

FOR AREAS NOT LISTED, CONTACT:
Hewlett-Packard
Inter-Americas
3200 Hillview Ave.
Palo Alto, California 94304
Tel: (415) 493-1501
TWX: 910-373-1260
Cable: HEWPACK Palo Alto
Telex: 034-8300, 034-8493

EUROPE, NORTH AFRICA AND MIDDLE EAST

AUSTRIA
Hewlett-Packard Ges.m.b.H.
Handelskai 52
P.O. box 7
A-1205 **Vienna**
Tel: (0222) 35 16 21 to 27
cable: HEWPAK Vienna
Telex: 75923 hewpak a

BELGIUM
Hewlett-Packard Benelux
S.A./N.V.
Avenue de Col-Vert, 1,
(Groenkraaglaan)
B-1170 **Brussels**
Tel: (02) 672 22 40
Cable: PALOBEN Brussels
Telex: 23 494 paloben bru

CYPRUS
Kypronics
19, Gregorios & Xenopoulos Rd.
P.O. Box 1152
CY-**Nicosia**
Tel: 45628/29
Cable: KYPRONICS PANDEHIS
Telex: 3018

CZECHOSLOVAKIA
Vyvojova a Provozni Zakladna
Vyzkumnych Ustavu v Bechovicich
CSSR-25097
Bechovice u Prahy
Tel: 89 93 41
Telex: 121333

DDR
Entwicklungslabor der TU Dresden
Forschungsinstitut Meinsberg
DDR-7305
Waldheim/Meinsberg
Tel: 37 667
Telex: 518741

Firma Forgber
Schlegelstrasse 15
1040 **Berlin**
Tel: 28 27 411
Telex: 112889

DENMARK
Hewlett-packard A/S
Datavej 52
DK-3460 **Birkerød**
Tel: (02) 81 66 40
Cable: HEWPACK AS
Telex: 166 40 hpas

Hewlett-Packard A/S
Navervej 1
DK-8600 **Silkeborg**
Tel: (06) 82 71 66
Telex: 166 40 hpas
Cable: HEWPACK AS

FINLAND
Hewlett-Packard OY
Nahkahousuntie 5
P.O. Box 6
SF-00211 **Helsinki** 21
Tel: 6923031
Cable: HEWPACKOY Helsinki
Telex: 12-1563

FRANCE
Hewlett-Packard France
Quartier de Courtaboeuf
Boite Postale No. 6
F-91401 **Orsay** Cédex
Tel: (1) 907 78 25
Cable: HEWPACK Orsay
Telex: 600048

Hewlett-Packard France
"Le Saquin"
Chemin des Mouilles
Boite Postale No. 12
F-69130 **Ecully**
Tel: (78) 33 81 25,
Cable: HEWPACK Eculy
Telex: 310617

Hewlett-Packard France
Agence Régionale
Péricentre de la Cépière
Chemin de la Cépière, 20
F-31300 **Toulouse-Le Mirail**
Tel:(61) 40 11 12
Cable: HEWPACK 51957
Telex: 510957

Hewlett-Packard France
Agence Régionale
Aéroport principal de
Marseille-Marignane
F-13721**Marignane**
Tel: (91) 89 12 36
Cable: HEWPACK MARGN
Telex: 410770

Hewlett-Packard France
Agence Régionale
63, Avenue de Rochester
Boîte Postale
F-35014 **Rennes** Cédex
Tel: (99) 36 33 21
Cable: HEWPACK 74912
Telex: 740912

Hewlett-Packard France
Agence Régionale
74, Allée de la Robertsau
F-67000 **Strasbourg**
Tel: (88) 35 23 20/21
Telex: 890141
Cable: HEWPACK STRBG

Hewlett-Packard France
Agence Régionale
Centre Vauban
201, rue Colbert
Entrée A2
F-59000 **Lille**
Tel: (20) 51 44 14
Telex: 820744

GERMAN FEDERAL REPUBLIC
Hewlett-Packard GmbH
Vertriebszentrale Frankfurt
Bernerstrasse 117
Postfach 560 140
D-6000 **Frankfurt** 56
Tel: (0611) 50 04-1
Cable: HEWPACKSA Frankfurt
Telex: 04 13249 hpffmd

