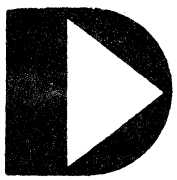
DOS ASSEMBLER 5
DOSASM5
User's Guide
Version 3

January, 1976

Model Code No. 50019

DATAPOINT CORPORATION



The Leader in
Dispersed Data Processing

PREFACE

This manual explains the operating instructions for the ASSEMBLER 5 and defines the directives and macros which are available to the user. The programmer will find the Datapoint DOS User's Guide helpful if more detailed systems information is required, and the Datapoint 2200 Reference Manual should be consulted for further instruction definition.

TABLE OF CONTENTS

CHAPTER 1. INTRODUCTION

Generating machine language programs for the Datapoint 2200 with ASSEMBLER 5 consists of using the DOS EDITOR to create one or more symbolic source file(s) comprised of mnemonic instructions, symbolic variables, and symbolic routine names which can then be processed by the ASSEMBLER to create an absolute, executable object file which can be loaded and executed by the OPERATING SYSTEM.

Since ASSEMBLER 5 and this manual assume many details which are inherent to the DOS and 2200, a working knowledge of both the DOS and the 2200 VI and VII processors is recommended before proceeding.

Basically, the ASSEMBLER is a program that assigns numerical values to symbols and puts out these values upon input of the associated symbols. Symbols in certain fields have preassigned values (such as instruction mnemonics) while other symbols are defined by the user (such as labels).

The value assigned to an instruction mnemonic is the binary bit configuration recognized by the 2200 processor for that instruction. For example, the following instruction mnemonics have the following octal values:

MNEMONIC	VALUE
ADB	0201
RET	0007
SU	0024

Symbols in fields other than the instruction field (except for the expression field in EXternal commands) may be defined by the user. Pre-defined symbols are kept separately by the ASSEMBLER so that the user may define symbols that are the same as the pre-defined symbols without encountering any difficulties. For example:

```
LABEL       INSTRUCTION    EXPRESSION

  L1            AD             1
                JMP            CALL
  L2            AD             2
  CALL          CALL           SUBR1
  INPUT         INPUT
```

will not present a problem in differentiating the two CALL and INPUT symbols since the ones in the instruction field are pre-defined and the ones in the label and expression fields are user-defined.

Along with relating symbols to numbers, another major function of the ASSEMBLER is to enable the programmer to reference a symbol that is defined later in the program. This is called FORWARD REFERENCING, and may be handled in a variety of ways. One of the simplest is to look at the source code twice. The first look determines the definitions of all the symbols and the second look uses the symbols to produce the object code. Each "look" at the source code is called a "PASS". Therefore, we end up with a two pass assembly process.

An optional function of the ASSEMBLER is that of producing a tabularized listing of all user-defined symbols, their octal value, and all references to them. This cross-reference table generation consists of recording all references to user-defined symbols during pass two, sorting the references, and merging them with their values.

The ASSEMBLER maintains two internal counters called the ADDRESS COUNTER and the LOCATION COUNTER. The ADDRESS COUNTER indicates the memory address of the object code currently being generated and the LOCATION COUNTER indicates the memory address at which the object code currently being generated will be executed. These counters are usually the same except in the case of Located Code (see Section 3.9). Each time a byte of code is generated, both counters are incremented. The values of these counters are initially set to 010000 but directives are available for changing their values either initially or dynamically (see Sections 3 and 5). The content of the ADDRESS COUNTER when processing of the current line is initiated is usually displayed at the left side of the listing. When the Location flag is set by a LOC directive, the LOCATION COUNTER (identified by a trailing L) is displayed instead of the ADDRESS COUNTER. The symbol $ has special meaning in that it has the value of the LOCATION COUNTER when processing of the current line began. For example:

```
ADRCTR      OBJECT CODE     SOURCE CODE

01000                               SET   01000
01000       104 000 002     XXX     JMP   XXX
01003       104 003 002     DOG     JMP   $
01006                       A       EQU   $
00001                       B       EQU   1
01006       123 123                 DC    0123,83
05400L                              LOC   05400
05400L      104 000 013     C       JMP   $
05403                       D       EQU   $
01013                               LOC   *
```

The ASSEMBLER maintains a stack of 16 dynamic Program Address Blocks (PAB'S) which may be used to locate data and code at Assembly time. A PAB is actually an ADDRESS COUNTER which has been given a symbolic name. This name is not used as a dictionary entry but is used solely for the purpose of requesting an ADDRESS COUNTER swap with the current PAB (see Sections 3.10 and 3.11).

An ABSOLUTE PAB is defined by the ASSEMBLER and is implicitly used anytime the programmer neglects to Originate (ORG) and Use (USE) additional PAB's (see Section 3.10 and 3.11). When a new PAB is requested, the current PAB's ADDRESS COUNTER is stored and the next available address associated with the requested PAB is placed in the ADDRESS and LOCATION COUNTERS.

The first word address and the length of each PAB is printed at the end of pass 1.

Example of PAB usage:

```
            ADRCTR    OBJECT CODE     SOURCE CODE

01000                       BUFFER  ORG   01000
07000                       CODE    ORG   07000
00120                       LTH     EQU   80
07000                               USE   CODE
07000    002 000 120                DC    *BUF1,LTH
07003    002 120 120                DC    *BUF2,LTH
01000                               USE   BUFFER
01000                       BUF1    SK    LTH
01120                       BUF2    SK    LTH
07006                               USE   *
07006    377                        HALT
```

CHAPTER 2. STATEMENTS

A 2200 assembly code statement consists of a label field, an instruction field, an expression field and a comment field. For example:

```
   1        2       3          4
LABEL      JTC     START    THIS IS A COMMENT FIELD
```

Field 1 is the LABEL FIELD
Field 2 is the INSTRUCTION FIELD
Field 3 is the EXPRESSION FIELD
Field 4 is the COMMENT FIELD

The 2200 editor provides tabulation so that the fields may be justified to begin in a certain column for ease of reading. Tab stops at columns 9, 15 and 30 create a good appearance. However, the ASSEMBLER only requires the following:

A non-space in the first column means that the first field is a label except for a leading period, plus, or asterisk, which designates the entire line as a comment line.

A space in the first column means no label and the first symbol on the line is an instruction.

Scanning proceeds from left to right. One or more spaces serve as delimiters for the LABEL and INSTRUCTION fields. Spaces may appear in the expression field without terminating the expression (see, however, Section 2.3).

2.1 LABEL FIELD

The Label Field may consist of any number of characters. However, only the first six will be used as a label name in the dictionary and, therefore, the first six must be unique. The first character may be any alphabetic character or a $ sign. The other characters may be any alphanumeric character or a $ sign. An asterisk or colon immediately following the label (with no intervening spaces) will declare the label as a program entry point and the label will be written to the entry point file by the ASSEMBLER (see Section 5). If the label field is terminated by an equal sign followed by a space, this occurrence of the label may

not be the first; in which case, a redefinition of the label's value will occur and the normal 'D' error flag will not be generated. Extreme care must be exercised when using this redefinition capability as directives must not use multiply defined symbols in their expression field and the ASSEMBLER will not error flag such usage. Some examples of labels follow.

VALID LABELS

LBL123

LABEL$

LABELA*

LABELB=

INVALID LABELS	REASON FOR ILLEGALITY
1LABEL	Starts with numeric
LABEL#	Non-alphanumeric or $ character (#)
LABELA.	Non-alphanumeric or $ character (.)
L1-2L3	Non-alphanumeric or $ character (-)

A label may appear in a statement which has empty instruction and expression fields. Invalid labels are flagged as 'E' errors (see 5.5.3).

2.2 INSTRUCTION FIELD

The Instruction Field may be any of the instruction mnemonics, assembler directives, or assembler macros. It has the same syntactical restrictions as the Label Field (any number of characters starting with a letter and containing only alphanumerics or $'s) except only the first two or three characters are used and consequently the user may abbreviate some instructions. For example:

CALL CALBCDEFG	These are both CALL instructions
INP INPUT	These are both INPUT instructions
INC	This is an INCLUDE directive
RET RETURN	These are both RETURN instructions

Any illegal or undefined instruction mnemonics will cause 'I' error flags to be generated.

2.3 EXPRESSION FIELD

The Expression Field consists of one or more expressions, delimited by commas, comprised of any number of strings, numbers, or symbols with operators between them. However, only the first expression will have significance except in the case of certain assembler directives (DC, DA, and IFnn) as noted in Sections 3.7, 3.8, and 3.19. DOSASM5 will allow spaces within an expression field in most cases; however, the use of spaces within expressions is not encouraged since other assemblers (i. e. SNAP1 and SNAP2) will not accept expressions with imbedded spaces. Numbers are assumed to be decimal (base 10) unless they have one or more leading zeros, in which case they are taken to be octal. That is, 123 is 123 decimal, whereas 0123 or 00123 (the octal number 123) is really 83 decimal.

