X8Ø8Ø

USERS MANUAL

SIERRA DIGITAL SYSTEMS

X8080

USER MANUAL FOR THE
8080 X8 SERIES CROSS-ASSEMBLER ON THE PDP8-E.

APRIL 1976

SIERRA DIGITAL SYSTEMS
1440 WESTFIELD AVE.
RENO, NEVADA 89509
702-329-9548

ALTHOUGH THE INFORMATION IN THIS MANUAL HAS BEEN CHECKED FOR ACCURACY, NO RESPONSIBILITY IS ASSUMED FOR ERRORS. THIS DOCUMENTATION IS SUBJECT TO CHANGE WITHOUT NOTICE.

TABLE OF CONTENTS: SECTION #

****

****

TABLE OF CONTENTS: (CONT.) SECTION #

****

****

1.0.0 INTRODUCTION.

THIS MANUAL DESCRIBES ONE OF THE X8 (CROSS EIGHT) SERIES OF MICRO-PROCESSOR CROSS-ASSEMBLERS SIERRA DIGITAL SYSTEMS HAS DEVELOPED FOR PDP8 USERS. THE X8 SERIES WILL HANDLE ALL OF THE POPULAR MICRO-PROCESSORS WITHIN A UNIVERSAL ASSEMBLER FORMAT. THIS COMMON BASE OF ASSEMBLER DIRECTIVES AND TECHNIQUES IS A SELECTED COMBINATION OF DESIRABLE FEATURES OBSERVED IN A SURVEY OF MANY EXISTING MINI-COMPUTER AND MICROPROCESSOR ASSEMBLERS. THE INSTRUCTION MNEMONICS AND ASSOCIATED SYNTAX OF EACH PARTICULAR MICROPROCESSOR ARE RETAINED UNCHANGED.

THIS MANUAL DESCRIBES THE USAGE OF ONE OF THE MICROPROCESSOR CROSS-ASSEMBLERS FROM THE SIERRA DIGITAL X8 SERIES. IN ORDER TO SIMPLIFY THE LEARNING PROCESS FOR INDIVIDUALS USING MORE THAN ONE CROSS-ASSEMBLER FROM THE SERIES, THIS MANUAL HAS BEEN DIVIDED INTO TWO MAJOR PARTS. SECTIONS 1 THROUGH 11 DOCUMENT THE UNIVERSAL ASSEMBLER FORMAT AS IT APPLIES TO ALL CROSS-ASSEMBLERS IN THE SERIES. THESE SECTIONS WILL BE IDENTICAL IN EVERY CROSS-ASSEMBLER MANUAL. SECTION 12 PRESENTS INFORMATION ON APPLICATION OF THE UNIVERSAL ASSEMBLER FORMAT TO THE SPECIFIC MICROPROCESSOR CROSS-ASSEMBLER. SECTION 13 PRESENTS A SUMMARY OF THE MNEMONIC INSTRUCTION CODES ASSIGNED BY THE MICROPROCESSOR VENDOR AND RECONIZED BY THE CROSS-ASSEMBLER. NO ATTEMPT HAS BEEN MADE TO DESCRIBE THE OPERATION OF THE MICROPROCESSOR ITSELF. SUCH INFORMATION MUST BE OBTAINED FROM THE MICROPROCESSOR VENDOR OR OTHER SOURCES. SECTION 14, THE APPEN-DICES, CONTAINS SUMMARY TABLES FOR QUICK REFERENCE ONCE THE USER GAINS EXPERTISE IN USING THE CROSS-ASSEMBLER.

WE AT SIERRA DIGITAL LOOK FORWARD TO DEVELOPING MORE ASSEMBLERS IN OUR X8 SERIES TO PROVIDE YOU, THE USER, WITH THE MEANS OF PIONEERING THE NEW WORLD OF MICROPROCESSORS.

2.0.0 OPERATION.

SIERRA DIGITAL'S CROSS-ASSEMBLER IS AN 8K, TWO PASS ASSEMBLER WHICH RUNS UNDER THE OS/8 OPERATING SYSTEM. THE CROSS-ASSEMBLER IS CODED IN PDP/8 ASSEMBLY LANGUAGE (PAL8) TO GIVE FAST EXECUTION TIMES. (LESS THAN 30 SECONDS FOR A NORMAL 4K BYTE PROGRAM IS TYPICAL).

PASS 1 READS THE INPUT FILES AND SETS UP THE SYMBOL TABLES. PASS 2 THEN GENERATES THE OUTPUT FILE IN THE BINARY (OBJECT) FORMAT OF THE PARTICULAR MICROPROCESSOR. THE OUTPUT FILE CAN BE CHANGED TO BNPF FORMAT THROUGH USE OF THE /B RUN-TIME OPTION.

A THIRD ASSEMBLY PASS IS DONE WHEN A LISTING OUTPUT FILE IS SPECI-FIED. WHEN NO BINARY FILE IS SPECIFIED, THE ASSEMBLER GOES DIRECTLY TO THE PASS 3 LISTING.

THE CROSS-ASSEMBLER IS NOT RESTARTABLE. IF AN ATTEMPT IS MADE TO RESTART THE ASSEMBLER WITH A .ST COMMAND, THE KEYBOARD MONITOR RETURNS A "NO!!" .

TYPING CTRL/C WILL HALT ASSEMBLY AND CAUSE AN IMMEDIATE EXIT TO THE KEYBOARD MONITOR.

TYPING CTRL/O AT THE KEYBOARD DURING ASSEMBLY WILL SUPPRESS THE LISTING OF ERROR MESSAGES TO THE CONSOLE DURING PASSES 1 AND 2. THE OUTPUT FILE WILL STILL SHOW THE ERROR MESSAGES IMMEDIATELY BEFORE THE LINE THAT IS IN ERROR.

2.1.0 LOADING AND SAVING THE CROSS-ASSEMBLER.

THE CROSS-ASSEMLER IS PROVIDED IN BINARY FORMAT ON PAPER TAPE OR IN BOTH BINARY AND IMAGE FORMATS ON FILE-STRUCTURED MEDIA.

TO LOAD THE ASSEMBLER FROM PAPER TAPE AND SAVE IT, PLACE THE TAPE IN THE READER AND CALL THE ABSOLUTE LOADER:

```
.R ABSLDR
*PTR:$

.SAVE SYS:XNAME
```

FROM FILE STRUCTURED MEDIA, THE IMAGE FORMAT PROGRAM MAY BE COPIED DIRECTLY TO THE SYSTEM DEVICE OR THE BINARY FORMAT FILE MAY BE LOADED WITH THE ABSOLUTE LOADER. MODIFICATIONS TO THE IMAGE FILE, SUCH AS INVERTING THE SENSE OF A RUN-TIME OPTION, MAY BE IMPLEMENTED ACCORDING TO THE NOTES IN SECTION # 11.0.0 .

2.2.0 CALLING SEQUENCE.

ONCE LOADED AND SAVED, THE CROSS-ASSEMBLER IS CALLED FROM THE SYSTEM DEVICE BY TYPING:

```
.R XNAME
```

THE ASSEMBLER CALLS THE COMMAND DECODER WHICH RESPONDS WITH AN ASTERISK IN THE LEFT HAND MARGIN. THE USER MAY THEN TYPE IN THE INPUT AND OUTPUT FILE SPECIFICATIONS AND RUN-TIME OPTIONS:

```
*DEV:BIN,DEV:LIST<DEV:IN1,...DEV:IN9/OPT
```

THE FIRST OUTPUT FILE IS THE MICROPROCESSOR BINARY OBJECT FILE WRITTEN IN THE FORMAT SPECIFIED BY THE VENDOR OF THE PARTICULAR MICROPROCESSOR. (SEE SECTION 12.0.0 FOR THE FORMAT SPECIFICATIONS).

THE SECOND OUTPUT FILE IS THE OPTIONAL LISTING. WHEN ONLY THE FIRST OUTPUT FILE IS SPECIFIED, THE ASSEMBLER ASSUMES THAT IT WILL BE THE BINARY OUTPUT FILE AND THE LISTING IS OMITTED.

THE FOLLOWING EXAMPLE SPECIFIES FILE "IN1" TO BE READ FROM DECTAPE 0 AND THE BINARY (OBJECT) FILE TO BE OUTPUT TO THE PAPER TAPE PUNCH WITH NO LISTING:

```
.R XNAME
*PTP:<DTA0:IN1
```

THIS EXAMPLE SPECIFIES 2 FILES AS THE SOURCE INPUT (FROM THE DSK: DEVICE) WITH ONLY THE PASS 3 LISTING BEING OUTPUT TO THE LINE PRINTER:

```
.R XNAME
*,LPT:<IN1,IN2
```

UP TO NINE INPUT FILES CAN BE SPECIFIED AS ONE PROGRAM WHERE THE LAST FILE IS TERMINATED WITH AN .END STATEMENT.

2.3.0 INPUT/OUTPUT FILE EXTENSIONS.

IF THE EXTENSION TO AN INPUT FILE NAME IS OMITTED, THE ASSEMBLER ASSUMES THE .MS EXTENSION. IF THERE IS NO FILE WITH THAT NAME AND AN .MS EXTENSION, THE ASSEMBLER ASSUMES THE NULL EXTENSION. UNLESS EXTENSIONS ARE SPECIFIED, THE .MB AND .LS EXTENSIONS ARE ADDED TO THE OUTPUT BINARY AND LISTING FILES.

```
.MB     - MICROPROCESSOR BINARY OUTPUT FILE EXTENSION.
.LS     - OUTPUT LISTING FILE EXTENSION.
.MS     - MICROPROCESSOR SOURCE FILE EXTENSION.
```

2.4.0 RUN-TIME OPTIONS.

TABLE #1 DESCRIBES THE OPTIONS WHICH MAY BE SPECIFED AT RUN-TIME IN THE INPUT LINE TO THE COMMAND DECODER.

IF ONE OR MORE OF THESE OPTIONS IS CONTINUALLY CALLED, THE USER SHOULD CONSIDER MODIFYING THE ASSEMBLER TO INVERT THE SENSE OF THE OPTION. THE MODIFICATION NOTES IN SECTION #11.0.0 EXPLAIN HOW THIS MAY BE DONE. FOR EXAMPLE, A USER WHO PREFERS TO OUTPUT FILES IN BNPF FORMAT RATHER THAN BINARY CAN INVERT THE SENSE OF THE /B OPTION. THEN THE BINARY FILES ARE NORMALLY WRITTEN IN BNPF FORMAT. USE OF THE /B OPTION THEN CAUSES THE OUTPUT FILE TO BE WRITTEN IN THE STANDARD MICROPROCESSOR BINARY CODE. SPACE IS PROVIDED IN TABLE #1 TO CHECK OFF WHICH OPTIONS HAVE BEEN INVERTED FOR YOUR REFERENCE.

TABLE #1. RUN-TIME OPTIONS. #2.4.0

OPTION	MEANING	INVERT?
/B	THE BINARY OUTPUT FILE IS WRITTEN IN BNPF FORMAT. INSTEAD OF IN THE MICROPROCESSOR VENDOR'S STANDARD BINARY FORMAT. FOR THE BNPF FORMAT, THE BINARY OUTPUT IS CONVERTED TO ASCII TEXT WHERE "B" INDICATES THE BEGINNNING OF A BYTE, "F" INDICATES THE END OF A BYTE, "P" INDICATES A 1 BIT AND "N" INDICATES A 0 BIT. FOUR BYTES , SEPARATED BY SPACES, ARE WRITTEN PER LINE. THE ADDRESS OF THE FIRST BYTE IS GIVEN IN SIX DIGIT OCTAL AT THE BEINNING OF THE LINE. LEADING ZEROES IN THE ADDRESS ARE CONVERTED TO SPACES. EACH LINE IS PRECEDED BY 2 SPACES. LEADER CONSISTS OF 100 NULL CHARACTERS WITH 20 RUBOUTS IMMEDIATELY PRECEEDING AND FOLLOWING THE ASCII TEXT. EXAMPLE: THE FOLLOWING CODE IS SHOWN REWRITTEN IN BNPF FORMAT. .ORG 100 .BYTE 27,C7,AF,D7,FF,72,0,D0 100 BNNPNNPPPF BPPNNNPPPF BPNPNPPPPF BPPNPNPPPF 104 BPPPPPPPPF BNPPPNNPNF BNNNNNNNNF BPPNPNNNNF	-------
/E	INHIBIT ERROR MESSAGES TO THE CONSOLE. NORMALLY ERROR MESSAGES ARE OUTPUT TO THE CONSOLE DURING ASSEMBLY PASSES 1 AND 2. SINCE ERROR MESSAGES ARE INCLUDED IN THE LISTING, USERS WITH SLOW CONSOLE DEVICES SUCH AS TTY'S CAN SPEED ASSEMBLY TIME WITH THIS OPTION. ALSO, IF THE BINARY FILE IS TO BE OUTPUT TO THE CONSOLE DEVICE, THE ERROR MESSAGES AND BINARY OUTPUT LINES WILL BE INTERMIXED. THE /E OPTION WILL INHIBIT ALL BUT FATAL ERROR MESSAGES SO THAT ONLY THE BINARY FILE IS OUTPUT.	-------