Hewlett-Packard GmbH
Technisches Buero Böblingen
Herrenbergerstrasse 110
D-7030 **Böblingen**, Württemberg
Tel: (07031) 667-1
Cable: HEPAK Böblingen
Telex: 07265739 bbn

Hewlett-Packard GmbH
Technisches Buero Düsseldorf
Emanuel-Leutze-Str.1 (Seestern)
D-4000 **Düsseldorf**
Tel: (0211) 59 71-1
Telex: 085/86 533 hpdd d

Hewlett-Packard GmbH
Technisches Buero Hamburg
Wendenstrasse 23
D-2000 **Hamburg** 1
Tel: (040) 24 13 93
Cable: HEWPACKSA Hamburg
Telex: 21 63 032 hphh d

Hewlett-Packard GmbH
Technisches Buero Hannover
Am Grossmarkt 6
D-3000 **Hannover 91**
Tel: (0511) 46 60 01
Telex: 092 3259

Hewlett-Packard GmbH
Technisches Buero Nuremberg
Neumeyer Str. 90
D-8500 **Nuremberg**
Tel: (0911) 56 30 83/85
Telex: 0623 860

Hewlett-Packard GmbH
Technisches Buero München
Unterhachinger Strasse 28
ISAR Center
D-8012 **Ottobrunn**
Tel: (089) 601 30 61/7
Cable: HEWPACKSA München
Telex: 0524985

Hewlett-Packard GmbH
Technisches Buero Berlin
Keith Strasse 2-4
D-1000 **Berlin** 30
Tel: (030) 24 90 86
Telex: 18 3405 hpbln d

GREECE
Kostas Karayannis
18, Ermou Street
GR-**Athens** 126
Tel: 3237731
Cable: RAKAR Athens
Telex: 21 59 62 rkar gr

Analytical Only
"INTECO"
G. Papathanassiou & Co.
Marni 17
GR - **Athens** 103
Tel: 522 1915
Cable: INTEKNIKA Athens
Telex: 21 5329 INTE GR

Medical Only
Technomed Hellas Ltd.
52,Skoufa Street
GR - **Athens** 135
Tel: 362 6972, 363 3830
Cable:etalak athens
Telex: 21-4693 ETAL GR

HUNGARY
MTA
Müszerügyi és Méréstechnikai
Szolgalata
Lenin Krt. 67
1391 **Budapest** VI
Tel: 42 03 38
Telex: 22 51 14

ICELAND
Medical Only
Elding Trading Company Inc.
Hafnarhvoli - Tryggvatotu
IS-**Reykjavik**
Tel: 1 58 20
Cable: ELDING Reykjavik

IRAN
Hewlett-Packard Iran Ltd.
No. 13, Fourteenth St.
Miremad Avenue
P.O. Box 41/2419
IR-**Tehran**
Tel: 851082-7
Telex: 213405 HEWP IR

IRELAND
Hewlett-Packard Ltd.
King Street Lane
GB-**Winnersh,**Wokingham
Berks, RG11 5AR
Tel: (0734) 78 47 74
Telex: 847178

ITALY
Hewlett-Packard Italiana S.p.A.
Casella postale 3645
I-20100 **Milano**
Tel: (2) 6251 (10 lines)
Cable: HEWPACKIT Milano
Telex: 32046

Hewlett-Packard Italiana S.p.A.
Via Pietro Maroncelli 40
(ang. Via Visentin)
I-35100 **Padova**
Tel: (49) 66 48 88
Telex: 41612 Hewpacki

Medical only
Hewlett-Packard Italiana S.p.A.
Via d'Aghiardi, 7
I-56100 **Pisa**
Tel: (050) 2 32 04
Telex: 32046 via Milano

Hewlett-Packard Italiana S.p.A.
Via G. Armellini 10
I-00143 **Roma**
Tel: (06) 54 69 6l
Telex: 61514
Cable: HEWPACKIT Roma

Hewlett-Packard Italiana S.p.A.
Via San Quintino, 46
I-10121 **Torino**
Tel: (011) 52 82 64/54 84 68
Telex: 32046 via Milano

