NUCLEAR DATA, INC.
Post Office Box 451
Palatine, Illinois 60067

January, 1971

IM41-0001
SOFTWARE INSTRUCTION MANUAL
BASC-12 GENERAL ASSEMBLER

TABLE OF CONTENTS

1. INTRODUCTION

A. PROGRAM SUMMARY

BASC-12 is a 2-Pass Assembler (optional 3rd pass) for the ND812 Central Processor. BASC-12 translates symbolic mnemonics (source) into machine instructions (object), executable by the Central Processor hardware. The BASC-12 Language is structured to give the programmer complete control over the machine language instructions to be executed by the processor, while allowing easily remembered and understood symbols (letters, numbers, and special characters) to specify the desired instructions. BASC-12 is designed to run in a 4K Processor equipped with paper tape and/or magnetic tape cassette peripherals.

B. PROGRAM AREA

0000_8 through 5303_8 with locations 3053_8 through 4176_8 used as the Input Buffer.

C. STARTING ADDRESS

0200_8.

D. EQUIPMENT CONFIGURATION

Minimum requirements are a 4K ND812 Central Processor equipped with an ASR33 Teletype (low speed punch/reader). Optional peripherals include the high speed punch or reader and/or a Magnetic Tape Cassette Unit.

E. DEFINITIONS

1. Source

The Symbolic Program is written by the programmer in the symbols allowed in BASC-12 language. The program source (usually punched on tape) is the input to the Assembler which then "assembles" or "translates" it into a machine language or "binary" version of of the program. The Symbolic Text Editor (ND41-0002) is usually employed to prepare a "Source" tape.

2. Binary

The output from an assembler is often referred to as the "Object Program". If the Object Program is not produced in directly executable form by the Assembler, it must either be assembled further, or else must be loaded and executed with a run-time monitor which interprets the object coding and performs the required operations. The output from the BASC-12 Assembler may be loaded without modification as it is in absolute binary form, so that, once in core, it is executed directly by the processor without the need for a run-time monitor. To distinguish this type of object program, it will be referred to as the Binary Output or simply "Binary". Therefore, a binary file is a one-to-one copy of core which may be loaded and executed directly.

3. Listing

This consists of a hard-copy symbolic listing of the program with the octal equivalents of the assembled instructions and their octal locations in memory. It represents the documentation of the program, including the actual instructions assembled, and their execution locations. The format used is:

```
2345 6740 SYMBOLIC LINE OF CODING,
```

where "2345" and "6740" are octal numbers representing a memory address and the contents of that address respectively. The balance of the line consists of a repetition of the Program Source which was assembled to yield the machine instruction "6740".

4. Numbers

All absolute numbers are interpreted by the BASC-12 Assembler as four-digit octal numbers. The high order digits of longer numbers will be ignored. The decimal digits "8" and "9" will be interpreted as non-numeric characters.

5. Characters

A single letter, number, etc. When referred to paper tape, a character occupies one frame.

6. Symbols

A symbol consists of any collection of letters, numbers, or other characters appearing anywhere on a line, not in a comment, and followed by a terminator.

NOTE

Spaces are terminators.

A legal symbol should be no more than 6 characters long and must begin with an alphabetic character. Longer symbols are allowed but will be truncated to 6 characters with the excess being ignored. Thus, the symbol "STARTS", "STARTSA", and "STARTSAB" will be interpreted as the same symbol by the Assembler, namely "START".

7. Non-Printing Characters

The following symbols are used to represent non-printing symbols:

Horizontal Tab = ⇥ Carriage Return = ↵
Vertical Tabe = ⤒ Line-Feed = ↓

These characters, although non-printing, may be typed by the user on the keyboard and may appear in a Program Source. They generally initiate some action in the printing device (such as a carriage return) or cause a special response from a program (such as a tab).

A space is represented by the symbol " △ " even though it is a printing character in the sense that it causes the carriage to move one column to the right. It is, however, often important to know how many spaces appear in a line of text, so that it is convenient to have a symbol to represent it in the documentation.. None of these special symbols are never printed, but are used whenever necessary to clarify the contents of a line of text in the examples and discussions which follows.

8. User Entry

User entry is always underscored. All other characters are printed by the program.

9. Ignored Characters

A character "ignored" by the BASC-12 Assembler is reproduced in the listing, but does not affect assembly of the program. Ignored characters will terminate a symbol, but otherwise initiate no special action. The "Space" is an example of an ignored character. If spaces intervene between a symbol and another special character, the assembler interprets the resulting coding as though the spaces were not present. Spaces between symbols serve only to separate the symbols.

10. Terminators

The following characters have special meaning to the Assembler and will also terminate a symbol: Comma (,), Slash (/), Asterisk (*), Plus (+), Minus (-), Equal (=), Number (#), Left Bracket ([ ), At (@), Horizontal Tab (⇥), Vertical Tabe (⤒), Form-Feed, Up Arrow (↑), Carriage-Return (↵), and Dollar ($). Blank tape, rubouts, and line-feeds are not stored in the Assembler buffer. Twenty-four consecutive Rubouts are interpreted as an out-of-tape condition.