OPTION	MEANING	INVERT?
/H	INHIBIT HEADINGS AND PAGINATION. NORMALLY, THE ASSEMBLER AUTOMATICALLY PAGES THE OUTPUT, ADDING A HEADER TO THE TOP OF THE PAGE. USE OF THE /H OPTION WILL ELIMINATE THE HEADING AND THE PAGINATION.	-------
/J	LIST UNASSEMBLED STATEMENTS AND CONDITIONAL ASSEMBLY PSEUDO-OPS. STATEMENTS WHICH DO NOT GET ASSEMBLED DUE TO CONDITIONAL ASSEMBLY PSEUDO-OPS ARE NORMALLY NOT LISTED. NEITHER ARE THE CONDITIONAL PSEUDO-OPS THEMSELVES. USE OF THE /J OPTION WILL ADD THESE STATEMENTS TO THE LISTING.	-------
/K	EXPAND SYMBOL TABLE STORAGE INTO EXTRA CORE. NORMALLY MOST OF FIELD 1 IS USED FOR BOTH LOCAL AND NORMAL USER SYMBOL STORAGE. USE OF THE /K OPTIONS EXPANDS CORE USAGE TO 12K WHERE THE LOCAL SYMBOL TABLE RESIDES IN FIELD 2 AND THE REGULAR SYMBOL TABLE RESIDES IN FIELD 1.	-------
/L	OUTPUT LEADER IN BINARY FILE FOR .ORG STATEMENTS THIS OPTION MAY BE USED TO PHYSICALLY SEPARATE DISCONTINUOUS SECTIONS OF THE BINARY OUTPUT ON A PAPER TAPE.	-------
/O	OUTPUT LISTING WITH BINARY CODE IN OCTAL FORMAT. THE GENERATED BINARY CODE IS NORMALLY PRINTED IN HEXADECIMAL AT THE LEFT OF THE PROGRAM STATEMENTS IN THE LISTING FILE. THE /O OPTION WILL CAUSE THE BINARY CODE TO BE LISTED IN OCTAL INSTEAD OF HEXADECIMAL.	-------
/N	LIST ONLY THE SYMBOL TABLE. THE THIRD PASS LISTING NORMALLY CONSISTS OF THE STATEMENT LISTING PLUS THE USER SYMBOL TABLE LISTING. THE /N OPTION CAUSES ONLY THE SYMBOL TABLE TO BE LISTED.	-------
/P	INCLUDE NORMALLY UNLISTED PSEUDO-OPS IN THE LISTING SOME PSEUDO-OPS WILL NOT BE LISTED BY PASS 3 UNLESS THE /P OPTION IS USED.	-------
/S	OMIT THE SYMBOL TABLE FROM LISTING. ONLY THE PROGRAM STATEMENTS ARE LISTED WITH THIS OPTION.	-------

TABLE #1. RUN-TIME OPTIONS. (CONT.) #2.4.0

OPTION	MEANING	INVERT?
/T	REPLACE FORM/FEED WITH 3 CR/LF'S. WHEN LISTING TO A DEVICE SUCH AS A TTY WHICH DOES NOT HAVE A FORM/FEED CONTROL, USE OF THE /T OPTION WILL REPLACE THE FORM/FEED WITH 3 BLANK LINES .	-------
/W	INHIBIT WARNING MESSAGES. WHEN WARNING MESSAGES CAN BE SAFELY IGNORED, THIS OPTION WILL PREVENT THEM FROM BEING OUTPUT.	-------
/0 TO /9	USER FLAGS, USED WITH THE ? OPERATOR, SEE SECTION # 8.1.4 .	

3.0.0 ASSEMBLER CHARACTER SET.

THE FOLLOWING CHARACTERS ARE LEGAL SOURCE CODE CHARACTERS:

1) ALPHABETICS A-Z, UPPER CASE ASCII
2) NUMERICS 0-9
3) THE SPECIAL CHARACTERS LISTED BELOW.

*	MULTIPLICATION
/	DIVISION
&	BOOLEAN AND
!	INCLUSIVE OR
+	ADDITION
-	SUBTRACTION
[ ]	PRECEDENCE INDICATORS
^	UNIVERSAL UNARY OPERATOR (UPARROW). USED WITH: ^C - COMPLEMENT (UPARROW C) ^B - BINARY RADIX INDICATOR (UPARROW B) ^D - DECIMAL RADIX INDICATOR (UPARROW D) ^H - HEXADECIMAL RADIX INDICATOR (UPARROW H) ^O - OCTAL RADIX INDICATOR (UPARROW O) ^L - LEAST SIGNIFICANT BYTE ACCESS OPERATOR ^M - MOST SIGNIFICANT BYTE ACCESS OPERATOR
;	COMMENT INDICATOR
" OR '	ASCII INDICATOR
?	USER FLAG OPERATOR
.	CURRENT LOCATION COUNTER (PERIOD)

THE CARRIAGE RETURN CHARACTER IS RECOGNIZED AS THE TERMINATOR FOR EACH SOURCE LINE. THE LINE-FEED, RUBOUT, FORM-FEED, AND NULL CHARACTERS ARE IGNORED BY THE ASSEMBLER. FORM-FEED CHARACTERS OCCURING IN THE SOURCE HAVE NO AFFECT ON THE LISTING. ALL ASCII CHARACTERS MAY BE USED IN THE COMMENT FIELD OF A STATEMENT.

4.0.0 STATEMENT FORMAT.

STATEMENTS ARE WRITTEN IN THE GENERAL FORM:

```
        LABEL   OPERATOR        OPERAND         ;COMMENT
```

LABELS MUST START IN COLUMN 1. THEY MAY BE DIRECTLY FOLLOWED WITH AN OPTIONAL COMMA IF DESIRED. THE MODIFICATION NOTES EXPLAIN HOW TO REPLACE THE COMMA WITH ANOTHER DELIMITER SUCH AS A COLON.

OPERATORS MUST BE SEPARATED FROM THE LABEL WITH AT LEAST ONE SPACE OR TAB. WHEN NO LABEL IS PRESENT, THE OPERATOR MAY BEGIN IN ANY COLUMN BEYOND COLUMN 1.

THE OPERAND (IF ANY) MUST BE SEPARATED FROM THE OPERATOR WITH AT LEAST ONE SPACE OR TAB.

THE COMMENT (IF ANY) MUST BE SEPARATED FROM THE OPERAND (OR OPERATOR IF THERE IS NO OPERAND BY A SEMICOLON (;).

AN INPUT LINE MAY BE UP TO 127 CHARACTERS LONG (NOT INCLUDING THE CARRIAGE RETURN). WHEN THE INPUT LINES ARE OUTPUT TO THE LISTING FILE, ANY CHARACTERS AFTER THE 72D COLUMN ARE WRITTEN ON THE NEXT LINE(S) BEGINNING AT THE 25TH COLUMN OF THE FIRST SOURCE LINE (NORMAL COMMENT COLUMN). SEE THE MODIFICATION NOTES IN SECTION #11.0.0 TO ADJUST FOR NARROWER OR WIDER PAGE OUTPUT. THE CARRIAGE RETURN IS A TERMINATOR FOR BOTH THE STATEMENT AND THE LINE. ONLY ONE STATEMENT IS ALLOWED PER 127 CHARACTER LINE.

4.1.0 CODING CONVENTIONS:

ALTHOUGH THE ASSEMBLER WILL ACCEPT PROGRAMS WRITTEN IN FREE FORMAT, THE USE OF TABS MAKES FOR MORE READABLE CODE. TAB STOPS ARE SET EVERY 8 CHARACTERS IN THE LINE SO THAT THE USE OF THE TAB KEY SIMPLIFIES INPUT. GENERALLY:

LABELS	OCCUPY THE FIRST TAB FIELD,	COLUMNS 1 THROUGH 8
OPERATORS	OCCUPY THE SECOND TAB FIELD,	COLUMNS 9 THROUGH 16.
OPERANDS	OCCUPY THE THIRD TAB FIELD,	COLUMNS 17 THROUGH 24.
COMMENTS	OCCUPY THE REMAINING FIELDS,	COLUMNS 25 THROUGH 127.

4.2.0 LABELS.

A LABEL IS A SYMBOL WHICH PRECEDES THE OPERATOR AND MUST FOLLOW THE SYMBOL NAMING CONVENTIONS DESCRIBED IN SECTION # 6.2.0 . IN ALL BUT THE SYMBOL DEFINITION PSEUDO-OPS, (.EQU, .SET, .DINST) THE LABEL IS A LOCATION TAG AND IS EQUAL TO THE VALUE OF THE CURRENT LOCATION COUNTER.

EXAMPLE:

```
      2  1                .ORG     201
      0  6     LABEL1     .EQU      6          ;LABEL1=6
 201  1        LABEL2     .BYTE     1          ;LABEL2=LOCATION TAG=201
```

NOTE THAT A JUMP TO LABEL1 WILL TRANSFER TO ADDRESS 6 WHILE A JUMP TO LABEL2 GOES TO ADDRESS 201.

A LABEL LACKING BOTH AN OPERATOR AND OPERAND IS SET EQUAL TO THE VALUE OF THE NEXT ADDRESS TO BE ASSEMBLED. IF USED AT THE BEGINNING OF THE PROGRAM, IT IS SET EQUAL TO THE VALUE OF THE FIRST ADDRESS. WHEN A SOLITARY LABEL IS FOLLOWED BY AN .ORG STATEMENT, IT RETAINS THE ORIGINAL VALUE ASSIGNED BEFORE THE ORIGIN CHANGE.

4.3.0 OPERATORS.

AN OPERATOR IS A MNEMONIC WHICH INDICATES THE ACTION TO BE PERFORMED AND IS EITHER A PSEUDO-OP OR ONE OF THE MICROPROCESSOR INSTRUCTIONS. PSEUDO-OPS ARE DESCRIBED IN SECTION #9.0.0. THE MICROPROCESSOR INSTRUCTION SET IS DESCRIBED IN SECTION #13.0.0 . THESE OPERATORS SHOULD NOT BE CONFUSED WITH ARITHMETIC OPERATORS USED IN OPERAND EXPRESSIONS.

4.4.0 OPERANDS.

AN OPERAND REPRESENTS THE PART OF THE INSTRUCTION WHICH IS TO BE ACTED ON. IT CAN BE A TERM OR AN EXPRESSION.

THE .BYTE, .DBYTE, AND .ADDR PSEUDO-OPS CAN HAVE MULTIPLE OPERANDS.

REFER TO THE EXPLANATION OF EACH OPERATOR FOR THE PROPER OPERAND FORMAT.

IT SHOULD BE NOTED THAT OPERAND EXPRESSIONS ARE EVALUATED TO A SINGLE NUMERICAL VALUE BY THE ASSEMBLER. BINARY CODE IS NOT GENERATED TO MAKE THE MICROPROCESSOR EVALUATE THE EXPRESSION.

4.5.0 TERMS AND EXPRESSIONS.

A TERM IS A SINGLE VALUE, A CONSTANT OR SYMBOL. THE CURRENT LOCATION COUNTER (REPRESENTED BY A PERIOD) IS CONSIDERED A TERM.

TERMS ARE COMBINED WITH OPERAND ARITHMETIC OPERATORS TO FORM EXPRESSIONS.

EXAMPLE: IN THE INSTRUCTION BELOW THE OPERAND IS AN EXPRESSION WHICH HAS TWO ARITHMETIC OPERATORS AND THREE TERMS.

```
        SYMBOL .EQU     1+NEW * 15
```

16 BIT INTEGER ARITHMETIC IS USED TO EVALUATE EXPRESSIONS.

5.0.0 NUMERIC CONSTANTS.

A CONSTANT IS A NUMERIC VALUE REPRESENTED BY A STRING OF DIGITS. THE DEFAULT RADIX OR TEMPORARY RADIX INDICATORS IDENTIFY THE RADIX OF THE CONSTANT. A CONSTANT WITHOUT ANY TEMPORARY RADIX INDICATOR IS CONSIDERED TO BE IN THE DEFAULT RADIX, WHICH IS INITIALLY HEXADECIMAL.

EXAMPLE: THE HEXADECIMAL NUMBER 16 (22 IN BASE 10) IS STORED IN "VALUE" :

```
    0  16           VALUE   .EQU    16
```

THE MAXIMUM VALUE FOR A CONSTANT IS 65535 (BASE 10 UNSIGNED).

THE MINIMUM VALUE FOR A CONSTANT IS -32768 (BASE 10 SIGNED).

5.1.0 CONSTANTS WITH RADIX INDICATORS.

CONSTANTS IN A BASE DIFFERENT FROM THAT OF THE DEFAULT RADIX CAN BE SPECIFIED THROUGH USE OF THE TEMPORARY RADIX INDICATORS. THESE INDICATORS ARE VERY USEFUL FOR ENTERING INDIVIDUAL CONSTANTS. HOWEVER, IF A LARGE GROUP OF VALUES IN ANOTHER RADIX MUST BE ENTERED, IT IS MORE CONVENIENT TO CHANGE THE DEFAULT RADIX USING THE PSUEDO-OPS DESCRIBED IN SECTION # 9.2.0 .

THE TEMPORARY RADIX INDICATORS ARE:

^B	BINARY
^D	DECIMAL
^H	HEXADECIMAL
^O	OCTAL

THE ^ IS THE UPARROW CHARACTER (UNIVERSAL UNARY OPERATOR).

A HEXADECIMAL CONSTANT WHICH DOES NOT BEGIN WITH A NUMBER SHOULD BE WRITTEN WITH A LEADING ZERO TO DISTINGUISH IT FROM FROM A SYMBOL. A RADIX INDICATOR PRECEDING A SYMBOL IS IGNORED.

EXAMPLE: THE FIRST STATEMENT IS VALID, THE SECOND IS NOT.

```
VALUE   .EQU    ^H0A302         ;VALUE=A302, BASE 16
VALUE   .EQU    ^HA302          ;VALUE = SYMBOL A302
```

SINCE THE SYMBOL A302 MAY NOT EXIST, THE SECOND STATEMENT WILL PROBABLY CAUSE AN UNDEFINED SYMBOL ERROR. TEMPORARY RADIX INDICATORS AFFECT THE NEXT DIGIT STRING IN THE EXPRESSION UNLESS A SYMBOL NAME OR BINARY OPERATOR OCCURS FIRST. IN THAT CASE, THE TEMPORARY RADIX INDICATOR WOULD BE IGNORED. NO ERROR MESSAGE IS GIVEN.

5.2.0 CONSTANTS WITH ASCII INDICATORS.

THE " AND ' INDICATORS ARE USED TO FORM THE 7 BIT ASCII VALUE OF A CHARACTER. THERE ARE FOUR ACCEPTABLE WAYS TO WRITE THE INDICATORS:

```
"A"  OR  "A  OR  'A'  OR  'A      ALL EQUAL 41 (BASE 16).
```

NOTE THAT THE CLOSING QUOTE IS OPTIONAL, BUT IF USED IT MUST MATCH THE OPENING QUOTE. ONLY ONE CHARACTER CAN FOLLOW THE INDICATOR.

THE " IS SPECIALLY HANDLED IN THE .BYTE PSEUDO-OP WHERE IT IS USED TO INPUT TEXT STRINGS. SEE SECTION # 9.3.1 .

