AAEC/TM566

AUSTRALIAN ATOMIC ENERGY COMMISSION

RESEARCH ESTABLISHMENT
LUCAS HEIGHTS

NOVASM AND NOVASIM - AN ASSEMBLER AND A SIMULATOR FOR THE NOVA AND SUPERNOVA COMPUTERS WRITTEN TO RUN ON AN IBM 360 COMPUTER

by

P.L. SANGER

September 1970

ISBN 0 642 99381 5

AUSTRALIAN ATOMIC ENERGY COMMISSION

RESEARCH ESTABLISHMENT

LUCAS HEIGHTS

NOVASM AND NOVASIM

AN ASSEMBLER AND A SIMULATOR FOR THE NOVA AND SUPERNOVA COMPUTERS

WRITTEN TO RUN ON AN IBM 360 COMPUTER

by

P. L. SANGER

ABSTRACT

An Assembler, NOVASM, and a Simulator, NOVASIM, for the NOVA and SUPERNOVA computers are described. Both programs are written in IBM 360 Assembler language and can be run on any size IBM 360 computer operating under OS/360. Their combined use provides a fast and convenient way of preparing and debugging machine language programs for the NOVA and SUPERNOVA computers.

National Library of Australia card number and ISBN 0 642 99381 5

CONTENTS

continued...

CONTENTS (continued)

1. INTRODUCTION

NOVASM is an assembler program for the NOVA and SUPERNOVA computers, written in IBM 360 Assembler language* and can be run on any size IBM 360 computer that operates under the IBM 360 operating system. The IBM standard instruction set plus the decimal feature instructions are used in the Assembler, with the exception of one assembler instruction that uses the floating-point feature instructions (see Section 5.1.5).

The output from the Assembler includes a listing of the source statements and the corresponding machine language code (object code), and additional information useful to the programmer in analyzing his program, such as error indications and cross-reference listings. The object code can be punched out onto cards as an object deck.

It is possible to set up a pseudo-core area (an image of the core storage of the NOVA or SUPERNOVA computer containing the assembled object code) and to transfer control to other programs, such as the Simulator NOVASIM, at the end of assembly.

The NOVASIM program uses the information set up in pseudo-core to simulate and display the action of a NOVA or SUPERNOVA machine language program. All memory reference instructions and arithmetic and logical instructions are simulated, but in the input/output class of instructions only the optional hardware multiply-divide instructions are simulated.

By using the trace feature in the Simulator, the result of 'executing' selected machine instructions is printed out. Additional information such as the contents of Carry and the four accumulators after the instruction is executed, and the time in microseconds relative to the start of simulation, is also printed. The calculation of the elapsed time takes into account the fact that the computer could be a NOVA or a SUPERNOVA and that certain parts of core storage may contain read-only memory.

NOVASIM is written in IBM 360 Assembler language and can be run on any IBM 360 computer operating under OS/360 with the standard instruction set and the decimal feature instructions. It can also be executed under the control of NOVASM.

NOVASM and NOVASIM provide a very fast and convenient way of preparing machine language programs for the NOVA or SUPERNOVA computers. Programs for these computers can be assembled and essential parts completely debugged by the Simulator before they are installed. Even if the NOVA or the SUPERNOVA

---

* IBM 360 Operating System - Assembler Language, Form C28-6514-6, June 1969.

is available for program preparation, NOVASM and NOVASIM provide a facility that is many times faster than that provided by the Assembler and Editor on the NOVA or SUPERNOVA.

2. THE ASSEMBLER

NOVASM accepts a source program for the NOVA or SUPERNOVA computer and assembles the corresponding object code. The source statements contain machine instruction mnemonics and assembler instruction mnemonics, the various machine instruction mnemonics being identical to those listed in Appendix D of the Systems Reference Manual for the NOVA and SUPERNOVA*

Machine instructions produce object code that may later be executed by the NOVA or SUPERNOVA computer.

The assembler instructions are used to specify auxiliary functions to be performed by the Assembler. These are instructions to the Assembler itself and, with a few exceptions, do not result in the generation of any object code by the Assembler.

The various machine and assembler instructions are described in the sections to follow.

2.1 Notation

The following notation is used in this report to describe the format of Assembler source statements.

If a number of parameters are enclosed in braces, then one of these parameters must be chosen. For example

$$\left\{\begin{matrix}\text{ADD}\\\text{SUB}\\\text{NEG}\end{matrix}\right\}$$

indicates that one of the parameters 'ADD', 'SUB' or 'NEG' must be used. Furthermore, since these parameters are specified using upper case letters this is the only form of each parameter that can be used. On the other hand

$$\left\{\begin{matrix}\text{label}\\\text{number}\end{matrix}\right\}$$

indicates that a label or number must be used in the appropriate field, but any valid label or number may be substituted for the parameters which have been specified using lower case letters.

---

* System Reference Manual for the NOVA and SUPERNOVA, DG NM-2, November 1969.

Square brackets indicate an optional field, where one or none of the enclosed parameters may be chosen. For example [label[:]] indicates that a valid label may be used in this field and it may optionally be followed by a colon.

2.2 Card Fields

Each source statement contains four basic fields - a label, an operator, operand(s) and a comment field. The first character in a label must be in column one and, apart from this restriction, the fields may appear anywhere on the source card provided that they are separated by at least one blank.

Using the notation described in Section 2.1 the general form of the source statement is:

[label[:]] blank(s) {operator} blank(s) [operand(s)] blank(s) [[;]comments]

Note that the comment field may optionally be preceded by a semicolon.

If an asterisk (*) appears in column one, then the rest of the card is treated as a comment field.

2.3 Labels

The first character in a label must be alphabetic and must appear in column one. This character may be followed by alphabetic characters or numbers provided that the total length of the label does not exceed eight characters. If the label is longer than eight characters the first eight characters are taken as the definition of the label. The label may optionally be terminated by a colon.

2.4 Operators

Operators determine the action the Assembler takes when it interprets the rest of the source statement. The allowable operations fall into two categories: machine instruction mnemonics and assembler instruction mnemonics. These operators are described in later sections.

2.5 Operands

An operand, if required, may be the value of an accumulator, a memory address or one of the conditional mnemonics that may be appended to an arithmetic and logical class instruction. If more than one operand is required they must be separated by commas, unless stated differently below, and must contain no intervening blanks.

2.6 Symbol Table Assignments

A program counter (PC) is maintained by the Assembler and incremented after every word of object code is assembled. The current value of PC is the memory address of the object code currently being assembled. The label field on a

machine instruction or some of the assembler instructions can be used to give a symbolic name to this address. The value of a label may also be predefined by using the EQU operator (see Section 5.4) or the ORG operator (see Section 5.2).

Each label and its attributes (including its value) is entered into a symbol table. Each symbol table entry is 18 bytes long and is placed in the table in a position based on the binary value of the first nine bytes of the entry. The symbol table entries are described in detail in Appendix 1.

The value of a label is obtained by searching for the corresponding symbol table entry using a binary search method. This binary search is based on the first nine bytes of the symbol table entries.

2.7 Expressions

Many machine and assembler instructions require numeric operand fields. These operand fields may be defined by expressions. The terms that can be used in a general expression are labels, decimal integers and the current value of PC (represented by an asterisk or a full stop). These terms may be combined using + or - signs, but must contain no intervening blanks.

An undefined label is given the value zero in expression evaluation, and if a decimal integer contains more than eight digits, the first eight digits are used. Expression evaluation is abnormally terminated if a syntax error is encountered.