Medical/Calculators Only
Hewlett-Packard Italiana S.p.A.
Via Principe Nicola 43 G/C
I-95126 **Catania**
Tel:(095) 37 05 04

Hewlett-Packard Italiana S.p.A.
Via Amerigo Vespucci, 9
I-80142 **Napoli**
Tel: (081) 33 77 11

Hewlett-Packard Italiana S.p.A.
Via E. Masi, 9/B
I-40137 **Bologna**
Tel: (051) 30 78 87

KUWAIT
Al-Khaldiya Trading &
Contracting Co.
P.O. Box 830
Kuwait
Tel: 42 49 10
Cable: VISCOUNT

LUXEMBURG
Hewlett-Packard Benelux
S.A./N.V.
Avenue du Col-Vert, 1,
(Groenkraaglaan)
B-1170 **Brussels**
Tel: (02) 672 22 40
Cable: PALOBEN Brussels
Telex: 23 494

MOROCCO
Gerep
190, Blvd. Brahim Roudani
Casablanca
Tel: 25-16-76/25-90-99
Cable: Gerep-Casa
Telex: 23739

NETHERLANDS
Hewlett-Packard Benelux N.V.
Van Heuven Goedhartlaan 121
P.O. Box 667
NL- **Amstelveen** 1134
Tel: (020) 47 20 21
Cable: PALOBEN Amsterdam
Telex: 13 216 hepa nl

NORWAY
Hewlett-Packard Norge A/S
Nesveien 13
Box 149
N-1344 **Haslum**
Tel: (02) 53 83 60
Telex: 16621 hpnas n

POLAND
Biuro Informacji Technicznej
Hewlett-Packard
U1 Stawki 2 6P
00-950**Warsaw**
Tel: 39 67 43
Telex: 81 24 53 hepa pl

UNIPAN
Zaklad Doswiadczalny
Budowy Aparatury Naukowej
U1. Krajowej Rady
Narodowej 51/55
00-800 **Warsaw**
Tel: 20 62 21
Telex: 81 46 48

Zaklady Naprawcze Sprzetu
Medycznego
Plac Komuny Paryskiej 6
90-007 **Lodz**
Tel: 334-41, 337-83

PORTUGAL
Telectra-Empresa Técnica de
Equipamentos Eléctricos S.a.r.l.
Rua Rodrigo da Fonseca 103
P.O. Box 2531
P-**Lisbon** 1
Tel: (19) 68 60 72
Cable: TELECTRA Lisbon
Telex: 12598

Medical only
Mundinter
Intercambio Mundial de Comércio
S.a.r.l.
Av.A.A.de Aguiar 138
P.O. Box 2761
P - **Lisbon**
Tel: (19) 53 21 31/7
Cable: INTERCAMBIO Lisbon

RUMANIA
Hewlett-Packard Reprezentanta
BD.N. Balcescu 16
Bucharest
Tel: 158023/138885
Telex: 10440

I.I.R.U.C.
Intreprinderea Pentru
Intretinerea
Si Repararea Utilajelor de Calcul
B-dul prof. Dimitrie Pompei 6
Bucharest-Sectorul 2
Tel: 12 64 30
Telex: 01183716

SAUDI ARABIA
Modern Electronic Establishment
King Abdul Aziz str.(Head office)
P.O. Box 1228
Jeddah
Tel: 31173-332201
Cable: ELECTRA

P.O. Box 2728 (Service center)
Riyadh
Tel: 62596-66232
Cable: RAOUFCO

SPAIN
Hewlett-Packard Española, S.A.
Jerez No. 3
E-**Madrid** 16
Tel:(1) 458 26 00 (10 lines)
Telex: 23515 hpe

Hewlett-Packard Española, S.A.
Milanesado 21-23
E-**Barcelona** 17
Tel: (3) 203 6200 (5 lines)
Telex: 52603 hpbe e

Hewlett-Packard Española, S.A.
Av Ramón y Cajal, 1-9°
(Edificio Sevilla I)
E-**Seville 5**
Tel: 64 44 54/58

Hewlett-Packard Española S.A.
Edificio Albia II 7° B
E-**Bilbao-1**
Tel: 23 83 06/23 82 06