String quantities are denoted (preceded and followed) by apostrophes. In expressions, only one character is allowed with the exception of the DC directive. The character's value is the ASCII binary number with the parity bit always a zero. A null string is illegal. A forcing character (#) is used in strings to indicate that the next character should be taken as ASCII no matter what it is. Thisis useful for entering the characters (') and (#) themselves string. For example:

'#'##' is the character string '#

Expressions are evaluated strictly from left to right and all operators have the same precedence (with no parentheses allowed). The expression scanner generates a 16-bit two's complement value giving a decimal range of -32768 to +32767. Instructions which use only eight bits will discard the most significant byte (MSB) of the value generated by the expression scanner and use only the least significant byte (LSB) of the value. Syntax errors in

expressions will be flagged with 'E' error flags. A 'U' flag is issued in pass one when an assembler directive other than DA or DC is operating on an expression containing a label not yet in the dictionary. A 'U' flag is issued in pass two whenever an expresssion contains an undefined label. The expression field is omitted for instructions which require no expression.

There are eleven operators allowed in expressions:

2.3.1 + This means addition.

2.3.2 - This means subtraction. Note that the minus sign may be placed at the beginning of an expression if the value of the first item is to be negated.

2.3.3 * When used as the first character in the expression, this operator will set the ASSEMBLER'S star flag (see Sections 3.7, 3.8, 4.4, and 4.5). It may be followed by a minus operator (e.g. *-DOG+1).

2.3.4 * When used as other than the first character in the expression, signifies 16-bit integer multiplication.

2.3.5 / A slash indicates least whole integer division. This means that any remainder produced by the division will not be used.

2.3.6 > This means shift right. The value accumulated up to this point is logically shifted right the number of places indicated in the following label value or number (all bits shifted off the end are discarded and zeros are filled in on the left). Because the operation is a logical shift, sign is not maintained. Thus, negative numbers will be treated as positive 16-bit values instead of two's complement 16-bit values.

2.3.7 < This is the same as > except shifting is to the left with zero fill on the right.

2.3.8 .AND. This means to perform a logical 'AND' of the two positive 16-bit numbers.

2.3.9 .OR. .IOR. These mean to perform a logical inclusive 'OR' of the two positive 16-bit numbers.

2.3.10 .XOR. This means to perform a logical exclusive 'OR' of the two positive 16-bit numbers.

Note that only the first character of a logical operation is used to determine the operation type and that additional characters prior to the second period are ignored.

2.4 EXAMPLES OF EXPRESSIONS

The following examples assume that the value of DOG is 1 and that the value of CAT is 2.

VALID EXPRESSIONS	VALUE	
DOG	1	
DOG+1	2	
1+DOG	2	
DOG+CAT	3	
'A'+1	0102	
*-CAT+1	-1	Note that star flag will be set.
-DOG<3	-8	
-DOG>3	8191	Note that sign is not extended on right shifts.
8>3+1	2	Note that shift occurs before addition.
CAT*CAT	4	
CAT.AND.DOG	0	
DOG.OR.CAT	3	
0377.XOR.DOG	0376	

ILLEGAL EXPRESSIONS

DOG+	Terminating character not a space or comma.
DOG#1	Illegal binary operator.
'AB'	Illegal if not a DC statement. Only 1 character allowed in all other expression strings.
'A'+1	Illegal only in a DC statement because a separator is expected after a string and + is not a valid separator (see Section 3.7).
CAT+DOG=	Illegal terminator character.
CAT.NOT.1	Illegal binary operator.
**12	Star flag set but no multiplier exists for second asterisk.
.XOR.1	No value prior to operator.

2.5 COMMENT FIELD

The Comment Field begins anywhere after the Expression Field, Instruction Field (if the Expression Field is not used), or Column 2 (if Column 1 contains a period, plus, or asterisk as noted in Sections 3.14, 3.15, and 3.16). The Comment Field may contain any character and is terminated by the end of the line. The ASSEMBLER puts out its listing of the source line exactly as it is provided in the source code so formatting of comments will be maintained.

CHAPTER 3. ASSEMBLER DIRECTIVES

Assembler Directives are used for setting the LOCATION COUNTER, ADDRESS COUNTER, and LABEL values to other than the normal sequential assignments and for defining constants. Other Directives are used to control certain ASSEMBLER functions such as input file linking, source file assembly, and program listing. Note that the normal forward referencing in the expression field is only permitted in the DC and DA directives.

3.1 INCLUDE

INC — Includes the source from the file specification given in DOS format in the expression field. Up to 25 files may be included at nesting levels of up to 4 deep. Exceeding these limits will result in 'F' errors and the inclusion(s) will not be made. Lines of source code originating from an included file are noted by a trailing alphabetic character in the line number. Note that the label field of an INCLUDE directive is ignored and no dictionary entry made.

3.2 EQUIVALENCE

EQU — Sets the value of the label on the statement to the value of the expression field. Object code is not generated by EQU's, but dictionary labels are. External references can be handled by equating labels to external locations and then referencing the labels. Will produce an 'E' error if no label is found.

3.3 SET

SET — Sets the first word and current word address of the ABSOLUTE PAB (initially 010000) to the value of the expression field, clears the Location flag, and initiates usage (USE) of the ABSOLUTE PAB (see Section 3.11).

3.4 SKIP

SK — Increments the values of the LOCATION and ADDRESS COUNTERS by the value of the expression field.

3.5 TABULATE PAGE

TP — Increments the value of the ADDRESS COUNTER until it is a multiple of 256 (LSB = 000). This is useful for setting up page-dependent data areas which are addressable by single precision (leaving H fixed and manipulating only the L-register). If the Location flag is set, the ADDRESS COUNTER is not incremented and an 'I' error is produced.

3.6 TABULATE MAYBE

TM — Performs a Tabulate Page if the value of the expression field would cause a page overflow if added to the current ADDRESS COUNTER. Will produce an 'I' error if the Location flag is set.

3.7 DEFINE CONSTANT

DC — Generates eight bit object bytes from one or more expressions or strings delimited by commas found in the expression field. A leading asterisk on any expression will produce two object bytes (LSB, MSB) and therefore addresses may be imbedded within DC directives. A special exception is made for string items found in the DC directive. All the characters of a string item are significant and as many words as necessary are generated to accommodate all the characters of the given string. This special string item is in effect only if the expression is opened with an apostrophe.

String items in expressions still have only one character of significance. For example:

```
DC 1,2+3,2+'A','ABC'
```

generates the following octal values:

```
001,005,0103,0101,0102,0103
```

Note that 'A'+2 is illegal as the DC directive will consider it as a special multiple-character string and the + is not a legal terminator (only space or comma) but that 2+'A' is legal since the normal expression scanner will be used to determine its value.

3.8 DEFINE ADDRESS

DA Generates a two byte constant which is the address, LSB first, of the expression. Placing an * in front of an expression will cause the two bytes to be generated in the reverse order (MSB first, LSB second). For example:

```
DOG   EQU   01234
      DA    DOG,*DOG,1
```

gives the following octal values:

```
234,002,002,234,001,000
```

3.9 LOCATION

LOC Sets LOCATION COUNTER to the value of the expression field and sets the Location flag. If the expression field consists of an asterisk, the Location flag is cleared and the LOCATION COUNTER is set to the ADDRESS COUNTER. Note that the listing will have the LOCATION COUNTER (noted by a trailing L) printed instead of the ADDRESS COUNTER.

3.10 ORIGINATE

ORG — Initializes a new PAB and sets its first and current word addresses to the value of the expression field. The label field only defines the PAB's name and not a label for the dictionary. An 'E' error flag will be issued if the PAB has been previously defined or if there is no label. This error is fatal and causes pass two to abort.

3.11 USAGE

USE — Declares the usage of the PAB whose name is given in the expression field. An asterisk in the field will revert back to the last PAB used. An 'E' error will be issued if the PAB named has not been originated. This error is fatal and will abort pass two.

3.12 REPEAT

RPT — Will cause the following line of source code to be processed the number of times indicated by the LSB of the expression field's value. For example:

```
RPT 5
CALL INCHL
```

will produce the same code as:

```
CALL INCHL
CALL INCHL
CALL INCHL
CALL INCHL
CALL INCHL
```

Repeating statements with labels which do not have a trailing = to signify a multiple definition will result in 'D' error flags.

3.13 END

END — Indicates that there is no more source code to be processed and that the ASSEMBLER should proceed to pass 2 if in pass 1 or complete generating the output if in pass 2. Note that an 'F' error will be issued if an END is found in an Included file. The expression field has special significance in the END statement in that its value is taken as the Primary Transfer Address at which program execution will begin. This is optional and a Secondary Transfer Address is set by the ASSEMBLER to the location of the first byte of object code.

3.14 PERIOD

. — A period in the first column will cause the ASSEMBLER to treat the entire line as a comment line.

3.15 PLUS SIGN

\+ — A plus sign in the first column will cause a page eject during the listing of the program. The line will be treated as a comment line as well and printing will occur after the ejection.

