SENTRY

FST-1 ASSEMBLER MANUAL

FAIRCHILD
SYSTEMS TECHNOLOGY
A DIVISION OF FAIRCHILD CAMERA AND INSTRUMENT CORPORATION

FST-1 Assembler Manual

STOCK NO. 67094951
ISSUED: September 1970

Fairchild Systems Tech.
1725 Technology Drive
San Jose, California 95110

FAIRCHILD
SYSTEMS TECHNOLOGY

TABLE OF CONTENTS

Page

SECTION I OPERATING INSTRUCTIONS

1.1 Introduction 1-1
1.2 Hardware Configuration 1-1
1.3 Loading the Program 1-1
1.4 Messages 1-2

SECTION II SYNTAX

2.1 Introduction 2-1
2.2 Character Set 2-1
2.3 Record Format 2-1
Comment Records 2-1
Statement Records 2-1
2.4 Label Field 2-2
2.5 Operation Field 2-2
2.6 Operand Field 2-2
Expression List 2-2
Symbols 2-3
Decimal/Octal Integers 2-3
Strings 2-3
Current Location Reference 2-4
Operators 2-4
2.7 Comments Field 2-4

SECTION III INSTRUCTIONS

3.1 Introduction 3-1
3.2 Indexable Instructions 3-1
3.3 Indexable Instructions 3-2
3.4 SST, RST 3-2
3.5 BAT, BOI Augments 3-2
3.6 SPU Augments 3-3

SECTION IV DIRECTIVES

4.1 Introduction 4-1
4.2 BSS 4-1
4.3 DATA 4-1
4.4 ORG 4-2
4.5 EQU 4-2

SECTION IV DIRECTIVES (Continued) Page

4.6 PZE 4-3
4.7 PROC 4-3
4.8 CALL 4-4
4.9 LIST/NOLIST 4-4
4.10 OBJ/NOOBJ 4-4
4.11 SYM 4-4
4.12 PAGE 4-5
4.13 ABS 4-5
4.14 END 4-5
4.15 TPASS 4-5
4.16 INSEQ 4-5
4.17 NOSEQ 4-5

SECTION V DIRECTIVES

5.1 Introduction 5-1
5.2 Symbolic Output 5-1
5.3 Symbol Table Output 5-2
5.4 Object Output 5-2

APPENDIX A INSTRUCTION MNEMONICS

A.1 (Opcodes Sorted by Ascending Alpha Opcode) A-1
A.2 (Opcodes Sorted by Ascending Octal Opcode) A-4

LIST OF TABLES

2-1 Character Coding 2-2

List of Effective Pages

FST - 1 ASSEMBLER

The total number of pages of this publication is 31, consisting of the following:

Page No.	Issue
Title	Original
Warranty	Original
i through v	Original
1-1 through 1-2	Original
2-1 through 2-5	Original
3-1 through 3-3	Original
4-1 through 4-5	Original
5-1 through 5-3	Original
A1 through A6	Original

PREFACE

This document describes the FST-1 Assembler. It does not describe each possible instruction in detail. (If the Appendix is not satisfactory to the user in this regard, he should consult the FST-1 Systems Manual.)

The FST-1 Assembler operates either in a 1 - Pass or 2 - Pass mode and will generate either absolute or relocatable object programs (depending upon which is specified).

SECTION I

OPERATING INSTRUCTIONS

1.1 INTRODUCTION

This section is concerned primarily with the mechanical motions required in using the assembler; that is, in getting a program assembled. Subsequent sections will give the information necessary to write programs that are acceptable to the assembler.

1.2 HARDWARE CONFIGURATION

The minimum hardware required by the assembler is:

1) 4K Core Memory
2) Console typewriter with paper tape reader/punch
3) Disc File

The other I/O devices supported by the MONITOR assembler are the card reader, line printer, and the magnetic tape unit. The card code required is that of the 029 keypunch (EBCDIC).

Additional memory, if available, enables larger programs to be assembled. For two-pass assemblies the only limitation on the size of the program is in the number of labels present. For 4K this about 250 and increases by 1365 for each additional 4K. For single-pass assemblies the size of the program depends upon the number of labels, the number of external references/declarations, and the fragmentation of the program. Assuming one label per five statements a 4K core memory will handle about 1200 assembled instructions and/or data in the one-pass mode, with each additional 4K adding another 2275. Also, note that there are further restrictions to using the one-pass mode, viz: symbolic referencing may be forward only and operand-address arithmetic is not permitted.

