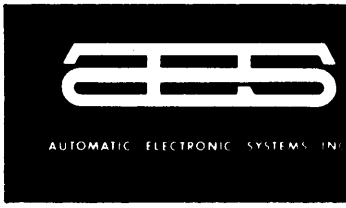


THE AES-80 MICROPROCESSOR

ASSEMBLER REFERENCE MANUAL

AUTOMATIC ELECTRONIC SYSTEMS INC., 5455 PARE ST., MONTREAL 309 CANADA, TEL. (514) 735-6581
AES DATA INC., P.O. BOX 143 ST ALBANS, VERMONT 05478, TEL. (802) 524-3660

8Ø

MICROPROCESSOR

ASSEMBLER

REFERENCE MANUAL

AUTOMATIC ELECTRONIC SYSTEMS INC.,
5455 Pare Street
Montreal 309

TEL.: (514) 735-6581

TABLE OF CONTENTS

CHAPTER 1

1.Ø INTRODUCTION

The A.E.S. Standard Assembler allows programmers to write their microprocessor programs in a symbolic language rather than in machine language, the translation being performed by the Assembler.

The Assembler input is usually a paper tape, punched in ASCII code on an off line teleprinter.

The output consists of:

- another paper tape called the object program punched in the appropriate binary code required by the microprocessor program loader.

- a source program listing which displays pertinent information about the program such as: Line count, instruction addresses, memory content, error messages, etc.

A program can be broken down into several smaller programs which can be assembled separately.

This Assembler manual will be more profitable to the reader already familiar with the general architecture of the microprocessor as described in the hardware manual.

CHAPTER 2

2.Ø DESCRIPTION

2.1 ASSEMBLY PASSES

The Assembler is a 2 or 3 pass system depending on the types of peripheral devices available: if a KSR-33 Teletype is the only output device, 3 passes are required; if the computer system includes a printer and an independent paper perforator, then only 2 passes are required since the listing and the binary tape can be produced simultaneously.

A pass is defined as one complete reading of the source tape (s).

During pass 1, the Assembler builds a label table with the label (or names) found in some of the source statements.

During pass 2/3 the Assembler decodes the instruction mnemonic, searches in the label table the actual address associated with a label used as an operand, prints the listing and/or punches the binary tape and prints the eventual error messages.

2.2 SYMBOLIC ADDRESSING

There are several advantages to writing a program in Assembler rather than in machine language: One obvious one is the use of mnemonics. (E.g: F=D-B) instead of a hard-to remember binary code such as L&(or 1446 Octal). Another one is the inherent redundancy of an Assembler statement which allows error detection and yet another one is the automatic documentation of a program.

By far the most important advantage, however, is due to the use of symbolic addresses instead of explicitely defined absolute addresses. For descriptive purposes, assume a program is written in machine language and that a single instruction must be inserted (or deleted) into this program. Every instruction following the insertion (or deletion) point will be moved by one location, which means that all the

instructions referring to any moved instruction have to be modified if the original flow chart is to be respected. This task can be time consuming and is prone to numerous errors.

Consider the same program written in Assembly language. The source tape is edited instead of the binary tape. Once edited, this new source tape is processed by the assembler and all the memory reference instructions will be automatically updated when necessary.

The same procedure applies if a complete program is moved from one place in memory to another.

A symbolic address is defined by a label (or name) placed in the label field of a statement. To refer to this address the same label is used in the operand field of the referring instruction.

Labels appearing more than once in the label field of a program are considered in error. (Double Defined). Labels appearing in the operand field and never appearing in the label field are also considered in error (Undefined).

2.3 PROGRAM LOCATION COUNTER

For symbolic addressing and documentation purposes the Assembler maintains a program location counter.

This counter is initiated or reinitiated only by an ORIGIN Pseudo-Instruction (ORG) which defines the absolute address of the first instruction of a program or a section of program.

From there on the assembler will assign consecutive locations to every instruction it encounters. During the first pass the Assembler uses this counter to assign absolute addresses to every label it finds in the label field of an instruction and stores both, label and address, into the Label Table.

2.4 INSTRUCTIONS AND PSEUDO-INSTRUCTIONS

Instructions are defined as actual commands to the microprocessor to be executed at program run time. The Assembler always converts them into machine instructions punched on the object tape.

Pseudo-instructions are defined as commands to the Assembler itself. They are not punched on the object tape and they are executed by the Assembler at assembly time.

CHAPTER 3

FORMATS

3.1 STATEMENT FORMATS

There are three types of statements in the A.E.S. Microprocessor Assembler:

-Comment statements
-Instruction statements
-Pseudo-Instruction statements

A statement is always contained in an ASCII line, i.e., a string of no more than 72 ASCII characters terminated by a RETURN CARRIAGE and a LINE-FEED character.

A statement is divided into several fields in order to distinguish between the different types of symbols.

For descriptive purposes, the characters within a line are numbered from 1 to 72 from left to right and designated by Ch x and the space character will be marked as ^ whenever it is necessary to show its place.

3.2 LABELS

A label is a word of 1 to 5 characters whose first one must be alphabetic or one of the following characters: @ [ ] ← ↑

The remaining characters can be any character except a space, a line-feed, a+, a-, a return carriage or a rubout. Non-printing characters will be accepted although it is not recommended to use them.

NOTE:

Since valid operands can be either octal numbers, decimal numbers or symbolic (labels) and a decimal number is of the form DXXXX e.g., D1Ø or D38Ø, labels shall not be of the form D followed by a number.