3.16 ASTERISK

* — An asterisk in the first column will cause a page eject if the listing is within two inches of the bottom of a page. The line is treated as a comment line and printing occurs after any possible ejection.

3.17 LIST

LIS This is a directive which is used to alter the settings of the ASSEMBLER'S listing control flags. Each flag is specified by one character which turns the flag on when mentioned in a LIST statement unless it is preceded by a minus sign which will turn the flag off. Commas may be used to delimit more than one flag character. The flag characters, their default settings, and their usage are as follows:

L	ON	Master list control. If turned off, no pass two output will be listed until this flag is turned on again regardless of other control flags.
G	OFF	Generated lines. If turned off, this flag will suppress the listing of code lines generated by DC, DA, and RPT statements.
I	OFF	Included lines. Lines of source code included from additional source files will not be listed unless this flag is on.
F	OFF	If-skipped lines. This flag must be on to produce a listing of all lines of code skipped by an IF<nn> statement.

3.18 ERROR

ERR Produces a 'P' error in both pass 1 and pass 2. Usually follows a conditional assembly statement to trap a page, table, overflow etc. For example:

```
TABLE       SK      LEN
            IFNE    $>8,TABLE>8
            ERR     TABLE OVERFLOWS A PAGE!
            XIF
```

3.19 IF

IFnn This is the conditional assembly directive. Condition 'nn' (assumed to be 'EQ' if not given) must be met when comparing the two expressions found separated by a comma in the expression field in order to assemble following lines of code. The second expression will be assumed zero if not given. Only an XIF directive will turn the conditional assembly back on. However, IF statement nesting may occur to any depth. An undefined expression operand in pass 1 is fatal and this occurrence will cause pass 2 to be aborted. The available condition codes are:

EQ	Field 1 must be equal to field 2
GT	Field 1 must be greater than field 2
LT	Field 1 must be less than field 2
NE	Field 1 must not be equal to field 2
GE	Field 1 must be either greater than or equal to field 2
LE	Field 1 must be either less than or equal to field 2
Z	Field 1 must be zero
NZ	Field 1 must be non-zero
C	Field 1 must be clear (flag-testing, same as Z)
S	Field 1 must be set (flag-testing, same as NZ)

3.20 XIF

XIF Forces the assembly on if it has been conditionally turned off.

CHAPTER 4. ASSEMBLER MACROS

Assembler MACROS are opcode mnemonics which directly result in the generation of a sequence of machine instructions.

4.1 HL

HL (exp) — The HL macro generates the load H register and load L register instructions necessary to place the value of the expression field in the H and the L registers properly so that a load to or from memory will use that address i.e. H contains the MSB and L contains the LSB. The HL macro generates four bytes of object code. For example:

```
OOPS  EQU  02005
      HL   OOPS
```

generates the following code:

```
066 005 056 004
```

4.2 DE

DE (exp) — The DE macro works the same as the HL macro except it loads the D and E registers instead of H and L.

4.3 BC

BC (exp) — The BC macro works the same as the HL macro except it loads the B and C registers instead of H and L.

4.4 MEMORY STORE

MS(r) (*)(exp) The Memory Store macro allows the user to store a given register into a given memory location. Placing an * in front of the expression causes the H-register to be loaded as well as the L. The expansion is as follows:

```
LL      (exp)
LH      (exp)>8 if * is present
LM(r)
```

4.5 MEMORY LOAD

ML(r) (*)(exp) The Memory Load macro works the same as the Memory Store (MSr) macro with the exception that the register is loaded from memory rather than being stored into memory.

4.6 SHIFT RIGHT NUMBERIC

SRN (exp) The Shift Right Numeric macro allows the user to generate SRC instructions the number of times specified in the expression field. The expression must be defined in pass one and must have a value between zero and seven. For example:

```
SRN  3
```

will generate the following code:

```
012 012 012
```

4.7 SHIFT LEFT NUMERIC

SLN	(exp)	The Shift Left Numeric macro works the same as the SRN macro with the exception that SLC instructions (002) are generated.

CHAPTER 5. OPERATING PROCEDURES

The DOS command requesting execution of the Version 5 ASSEMBLER should be as follows:

ASM source(,object(,entryp))(;(D)(L)(X)(F)(I)(G))

where each pair of parentheses and their content is optional.

5.1 PARAMETERIZATION

The first file spec (which is required) is the source file, the second file specification is for the object file, and the third file specification is for the entry point file. Each of these three files must be physically different. The source file has a default extension of TXT. The object file, if not given, is assumed to have the same name as the source file and has a default extension of ABS. The entry point file, if required and not given, is assumed to have the same name as the source file with a default extension of EPT. The entry point file is written after pass one only if entry points have been declared in the program. The EPT file is written in a compressed symbolic format which can be INCLUDED by a later assembly to provide a program linking capability.

The characters on the command line following the semicolon specify output options. The character L will cause the output to be listed on a Servo printer, if one is on-line. Otherwise, L produces a listing on the local printer. If the character X appears, a cross-reference map will be listed on the Servo or local printer (as above). X may appear in the command line without L if a cross-reference table but no program listing is desired. The character D signifies that the output should be displayed on the 2200 CRT, and the remaining valid characters instruct the ASSEMBLER to turn on their respective listing control flags (see Section 3.17). If neither L nor X appear as parameters, no printed output is produced. Assembly error messages are displayed on the CRT regardless of which options are specified.

5.2 EXECUTION-TIME COMMANDS

During the ASSEMBLER pass one and pass two, two execution-time commands may be invoked. Depressing the DISPLAY key will prevent the 2200 CRT screen from rolling up until the key is released. The KEYBOARD key may be used to terminate the

assembly and return to DOS.

5.3 ASSEMBLER PASS ONE

Initially the ASSEMBLER will validate the three file specifications and the parameter string and will then request an 80-character heading if either the L or X parameter was specified. Next it will print its Version number and the maximum number of labels it can handle in its dictionary. The ASSEMBLER will then read the source file and any INCLUDED files in order to build a dictionary containing all symbolic names used by the programmer and their equivalent octal value or address. A notation is printed as each INCLUDE is processed along with any lines which contain errors. At the end of pass one, one or more of the following items will be printed:

1) Any pass one error flags
2) Dictionary overflow message if overflow occurred
3) Fatal error message if error occurred
4) Program Entry Points--name, value
5) List of undefined symbols
6) List of unused symbols
7) List of multiply defined symbols
8) PAB starting locations and lengths

5.4 ASSEMBLER PASS TWO

If no fatal pass one errors occurred, the ASSEMBLER will now write the entry point file, if required, and proceed into pass two. Pass two is responsible for the actual generation of object code and a program listing. However, if a cross-reference listing is to be generated, pass two will also write a reference file (ASMXREF/SYS) which will contain all symbolic references made in the program.

5.5 CROSS-REFERENCE GENERATION

At the completion of pass two, the ASSEMBLER will call in overlay 1 if a cross-reference listing is desired. Overlay 1 uses DOS SORT to sort ASMXREF/SYS into the ordered file ASMSREF/SYS and produces from it a cross-reference table. The actual listing of references will contain the symbolic name preceded by its actual octal value, unless the name is undefined in which case it is preceded by asterisks. Following the symbolic name is a list of all line numbers at which that symbolic name was defined or referenced. All definition lines are flagged with a leading asterisk while all Inclusions are noted by a trailing colon followed by the Inclusion file character (see Section 3.1). For

example:

```
     11304 DECHL          *32:A    *32:B
     00341 DISPL          *24
     00024 IDLE          *197212            212      212
     10176 INCHL           71     *102      151      156
     00007 MANY            21:A   *25:A     21:B     *25:B
     ***** ILDE           213
```

5.6 ASSEMBLY ERRORS

The ERROR FLAGS produced by the 2200 ASSEMBLER during both passes are as follows:

5.5.1 D The D flag means DIFFERENT DEFINITION. It is generated if an attempt has been made to define the label more than once without a trailing = mark. Generated in pass one only.

5.5.2 I The I flag means INSTRUCTION MNEMONIC UNDEFINED. The instruction was not an acceptable instruction and a zero or 0377 is inserted for the instruction.

5.5.3 E The E flag means that an error has occurred in an EXPRESSION or some unrecognizable character appeared in the wrong place. In this case, a zero is substituted for the expression or in whatever was unrecognizable if code generation was expected.

5.5.4 U The U flag means UNDEFINED LABEL. It is issued in pass two whenever a label is referenced and is not defined and it is issued in pass one when an assembly directive (except DA or DC) is operating on an expression containing a label not yet in the dictionary (forward referencing).

5.5.5 F The F flag means FILE error. It can be issued in either pass when the ASSEMBLER'S limits for an inclusion are exceeded or when an INCLUDED file contains an END directive.

5.5.6 P The P flag means PROGRAMMER PRODUCED. It is issued in both passes when an ERR directive is encountered.

APPENDIX A. INSTRUCTION REPERTOIRE

Notes: Opcodes shown without mnemonics are undefined. See Datapoint 2200 reference manual for further instruction definition.