1.3 LOADING THE PROGRAM

The program source may be loaded from the console typewriter keyboard, paper tape, cards, magnetic tape or a disc file.

Examples of Commands appropriate to the assembler are found in the DOPSY Manual, section 3.3.

If while assembling, the card reader should run out of cards the last card at the output station will not be read until the card reader returns to the ready state. Therefore, the last card of any input deck should be blank.

1.4 MESSAGES

The following table shows the messages that can be produced during an assembly and the action that is required before proceeding.

Message Text	Action
'SYMBOL TABLE OVERFLOW'	The assembly is terminated.
'ERROR - - SYSTEM-5	The assembler cannot be found, return to the monitor is automatically made.
'MAGTAPE I/O CONFLICT'	A return to the monitor is automatically made.

SECTION II

SYNTAX

2.1 INTRODUCTION

This section gives the information necessary to produce programs in the format required by the assembler.

2.2 CHARACTER SET

Letters	A, B, C, ...Z, and $
Digits	0, 1, 2, 3, 4, 5, 6, 7, 8, 9
Special	! " # % & ' ( ) * + , - . / : ;
Characters	< = > ? @ [\ ] ↑ SPACE

Table 2-1 on the following page shows the internal code for the character set. These are the printing ASCII characters and are produced directly by the teletype. The 029 keypunch character set differs from the teletype character set in a few instances. These are indicated in this table under the '029' heading.

2.3 RECORD FORMAT

The assembler obtains its input from either cards, paper tape, magnetic tape or disc files. Records obtained from the card reader are fixed length and occupy the first 72 characters of the record. Paper tape records are variable in length and are terminated by a carriage return; all other control characters are ignored. Disc files are normally variable length string files.

Assembler records are of two types: comment records and statement records.

2.3.1 Comment Records

Comment records are characterized by having an '*' as the first character of the record. These records have no effect on the assembly process. They do occur in the assembly listing, however, and serve to document or explain program segments.

2.3.2 Statement Records

A statement record is the basic unit of an assembly language program. The assembler requires that the first character of a record should be either alphabetic or a blank.

Table 2-1. Character Coding

Code	Char.	029	Code	Char.	029
00	SPACE		40	@	
01	!		41	A	
02	"		42	B	
03	#		43	C	
04	$		44	D	
05	%		45	E	
06	&		46	F	
07	'		47	G	
10	(		50	H	
11	)		51	I	
12	*		52	J	
13	+		53	K	
14	,		54	L	
15	-		55	M	
16	.		56	N	
17	/		57	O	
20	0		60	P	
21	1		61	Q	
22	2		62	R	
23	3		63	S	
24	4		64	T	
25	5		65	U	
26	6		66	V	
27	7		67	W	
30	8		70	X	
31	9		71	Y	
32	:	0-8-2	72	Z	
33	;		73	[	<
34	<	12-0	74	\	¬
35	=		75	]	>
36	>	11-0	76	↑	┤
37	?		77	←	—

If the character is alpha, then it is assumed to be the first character of a label; if it is blank, then no label exists. Apart from this limitation, the input format is free field through column 72 of the card and consists of 1 - 4 fields separated by one or more blanks: label (optional), op-code (mandatory), operand (optional) and comments (optional).

2.4 LABEL FIELD

The label field is optional. It provides the user with a means of referencing a statement from different areas of his program.

The label has the same syntax as a symbol (2.6.2) and like a symbol may be any length, but only the first six characters are retained. The label must begin with the first character of the record and terminate with a SPACE.

2.5 OPERATION FIELD

The operation field specifies either a machine instruction or a directive. This field must be present and begins with the first non-blank character after the label terminator. The syntax of the operation field is the same as for symbols. The first special character terminates the field and if it is an '*' the instruction will be generated with indirect addressing, otherwise this character must be a space. APPENDIX A contains a list of all the symbols that may occur in the operation field. The user may extend this list by means of the EQU directive (4.5).

2.6 OPERAND FIELD

The operand field consists of one or more expressions. These expressions are used to represent storage locations, index registers and other constant values for instructions, and to provide information required by the assembler to process directives correctly.

Operand expressions may be uni-term or multi-term; however, in the single-pass mode, multi-term expressions cannot contain forward references (symbols that are not defined prior to their use).