Calculators Only
Hewlett-Packard Española S.A.
Gran Via Fernando El Católico, 67
E-**Valencia**-8
Tel: 326 67 28/326 85 55

SWEDEN
Hewlett-Packard Sverige AB
Enighetsvägen 3
Fack
S-161 20 **Bromma** 20
Tel: (08) 730 05 50
Cable: MEASUREMENTS
Stockholm
Telex: 10721

Hewlett-Packard Sverige AB
Frötallsgatan 30
S-421 32 **Västra Frölunda**
Tel: (031) 49 09 50
Telex: 10721 Via Bromma Office

SWITZERLAND
Hewlett-Packard (Schweiz) AG
Zürcherstrasse 20
P.O. Box 307
CH-8952 **Schlieren-Zurich**
Tel: (01) 730 52 40
Cable: HPAG CH
Telex; 53933 hpag ch

Hewlett-Packard (Schweiz) AG
Château Bloc 19
CH-1219 **Le Lignon-Geneva**
Tel: (022) 96 03 22
Cable: HEWPACKAG Geneva
Telex: 27 333 hpag ch

SYRIA
Medical/Calculator only
Sawah & Co.
Place Azmé
B.P. 2308
SYR-**Damascus**
Tel: 16367, 19697, 14268
Cable: SAWAH, Damascus

TURKEY
Telekom Engineering Bureau
P.O. Box 437
Beyoglu
TR-**Istanbul**
Tel: 49 40 40
Cable: TELEMATION Istanbul
Telex: 23609

Medical only
E.M.A.
Muhendislik Kollektif Sirketi
Adakale Sokak 41/6
TR-**Ankara**
Tel: 175622

Analytical only
Yilmaz Ozyurek
Milli Mudafaa Cad No. 16/6
Kizilay
TR-**Ankara**
Tel: 25 03 09
Telex: 42576 Ozek tr

UNITED KINGDOM
Hewlett-Packard Ltd.
King Street Lane
GB-**Winnersh,** Wokingham
Berks. RG11 5AR
Tel: (0734) 78 47 74
Cable: Hewpie London
Telex:847178/9

Hewlett-Packard Ltd.
"The Graftons"
Stamford New Road
GB-**Altrincham**
Cheshire WA14 IDQ
Tel: (061) 9289021
Cable: Hewpie Manchester
Telex: 668068

Hewlett-Packard Ltd.
Lygon Court
Dudley Road
GB-**Halesowen**, Worcs
Tel: (021) 550 9911
Telex: 339105

Hewlett-Packard Ltd.
Wedge House
799, London Road
GB-**Thornton Heath**
Surrey CR4 6XL
Tel: (01) 6640103
Telex: 946825

Hewlett-Packard Ltd.
c/o Makro
South Service Wholesale Centre
Wear Industrial Estate
Washington
GB-**New Town**, County Durham
Tel: Washington 464001 ext. 57/58

Hewlett-Packard Ltd
10, Wesley St.
GB-**Castleford**
West Yorkshire WF10 1AE
Tel: (09775) 50402
Telex: 557355

Hewlett-Packard Ltd
1, Wallace Way
GB-**Hitchin**
Herts
Tel: (0462) 52824/56704
Telex: 825981

USSR
Hewlett-Packard
Representative Office USSR
Pokrovsky Boulevard 4/17-KV 12
Moscow 101000
Tel:294-2024
Telex: 7825 hewpak su

YUGOSLAVIA
Iskra-standard/Hewlett-Packard
Miklosiceva 38/VII
61000 **Ljubljana**
Tel: 31 58 79/32 16 74
Telex: 31300

SOCIALIST COUNTRIES NOT SHOWN PLEASE CONTACT:
Hewlett-Packard Ges.m.b.H
P.O. Box 7
A-1205 **Vienna**, Austria
Tel: (0222) 35 16 21 to 27
Cable: HEWPAK Vienna
Telex: 75923 hewpak a

MEDITERRANEAN AND MIDDLE EAST COUNTRIES NOT SHOWN PLEASE CONTACT:
Hewlett-Packard S.A.
Mediterranean and Middle
East Operations
35, Kolokotroni Street
Platia Kefallariou
GR-Kifissia-**Athens**, Greece
Tel: 8080337/359/429
8081741/742/743/744
Telex: 21-6588
Cable: HEWPACKSA Athens