OP CODE		MNEMONIC	OP CODE		MNEMONIC
000			040		DI
001			041		
002		SLC	042		
003		RFC	043		RTC
004	XXX	AD <exp>	044	XXX	ND <exp>
005			045		
006	XXX	LA <exp>	046	XXX	LE <exp>
007		RET	047		
010		SYNC	050		EI
011			051		
012		SRC	052		
013		RFZ	053		RTZ
014	XXX	AC <exp>	054	XXX	XR <exp>
015			055		
016	XXX	LB <exp>	056	XXX	LH <exp>
017			057		
020		BETA	060		POP
021			061		
022			062		
023		RFS	063		RTS
024	XXX	SU <exp>	064	XXX	OR <exp>
025			065		
026	XXX	LC <exp>	066	XXX	LL <exp>
027			067		
030		ALPHA	070		PUSH
031			071		
032			072		
033		RFP	073		RTP
034	XXX	SB <exp>	074	XXX	CP <exp>
035			075		
036	XXX	LD <exp>	076		
037			077		

OP CODE	MNEMONIC	OP CODE	MNEMONIC
100 LSB MSB	JFC <exp>	140 LSB MSB	JTC <exp>
101	INPUT	141	
102 LSB MSB	CFC <exp>	142 LSB MSB	CTC <exp>
103		143	
104 LSB MSB	JMP <exp>	144	
105		145	
106 LSB MSB	CALL <exp>	146	
107		147	
110 LSB MSB	JFZ <exp>	150 LSB MSB	JTZ <exp>
111		151	EX BEEP
112 LSB MSB	CFZ <exp>	152 LSB MSB	CTZ <exp>
113		153	EX CLICK
114		154	
115		155	EX DECK1
116		156	
117		157	EX DECK2
120 LSB MSB	JFS <exp>	160 LSB MSB	JTS <exp>
121	EX ADR	161	EX RBK
122 LSB MSB	CFS <exp>	162 LSB MSB	CTS <exp>
123	EX STATUS	163	EX WBK
124		164	
125	EX DATA	165	
126		166	
127	EX WRITE	167	EX BSP
130 LSB MSB	JFP <exp>	170 LSB MSB	JTP <exp>
131	EX COM1	171	EX SF
132 LSB MSB	CFP <exp>	172 LSB MSB	CTP <exp>
133	EX COM2	173	EX SB
134		174	
135	EX COM3	175	EX REWND
136		176	
137	EX COM4	177	EX TSTOP

OP CODE	MNEMONIC	OP CODE	MNEMONIC
200	ADA	240	NDA
201	ADB	241	NDB
202	ADC	242	NDC
203	ADD	243	NDD
204	ADE	244	NDE
205	ADH	245	NDH
206	ADL	246	NDL
207	ADM	247	NDM
210	ACA	250	XRA
211	ACB	251	XRB
212	ACC	252	XRC
213	ACD	253	XRD
214	ACE	254	XRE
215	ACH	255	XRH
216	ACL	256	XRL
217	ACM	257	XRM
220	SUA	260	ORA
221	SUB	261	ORB
222	SUC	262	ORC
223	SUD	263	ORD
224	SUE	264	ORE
225	SUH	265	ORH
226	SUL	266	ORL
227	SUM	267	ORM
230	SBA	270	CPA
231	SBB	271	CPB
232	SBC	272	CPC
233	SBD	273	CPD
234	SBE	274	CPE
235	SBH	275	CPH
236	SBL	276	CPL
237	SBM	277	CPM

OP CODE	MNEMONIC	OP CODE	MNEMONIC
300	NOP	340	LEA
301	LAB	341	LEB
302	LAC	342	LEC
303	LAD	343	LED
304	LAE	344	
305	LAH	345	LEH
306	LAL	346	LEL
307	LAM	347	LEM
310	LBA	350	LHA
311		351	LHB
312	LBC	352	LHC
313	LBD	353	LHD
314	LBE	354	LHE
315	LBH	355	
316	LBL	356	LHL
317	LBM	357	LHM
320	LCA	360	LLA
321	LCB	361	LLB
322		362	LLC
323	LCD	363	LLD
324	LCE	364	LLE
325	LCH	365	LLH
326	LCL	366	
327	LCM	367	LLM
330	LDA	370	LMA
331	LDB	371	LMB
332	LDC	372	LMC
333		373	LMD
334	LDE	374	LME
335	LDH	375	LMH
336	LDL	376	LML
337	LDM	377	HALT

APPENDIX B. MNEMONIC OPCODE REPERTOIRE

This appendix contains a list of all valid mnemonics which can be used in the opcode field of the ASSEMBLER. Each mnemonic is followed with a brief definition of its usage. Note that the condition flip-flops which are specified as <c> may be specified as follows:

Carry --- C or B
Zero ---- Z or E
Sign ---- S or L or N
Parity -- P

The processor registers which are specified as <r> may be specified by the register name, i.e. A, B, C, D, E, H, L or M (if Memory Ref).

OPCODE	DESCRIPTION	SECTION
AC	Add with Carry Immediate Instruction	
AC<r>	Add with Carry Register Instruction	
AD	Add Immediate Instruction	
AD<r>	Add Register Instruction	
ALPHA	Select Alpha Mode Instruction	
BC	Load B and C Macro	4.3
BETA	Select Beta Mode Instruction	
CALL	Subroutine Call Instruction	
CF<c>	Conditional Subroutine Call Instruction	
CT<c>	Conditional Subroutine Call Instruction	
CP	Compare Immediate Instruction	
CP<r>	Compare With Register Instruction	
DA	Define Address Directive	3.8
DC	Define Constant Directive	3.7
DE	Load D and E Macro	4.2
DI	Disable Interrupt Instruction	
EI	Enable Interrupt Instruction	
END	End Source Code Directive	3.13
EQU	Equivalence Directive	3.2
ERR	Produce Error Directive	3.18

OPCODE	DESCRIPTION	SECTION
EX	External I/O Instruction	
HALT	Processor Halt Instruction	
HL	Load H and L Macro	4.1
IF<nn>	Conditional Assembly Directive	3.19
INCLUDE	Source File Inclusion Directive	3.1
INPUT	I/O Input Instruction	
JF<c>	Jump on False Condition Instruction	
JMP	Jump Instruction	
JT<c>	Jump on True Condition Instruction	
L<r>	Load Immediate Instruction	
L<r><r>	Load from Register Instruction	
LIST	Listing Control Directive	3.17
LOC	Location Counter Manipulative Directive	3.9
ML<r>	Memory Load Macro	4.5
MS<r>	Memory Store Macro	4.4
ND	And Immediate Instruction	
ND<r>	And with Register Instruction	
NOP	No Operation Instruction	
OR	Or Immediate Instruction	
OR<r>	Or with Register Instruction	
ORG	PAB Origination Directive	3.10
POP	Pushdown Stack Manipulation Instruction	
PUSH	Pushdown Stack Manipulation Instruction	
RET	Subroutine Return Instruction	
RF<c>	Conditional Subroutine Return Instruction	
RT<c>	Conditional Subroutine Return Instruction	
RPT	Repeat Source Line Directive	3.12
SB	Subtract with Borrow Immediate Instruction	
SB<r>	Subtract with Borrow Register Instruction	
SET	Address Counter Manipulation Directive	3.3
SKIP	Address/Location Counter Directive	3.4
SLC	Shift Left Circular Instruction	
SLN	Shift Left Numeric Macro	4.7
SRC	Shift Right Circular Instruction	
SRN	Shift Right Numeric Macro	4.6

OPCODE	DESCRIPTION	SECTION
SU	Subtract Immediate Instruction	
SU<r>	Subtract Register Instruction	
SYNC	Processor Sync Instruction	
TM	Tab Page Maybe Directive	3.6
TP	Tab Page Directive	3.5
USE	PAB Manipulation Directive	3.11
XIF	End Conditional Assembly Directive	
XR	Exclusive Or Immediate Instruction	
XR<r>	Exclusive Or Register Instruction	

APPENDIX C. EXTERNAL COMMAND REPERTOIRE

MNEMONIC	SIGNAL	ADDRESS	DESCRIPTION
ADR	Address	All	Select New Device
BEEP	--	All	Activate Tone Producing Mechanism
BSP	--	0360	Back Up One Record
CLICK	--	All	Activate Click Producing Mechanism
COM1	Command 1	All	Output a Control Function
COM2	Command 2	All	Output a Control Function
COM3	Command 3	All	Output a Control Function
COM4	Command 4	All	Output a Control Function
DATA	Sense Data	All	Connects Device Data to Input Lines
DECK1	--	0360	Select Cassette Deck 1
DECK2	--	0360	Select Cassette Deck 2
RBK	--	0360	Enable Read Circuitry and Forward Motion
REWIND	--	0360	Rewind The Selected Deck
SB	--	0360	Slew Backward Motion
SF	--	0360	Slew Forward Motion
STATUS	Sense Status	All	Connects Device Status to Input Lines
TSTOP	--	0360	Stop Any Deck Motion