Space (△)	Serves only to terminate a symbol. Spaces are otherwise ignored.
Comma (,)	Indicates that the symbol preceding it (ignoring spaces) should be interpreted as a tag for that location. The symbol is set equal to the Program Location Counter (PLC). A symbol may not be used as the tag for two different program locations. Command mnemonics may not be used as tags.
Slash (/)	The slash indicates the beginning of a comment. All characters, including the terminators, following the slash will be ignored by the assembler until the next carriage-return. The Number sign (#) will be flagged as an illegal character. Comments, including the Slash, are ignored and are reproduced only in the Pass 3 Listing.
Asterisk (*)	The asterisk is an instruction to the assembler to reset the program location counter (PLC) to the value of the symbol(s) following the asterisk. If the asterisk is followed by several symbols, they will be assembled according to the rules of assembly applying to a single location and the PLC will be set equal to the result. Numbers are, as usual, interpreted literally. Octal numbers larger than four digits are truncated from the high-order end. If any one of the symbols appearing after the "*" have not been previously defined, the expression will be ignored and the undefined origin (UO) diagnostic printed.
Plus (+)	Causes the symbol following the plus sign to be added to the symbol preceding the sign. An undefined symbol at the time of evaluation is ignored.
Minus (-)	Causes the symbol following the minus sign to be subtracted from the symbol preceding the minus sign.
Equal (=)	Causes the symbol preceding the sign to be set equal to the symbol or symbols following the sign. If the symbol following the equal sign is undefined, the Assembler reacts differently depending on the current pass number and the type and kind of symbols

	involved. The treatment of statements containing an equal sign is discussed under "equivalence Statements".
Number (#)	Denotes an illegal character. A character other than blanks, line-feeds, or rubouts, and with octal code less than 0240 or greater than 0337 is stored in the input buffer as a "#". When this character is encountered by the Assembler during processing of its buffer, the message "IC AT XXXX" is generated where "XXXX" represents the current value of the PLC. If the character "#" appears in the user's source, it will be interpreted as an illegal character during assembly.
Left Brack ([ )	Indicates that the symbol following the " " is to be taken as an Assembler Directive; that is, an instruction to the assembler to interrupt processing and perform some other task indirectly related or not related to processing the source program. The section on Assembler Directives describes the various directives included in the assembler and the procedure necessary to create user directives.
At Symbol (@)	Indicates an indirect address for a memory reference instruction. The "@" symbol should appear immediately after the instruction mnemonic without separating spaces (i.e. "JMP@").
Horizontal Tab (⇥)	Equivalent to a space. When creating the Pass 3 Listing, the assembler will generate the proper number of spaces on the teletype or other output device. The output device need not have mechanical tabbing ability.
Vertical Tab (⤒)	Terminates a line and advances the paper to the top of the next page.
Form-Feed	Interpreted by the Assembler as a vertical tab.
Up Arrow (↑)	Automatically generates a two word I/O where the symbol following the up arrow is taken as the second word of the I/O. The octal code 0740 is automatically generated in the first location of

	the two word instruction. This symbol's action is an exception to the one location per line rule in that it generates two locations for one line of coding. It does not, however, restrict the programmer's flexibility of control over the final binary program since use of ( ) is optional. The up arrow need appear only once on a line. Succeeding up arrows on the same line are ignored. Symbols preceding the up arrow on a line are ignored.
Carriage Return ↵	Carriage Return (↵) terminates a line of coding. A location is not assigned to lines containing an (=) or (*) or blank lines containing no coding or a comment only. Extra carriage-returns in the source, though they generate extra lines in the program listing, do not affect the length of the machine language program and may be used to format the listing for better legibility. It is recommended that blank lines be used to separate logical blocks of the Source and to separate the constants and variables from instructions.
Dollar ($)	The Dollar ($) indicates the end of the Source. The dollar sign will initiate termination of the current pass and so should be the last symbol in a Program Source. If a dollar sign is not read by the Assembler before it runs out of input tape, the assembler waits for teletype input to supply the "$", indicating the end of the pass, or any other character indicating that the input device was reloaded and the assembly should continue with the additional source. The "$" should not be used anywhere in the body of the source except as the last line.
Period or Dot (.)	Is equivalent to a symbol representing the current value of the Program Location Counter (PLC). The period or dot does not serve as a terminator and must be used in all respects like any other symbol.

2. PROGRAM DESCRIPTION

When using the BASC-12 Assembler it is imperative that the user have a solid knowledge of the source language. (See Appendix A and B for a brief listing of these.) The source program consists of ASCII coded binary words that represent symbols, special characters and terminators written in a format acceptable by the BASC-12 Assembler. The following example illustrates the statement format which can be interpreted by the BASC-12 Assembler:

```
LABEL          OPERATION          ADDRESS          /COMMENT

READ,          JMP                END              /INPUT ONE CHARACTER
```

Fields of each instruction line are not rigidly defined. They are separated from one another by any of the terminators except the + and - characters. Plus (+) and Minus (-) are used to connect multiple symbols to form a single field. The + and - signs are evaluated from left to right before a field is connected with another. Parentheses are not permitted.

The commend field must begin with a slash (/). A comment may start anywhere on the line and always continues to the end of the line.

It is suggested, in the interest of neatness and legibility of final output, that the OPERATION field begin in the ninth column (first tab), that the ADDRESS begin in the fourteenth column (automatic provision for inserting an extra space after 3-letter instructions is made in the Editor), and the comment begin in the 25th column (4th tab position; one tab after most instructions). On lines which contain no instructions or definitions, the comment may start anywhere on the line and is generally left-justified. A line containing a comment only is left-justified in Pass 3 output of the Assembler.

A. LABEL FIELD

The label field may consist of one symbol followed by a comma. Plus (+) and Minus (-) signs are not permitted in the Label Field. The symbol may not have been previously defined by appearing in an equivalence statement or by having been used as a tag. There may be more than one tag on a line, but each may consist of only one symbol and each must be followed by a comma. All tags on the same line will reference the same location in the assembled program.

An attempt to use a symbol as a tag which has already appeared in an equivalence statement, or was used as a tag, will result in the duplicate tag error diagnostic "DT AAAAAA AT XXXX" where the location counter at the time the error was detected.

The label field does not affect the contents of the current location.

B. OPERATION AND ADDRESS FIELDS

If, after assembling an element other than a label element, the result is between 2400_8 - 7737_8, the next field will be treated as the address field of a Single-Word Memory Reference Instruction. In this case, no field, except a comment, may follow on the same line. Each symbol beyond the Address field of a Single-Word Memory Reference Instruction will generate the Double Address Diagnostic "DT AAAAAA AT XXXX", where "AAAAAA" is the symbol and "XXXX" is the value of the location counter at the line where the double address was detected.