Examples of valid labels:

```
START
A1Ø
END
[#?
Z
```

Examples of invalid labels:

```
D2
128
#SUB
READER
```

LABEL FIELD

The label field exists in the instruction and in the pseudo-instruction statements. It always begins with Ch. 1 and ends with Ch. 5.

Ch. 6 must always be a space and acts as a separator between the label field and the mnemonic field. If the label has less than 5 characters, the remaining positions must be filled up with spaces.

If no label is used, then the label field must be filled up with spaces.

Examples:

```
START^D=M
^^^^^^F=D
      ...
Z^^^^^^RET
       NOP
```

3.3 MNEMONICS

A mnemonic is a conventional word identifying one instruction or pseudo instruction.

The list of valid mnemonics is given in chapter 4.

Their length can be from 3 characters up to 12 characters but they must always begin in Ch. 7. No space is allowed within a mnemonic. The first space, return carriage, or line feed encountered by the Assembler while reading a mnemonic is interpreted as an "end of mnemonic" mark.

Examples:

```
^^^^^^D=F
^^^^^^JSR^SUBRO
LP1^^^F=D-B+1
```

3.4 OPERANDS

3.4.1 There are two types of instruction operands:

3.4.1.1 - Those used in Instruction Memory Reference Instruction (Jump, Jump subroutine, Jump if)
For these instructions the Operand field is separated by one (and only one) space from the Mnemonic field.

Examples:

```
^^^^^^JMP^2300
      JSR^SUBRO
      JIS,RTC^NEXT
```

3.4.1.2 - Those used in Data Memory Reference Instructions, Literal Register Instructions and Channel Reference Instructions. In this case the operand field follows immediately the last character of the Mnemonic (which is always an "=" character)

Examples:

```
^^^^^^L=D10
      L=RAMAD
      A=400
      CHL=TTY
```

3.4.2 There are also two types of pseudo-instruction operands:

3.4.2.1 The first one is used with the original definition and follows the same rules as the first-type of instruction operand (3.4.1.1)

Examples

```
^^^^^^ORG^100
^^^^^^ORG^LAST+1
```

3.4.3.2 The second type is used in the label definition statements following a RAM or an EXT pseudo-instruction. These two pseudos are used to define data memory labels RAM) or instruction memory labels external to the program (EXT).

In both cases the label definition operand begins at Ch. 7.

Examples:

```
^^^^^^RAM
WORD1^74
LABEL WORD1+1
      ---
^^^^^^EXT
SUB1^^2ØØØ
S2^^^^SUB1+D2Ø
      ---
      ORGXXX
```

3.4.3 COMMON RULES FOR ALL OPERANDS:

3.4.3.1 No space allowed within an operand.
Operands can be either numeric or symbolic or a combination of both:

3.4.3.1.1 Numeric operands:

- A single numeric term:
- Octal term: a number of no more than 4 octal digits preceded, optionally by a sign (+,-)
- Decimal term: a number of no more than 4 decimal digits preceded by the letter D itself preceded, optionally, by a sign (+,-)

Examples
```
377
+377
-1
D1Ø8
+D1Ø8
-D1Ø8
```

3.4.3.1.2 Symbolic operands:

A label which must be defined elsewhere in the program, i.e., it must appear once and only once in the label field of the program (ch. 1 to ch. 5).

3.4.3.1.3 Symbolic operands modified by a numeric displacement:

A defined label immediately followed by a numeric term. The resulting address is the address assigned to the label displaced by an amount equal to the numeric term.

Examples

```
^^^^^^A=BUFF+2
```

Let's assume that BUFF has been defined as data memory address 1ØØ, then the A - register will point to address 1Ø2 after execution of this instruction.

NOTE:

Since A=A+1 is a special instruction to increment the A - Register, A can be used as a label as long as it is not used with a displacement of + 1.

3.4.4 SPECIAL RULES FOR PSEUDO INSTRUCTION OPERANDS

Since instruction operands are evaluated during the second pass they can refer to labels defined anywhere in the program.

However pseudo instructions operands are always evaluated during the first pass and therefore, if they are symbolic, they must refer to a label defined BEFORE this pseudo is encountered.

Examples: Valid pseudo instruction operands

```
      ...
LASTW^NOP
      RAM
DATA1^1Ø4
DATA2^DATA1+1Ø
DATA3 DATA1+2Ø
      ORG^LASTW+1
      ...
```

Illegal pseudo-instruction operands

```
      ...
      RAM
DATA2^DATA1+1Ø
DATA3^DATA1+2Ø
DATA1^1Ø4
      ORG LASTW+1
      ...
LASTW^NOP
```

3.5 COMMENTS

3.5.1 Comment Statement

An entire line can be devoted to comments, i.e.,

information useful to programmers but ignored by the assembler. To do this an asterisk (*) must be the first character of the line.

Example:

```
*^^THIS IS A COMMENT LINE
```

2.5.2 COMMENT FIELD

The right hand part of an instruction or pseudo-instruction statement can be used for additional comments: in both cases the comment must be separated from the operand, or from the mnemonic if there is no operand, by at least one space.

Examples:

```
PRG1^^F=D^COMMENT
      JSR^SUBRO^COMMENT
      RET            COMMENT
```

3.5.3 COMMENT SIZES

Since the printed listing includes information not originally punched in the source tape the number of characters allowed for a comment is limited accordingly:

- A comment statement cannot exceed 67 characters.
- An instruction statement cannot exceed 54 characters.
- A pseudo-instruction statement cannot exceed 67 characters.

Otherwise the comment characters in excess won't appear on the listing.