MNEMONIC	SIGNAL	ADDRESS	DESCRIPTION
WBK	--	0360	Enable Write Circuitry and Forward Motion
WRITE	Write Strobe	All	Indicates Output Data Availability

APPENDIX D. OBJECT FILE FORMAT

The object file created by the ASSEMBLER has a system loader object format (see DOS User's Guide, Part IV, Sections 3.1 and 3.2):

Logical Record Number	Byte #	Description
LRN 0 (RIB)	0	Physical File Number
	1	Logical Record Number (LSB)
	2	Logical Record Number (MSB)
	3	0377
	4	Segment Descriptor 1
	5	
	6	Segment Descriptor 2
	.	
	.	
	.	
	2N+2	Segment Descriptor N
	2N+3	
	2N+4	0377
	2N+5	0377
LRN 1 (RIB COPY)		
LRN 2	0	Physical File Number
	1	Logical Record Number (LSB)
	2	Logical Record Number (MSB)
	3	0 - indicating data block
	4	Starting address of block (LSB)
	5	Starting address of block (MSB)
	6	One's complement of LSB of starting address
	7	One's complement of MSB of starting address
	8	Block length (n)
	9	Beginning of data
	.	
	.	
	.	
	n+9	0 - Next data block
	n+10	Starting address of block (LSB)
	n+11	Starting address of block (MSB)
	n+12	One's complement of LSB of

```
                        starting address
                n+13    One's complement of MSB of
                        starting address
                n+14    Block length (m)
                n+15    Beginning of block data
                .
                .
                .
                n+m+15  0 - Next data block
                .
                .
                .
                .       0377 - End of Record

LRN 3           0       Physical File Number
                1       Logical Record Number (LSB)
                2       Logical Record Number (MSB)
                3       0 - Next data block
                .
                .
                .

LRN N           0
                .
                .
                .
                .       0 - Last data block
                .       Transfer address (LSB)
                .       Transfer address (MSB)
                .       One's complement of the LSB of the
                        transfer address
                .       One's complement of the MSB of the
                        transfer address
                .       0 - block length equal to zero
                        signifies end-of-file
```