The operand field begins with the first non-blank character after the operation field terminator. If the last (or only) expression of the operand field is blank it is assumed to be zero.

2.6.1 Expression List

The series of expressions making up the operand must be separated by a comma and terminated by a space. The elements that make up expressions are symbols, numbers (octal, decimal, string), the current location symbol and operators.

A single-term expression such as A5, 55B, or * has the value of the term itself while multi-term expressions, such as TABLE+3, are reduced by the assembler to a single value by arithmetically combining the terms.

An expression is absolute or relocatable depending on the terms that comprise it. (See 2.6.2). The relocatability of an expression can be determined by replacing each relocatable term with a '1' and each absolute term with a '0' and then evaluating the expression. If during the evaluation the value goes negative or exceeds one an error condition exists. A '0' result indicates an absolute expression and '1' a relocatable one.

2.6.2 Symbols

A symbol is either a single letter or a letter followed by one or more letters and/or digits. Symbols may be any length, but only the first six characters are retained by the assembler. They must, therefore, be unique through the first six characters. Good practice dictates that symbols should be limited to six characters because of the risk of unintentional duplication. With the exception of CALL operands, every symbol occurring in an operand expression must occur as a label somewhere else in the program. The symbol then becomes defined and the value assigned to it is the operand expression for the EQU directive or the current location counter for instructions and other directives.

The values assigned to symbols are either absolute or relocatable. An absolute value is one that is unaffected by the relocation of the program that contains it. A symbol can be assigned an absolute value only by using the EQU directive to equate it to an absolute numeric quantity or another absolute symbol.

Symbols referenced before they are defined are called forward references and cannot be used in multi-term expressions in the single-pass mode.

Examples:

```
TEST10
A15
AVERYLONGSYMBOL
$15C
```

2.6.3 Decimal/Octal Integers

A decimal number is a sequence of 8 or fewer digits. Only the low-order 24 bits of decimal numbers are retained. An octal number is a sequence of 8 or fewer digits terminated by a 'B'. If more than 8 digits are present, only the low-order eight digits are retained.

Examples:

```
1579
10B = 8
7777B = 4095
7B = 7
```

2.6.4 Strings

A string is a sequence of four (or fewer) characters enclosed in single quotes. Any of the characters in table 2.2 except the single quote may be part of a string.

Strings are used in expressions as 24-bit binary numbers. If more than four characters are present, only the first four are retained and if there are less than four characters they are left justified in the word.

```
‘△△△A’ = 41B
    ‘B’ = 42000000B
    ‘#!$@’
```

2.6.5 Current Location Reference

The value of an ‘*’ in an operand expression is that of the current location counter. That is, the value is the location of the current instruction or data item.

Examples:

```
BRU     *+3
DATA    *+1, 3, *+1, ‘EOF’, ‘ON C’, ‘R’
```

2.6.6 Operators

Strings, numbers, identifiers and ‘*’ can be combined by the operators, + (plus), - (minus), ↑ (up arrow). ‘Plus’ and ‘Minus’ can be either unary or binary while ‘up arrow’ is always unary. ‘Plus’ and ‘Minus’ have the usual arithmetic meaning of add and subtract. ‘↑a’ has the value ‘a’ if ‘a’ is even, ‘a + 1’ if it is odd: its primary use is for aligning the location counter on an even boundary. *This is very important when working with two-word operands, using the ‘double’ codes.*

Examples:

```
L + 3
TABLELAST-TABLEFIRST + 3
-6
↑*
```

2.7 Comments Field

The comments field is not processed by the assembler. It does occur in the assembly listing, however, and can contain any of the characters shown in Table 2-1.

Example:

```
LDA TABLE+1,XP GET TABLE ENTRY
```

Note that the operand field *cannot* be omitted if the comment field is present because the assembler would assume the comment field is the operand field.

SECTION III

INSTRUCTIONS

3.1 INTRODUCTION

The operand of every instruction must have a particular format. Appendix A contains a list of all of the legal mnemonics, with a reference to the section describing the type of operand required by the mnemonics. If no reference is given no operand is required.

Operands are expression lists where the expressions reference memory locations or hardware features such as the comparison indicator, index registers, or state flip-flops. Some general comments concerning these expressions are:

- Expressions referencing hardware elements must be absolute (not relocatable). The magnitude of these expressions must not exceed 7 for index registers or 15 for states and indicators.
- An address expression is automatically truncated to 14 bits for regular instructions and to 10 for augmented ones. In the latter case the address expression *cannot* be relocatable.