Since Single-Word Memory Reference Instructions are recognized by their octal coding, it is possible for the user to devise his own mnemonics for special applications. In addition, the assembler will always recognize error conditions involving Single-Word Memory Reference Instructions no matter how the octal code is created. This parallels the fact that the processor also recognizes instructions by their octal codes and errors made in the Source will be carried through to run-time.

It should be noted that the Assembler does not check for the absence of the Address field. If a Single-Word Memory Reference is assembled into a location and no address field is provided, reference to location 0000_8 will be assembled by default.

In any case, proper assembly of a Single-Word Memory Reference Instruction demands that it appear in the first non-label field on the line. Although not essential for proper assembly, the same rule should be applied to all instructions to minimize the possibility of error and to make the Source Listing easiest to read.

Fields evaluated to a number less than 2400_8 or greater than 7377_8 are combined with an inclusive logical "OR". This allows micro-programming of the Operate and I/O instructions.

C. EQUIVALENCE STATEMENT

An Equivalence Statement consists of an "Object Symbol" followed by an equal sign (=) followed by an expression consisting of symbols, plus signs (+), minus signs (-), and spaces (to separate symbols in the absence of a plus or minus sign). The expression is assembled according to the algorithm for the assembly of an instruction and the symbol appearing to the left of the equal sign is set equal to the result.

D. DEFINITION STATEMENT

A Definition Statement is the special case of an Equivalence Statement in which all of the symbols appearing to the right of the equal sign are defined when the statement is encountered. Ultimately, it is the purpose of all Equivalence Statements to assign an octal value to the object symbol or to set the object symbol equal to another symbol which has been assigned an octal value. In other words, it is the purpose of the Assembly to change all Equivalence Statements into Definition Statements. "RD XXXXXX AT NNNN" indicates an attempt to assign a second value to the same symbol. The error diagnostic "DT XXXXXX AT NNNN" is generated if a symbol appearing as the object of an Equivalence Statement also appears as a Program Tag (Where "XXXXXX" is the symbol and "NNNN" is the current value of the Location Counter).

Equivalence Statements are closely related to the use of symbols as Program Tags. Use of a symbol as a Program Tag sets it equal to the Memory Address assigned to the instruction or constant it tags. Since a symbol may have only one octal value, any attempt to also use the symbol as the object of an Equivalence Statement will generate the error diagnostic "RD XXXXXX AT NNNN" and the prior value of the symbol will not be affected.

Similarly, any attempt to use a symbol two or more times as the object of an Equivalence Statement will generate the same error message: "RD XXXXXX AT NNNN" where "XXXXXX" is the symbol and "NNNN" is the current value of the Location Counters.

The Assembly of a Definition Statement is straight-forward if all of the symbols on the right-hand side of the equal sign are defined. The Assembler will assemble the right-hand symbols according to the algorithm used for instruction locations and set the left-hand symbol equal to the result. For example, the statements:

```
DBSKIP = JMP .+3
CSLCT = 7466
```

would cause "DBSKIP" to be set equal to 6003 and "CSLCT" to be set equal to 7466 in the Pass 1 symbol table. Thereafter, the octal number 6003 would be substituted for "DBSKIP" whenever the symbol occurred.

A statement of the form:

```
READ = JPS INPUT
```

is a definition (and is handled as described above) only if both "JPS" and "INPUT" were defined or used as a tag prior to the appearance of the above statement. If "INPUT" is used as a tag or appears in an equivalence statement after the above statement, then assembly of the above statement must be deferred to a later pass at which time "INPUT" will be presumably defined. In this case, "READ" will not appear in the Pass 1 Symbol Table. On the second or subsequent passes, "READ" will be properly defined and listed normally in the Pass 3 Symbol Table. The user should be aware, however, that "READ" remains undefined until it is encountered during processing of Pass 2 or 3 and will generate an Undefined Symbol error diagnostic during Pass 2 or 3 if used prior to the appearance of the statement which defines "INPUT".

If "INPUT" is not defined anywhere in the program, the error diagnostic "UE XXXXXX YYYYYY AT NNNN" (Undefined Equivalence) will be generated during Pass 2 and 3 where "XXXXXX" is the left-hand symbol and "YYYYYY" is the undefined right-hand symbol. The diagnostic will be generated as many times as an undefined right-hand symbol appears. In any case, "READ" will remain undefined and will appear in the Pass 3 Symbol Table as

READ = **

One special circumstance arises if a symbol appears in both a Pass 1 Redefinition or Duplicate Tag error and a Pass 2 or 3 Undefined Equivalence error. For example suppose the following coding were assembled:

```
READ = JPS RD

*0200

READ, 0

$
```

Note that "RD" is never defined and that "READ" appears as the left-hand portion of an equivalence statement and as a program tag.

During Pass 1, the error diagnostic:

```
DT READ AT 0200
```

would be generated and "READ" would not be listed in the Pass 1 Symbol Table.

During Pass 2 or 3, the error diagnostic:

```
UE READ RD AT 0100
```

would be generated but "READ" would remain undefined only until the "READ, 0" statement was encountered. Inasmuch as "READ" is still undefined, the assembler will define it as a tag (giving it the value "0200") and it would appear in the Pass 3 Symbol Table as

READ = 0200

A statement of the form:

READ = INPUT = GT = JPS RD

will cause the following entries in the Pass 1 Symbol Table (assuming "RD" is undefined when the above statement appears):

GET	= **
INPUT	= GET
READ	= INPUT

During Pass 2 or 3, "GET" will be defined, and be listed in the Symbol Table with the appropriate octal number to replace the double asterisk. In any case, "READ" and "INPUT" will be treated by the assembler exactly as though the symbol "GET" were being used. If "GET" is never defined, "READ" and "INPUT" will also remain undefined.