APPENDIX E. SAMPLE PROGRAM

The following pages contain a sample assembly language program to give the reader a better understanding of the output he will see from the ASSEMBLER. Due to its tutorial nature, the program itself does not do anything useful. However, an example of every instruction, expression, directive, and Assembler Macro is given along with examples of how various errors are flagged and treated.

```
DOSASM5 3.1    568 LABELS


INCLUSION A: ASM5TST2/TXT
E    17.A                       4DOGS   EQU   $              THIS IS AN ERROR DURING INCLUSION
U    19.A                       HICUPS  EQU   MANY           THIS WILL PRODUCE A 'D' ERROR
F    43.A                               END                  ERROR!  THIS IS AN INCLUDED LINE
INCLUSION B: ASM5TST2/TXT
E    17.B                       4DOGS   EQU   $              THIS IS AN ERROR DURING INCLUSION
D    19.B                       HICUPS  EQU   MANY           THIS WILL PRODUCE A 'D' ERROR
F    43.B                               END                  ERROR!  THIS IS AN INCLUDED LINE
D   287.                        TABLE   DC    010,020,030    THREE CODE BYTES ARE GENERATED
E   306.                                DC    'PROGRAM XXXXX VERSION ','0'+VER
E   313.                                DC    'PROGRAM XXXXX VERSION ','0'+VER
E   320.                        1DOG    LA    5
E   321.                        2DOG    LA    DOGGIE         ONLY ONE ERROR FLAG WILL BE SEEN
E   322.                        QUICK%  DC    'INVALID DUE TO TRAILING % MARK'
E   332.                                SLN   25
E   341.                                DA    **3+TABLE

ERRORS: DEUF

 PROGRAM ADDRESS BLOCKS:    007314    /ABSOLUTE/    LTH=000673
                            010000    /DATA/        LTH=000364
                            012000    /CODE/        LTH=000000
                            014000    /BUFFER/      LTH=000360
                            014000    /INIT/        LTH=000137

 PRIMARY TRANSFER ADDRESS:  014000

 ENTRY POINTS:              014000    BUFFER
                            010006    PNTR
                            010176    INCHL

 MULTIPLE DEFINITIONS:                SEC
                                      MANY
                                      PNTR
                                      DECHL

 UNDEFINED LABELS:                    DOGGIE

 UNUSED LABELS:                       QUICK
                                      DISPL
```

```
 1.
 2.
 3.
 4.                              . ASSEMBLER 5 SAMPLE PROGRAM
 5.                              .
 6.                              . THESE LINES ARE COMMENT LINES
 7.                              . IT IS USUALLY A GOOD IDEA TO IDENTIFY YOUR PROGRAM AT THE
 8.                              . BEGINNING WITH SEVERAL COMMENT LINES WHICH CONTAIN THE
 9.                              . NAME OF THE PROGRAM AND ITS PURPOSE.  ANOTHER IMPORTANT
10.                              . USE OF COMMENT LINES IS TO DESCRIBE THE LOGIC FLOW OF THE
11.                              . PROGRAM IN A BLOCK FORM WHICH CAN BE DESCRIBED IN DETAIL
12.                              . IN THE COMMENT FIELD OF SOURCE CODE.
13.                              .
14.                              .
15.                              .
16.                              . THE USE OF EQU'S AND ORG'S AT THE BEGINNING OF THE PROGRAM FOR
17.                              . SETTING UP CONSTANTS WHICH ARE SUBJECT TO CHANGE (SUCH AS
18.                              . DEVICE ADDRESS, BUFFER LENGTHS, PROGRAM BLOCK ADDRESSES,
19.                              . TABLE ENTRY LENGTHS, COUNTER STEP SIZES, ETC) MAKES THE
20.                              . PROGRAM EASIER TO 'READ' AND CAN SAVE A LOT OF TIME WHEN
21.                              . THE NEED ARISES TO CHANGE ONE OF THESE CONSTANTS.
22.                              .
23.                              .
24.                              .
25.                              .
26. 000341                       KEYBD   EQU   0341          DEVICE ADDRESS OF 2200 KEYBOARD
27. 000341                       DISPL   EQU   KEYBD         NOTICE THAT SIMPLY CHANGING THE
28.                              .                           ADDRESS OF THE KEYBD WILL CHANGE
29.                              .                           THE VALUE OF BOTH CONSTANTS
30.                              .
31.                              . THE LABEL 'DISPL' WILL BE LISTED ON PAGE 1 UNDER 'UNUSED'
32.                              . LABELS SINCE IT IS NOT REFERENCED ANYWHERE IN THE PROGRAM.
33.                              . 'UNUSED' LABELS ARE DEFINED AS LABELS WHICH ARE UNREFERENCED
34.                              . IN THE PROGRAM, NOT DECLARED AS ENTRY POINTS FOR USE BY
35.                              . OTHER PROGRAMS, AND WHICH ARE NOT DEFINED IN AN INCLUDED
36.                              . FILE.  ALL THREE CONDITIONS MUST BE MET!
37.                              .
38. 000252                       M252    EQU   0252          THESE ARE MASKS WHICH WILL
39. 000125                       M125    EQU   0125          BE USED LATER WITH THE LOGICAL
40. 000377                       M377    EQU   0377          EXPRESSION FIELD OPERANDS
41.                              .
42. 010000                       DATA    ORG   010000        DATA BLOCK
43. 012000                       CODE    ORG   012000        CODE BLOCK
44. 014000                       BUFFER  ORG   014000        BUFFERS
45. 014000                       INIT    ORG   014000        INITIALIZATION  (OVERLAYS BUFFER AREA)
```

```
                        + THIS WILL FORCE A NEW LISTING PAGE
                        .
                        .
007314                      SET   007314        NOTICE THAT THE ABSOLUTE PAB
                        .                         WILL IMPLICITLY BE USED HERE
                        .
007400                      TP                  THIS WILL FORCE A NEW MEMORY PAGE
                        .
007400                      SK    200           IF WE HAVE A TABLE 60 BYTES IN LENGTH
                        .                         WHICH WE ARE COUNTING ON AS BEING
                        .                         ON ONE PAGE (PAGE-DEPENDENT), WE
                        .                         CAN USE A TM DIRECTIVE LIKE THIS
010000                      TM    60              TO MAKE SURE THAT THE ENTIRE TABLE
                        .                         WILL EITHER FIT ON THE CURRENT MEMORY
                        .                         PAGE OR A NEW PAGE WILL BE STARTED
```

```
 61.                                    +
 62.                                    . THE FOLLOWING PORTION OF CODE IS A SAMPLE 2200 I/O ROUTINE
 63.                                    .
 64.  010000  006 341                          LA    KEYBD          PICK UP AN OCTAL 341 IN THE A-REGISTER
 65.  010002  121                              EX    ADR              AND ADDRESS THE KEYBOARD
 66.  010003  101                    WAITI     INPUT                GET THE DEVICE STATUS IN A
 67.  010004  044 002                          ND    2              CHECK FOR THE READ READY BIT
 68.  010006  150 003 020                      JTZ   WAITI            AND WAIT UNTIL IT IS SET TO A 1
 69.  010011  125                              EX    DATA           SWITCH FROM STATUS TO DATA ON THE
 70.                                    .                              2200 INPUT LINES
 71.  010012  101                              IN                     AND INPUT THE ACTUAL DATA CHARACTER
 72.  010013  066 056 056 020                  HL    CHAR           POINT H AND L TO MEMORY LOCATION 'CHAR'
 73.  010017  370                              LMA                    AND STORE THE CHARACTER WHICH IS IN A
 74.  010020  106 176 020                      CALL  INCHL          POINT H AND L TO THE NEXT MEMORY LOCATION
 75.  010023  377                              HALT                 STOP THE 2200
 76.  010024  151                              EX    BEEP             AND BEEP WHEN 'RUN' IS PRESSED
 77.                                    . NOTICE THE FOLLOWING TWO LINES AND THE SEPARATE USES OF 'STATUS'
 78.  010025  123                              EX    STATUS         SWITCH BACK TO THE DEVICE STATUS
 79.  010026  101                    STATUS    IN                     AND GET THE NEW STATUS
 80.  010027  012                              SRC                  CHECK FOR DISPLAY READY BY POSITIONING
 81.                                    .                              THE READY BIT SUCH AS TO SET THE CARRY FLAG
 82.  010030  100 026 020                      JFC   STATUS           AND WAIT FOR THE BIT TO BECOME NON-ZERO
 83.                                    . BELOW ARE THREE MANNERS IN WHICH TO GENERATE THE 5 BYTES OF
 84.                                    . CODE WHICH ARE REQUIRES TO STORE THE A-REGISTER IN 'CHAR'
 85.  010033  066 056 056 020 370              MSA   *CHAR            YOU CAN USE A SINGLE MACRO
 86.                                    .
 87.  010040  066 056 056 020                  HL    CHAR           YOU CAN USE A MACRO
 88.  010044  370                              LMA                    AND A 2200 MNEMONIC
 89.                                    .
 90.  010045  056 020                          LH    CHAR>8         YOU CAN ALSO USE THREE
 91.  010047  066 056                          LL    CHAR             INDEPENDENT 2200
 92.  010051  370                              LMA                    INSTRUCTION MNEMONICS
 93.                                    .
 94.  010052  127                              EX    WRITE          OUTPUT THE CHARACTER IN A
 95.  010053  104 053 020                      JMP   $              THIS WILL HANG IN AN ENDLESS LOOP!
 96.                                    . NOTICE THAT THE STORAGE ARRAY CALLED 'CHAR' MAY BE INTERSPERSED
 97.                                    . WITH THE CODE PROVIDING THAT NO ATTEMPT IS MADE TO EXECUTE IT
 98.  010056                         CHAR      SK    80             THIS WILL ALLOW 80 MEMORY LOCATIONS
 99.                                    .                              TO BE USED FOR THE ARRAY BUT NO
100.                                    .                              DATA WILL BE LOADED INTO THE ARRAY
101.                                    .
102.                                    * FORCE THE FOLLOWING ROUTINE TO BE PRINTED ON ONE PAGE
103.                                    . INCHL -- INCREMENT H AND L BY 1
104.                                    .
105.  010176  306                    INCHL*    LAL                  LOAD THE L-REG INTO THE A-REG
106.  010177  004 001                          AD    1              ADD ONE TO THE A-REG
107.  010201  360                              LLA                  LOAD THE L-REG BACK FROM THE A-REG
108.  010202  305                              LAH                  LOAD THE H-REG INTO THE A-REG
109.  010203  014 000                          AC    0              ADD 1 ONLY IF L OVERFLOWED
110.  010205  350                              LHA                  RELOAD H-REG FROM THE A-REG
```

```
112.                                +
113.                                . NOTICE THAT THE ASSEMBLER KEEPS TRACK OF THE ADDRESS COUNTER
114.                                . WHEN SWITCHING BETWEEN PABS TO GENERATE TABLES
115.                                .
116.  014000                        BUFFER* USE   BUFFER         SET UP 3 BUFFERS IN ONE BLOCK
117.  014000                        BUF1    SK    80
118.  010000                              USE   DATA           POINT TO THE 3 BUFFERS
119.  010000  000 030                PNTR*   DA    BUF1
120.  014120                              USE   BUFFER         RETURN TO THE BUFFER BLOCK
121.  014120                        BUF2    SK    80
122.  010002                              USE   DATA
123.  010002  120 030                      DA    BUF2