The final value of an expression is checked for validity depending on its context.

3. INSTRUCTION ASSEMBLY

All of the possible machine instruction mnemonics and assembler instruction mnemonics are stored in an operation code table in the Assembler. Each table entry is ten bytes long. The first six bytes of each entry define the instruction mnemonic (left justified and padded with blanks where necessary). The next two bytes contain the basic object code for that instruction mnemonic (or zero for assembler instruction mnemonics). The ninth byte contains an index (used as a branch address by the Assembler) that defines the type of the instruction mnemonic. The last byte of each table entry is unused. Entries in the operation code table are stored in an order based on the binary value of the first six bytes of each entry.

The object code for a machine instruction is assembled in the following way. The entry in the operation code table corresponding to the machine instruction mnemonic is located by using a binary search method and the type of instruction is determined. Once the type is known, the operands are examined, evaluated and encoded into the proper bit positions of the corresponding basic object code.

The assembler instructions are processed in a similar manner except that the value of the operands are used to specify some auxiliary function that is required for the assembly of subsequent source statements, such as the resetting of PC.

4. MACHINE INSTRUCTIONS

NOVA and SUPERNOVA machine instructions fall into three classes: arithmetic and logical (ANL), input/output (I/O), and memory reference (MR). The format of the source statement for each class of machine instruction is described in the following sections. The label field is optional on all of these instructions and, if present, has the current value of PC.

4.1 Arithmetic and Logical Instructions

The format of a source statement for an ANL instruction is as follows:

```
[label[:]  blank(s) { ADD }  [ O ] [ L ]  [#] blank(s) {acs,acd}
                    { ADC }  [ Z ] [ R ]
                    { AND }  [ C ] [ S ]
                    { COM }
                    { INC }
                    { MOV }
                    { NEG }
                    { SUB }

                    [ [ , { SKP } ] ]
                          { SZC }
                          { SNC }
                          { SZR }
                          { SNR }
                          { SEZ }
                          { SBN }

                                    blanks(s) [[;]comments]
```

The expression 'acs' defines the source accumulator, while the expression 'acd' defines the destination accumulator.

4.2 Input/Output Instructions

There are four possible source statement formats for I/O instructions and these are described below.

4.2.1 I/O instructions with no operands

```
[label[:]] blank(s) { HALT }  blank(s) [[;]comments]
                    { INTEN }
                    { INTDS }
                    { IORST }
                    { MUL }
                    { DIV }
```

Note that the instruction mnemonics MUL and DIV apply only to the SUPERNOVA computer.

4.2.2 I/O instructions with an accumulator

```
[label[:]] blank(s) { READS } blank(s) {ac} blank(s) [[;]comments]
                    { INTA  }
                    { MSKO  }
```

The expression 'ac' defines the value of the accumulator that is to contain the contents of the console data switches, the device code of the first device on the bus that is requesting an interrupt, or the accumulator that contains the mask to be used to set up the Interrupt Disable flags in the devices.

4.2.3 I/O instructions with a device code

```
                  { NIO [S]                }
[label[:]] blank(s) { [C]                  } blank(s) { CPU    } blank(s) [[;]comments]
                  { [P]                    }          { device }
                  { SKP {B} {N}            }
                  {     {D} {Z}            }
```

The expression 'device' defines the I/O device involved. The mnemonic CPU automatically generates the device code 63.

4.2.4 I/O instructions with an accumulator and device code

```
                  { D {I} { A } [ S ] }                { ac, { CPU    } }
[label[:]] blank(s) {   {O} { B } [ C ] } blank(s)     {     { device } }
                  {         { C } [ P ] }

blank(s) [[;]comments]
```

The expression 'ac' defines the accumulator source of output data or destination for input data. The device code field is as described in Section 4.2.3.

4.3 Memory Reference Instructions

There are two possible source statement formats for MR instructions and these are described below.

4.3.1 MR instructions without an accumulator

```
                  { JMP }                  { [@] disp [ {(base)} ] }
[label[:]] blank(s) { ISZ } blank(s) [I] blank(s) { [@] disp [ {,base} ] }
                  { DSZ }                  {                            }
                  { JSR }

blank(s) [[;]comments]
```

Either of the mnemonics, 'I' or '@', can be used to indicate indirect addressing.

If the expression 'disp' appears on its own, then the Assembler determines if it is an address in page zero ($0 \leq disp \leq 255$). If the address is not in page zero the Assembler checks whether it is in the range of PC, that is, $(PC - 128) \leq disp \leq (PC + 127)$. Finally if the address is neither in page zero or within the range of PC the Assembler checks whether it is in the range of accumulators 2 or 3, that is, $(base - 128) \leq disp \leq (base + 127)$, provided that these had previously been set up as base registers by the USING instruction (see Section 5.6).

If the second operand 'base' appears its value must be 2 or 3 and the expression 'disp' is then taken to be a displacement relative to the contents of accumulator 2 or 3, that is $-128 \leq disp \leq +127$.

4.3.2 MR instructions with an accumulator

[label[:]] blank(s) {LDA / STA} blank(s) [I] blank(s) {ac, [@] disp [{(base) / ,base}]} blank(s) [[;]comments]

The expression 'ac' defines the source or destination accumulator for memory data. The address part of the statement is determined in the same manner as described in Section 4.3.1.

5. ASSEMBLER INSTRUCTIONS

The assembler instructions are requests to the Assembler to perform certain operations during the assembly process. Some, such as DS and DC, cause storage areas to be set aside for constants and other data. Others, such as EQU and EJECT, are effective only at assembly time; they generate no object code and have no effect on PC.

Labels can only appear on the EQU, DC, ORG, DS and REPEAT assembler instructions. A label must appear on the EQU statement and is given a value as described in Section 5.4. A label may optionally be used on the other four assembler instructions just mentioned, and, if present, is assigned the current value of PC.

Any labels that are used in expressions that define the operand field of an ORG, DS, EQU or REPEAT assembler instruction must be predefined.

5.1 Data Assembly

The DC (Define Constant) assembler instruction can be used to initialise certain locations in memory to contain specific data before a program is executed.

The instruction takes the following general form:

[label[:]] blank(s) {DC} blank(s) {operand} blank(s) [[;]comments]

There are eight different forms of the operand field and these are described below.

5.1.1 Address constants

Address constants are set up using the following form of the operand field of the DC instructions:

[I] blank(s) {A([@]addr)}

The expression 'addr' must define a valid address for the NOVA or SUPERNOVA computer being used (see Section 6.2.1) and the corresponding 15-bit address is set up in one word of object code.

Either of the mnemonics I or @ can be used to indicate an indirect address, and cause bit zero of the address constant to be set to 1.

Examples (octal print)

PC	OBJ CODE		STATEMENT	
01000	000024	ADRT	DC	A(20)
01001	100005		DC I	A(5)
01002	100036	ADRP	DC	A(@30)
01003	100002		DC	A(@ADRP-ADRT)

5.1.2 Integer constants

Integer constants may be set up using the following form of the operand field in the DC instruction: F(integer).

The value of the expression 'integer' is set up as one word of object code.

Examples (octal print)

PC	OBJ CODE		STATEMENT	
01000	000024	A	DC	F(20)
01001	177747	B	DC	F(-25)
01002	000777		DC	F(A + B -*)

5.1.3 Octal constants

The following format of the operand field in the DC instruction can be used to set up octal constants: O(octal).