Finally, there may be only one complex expression or previously defined symbol on a line and it must be terminated with a carriage-return. Thus, the following statement is illegal:

READ = JPS RD = JPS INPUT

No error diagnostic will be immediately produced, but "RD" will be listed in the Pass 1 Symbol Table as "RD" will be listed in the Pass 1 Symbol Table as "RD = **" and any subsequent attempts to define it or use it as a tag will generate the Redefinition or Duplicate Tag error messages.

During Pass 2 or 3, the Undefined Equivalence message will be produced and then "RD" will be defined with a somewhat arbitrary value. Additional error diagnostics may be produced depending on the current value of the Location Counter and the current definition status of the rest of the symbols on the line.

Coding of the following forms:

READ, 0

INPUT = READ = GET = FIND

are also illegal and will generate the Redefinition error message.

3. OPERATOR OR USER CONTROL

The BASC-12 General Assembler is a 2 Pass Assembler that examines the source program and creates a table of symbol address on Pass 1, outputs the assembled binary translation of the source program on Pass 2. An optional third pass is possible during which a symbolic listing (in ASCII Code) is created. Passes 2 and 3 are combined on BASC-12 for DISC systems. Switch Register settings specify which pass the assembler is to undertake and what input and output devices are to be utilized. The following is a detailed description of the Switch Register bit assignments:

Bits 0 and 1 indicate which Pass the assembler is to undertake. Pass 0 is illegal

	Bit 0	Bit 1
Pass 1	0	1
Pass 2	1	0
Pass 3	1	1

Bit 2 is unassigned.

Bit 3 set to "1" causes the assembler to generate the symbol table at the end of Pass 1.

Bit 4 is unassigned.

Bit 5 set to "1" suppresses page formatting.

Bits 6 and 7 select the input device.

	Bit 6	Bit 7
Low Speed Reader (TTY)	0	0
High Speed Reader	0	1
None	1	0
Cassette	1	1

Bits 8 and 9 select the output device for Pass 1 and 3. When the non-existent output device is selected, the symbolic listing is destroyed but the error messages are printed. This is of value to a user assembling a rough program and anticipating many errors.

	Bit 8	Bit 9
Low Speed Punch (TTY)	0	0
High Speed Punch	0	1
Non-existent Device	1	0
Non-existent Device	1	1

Bits 10 and 11 select the output device for Pass 2. When the non-existent output device is selected, the binary is destroyed but the error messages are printed.

	Bit 10	Bit 11
Low Speed Punch (TTY)	0	0
High Speed Punch	0	1
Non-existent Device	1	0

An important "control" the user must not overlook, is the understanding of Instruction Codes, formatting acceptable to the assembler, terminators, directives and special characters. This knowledge can be attained by reading the REFERENCE MANUAL, the DEFINITIONS and PROGRAM DESCRIPTION Sections of this manual.

4. OPERATIONAL PROCEDURE

1) Load the BASC-12 General Assembler program tape with the Binary Loader (Refer to the Binary Loader ND41-003 for a detailed description).

2) Set the Switch Register to 0200_8 and depress LOAD AR.

3) Turn on all input and output devices that are intended for use during this assembly.

4) Place the paper tape or the cassette storing a source program in the appropriate input device.

5) Set the pass number (Bits 0 and 1), input device (Bits 6 and 7), pass 2 output device (Bits 10 and 11), and pass 1 and 3 output device (Bits 8 and 9) into the Switch Register. Refer to the OPERATOR or USER CONTROL Section for a detailed description of this operation.

6) Depress START.

7) The source program will now be read and the assembled binary or symbolic translation written on the specified device.

NOTE

The storage buffer of BASC-12 General Assembler is 1024 characters long, excluding line feeds, blank tape and rub-outs. If a source program is contained on paper tape and exceeds the limits of the storage buffer, it will be necessary to reload the source program for each pass of the assembler (return to Step 4 and 5 above and depress CONTINUE in place of START in Step 6).

The Magnetic Tape Cassette Unit incorporates circuitry which automatically rewinds the cassette at the end of tape, allowing the assembler to reload the source program via software control.

5. ERROR DIAGNOSTICS

The error messages generated by the BASC-12 Assembler refer to syntactic errors appearing in the source. A syntactic error will cause a zero to be assembled for the statement in which the error occurs. (This is equivalent to a STOP instruction and will cause the processor to STOP on encountering the incorrectly written instruction when the object program is executed). Numbers in parentheses indicate the pass during which the error diagnostic may occur.

A. IC AT NNNN (ILLEGAL CHARACTER) (1, 2, 3)

An illegal character was detected during input. The character will appear in the Pass 3 Symbolic Text as "#". The illegal character (or "#") is counted as a blank during assembly. Illegal character messages are not counted among the syntactical errors which follow.

B. RD XXXXXX AT NNNN (REDEFINITION) (1)

The symbol "XXXXXX" was encountered as the left-hand portion of an Equivalence Statement for the second time or was previously used as a program tag. The Equivalence is ignored.

C. DT XXXXXX AT NNNN (DUPLICATE TAG) (1)

A symbol "XXXXXX" followed by a comma was encountered and had already been used as a program tag or as the left-hand portion of an equivalence statement. The symbol and comma are ignored.

D. UO XXXXXX AT NNNN (UNDEFINED ORIGIN) (1, 2, 3)

The symbol "XXXXXX" was used in a statement preceeded by a "*" without having been previously defined. An origin specification may consists of any complex expression, but <u>all</u> of the symbols used in the expression must be defined in order for the assembler to reset its Program Location Counter (PLC) to the proper value. It is not possible to defer the definition of an origin since it is required for Pass 1 as well as 2 and 3.

E. US XXXXXX AT NNNN (UNDEFINED SYMBOL) (2, 3)

The symbol "XXXXXX" was used in the program without ever having appeared in a legal Equivalence statement or having been used as a program tag. The symbol will also appear in the Pass 3 Symbol Table followed by the double asterisk (**). The symbol is ignored.