124.  014240                              USE   *              IMPLIED USAGE OF 'BUFFER' PAB
125.  014240                        BUF3    SK    80
126.  010004                              USE   *              IMPLIED USAGE OF 'DATA' PAB
127.  010004  240 030                      DA    BUF3
128.                                .
129.                                . NOTICE THAT A SIMPLIER MANNER IN WHICH TO GENERATE THE POINTER
130.                                . TABLE FOLLOWS AND THAT THIS PROCEDURE WILL NOT GENERATE
131.                                . DICTIONARY ENTRIES WHICH MAY NOT OTHERWISE BE NEEDED.
132.                                .
133.  010006                        PNTR=   EQU   $              NOTICE THAT ALL REFERENCES TO 'PNTR'
134.                                .                              WILL USE THE CURRENT ADDRESS
135.                                        RPT   3
136.  010006  000 030                      DA    $-PNTR/2*80+BUFFER
136.  010010  120 030                      DA    $-PNTR/2*80+BUFFER
136.  010012  240 030                      DA    $-PNTR/2*80+BUFFER
137.                                .
138.                                . ALSO NOTICE THAT THE EXPRESSION FIELD IS TOTALLY ORDER-DEPENDENT!
```

```
                                        +
                                        . ANOTHER FEATURE OF USING PAB'S IS THAT THE ASSEMBLER CAN KEEP
                                        . TRACK OF THE USAGE OF THE SAME BLOCK OF CODE WITH TWO DIFFERENT
                                        . PAB'S FOR USE IN SETTING UP BUFFERS, ETC. WHICH DO NOT ACTUALLY
                                        . LOAD MEMORY (THEY GENERALLY ARE SKIPS) AND THEN OVERLAY THESE
                                        . BUFFERS WITH ONE-SHOT INITIALIZATION PROCESSES.
                                        .
                                        .
014000                                          USE   INIT          OVERLAY THE BUFFERS
                                        . THIS CODE MOVES THE ROUTINE 'INT' INTO LOW CORE (LOC'N 0000)
014000  066 000 056 000                 START   HL    0000          POINT TO DESTINATION ADDRESS
014004  070                                     PUSH                  AND PUSH IT ON THE STACK
014005  066 040 056 030                         HL    INT           POINT TO SOURCE ADDRESS
014011  026 077                                 LC    INTEND-INT    C = ROUTINE LENGTH
014013  307                             MOVE    LAM                 PICK UP A BYTE
014014  106 176 020                             CALL  INCHL         BUMP THE SOURCE ADDRESS
014017  335                                     LDH                   AND STORE IT IN D AND E
014020  346                                     LEL
014021  060                                     POP                 GET THE DESTINATION ADDRESS
014022  370                                     LMA                   AND STORE THE BYTE
014023  106 176 020                             CALL  INCHL         BUMP THE DESTINATION ADDRESS
014026  070                                     PUSH                  AND STORE IT ON THE STACK
014027  302                                     LAC                 MOVE THE COUNTER FROM C TO A
014030  024 001                                 SU    1             DECREMENT THE COUNT BY ONE
014032  320                                     LCA                 RELOAD C WITH THE COUNTER
014033  353                                     LHD                 RESTORE SOURCE ADDRESS
014034  364                                     LLE
014035  100 013 030                             JFC   MOVE          LOOP UNTIL ENTIRE ROUTINE IS MOVED
                                        .
                                        . NOTICE THAT THE STACK IS LEFT WITH THE LAST DESTINATION ADDRESS+1
                                        . AT THIS POINT.  THIS WILL DO NO HARM IF THE PROGRAMMER DOES NOT
                                        . EXECUTE A RETURN INSTRUCTION FROM THIS STACK LEVEL.  HOWEVER,
                                        . IT IS A GOOD PRACTICE FOR SUBROUTINES TO ALWAYS RETURN THE STACK
                                        . TO THE SAME LEVEL AS WHEN ENTERED UPON EXIT.
                                        .
                                        .
```

```
                                        +
                                        .
                                        . THE FOLLOWING ROUTINE 'INT' IS DESIGNED TO BE MOVED INTO LOW
                                        . CORE PRIOR TO ACTUAL EXECUTION BY SOME TYPE OF A RELOCATING
                                        . PROGRAM AS SEEN ABOVE.  THE ROUTINE 'INT' IS AN INTERRUPT DRIVEN
                                        . CLOCK PROGRAM WHICH KEEPS TIME IN LOCATION 'SEC' IN ONE SECOND
                                        . STEPS DERIVED FROM THE 2200'S 1 MILLISECOND INTERRUPT CLOCK.
                                        . PLEASE NOTICE THAT THE ROUTINE MAKES USE OF A TIMING LOOP AND
                                        . NOP'S IN ORDER TO CONSUME A PRECISE AMOUNT OF TIME (THAT BEING
                                        . 116.8 MICROSECONDS) REGARDLESS OF WHETHER UPDATING ONE BYTE OR
                                        . THREE.  THIS IS NECESSARY IF MULTIPLE TIME-CRITICAL I/O TASKS
                                        . ARE TO BE HANDLED AT EACH INTERRUPT TIME.
                                        .
014040                          INT     EQU   $             THIS IS THE LOADING ADDRESS
000000L                                 LOC   0000          WHEREAS THIS IS THE EXECUTION ADDRESS
000000L 040                             DI                  DISABLE INTERRUPTS
000001L 020                             BETA                SWITCH TO BETA MODE REGISTERS
000003                          MSEC    EQU   $+1           MILLISECOND TIMER
000002L 006 000                         LA    0000          LOAD THE MILLISECOND TIMER
000004L 004 001                         AD    1               AND BUMP IT BY ONE
000006L 074 372                         CP    250           CHECK FOR 1/4 SECOND VALUE
000010L 100 037 000                     JFC   INT2            AND JUMP IF ATTAINED
000013L 066 003 056 000                 HL    MSEC          POINT HL TO 'MSEC'
000017L 300                             NOP
000020L 300                             NOP                 THESE NOP'S ARE FOR TIMING!
000021L 300                             NOP
000022L 026 004                         LC    4             SET TIMING LOOP COUNT
000024L 370                     IDLE    LMA                 STORE THE NEW TIMER VALUE
000025L 302                             LAC                 GET THE LOOP COUNT
000026L 024 001                 TIMEIT  SU    1             DECREMENT THE COUNT
000030L 300                             NOP                 USED FOR TIMING
000031L 110 026 000                     JFZ   TIMEIT        CYCLE IN TIMING LOOP
000034L 030                     GOBACK  ALPHA               RETURN TO THE ALPHA MODE REGISTERS
000035L 050                             EI                  ENABLE THE INTERRUPT SYSTEM
000036L 007                             RET                 RETURN TO INTERRUPTED PROGRAM
000037L 250                     INT2    XRA                 LOAD THE A-REGISTER WITH ZEROES
000040L 066 003 056 000 370             MSA   *MSEC         CLEAR THE MILLISECOND TIMER
                                . NOTICE THE USE OF THE * IN THE MSA AND MLA MACROS
000045L 066 075 307                     MLA   QSEC          PICK UP THE QUARTER SECOND TIMER
000050L 004 001                         AD    1               BUMP IT
000052L 044 003                         ND    3               AND FORCE MODULO 4 (SAME AS CP 4 HERE)
000054L 026 002                         LC    2             SET UP TIMING LOOP COUNTER
000056L 110 024 000                     JFZ   IDLE          GO BACK IF NOT A FULL SECOND
000061L 006 000                         LA    0
000063L 370                             LMA                 CLEAR THE QUARTER SECOND TIMER
000064L 066 076 307                     MLA   SEC           PICK UP THE ACCUMULATED SECONDS
000067L 004 001                         AD    1               BUMP IT
000071L 370                             LMA                   AND STORE IT
000072L 104 034 000                     JMP   GOBACK        THEN RETURN
000075L 000                     QSEC    DC    0
000076L 000                     SEC     DC    0             THIS IS THE ACTUAL 1 SECOND TIMER
014137                          INTEND  LOC   *             THIS IS THE END OF THE ROUTINE
000076                          SEC=    EQU   $-1-INT       THIS WILL ALSO SET THE TIMER'S ADDRESS
```

```
                                +
                                . HERE ARE A FEW EXAMPLES OF CONDITIONAL ASSEMBLY IF STATEMENTS
                                .
010014                                  USE   DATA
                                        LIST  -G
010014  000 002 000 004 000     TABLE   DA    01000,02000,03000,04000,05000,06000
                                . NOW TEST TO MAKE SURE THAT THE ABOVE 'TABLE' IS ALL ON THE SAME PAGE
                                        IFNE  $>8,TABLE>8
                                        ERR   ******* TABLE ERROR *******
                                        XIF
                                .
                                .
                                . MASK TESTING CAN ALSO BE DONE
                                .
                                        IF    M252.0.M125,M377
010030  300                             NOP                   THIS WILL BE ASSEMBLED
                                        XIF
                                .
                                        IFZ   M252.A.M125
                                        IFGT  5,4
                                        IFS   M377
                                . CONDITIONAL ASSEMBLY TESTS MAY BE NESTED TO ANY DEPTH.  IN SUCH A
                                . CASE, THE CODE WILL ONLY BE ASSEMBLED IF ALL PRECEEDING TESTS ARE
                                . TRUE.  NOTICE THAT ALTHOUGH THE THREE PRECEEDING IF'S ARE TRUE AS
                                . CAN BE SEEN BY THE ASSEMBLY OF THIS DC STATEMENT,
010031  001 002 003 004 005             DC    1,2,3,4,5
                                . THE FOLLOWING IF IS NOT SATISFIED AND THEREFORE THE LINES OF CODE
                                . WHICH FOLLOW IT WILL NOT BE EXAMINED BY THE STATEMENT SCANNER.
                                        IFLE  010,7
                                        SKIP  4               ALL OF THESE STATEMENTS ARE IGNORED
                                        HALT
                                        ERR
                                        DC    0,0
                                . DO NOTE THAT SINCE THE SCANNER IS ONLY LOOKING FOR AN XIF AT THIS
                                . TIME THAT ALTHOUGH THE FOLLOWING IF'S ARE SATISFIED ON A LINE FOR
                                . LINE BASIS, THEY WILL BE IGNORED AND WILL NOT ALLOW ASSEMBLY OF
                                . THE CODE WHICH FOLLOWS THEM.
                                        IFEQ  2,2
                                        DC    2
                                        IF    3,2*1+1
                                        DC    3
                                        NOP
                                        XIF                   BUT THIS WILL TURN ON THE ASSEMBLY
010036  005 004                         DC    5,4             ALLOWING THIS TO ASSEMBLE!
```

```
   272.                                 +
   273.                                 . INCLUDE TEXT FROM ANOTHER FILE
   274.                                 .
   275.                                         LIST  -I              DO NOT LIST INCLUDED LINES