CHAPTER 4

MACHINE INSTRUCTIONS

4.1 DATA MEMORY REFERENCE INSTRUCTIONS

A single instruction can set a 1Ø bit address in the 12 bit address register (A).

A=xxx

The two most significant bits (bit 1Ø, 11) of the A register are left unmodified by the A= instruction. Thus one can directly address any one location of a 1Ø24 word data memory page.

The quantity xxx can be either Octal, Decimal or Symbolic. In any case it cannot exceed 1777 (8).

Examples:
```
A=77
A=D1Ø8
A=FLAG+3
```

In order to reach the other 1Ø24 word pages and also to load the A register with a computed address, two other instructions are provided.

AL=D	The data bus is loaded into the 8 LEAST significant bits of the A-register
AH=D	Bits 8, 9, 1Ø, 11 of the A-register are loaded with bits Ø, 1, 2, 3 of the data bus.

Examples: 1) Retrieval of one element of an array

```
A=PTR     Pointer address
D=M       Pointer on data bus
AL=D      Pointer in A-Register
          At this point, retrieved data is on the
          data bus.
```

2) Same requirement but the pointer is now in page Ø and the array in page 2 (256 Wd pages).

```
A=PTR     Pointer address
D=M       Pointer on data bus
F=D       Set ALU for direct load
B=F       Pointer in B-register
L=2       Literal ØØ2
D=L       On data bus
```

AH=D	Load bit 11, 1Ø, 9, 8 of the A-Register with Ø2
D=B	Pointer on data bus
AL=D	Load the 8 LSB of A-register with the pointer.

The A-Register can be incremented

A=A+1

Thus permitting easy scanning of an array:

Example: Find the first even element of an array

```
^^^^^^A=AR(Ø)-1      first element address minus 1
      D=M            Address on data bus
LOOP^^A=A+1          element on data bus
^^^^^^JIS,DBØ LOOP   Test least significant bit
```

4.2 DATA BUS INSTRUCTIONS

Four mutually exclusive latching instructions are available which enable programmers to choose the source of the data present on the bus.

D=L	Literal Register (L) on data bus
D=M	Data Memory location pointed by A on data bus
D=U	Universal Register (U) on data bus
D=B	ALU Register (B) on data bus

4.3 INSTRUCTION MEMORY REFERENCE INSTRUCTION

They are divided into 3 groups. All of them require two memory locations and are executed in two machine cycles.

4.3.1 UNCONDITIONAL JUMP

JMP xxx

The program flow is altered:
The operand defines the address of the next instruction to be executed.

4.3.2 JUMP TO AND RETURN FROM SUBROUTINE

JSR xxx

The operand defines the address of the first instruction of the subroutine.
The return address is automatically stored in the upper location of the push down stack.

Note: Subroutines can be nested up to 16 levels

RET xxx

The next instruction to be executed is the one following immediately the RET instruction. Then the program flow is altered: the last address stored in the push down-down stack defines the address of the next instruction to be executed. The push down stack is pushed-up.

Example: Increment a word in data memory

```
^^^^^^A=WORD        Address of word
^^^^^^JSR^INC       Increment it
      ...
*
*  INCREMENT SUBROUTINE
*
INC^^^D=M     Word on bus
      F=D+1   Set ALU for increment operation
      B=F     LOAD B-Register with incremented value
      D=B     Incremented word on bus
      RET     Initiate the return from subroutine
      M=D     Store incremented word in original address.

*
```

4.3.3 CONDITIONAL JUMP INSTRUCTIONS

They are of the form:

JIS,yyy^xxx or JIC,yyy^xxx
where yyy is a mnemonic defining the condition to be tested
and xxx the address of the next instruction to be executed if the condition is met.
JIS stands for Jump If Set (logical 1)
JIC stands for Jump If Clear (logical Ø)

JIS,BR7 xxx or JIC,BR7 xxx

Jump if B-Register bit-7 is Set/Clear.

Note: This bit can be considered as the sign of a 7 bit word in two's complement form.

JIS,CRY xxx or JIC,CRY xxx

Jump if ALU carry output flag is Set/Clear.
This flag is set whenever the result of an ALU arithmetic operation produces an overflow.

Example: Check if the quantity X present on the data bus is Ø.

```
F=D-1          Set ALU for X-1
JIS,CRY ZERO   If X was Ø, then X -1 = -1 and
               the carry flag is set.
```

JIS,D=B xxx or JIC,D=B xxx	Jump if data bus and B-Register are equal/ different. ALU must be in the F = D-B-1 mode

Example: Check if 2 quantities in memory are equal

```
A=WORD1        Address of first word
D=M            Word 1 on data bus
F=D            Set ALU for a direct load
B=F            Load word 1 into B-register
A=WORD2        Word 2 on data bus
F=D-B-1        Set ALU for compare
JIS,D=B EQUAL  If equal, the compare flag is
               set and the jump is executed
```

JIS,DBn xxx or JIC,DBn xxx	Jump if bit n of the data bus is set/clear where n is Ø to 7 (Ø for the least significant bit)

Example: Test an internal flag in position 4 of the data memory word FLAGS

```
A=FLAGS
D=M
JIS,DB4 FLSET  If bit 4 of the word FLAGS
               is set, then jump is executed
```

JIS,PDS xxx or JIC,PDS xxx	Jump if push down stack overflow flag is set/clear.

The remaining conditional jumps will be described in the Input/Output instruction section.

4.3.4 INSTRUCTION MEMORY PAGES

Since 11 bits are used to specify an instruction memory address, we can address directly any location in a 2Ø48 word page. To jump across a page boundary, a page instruction is provided.