The term 'octal' may be replaced by octal digits and the resulting constant set up as one word of object code.

Examples (octal print)

PC	OBJ CODE		STATEMENT	
00700	000027	LAB1	DC	0(27)
00701	173772	LAB2	DC	0(173772)
00702	001145		DC	0(1145)

5.1.4 Hexadecimal constants

Hexadecimal constants may be set up using the following form of the operand field in the DC instruction: $\left\{ \begin{matrix} X \\ Z \end{matrix} \right\}$ (hexadecimal).

The term 'hexadecimal' can be replaced by hexadecimal numbers and the resulting constant set up as one or two words of object code if the mnemonic used is X or Z respectively.

Examples (hexadecimal print)

PC	OBJ CODE		STATEMENT	
012A	00A2	LAB	DC	X(A2)
012B	0112		DC	X(112)
012C	0000000A		DC	Z(A)
012E	000000A4		DC	Z(A4)

Examples (octal print)

PC	OBJ CODE		STATEMENT	
00452	000242	LAB	DC	X(A2)
00453	000422		DC	X(112)
00454	000000		DC	Z(A)
	000012			
00456	000000		DC	Z(A4)
	000244			

5.1.5 Floating point constants

Floating point constants can be set up using the DC Z(hexadecimal) instruction, but can be more conveniently defined using the following operand field in the DC instruction: E(number [E {exponent}]).

The term 'number' can be replaced by any signed decimal number or integer. The term 'exponent' may be used to define the decimal exponent of the floating point number. The floating point constant is set up as two words of object code and represents a decimal range of approximately 5.4×10^{-79} to 7.2×10^{75}.

The DC E(---) instruction is the only assembler instruction that uses the IBM 360 floating point feature instructions. If this feature is not available on the IBM 360 computer being used, then floating point constants must be set up using the DC Z(---) instruction.

Examples (hexadecimal print)

PC	OBJ CODE		STATEMENT	
012A	4116A090	LAB1	DC	E(1.4142)
012C	4116A090		DC	E(14142E-04)
012E	C2400000		DC	E(-64)
0130	41A00000	LAB2	DC	E(1E1)

5.1.6 Character strings

The ASCII codes (with even parity) for character strings may be set up as data by using the following operand field in the DC instruction:

$\left\{\begin{matrix}C\\T\end{matrix}\right\}$ 'character(s)'

The term 'character(s)' may be replaced by a series of characters and the ASCII code for each pair of characters is combined to form one word of object code. If an odd number of characters appear within apostrophes, the spare eight bits of the last word of object code are padded out with zeros.

Only the characters that are available on an IBM 029 card punch can be enclosed within apostrophes. However, some of the special characters on the IBM 029 card punch can be used to set up the ASCII code for the teletype characters assignment arrow (←), carriage return, line feed and vertical arrow (↑). If an apostrophe is required in the character string, it must be followed by another apostrophe (see examples). A list of the allowable characters, with the corresponding ASCII code generated by the Assembler, is given in Table 1.

If the mnemonic C is used in the operand field, the characters are packed two per word in the following manner:

DC C'TODAY' produces

0 7	8 15
O	T
A	D
ØØØ	Y

The mnemonic T, on the other hand, causes the characters to be packed two per word as shown below:

DC T'TODAY' produces

0 7	8 15
T	O
D	A
Y	ØØØ

Examples (octal print)

PC	OBJ CODE		STATEMENT	
00100	000254	LAB1	DC	C','
00101	126000	LAB2	DC	T','
00102	023450		DC	T''('

5.2 ORG Instruction

The ORG instruction is used to change the current value of PC and has the following statement format:

[label[:]] blank(s) {ORG} blank(s) $\left\{\begin{matrix} 0(\text{octal}) \\ \text{addr} \end{matrix}\right\}$ blank(s) [[;]comments]

The value of the expression 'addr' or the octal constant becomes the current value of PC. Note that this value must represent a location within the core size of the computer being used (see Section 6.2.1). The Assembler initially sets PC to zero.

5.3 DS Instruction

Blocks of storage may be explicitly allocated by using the DS (Define Storage) instruction and the statement takes the form:

[label[:]] blank(s) {DS} blank(s) {number} blank(s) [[;]comments]

The expression 'number' defines the number of words of storage to be allocated.

5.4 EQU Instruction

A label may be directly assigned a numeric value by the use of the EQU (equivalence) instruction. The statement takes the form:

[label[:]] blank(s) {EQU} blank(s) $\left\{\begin{matrix} 0(\text{octal}) \\ \text{value} \end{matrix}\right\}$ blank(s) [[;]comments]

The label on this statement is assigned the value of the expression 'value' or the octal constant.

5.5 REPEAT Instruction

The REPEAT instruction can be used to initialise blocks of words or double words of storage so that they contain a copy of the last word or double word of object code assembled. The REPEAT instruction takes the form:

[label[:]] blank(s) {REPEAT} blank(s) {number} blank(s) [[;]comments]

The expression 'number' defines the number of times the last word or double word of object code assembled is repeated. If a REPEAT instruction occurs after a DC $\left\{\begin{matrix}T\\C\end{matrix}\right\}$ 'character(s)' instruction then the last word or double word of object code assembled for any instruction other than a DC $\left\{\begin{matrix}T\\C\end{matrix}\right\}$ 'character(s)' instruction is repeated.

Examples (octal print)

PC	OBJ CODE		STATEMENT	
00460	000000	LAB	DC	Z(A)
	000012			
00462	000000		REPEAT	30
	000012			
00556	100001		DC I	A(1)
00557	100001		REPEAT	12
00573	000101		DC	C'A'
00574	100001		REPEAT	1
00200			ORG	0(200)
00200	100001		REPEAT	2

5.6 USING Instruction

The USING instruction indicates to the Assembler that there is a value in accumulator 2 or 3 to be used for relative addressing. The value set up by the USING instruction is used, when required, if the expression 'disp' appears on its own as an operand address in a MR instruction (see Section 4.3.1,2). The statement takes the form:

blank(s) {USING} blank(s) {base, addr} blank(s)[[;]comments]

The expression 'base' must have the value 2 or 3 and indicates that the corresponding accumulator contains an address that can be used for relative addressing purposes. The expression 'addr' defines the address assumed to be contained in the specified accumulator.

It is the programmer's responsibility to ensure that accumulators 2 and 3 are correctly loaded at execution time.

5.7 DROP Instruction

The DROP instruction indicates to the Assembler that accumulator 2 or 3 no longer contains an address that can be used for relative addressing purposes. The statement takes the form:

blank(s) {DROP} blank(s) {base} blank(s) [[;]comments]

5.8 PRINT Instruction

The assembly listing may be printed out with the numeric fields represented in octal or hexadecimal by specifying the required option on a PRINT statement. This statement takes the form:

blank(s) {PRINT} blank(s) { HEX[ADECIMAL] / OCT[AL] } blank(s) [[;]comments]

The first three characters in the operand field indicate whether hexadecimal or octal output is required. By using a number of PRINT statements different parts of the assembly can be printed out in octal and hexadecimal.

The final selection of the PRINT option determines the representation used to print the value of labels in the cross-reference that may appear at the end of the assembly. This final selection of the PRINT option also determines the representation used to print the numeric fields in the trace output if control is passed to the Simulator.

PRINT OCTAL is initially assumed in the Assembler.