6.0.0 SYMBOLS.

THE WORD "SYMBOL" IS USED HERE AS A GENERAL TERM FOR ANY MNEMONIC WHICH IS TO HAVE A VALUE. THIS IS IN CONTRAST TO AN OPERATOR, WHICH IS A MNEMONIC WHICH SPECIFIES A PROCESS.

A LABEL IS A SYMBOL THAT PRECEDES AN OPERATOR IN THE STATEMENT. IF THE LABEL IS USED TO STORE THE VALUE OF THE CURRENT LOCATION COUNTER , IT IS CALLED A LOCATION TAG.

6.1.0 PERMANENT SYMBOLS.

PERMANENT SYMBOLS ARE THE CROSS-ASSEMBLER PSEUDO-OPS AND MICROPROCESSOR OPERATORS. IF NECESSARY, THE .DINST STATEMENT CAN BE USED TO RENAME A MICROPROCESSOR OPERATOR. THE CROSS-ASSEMBLER PSEUDO-OPS CANNOT BE USED IN A .DINST INSTRUCTION. THE TABLES IN THE APPENDICES SUMMARIZE THE PERMANENT SYMBOL SET.

6.2.0 USER DEFINED SYMBOLS.

THESE SYMBOLS CAN BE LOCATION TAGS OR REPRESENT A VALUE.

A SYMBOL IS A STRING OF FROM ONE TO SIX ALPHANUMERIC CHARACTERS DELIMITED BY A NON-ALPHANUMERIC CHARACTER. USER-DEFINED SYMBOLS MUST CONFORM TO THE FOLLOWING RULES:

1) THE CHARACTERS MUST BE LEGAL ALPHA-NUMERICS. (A-Z OR 0-9)
2) THE FIRST CHARACTER MUST BE ALPHABETIC (A-Z).
3) ONLY THE FIRST SIX CHARACTERS ARE USED, ANY OTHERS ARE IGNORED. SYMBOLS ARE STORED IN THE SYMBOL TABLE AND REFERENCED ONLY BY THE FIRST SIX CHARACTERS.
4) A USER-DEFINED SYMBOL CANNOT HAVE THE SAME NAME AS ANY OF THE PERMANENT SYMBOL NAMES. AS THE PERIOD IS CONSIDERED AS PART OF THE ASSEMBLER PSEUDO-OP NAME, A USER-DEFINED SYMBOL WHICH IS IDENTICAL EXCEPT FOR THE LEADING PERIOD IS LEGAL.

6.3.0 LOCAL SYMBOLS.

OFTEN, WHEN PROGRAMMING SHORT SECTIONS OF CODE WHICH INVOLVE NUMEROUS JUMP OR BRANCHING INSTRUCTIONS, THE USER FINDS IT DIFFICULT TO CREATE MEANINGFUL LABELS THAT WILL NOT CONFLICT WITH OTHER SYMBOLS IN THE PROGRAM. IN CASES LIKE THIS, LOCAL SYMBOLS CAN BE USED INSTEAD OF REGULAR SYMBOLS.

LOCAL SYMBOLS HAVE THE FORMAT "$N" WHERE "N" IS A DECIMAL INTEGER FROM 0-255 INCLUSIVE.

LOCAL SYMBOLS MUST BE DEFINED AND REFERENCED WITHIN LOCAL SYMBOL BLOCKS. LOCAL SYMBOL BLOCKS ARE SECTIONS OF THE PROGRAM THAT START ON A STATEMENT HAVING A REGULAR SYMBOL USED AS A LOCATION TAG AND END ON THE STATEMENT JUST BEFORE THE OCCURANCE OF THE NEXT REGULAR SYMBOL LOCATION TAG. NOTE THAT LABELS FOR THE .EQU, .DINST AND .SET PSEUDO-OPS ARE NOT LOCATION TAGS AND DO NOT DELIMIT LOCAL SYMBOL BLOCKS.

THERE IS NO EFFECTIVE LIMIT TO THE SIZE OF A LOCAL SYMBOL BLOCK.

THE SAME LOCAL SYMBOL CAN BE DEFINED AND USED IN AN UNLIMITED NUMBER OF LOCAL SYMBOL BLOCKS.

EXAMPLE:

```
TAG1    .BYTE   "TEXT"  ;SYMBOL BLOCK BEGINS
$1      .EQU    VALUE   ;DEFINE LOCAL $1
$2      .EQU    -1      ;DEFINE LOCAL $2
VALU1   .EQU    $1-$2   ;CALCULATE NEW VALUE
TAG2    .BYTE   "TEXT"  ;NEW SYMBOL BLOCK
$1      .EQU    VALU1   ;DEFINE LOCAL $1
$2      .EQU    -2      ;DEFINE LOCAL $2
VALU2   .EQU    $1*$2   ;CALCULATE NEW VALUE.
TAG3    .BYTE   "TEXT"  ;ENDS SECOND BLOCK
```

7.0.0 CURRENT LOCATION COUNTER.

THE CURRENT LOCATION COUNTER IS INDICATED BY A PERIOD. IT REPRESENTS THE ADDRESS OF THE NEXT BYTE TO BE ASSEMBLED.

THE CURRENT LOCATION COUNTER CANNOT BE USED IN THE LABEL FIELD.

AT THE BEGINNING OF THE SOURCE INPUT THE CURRENT LOCATION COUNTER IS SET TO ZERO. IT CAN BE REASSIGNED THROUGH USE OF THE .ORG PSEUDO-OP.

EXAMPLE:

```
      0  60                    .ORG        60     ;INITIAL ADDRESS
      0   0        VALUE       .EQU         0     ;NO EFFECT ON .
   60 22           TAG         .BYTE       22     ; .  = 60 (BASE 8)
      1  00                    .ORG       100     ; REASSIGN COUNTER
  100 10           TAG1        .BYTE       10     ; .= 100
```

LOCATION TAGS ARE ALWAYS SET EQUAL TO THE VALUE OF THE CURRENT LOCATION COUNTER WHEN THEY ARE ASSEMBLED. IN THE EXAMPLE ABOVE, THE LOCATION TAG "TAG" = 60.

THE CURRENT LOCATION COUNTER IS AUTOMATICALLY UPDATED IN THE ASSEMBLER AS SOON AS THE CURRENT INSTRUCTION IS ASSEMBLED. NOTE THAT IN THE MULTI-OPERAND DATA STORAGE PSEUDO-OPS, (.BYTE, .DBYTE, AND .ADDR) THE LOCATION COUNTER IS CHANGING AS THE OPERANDS ARE ASSEMBLED.

EXAMPLE: THE LOCATION COUNTER IS USED AS AN OPERAND 3 TIMES IN AN .ADDR PSEUDO-OP.

```
      0 20              .ORG      20
   20 20  0             .ADDR     .,.,.
   22 22  0
   24 24  0
   20 20  0
```

THE CURRENT LOCATION COUNTER USES THE FULL ADDRESS RANGE OF THE MICROPROCESSOR.

8.0.0 THE ARITHMETIC OPERATOR SET.

THERE ARE TWO TYPES OF ARITHMETIC OPERATORS: UNARY AND BINARY OPERATORS.

UNARY OPERATORS ACT ON ONLY ONE ITEM, THE TERM OR EXPRESSION FOLLOWING THEM.

BINARY OPERATORS ACT ON TWO ITEMS: THE TERM OR EXPRESSION PRECEEDING THEM AND THE TERM OR EXPRESION FOLLOWING THEM.

8.1.0 UNARY OPERATORS.

THE + (PLUS) AND - (MINUS) UNARY OPERATORS ASSIGN A POSITIVE OR NEGATIVE SIGN TO THE EXPRESSION FOLLOWING THEM. AN EXPRESSION IS ASSUMED TO BE POSITIVE IF NOT OTHERWISE SPECIFIED.

8.1.2 BYTE ACCESS OPERATORS.

THE ^L AND ^M (WHERE ^ IS THE UPARROW CHARACTER) ARE UNARY OPERATORS WHICH PROVIDE ACCESS TO THE LEAST AND MOST SIGNIFICANT 8 BIT BYTES OF THE VALUE OF AN EXPRESSION OR TERM.

EXAMPLE: TO SET "VALUE" EQUAL TO THE MOST SIGNIFICANT BYTE OF 3B61 (BASE 16), THE STATEMENT BELOW IS USED.

```
VALUE   .SET    ^M3B61          ;VALUE = 003B
```

THIS NEXT STATEMENT TAKES THE LEAST SIGNIFICANT BYTE.

```
VALUE   .SET    ^L3B61          ;VALUE = 0061
```

BYTE ACCESS OPERATORS MAY BE COMBINED WITH THE OTHER UNARY OPERATORS AND THE RADIX INDICATORS.

8.1.3 THE COMPLEMENT OPERATOR.

THE ^C (UPARROW C) IS A LOGICAL UNARY OPERATOR WHICH COMPLEMENTS THE EXPRESSION FOLLOWING IT.

EXAMPLE:

```
VALUE   .EQU    ^C7241  ;VALUE = 8DBE
```

THE COMPLEMENT OPERATOR CAN BE COMBINED WITH THE OTHER UNARY OPERATORS AND THE RADIX INDICATORS.

8.1.4. ? OPERATOR.

THIS IS THE USER FLAG OPERATOR, A UNARY OPERATOR USED IN CONJUNCTION WITH THE COMMAND DECODER USER FLAG OPTIONS (/0 TO /9). IT HAS THE FORM ?EXPRESSION AND MAY BE USED IN OPERANDS LIKE ANY OTHER TERM. THE RESULTING VALUE OF THE QUESTION MARK OPERATOR EQUALS 1 IF THE VALUE OF ITS EXPRESSION MATCHES A USER FLAG THAT WAS SPECIFIED TO THE COMMAND DECODER AT RUN-TIME. OTHERWISE IT EQUALS 0. THIS OPERATOR IS USEFUL FOR CONTROLLING CONDITIONAL ASSEMBLY AND LISTING PARAMETERS WITHOUT HAVING TO MODIFY THE SOURCE FILE.

EXAMPLE: THE /2 OPTION WAS SPECIFIED TO THE COMMAND DECODER AT RUN-TIME.

```
.R XNAME
*BIN,LOUT<SOURCE/2
```

THE SOURCE FILE CONTAINS THE FOLLOWING .LIST STATEMENTS:

```
.
.LIST       ?2-1
.
.
.LIST        1
.
```

AT THE FIRST .LIST STATEMENT, THE ?2 TERM EQUALS 1 SINCE /2 WAS SPECIFED AT RUN-TIME. THE OPERAND (?2-1) EQUALS ZERO. THEREFORE LISTING IS INHIBITED UNTIL THE SECOND .LIST INSTRUCTION. AS THE OPERAND VALUE OF THIS STATEMENT IS 1, LISTING IS ENABLED AGAIN. NOTE THAT IF THE /2 OPTION WAS NOT SPECIFIED, THE INSTRUCTIONS AFTER THE FIRST .LIST WOULD BE INCLUDED IN THE "LOUT" FILE LISTING.

8.2.0 BINARY OPERATORS.

SIX SPECIAL CHARACTERS ARE USED TO PERFORM THE FOLLOWING BINARY OPERATIONS:

*	MULTIPLICATION
/	DIVISION
&	BOOLEAN AND
!	INCLUSIVE OR
+	ADDITION
-	SUBTRACTION

THE UNARY OPERATORS TAKE PRECEDENCE OVER THE BINARY OPERATORS DURING ASSEMBLY. THE * AND / OPERATORS ARE EXECUTED NEXT, THEN THE OTHER BINARY OPERATORS FROM LEFT TO RIGHT. BRACKETS, [AND], ARE USED TO CHANGE THE ORDER OF PRECEDENCE WHEN NECESSARY. A [IS A SHIFT/K ON TTY KEYBOARDS, AND A] IS A SHIFT/M.

EXAMPLE: IN THE STATEMENT BELOW THE OPERAND EXPRESSION IS EVALUATED IN THIS ORDER: [A* [-B]] + [[2/D] * [^C [^B101]]]

```
VALUE  .EQU    A*-B+2/D*^C^B101
```

ADDITION AND SUBRACTION ARE ACCOMPLISHED BY TWO'S COMPLEMENT 16 BIT ARITHMETIC. NO CHECKS FOR OVERFLOW ARE MADE.

MULTIPLICATION IS ACCOMPLISHED BY REPEATED ADDITION. NO CHECKS FOR SIGN OR OVERFLOW ARE MADE.

DIVISION IS ACCOMPLISHED BY REPEATED SUBRACTION. THE QUOTIENT IS THE NUMBER OF SUBTRACTIONS PERFORMED. THE REMAINDER IS NOT SAVED. NO CHECKS ARE MADE FOR SIGN. DIVISION BY ZERO RESULTS IN ZERO.

THE BOOLEAN AND FUNCTION (&) IS A BIT BY BIT LOGICAL AND OF TWO NUMBERS:

THE BOOLEAN INCLUSIVE OR (!) IS A BIT BY BIT LOGICAL OR OF TWO NUMBERS.

9.0.0 PSEUDO-OPERATORS.

PSEUDO-OPERATORS ARE INSTRUCTIONS TO THE ASSEMBLER WHICH ALLOW GREATER FLEXIBILTIY IN PROGRAMMING.

A SUMMARY OF THE PSEUDO-OPS AND THEIR FUNCTIONS IS GIVEN IN THE APPENDIX.

9.1.0 ASSIGNMENT PSEUDO-OPS.

ASSIGNMENT PSEUDO-OPS ARE USED TO DEFINE VALUES, INPUT ASCII TEXT AND REASSIGN THE LOCATION COUNTER.

9.1.1 .EQU PSEUDO-OP.

THE .EQU IS USED TO ASSIGN A VALUE TO A SYMBOL. THIS SYMBOL VALUE CANNOT BE CHANGED ONCE DEFINED. .EQU IS USEFUL FOR ASSIGNING NAMES TO LOCATIONS WHICH ARE NOT LOADED BY THE OBJECT CODE.

EXAMPLE:

```
NAME1    .EQU    300*6
```

9.1.2 .SET PSEUDO-OP.