PG=Ø PG=1	The next jump (any type) instruction will be made to the location defined by the jump operand but within the specified page (Ø or 1)

Example: Let's assume we are in page Ø
PG = 1
...
JSR 31Ø Jump subroutine to address 431Ø

4.4 ARITHMETIC - LOGIC UNIT INSTRUCTIONS

Generally an arithmetic or logic operation will be performed in two steps:

1) Set the ALU to the selected operation
2) Save the ALU output (called F) in the B-register.

Note: The ALU will remain set for an operation until it is changed by another ALU instruction.

Special characters used in the ALU instruction mnemonics

#	logical OR operation
.	logical AND operation
↑	logical EXCLUSIVE OR operation
'	logical COMPLEMENT of preceeding quantity: B' or (D#B)'
+	arithmetic ADDITION (2's complement)
-	arithmetic SUBTRACTION (2's complement)

Expressed with these symbols the De Morgan's theorem becomes:

(A#B)'=A'.B'
(A.B)'=A'#B'

Knowing these symbols, the ALU mnemonics are self explanatory.

4.4.1 LOADING INSTRUCTIONS

	The output F of the ALU is equal to the :
F=D	Data bus
F=D'	Data bus complement
F=B	B-register
F=B'	B-register complement
F=-1	Octal 377
F=Ø	Octal ØØØ

4.4.2 LOGIC INSTRUCTIONS

Logical OR

			Equivalent to
F=D#B	or	F=B#D	(D'.B')'
F=D#B'	or	F=B'#D	(D'.B)'
F=D'#B	or	F=B#D'	(D.B')'
F=D'#B'	or	F=B'#D'	(D.B)'

Logical AND

F=D.B	or	F=B.D	(D'#B')'
F=D.B'	or	F=B'.D	(D'#B)'
F=D'.B	or	F=B.D'	(D#B')'
F=D'.B'	or	F=B'.D'	(D#B)'

Exclusive OR

F=D↑B	or	F=B↑D
F=D↑B'	or	F=(D↑B)'

4.4.3 ARITHMETIC INSTRUCTIONS

Addition

F=D+D	or	F=2D
F=D+B	or	F=B+D
F=D+D+1	or	F=2D+1
F=D+B+1	or	F=B+D+1
F=D+1		
F=D+B'		
F=D+B'+1		

Subtraction

F=D-B	F=D-B'
F=D-B-1	F=D-B'-1
F=D-1	

4.4.4 COMBINED LOGIC AND ARITHMETIC INSTRUCTIONS

The logical operation is executed before the arithmetic one.

F=D#B+D	F=D#B'+D
F=D#B+1	F=D#B'+1
F=D#B+D+1	F=D#B'+D+1
F=D.B+D	F=D.B'+D
F=D.B+D+1	F=D.B'+D+1
F=D.B-1	F=D.B'-1
F=D#B+D.B'	F=D#B'+D.B
F=D#B+D.B'+1	F=D#B'+D.B+1

4.4.5 SHIFT ROTATE INSTRUCTIONS

F=BSL	F = B-register shifted left by one bit. LSB of F is set to Ø.
F=BRL	F = B-register rotated left by one bit. Thus the MSB of B becomes the LSB of F. Both instructions require that the B-register output is enabled onto the data bus (D=B).
B=BRR	B is ready for a one bit right rotation independently of the ALU output F.

4.4.6 B-REGISTER INSTRUCTIONS

B=Ø	Clear B-register
B=F	Load B-register with ALU output
B=FH	Load B-register with ALU output 4 most significant bits.
B=FL	Load B-register with ALU output 4 least significant bits.

Note: For B=FH and B=FL the remaining bits of the B-register are left unchanged.

Example: Convert one ASCII digit to binary

A=DIGIT	Digit address
D=M	Digit on data bus
F=D	Set ALU for direct load
B=Ø	Clear the B-register
B=FL	Load the 4 LSB in the B-register

Or, if the digit is already present in the B-register

F=Ø	
B=FH	Load Ø's into the 4 MSB of the B-register.

Note: If the ALU was set in the B=BRR mode then a B=F (or B=FL or B=FH) instruction will actually rotate the B-register one bit right.

4.5 LITERAL AND UNIVERSAL REGISTER INSTRUCTIONS

L=xxxxx	Set the literal register to the value of the operand. Since the L-Register has 8 bits, the operand, once evaluated by the Assembler, should not be greater than 377 (8). If it is, the 8 LSB will be used as operand. No diagnostic will be given.

Examples: L=-1, L=D255, L=377, L=1777, L=LABEL — All result in the loading of Octal 377

U=Ø	Clear the universal register.
U=U#D	Load the logical OR of the data bus and the present U-register content.

4.6 INPUT/OUTPUT INSTRUCTIONS

4.6.1 SERIAL I/O

CONTROL LINES:

CLK=Ø	CLK=1	Clock line low or high
LD=Ø	LD=1	Load line low or high
R/W=R	R/W=W	R/W Line low (read) or high (write)

For serial I/O all 8-bit data words pass through the Universal Register.

CHANNEL SELECTION

CHL=xx	xx = any operand (octal, decimal or symbolic) Its value must not exceed 31.

REGISTER SELECTION (One at a time)

RG=x	x = one octal digit (Ø to 7)
RG=B	In this case the register is selected by the 3 least significant bits of the B-register.

TIMING

SIO	Stand for Start Input-Output automatic transfer between the U-Register and the addressed I/O device register. This is a strobe function.
JIS,IOR xxx or JIC,IOR xxx	Test the I/O Ready flag. IOR flag is normally set. The SIO instruction clears it. When the transfer is completed the IOR flag is set again.