In subsequent subsections the following notation is used.

- (address) is an expression for the instruction operand address,
- (state) is an expression referencing a state flip-flop,
- (indicator) is an expression referencing the comparison indicator,
- (index) is an expression referencing an index register,
- Items enclosed in square brackets '[' , ']' denote optional items,
- '. . .' is read as 'zero' or more of the following elements'.

3.2 INDEXABLE INSTRUCTIONS

If the instruction (opcode) can be indexed its operand must be in the following form.

(address) [,(index)]

Examples:

```
LDA     TABLE-1, 5
STA     TEMP1
BRU     L2
BAH     * + 1
LDA*    0,UTX1
```

3.3 NON INDEXABLE INSTRUCTIONS

If the instruction cannot be indexed, its operand must be in one of the following forms:

(index) [,(address)]
(state), (address)
(indicator), (address)

Examples:

```
LAX     X3
LXA     X5
STX     XR1, TEMP
BOS     10,L1    TEST SS 1
BOI     3,LEQ
LDX     6, -2
```

The values of the states (switches) that can be tested by BOS are 0-15. These have the value shown below:

0-7	Defined by programmer (See SST, RST)
8	Interrupt Enable
9	Overflow
10-15	Console switches 1-6 respectively

3.4 SST, RST

RST and SST can be used to turn the eight programmable switches interrupt enable and the overflow flag on or off. Their operands have the following form:

(state) . . . , (state)

Example:

```
SST     PASS1
RST     0, 4, 6, EXIT
```

3.5 BAT, BOI AUGMENTS

These instructions require operands of the following form:

(address)

Example:

```
BNE     L1
BP      L2
BGE     * + 2
```

3.6 SPU AUGMENTS

These instructions require an operand expression that is the device number of the peripheral to be affected. This expression must be absolute and less than 200_8 in magnitude. A complete list of the instruction mnemonics may be found in the Systems Reference Manual.

Examples:

```
ARD     40B
STST    60B
```

SECTION IV

DIRECTIVES

4.1 INTRODUCTION

A directive is a command to the assembler that allows the user to describe or select assembly options or to specify such elements as groups of data, character strings, or storage areas.

The format of directive records is the same as that given for instruction records.

4.2 BSS

```
[LABEL]     BSS     (expression)
```

This statement saves a block of storage N words in size; N is the value of the operand expression. This expression must be absolute and cannot contain any forward references.

The label, if present, references the first word of the block.

Examples:

```
            BSS     100B
  TABLE     BSS     50
```

4.3 DATA

```
[LABEL]     DATA (expression)[. . .,(expression)]
```

Each expression in the operand of the DATA statement generates on 24-bit binary value. The label, if present, references the first operand expression. The DATA statement provides a means of entering constants and data into the program.

Examples:

```
O10      DATA      10B
MSG3     DATA      5, *, *+1, ‘ERRO’, R ON’, ‘TES’, ‘TER.’,‘1’
GA       DATA      ‘A’
DM3      DATA      -3
```

4.4 ORG

```
     [LABEL]          ORG (expression)
```

ORG sets the value of the current location counter to the value of the operand expression which must be completely defined; i.e., it can contain no forward references.

The label, if present, is assigned the value of the location counter *before* the counter is assigned its new value.

The special expression ↑* is used to force an even boundary for the operands of 'double' instructions. ('*' is taken as the value of the current location counter. See 2.6.5 and 2.6.6).

An 'ORG 0' is assumed if none is given. A relocatable assembly is assumed to be assembled relative to '0' and all ORG statements are relative to '0' for relocatable assemblies.

Examples:

```
AORG    ORG         100B
        ORG         AORG
```

4.5 EQU

```
     LABEL       EQU (expression) [,(expression)]
```

EQU is used to assign values to symbols. It does not generate object program code.

The label is assigned the value of the first operand expression; the expression cannot contain any forward references. If a second expression is present its value is entered into the symbol table to further define the symbol as an opcode mnemonic. The second expression must be absolute and cannot contain forward references.

EQU directives assigning absolute values to symbols must occur before the symbol is referenced. The EQU directive will produce an 'R' error if this restriction is violated.