THE .SET IS USED EXACTLY LIKE THE .EQU EXCEPT THAT THE SYMBOL CAN BE REDEFINED WITH ANOTHER .SET AT ANY POINT IN THE PROGRAM:

EXAMPLE: THE FOLLOWING IS PERFECTLY LEGAL FOR A .SET BUT NOT AN .EQU.

```
NAME1    .SET    300*6
NAME1    .SET    22
```

NOTE THAT IT IS GOOD PRACTICE TO USE THE .EQU FOR ASSIGNMENTS RATHER THAN THE .SET EXCEPT (OF COURSE) WHERE THERE IS A SPECIFIC NEED TO REDEFINE A VALUE. THIS HELPS PREVENT THE ACCIDENTAL REDEFINITION OF A VALUE IN A PROGRAM.

9.1.3 .DINST PSEUDO-OP.

THE .DINST IS USED TO GIVE A MICROPROCESSOR OPERATOR ANOTHER NAME. THE ORIGINAL OPERATOR NAME WILL STILL BE VALID. NOTE THAT THE ASSEMBLER PSEUDO-OPS CANNOT BE RENAMED.

EXAMPLE: THE MICROPROCESSOR INSTRUCTION "OPR" IS DEFINED AS "NEWOP". ANY FURTHER REFERENCES TO "NEWOP" IN THE PROGRAM WILL BE TREATED ACCORDING TO THE DEFINITION OF "OPR".

```
NEWOP   .DINST  OPR
```

"NEWOP" IS DEFINED TO BE THE EQUIVALENT TO THE MICROPROCESSOR INSTRUCTION "OPR" AND IS ADDED TO THE OPERATOR SET FOR THE REMAINDER OF THE ASSEMBLY.

REFERENCES TO USER DEFINED OPERATORS ARE NOT ALLOWED TO PRECEDE THEIR .DINST STATEMENT.

ASSEMBLER PSEUDO-OPS CANNOT BE USED IN EITHER THE LABEL OR OPERAND FIELDS OF ANY STATEMENT AND THEREFORE CANNOT BE DEFINED WITH THE .DINST STATEMENT.

LOCAL SYMBOLS CANNOT BE USED IN THE OPERATOR FIELDS, THEREFORE THEY SHOULD NOT BE USED WITH A .DINST STATEMENT.

9.1.4 .ORG PSEUDO-OP.

THE .ORG REASSIGNS THE LOCATION COUNTER.

THE LOCATION COUNTER WILL BE 0 AT THE START OF THE SOURCE INPUT.

THE .ORG OPERAND CANNOT BE FORWARD REFERENCED, (REFERRED TO A LABEL DEFINED FURTHER ON IN THE PROGRAM) AND CANNOT HAVE A LABEL.

9.2.0 DEFAULT RADIX PSEUDO-OPS.

INITIALLY, THE DEFAULT RADIX IS SET TO HEXADECIMAL SO THAT CONSTANTS ARE READ IN AS BASE 16 VALUES. (SEE MODIFICATION NOTES IF ANOTHER INITIAL DEFAULT RADIX IS DESIRED.)

AT ANY POINT IN THE PROGRAM, THE DEFAULT RADIX CAN BE REASSIGNED THROUGH USE OF THESE PSEUDO-OPS:

```
.BIN            ;BINARY RADIX
.DECM           ;DECIMAL RADIX
.HEX            ;HEXADECIMAL RADIX
.OCT            ;OCTAL RADIX
```

THE DEFAULT RADIX PSEUDO-OPS CANNOT HAVE AN OPERAND OR A LABEL.

ADDITIONALLY, THE RADIX OF INDIVIDUAL CONSTANTS CAN BE SPECIFIED BY THE USE OF THE ^B, ^D, ^H AND ^O INDICATORS. SEE SECTION # 5.1.0 . THESE INDICATORS DO NOT CHANGE THE DEFAULT RADIX.

9.3.0 DATA STORAGE PSEUDO-OPS.

THREE PSEUDO-OPS CAN BE USED TO STORE DATA. THEIR FORMAT IS:

```
      LABEL     PSEUDO-OP     OPERAND,OPERAND,....     ;COMMENT
```

THE PSEUDO-OPS CAN HAVE AS MANY OPERANDS AS WILL FIT ON ONE 127 CHARACTER LINE.

EACH OPERAND CAN BE A SYMBOL,CONSTANT, OR EXPRESSION. COMMAS SEPARATE THE OPERANDS.

THE DOUBLE QUOTE (") CHARACTER IS USED DIFFERENTLY IN THE .BYTE COMMAND, BUT THE SINGLE QUOTE (') RETAINS ITS NORMAL FUNCTION.

9.3.1 .BYTE PSEUDO-OP.

THE .BYTE PSEUDO-OP STORES DATA IN SINGLE BYTES OF MEMORY. NUMERICAL BYTE VALUES CAN RANGE FROM -128 TO +255 (DECIMAL). NORMALLY, DOUBLE QUOTES AND SINGLE QUOTES ARE TREATED IDENTICALLY AND ARE USED TO FORM THE ASCII VALUE OF A SINGLE CHARACTER. HOWEVER, IN THE .BYTE PSEUDO-OP , THE DOUBLE QUOTE IS USED TO INDICATE TEXT STRINGS. DATA IS STORED SEQUENTIALLY AS IT IS PROCESSED, LEFT TO RIGHT. A TEXT STRING MUST BE CLOSED WITH A DOUBLE QUOTE.

EXAMPLE: THE ASCII VALUES OF THE TEXT ABC IS STORED:

```
          2  00      .ORG    200
      200 41         .BYTE   "ABC",0,'B
      201 42
      202 43
      203  0
      204 42
```

THESE STATEMENTS WOULD BE INVALID:

```
            .BYTE   'ABC'   ;THE ' IS NOT FOR TEXT STRINGS
            .BYTE   "ABC    ;TEXT MUST END WITH A "
```

9.3.2 .DBYTE PSEUDO-OP.

THE .DBYTE IS SIMILAR TO THE .BYTE EXCEPT THAT IT STORES DOUBLE BYTE QUANTITIES. IT DOES NOT ACCEPT TEXT STRINGS. THE THE MOST SIGNIFICANT BYTE IS STORED FIRST, THEN THE LEAST SIGNIFICANT BYTE.

9.3.3 . ADDR PSEUDO-OP.

THE . ADDR PSEUDO-OP IS THE SAME AS THE . DBYTE PSEUDO-OP EXCEPT THAT THE LEAST SIGNIFICANT BYTE IS STORED FIRST. MANY MICROPROCESSORS USE THIS REVERSED FORMAT FOR ADDRESSES. FOR EXAMPLE:

```
      2 00     . ORG     200
 200  1 32     . DBYTE   ^H3132   ;HEX CONSTANT
 202 32 31     . ADDR    ^H3132   ;REVERSED BYTES
```

9.3.4 . ZERO PSEUDO-OP.

THE . ZERO PSEUDO-OP RESERVES THE NUMBER OF BYTES INDICATED BY THE OPERAND AND SETS THEM TO ZERO.

EXAMPLE: 16 ADDRESSES, 1 TO 10 (BASE 16) ARE ZEROED.

```
         0  1              . ORG       1
     1   0                 . ZERO      10
    11  10                 . BYTE      10
```

ONLY THE FIRST BYTE WILL BE PRINTED IN THE LISTING. THE LOCATION COUNTER IS ADVANCED. THE OPERAND OF . ZERO CANNOT BE FORWARD REFERENCED, (REFERED TO A LABEL DEFINED FURTHER ON IN THE PROGRAM).

9.4.0 LISTING CONTROL DIRECTIVES.

THROUGH USE OF THE . LIST, . PAGE AND . TITLE PSEUDO-OPS, PLUS SEVERAL RUN-TIME OPTIONS, THE SOURCE PROGRAM CAN BE LISTED IN VARIOUS WAYS AT ASSEMBLY TIME.

NORMALLY, THE ASSEMBLER AUTOMATICALLY PAGES THE OUTPUT, ADDING A HEADER AT THE TOP OF THE PAGE. (NOTE THAT PAGE NUMBERS REPRESENT THE LISTING PAGE NUMBERS, NOT INPUT FILE PAGES.)

NOT ALL PSEUDO-OPS ARE LISTED IN THE OUTPUT. THE CONDITIONAL ASSEMBLY AND LISTING CONTROL PSEUDO-OPS ARE NOT LISTED UNLESS THE /P OPTION IS SPECIFED. SEE RUN-TIME OPTIONS # 2.4.0 .

NORMALLY THE STATEMENTS WHICH ARE NOT ASSEMBLED DUE TO CONDITIONAL ASSEMBLY ARE NOT LISTED. USE OF THE /J COMMAND DECODER OPTION WILL ENABLE LISTING OF THESE STATEMENTS PLUS THE NORMALLY UNLISTED CONDITONAL ASSEMBLY PSUEDO-OPS.

THE PAGINATION AND HEADING CAN BE SUPPRESSED THROUGH USE OF THE /H COMMAND DECODER OPTION.

IF THE OUTPUT DEVICE IS ONE WHICH DOES NOT PAGE ON A FORM FEED (A TTY), THE /T DECODER OPTION CAN BE USED TO CHANGE THE FORM FEED (WHICH NORMALLY STARTS A NEW PAGE) TO 3 CARRIAGE RETURN/LINE FEEDS SO THAT PAGES WILL BE SEPARATED BY 3 BLANK LINES IN THE LISTING.

WARNING MESSAGES ARE NORMALLY OUTPUT TO BOTH THE TERMINAL AND THE SOURCE LISTING. TO INHIBIT THESE MESSAGES, THE /W DECODER OPTION IS USED.

9.4.1 .LIST PSEUDO-OP.

A LIST FLAG IS USED DURING ASSEMBLY TO INDICATE WHETHER OR NOT THE STATEMENTS ARE TO BE LISTED. INITIALLY, THE FLAG IS ON AND STAYS ON UNLESS A .LIST PSEUDO-OP IS ENCOUNTERED.

A .LIST PSEUDO-OP CAN BE USED WITH OR WITHOUT AN OPERAND. A LABEL CANNOT BE USED WITH THE .LIST PSEUDO-OP.

WHEN A .LIST PSEUDO-OP WITHOUT AN OPERAND IS ENCOUNTERED, THE LIST FLAG IS INVERTED.

EXAMPLE:

```
                                        ;LIST FLAG INITIALLY ON
            .ORG        200             ;LISTED
VALUE       .SET        1               ;LISTED
            .LIST                       ;LIST FLAG OFF
VALU2       .SET        70              ;NOT LISTED
            .LIST                       ;LIST FLAG BACK ON
```

NOTE THAT UNLESS THE /P OPTION IS USED, THE .LIST OPERATOR ITSELF WILL NOT BE LISTED.

WHEN A .LIST PSEUDO-OP WITH AN OPERAND IS ENCOUNTERED, THEN LISTING IS INHIBITED IF THE OPERAND IS EQUAL TO ZERO. (THE LIST FLAG IS SET OFF). IF THE OPERAND IS NOT ZERO, LISTING IS ENABLED. (THE LIST FLAG IS SET ON).

9.4.2 .PAGE PSEUDO-OP.

INSERTING A .PAGE PSEUDO-OP IN THE PROGRAM WILL NORMALLY START A NEW PAGE BEGINNING WITH THE NEXT LINE. (THE .PAGE STATEMENT ITSELF IS NOT NORMALLY LISTED.) IF THE /P COMMAND DECODER OPTION IS USED, THE .PAGE STATEMENT WILL BE THE FIRST LINE OF THE NEW PAGE.

THE /H COMMAND DECODER OPTION INHIBITS THE .PAGE PSEUDO-OP.

THE .PAGE PSEUDO-OP CAN HAVE NO LABEL OR OPERAND.

9.4.3 .TITLE PSEUDO-OP.

THE .TITLE IS USED TO REPLACE THE HEADING WITH UP TO 32 CHARACTERS OF TEXT. ITS FORMAT IS:

```
.TITLE  HEADING OF 32 CHARACTERS
```

THE FIRST CHARACTER AFTER THE .TITLE IS THE PSEUDO-OP DELIMITER WHICH CANNOT BE AN ALPHA-NUMERIC CHARACTER . THE DELIMITER IS CONSIDERED THE FIRST CHARACTER OF THE 32 CHARACTER GROUP AND WILL BE PRINTED OUT. ANY TEXT AFTER 32 CHARACTERS WILL BE IGNORED. TABS CAN BE USED IN THE HEADING.

THE /H COMMAND DECODER OPTION INHIBITS THE .TITLE PSEUDO-OP.

THE /P COMMAND DECODER ENABLES THE LISTING OF THE .TITLE PSEUDO-OP.

A SEMICOLON DOES NOT DELIMIT THE HEADING TEXT. COMMENTS CAN BE MADE ONLY AFTER THE 32 CHARACTER HEADING GROUP.

WHEN PLACED AT THE BEGINNING OF THE PROGRAM, THE .TITLE PSEUDO-OP WILL SET THE HEADING FOR THE FIRST PAGE. THE .TITLE MUST APPEAR BEFORE THE FIRST LINE TO BE LISTED.

EXAMPLE: THE FOLLOWING STATEMENTS WILL CAUSE THE HEADING OF THE FIRST PAGE TO BE "*MAIN PROGRAM".

```
        .TITLE*MAIN PROGRAM
VALUE   .EQU    1
        .LIST   VALUE
```

9.5.0 CONDITIONAL ASSEMBLY PSUEDO-OPERATORS.

THE .IFZERO, .IFNZRO, .IFDEF AND .IFNDEF OPERATORS ARE USED TO PROVIDE FOR THE CONDITIONAL ASSEMBLY IN A PROGRAM, SO THAT GROUPS OF STATEMENTS CAN BE ADDED (OR OMITTED) DURING THE ASSEMBLY PROCESS. EACH IS DESCRIBED INDIVIDUALLY IN THE SECTIONS THAT FOLLOW. ALL HAVE THE GENERAL FORM:

```
        PSEUDO-OP       OPERAND         ;COMMENT
```

EACH OPERAND MUST MEET THE CONDITIONS OF ITS PSEUDO-OP IN ORDER FOR THE STATEMENTS THAT FOLLOW IT TO BE ASSEMBLED. IF THE CONDITIONS ARE NOT MET; THESE STATEMENTS ARE OMITTED. THE .ENDC PSEUDO-OP INDICATES THE END OF THE GROUP OF STATEMENTS WHICH ARE AFFECTED. EACH CONDITIONAL PSEUDO-OP MUST HAVE ONE .ENDC STATEMENT.