Examples: Input one 8 bit word from channel 1Ø, register Ø

```
        ...
        CHL=1Ø          Select channel
        RG=Ø            Select register
        R/W=W
        LD=1
        CLK=1           Load data into I/O register
        CLK=Ø
        LD=Ø
        R/W=R           Initiate serial transfer between I/O
        SIO             Register and U-Register
WAIT^^JIC,IOR WAIT        Finished? No loop
        D=U             Yes, data on bus
        ...
```

Output 8 8-bit words to channel Ø, registers Ø to 7 from 8 consecutive locations in data memory

```
        CHL=Ø           Select channel
        A=AR(Ø)         Address of first word
        B=Ø             Clear B-register
        F=D+1           Set ALU for increment
LOOP^^  D=M             Data on data bus
        U=Ø
        U=U#D           Data in U-Register
        RG=B            Select I/O register
        R/W=W           Initiate
        SIO             Serial transfer
WFLAG^JIC,IOR WFLAG       Wait if I/O Ready flag not set
        R/W=R
        A=A+1           Increment array pointer
        D=B
```

```
B=F             Increment B-register
JIC,DB3 LOOP      Done it 8 times ?  No, loop
...             Yes, exit
```

4.6.2 PARALLEL I/O

No special instruction is devoted to parallel I/O since the I/O registers are considered as data memory locations and, therefore, are operated by the M=D or D=M instructions.

Example: For descriptive purposes, assume the microprocessor is used with the Bose-Chaudhury Error code generator option. This option uses one input register (new data) and one output register (new Bose-Chaudhuri Error code) and their addresses are respectively $6ØØ1_{(8)}$ and $6ØØØ_{(8)}$. New data is supposed to be initially in the U-Register and we want to get the result (new Error Code) in the B-register.

```
L=14
A=1
D=L
AH=D    Set A-register to 6ØØ1
D=U     New data on data bus
M=D     Compute new Error Code
A=Ø     Set A-Register to 6ØØØ
D=M     Error code on bus
F=D
B=F     New Error code in B-register
```

4.6.3 INTERRUPT INSTRUCTIONS

JIS,SIN xxx or JIC,SIN xxx	Jump If the Serial Interrupt flag is Set/Clear.
JIS,PIN xxx or JIC,PIN xxx	Jump If the Parallel Interrupt flag is Set/Clear.
JIS,INT xxx or JIC,INT xxx	Jump If the Master Interrupt flag is Set/Clear.

Note: This flag is set whenever any one of the decision flags, strapped into the interrupt structure, is set. These decision flags are usually chosen from the following.

- Push Down Stack flag
- Console Alarm flag
- Power Fail flag
- Relinquish Bus flag
- Parallel I/O Interrupt flag
- Serial I/O Interrupt flag
- Real Time Clock flag

IAK=1 IAK=Ø	Interrupt Acknowledge. Used to clear the device status flag if it is an interrupting device. If the device was the only interrupting device IAK will also clear the serial or parallel interrupt flag. To clear a serial I/O interrupt and status flag:

```
CHL=x
RG=y
R/W=W
IAK=1
IAK=Ø
```

To clear a parallel I/O interrupt and status flag

```
A=z
D=M
IAK=1
IAK=Ø
```

In both cases, if the status flag is not connected to the interrupt structure, only the device status flag will be cleared.

JIS,SFL xxx or JIC,SFL xxx	Jump If Serial I/O Status Flag Is Set/Clear.
JIS,PFL xxx or JIC,PPL xxx	Jump If Parallel I/O Status Flag is Set/Clear.

Prior to testing a status flag, the proper device must be addressed: channel and Register selection for serial I/O Data memory Address Register (A) for parallel I/O.