   276.                                         INC   ASM5TST2
E   17.A 010040                         4DOGS   EQU   $               THIS IS AN ERROR DURING INCLUSION
F   43.A                                         END                   ERROR!  THIS IS AN INCLUDED LINE
   277.                                 .
   278.                                 . NOW INCLUDE THE SAME TEXT BUT LIST IT
   279.                                 .
   280.                                         LIST  I
   281.                                         INC   ASM5TST2        THIS INCLUSION USES SUFFIX 'B'
     1.B                                .
     2.B                                .
     3.B                                .
     4.B                                . NOTICE THAT THIS IS LINE NUMBER 4 OF AN INCLUDED FILE.
     5.B                                . THE INCLUSION LEVEL IS INDICATED BY THE TRAILING CHARACTER
     6.B                                . OF THE LINE NUMBER WHICH WILL START WITH AN 'A' AND INCREMENT
     7.B                                . THROUGH THE ALPHABET AS EACH INCLUSION IS MADE REGARDLESS
     8.B                                . OF WHETHER THE INCLUDED LINES ARE LISTED OR NOT.
     9.B                                .
    10.B                                . ALL OF THE INCLUSION FILE INFORMATION LINKING EACH INCLUSION
    11.B                                . OF EACH FILE WITH AN ALPHABETIC CHARACTER IS GIVEN ON PAGE 1
    12.B                                . OF THIS LISTING IN SEQUENTIAL ORDER.
    13.B                                .
    14.B                                . NOTICE THAT THE CROSS-REFERENCE MAP WILL FLAG ALL REFERENCES
    15.B                                . WITH THEIR FILE IDENTIFICATION CHARACTER.
    16.B                                .
E   17.B 010040                         4DOGS   EQU   $               THIS IS AN ERROR DURING INCLUSION
    18.B                                .
    19.B 000007                         HICUPS  EQU   MANY            THIS WILL PRODUCE A 'D' ERROR
    20.B                                .                              SINCE IT IS NOT SPECIFIED AS
    21.B                                .                              MULTIPLY DEFINABLE.
    22.B                                .
    23.B 000007                         MANY=   EQU   7               ALL INCLUDED ITEMS MUST HAVE THE
    24.B                                .                              TRAILING = MARK IF THEY MIGHT
    25.B                                .                              POSSIBLY BE DEFINED MORE THAN
    26.B                                .                              ONCE, SUCH AS IN THE CASE OF
    27.B                                .                              INCLUDED SUBROUTINES WHICH ALL
    28.B                                .                              USE COMMON OPERATING SYSTEM
    29.B                                .                              ROUTINES
    30.B 011304                         DECHL=  EQU   011304          HERE IS SUCH A CRITTER
    31.B                                .
    32.B                                .
    33.B                                . THE INCLUDED FILE IS ALSO AN INTEGRAL PART OF THE ASSEMBLER'S
    34.B                                . ENTRY POINT / EXTERNAL REFERENCE 'LINKING' CAPABILITY.  THE
    35.B                                . ONLY MANNER IN WHICH THE ENTRY POINT FILE CAN BE USED IS BY
    36.B                                . AN INCLUSION IN AN OVERLAYED PROGRAM'S SOURCE FILE.
    37.B                                .
    38.B                                .
    39.B                                . INCLUDED FILES MAY NOT CONTAIN AN 'END' STATEMENT.  FILES
    40.B                                . WHICH DO CONTAIN END'S ARE NOT TERMINATED AT THE END BUT ARE
    41.B                                . READ TO THEIR END OF FILE MARK WITH THE END STATEMENT(S) FLAGGED
    42.B                                . BY 'F' ERRORS WHICH ARE PRINTED REGARDLESS OF THE 'I' CONTROL FLAG.
```

```
F   43.B                                END                     ERROR!  THIS IS AN INCLUDED LINE
    44.B                        .
    45.B                        .
    46.B                        . THESE ARE THE LAST TWO LINES OF THE INCLUDED FILE 'ASM5TST2' AND
    47.B                        . THE LINE IMMEDIATELY FOLLOWING THE INCLUDE WILL NOW BE SCANNED.
```

```
   282.                                 +
   283.                                 . NOW A MULTIPLE DEFINITION OF A PREVIOUS LABEL
   284.                                 . WHERE NO TRAILING EQUAL (=) IS USED PRODUCES
   285.                                 . AN ERROR IN PASS ONE (SEE PAGE ONE)
   286.                                 .
   287.  010040  010 020 030            TABLE   DC    010,020,030    THREE CODE BYTES ARE GENERATED
   288.                                 *
   289.                                 . HERE IS AN EXAMPLE OF THE SPECIAL DC STRINGS
   290.                                 .
   291.                                         LIST  G
   292.  010043  124 110 111 123 040            DC    'THIS MULTI-CHARACTER MESSAGE IS GOOD'
         010050  115 125 114 124 111
         010055  055 103 110 101 122
         010062  101 103 124 105 122
         010067  040 115 105 123 123
         010074  101 107 105 040 111
         010101  123 040 107 117 117
         010106  104
   293.  010107  102 105 103 101 125            DC    'BECAUSE','EACH','EXPRESSION','STARTS'
         010114  123 105 105 101 103
         010121  110 105 130 120 122
         010126  105 123 123 111 117
         010133  116 123 124 101 122
         010140  124 123
   294.  010142  127 111 124 110 101            DC    'WITH','A','STRING'
         010147  123 124 122 111 116
         010154  107
   295.                                 .
   296.                                 . A GOOD MANNER IN WHICH TO CHANGE A NUMBER WITHIN A DISPLAY
   297.                                 . MESSAGE WITHOUT SEARCHING FOR THE ACTUAL DC STATEMENT COULD
   298.                                 . BE HANDLED FROM AN EQU AT THE START OF THE PROGRAM.
   299.                                 .
   300.  000003                        VER     EQU   3               PROGRAM VERSION NUMBER
   301.                                 . AND IMBEDDED WITHIN THE PROGRAM WOULD BE THE FOLLOWING:
   302.  010155  120 122 117 107 122            DC    'PROGRAM XXXXX VERSION ',VER+'0'
         010162  101 115 040 130 130
         010167  130 130 130 040 126
         010174  105 122 123 111 117
         010201  116 040 063
   303.                                 .
   304.                                 . HOWEVER, THIS WILL PRODUCE AN ERROR:
   305.                                 .
   306.  010204  120 122 117 107 122            DC    'PROGRAM XXXXX VERSION ','0'+VER
         010211  101 115 040 130 130
         010216  130 130 130 040 126
         010223  105 122 123 111 117
E        010230  116 040 060 000
   307.                                 .
   308.                                 . THE ASSEMBLER WILL PRINT ALL ERROR LINES REGARDLESS OF THE
   309.                                 . LIST CONTROL FLAGS.  HERE IS THE ABOVE EXAMPLE WITH THE 'G'
   310.                                 . FLAG TURNED OFF AND THE ERROR PRINTED.
   311.                                 .
   312.                                         LIST  -G
   313.  010234  120 122 117 107 122            DC    'PROGRAM XXXXX VERSION ','0'+VER
```

```
E        010260  116 040 060 000
    314.                                 *
    315.                                 .
    316.                                 . NOW NOTICE THE FLAGGING OF AN ILLEGAL LABEL WILL STILL
    317.                                 . ALLOW THE PRODUCTION OF CODE
    318.                                 .
    319.                                       LIST  G
E   320.  010264  006 005                  1DOG    LA    5
U   321.  010266  006 000                  2DOG    LA    DOGGIE         ONLY ONE ERROR FLAG WILL BE SEEN
E   322.  010270  111 116 126 101 114     QUICK%  DC    'INVALID DUE TO TRAILING % MARK'
          010275  111 104 040 104 125
          010302  105 040 124 117 040
          010307  124 122 101 111 114
          010314  111 116 107 040 045
          010321  040 115 101 122 113
    323.                                 *
    324.                                 . THE SLN AND SRN MACROS GENERATE SEVERAL SLC OR SRC INSTRUCTIONS
    325.                                 .
    326.  010326  002 002 002 002 002          SLN   5
    327.  010333  012 012 012                  SRN   3
    328.                                 . SINCE A ZERO SHIFT COUNT WILL NOT PRODUCE CODE, A VARIABLE
    329.                                 . MAY BE USED AS LONG AS THE RANGE 0-7 IS MAINTAINED
    330.                                       SLN   VER-VER
    331.                                 . ERROR OCCURS IF THE COUNT IS OUT OF RANGE
E   332.  010336  002 002 002 002 002          SLN   25
          010343  002 002 002
    333.                                 *
    334.                                 . NOW NOTICE THE ORDER OF GENERATION OF THE MSP AND LSP IN THE
    335.                                 . FOLLOWING DA DIRECTIVES
    336.                                 .
    337.  010346  000 030 030 000              DA    START,*START
    338.  010352  014 020 020 014 357          DA    TABLE,*TABLE,*-TABLE,-TABLE
          010357  364 364 357
    339.                                 . ALTHOUGH A LEADING ASTERISK WILL SET THE 'STAR' FLAG, THE SECOND
    340.                                 . ASTERISK IMPLIES MULTIPLICATION AND THE MULTIPLIER DOES NOT EXIST
E   341.  010362  000 000                      DA    **3+TABLE
    342.                                 *
    343.                                 .
    344.                                 .
    345.                                 . THE MNEMONIC STARTING ADDRESS IS 'START' AND BY PLACING THIS
    346.                                 . NAME IN THE EXPRESSION FIELD OF THE END STATEMENT, THE
    347.                                 . ASSEMBLER WILL FLAG ITS OCTAL ADDRESS AS THE 'PRIMARY
    348.                                 . TRANSFER ADDRESS' FOR THE LOADER.  (ALSO NOTED AT THE FIRST
    349.                                 . OF THE LISTING)
    350.                                 .
    351.  014000                             END   START          END OF ALL SOURCE STATEMENTS
ERRORS: EUF
```

```
14000  BUF1           *117     119
14120  BUF2           *121     123
14240  BUF3           *125     127
14000  BUFFER         *116     136     136     136
10056  CHAR            72       85      87      90      91     *98
11304  DECHL          *30:A    *30:B
00341  DISPL          *27
*****  DOGGIE          321
00034  GOBACK         *207     223
00007  HICUPS          *19:A   *19:B
00024  IDLE           *202     217
10176  INCHL           74      *105     154     159
14040  INT             151      152    *188     227
00037  INT2            196     *210
14137  INTEND          152     *226
00341  KEYBD           *26      27      64
00125  M125            *39     242     246
00252  M252            *38     242     246
00377  M377            *40     242     248
00007  MANY             19:A     19:B  *23:A   *23:B
14013  MOVE           *153     166
00003  MSEC           *192     197     211
10000  PNTR           *119     *133    136     136     136
00075  QSEC            213     *224
10270  QUICK          *322
00076  SEC             220     *225    *227
14000  START          *149     337     337     351
10026  STATUS          *79      82
10014  TABLE          *233     235    *287     338     338     338     338
00026  TIMEIT         *204     206
00003  VER            *300     302     330     330
10003  WAITI           *66      68

NUMBER OF SYMBOLS USED   32
```