CONDITIONAL PSEUDO-OPS CANNOT HAVE LABELS.

CONDITIONAL PSEUDO-OPS CAN BE NESTED UP TO 4095 LEVELS.

EXAMPLE:

```
VALUE1    .EQU        0          ;DEFINE VALUE1
          .IFZERO     VALUE1     ;VALUE1 = 0 ?  - YES.
          .BYTE       "TEXT"     ;ASSEMBLED.
          .IFDEF      VALUE2     ;VALUE2 DEFINED? - NO.
          .BYTE       "TEXT"     ;OMITTED.
          .ENDC                  ;END OF INNER CONDITIONAL
DOC       .EQU        17         ;ASSEMBLED.
          .ENDC                  ;END OF OUTER CONDITIONAL
          .
          .
```

THE CONDITIONAL PSEUDO-OPS ARE NOT INCLUDED IN THE ASSEMBLY LISTING UNLESS THE /P OR /J COMMAND DECODER OPTION IS SPECIFIED.

ONE CONDITIONAL CAN INHIBIT ANOTHER.

EXAMPLE: THREE DIFFERENT RESULTS CAN OCCUR IN THE FOLLOWING TYPE OF CONDITIONAL NESTING:

```
          CONDITIONAL 1
          .                      ;STATEMENT GROUP 1.
          CONDITIONAL 2
          .                      ;STATEMENT GROUP 2.
          .
          .ENDC                  ;END CONDITIONAL 2.
          .                      ;STATEMENT GROUP 3.
          .ENDC                  ;END CONDITIONAL 1.
```

IF BOTH CONDITIONALS ARE MET, ALL THE STATEMENTS, GROUPS 1 THROUGH 3, WILL BE ASSEMBLED.

IF CONDITIONAL 2 IS NOT MET, BUT CONDITONAL 1 IS MET, THEN GROUP 1 AND GROUP 3 WILL BE ASSEMBLED. GROUP 2 IS NOT ASSEMBLED.

IF CONDITIONAL 1 IS NOT MET, CONDITIONAL 2 IS IGNORED AND GROUPS 1 THROUGH 3 WILL NOT BE ASSEMBLED.

9.5.1 . IFZERO PSEUDO-OP.

IF THE OPERAND OF THE . IFZERO IS:

EQUAL TO ZERO	- ASSEMBLY IS UNAFFECTED.
NOT EQUAL TO ZERO	- STATEMENTS TO NEXT . ENDC ARE OMITTED.

THE OPERAND CANNOT BE FORWARD REFERENCED.

9.5.2 . IFNZRO PSEUD-OP.

IF THE OPERAND OF THE . IFNZRO IS:

EQUAL TO ZERO	- STATEMENTS TO NEXT . ENDC ARE OMITTED.
NOT EQUAL TO ZERO	- ASSEMBLY IS UNAFFECTED.

THE OPERAND CANNOT BE FORWARD REFERENCED.

9.5.3 . IFDEF PSEUDO-OP.

IF THE SYMBOL OPERAND OF THE . IFDEF IS:

DEFINED	- ASSEMBLY IS UNAFFECTED.
NOT DEFINED	- STATEMENTS TO NEXT . ENDC ARE OMITTED.

NOTE THAT . IFDEF WILL ACCEPT ONLY A SINGLE SYMBOL NAME AS THE OPERAND.

A SYMBOL IS CONSIDERED TO BE DEFINED IF IT HAS BEEN USED IN THE LABEL FIELD OF A STATEMENT PRECEEDING THE CONDITIONAL PSEUDO-OP.

9.5.4 . IFNDEF PSEUDO-OP.

IF THE SYMBOL OPERAND OF THE . IFNDEF IS:

DEFINED	- STATEMENTS TO NEXT . ENDC ARE OMITTED.
NOT DEFINED	- ASSEMBLY IS UNAFFECTED.

NOTE THAT ONLY A SINGLE SYMBOL NAME IS ALLOWED AS THE OPERAND.

A SYMBOL IS CONSIDERED TO BE DEFINED IF IT HAS BEEN USED IN THE LABEL FIELD OF A STATEMENT PRECEEDING THE CONDITIONAL PSEUDO-OP.

9.5.5 .ENDC PSEUDO-OP.

THIS PSEUDO-OP INDICATES THE END OF A CONDITONAL ASSEMBLY GROUP.

EVERY CONDITIONAL PSUEDO-OP MUST BE PAIRED WITH A .ENDC.

9.6.0 .END PSEUDO-OP.

THIS INDICATES THE END OF THE SOURCE PROGRAM. IT CANNOT HAVE EITHER A LABEL OR AN OPERAND. A WARNING MESSAGE WILL OCCUR IF THE .END STATEMENT IS LEFT OFF.

#10.0.0 ERROR MESSAGES AND WARNINGS.

BOTH PASS #1 AND PASS #2 CAN GENERATE ERROR MESSAGES. THESE ARE PRINTED ON THE CONSOLE DEVICE AS THEY OCCUR. IF A LISTING IS SPECIFIED, PASS 3 WILL LIST THE ERROR MESSAGE ABOVE THE LINE IN WHICH THE ERROR OCCURS.

ERROR MESSAGES WHICH ARE SENT TO THE CONSOLE HAVE THE FORM:

E:XX AT LABEL+N

WHERE "N" IS A DECIMAL NUMBER OF LINES BEYOND THE STATEMENT WHICH CONTAINED THE GIVEN LABEL. IF NO LABEL WAS GIVEN, "N" IS THE NUMBER OF LINES FROM THE BEGINNING LINE OF THE PROGRAM.

IF THE BINARY OUTPUT FILE IS SENT TO THE CONSOLE, AND ERROR MESSAGES OCCUR, THE OUTPUT FILE LINES AND ERROR MESSAGES WILL BE INTERMIXED. USE OF THE /E OPTION WILL INHIBIT THE ERROR MESSAGES TO THE CONSOLE SO THAT ONLY THE BINARY FILE IS OUTPUT. THIS IS USEFUL WHEN A USER WOULD LIKE TO TRY OUT CERTAIN PARTS OF A PROGRAM AND IS NOT YET CONCERNED WITH OTHER PARTS KNOWN TO HAVE ERRORS.

INDIVIDUAL ERROR MESSAGES ARE EXPLAINED IN TABLE #2 WHICH DIVIDES THE MESSAGES INTO THREE TYPES:

1) FATAL ERRORS- THESE ERRORS CAUSE THE IMMEDIATE EXIT TO THE OS/8 MONITOR. THE CURRENT OUTPUT FILE IS NOT CLOSED. /E WILL NOT INHIBIT FATAL ERROR MESSAGES. FATAL ERROR MESSAGES ARE ALWAYS SENT TO THE CONSOLE DEVICE.

2) WARNING MESSAGES INDICATE MINOR PROGRAM PROBLEMS. ASSEMBLY IS NOT HALTED. GOOD PROGRAMMING PRACTICES WILL ELIMINATE ALL WARNING MESSAGES.

3) NON-FATAL ERRORS - THE OCCURANCE OF A NON-FATAL ERROR WILL NOT HALT ASSEMBLY. THE ASSEMBLER ATTEMPTS TO DO AS MUCH OF THE LINE AS POSSIBLE. FOR EXAMPLE, IF THE OPERAND CANNOT BE EVALUATED, IT GIVES IT A VALUE OF ZERO, WRITES THE ERROR MESSAGE AND CONTINUES.

TABLE #2. #10.0.0

****

**** FATAL ERRORS ****

E:DF - DEVICE FULL:
FILE #N THERE IS NOT ENOUGH ROOM LEFT ON THE OUTPUT DEVICE FOR THE FILE. "N" INDICATES WHICH OF THE TWO OUTPUT FILES WAS IN ERROR.

E:LT - LOCAL SYMBOL TABLE OVERFLOW:
THIS ERROR OCCURS ONLY IF THE /K OPTION IS IN USE. CONVERSION OF SOME OF THE LOCAL SYMBOLS TO REGULAR SYMBOL NAMES WILL USUALLY SOLVE THIS PROBLEM. SEE THE NOTES ON THE /K RUN-TIME OPTION.

E:OE - OPEN ERROR IN OUTPUT FILE:
FILE #N AN ATTEMPT WAS MADE TO OPEN AN OUTPUT FILE ON AN INPUT-ONLY DEVIDE. (PTR:, CDR:, ETC.) "N" INDICATES WHICH ONE OF THE TWO POSSIBLE OUTPUT FILES WAS IN ERROR.

E:PE - PHASE ERROR:
A LOCATION TAG HAS A DIFFERENT ADDRESS IN ONE PASS THAN IT HAD IN THE PREVIOUS PASS.

E:RE - READ ERROR:
FILE #N AN ERROR HAS OCCURRED WHILE READING FROM AN INPUT FILE DEVICE. "N" INDICATES WHICH ONE OF THE NINE POSSIBLE INPUT FILES HAD THE ERROR.

E:ST - SYMBOL TABLE OVERFLOW:
THE PROGRAM IS TOO LARGE. WHERE CONVENIENT, DIVIDE IT AND ASSEMBLE EACH PART SEPARATELY. ALSO REFER TO THE NOTES ON THE /K RUN-TIME OPTION.

E:WE - WRITE ERROR:
FILE #N AN ERROR HAS OCCURRED WHILE WRITING TO AN OUTPUT FILE DEVICE. "N" INDICATES WHICH ONE OF THE TWO OUTPUT FILES HAD THE ERROR.

**** WARNING MESSAGES ****

W:EF - NO .END STATEMENT:
THE LAST INPUT FILE MUST HAVE AN .END STATEMENT. THE ASSEMBLER PROCEEDS AS IF AN .END WERE PRESENT.

W:UC - ASSEMBLY WAS CONDITIONALLY INHIBITED AT THE END OF THE PROGRAM: EACH CONDITIONAL ASSEMBLY PSEUDO-OP MUST BE PAIRED WITH AN .ENDC STATEMENT.

****

****

**** NON-FATAL ERRORS ****

E:BN – BAD NESTING OF BRACKETS:
EACH OPEN BRACKET MUST BE PAIRED WITH A CLOSED BRACKET.

E:DR – DIGIT OUTSIDE OF RADIX:
THE CONSTANT CONTAINS A DIGIT NOT RECOGNIZED UNDER THE SPECIFIED RADIX. FOR EXAMPLE, THE DIGIT "2" IS NOT RECOGNIZED IN BINARY RADIX. THE CONSTANT WILL BE EVALUATED AS IF THAT DIGIT WERE ZERO.

E:IL – ILLEGAL LABEL FIELD:
THE LABEL MAY NOT BE IN THE PROPER SYMBOL FORMAT, SEE SECTION #6.2.0 . ALSO, SOME PSEUDO-OPS CANNOT HAVE LABELS.

E:IO – ILLEGAL OPERAND VALUE:
REFER TO THE SECTION ON THE STATEMENT'S OPERATOR TO DETERMINE THE ALLOWABLE OPERAND TERMS.

E:LO – LINE INPUT OVERFLOW:
ONLY 127 CHARACTERS, NOT INCLUDING THE CARRIAGE RETURN AND LINE FEED, ARE ALLOWED IN AN INPUT LINE.

E:LS – LOCAL SYMBOL SYNTAX ERROR:
THE CORRECT FORMAT FOR A LOCAL SYMBOL IS $N WHERE "N" IS A DECIMAL NUMBER FROM 0 TO 255.

E:ML – MULTIPLE LABEL DEFINITION:
THE SAME LABEL HAS A DIFFERENT VALUE AND IS USED WITH AN OPERATOR OTHER THAN A .SET PSEUDO-OP.

E:MO – MISSING OR ILLEGAL MNEMONIC IN OPERATOR FIELD:

E:OC – OPERAND TOO COMPLEX:
TOO MANY TERMS AND OPERATORS EXIST IN THE OPERAND. DIVIDE THE EXPRESSION USING THE .SET COMMAND.

EXAMPLE: THE FIRST EXPRESSION IS DIVIDED INTO THE TWO STATEMENTS FOLLOWING IT.

```
WORD        .EQU        [ EXPR1 ] + [ EXPR2 ]

TEMP        .SET        [ EXPR1 ]
WORD        .EQU        TEMP + [ EXPR2 ]
```

E:OM – OPERAND MISSING.

****

****

E:OS - OPERAND SYNTAX ERROR.

E:PS - ILLEGAL PERMANENT SYMBOL USAGE IN OPERAND:
REFER TO THE APPENDICES TABLES TO SEE WHICH NAMES ARE USED IN THE ASSEMBLER AND MICROPROCESSOR INSTRUCTION SETS AND RENAME YOUR SYMBOL SO THAT IT WILL NOT CONFLICT.

E:TL - LABEL DEFINED TOO LATE:
ONLY ONE LEVEL OF FORWARD REFERENCING IS ALLOWED.

E:US - UNDEFINED SYMBOL:

****

NOTE: REFER TO SECTION #12.0.0 FOR ADDITIONAL ERROR MESSAGES WHICH ARE SPECIFIC TO THE TYPE OF MICROPROCESSOR BEING USED.

#11.0.0 MODIFICATION NOTES.

VARIOUS MODIFICATIONS CAN BE MADE TO THE ASSEMBLER FOR GREATER OPERATING CONVENIENCE. BEFORE MAKING ANY CHANGES, THE USER SHOULD READ THE DESCRIPTION OF EACH OPTION CAREFULLY. NO CHECKS ON PATCH VALIDITY ARE MADE. ALSO KEEP A RECORD OF ALL CHANGES SO THAT THE STATUS OF THE CROSS-ASSEMBLER IS ALWAYS KNOWN.

MODIFICATIONS ARE MADE BY PATCHING LOCATIONS IN THE IMAGE (.SV) FILE USING ODT. REFER TO THE OS/8 MANUAL FOR A DETAILED EXPLANATION OF ODT OPERATION.

THE EXAMPLE BELOW SHOWS AN ODT PATCH BEING MADE TO FILE "XNAME.SV" WHERE THE CONTENT OF LOCATION 10107 IS CHANGED FROM 3 TO 2.

```
.GET SYS:XNAME
.ODT
10107/0003 2
^C
.SA SYS:XNAME
```

#11.1.0 CHANGING THE DEFAULT INPUT FILE EXTENSION (.MS).