5.9 EJECT Instruction

The assembly listing is continued at the top of a new page after an EJECT instruction is processed. This statement takes the form:

blank(s) {EJECT} blank(s) [[;]comments]

5.10 NOLIST and LIST Instructions

Listing of certain parts of a source program can be suppressed by using the NOLIST and LIST instructions. The statements take the form:

blank(s) { NOLIST / LIST } blank(s) [[;]comments]

Once the 'NOLIST' instruction is processed by the Assembler, the listing of all subsequent source statements (provided they are error-free) is suppressed until a 'LIST' instruction is processed. However, if a source statement contains an error, then that source statement and the corresponding error messages are printed out even if NOLIST is specified.

LIST is initially assumed in the Assembler.

5.11 END Instruction

The END instruction indicates the physical end of the source program and the statement takes the form:

blank(s) {END} blank(s) [addr] blank(s) [[;]comments]

If the expression 'addr' is present its value is taken to be the entry point of the program; otherwise an entry point of zero is assumed.

5.12 TRON and TROFF Instructions

A trace bit is associated with each word of object code that is stored in pseudo-core (see Appendix 4). Once a TRON instruction is processed by the Assembler, all words of object code that are subsequently stored in pseudo-core have the trace bit set ON. This continues until a TROFF statement is processed.

The trace bit is used by the Simulator to check if the result of simulating an instruction is to be printed (traced) or not (see Section 9.1). The TRON, TROFF instructions thus allow a programmer to suppress the trace when tested parts of a program are being simulated.

The statements take the form:

blank(s) $\left\{\begin{matrix}\text{TRON}\\\text{TROFF}\end{matrix}\right\}$ blank(s) [[;]comments]

'TRON' is initially assumed in the Assembler.

5.13 RDONLY Instruction

A read-only memory bit is associated with every word in pseudo-core (see Appendix 4). Once a RDONLY instruction is processed by the Assembler, the locations in pseudo-core that correspond to the read-only memory (ROM) module have the read-only memory bit set on.

The statement has the following format:

blank(s) {RDONLY} blank(s) {start, finish} blank(s) [[;]comments]

The expressions 'start' and 'finish' define the core address of the beginning and end of a read-only memory module.

A number of RDONLY instructions may have to be used to describe all of the ROM modules associated with a particular NOVA or SUPERNOVA computer.

6. OPTIONS AVAILABLE WITH THE ASSEMBLER

6.1 Control Cards Needed to Execute the Assembler

The Assembler can be executed by using the following job control language

(JCL)* cards, in addition to the usual job card and JOBLIB or STEPLIB card needed to point to the private library containing the program NOVASM:

```
//ASM        EXEC PGM=NOVASM
//SYSPRINT   DD SYSOUT=A
//SYSPUNCH   DD UNIT=SYSCP
//SYSUT1     DD DSN=&&NOVA,SPACE=(CYL,(1,1)),UNIT=SYSDA
//ASM.SYSIN  DD *
   {source statements}
/*
```

This form of the job control cards for NOVASM introduces the DD (control card) names SYSPRINT, SYSPUNCH, SYSUT1 and SYSIN that are used by the Assembler.

The necessary JCL cards may be set up as a catalogued procedure,* and the Assembler can be executed by simply using the cards:

```
//          EXEC  NOVASM
//ASM.SYSIN  DD *
   {source statements}
/*
```

6.2 Parameters on the EXEC Card

There are a number of Assembler options available to the programmer, and these may be selected by specifying certain parameters in the PARM field of the EXEC card.* The allowable parameters are shown below and must be enclosed between apostrophes if more than one parameter is to be specified or if a single option is required and it includes an equal sign.

```
//EXEC NOVASM,PARM='[CORE=value{K},] [SYMTAB=number(s),]
[{NOXREF       },][{SNOVA  },][{LOAD           },]
 {XREF         }, {NOVA    }, {LOAD=character(s)}, ]
 {XREF=number(s)}              {NOLOAD          }
[{DECK  },]   [LNCNT=number(s)]'
 {NODECK}, ]
```

Throughout the PARM field the term 'number(s)' must be replaced by decimal digits and define the value of the corresponding parameter.

---

* IBM 360 Operating System - Job Control Language, Form C28-6539-9, July 1969.

6.2.1 CORE parameter

The CORE parameter defines the core storage size of the NOVA or SUPERNOVA computer being used. Its value is used to check the validity of addresses specified by the source program, or addresses referenced by a machine language program when the Simulator is used. The term 'value' must be an integer in the range 1-32.

The Assembler initially assumes CORE=12K.

6.2.2 SYMTAB parameter

The maximum number of labels that can be used in a source program is defined by the SYMTAB parameter. The parameter actually restricts the number of symbol table entries and limits the amount of IBM 360 core storage required for the symbol table.

The default option is SYMTAB=1000.

6.2.3 XREF parameter

A cross-reference is printed at the end of the assembly if XREF is specified. This cross-reference gives the value and definition (statement number where defined) of each label, all references to each label (statement number of statements where this label is referred to), or notes if the label is undefined or multiply defined.

The value given for the XREF parameter defines the maximum number of references to labels that is allowed in the source program. This parameter also limits the amount of IBM 360 core storage required for the cross-reference list (see Appendix 2).

The default option is XREF=3000.

6.2.4 SNOVA and NOVA parameters

The SNOVA and NOVA parameters are used to indicate if the assembly is to be for a SUPERNOVA or NOVA computer.

The default option is NOVA.

6.2.5 LOAD parameter

If LOAD is specified a pseudo-core area is set up by the Assembler. The size of this area depends on the value of the CORE parameter, four bytes of IBM 360 core storage being required for each NOVA or SUPERNOVA word (see Appendix 4).

If the 'LOAD=character(s)' form of this parameter is used, then the term 'character(s)' may be replaced by up to four alphanumeric characters. These characters are appended to the characters NOVA and control is passed to this program at the end of assembly. For example, control is passed to the Simulator

by specifying the LOAD=SIM option on the EXEC card.

When LOAD is specified, register 1 points to a 24-byte parameter list* on normal exit from the Assembler (see Section 8). This six full-word parameter list contains the IBM 360 address of pseudo-core, the value of the CORE parameter, the source program entry point, the type of NOVA computer (value zero for NOVA and one for SUPERNOVA), the number of lines of output per page (for the device specified on the SYSPRINT DD card) and a PRINT OCTAL or HEX indicator (value one for PRINT OCTAL and zero for PRINT HEX).

The default option is NOLOAD.

6.2.6 DECK parameter

A copy of the object code set up by the Assembler is written out onto the device defined by the SYSPUNCH DD card in object deck format (see Appendix 5) if DECK is specified.

The default option is NODECK.

6.2.7 LNCNT parameter

The number of lines of output per page (including headings) on the device defined by the SYSPRINT DD card is defined by the LNCNT parameter.

The default option is LNCNT=60. If a value less than seven is given for LNCNT, the default option of 60 is used.

7. ALLOCATION OF CORE STORAGE BY THE ASSEMBLER

NOVASM is a two-pass Assembler. Source statements are read one record (80-bytes) at a time from the device specified by the SYSIN DD card into core storage and undergo Pass 1 processing. After being processed in Pass 1, each source statement plus appropriate error information (see Appendix 3) is spooled as an 88-byte record onto the device specified by the SYSUT1 DD card. This SYSUT1 output is read back into core storage and processed in Pass 2.