FOR OTHER AREAS NOT LISTED CONTACT
Hewlett-Packard S.A.
7, rue du Bois-du-Lan
P.O. Box
CH-1217 Meyrin 2 - **Geneva**
Switzerland
Tel: (022) 41 54 00

UNITED STATES

ALABAMA
8290 Whitesburg Dr., S.E.
P.O. Box 4207
Huntsville 35802
Tel: (205) 881-4591

Medical Only
228 W. Valley Ave.,
Room 220
Birmingham 35209
Tel: (205) 942-2081

ARIZONA
2336 E. Magnolia St.
Phoenix 85034
Tel: (602) 244-1361

2424 East Aragon Rd.
Tucson 85706
Tel: (602) 294-3148

***ARKANSAS**
Medical Service Only
P.O. Box 5646
Brady Station
Little Rock 72205
Tel:)501) 664-8773

CALIFORNIA
1430 East Orangethorpe Ave.
Fullerton 92631
Tel: (714) 870-1000

3939 Lankershim Boulevard
North Hollywood 91604
Tel: (213) 877-1282
TWX: 910-499-2170

6305 Arizona Place
Los Angeles 90045
Tel: (213) 649-2511
TWX: 910-328-6147

***Los Angeles**
Tel: (213) 776-7500

3003 Scott Boulevard
Santa Clara 95050
Tel: (408) 249-7000
TWX: 910-338-0518

***Ridgecrest**
Tel: (714) 446-6165

646 W. North Market Blvd
Sacramento 95834
Tel: (916) 929-7222

9606 Aero Drive
P.O. Box 23333
San Diego 92123
Tel: (714) 279-3200

COLORADO
5600 South Ulster Parkway
Englewood 80110
Tel: (303) 771-3455

CONNECTICUT
12 Lunar Drive
New Haven 06525
Tel: (203) 389-6551
TWX: 710-465-2029

FLORIDA
P.O. Box 24210
2806 W. Oakland Park Blvd.
Ft. Lauderdale 33307
Tel: (305) 731-2020

***Jacksonville**
Medical Service only
Tel: (904) 725-6333

P.O. Box 13910
6177 Lake Ellenor Dr.
Orlando 32809
Tel: (305) 859-2900

P.O. Box 12826
Pensacola 32575
Tel: (904) 434-3081

GEORGIA
P.O. Box 105005
Atlanta 30348
Tel: (404) 955-1500
TWX:810-766-4890

Medical Service Only
***Augusta** 30903
Tel: (404) 736-0592

HAWAII
2875 So. King Street
Honolulu 96814
Tel: (808) 955-4455

ILLINOIS
5201 Tollview Dr.
Rolling Meadows 60008
Tel: (312) 255-9800
TWX: 910-687-2260

INDIANA
7301 North Shadeland Ave.
Indianapolis46250
Tel: (317)842-1000
TWX: 810-260-1797

IOWA
1902 Broadway
Iowa City 52240
Tel: (319) 338-9466
Night: (319) 338-9467

KENTUCKY
Medical Only
Atkinson Square
3901 Atkinson Dr.
Suite 207
Louisville 40218
Tel: (502) 456-1573

LOUISIANA
P.O. Box 840
3239 Williams Boulevard
Kenner 70062
Tel: (504) 721-6201

MARYLAND
6707 Whitestone Road
Baltimore 21207
Tel: (301) 944-5400
TWX: 710-862-9157

2 Choke Cherry Road
Rockville 20850
Tel: (301) 948-6370
TWX: 710-828-9684

MASSACHUSETTS
32 Hartwell Ave.
Lexington 02173
Tel: (617) 861-8960
TWX: 710-326-6904

MICHIGAN
23855 Research Drive
Farmington Hills 48024
Tel: (313) 476-6400
TWX: 810-242-2900

MINNESOTA
2400 N. Prior Ave.
Roseville 55113
Tel: (612) 636-0700
TWX: 910-563-3734

MISSISSIPPI
***Jackson**
Medical Service only
Tel: (601) 982-9363

MISSOURI
11131 Colorado Ave.
Kansas City 64137
Tel: (816) 763-8000
TWX: 910-771-2087