The second expression defines the opcode and operand formats according to the following table:

Bits	Meaning	
0-2	Operand Type	
	0	User - operand expressions are ORed with opcode value
	1	(address)
	2	(address) [,(index)]
	3	(index)
	4	(index), (address)
	5	(indicator/state), (address)

Bits		Meaning
	6	(state) [. . . . , (state)]
3	0	Not Augmented
	1	Augmented
4-5	Opcode Type	
	1	No Operand Required
	2	Operand Required

Examples:

```
        A      EQU      3
        STO    EQU      14000000B,42B
        DO     EQU      ZERO
```

4.6 PZE

```
[LABEL] PZE 0
```

The label is assigned the value of the location counter. The PZE "instruction" forces a word of zeros; it is generally used as the entry point of an internal subroutine, whereas PROC (See 4.7), is used as the entry point for subroutines which are called externally.

4.7 PROC

```
LABEL          PROC        [(expression)]
```

The label is assigned the value of the location counter. In addition, a record is placed in the object program that allows the relocating loader (ROL) to link CALL directives to the generated PZE; the CALL statements may be in the current assembly or other ones.

If the operand expression is non-zero, the loaders assume that the PROC statement is an entry point to an interrupt service routine. The expression value is the location of the interrupt entry address and the loader will establish a linkage at the location to the PROC statement. This expression must be absolute and cannot exceed 64 in magnitude. The PROC statement can also be linked to CALL directives when the operand is non zero.

This directive is also used to specify the entry point to the main program. This is done by using the label 'MAINPR'.

Note that the label of a PROC directive is treated like any other label and must be unique or a duplicate label message is issued.

Examples:

```
        SIN        PROC
        INTI       PROC    1
        MAINPR     PROC    0
```

4.8 CALL

```
[LABEL]     CALL          (symbol)
```

The label, if present, is assigned the value of the current location counter. A record is placed in the object program that will allow the relocating loader to link the generated BSM to the PROC directive whose label symbol matches the operand symbol of the CALL statement. CALL can be used to link to PROC statements in the same assembly or an independent assembly.

Example:

```
                        CALL     SIN
        CALLT2          CALL     TEST2
```

4.9 LIST/NOLIST

These two directives may be used to control which portions of the program will produce an assembly listing.

Statements containing errors or warnings are listed independently of LIST.

Example:

```
        LIST
        NOLIST
```

4.10 OBJ/NOOBJ

```
OBJ       expression
```

These two directives may be used to control which portions of the program will be placed in the object program.

OBJ allows an operand expression that is used to specify the maximum number of instructions that will be placed in a single object record. This number must be in the range of 1 - 16 and if unspecified is assumed to be eight (8).

Example:

```
        OBJ         Means:  produce object program for following statements.
        NOOBJ       Means:  do not produce object program.
```

4.11 SYM

This directive causes the symbol table to be listed after the assembly listing.

All symbols that are either EXTERNAL, NOT USED, or UNDEFINED are listed independently of SYM.

Example: SYM

4.12 PAGE

The assembly and symbol table listings produced by the assembler are formatted on 8 ½" x 11" pages. The PAGE directive will force a top-of-form; i.e., the PAGE record will be the first line listed on the next page.

In the case of the teleprinter, however, the PAGE directive will not cause another TOF if the next line to be printed is the 'top-of-form' line.

Example: PAGE

4.13 ABS

This directive indicates that the object program produced is not to be relocated when it is loaded. That is, assembly addresses and execution addresses are the same.

This statement must occur before any object code is produced or the assembly will be produced subject to relocation at load time.

Example: ABS

4.14 END

This statement is a signal to the assembler that the end of the source program has been reached. All source programs must have this statement in order to terminate the assembly.

Example: END

4.15 TPASS

This directive informs the assembler that it is to assemble the program in two passes rather than one.

Example: TPASS

4.16 INSEQ

This directive informs the assembler to test the source records for an assending sequence of numbers in columns 73 - 80.

4.17 NOSEQ

This statement nullifies the INSEQ directive.

SECTION V

ASSEMBLER OUTPUT

5.1 INTRODUCTION

The assembler produces the following types of hard copy output.

1) Assembly listing
2) Symbol table listing
3) Object program listing

In this section the formats of these outputs will be described and explained.

5.2 SYMBOLIC OUTPUT

The symbolic output from the assembler has two formats:

1) Comment records, PAGE, LIST, NOLIST, NOOBJ, SYM, ABS, and TPASS statements (Source Statement)
2) Other Records
   EEEELLLLL VVVVVVVV
   (Source Statement)

EEEE	-	error code
LLLLL	-	value of current location counter
VVVVVVVV	-	value stored at that location or value of operand expression of some directives; BSS and EQU for example.