F. IR MMMM XXXXXX AT NNNN (ILLEGAL REFERENCE) (2, 3)

The Symbol "XXXXXX" (with a definition of "MMMM") was used as the address of a Single-Word Memory Reference Instruction and the difference between the Program Location Counter (PLC) and "MMMM" is greater than $63_{10}(77_8)$ placing "XXXXXX" out of range of the Single-Word MRI. This diagnostic is also produced if the address of an "ANDF" instruction is less than the PLC (Reverse Reference), if the absolute octal portion of a literal instruction is greater than 77_8, negative or zero or if any Single-Word instruction attempts to reference itself (MMMM=NNNN=(LC)).

G. DA XXXXXX AT NNNN (DOUBLE ADDRESS) (2, 3)

The address of a Single-Word Instruction may consist of only one symbol not connected by plus or minus signs. For example, the statement: JPS A+B is legal; JPS A B is not and will generate the error message "DA B AT NNNN". The second and succeeding symbols are ignored.

H. UE YYYYYY XXXXXX AT NNNN (UNDEFINED EQUIVALENCE) (2, 3)

The symbol "XXXXXX" appears in the right-hand portion of an Equivalence Statement attempting to define YYYYYY and XXXXXX is undefined. "YYYYYY" will not be defined, regardless of the status of other symbols in the right-hand portion of the Equivalence Statement. The diagnostic is generated for every undefined symbol which appears in the right-hand portion of the statement.

I. UN XXXXXX AT NNNN (UNDEFINED DIRECTIVE) (1, 2, 3)

An Assembler Directive was encountered which is not defined at all, or is a user defined directive which appears in the program before the appearance of the "ENABLE" Directive. It is ignored.

J. ST OV XXXXXX AT NNNN (SYMBOL TABLE OVER-FLOW) (1, 2, 3)

No room could be found in the Symbol Table for the symbol "XXXXXX". The current pass is automatically terminated. The Symbol Table may be extended by 128 symbols with use of the "EXTEND" directive. This, however, risks destruction of the Binary Loader. If the "EXTEND" directive has already been used in the source, it will be necessary to alter the program to use fewer symbols.

K. SYMBOL TABLE DIAGNOSTICS

Except for the double asterisk, no diagnostics are produced in either symbol table. The double asterisk is used to flag "Undefined Symbols", which, after pass 3, consists of all those symbols for which no definition could be assembled. The message "SE MMMM" which appears on the first line of the Symbol Table (Pass 1 or 3) indicates the first Memory Address not used for Symbol Table storage. Programs previously loaded into memory which use no locations less than MMMM are therefore not impaired in any way. The number also allows the user to determine how much room is left for additional symbols. (Each symbol requires 4 memory locations for storage). Maximum Capacity of the symbol table is as follows:

Normal Length (Ending at 6776_8) = 355_{10}
Extended Length (Ending at 7776_8) = 483_{10}

These include the Permanent Symbol Table, which contains 89_{10} symbols. The message "ER XXXX" is generated as the last line of the Symbol Table where "XXXX" is the octal number of syntactical errors occurring in the program during the current Pass. This message appears only at the end of Pass 1 or Pass 3. Illegal characters do not affect this count.

Finally, it should be noted that the Assembler does not check for the absence of the Address element. If a Single-Word Memory Reference is assembled into a location and no address element is provided, reference to location 0000_8 will be assembled by default.

In any case, proper assembly of a Single-Word Memory Reference Instruction demands that the instruction appears in the first non-label element on the line. Although not essential for proper assembly, the same rule should be applied to all instructions to minimize the possibility of error and to make the Source Listing easiest to read.

Elements evaluated to a number less than 2400_8 or greater than 7377_8 are combined with an inclusive logical "OR". This allows micro-programming of the Operate and I/O Instructions.

APPENDIX A
INSTRUCTION MNEMONICS IN ALPHABETICAL ORDER

OP	OCTAL	DESCRIPTION	TIMING
ADDL	22XX	Add Literal to J	1 cy
ADJ	4400	Add to J	2 cy
ADR J	1122	R + J to J	1 cy
ADR K	1222	R + K to K	1 cy
ADS J	1124	S + J to J	1 cy
ADS K	1224	S + K to K	1 cy
AJK J	1120	J + K to J	1 cy
AJK K	1220	J + K to K	1 cy
AJK JK	1320	J + K to J, K	1 cy
ANDF	20XX	AND with J, Forward	2 cy
AND J	1100	AND J, K into J	1 cy
AND K	1200	AND J, K into K	1 cy
AND JK	1300	AND K, J into K, J	1 cy
ANDL	21XX	AND Literal with J	1 cy
CCLF	0141	Clear all Cassette Flags (TWIO)	5 μs
CHSF	0101	High Speed Forward to EOT (TWIO)	5 μs
CHSR	0121	High Speed Reverse to BOT (TWIO)	5 μs
CLR	1410	Clear Flag Register	1 cy
CLR J	1510	Clear J	1 cy
CLR K	1610	Clear K	1 cy
CLR JK	1710	Clear J, K	1 cy
CLR O	1450	Clear Overflow Register	1 cy
CMP	1420	Complement Flag Register	1 cy
CMP J	1520	Complement J	1 cy
CMP K	1620	Complement K	1 cy
CMP JK	1720	Complement J, K	1 cy
CMP O	1460	Complement Overflow Register	1 cy
CRDT	0144	Read Cassette Tape to J (TWIO)	5 μs
CSBT	0130	Skip if Cassette at BOT (TWIO)	5 μs
CSET	0110	Skip if Cassette at EOT (TWIO)	5 μs
CSFM	0104	Skip if Cassette Read File Mark (TWIO)	5 μs