The SYSIN data can be blocked, and to provide efficient spooling the SYSUT1 output should be blocked. If no record length (LRECL) or block size (BLKSIZE) is specified on the SYSUT1 DD card, then SYSUT1 output is automatically blocked, the corresponding BLKSIZE being dependent on the device referred to by the SYSUT1 DD card. For a 2314 disk drive, the parameters (RECFM=FB,LRECL=88, BLKSIZE=7216) are allocated. For a 2311 disk drive or a magnetic tape unit the parameters (RECFM=FB,LRECL=88,BLKSIZE=3608) are allocated.

If DECK is specified in the PARM field of the EXEC card, then a copy of the

---

* IBM 360 Operating System - Supervisor and Data Management Services, Form C28-6646-2, November 1968.

object code set up by the Assembler is written out onto the device defined by the SYSPUNCH DD card. The value of LRECL or BLKSIZE may be specified on the SYSPUNCH DD card, but if neither of these are defined then certain default values are allocated depending on the SYSPUNCH device type. For a unit record device (such as a card punch), the parameters (RECFM=F,LRECL=80,BLKSIZE=80) are allocated. For a 2314 disk drive, the parameters (RECFM=FB,LRECL=80,BLKSIZE=7200) are allocated. For a 2311 disk drive or magnetic tape unit, the parameters (RECFM=FB,LRECL=80,BLKSIZE=3600) are allocated.

The SYSPRINT output may also be blocked, but if neither the LRECL nor the BLKSIZE are specified on the SYSPRINT DD card, then the parameters (RECFM=FA, LRECL=121,BLKSIZE=121) are automatically allocated. This comment also applies to the SYSPRINT output from NOVASIM.

The Assembler begins by analysing the information in the PARM field of the EXEC card and then OPENS the DCB's (Data Control Blocks) for the SYSPRINT,SYSIN, SYSUT1 and SYSPUNCH (if DECK is specified) DD cards.

Space is then requested for the Assembler work areas by using the GETMAIN macro instruction in IBM 360 Assembler language*. The work area consists of:

(i) Pseudo-core and 24 bytes for the parameter list if LOAD is specified.

(ii) Cross-reference list area if XREF is specified.

(iii) Symbol table area.

An upper limit for the GETMAIN request is calculated on the basis of the values of the CORE, XREF and SYMTAB parameters and whether LOAD and/or XREF are specified.

However, the amount of core storage available for the GETMAIN request depends on the size of the IBM 360 computer being used and the storage required for OS use on this computer. The core storage required for system use is difficult to state, especially if the MFT (multiprogramming with a fixed number of tasks) or MVT (multiprogramming with a variable number of tasks) version of OS is being used.

Consequently the amount of core storage obtained by the GETMAIN request may be less than the upper limit calculated. The actual amount of work area obtained is printed out on the first page of the Assembler output under the usual job control language messages that appear at the start of the job step (see the sample programs in Appendix 7).

---

* IBM 360 Operating System - Supervisor and Data Management Macro Instructions, Form C28-6647-3, November, 1968.

If LOAD is specified, a pseudo-core area, based on the value of the CORE parameter, and a 24-byte parameter list are set up from the Assembler work area. If it is not possible to allocate this area an error message is printed out and the job step terminated.

The remaining work area is then divided up into a cross-reference list area, if XREF is specified, and a symbol table area. If neither the XREF nor the SYMTAB parameters appear in the PARM field, this division is carried out automatically to allow three times as many cross-references as symbol table entries. If only the XREF parameter appears in the PARM field, the requested cross-reference list area is allocated, or an error message given, and the remaining work area is divided up as a symbol table area. If only the SYMTAB parameter appears in the PARM field, the requested symbol table area is allocated, or an error message given, and the remaining work area is divided up for a cross-reference list area, if XREF is specified.

When both the XREF and SYMTAB parameters appear in the PARM field, the requested cross-reference list area is allocated, or an error message given, and then the requested symbol table area is allocated, or an error message given.

The values of the CORE, SYMTAB and XREF parameters that correspond to the work areas set up for a particular Assembler run are printed on the first page of the Assembler output.

For example, on the first page of the Assembler output for Sample Program 1 in Appendix 7, the above scheme gives:

WORK AREA ALLOCATED (HEX BYTES) = 013548
CORE = 12K
XREF = 3000
SYMTAB = 1000

Note that,

(CORE*4+24)+(XREF*4)+(SYMTAB*18)=work area in bytes (decimal).

At the end of assembly the core storage allocated for the cross-reference list, if XREF is specified, and the symbol table is freed by using the FREEMAIN macro instruction.*

The dynamic allocation of cross-reference list and symbol table areas should prove to be very useful, particularly for an initial Assembler run on an IBM 360

---

* IBM System 360 Operating System - Supervisor and Data Management Macro Instructions, Form C28-6647-3, November 1968.

computer with a small amount of core storage or for a run that must operate within a small partition in a multiprogramming environment.

The above scheme also has the advantage that the programmer knows the amount of work area allocated by the Assembler without requiring any knowledge of the core storage needed for system use.

It should also be possible to assemble and load pseudo-core (and hence simulate) for a 4K NOVA or SUPERNOVA computer on an IBM 360 computer with only 64K bytes of core storage, provided that SYSPRINT, SYSIN, SYSUT1 and SYSPUNCH are unblocked.

8. ERROR MESSAGES FROM THE ASSEMBLER

Any source statement errors that are detected by the Assembler cause an error code to be stored with the appropriate statement in the source statement buffer (see Appendix 6). Up to four different error codes are stored for each statement and these cause the statement to be flagged as an error on the assembly listing. An error message for each error code present is printed under the relevant source statement.

A severity code is associated with each error code and the highest severity of error (zero if no errors) is returned as a condition code in register 15 on exit from the Assembler. However it is lost if control is passed to another program using the 'LOAD-character(s)' option.

At the end of the assembly the number of statements flagged as errors and the highest severity of error (unless no statements are flagged) are printed out.

The Assembler responds to the severity of errors in the following way:

Severity 4 - Does not affect the assembly process, statements treated as comments.

Severity 8 - Usually causes invalid object code to be generated, but does not affect Assembly process.

Severity 12 - Assembly process continues, but a parameter list is not set up, whether LOAD is specified or not, and control is not passed to a program selected by the 'LOAD=character(s)' option.

Severity 16 - Assembly is immediately terminated and the job step flushed.

A list of the error messages and their severity codes that can be printed by the Assembler is given in Appendix 6.

9. THE SIMULATOR

The NOVASIM program uses the information set up in pseudo-core to simulate and display the action of a NOVA or SUPERNOVA machine language program. All memory reference and arithmetic and logical instructions are simulated, but in the I/O class of instructions only the optional hardware multiply-divide instructions are simulated.

NOVASIM can be executed by passing control to it via the LOAD=SIM option in the Assembler. The program checks that register 1 points to a six full-word parameter list (see Section 6.2.6) and assumes that pseudo-core has been set up to contain the relevant NOVA or SUPERNOVA object code.

9.1 Trace Feature

NOVASIM 'executes' a machine language program by simulating the action of instructions fetched from pseudo-core. Simulation commences by fetching an instruction from the pseudo-core location that corresponds to the program entry point passed in the third word of the register 1 parameter list.

As each instruction is executed the results of simulation are printed if the trace bit is set in the pseudo-core location corresponding to the value of PC for that instruction.

Any trace output is printed under the following headings (see the sample programs in Appendix 7):

NOVA or SNOVA TIME - This is followed by the time, in microseconds, after the instruction is executed relative to the start of simulation.