The error code can be up to four characters. A description and explanation of these characters is shown in the following table.

Character	Description
D	Duplicate label
L	Label error - label is not an identifier or not terminated by a space.
O	Opcode error - opcode is not an identifier; opcode is not in symbol table.

Character	Description
U	Undefined operand identifier - symbolic index register, state or switch is not defined yet; forward reference is used in multi-term expressions.
S	Syntax error - operand contains illegal operator; magnitude of expression is too large.
R	Relocation error - two relocatable expressions are being added; a relocatable expression is being subtracted from an absolute one, a relocatable expression is being used to reference a hardware element.
N	A number with more than 8 characters
X	Monitor control record
↑	Out of sequence record.

5.3 SYMBOL TABLE OUTPUT

A single symbol table entry produces the following symbol table output:

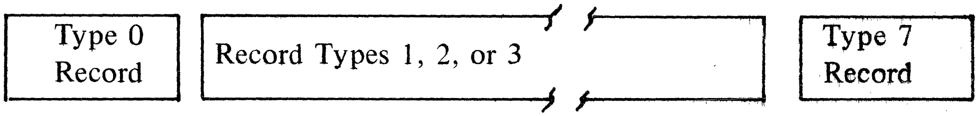
```
SYMBOL   C   VVVVVVVV
```

C is one of the characters SPACE, U, N, or E where U stands for undefined, N for not used and E means the symbol is the label of a PROC directive or the operand of a CALL.

The output will contain three entries per line.

5.4 OBJECT OUTPUT

The object program produced by the assembler consists of a series of records. There are five types of records produced. A typical object program has the following format:



Each record has a two word record header. The first word of each record header has the following format:

Bits	Description
23-21	Record Type
20-15	Size of Record Body
13-0	Address

A complete description of the fields in the record header and of the record body is given in the following table:

Record Type		Word	Bits	Description
START	0	1	13-0	Relocation base (normally 0).
		2	0	Relocatable/absolute (1/0).
LOAD	1	1	13-0	First location loaded by record.
		2	23-0	Relocation indicators. Bit 0 for first instruction, 1 for second, etc.
		3, 4, 5...26		Instructions/data to be loaded.
PROC	2	1	13-0	Address of procedure entry point.
		2	23-0	First 4 characters of name.
		3	23-12	Last 2 characters of name.
CALL	3	1	13-0	Address of CALL instruction.
		2	23-0	First 4 characters of name of called procedure.
		3	23-12	Last 2 characters of name of called procedure.
END	7	1	13-0	Maximum value location counter attained.
			23	Checksum flag
			22-0	Checksum

APPENDIX A

INSTRUCTION MNEMONICS

A.1 (OPCODES SORTED BY ASCENDING ALPHA OPCODE)

OPCODE	MNEMONIC	CODE DESCRIPTION	CYCLES	REFERENCE TO OPERAND TYPE (SECTION)
	ABS	ABSOLUTE PROGRAM LOCATOR		4.13
20000000	ADD	ADD	2	3.2
26000000	AND	LOGICAL AND	2	3.2
36000000	AOM	ADD ONE TO MEMORY	4	3.2
06403400	ARD	ALTERNATE READ	1	3.6
06613400	ARDS	ALTERNATE READ STATUS	1	3.6
06422400	ASPAC	ALTERNATE SPACE	1	3.6
11000000	ATX	ADD TO INDEX	2	3.3
07000000	AUG	AUGMENT		3.2
06423400	AWRIT	ALTERNATE WRITE	1	3.6
00000000	BAH	BRANCH AND HALT	1	3.2
02000000	BAT	BRANCH ON A-REGISTER TEST	1	3.3
03040000	BBC	BRANCH BIT COMPARE	1	3.5
03200000	BE	BRANCH IF EQUAL	1	3.5
03400000	BG	BRANCH IF GREATER	1	3.5
03600000	BGE	BRANCH IF GREATER OR EQUAL	1	3.5
03100000	BL	BRANCH IF LESS	1	3.5
03300000	BLE	BRANCH IF LESS OR EQUAL	1	3.5
02100000	BN	BRANCH IF NEGATIVE	1	3.5
03500000	BNE	BRANCH NOT EQUAL	1	3.5
02500000	BNEZ	BRANCH IF NOT EQUAL TO ZERO	1	3.5
02300000	BNZ	BRANCH IF NEGATIVE OR ZERO	1	3.5
02040000	BO	BRANCH IF ODD	1	3.5
03000000	BOI	BRANCH ON INDICATOR	1	3.3
04000000	BOS	BRANCH ON STATE	1	3.3
04440000	BOV	BRANCH ON OVERFLOW	1	3.5
02400000	BP	BRANCH IF POSITIVE	1	3.5
02600000	BPZ	BRANCH IF POSITIVE OR ZERO	1	3.5
01000000	BRU	BRANCH UNCONDITIONAL	1	3.2
12000000	BSM	BRANCH STORE RETURN AT M	2	3.2
	BSS	BLOCK STORAGE SIZE		4.2
13000000	BSZ	BRANCH STORE RETURN AT ZERO	2	3.2