OP	OCTAL	DESCRIPTION	TIMING
CSLCT1	7601	Place Cassette 1 On-Line	1 cy
CSLCT2	7602	Place Cassette 2 On-Line	1 cy
CSLCT3	7604	Place Cassette 3 On-Line	1 cy
CSNE	0122	Skip if No-Error Cassette (TWIO)	5 µs
CSPF	0102	Space Cassette Forward to File Mark (TWIO)	5 µs
CSRR	0142	Skip if Cassette Read Ready (TWIO)	5 µs
CSTR	0124	Skip if On-Line Cassette Ready (TWIO)	5 µs
CSWR	0152	Skip if Cassette Write Ready (TWIO)	5 µs
CWFM	0151	Cassette Write File Mark (TWIO)	5 µs
CWRT	0154	Cassette Write Transfer (TWIO)	5 µs
DIV	1001	Divide J and K by R	11 µs
DSZ	3000	Decrement Memory and Skip	2 cy
EXJR	1103	Exchange J and R	1 cy
EXJKRS	1303	Exchange J, K with R, S	1 cy
EXKS	1203	Exchange K and S	1 cy
HIF	7421	HS Reader Fetch	3 µs
HIR	7422	CLR Flag; Read HS Buffer	3 µs
HIS	7424	Skip if HS Reader Reader	3 µs
HLP	7433	HS Punch Load and Punch	3 µs
HRF	7423	HS Reader Read-Fetch	3 µs
HOP	7431	HS Punch On	3 µs
HOS	7434	Skip if HS Punch Ready	3 µs
INC J	1504	Increment J	1 cy
INC K	1604	Increment K	1 cy
INC JK	1704	Increment J, K	1 cy
IOFF	1003	Disable all Interrupt Levels	1 cy
IONA	1005	Enable Interrupt Levels H & A	1 cy
IONB	1006	Enable Interrupt Levels H & B	1 cy
IONH	1004	Enable Interrupt Level H Only	1 cy
IONL	1007	Enable All Interrupt Levels	1 cy
ISZ	3400	Increment Memory and Skip	2 cy
JMP	6000	Unconditional Jump	1 cy
JPS	6400	Jump Subroutine	2 cy
LDJ	5000	Load J	2 cy
LDJK	7721	Load JPS Reg to J, INT reg to K	1 cy
LDREG	7720	Load JPS Reg From J, INT reg to K	1 cy
LJIPB	7722	Set JPS and INT Status Bits	1 cy
LJKFRS	1302	Load J, K from R, S	1 cy
LJST	1011	Load J from Status Register	1 cy
LJSW	1010	Load J from Switches	1 cy
LKFJ	1204	Load K from J	1 cy
LJFR	1102	Load J from R	1 cy
LKFS	1202	Load K from S	1 cy
LRFJ	1101	Load R from J	1 cy

OP	OCTAL	DESCRIPTION	TIMING
LRSFJK	1301	Load R, S from J, K	1 cy
LSFK	1201	Load S from K	1 cy
MPY	1000	Multiply J by K	10.75 μs
NADR J	1132	-(R + J) to J	1 cy
NADR K	1232	-(R + K) to K	1 cy
NADS J	1134	-(S + J) to J	1 cy
NADS K	1234	-(S + K) to K	1 cy
NAJK J	1130	-(J + K) to J	1 cy
NAJK K	1230	-(J + K) to K	1 cy
NAJK JK	1330	-(J + K) to J, K	1 cy
NEG J	1524	Negate J	1 cy
NEG K	1624	Negate K	1 cy
NEG JK	1724	Negate J, K	1 cy
NSBR J	1133	J-R to J	1 cy
NSBR K	1233	K-R to K	1 cy
NSBS J	1135	J-S to J	1 cy
NSBS K	1235	K-S to K	1 cy
NSJK J	1131	K-J to J	1 cy
NSJK K	1231	K-J to K	1 cy
PIOF	1600	Powerfail system off	1 cy
PION	1500	Powerfail system on	1 cy
RFOV	1002	Read Flag and Overflow Bits from J	1 cy
ROTD J	1160	Rotate J Left N	1 cy +
ROTD K	1260	Rotate K Left N	1 cy +
ROTD JK	1360	Rotate J, K Left N	1 cy +
SBJ	4000	Subtract J	2 cy
SBR J	1123	R-J to J	1 cy
SBR K	1223	R-K to K	1 cy
SBS J	1125	S-J to J	1 cy
SBS K	1225	S-K to K	1 cy
SET	1430	Set Flag Register	1 cy
SET J	1530	Set J to -1	1 cy
SET K	1630	Set K to -1	1 cy
SET JK	1730	Set J, K to -1	1 cy
SET 0	1470	Set Overflow Register	1 cy
SFTZ J	1140	Shift J Left N	1 cy +
SFTZ K	1240	Shift K Left N	1 cy +
SFTZ JK	1340	Shift J, K Left N	1 cy
SIN J	1506	Skip if J Negative	1 cy
SIN K	1606	Skip if K Negative	1 cy
SIN JK	1706	Skip if J, K Negative	1 cy
SIP J	1502	Skip if J Positive	1 cy
SIP K	1602	Skip if K Positive	1 cy
SIP JK	1702	Skip if J, K Positive	1 cy