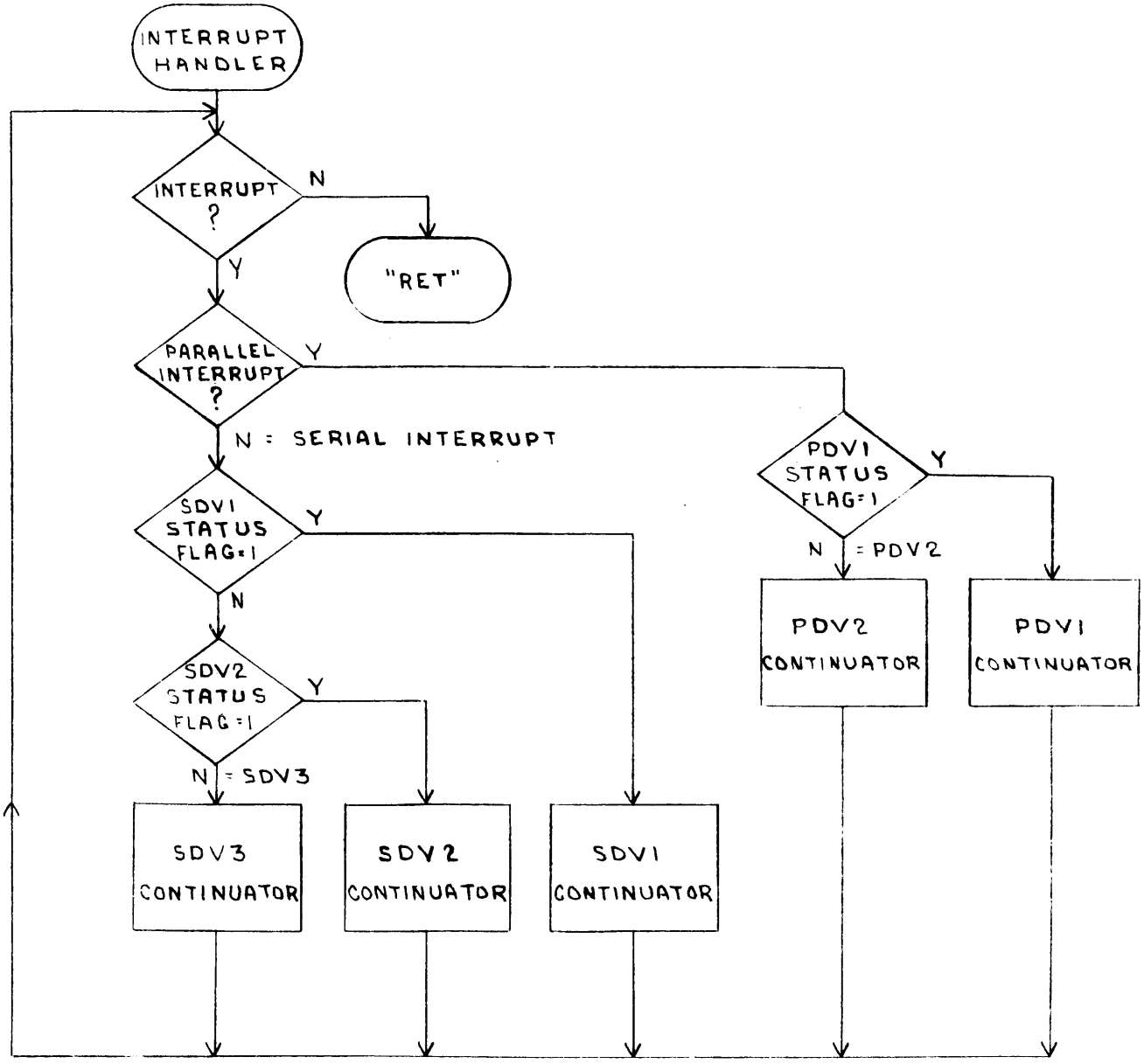
Example: For descriptive purposes, assume a configuration of 3 devices connected to the serial I/O and 2 devices connected to the parallel I/O. All have status flags and are connected to their respective interrupt systems. They are called SDV1, SDV2, SDV3 and PDV1, PDV2 respectively. SDV1, SDV3 and PDV1 are for input, SDV2 and PDV2 are for output. The software system will consist of a background program called "PROCESS" and a foreground program called "ACQUISITION". PROCESS will initiate I/O operations and ACQUISITION will service these I/O operations once they have been initiated. Every device handler is composed of two parts:

1) the Initiator, which will be part of PROCESS
2) the Continuator, which will be part of ACQUISITION

For example simplicity we will assume that ACQUISITION is non-interruptable and thus a reentrant Interrupt Handler is not needed. Communication between PROCESS and ACQUISITION will be accomplished through data buffers and software flags: one BUSY flag for every device handler. This flag will be automatically set by the Initiator and cleared

by the continuator when the I/O operation is completed.
Thus PROCESS is given the ability to know the state of the I/O operations.
Devices will be serviced according to the following priority:

1 PDV1 input
2 PDV2 output
3 SDV1 input
4 SDV2 input
5 SDV3 output



To allow an interrupt to take place all subroutines used by PROCESS return to the calling program by making a direct jump to the interrupt handler entry point subroutine example:

```
SUBRO^...
      ...
      JMP^INTH
```

Thus the maximum time an interrupt request is kept waiting will be determined by the longest instruction sequence without a jump to the interrupt handler. Our experience shows that the average sequence will be around 1Ø instructions and, without any special attention it has never exceeded 4Ø instructions in any of our past program applications. To limit the waiting time to a given value, calls to a dummy subroutine may be inserted in all sequences exceeding the limit.

The Interrupt handler for the previous example would be:

```
INTH^^JIS,INT^L1        Test master Interrupt
      RET               Not set, return
      NOP
L1^^^^JIS,PIN^L2        Test parallel interrupt
      R/W=W             Not set:  serial I/O is interrupting
      CHL=
      RG=               Address SDV1
      JIS,SFL L3        Test SDV1 status flag
      CHL=              Not set
      RG=               Address SDV2
      JIS,SFL L4        Test SDV2 status flag
      CHL=              Not set
      RG=               Address SDV3
      IAK=1             Clear SDV3 status flag
      IAK=Ø
      JMP CONS3         Jump to SDV3 continuator
*
L2^^^^A=                Address PDV1
      D=M
      JIS,PFL L5        Test PDV1 status flag
      A=                Not set, Address PDV2
      IAK=1             Clear PDV2 status flag
      IAK=Ø
      JMP CONP2         Jump to PDV2 continuator
*
L3^^^^IAK=1             Clear SDV1 status flag
      IAK=Ø
      JMP CONS1         Jump to SDV1 continuator
*
L4^^^^IAK=1             Clear SDV2 status flag
      IAK=Ø
      JMP CONS2         Jump to SDV2 continuator
*
L5^^^^IAK=1             Clear PDV1 status flag
      IAK=Ø
      JMP CONP1         Jump to PDV1 continuator
*
```

4.6.4 OTHER I/O INSTRUCTIONS

JIS,IOD xxx or JIC,IOD xxx	Jump If serial I/O bus data line is Set/Clear. The following instructions are independent of the standard I/O structure.
JIS,ALM xxx or JIC,ALM xxx	Jump If external ALARM flag is Set/Clear
JIS,PWR xxx or JIC,PWR xxx	Jump If POWER fail interrupt flag is Set/Clear
JIS,RTC xxx or JIC,RTC xxx	Jump If Real Time Clock flag is Set/Clear
JIS,RBF xxx or JIC,RBF xxx	Jump If Relinquish Bus Flag is Set/Clear
RBC or EBC	Disable or Enable all serial and parallel I/O line drivers and receivers (Relinquish or Enable Bus Control)

4.6.5 HALT INSTRUCTIONS

HLT=xx	xx=octal number between Ø and 17. The halt instruction is available only with the maintenance and control chassis connected to the micro-processor. Otherwise it is treated as a NOP instruction.