PATCH LOCATION 10100 TO CONTAIN THE NEW 2 CHARACTER 6 BIT ASCII EXTENSION.

#11.2.0 CHANGING THE DEFAULT BINARY OUTPUT FILE EXTENSION (.MB)

PATCH LOCATION 10101 TO CONTAIN THE NEW 2 CHARACTER 6 BIT ASCII EXTENSION.

#11.3.0 CHANGING THE DEFAULT LISTING OUTPUT FILE EXTENSION (.LS).

PATCH LOCATION 10102 TO CONTAIN THE NEW 2 CHARACTER 6 BIT ASCII EXTENSION.

#11.4.0 CHANGING THE BASE YEAR DATE.

IN OS/8 ONLY 3 BITS ARE PROVIDED TO INDICATE THE CURRENT YEAR. THIS ALLOWS ONLY NUMBERS FROM 0 TO 7 WHICH MUST BE ADDED TO A BASE YEAR TO FORM THE ACTUAL YEAR NUMBER. IN 1978 AND AT ADDITIONAL 8 YEAR INTERVALS THE BASE YEAR MUST BE CHANGED TO PROVIDE THE PROPER DATE PRINTOUT. TO DO THIS, PATCH LOCATION 10104 TO CONTAIN THE TWO CHARACTER 6 BIT ASCII REPRESENTATION OF THE TWO LEAST SIGNIFICANT DIGITS OF THE YEAR.

BASE YEAR:	PATCH TO LOCATION 10104 (IN OCTAL).
1978	6770
1986	7066
1994	7164
2002	6062

SHOULD THIS PROGRAM SURVIVE UNTIL THE YEAR 2000 THE TWO MOST SIGNIFICANT DIGITS MAY BE CHANGED BY PATCHING LOCATION 10103 TO CONTAIN 6260.

#11.5.0 CHANGING THE DEFAULT RADIX. (HEXADECIMAL)

INITIALLY THE DEFAULT RADIX IS SET TO HEXADECIMAL. THIS MAY BE MODIFIED TO BINARY, OCTAL, OR DECIMAL BY PATCHING LOCATION 10105 FROM THE FOLLOWING TABLE.

RADIX:	PATCH LOCATION 10105 TO:
OCTAL	1
HEXADECIMAL	2
DECIMAL	3
BINARY	4

#11.6.0 GENERATING 8 BIT ASCII CHARACTERS WITHIN THE BINARY PROGRAM.

THE ASCII CHARACTERS GENERATED AS OPERANDS WITH THE QUOTE CHARACTERS ARE SEVEN BIT REPRESENTATIONS TYPICAL OF MOST MICROPROCESSOR SYSTEMS. TO GENERATE EIGHT BIT ASCII WITH THE EIGHTH BIT ALWAYS SET (AS IS DONE IN SOME PDP8 SOFTWARE), PATCH LOCATION 10106 TO CONTAIN 377. (ORIGINAL CONTENT WAS 177).

#11.7.0 RUNNING UNDER OS8 VERSION 2.

THE CROSS-ASSEMBLER IS SET UP TO USE THE OS/8 VERSION 3 METHOD FOR CORE SIZE DETERMINATION. IN OS/8 V3 THE CORE SIZE IS CONTAINED IN A MONITOR LOCATION. IN PREVIOUS VERSIONS, THE CORE SIZE MUST BE DETERMINED BY ACCESSING EACH FIELD OF MEMORY TO SEE IF IT EXISTS ON THE SYSTEM. THEREFORE , TO RUN THE CROSS-ASSEMLER UNDER VERSION 2, PATCH LOCATION 10107 TO CONTAIN 2. (ORIGINAL CONTENT WAS 3).

#11.8.0 CHANGING THE NUMBER OF LINES PER PAGE. (6)

THE NORMAL NUMBER OF LINES PER PAGE IS SET AT 66. 6 OF THE 66 LINES ARE USED BY THE ASSEMBLER FOR THE HEADING AND MARGIN. TC ALTER THE NUMBER OF LINES ON A PAGE, PATCH LOCATION 10110 TO BE THE TOTAL POSITIVE LINES PER PAGE INCLUDING HEADING AND MARGIN.

#11.9.0 CHANGING THE NUMBER OF CHARACTERS PER LINE. (72)

THE TOTAL NUMBER OF CHARACTERS PRINTED ON ONE LINE (EXCLUDING CARRIAGE RETURN AND LINE FEED) IS SET AT 72 (BASE 10). TO MODIFY THIS COUNT, PATCH LOCATION 10111 TO CONTAIN THE POSITIVE NUMBER OF CHARACTERS TO BE PRINTED ON A LINE (EXCLUDING THE CR AND LF).

#11.10.0 INITIAL FORM/FEED CONTROL.

SOME LINE PRINTER HANDLERS WHEN FIRST INITIALIZED WILL ISSUE AN AUTOMATIC FORM FEED. TO AVOID EJECTING AN ADDITIONAL PAGE EACH TIME THE ASSEMBLER IS CALLED, THE FIRST FORM FEED FROM THE HEADING HAS BEEN SUPPRESSED. TO REENABLE THIS FIRST FORM FEED, PATCH LOCATION 10112 WITH 214 (BASE 8).

#11.11.0 CHANGING LABEL DELIMINATOR (,).

TO PROVIDE COMPATIBILITY WITH OTHER ASSEMBLER FORMATS AN OPTIONAL LABEL DELIMITER WILL BE ACCEPTED. NORMALLY, THIS DELIMITER IS A COMMA, BUT IT CAN BE MODIFIED TO ANY OTHER NON-ALPHANUMERIC CHARACTER (EXCEPT THE SEMICOLON OR CARRIAGE RETURN). TO MODIFY THE DELIMITING CHARACTER PATCH LOCATION 10113 WITH THE 8 BIT ASCII VALUE FOR THE CHARACTER.

#11.12.0 CHANGING FROM 8 BIT TO 7 BIT ASCII IN THE OUTPUT FILES.

ALL ASCII OUTPUT TO THE BINARY (OBJECT) AND LISTING FILES IS IN 8 BIT ASCII FORMAT. TO OUTPUT 7 BIT ASCII FORMAT PATCH LOCATION 10114 TO CONTAIN 177. (ORIGINAL CONTENT WAS 377).

#11. 13. 0 CHANGING THE SENSE OF THE RUN-TIME OPTIONS.

EACH SLASH OPTION (EXCEPT /0 TO /9) MAY HAVE ITS SENSE INVERTED BY PATCHING THE LOCATIONS SHOWN IN THE FOLLOWING TABLE WITH THE DESCRIBED VALUE.

OPTION:	LOCATION:	STANDARD:	INVERTED:
/B	10116	7650	7640
/E	10117	7640	7650
/H	10120	7650	7640
/J	10121	7650	7640
/K	10122	7650	7640
/L	10123	0	1
/N	10124	7650	7640
/O	10125	7650	7640
/P	10126	7640	7650
/S	10127	7650	7640
/T	10130	7650	7640
/W	10131	7650	7640

#12.0.0 8080 CROSS-ASSEMBLER SPECIFICS.

THE FIRST ELEVEN SECTIONS OF THIS MANUAL HAVE PRESENTED SIERRA DIGITAL'S UNIVERSAL ASSEMBLER FORMAT AS IT IS APPLIED TO ALL CROSS-ASSEMBLERS IN THE X8 SERIES. THIS SECTION PRESENTS ADDITIONAL INFORMATION ON THE APPLICATION OF THE UNIVERSAL ASSEMBLER FORMAT TO A SPECIFIC CROSS-ASSEMBLER FOR THE 8080 MICROPROCESSOR. THE 8080 MICROPROCESSOR WAS DESIGNED BY INTEL CORPORATION, 3065 BOWERS AVENUE, SANTA CLARA, CALIFORNIA 95051. VERSIONS OF THE 8080 USING THE SAME INSTRUCTION SET ARE ALSO PRODUCED BY ADVANCED MICRO DEVICES INC., 901 THOMPSON PLACE, SUNNYVALE, CALIFORNIA 94086; TEXAS INSTRUMENTS INC., HOUSTON, TEXAS 77001; AND NEC MICROCOMPUTERS INC, 5 MILITIA DRIVE, LEXINGTON MASS. 02173. NO ATTEMPTS WILL BE MADE IN THIS MANUAL TO EXPLAIN THE OPERATION OF THE MICROPROCESSOR. EXCELLENT MANUALS COVERING THE OPERATION AND PROGRAMMING OF THE MICROPROCESSORS ARE AVAILABLE FROM THEIR MANUFACTURERS. SECTION #13 PRESENTS A SUMMARY OF THE INSTRUCTION MNUEMONIC CODES DEFINED BY INTEL AND RECOGNIZED BY OUR CROSS-ASSEMBLER.

#12.1.0 CROSS-ASSEMBLER FILE NAMES.

THE CROSS-ASSEMBLER IS PROVIDED ON FILE STRUCTURED MEDIA UNDER THE NAMES:

X8080.SV	- FOR THE OS/8 SAVE IMAGE FILE.
X8080.BN	- FOR THE OS/8 BINARY FORMAT FILE.

IT IS SUGGESTED THAT THE SAME NAMING CONVENTIONS BE USED WHEN LOADING THE CROSS-ASSEMBLER FROM PAPER TAPE.

#12. 2. 0 PREASSIGNED SYMBOL VALUES.

THE SYMBOLS REPRESENTING THE REGISTERS, MEMORY, STACK POINTER, AND PROGRAM STATUS WORD HAVE BEEN GIVEN STANDARD VALUES AS SHOWN IN THE FOLLOWING TABLE:

SYMBOL	VALUE	REPRESENTS
A	7	REGISTER A
B	0	REGISTER B
C	1	REGISTER C
D	2	REGISTER D
E	3	REGISTER E
H	4	REGISTER H
L	5	REGISTER L
M	6	MEMORY REFERENCE
SP	16	STACK POINTER
PSW	26	PROGRAM STATUS WORD

ALTHOUGH NOT SUGGESTED, THESE SYMBOLS MAY BE USED TO REPRESENT THEIR CONSTANTS AT ANY POINT IN AN OPERAND IN ADDITION TO THEIR NORMAL USAGE AS REGISTER INDICATORS. ALSO LEGAL BUT NOT SUGGESTED IS THE USE OF THE APPROPRIATE CONSTANT IN PLACE OF THE REGISTER INDICATOR IN A STATEMENT. THUS THE STATEMENT 'MVI 2,A' MEANS LOAD REGISTER C WITH THE IMMEDIATE VALUE OF 7.

#12. 3. 0 LISTING FORMAT.

THE LISTING FILE IS OUTPUT WITH THE OBJECT CODE PRINTED TO THE LEFT OF THE SOURCE CODE LINES. AS EACH MICROPROCESSOR INSTRUCTION MAY CODE INTO ONE, TWO, OR THREE BYTES, ROOM IS PROVIDED FOR THREE COLUMNS OF GENERATED OBJECT CODE PLUS A COLUMN FOR THE ADDRESS. THE ADDRESS AND OBJECT CODE ARE NORMALLY PRINTED IN HEXADECIMAL BUT THIS MAY BE CHANGED TO OCTAL WITH THE /O COMMAND DECODER OPTION. SOURCE LINES WHICH EXCEED THE PRINTOUT LIMIT WILL CONTINUED AT COLUMN 25 (STANDARD COMMENT TAB STOP) OF THE SOURCE PRINTOUT POSITION. TABS OCCURING IN THE SOURCE PROGRAM ARE CONVERTED TO THE PROPER NUMBER OF BLANK CHARACTERS BY THE ASSEMBLER. THIS IS DONE BY THE ASSEMBLER RATHER THAN THE DEVICE HANDLER OR DEVICE BECAUSE THE BEGINNING OF THE SOURCE PRINTOUT DOES NOT OCCUR ON A STANDARD TAB STOP.

#12.4.0 BINARY FILE OUTPUT:

THE OBJECT (BINARY) OUTPUT FILE CONSISTS OF ASCII TEST REPRESENTING HEXADECIMAL NUMBERS IN THE FOLLOWING FORMAT:

LEADER STRINGS OF 100 NULL CHARACTERS PRECEED AND FOLLOW THE OBJECT OUTPUT. EACH LINE BEGINS WITH A COLON AND IS FOLLOWED BY A TWO HEX DIGIT ADDRESS, A TWO HEX DIGIT RECORD TYPE (ALWAYS 0), UP TO 16 BYTES OF DATA (EACH 2 HEX DIGITS), AND A TWO HEX DIGIT CHECKSUM.

EXAMPLE:

:CCAAAATTDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDSS

WHERE:

CC	IS THE TWO HEXADECIMAL DIGIT COUNT FOR THE NUMBER OF DATA BYTES (REPRESENTED BY PAIRS OF D'S) IN THE LINE. A COUNT OF ZERO INDICATES THE TERMINATION OF THE OBJECT OUTPUT. (:00)
AAAA	IS THE HEXADECIMAL ADDRESS FOR STORING THE FIRST DATA BYTE. EACH ADDITIONAL DATA BYTE IS TO BE STORED IN SEQUENTIAL ADDRESSES. THE ADDRESS IS PRESENTED WITH ITS MOST SIGNIFICANT BYTE FIRST.
TT	IS THE TWO HEXADECIMAL RECORD TYPE. THIS INDICATOR IS CURRENTLY UNUSED AND ASSIGNED A VALUE OF 00.
DD	REPRESENTS TWO HEXADECIMAL DIGITS FOR A BYTE OF OBJECT (BINARY) CODE. UP TO 16 BYTES MAY BE OUTPUT ON ONE LINE.
SS	IS THE TWO HEXADECIMAL DIGIT CHECKSUM OF THE LINE. ALL EIGHT BIT BYTES IN THE LINE AFTER THE RECORD MARK (':') ARE SUMMED. THE LEAST SIGNIFICANT BYTE OF THE NEGATIVE OF THIS VALUE IS THE CHECKSUM. THUS, IF ALL BYTES IN THE LINE ARE ADDED TOGETHER WITH CARRYS IGNORED, AND THIS SUM IS ADDED TO THE CHECKSUM, THE RESULT WILL BE ZERO.