148 Weldon Parkway
Maryland Heights 63043
Tel: (314) 567-1455
TWX: 910-764-0830

NEBRASKA
Medical Only
7171 Mercy Road
Suite 110
Omaha 68106
Tel: (402) 392-0948

NEW JERSEY
W. 120 Century Rd.
Paramus 07652
Tel: (201) 265-5000
TWX: 710-990-4951

NEW MEXICO
P.O. Box 11634
Station E
11300 Lomas Blvd., N.E.
Albuquerque 87123
Tel: (505) 292-1330
TWX: 910-989-1185

156 Wyatt Drive
Las Cruces 88001
Tel: (505) 526-2485
TWX: 910-983-0550

NEW YORK
6 Automation Lane
Computer Park
Albany 12205
Tel: (518) 458-1550

201 South Avenue
Poughkeepsie 12601
Tel: (914) 454-7330
TWX: 510-248-0012

39 Saginaw Drive
Rochester 14623
Tel: (716) 473-9500
TWX: 510-253-5981

5858 East Molloy Road
Syracuse 13211
Tel: (315) 454-2486

1 Crossways Park West
Woodbury 11797
Tel: (516) 921-0300
TWX: 710-990-4951

NORTH CAROLINA
P.O. Box 5188
1923 North Main Street
High Point 27262
Tel: (919) 885-8101

OHIO
16500 Sprague Road
Cleveland 44130
Tel: (216) 243-7300
TWX: 810-423-9431

330 Progress Rd.
Dayton 45449
Tel: (513) 859-8202
TWX: 810-474-2818

1041 Kingsmill Parkway
Columbus 43229
Tel: (614) 436-1041

OKLAHOMA
P.O. Box 32008
Oklahoma City 73132
Tel: (405) 721-0200

OREGON
17890 SW Lower Boones
Ferry Road
Tualatin 97062
Tel: (503) 620-3350

PENNSYLVANIA
111 Zeta Drive
Pittsburgh 15238
Tel: (412) 782-0400
TWX: 710-795-3124

1021 8th Avenue
King of Prussia Industrial Park
King of Prussia 19406
Tel: (215) 265-7000
TWX: 510-660-2670

SOUTH CAROLINA
6941-0 N. Trenholm Road
Columbia 29260
Tel: (803) 782-6493

TENNESSEE
***Knoxville**
Medical Services only
Tel: (615) 523-5022

1473 Madison Avenue
Memphis 38104
Tel: (901) 274-7472

Nashville
Medical Service only
Tel: (615) 244-5448

TEXAS
P.O. Box 1270
201 E. Arapaho Rd.
Richardson 75080
tel: (214) 231-6101

P.O. Box 27409
6300 Westpark Drive
Suite 100
Houston 77027
Tel: (713) 781-6000

205 Billy Mitchell Road
San Antonio 78226
Tel: (512) 434-8241

UTAH
2160 South 3270 West Street
Salt Lake City 84119
Tel: (801) 487-0715

VIRGINIA
Medical Only
P.O. Box 12778
No. 7 Koger Exec. Center
Suite 212
Norfolk 23502
Tel:(804) 497-1026/7

P.O.Box 9854
2914 Hungary Springs Road
Richmond 23228
Tel: (804) 285-3431

WASHINGTON
Bellefield Office Pk.
1203-114th Ave. S.E.
Bellevue 98004
Tel: (206) 454-3971
TWX: 910-443-2446

***WEST VIRGINIA**
Medical/Analytical Only
Charleston
Tel: (304) 345-1640

WISCONSIN
9004 West Lincoln Ave.
West Allis 53227
Tel: (414) 541-0550

FOR U.S. AREAS NOT LISTED:
Contact the regional office
nearest you: Atlanta, Georgia...
North Hollywood, California...
Rockville, Maryland...Rolling Meadows,
Illinois.Their complete
addresses are listed above.

***Service Only**

1/77

HEWLETT hp PACKARD

PART NO. 24307-90014
Printed in U.S.A. 7/74

Sales and service from 172 offices in 65 countries.
5303 Stevens Creek Blvd., Santa Clara, California 95050