OP	OCTAL	DESCRIPTION	TIMING
SIZ	1405	Skip if Flag Zero	1 cy
SIZ J	1505	Skip if J = 0	1 cy
SIZ K	1605	Skip if K = 0	1 cy
SIZ JK	1705	Skip if J, K = 0	1 cy
SIZ O	1445	Skip if Overflow Register = 0	1 cy
SJK J	1121	J-K to J	1 cy
SJK K	1221	J-K to K	1 cy
SKIP	6002	Unconditional Skip (Jump)	1 cy
SKPL	1440	Skip if Power Low	1 cy
SMJ	2400	Skip if Memory Not Equal J	2 cy
SNZ	1401	Skip if Flag One	1 cy
SNZ J	1501	Skip if J Not Equal Zero	1 cy
SNZ K	1601	Skip if K Not Equal Zero	1 cy
SIZ JK	1701	Skip if J, K Not Equal Zero	1 cy
SNZ O	1441	Skip if Overflow Register = One	1 cy
STJ	5400	Store J	1 cy
STOP	00XX	Stop Execution	1 cy
SUBL	23XX	Subtract Literal from J	1 cy
TCP	7413	Clear TTY Flag, Print-Punch	3 µs
TIF	7401	TTY Keyboard-Reader Fetch	3 µs
TIR	7402	TTY Load Keyboard into J	3 µs
TIS	7404	Skip if TTY Keyboard Ready	3 µs
TOC	7411	TTY Clear Flag	3 µs
TOP	7412	TTY Clear flag; Print-Punch	3 µs
TOS	7414	TTY Skip if Printer-Punch Ready	3 µs
TWADJ	0400	Two Word Add J	3 cy
TWADK	0450	Two Word Add K	3 cy
TWDSZ	0300	Two Word Decrement Memory and Skip	3 cy
TWISZ	0340	Two Word Increment Memory and Skip	3 cy
TWJMP	0600	Two Word Unconditional Jump	2 cy
TWJPS	0640	Two Word Jump Subroutine	3 cy
TWLDJ	0500	Two Word Load J	3 cy
TWLDK	0510	Two Word Load K	3 cy
TWSBJ	0400	Two Word Subtract J	3 cy
TWSBK	0410	Two Word Subtract K	3 cy
TWSMJ	0240	Two Work Skip if Memory Not Equal J	3 cy
TWSMK	0250	Two Work Skip if Memory Not Equal K	3 cy
TWSTJ	0540	Two Word Store J	3 cy
TWSTK	0550	Two Word Store K	3 cy
XCT	7000	Execute Displaced Instruction	3 cy

"@" adds one cycle to any memory reference instruction.

APPENDIX B
INSTRUCTIONS IN OCTAL SEQUENCE

OCTAL	SYMBOL	DESCRIPTION	TIMING
00XX	STOP	Stop Execution	1 cy
0101	CHSF	Cassette High-Speed Forward EOT (TWIO)	5 µs
0102	CSPF	Cassette Space Forward to File Mark (TWIO)	5 µs
0104	CSFM	Cassette Skip on File Mark (TWIO)	5 µs
0110	CSET	Cassette Skip if EOT (TWIO)	5 µs
0121	CHSR	Cassette High-Speed Reverse BOT (TWIO)	5 µs
0122	CSNE	Cassette Skip No-Error (TWIO)	5 µs
0124	CSTR	Cassette Skip if On-Line Tape Ready (TWIO)	5 µs
0130	CSBT	Cassette Skip if BOT (TWIO)	5 µs
0141	CCLF	Cassette Clear All Flags (TWIO)	5 µs
0142	CSRR	Cassette Skip if Read Ready (TWIO)	5 µs
0144	CRDT	Cassette Read to J (TWIO)	5 µs
0151	CWFM	Cassette Write File Mark (TWIO)	5 µs
0152	CSWR	Cassette Skip if Write Ready (TWIO)	5 µs
0154	CWRT	Cassette Write Transfer (TWIO)	5 µs
0240	TWSMJ	Two Word Skip if Memory Not Equal J	3 cy
0250	TWSMK	Two Word Skip if Memory Not Equal K	3 cy
0300	TWDSZ	Two Word Decrement and Skip	3 cy
0340	TWISZ	Two Word Increment and Skip	3 cy
0400	TWSBJ	Two Word Subtract J	3 cy
0410	TWSBK	Two Word Subtract K	3 cy
0440	TWADJ	Two Word Add J	3 cy
0450	TWADK	Two Word Add K	3 cy
0500	TWLDJ	Two Word Load J	3 cy
0510	TWLDK	Two Word Load K	3 cy
0540	TWSTJ	Two Word Store J	3 cy
0550	TWSTK	Two Word Store K	3 cy
0600	TWJMP	Two Word Unconditional Jump	2 cy
0640	TWJPS	Two Word Jump Subroutine	2 cy
1000	MPY	Multiply J by K	10.75 µs

OCTAL	SYMBOL	DESCRIPTION	TIMING
1001	DIV	Divide J and K by R	11 μs
1002	RFOV	Read Flag, Overflow from J	1 cy
1003	IOFF	Disable All Interrupt Levels	1 cy
1004	IONH	Enable Level H Only	1 cy
1005	IONA	Enable Interrupt Levels H & A	1 cy
1006	IONB	Enable Interrupt Levels H & B	1 cy
1007	IONL	Enable All Interrupt Levels	1 cy
1010	LJSW	Load J from Switches	1 cy
1011	LJST	Load J from Status Register	1 cy
1100	AND J	And J, K into J	1 cy
1101	LRFJ	Load R from J	1 cy
1102	LJFR	Load J from R	1 cy
1103	EXJR	Exchange J and R	1 cy
1120	AJK J	J + K to J	1 cy
1121	SJK J	J - K to J	1 cy
1122	ADR J	R + J to J	1 cy
1123	SBR J	R - J to J	1 cy
1124	ADS J	S + J to J	1 cy
1125	SBS J	S - J to J	1 cy
1130	NAJK J	-(J + K) to J	1 cy
1131	NSJK J	K - J to J	1 cy
1132	NADR J	-(R + J) to J	1 cy
1133	NSBR J	J - R to J	1 cy
1134	NADS J	-(S + J) to J	1 cy
1135	NSBS J	J - S to J	1 cy
1140	SFTZ J	Shift J Left N	
1160	ROTD J	Rotate J Left N	
1200	AND K	AND J, K into K	1 cy
1201	LSFK	Load S from K	1 cy
1202	LKFS	Load K from S	1 cy
1203	EXKS	Exchange K and S	1 cy
1204	LKFJ	Load K from J	1 cy
1220	AJK K	J + K to K	1 cy
1221	SJK K	J - K to K	1 cy
1222	ADR K	R + K to K	1 cy
1223	SBR K	R - K to K	1 cy
1224	ADS K	S + K to K	1 cy
1225	SBS K	S - K to K	1 cy
1230	NAJK K	-(J + K) to K	1 cy
1231	NSJK K	K - J to K	1 cy
1232	MADR L	-(R + K) to K	1 cy
1233	NSBR K	K - R to K	1 cy
1234	NADS K	-(S + K) to K	1 cy
1235	NSBS K	K - S to K	1 cy