4.6.6 MASTER RESET

RST	Strobe the micro-processor into the PORC condition.

CHAPTER 5

PSEUDO-INSTRUCTIONS

5.1 PROGRAM ORIGIN

During the first pass if the assembler encounters no origin statement before the first non-comment statement, it will request an origin on the teletype. The answer to be keyed-in must be an octal or a decimal number in the range Ø-7777(8).

This feature allows programmers to decide the program location at Assembly time if they choose to do so.

An origin statement has the form:

ORG xxxxx

where xxxxx can be either octal, decimal or symbolic.

In any case, when this operand is evaluated it must yield a result within the range Ø-7777(8)

Any number of ORG statements can be inserted in a program.

Examples.

```
        ORG 1ØØ
        ...             Program part I
        ...
        RET
LSTWD   NOP
        ORG D1Ø
        ...             Program part II
        ...
        ORG LSTWD+1
        ...             Program Part III consecutive
                        to Part I.
```

5.2 DATA MEMORY ADDRESSES

RAM

Since the data memory is usually built with read/write Random Access Memory a "RAM" pseudo-instruction has been created:

```
^^^^^^RAM
LABEL^xxxxx
.............
```

The Assembler will consider all non-comment statements following a "RAM" as pseudo-instructions defining the address associated with a label.

The operand xxxxx of a Label definition pseudo-instruction can be either octal, decimal or symbolic.

The Assembler will stay in the mode where it treats non-comment statements as Label definitions until it encounters an ORG, an EXT or an END statement.

Example

```
LSTWD NOP
*
      RAM
LAB1^^144
LAB2^^LAB1-1Ø
FLAGS^D3
*
*
        ORG LSTWD+1
```

5.3 EXTERNAL LABELS — EXT

When a label is not declared anywhere in the program as an operand (Example: the entry point of a subroutine not included in the program), an "EXT" pseudo-instruction is used to define it.

```
^^^^^^EXT
LABEL^xxxxx
. . . . . .
```

The assembler will consider non-comment statement following an "EXT" as pseudo-instructions defining the address associated with a label.

The operand xxxxx of a Label definition pseudo-instruction can be either octal, decimal or symbolic.

The Assembler will stay in the mode where it treats non-comment statements as Label definitions until it encounters an ORG, a RAM, or an

END statement.

Example

```
...
JSR SUBRO
CHL=TTY
*
EXT
*
SUBRO^277
SUB2^^SUBRO-1
TTY^^^3Ø
      RAM
      ...
```

5.4 END OF PROGRAM

END

A source program can be made of several pieces of tape. Each of them with a leader and a trailer (at least 5 inches of null characters).

As long as the Assembler does not encounter an "END" statement it will assume that there are additional source tapes to come.

As soon as it finds the "END" statement the Assembler stops reading and considers the pass as finished. Anything after an END statement is ignored.

CHAPTER 6

ASSEMBLER INPUT/OUTPUT

6.1 SOURCE PROGRAM

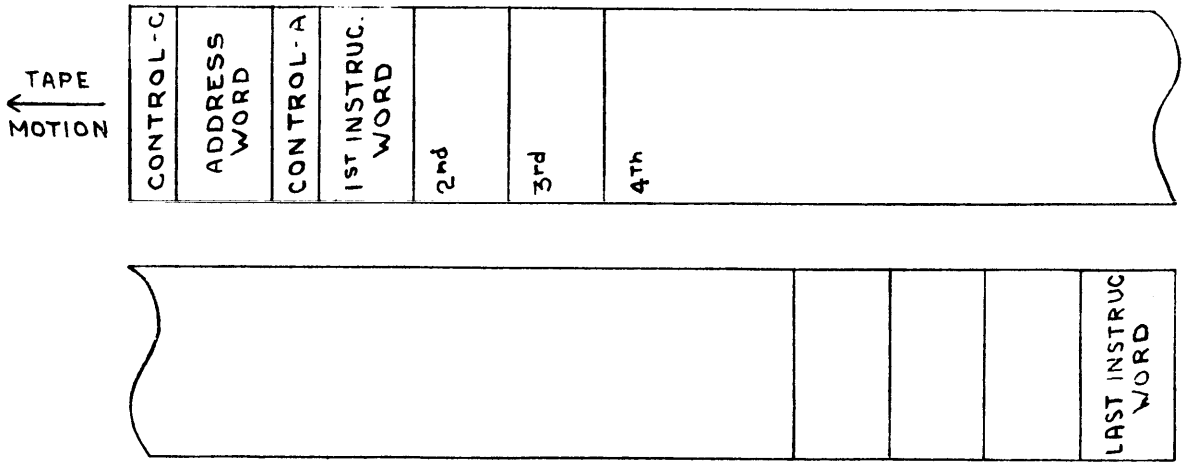
A source program can contain up to 9999 statements.

When a source program is divided into several pieces of tape, the last (non-null) character of any tape must be a return-carriage or a Line feed.