AC SAME - An asterisk is printed under this heading if the contents of Carry and the four accumulators are unaltered by an instruction that could change the value of these quantities.

PC - Value of the program counter for the instruction being executed is put under this heading.

INSTRUCTION MNEMONIC - Instruction mnemonics are printed that can be used to generate the object code fetched from pseudo-core for this instruction. All numbers are in decimal under this heading.

CORE ADDR - The effective address specified as an operand in an instruction that references core storage is given. For instructions using the NOVA hardware multiply-divide option, the buffer

used in the instruction is listed under the heading as BUFF $\left\{ \begin{matrix} A \\ B \\ C \end{matrix} \right\}$. This heading is also used to indicate NO-OPS by printing NOP under this heading.

CORE BEFORE - The contents of the location defined by the effective address before the instruction is executed are given for instructions that reference memory. For NOVA hardware multiply-divide instructions the contents of the appropriate buffer before execution are given under this heading.

CORE AFTER - Contents of the location defined by the effective address after the instruction is executed are given for an instruction that alters memory. For NOVA hardware multiply-divide instructions the contents of the appropriate buffer after execution are given under this heading.

CARRY, SOURCE AND DESTN ACS INTO AU - For an arithmetic and logical instruction the contents of Carry and the source and destination accumulators before entering the arithmetic unit are given.

CARRY AND AC FROM AU - Contents of Carry and the destination accumulator are given after the instruction is processed by the arithmetic unit.

CARRY, AC FROM SHIFTER - Contents of Carry and the destination accumulator are given after the instruction is processed by the shifter.

CARRY AND ACCUMULATORS 0,1,2,3 AFTER INSTRUCTION - The contents of Carry and the four accumulators are given after every instruction.

If indirect addressing is used in an effective address calculation then the number of levels of indirect addressing is shown under the heading INSTRUCTION MNEMONIC. For example the instruction mnemonic JSR 3 44 indicates that the effective address references location 44 (decimal), and that three levels of indirect addressing are used in this effective address calculation. A maximum of eight levels of indirect addressing is allowed by the Simulator.

Apart from the numbers that appear under the headings NOVA or SNOVA TIME and INSTRUCTION MNEMONIC, the output from the Simulator can be printed in octal or hexadecimal depending on the value of the last word of the register 1 parameter list (see Section 6.2.6).

The number of lines per page of Simulator output (for the device defined by the SYSPRINT DD card), including headings, is determined by the value of the fifth word in the register 1 parameter list.

9.2 Instruction Timing

The time required to execute a NOVA or SUPERNOVA machine instruction depends on a number of factors, and appropriate timing formulae are given in Table 2. The values of the constants used in the timing formulae of Table 2 are listed in Table 3.

9.3 Error Messages from the Simulator

Simulation is terminated if a HALT instruction is encountered or an error condition is detected. The value of PC for the instruction that caused simulation to be terminated and an appropriate message are then printed out.

The different conditions that end simulation are listed below:

(1) HALT INSTRUCTION ENCOUNTERED
(2) ATTEMPT TO EXECUTE I/O INSTRUCTION
(3) EIGHT LEVELS OF INDIRECT ADDRESSING EXCEEDED
(4) ADDRESS OUTSIDE AVAILABLE CORE STORAGE
(5) ADDRESS NEGATIVE
(6) ATTEMPT TO USE AUTO-INDEXING REGISTERS IN ROM
(7) ATTEMPT TO ALTER READ-ONLY MEMORY
(8) ATTEMPT TO FETCH RESULTS FROM DEVICE MDV BEFORE STARTING A MULTN OR DIVN
(9) REQUIRED TIME HAS NOT ELAPSED BEFORE TRYING TO FETCH RESULTS FROM DEV MDV
(10) ATTEMPT TO MULTIPLY OR DIVIDE BEFORE ALL THE BUFFERS A,B,C WERE LOADED
(11) ATTEMPT TO FETCH INSTRUCTION FROM AN UNASSIGNED LOCATION

If an incorrect number of parameters are passed to NOVASIM via register 1, or the SYSPRINT DD card is not supplied for the job step where the Simulator is executed, then simulation is terminated and a condition code of 16 is returned via register 15. In all other cases a condition code of zero is passed via register 15.

10. CONCLUSIONS

The Assembler NOVASM provides a fast and convenient way of preparing machine language programs for the NOVA and SUPERNOVA computers.

It is possible to specify a number of options on the PARM field of the EXEC card and included in these are the parameters CORE (if LOAD is specified),

XREF and SYMTAB. Their values are used to obtain IBM 360 core storage for pseudo-core, a cross-reference list and a symbol table. By choosing certain values of these three parameters it is possible to use the Assembler on any size IBM 360 computer operating under OS/360. If SYSPRINT, SYSIN, SYSUT1 and SYSPUNCH are unblocked it should also be possible to assemble and load pseudo-core (and hence simulate) for a 4K NOVA or SUPERNOVA computer on an IBM 360 computer with only 64K bytes of core storage.

Comprehensive error detection features are provided by the Assembler and its output includes a source program listing in octal or hexadecimal with appropriate error messages, a cross-reference if XREF is specified, and a copy of the assembled machine language code in object deck format if DECK is specified.

A copy of the assembled machine language code is also stored in pseudo-core if LOAD is specified, and control passed from the Assembler to another program if the 'LOAD-character(s)' form of this parameter is used.

The Simulator NOVASIM provides an efficient way of debugging machine language programs for the NOVA or SUPERNOVA computer, and can be run on any size IBM 360 computer operating under OS/360. All memory reference instructions, arithmetic and logical instructions, and the optional hardware multiply-divide instructions are simulated. Output from the Simulator can be printed out in octal or hexadecimal and provides valuable information for program debugging.

11. ACKNOWLEDGEMENTS

The author thanks Mr. D. J. Richardson, Mr. R. P. Backstrom and Mr. C. B. Mason for valuable discussions, and particularly thanks Mr. N. W. Bennett for helpful discussions in the early stages of the work.

TABLE 1

ASCII CODES WITH EVEN PARITY FOR THE CHARACTERS

AVAILABLE ON THE IBM 029 CARD PUNCH

Characters	ASCII Code (Even Parity)	
	Octal	Hexadecimal
Alphabetic (upper case only)		
A	101	41
B	102	42
C	303	C3
D	104	44
E	305	C5
F	306	C6
G	107	47
H	110	48
I	311	C9
J	312	CA
K	113	4B
L	314	CC
M	115	4D
N	116	4E
O	317	CF
P	120	50
Q	321	D1
R	322	D2
S	123	53
T	324	D4
U	125	55
V	126	56
W	327	D7
X	330	D8
Y	131	59
Z	132	5A
Numeric		
0	060	30
1	261	B1
2	262	B2
3	063	33
4	264	B4
5	065	35
6	066	36
7	267	B7
8	270	B8
9	071	39

continued...