THE BINARY OUTPUT FILE CAN BE CHANGED TO BNPF FORMAT THROUGH THE USE OF THE /B RUN-TIME OPTION. SECTION #2.4.0 DESCRIBES THE BNPF OUTPUT.

#12.5.0 ADDITIONAL ERROR MESSAGE FOR THE 8080:

STANDARD ERROR:

E:RV BAD REGISTER VALUE FIELD.
THE VALUE ASSIGNED TO A REGISTER SPECIFICATION FIELD DID NOT MATCH ONE OF THE ALLOWABLE VALUES FOR THE INSTRUCTION. (NOTE THAT THE ERROR MAY ALSO OCCUR FOR A 'RST' PARAMETER OUT OF THE RANGE 0 TO 7).

#12.6.0 SAMPLE PROGRAM #12.6.0

```
.R X8080
*TTY:,TTY:<SAMPLE/1/P/J
  E:MO AT POSITN+  21
  E:MO AT POSITN+  21

:10100000DB00A7F20010E60F0F4FDB01A7FA1E105E
:10101000CD2C103E80D300AFD300FFC30010E67F7D
:10102000CD2610C31310CD3510B612C9CD351047DB
:101030007E2FA012C947E678810F0F0FC6805F3E52
:1010400000CE305778E607C64E6F26101AC9010247
:0610500004081020408 09E
:00
```

```
        SAMPLE ROUTINE              MAR 12, 1976    X8080-V1A   PAGE    1

                                .TITLE  SAMPLE ROUTINE
                        ;       THIS ROUTINE READS IN STATUS INFORMATION AND
                        ;         UPDATES SIXTEEN DIFFERENT 128 BIT FLAG TABLES
                        ;         IN RAM.
         0   0      IPORT1  .EQU    0
         0   1      IPORT2  .EQU    1
         0   0      OPORT1  .EQU    0
        30  80      TABLES  .EQU    3080        ;BASE OF FLAG TABLES
        10   0              .ORG    1000
  1000  DB   0      LOOP    IN      IPORT1      ;GET READY FLAG AND TABLE #
  1002  A7                 ANA     A
  1003  F2   0 10         JP      LOOP        ;WAIT UNTIL DATA READY
  1006  E6   F            ANI     0F
  1008   F               RRC
  1009  4F                MOV     C,A         ;SAVE SHIFTED TABLE NUMBER
  100A  DB   1            IN      IPORT2      ;GET BIT POSITION NUMBER
  100C  A7                ANA     A
  100D  FA 1E 10          JM      $2          ;MOST SIG BIT MEANS SET BIT ON
  1010  CD 2C 10          CALL    CLEAR       ;CLEAR BIT FROM TABLE
  1013  3E 80      $1      MVI     A,80        ;STROBE ACKNOWLEDGE LINE
  1015  D3   0            OUT     OPORT1
  1017  AF                XRA     A
  1018  D3   0            OUT     OPORT1
                         .IFNZRO ?1          ;USER FLAG 1 IS SELECTED FOR
  101A  FF               RST     7           ;DEBUGGING.   A BREAKPOINT CALL
                         .ENDC               ;(RST 7) IS INSERTED FOR
                         .IFZERO ?1          ;DEBUGGING AND A 'NOP' IS
                         NOP                 ;INSERTED FOR NORMAL OPERATION
                         .ENDC
  101B  C3   0 10          JMP     LOOP        ;LOOP BACK FOR ANOTHER TRY
  101E  E6 7F      $2      ANI     ^B 01111111     ;MASK OUT SIGN
  1020  CD 26 10          CALL    SET         ;SET BIT INTO TABLE
  1023  C3 13 10          JMP     $1
```

```
                        .PAGE
                ;       SUBROUTINES TO SET AND CLEAR BITS IN A TABLE
1026 CD 35 10 SET       CALL    POSITN  ;POSITION THE POINTERS
1029 B6                 ORA     M       ;OR IN THE DECODED BIT
102A 12                 STAX    D       ;STORE RESULT BACK
102B C9                 RET
102C CD 35 10 CLEAR     CALL    POSITN  ;POSITION THE POINTERS
102F 47                 MOV     B,A     ;SAVE PREVIOUS BYTE TEMPORARILY
1030 7E                 MOV     A,M     ;GET DECODED BIT
1031 2F                 CMA
1032 A0                 ANA     B       ;MASK OUT SELECTED BIT
1033 12                 STAX    D       ;STORE BACK RESULT
1034 C9                 RET
                ;       ROUTINE TO POSITION THE TABLE BYTE POINTER AND
                ;         DECODE THE BIT POSITION.
1035 47         POSITN  MOV     B,A     ;SAVE TEMPORARILY
1036 E6 78              ANI     ^O 170  ;MASK FOR BYTE NUMBER IN TABLE
1038 81                 ADD     C       ;COMBINE WITH TABLE NUMBER
1039  F                 RRC             ;  AND FORM BYTE ADDRESS
103A  F                 RRC
103B  F                 RRC
103C C6 80              ADI     ^L TABLES
103E 5F                 MOV     E,A     ;SET UP ADDRESS IN D,E
103F 3E  0              MVI     A,0
1041 CE 30              ACI     ^M TABLES
1043 57                 MOV     D,A
1044 78                 MOV     A,B     ;DECODE BIT NUMBER WITHIN BYTE
1045 E6  7              ANI     7
1047 C6 4E              ADI     ^L $1   ;FORM LOOKUP TABLE ADDRESS
1049 6F                 MOV     L,A     ;  FOR DECODED BIT
104A 26 10              MVI     H,^M $1
104C 1A                 LDAX    D       ;GET TABLE BYTE
104D C9                 RET
                        .BIN            ;TABLE IS IN BINARY
104E  1         $1      .BYTE   1,10,100,1000
104F  2
1050  4
1051  8
1052 10                 .BYTE   10000,100000,1000000,10000000
1053 20
1054 40
1055 80
******  E:MO
                        JUNK            ;SAMPLE ERROR
                        .END
```



```
  102C CLEAR        0 IPORT1      1 IPORT2    1000 LOOP
     0 OPORT1    1035 POSITN    1026 SET      3080 TABLES
```

ERRORS: 1

#13.0.0 MICROPROCESSOR INSTRUCTION SET.

THIS SECTION IS A SUMMARY OF THE INSTRUCTION SET OF THE 8080 MICROPROCESSOR AS DEFINED BY THE VENDORS. THE ASSEMBLY CODE FORMAT FOR EACH INSTRUCTION IS SHOWN WITH THE HEXADECIMAL OBJECT CODE. EACH INSTRUCTION WILL BE CODED INTO THE PROPER NUMBER OF BYTES.

DATA TRANSFER INSTRUCTIONS:

INSTRUCTION	MEANING	HEX CODE	BYTES
LDA ADDR	LOAD ACC DIRECTLY FROM ADDR	3A	3
LDAX B	LOAD ACC INDIRECTLY FROM B,C	0A	1
LDAX D	LOAD ACC INDIRECTLY FROM D,E	1A	1
MOV B,B	MOVE REG B TO REG B	40	1
MOV B,C	MOVE REG C TO REG B	41	1
MOV B,D	MOVE REG D TO REG B	42	1
MOV B,E	MOVE REG E TO REG B	43	1
MOV B,H	MOVE REG H TO REG B	44	1
MOV B,L	MOVE REG L TO REG B	45	1
MOV B,M	MOVE MEMORY TO REG B	46	1
MOV B,A	MOVE ACC TO REG B	47	1
MOV C,B	MOVE REG B TO REG C	48	1
MOV C,C	MOVE REG C TO REG C	49	1
MOV C,D	MOVE REG D TO REG C	4A	1
MOV C,E	MOVE REG E TO REG C	4B	1
MOV C,H	MOVE REG H TO REG C	4C	1
MOV C,L	MOVE REG L TO REG C	4D	1
MOV C,M	MOVE MEMORY TO REG C	4E	1
MOV C,A	MOVE ACC TO REG C	4F	1
MOV D,B	MOVE REG B TO REG D	50	1
MOV D,C	MOVE REG C TO REG D	51	1
MOV D,D	MOVE REG D TO REG D	52	1
MOV D,E	MOVE REG E TO REG D	53	1
MOV D,H	MOVE REG H TO REG D	54	1
MOV D,L	MOVE REG L TO REG D	55	1
MOV D,M	MOVE MEMORY TO REG D	56	1
MOV D,A	MOVE ACC TO REG D	57	1
MOV E,B	MOVE REG B TO REG E	58	1
MOV E,C	MOVE REG C TO REG E	59	1
MOV E,D	MOVE REG D TO REG E	5A	1
MOV E,E	MOVE REG E TO REG E	5B	1
MOV E,H	MOVE REG H TO REG E	5C	1
MOV E,L	MOVE REG L TO REG E	5D	1
MOV E,M	MOVE MEMORY TO REG E	5E	1
MOV E,A	MOVE ACC TO REG E	5F	1

DATA TRANSFER INSTRUCTIONS (CONT.):

INSTRUCTIONS	MEANING	HEX CODE	BYTES
MOV H,B	MOVE REG B TO REG H	60	1
MOV H,C	MOVE REG C TO REG H	61	1
MOV H,D	MOVE REG D TO REG H	62	1
MOV H,E	MOVE REG E TO REG H	63	1
MOV H,H	MOVE REG H TO REG H	64	1
MOV H,L	MOVE REG L TO REG H	65	1
MOV H,M	MOVE MEMORY TO REG H	66	1
MOV H,A	MOVE ACC TO REG H	67	1
MOV L,B	MOVE REG B TO REG L	68	1
MOV L,C	MOVE REG C TO REG L	69	1
MOV L,D	MOVE REG D TO REG L	6A	1
MOV L,E	MOVE REG E TO REG L	6B	1
MOV L,H	MOVE REG H TO REG L	6C	1
MOV L,L	MOVE REG L TO REG L	6D	1
MOV L,M	MOVE MEMORY TO REG L	6E	1
MOV L,A	MOVE ACC TO REG L	6F	1
MOV M,B	MOVE REG B TO MEMORY	70	1
MOV M,C	MOVE REG C TO MEMORY	71	1
MOV M,D	MOVE REG D TO MEMORY	72	1
MOV M,E	MOVE REG E TO MEMORY	73	1
MOV M,H	MOVE REG H TO MEMORY	74	1
MOV M,L	MOVE REG L TO MEMORY	75	1
MOV M,A	MOVE MEMORY TO ACC	77	1
MOV A,B	MOVE REG B TO ACC	78	1
MOV A,C	MOVE REG C TO ACC	79	1
MOV A,D	MOVE REG D TO ACC	7A	1
MOV A,E	MOVE REG E TO ACC	7B	1
MOV A,H	MOVE REG H TO ACC	7C	1
MOV A,L	MOVE REG L TO ACC	7D	1
MOV A,M	MOVE MEMORY TO ACC	7E	1
MOV A,A	MOVE ACC TO ACC	7F	1
MOVI B,DATA	MOVE IMMEDIATE DATA TO REG B	06	2
MOVI C,DATA	MOVE IMMEDIATE DATA TO REG C	0E	2
MOVI D,DATA	MOVE IMMEDIATE DATA TO REG D	16	2
MOVI E,DATA	MOVE IMMEDIATE DATA TO REG E	1E	2
MOVI H,DATA	MOVE IMMEDIATE DATA TO REG H	26	2
MOVI L,DATA	MOVE IMMEDIATE DATA TO REG L	2E	2
MOVI M,DATA	MOVE IMMEDIATE DATA TO MEMORY	36	2
MOVI A,DATA	MOVE IMMEDIATE DATA TO ACC	3E	2
STA ADDR	STORE ACC DIRECT TO ADDR	32	3
STAX B	STORE ACC INDIRECTLY TO B,C	02	1
STAX D	STORE ACC INDIRECTLY TO D,E	12	1

ARITHMETIC INSTRUCTIONS:

INSTRUCTIONS	MEANING	HEX CODE	BYTES
ADC B	ADD REG B TO ACC WITH CARRY	88	1
ADC C	ADD REG C TO ACC WITH CARRY	89	1
ADC D	ADD REG D TO ACC WITH CARRY	8A	1
ADC E	ADD REG E TO ACC WITH CARRY	8B	1
ADC H	ADD REG H TO ACC WITH CARRY	8C	1
ADC L	ADD REG L TO ACC WITH CARRY	8D	1
ADC M	ADD MEMORY TO ACC WITH CARRY	8E	1
ADC A	ADD ACC TO ACC WITH CARRY	8F	1
ACI DATA	ADD IMMEDIATE TO ACC WITH CARRY	CE	2
ADD B	ADD REG B TO ACC	80	1
ADD C	ADD REG C TO ACC	81	1
ADD D	ADD REG D TO ACC	82	1
ADD E	ADD REG E TO ACC	83	1
ADD H	ADD REG H TO ACC	84	1
ADD L	ADD REG L TO ACC	85	1
ADD M	ADD MEMORY TO ACC	86	1
ADD A	ADD ACC TO ACC	87	1
ADI DATA	ADD IMMEDIATE TO ACC	C6	2
ANA B	AND REG B WITH ACC	A0	1
ANA C	AND REG C WITH ACC	A1	1
ANA D	AND REG D WITH ACC	A2	1
ANA E	AND REG E WITH ACC	A3	1
ANA H	AND REG H WITH ACC	A4	1
ANA L	AND REG L WITH ACC	A5	1
ANA M	AND MEMORY WITH ACC	A6	1
ANA A	AND ACC WITH ACC	A7	1
ANI DATA	ADD IMMEDIATE WITH ACC	E6	2
CMP B	COMPARE REG B WITH ACC	B8	1
CMP C	COMPARE REG C WITH ACC	B9	1
CMP D	COMPARE REG D WITH ACC	BA	1
CMP E	COMPARE REG E WITH ACC	BB	1
CMP H	COMPARE REG H WITH ACC	BC	1
CMP L	COMPARE REG L WITH ACC	BD	1
CMP M	COMPARE MEMORY WITH ACC	BE	1
CMP A	COMPARE ACC WITH ACC	BF	1
CPI DATA	COMPARE IMMEDIATE WITH ACC	FE	2

ARITHMETIC INSTRUCTIONS (CONT.):