On reader input, a rub-out character is ignored. On keyboard input (Origin request), The Assembler ignores the line which contains a rub-out.

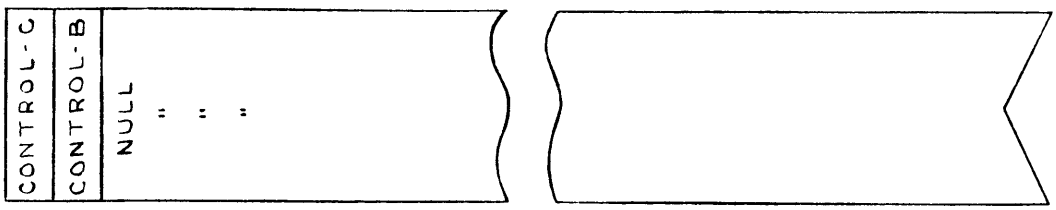
6.2 BINARY TAPE FORMAT

The binary tape is built with consecutive blocks and, of course, a leader and a trailer (null characters). Each block has the following format:



NOTE: There is a block for every ORG in the source program

The trailer has the following format



Every character is provided with even parity (i.e: the sum of all 8 bits must be even).

The address word and the instruction words have the following format:

FIRST CHARACTER

8	7	6	5	4	3	2	1

WORD BIT 11 1Ø 9 8 7 6

SECOND CHARACTER

8	7	6	5	4	3	2	1

5 4 3 2 1 Ø

Character BIT 7 is always the complement of BIT 6 in order to have a printing character rather than a non-printing one.

See Appendix A for the binary tape character set.

6.3 LISTING FORMAT

Pages are numbered from 1 to 99. The line number is reset to 1 at the beginning of a source tape.

LINE NUMBER	INSTRUCTION MEMORY ADDRESS	INSTRUCTION (ASCII)	INSTRUCTION (OCTAL)	ORIGINAL SOURCE INSTRUCTION STATEMENTS
1234	56789	10	11 12 13 14 15 16 17	18 19 72
ØØØ1	Ø1ØØ	F A	Ø 6 Ø 1	D=M MEMORY oa DATA BUS

LINE NUMBER	ORIGINAL SOURCE PSEUDO-INSTRUCTION OR COMMENT STATEMENTS
ØØ1Ø	ORG 4Ø
ØØ11	* PROGRAM

At the end of the listing the total number of locations used by the program is printed. (in decimal).

6.4 ERROR MESSAGES

The Assembler will print error messages when it finds syntax errors or it cannot recognize a label or a mnemonic or it cannot evaluate an operand:

These messages are:

ILLEGAL (format error, RAM or EXT block error, literal, channel, or halt, overflow, mnemonic error.

UNDEFINED (an operand cannot be evaluated).

ADDRESS ERROR (address above 3777 for instruction memory or above 1777 for data memory)

LABEL ERROR (label format).

DOUBLE DEFINED (label)

SYMBOL TABLE OVERFLOW

The faulty statement follows the error message.
The total error count for a program is given at the end of every pass (in decimal).

6.5 OPERATING INSTRUCTIONS

6.5.1

Data General Nova Computer: PAPER TAPE SYSTEM Minimum Configuration: 1 NOVA computer with 4K of Memory
1 Teletype ASR-33

LOADING 1-Turn computer ON, and, if available, set the fast tape punch and/or line printer to on.

2-Turn teletype ON-LINE.

3-Set switch Register to 7777 for a 4K Nova computer
or 17777 for a 8K Nova computer
or 27777 for a 12K Nova computer
or 37777 for a 16K Nova computer

4-Put the configured "CROSS-ASSEMBLER" binary tape into the tape reader (teletype reader or, if available, fast tape reader).

5-Press "Reset"

6-Press "Start", the Computer should load the cross Assembler

PASS 1 7-Set Switch Register to 4ØØ

8-Place the source tape into the tape reader

9-Press "Reset"

1Ø-Press "Start"-Computer should read the first tape.

11-If more than 1 tape is to be input (for 1 given program), repeat step 8 and press "Continue" for every tape.

After the last tape, which must contain an "END" statement, the computer will print "N" ERRORS.

PASS 2 12-choose your options:

sw Ø	sw 1	
1	Ø	Listing
Ø	1	Binary tape
1	1	Listing & Binary tape simultaneously providing they come from 2 independent devices.
Ø	Ø	No listing, no binary type: only the error messages.

13-Place the source tape into the tape reader.

14-Press "Continue"

15-If more than 1 tape, repeat steps 13 and 14

16-After the last tape, the computer is ready to execute another Pass 2 (Steps 12 to 16)

NOTE: During Pass 2, if the Listing option has been chosen, one can bypass unwanted sections of the listing by setting SW 14 to 1, thus only the line number and the page number are printed. By setting SW 15 to 1, only the page number is printed. As soon as these switches are reset to Ø the printing reverts to complete listing.

6.6 OBJECT PROGRAM LOADING

Set "PARITY" switch to parity or NO parity check.

6.6.1 TELETYPE LOADING

1. Set teletype/reader switch to "TTY".
2. Place binary tape into teletype tape reader.
3. Press "LOAD PROGRAM".

6.6.2 READER LOADING

1. Set reader switch to "RDR".
2. Place binary tape into tape reader.
3. Press "LOAD PROGRAM".

Note: It will automatically load until one of the following conditions occurs:

-It reads a control-B character (Valid end of loading).

-There is a parity error (if parity switch was set).

-There is a Teletype transmission error.

-The Operator pressed "HALT".