TABLE 1 (continued)

Special Characters		ASCII Code (Even Parity)	
		Octal	Hexadecimal
!	Exclamation mark	041	21
"	Quotation mark	042	22
#	Number sign	243	A3
$	Dollar sign	044	24
%	Per cent	245	A5
&	Ampersand	246	A6
'	Prime or apostrophe	047	27
(	Left parenthesis	050	28
)	Right parenthesis	251	A9
*	Asterisk	252	AA
+	Plus sign	053	2B
,	Comma	254	AC
-	Minus sign	055	2D
.	Period, decimal point	056	2E
/	Slash	257	AF
@	At sign	300	C0
?	Question mark	077	3F
:	Colon	072	3A
;	Semi-colon	273	BB
<	Less-than sign	074	3C
=	Equal sign	275	BD
>	Greater-than sign	276	BE
	Space, blank	240	A0
↑	Vertical arrow - use | on 029 punch (12-7-8 punchings)	336	DE
←	Assignment arrow - use → on 029 punch (11-7-8 punchings)	137	5F
	Carriage return - use ¢ on 029 punch (12-2-8 punchings)	215	8D
	Line feed - use underline on 029 punch (0-5-8 punchings)	012	0A

NOTE: The exclamation mark (!) and cent sign (¢) are not printed out when the QN chain is used on the IBM 1403 printer.

TABLE 2

INSTRUCTION TIMES FOR THE NOVA AND SUPERNOVA COMPUTERS

The formulae used in the Simulator to calculate instruction times for the NOVA and SUPERNOVA computers are listed below. Examples of instruction times are also given where it is assumed that there is no indirect addressing (and this precludes the use of the auto-indexing feature) and no use of accumulators 2 or 3 as base registers. The times given for the ROM case apply when the instruction comes from ROM and the operand comes from core memory.

Instruction	Formula for Calculating the Instruction Time	TIME (microseconds)			
		NOVA		SUPERNOVA	
		Core	Read-only	Core	Read-only
LDA	$M_{INST}+M_{OPD}+M_{BASE}+\Sigma M_{AUTO}+\Sigma M_{IND}$	5.2	5.0 4.8 *	1.6	1.4 1.2 *
STA	$M_{INST}+M_{OPD}+M_{STA}+M_{BASE}+\Sigma M_{AUTO}+\Sigma M_{IND}$	5.5	5.3	1.6	1.4
ISZ DSZ	$M_{INST}+M_{OPD}+M_{INCR}+M_{BASE}+\Sigma M_{AUTO}+\Sigma M_{IND}$	5.2	5.0	1.8	1.6
JMP	$M_{INST}+M_{BASE}+\Sigma M_{AUTO}+\Sigma M_{IND}$	2.6	2.4	0.8	0.6
JSR	$M_{INST}+M_{BASE}+M_{JSR}+\Sigma M_{AUTO}+\Sigma M_{IND}$	3.5	3.3	1.4	1.2
COM NEG MOV INC	$(M_{INST}+M_{ANL1}+M_{SKIP})*M_{PREV}$	5.6	5.4	0.8 1.6 ‡	0.6 0.3 † 1.2 ‡ 0.6 ▼
ADC SUB ADD AND	$(M_{INST}+M_{ANL2}+M_{SKIP})*M_{PREV}$	5.9	5.7	0.8 1.6 ‡	0.6 0.3 † 1.2 ‡ 0.6 ▼
DIA DIB DIC	M_{INPUT}	4.4	4.2	2.9	2.8
NIO	M_{NIO}	4.4	4.2	3.3	3.2
DOA DOB DOC	M_{OUTPUT}	4.7	4.5	3.3	3.2
MUL	M_{MUL} for SUPERNOVA M_{OUTPUT} or M_{NIO} for NOVA but must wait M_{MUL} microseconds before fetching results	6.4	6.4	5.4	5.3
DIV	M_{DIV} for SUPERNOVA M_{OUTPUT} or M_{NIO} for NOVA, but must wait M_{DIV} microseconds before fetching results	7.2	7.2	6.9 1.6 ♦	6.8 1.5 ♦

* Operand also from read-only memory

† Successive ANL instructions executed

‡ Skip condition obeyed

▼ Skip condition obeyed and successive ANL instructions executed

♦ Unsuccessful division

TABLE 3

CONSTANTS USED IN THE TIMING FORMULAE FOR THE NOVA AND SUPERNOVA COMPUTERS

Constant	Definition	TIME (microseconds)			
		NOVA		SUPERNOVA	
		Core	Read-only	Core	Read-only
M_{INST}	Time to fetch instruction	2.6	2.4	0.8	0.6
M_{OPD}	Time to fetch operand	2.6	2.4	0.8	0.6
M_{BASE}	Time for use of accumulator 2 or 3 as a base register	0.3	0.3	0	0
M_{IND}	Time for each level of indirect addressing	2.6	2.4	0.8	0.6
M_{AUTO}	Time for each use of auto-indexing locations	0	0	0.2	-
M_{SKIP}	Extra time if skip condition is obeyed	0	0	0.8	0.6
M_{STA}	Extra time for STA instruction	0.3	0.3	0	0
M_{JSR}	Extra time for JSR instruction	0.9	0.9	0.6	0.6
M_{INCR}	Extra time for DSZ or ISZ instruction	0	0	0.2	-
M_{ANL1}	Extra time for first group of ANL instructions	3.0	3.0	0	0
M_{ANL2}	Extra time for second group of ANL instructions	3.3	3.3	0	0
M_{PREV}	Multiplying factor if previous instruction was an ANL instruction	1	1	1	0.5
M_{INPUT}	Time for input instruction	4.4	4.2	2.9	2.8
M_{NIO}	Time for NIO instruction	4.4	4.2	3.3	3.2
M_{OUTPUT}	Time for output instruction	4.7	4.5	3.3	3.2
M_{MUL}	Multiplication time for SUPERNOVA. Time that NOVA must wait before fetching results of hardware multiply	6.4	6.4	5.4	5.3
M_{DIV}	Division time for SUPERNOVA. Time that NOVA must wait before fetching results of hardware division	7.2	7.2	6.9 1.6 *	6.8 1.5 *

* Unsuccessful division

APPENDIX 1

THE SYMBOL TABLE

A label and its attributes is entered into a table called the symbol table. Each entry in the table occupies 18 bytes. The label is stored in the first 8 bytes, left justified and padded with blanks where necessary.

The next byte is a TYPE field and is used to distinguish between multiple definitions of a label. This byte is equal to $(40)_{16}$ for a normal table entry, $(41)_{16}$ for the second definition of a label, $(42)_{16}$ for the third definition of a label etc.

The tenth byte is an INDEX byte. This byte contains $(FF)_{16}$ if the label was undefined. If there is one proper definition of the label, the INDEX byte is $(00)_{16}$. If there are two definitions of a label the INDEX byte for the original definition (TYPE=$(40)_{16}$) is $(01)_{16}$, while the INDEX byte for the second entry in the symbol table (TYPE=$(41)_{16}$) is $(00)_{16}$. Thus when a new definition of a label is to be added to the symbol table for a label that is already defined, the TYPE field plus the INDEX field of the original definition of the label are added to form the TYPE field of the new definition of the label. The INDEX field is increased by one and is stored as the new INDEX value for the original definition of the label. The INDEX field for the new definition of the label is $(00)_{16}$.

Bytes 11 and 12 contain the statement number where the label was defined.

The value of the label is stored in the next two bytes.

Bytes 15 and 16 contain a pointer that can be used to locate an entry in the cross-reference list where the statement number of the statement that first referred to this label is stored.

The last two bytes of the symbol table entry contain a pointer that can be used to locate an entry in the cross-reference list where the statement number of the statement that last referred to this label is stored.