INSTRUCTION	MEANING	HEX CODE	BYTES
DCR B	DECREMENT REG B	05	1
DCR C	DECREMENT REG C	0D	1
DCR D	DECREMENT REG D	15	1
DCR E	DECREMENT REG E	1D	1
DCR H	DECREMENT REG H	25	1
DCR L	DECREMENT REG L	2D	1
DCR M	DECREMENT MEMORY	35	1
DCR A	DECREMENT ACC	3D	1
INR B	INCREMENT REG B	04	1
INR C	INCREMENT REG C	0C	1
INR D	INCREMENT REG D	14	1
INR E	INCREMENT REG E	1C	1
INR H	INCREMENT REG H	24	1
INR L	INCREMENT REG L	2C	1
INR M	INCREMENT MEMORY	34	1
INR A	INCREMENT ACC	3C	1
ORA B	OR REG B WITH ACC	B0	1
ORA C	OR REG C WITH ACC	B1	1
ORA D	OR REG D WITH ACC	B2	1
ORA E	OR REG E WITH ACC	B3	1
ORA H	OR REG H WITH ACC	B4	1
ORA L	OR REG L WITH ACC	B5	1
ORA M	OR MEMORY WITH ACC	B6	1
ORA A	OR ACC WITH ACC	B7	1
ORI DATA	OR IMMEDIATE WITH ACC	F6	2
SBB B	SUBTRACT REG B FROM ACC WITH BORROW	98	1
SBB C	SUBTRACT REG C FROM ACC WITH BORROW	99	1
SBB D	SUBTRACT REG D FROM ACC WITH BORROW	9A	1
SBB E	SUBTRACT REG E FROM ACC WITH BORROW	9B	1
SBB H	SUBTRACT REG H FROM ACC WITH BORROW	9C	1
SBB L	SUBTRACT REG L FROM ACC WITH BORROW	9D	1
SBB M	SUBTRACT MEMORY FROM ACC WITH BORROW	9E	1
SBB A	SUBTRACT ACC FROM ACC WITH BORROW	9F	1
SBI DATA	SUBTRACT IMMEDIATE FROM ACC WITH BORROW	DE	2

ARITHMETIC INSTRUCTIONS (CONT.):

INSTRUCTIONS	MEANING	HEX CODE	BYTES
SUB B	SUBTRACT REG B FROM ACC	90	1
SUB C	SUBTRACT REG C FROM ACC	91	1
SUB D	SUBTRACT REG D FROM ACC	92	1
SUB E	SUBTRACT REG E FROM ACC	93	1
SUB H	SUBTRACT REG H FROM ACC	94	1
SUB L	SUBTRACT REG L FROM ACC	95	1
SUB M	SUBTRACT MEMORY FROM ACC	96	1
SUB A	SUBTRACT ACC FROM ACC	97	1
SUI DATA	SUBTRACT IMMEDIATE FROM ACC	D6	2
XRA B	EXCLUSIVE OR REG B WITH ACC	A8	1
XRA C	EXCLUSIVE OR REG C WITH ACC	A9	1
XRA D	EXCLUSIVE OR REG D WITH ACC	AA	1
XRA E	EXCLUSIVE OR REG E WITH ACC	AB	1
XRA H	EXCLUSIVE OR REG H WITH ACC	AC	1
XRA L	EXCLUSIVE OR REG L WITH ACC	AD	1
XRA M	EXCLUSIVE OR MEMORY WITH ACC	AE	1
XRA A	EXCLUSIVE OR ACC WITH ACC	AF	1
XRI DATA	EXCLUSIVE OR IMMEDIATE WITH ACC	EE	2

FLOW CONTROL INSTRUCTIONS:

INSTRUCTION	MEANING	HEX CODE	BYTE
CALL ADDR	CALL UNCONDITIONAL	CD	3
CC ADDR	CALL ON CARRY	DC	3
CNC ADDR	CALL ON NO CARRY	D4	3
CZ ADDR	CALL ON ZERO	CC	3
CNZ ADDR	CALL ON NON ZERO	C4	3
CP ADDR	CALL ON POSITIVE	F4	3
CM ADDR	CALL ON MINUS	FC	3
CPE ADDR	CALL ON EVEN PARITY	EC	3
CPO ADDR	CALL ON ODD PARITY	E4	3
JMP ADDR	JUMP UNCONDITIONAL	C3	3
JC ADDR	JUMP ON CARRY	DA	3
JNC ADDR	JUMP ON NO CARRY	D2	3
JZ ADDR	JUMP ON ZERO	CA	3
JNZ ADDR	JUMP ON NON ZERO	C2	3
JP ADDR	JUMP ON POSITIVE	F2	3
JM ADDR	JUMP ON MINUS	FA	3
JPE ADDR	JUMP ON EVEN PARITY	EA	3
JPO ADDR	JUMP ON ODD PARITY	E2	3
RET	RETURN	C9	1
RC	RETURN ON CARRY	D8	1
RNC	RETURN ON NO CARRY	D0	1
RZ	RETURN ON ZERO	C8	1
RNZ	RETURN ON NON ZERO	C0	1
RP	RETURN ON POSITVE	F0	1
RM	RETURN ON MINUS	F8	1
RPE	RETURN ON EVEN PARITY	E8	1
RPO	RETURN ON ODD PARIY	E0	1
RST 0	RESTART AT ADDRESS 00	C7	1
RST 1	RESTART AT ADDRESS 10	CF	1
RST 2	RESTART AT ADDRESS 20	D7	1
RST 3	RESTART AT ADDRESS 30	DF	1
RST 4	RESTART AT ADDRESS 40	E7	1
RST 5	RESTART AT ADDRESS 50	EF	1
RST 6	RESTART AT ADDRESS 60	F7	1
RST 7	RESTART AT ADDRESS 70	FF	1

REGISTER PAIR INSTRUCTIONS:

INSTRUCTIONS	MEANING	HEX CODE	BYTES
DAD B	ADD B&C TO H&L	09	1
DAD D	ADD D&E TO H&L	19	1
DAD H	ADD H&L TO H&L	29	1
DAD SP	ADD STACK POINTER TO H&L	39	1
DCX B	DECREMENT B&C REGISTERS	0B	1
DCX H	DECREMENT D&E REGISTERS	1B	1
DCX H	DECREMENT H&L REGISTERS	2B	1
DCX SP	DECREMENT STACK POINTER	3B	1
INX B	INCREMENT B&C REGISTERS	03	1
INX D	INCREMENT D&E REGISTERS	13	1
INX H	INCREMENT H&L REGISTERS	23	1
INX SP	INCREMENT STACK POINTER	33	1
LHLD ADDR	LOAD H&L DIRECT	2A	3
LXI B,DATA	LOAD IMMEDIATE REG PAIR B&C	01	3
LXI D,DATA	LOAD IMMEDIATE REG PAIR D&E	11	3
LXI H,DATA	LOAD IMMEDIATE REG PAIR H&L	21	3
LXI SP,DATA	LOAD IMMEDIATE STACK POINTER	31	3
PCHL	H&L TO PROGRAM COUNTER	E9	1
POP B	POP REG PAIR B&C OFF STACK	C1	1
POP D	POP REG PAIR D&E OFF STACK	D1	1
POP H	POP REG PAIR H&L OFF STACK	E1	1
POP PSW	POP ACC AND FLAGS OFF STACK	F1	1
PUSH B	PUSH REG PAIR B&C ON STACK	C5	1
PUSH D	PUSH REG PAIR D&E ON STACK	D5	1
PUSH H	PUSH REG PAIR H&L ON STACK	E5	1
PUSH PSW	PUSH ACC AND FLAGS ON STACK	F5	1
SHLD ADDR	STORE H&L DIRECT	22	3
SPHL	H&L TO STACK POINTER	F9	1
XCHG	EXCHANGE D&E WITH H&L REGISTERS	EB	1
XTHL	EXCHANGE TOP OF STACK WITH H&L	E3	1

ROTATE INSTRUCTIONS:

INSTRUCTIONS	MEANING	HEX CODE	BYTES
RLC	ROTATE ACC LEFT	07	1
RRC	ROTATE ACC RIGHT	0F	1
RAL	ROTATE ACC LEFT THROUGH CARRY	17	1
RAR	ROTATE ACC RIGHT THROUGH CARRY	1F	1

I/O INSTRUCTIONS:

INSTRUCTIONS	MEANING	HEX CODE	BYTES
IN PORT	INPUT FROM PORT	DB	2
OUT PORT	OUTPUT TO PORT	D3	2

MISCELLANEOUS:

INSTRUCTIONS	MEANING	HEX CODE	BYTES
CMA	COMPLEMENT ACC	2F	1
CMC	COMPLEMENT CARRY	3F	1
STC	SET CARRY	37	1
DAA	DECIMAL ADJUST ACC	27	1
EI	ENABLE INTERRUPTS	FB	1
DI	DISABLE INTERRUPTS	F3	1
HLT	HALT	76	1
NOP	NO OPERATION	00	1

APPENDIX A - RUN-TIME OPTIONS. #14.0.0

```
/B            - OUTPUT BINARY FILE IN BNPF FORMAT.
/E            - INHIBIT ERROR MESSAGES TO CONSOLE.
/H            - INHIBIT HEADINGS AND PAGINATION.
/J            - LIST UNASSEMBLED STATEMENTS AND CONDITIONAL
                ASSEMBLY PSEUDO-OPS.
/K            - EXPAND SYMBOL TABLE STORAGE INTO ADDITIONAL
                CORE.
/L            - OUTPUT LEADER (NULLS) IN BINARY FILE  FOR EACH
                .ORG STATEMENT.
/N            - LIST ONLY THE SYMBOL TABLE.
/O            - OUTPUT LISTING IN OCTAL FORMAT INSTEAD OF IN
                HEXADECIMAL.
/P            - INCLUDE NORMALLY UNLISTED PSEUDO-OPS IN THE
                LISTING.
/S            - OMIT THE SYMBOL TABLE FROM THE LISTING.
/T            - REPLACE THE FORM/FEED WITH 3 CR/LF'S.
/W            - INHIBIT WARNING MESSAGES.
/0 TO /9      - USER FLAGS, USED WITH THE ? OPERATOR.
```

APPENDIX B - INDICATOR SET.

```
*               MULTIPLICATION.
/               DIVISION.
&               BOOLEAN AND.
!               INCLUSIVE OR.
+               ADDITION.
-               SUBTRACTION.
^C              COMPLEMENT INDICATOR, (UPARROW B).
^B              BINARY RADIX INDICATOR, (UPARROW B).
^D              DECIMAL RADIX INDICATOR, (UPARROW D).
^H              HEXADECIMAL RADIX INDICATOR, (UPARROW H).
^O              OCTAL RADIX INDICATOR, (UPARROW O).
^L              LEAST SIGNIFICANT BYTE ACCESS OPERATOR,
                (UPARROW L).
^M              MOST SIGNIFICANT BYTE ACCESS OPERATOR,
                (UPARROW M).
;               COMMENT INDICATOR.
" OR '          ASCII CHARACTER INDICATOR.
?               USER FLAG OPERATOR.
.               CURRENT LOCATION COUNTER, (PERIOD).
```

****

.ADDR	DOUBLE BYTE DATA STORAGE, REVERSED FORMAT.
.BIN	CHANGES DEFAULT RADIX TO BINARY.
.BYTE	SINGLE BYTE DATA STORAGE.
.DBYTE	DOUBLE BYTE DATA STORAGE.
.DECM	CHANGES DEFAULT RADIX TO DECIMAL.
.DINST	RENAMES A MICROPROCESOR INSTRUCTION.
.END	PROGRAM TERMINATOR.
.ENDC	ENDS CONDITIONAL ASSEMBLY.
.EQU	ASSIGNS A PERMANENT VALUE TO A SYMBOL.
.HEX	CHANGES DEFAULT RADIX TO HEXADECIMAL.
.IFDEF	INCLUDE CODE TO .ENDC IF SYMBOL IS DEFINED.
.IFNDEF	INDLUDE CODE TO .ENDC IF SYMBOL IS NOT DEFINED.
.IFNZRO	INCLUDE CODE TO .ENDC IF OPERAND DOES NOT EQUAL 0.
.IFZERO	INCLUDE CODE TO .ENDC IF OPERAND EQUALS 0.
.LIST	PROVIDES SELECTIVE LISTINGS.
.OCT	CHANGES DEFAULT RADIX TO OCTAL.
.ORG	REASSIGNS THE CURRENT LOCATION COUNTER.
.PAGE	BEGINS NEW PAGE IN LISTING.
.SET	ASSIGNS A TEMPORARY VALUE TO A SYMBOL.
.TITLE	SPECIFIES HEADING.
.ZERO	ZEROS A SPECIFED NUMBER OF BYTES.

****

APPENDIX D - ERROR MESSAGES. #14.0.0

****

E:BN - BAD NESTING OF BRACKETS.
E:DF - OUTPUT FILE DEVICE FULL. (FATAL)
E:DR - DIGIT OUTSIDE OF RADIX.
E:IL - ILLEGAL LABEL FIELD.
E:IO - ILLEGAL OPERAND VALUE.
E:LO - LINE INPUT OVERFLOW.
E:LS - LOCAL SYMBOL SYNTAX ERROR.
E:LT - LOCAL SYMBOL TABLE OVERFLOW. (FATAL)
E:ML - MULTIPLE LABEL DEFINITION.
E:MO - MISSING OR ILLEGAL MNEMONIC IN OPERATOR FIELD.
E:OC - OPERAND TOO COMPLEX.
E:OE - OPEN ERROR IN OUTPUT FILE. (FATAL)
E:OM - OPERAND MISSING.
E:OS - OPERAND SYNTAX ERROR.
E:PE - PHASE ERROR, ADDRESS CONFLICT. (FATAL)
E:PS - ILLEGAL PERMANENT SYMBOL USAGE IN OPERAND.
E:RE - INPUT FILE READ ERROR. (FATAL)
E:RV - BAD REGISTER VALUE FIELD.
E:ST - SYMBOL TABLE OVERFLOW. (FATAL)
E:TL - LABEL DEFINED TOO LATE.
E:US - UNDEFINED SYMBOL.
E:WE - OUTPUT FILE WRITE ERROR. (FATAL)

W:EF - NO .END STATEMENT IN LAST FILE.
W:UC - UNINHIBITED CONDITIONAL ASSEMBLY IN EFFECT AT ASSEMBLY END.

****