OPCODE	MNEMONIC	CODE DESCRIPTION	CYCLES	(SECTION)
02200000	BZ	BRANCH IF ZERO	1	3.5
12000000	CALL	SUBROUTINE CALL		4.8
23000000	CAM	COMPARE A WITH MEMORY	2	3.2
30000000	DADD	DOUBLE ADD	3	3.2
	DATA	DATA DEFINITION		4.3
35000000	DIV	DIVIDE	26	3.2
31000000	DLD	DOUBLE LOAD	3	3.2
07034000	DSA	DOUBLE SHIFT AROUND		3.2
07036000	DSL	DOUBLE SHIFT LEFT		3.2
07016000	DSN	DOUBLE SHIFT NORMALIZED		3.2
07030000	DSR	DOUBLE SHIFT RIGHT		3.2
33000000	DST	DOUBLE STORE	3	3.2
32000000	DSUB	DOUBLE SUBTRACT	3	3.2
07014000	DTC	DOUBLE TWO'S COMPLEMENT	2	3.2
	END	PROGRAM TERMINATOR		4.14
21000000	EOR	EXCLUSIVE OR	2	3.2
	EQU	EQUIVALENCE		4.5
06010000	ETST	ERROR TEST	1	3.6
07010000	EXC	EXCHANGE A AND E	1	3.4
06051500	FSKIPB	SKIP FILE BACKWARD	1	3.6
06041500	FSKIPF	SKIP FILE FORWARD	1	3.6
07012400	IDA	INTERRUPT DISABLE	1	3.4
07004400	IEN	INTERRUPT ENABLE	1	3.4
	INSEQ	CHECK SEQUENCE NUMBERS	1	4.16
13000000	LAX	LOAD A FROM INDEX	1	3.3
24000000	LDA	LOAD A-REGISTER	2	3.2
25000000	LDE	LOAD E-REGISTER	2	3.2
07032000	LDS	LOGICAL DOUBLE SHIFT		3.2
05000000	LDX	LOAD INDEX	1	3.3
	LIST	PRODUCE ASSEMBLY LISTING		4.9
07022000	LS	LOGICAL SHIFT A		3.2
07000000	LXA	LOAD INDEX FROM A	1	3.2
34000000	MUL	MULTIPLY	25	3.2
	NOLIST	NO ASSEMBLY LISTING		4.9
	NOOBJ	NO OBJECT PROGRAM		4.10
10000000	NOP	NO OPERATION	1	3.2
	NOSEQ	STOP SEQUENCE CHECK		4.17
	OBJECT	PRODUCE OBJECT PROGRAM		4.10
27000000	OR	OR (INCLUSIVE)	2	3.2
	ORG	ORIGINATION CONTROL		4.4
	PAGE	PAGINATION CONTROL		4.12