OCTAL	SYMBOL	DESCRIPTION	TIMING
1240	SFTZ K	Shift K Left N	
1260	ROTD K	Rotate K Left N	
1300	AND JK	AND J, K into K, J	1 cy
1301	LRSFJK	Load R, S from J, K	1 cy
1302	LJKFRS	Load J, K from R, S	1 cy
1303	EXJKRS	Exchange J, K with R, S	1 cy
1320	AJK JK	J + K to J, K	1 cy
1330	NAJK JK	-(J + K) to J, K	1 cy
1340	SFTZ JK	Shift J, K Left N	
1360	ROTD JK	Rotate J, K Left N	
1401	SNZ	Skip if Flag Register One	1 cy
1405	SIZ	Skip if Flag Register Zero	1 cy
1410	CLR	Clear Flag Register	1 cy
1420	CMP	Complement Flag Register	1 cy
1430	SET	Set Flag Register to One	1 cy
1440	SKPL	Skip on Power Low	1 cy
1441	SNZ O	Skip if Overflow Register One	1 cy
1445	SIZ O	Skip if Overflow Register Zero	1 cy
1450	CLR O	Clear Overflow Register	1 cy
1460	CMP O	Complement Overflow Register	1 cy
1470	SET O	Set Overflow Register to One	1 cy
1500	PION	Powerfail System On	1 cy
1501	SNZ J	Skip if J Not Equal Zero	1 cy
1502	SIP J	Skip if J Positive	1 cy
1504	INC J	Increment J	1 cy
1505	SIZ J	Skip if J = 0	1 cy
1506	SIN J	Skip if J Negative	1 cy
1510	CLR J	Clear J	1 cy
1520	CMP J	Complement J	1 cy
1524	NEG J	Negate J	1 cy
1530	SET J	Set J to -1	1 cy
1600	PIOF	Powerfail System Off	1 cy
1601	SNZ K	Skip if K Not Equal Zero	1 cy
1602	SIP K	Skip if K Positive	1 cy
1604	INC K	Increment K	1 cy
1605	SIZ K	Skip if K = 0	1 cy
1606	SIN K	Skip if K Negative	1 cy
1610	CLR K	Clear K	1 cy
1620	CMP K	Complement K	1 cy
1624	NEG K	Negate K	1 cy
1630	SET K	Set K to -1	1 cy
1701	SNZ JK	Skip if J, K Not Equal Zero	1 cy
1702	SIP JK	Skip if J, K Positive	1 cy
1704	INC JK	Increment J, K	1 cy

OCTAL	SYMBOL	DESCRIPTION	TIMING
1705	SIZ JK	Skip if J, K = 0	1 cy
1706	SIN JK	Skip if J, K Negative	1 cy
1710	CLR JK	Clear J, K	1 cy
1720	CMP JK	Complement J, K	1 cy
1724	NEG JK	Negate J, K	1 cy
1730	SET JK	Set J, K to -1	1 cy
20XX	ANDF	AND with J, Forward	1 cy
21XX	ANDL	AND J Literal	2 cy
22XX	ADDL	ADD J Literal	1 cy
23XX	SUBL	SUBTRACT J Literal	1 cy
2400	SMJ	Skip if J not Equal Memory	1 cy
3000	DSZ	Decrement Memory and Skip	1 cy
3400	ISZ	Increment Memory and Skip	2 cy
4000	SBJ	Subtract from J	2 cy
4400	ADJ	Add to J	2 cy
5000	LDJ	Load J	2 cy
5400	STJ	Store J	2 cy
6000	JMP	Unconditional Jump	2 cy
6002	SKIP	Unconditional Skip	1 cy
6400	JPS	Jump Subroutine	1 cy
7000	XCT	Execute Displaced Instruction	1 +n
7401	TIF	TTY Keyboard-Reader Fetch	3 μs
7402	TIR	TTY Keyboard Into J	3 μs
7404	TIS	TTY Skip if Keyboard Ready	3 μs
7411	TOC	TTY Clear Flag	3 μs
7412	TOP	TTY Clear Flag, Print-Punch	3 μs
7413	TCP	TTY Clear Flag, Print-Punch	3 μs
7414	TOS	TTY Skip if Printer-Punch Reader	3 μs
7421	HIF	HS Reader - Fetch	3 μs
7422	HIR	HS Reader - CLR Flag, Read Buffer	3 μs
7431	HOP	HS Punch - Punch On	3 μs
7432	HOL	HS Punch - CLR Flag, Load Buffer	3 μs
7433	HLP	HS Punch - Load and Punch	3 μs
7434	HOS	HS Punch - Skip if punch ready	3 μs
7601	CSLCT1	Cassette - Unit 1 On-Line	3 μs
7602	CSLCT2	Cassette - Unit 2 On-Line	3 μs
7604	CSLCT3	Cassette - Unit 3 On-Line	3 μs
7720	LDREG	Load JPS Reg from J, INT Reg from K	1 cy
7721	LDJK	Load JPS Reg to J, INT Reg to K	1 cy
7722	LJIPB	Set JPS and INT Status Bits	1 cy

"@" adds one cycle to any memory reference instruction.

APPENDIX C
ASCII CHARACTER SET

CHARACTER	ASCII CODE	CHARACTER	ASCII CODE
A	301	0	260
B	302	1	261
C	303	2	262
D	304	3	263
E	305	4	264
F	306	5	265
G	307	6	266
H	310	7	267
I	311	8	270
J	312	9	271
K	313	$	244
L	314	*	252
M	315	+	253
N	316	!	254
O	317	-	255
P	320	.	256
Q	321	/	257
R	322	;	273
S	323	=	275
T	324	Space	240
U	325	Tab	211
V	326	Line Feed	212
W	327	Form Feed	214
X	330	Carriage Return	215
Y	331	Rubout	377
Z	332		