OPCODE	MNEMONIC	CODE DESCRIPTION	CYCLES	(SECTION)
06001000	PCOMP	PRIORITY COMPLETE	1	3.6
06011000	POFF	PRIORITY OFF	1	3.6
06013000	PON	PRIORITY ON	1	3.6
00000000	PROC	SUBROUTINE ENTRY POINT		4.7
00000000	PZE	POSITIVE ZERO (ENTRY PT)		3.2
06401400	RD	READ	1	3.6
06611400	RDS	READ STATUS	1	3.6
06501500	RDT	READ (MAGNETIC) TAPE	1	3.6
06601400	RDTT	READ TELETYPE	1	3.6
06611700	REWC	READ EXCESS WORD COUNT	1	3.6
06000500	REWIND	REWIND TAPE	1	3.6
06011500	RSKIPB	SKIP RECORD BACKWARD	1	3.6
06001500	RSKIPF	SKIP RECORD FORWARD	1	3.6
07006000	RSR	READ SWITCH REGISTER	1	3.4
07012000	RST	RESET STATE	1	3.4
17000000	RUM	REPLACE UNDER MASK	2	3.2
07024000	SA	SHIFT A AROUND LEFT		3.2
06461500	SKWR	SKIP AND WRITE	1	3.6
07026000	SL	SHIFT A LEFT		3.2
37000000	SOM	SUBTRACT ONE FROM MEMORY	4	3.2
06420400	SPAC	SPACE	1	3.6
06000000	SPU	SELECT PERIPHERAL UNIT	1	3.6
07020000	SR	SHIFT A RIGHT		3.2
07004000	SST	SET STATE	1	3.4
14000000	STA	STORE A-REGISTER	2	3.2
15000000	STE	STORE E-REGISTER	2	3.2
06000000	STST	STATUS TEST	1	3.6
16000000	STX	STORE INDEX	2	3.3
22000000	SUB	SUBTRACT	2	3.2
	SYM	PRODUCE SYMBOL TABLE		4.11
07002000	TCA	TWO'S COMPLEMENT A	1	3.2
06000400	TOF	TOP-OF-FORM	1	3.6
	TPASS	TWO PASS ASSEMBLY		4.15
06421400	WRIT	WRITE	1	3.6
06061500	WRITM	WRITE TAPE MARK	1	3.6

A.2 (OPCODES SORTED BY ASCENDING OCTAL OPCODE)

OPCODE	CODE	CODE DESCRIPTION	CYCLES
	ABS	ABSOLUTE PROGRAM LOCATOR	
	BSS	BLOCK STORAGE SIZE	
	DATA	DATA DEFINITION	
	END	PROGRAM TERMINATOR	
	EQU	EQUIVALENCE	
	INSEQ	START SEQUENCE CHECK	
	LIST	PRODUCE ASSEMBLY LISTING	
	NO SEQ	STOP SEQUENCE CHECK	
	NOLIST	NO ASSEMBLY LISTING	
	NOOBJ	NO OBJECT PROGRAM	
	OBJECT	PRODUCE OBJECT PROGRAM	
	ORG	ORIGINATION CONTROL	
	PAGE	PAGINATION CONTROL	
	SYM	PRODUCE SYMBOL TABLE	
	TPASS	TWO PASS ASSEMBLY	
00000000	BAH	BRANCH AND HALT	1
00000000	PROC	SUBROUTINE ENTRY POINT	
00000000	PZE	POSITIVE ZERO (ENTRY PT)	
01000000	BRU	BRANCH UNCONDITIONAL	1
02000000	BAT	BRANCH ON A-REGISTER TEST	1
02040000	BO	BRANCH IF ODD	1
02100000	BN	BRANCH IF NEGATIVE	1
02200000	BZ	BRANCH IF ZERO	1
02300000	BNZ	BRANCH IF NEGATIVE OR ZERO	1
02400000	BP	BRANCH IF POSITIVE	1
02500000	BNEZ	BRANCH IF NOT EQUAL TO ZERO	1
02600000	BPZ	BRANCH IF POSITIVE OR ZERO	1
03000000	BOI	BRANCH ON INDICATOR	1
03040000	BBC	BRANCH BIT COMPARE	1
03100000	BL	BRANCH IF LESS	1
03200000	BE	BRANCH IF EQUAL	1
03300000	BLE	BRANCH IF LESS OR EQUAL	1
03400000	BG	BRANCH IF GREATER	1
03500000	BNE	BRANCH NOT EQUAL	1
03600000	BGE	BRANCH IF GREATER OR EQUAL	1
04000000	BOS	BRANCH ON STATE	1
04440000	BOV	BRANCH ON OVERFLOW	1
05000000	LDX	LOAD INDEX	1
06000000	SPU	SELECT PERIPHERAL UNIT	1
06000000	STST	STATUS TEST	1
06000400	TOF	TOP-OF-FORM	1
06000500	REWIND	REWIND TAPE	1
06001000	PCOMP	PRIORITY COMPLETE	1
06001500	RSKIPF	SKIP RECORD FORWARD	1