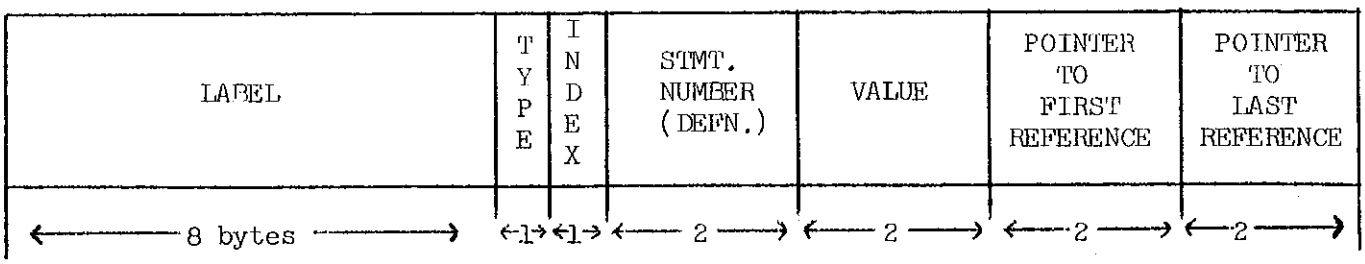
Four times the cross-reference pointer plus the address of the start of the cross-reference list, gives the address of the relevent entry in the cross-reference list.

If there were no references to this label or NOXREF were specified, each of the last four bytes of the symbol table entry would contain $(FF)_{16}$.

The format of each symbol table entry is summarised below:

continued...

APPENDIX 1 (continued)



APPENDIX 2

THE CROSS-REFERENCE LIST

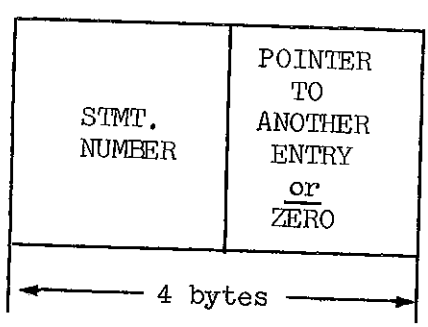
The cross-reference list consists of a series of full word (4-byte) entries for each reference to a label, and is set up if XREF has been specified.

The first two bytes of each entry contain the statement number of the statement that referred to a given label.

The last two bytes of the entry are zero if this was the last reference to the label. Otherwise, these two bytes contain a pointer that can be used to locate another entry in the cross-reference list that contains the statement number of the next statement that refers to the same label. Four times the pointer plus the address of the start of the cross-reference list gives the address of this second entry.

The format of each cross-reference list entry is summarised below:

CROSS-REFERENCE LIST ENTRY



APPENDIX 3

RECORDS SPOOLED ONTO SYSUT1

As discussed in Section 7, source statements plus appropriate error information are spooled as 88-byte records onto the device specified by the SYSUT1 DD card.

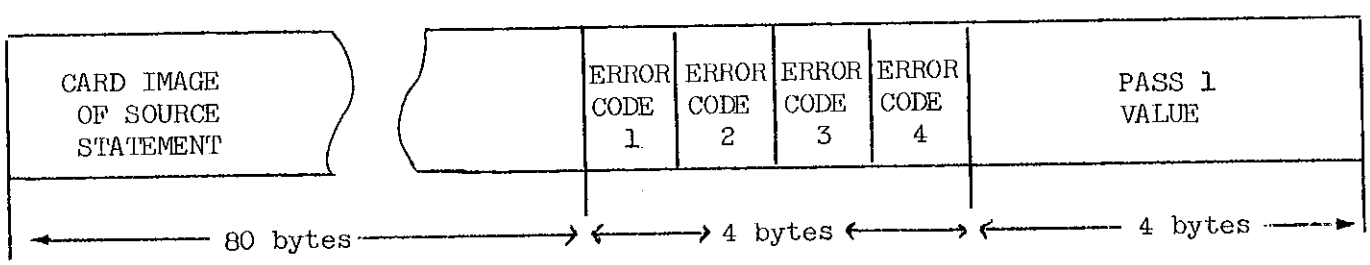
The first 80 bytes of each record contain the card image of a source statement.

Error information is held in the next four bytes. A one-byte error code is stored for each of the first four different types of error condition that occur for a given source statement (16 times the error code plus a base address points to an appropriate error message in the Assembler).

The last four bytes are used if the source statement is completely processed in Pass 1 of the assembly. This full word contains the new value of the program counter for an ORG statement, the operand value for a DS or REPEAT statement or the number of NOVA words to be initialised by a DC $\left\{\begin{matrix}T\\C\end{matrix}\right\}$ 'character(s)' statement.

The format of each record spooled onto SYSUT1 is summarised below:

FORMAT OF EACH RECORD SPOOLED ONTO SYSUT1



APPENDIX 4

PSEUDO-CORE

A pseudo-core area, consisting of one NOVA or SUPERNOVA word per 360 word (4 bytes), is set up if LOAD has been specified, and its size is determined by the value of the CORE parameter. The NOVA or SUPERNOVA word is stored in the last two bytes of the 360 word.

The first 3 bits of the 360 word are used to describe the attributes of the corresponding NOVA or SUPERNOVA word. These attributes are used by the Simulator and are defined as follows:

bit 0 - Assigned location bit. (Set to zero for any location in pseudo-core where object code is stored whether the code is valid or not).

bit 1 - Trace bit. (Set to one if the associated instruction in pseudo-core is to be traced).

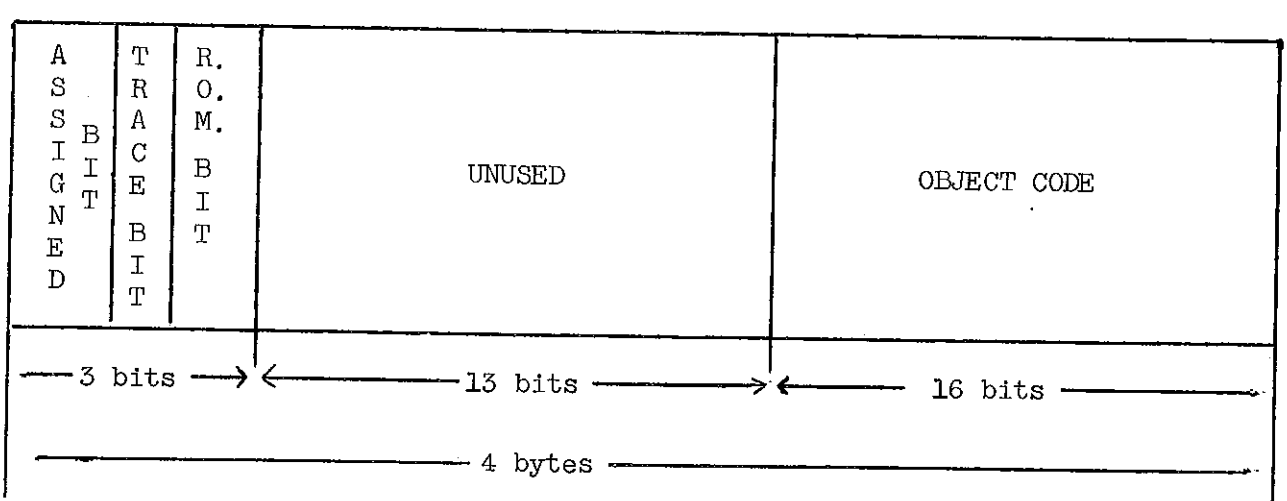
bit 2 - Read-only memory bit. (Set to one if the associated location is a part of read-only memory).

The remaining bits of the first two bytes of the 360 word are unused.

When pseudo-core is first set up by the Assembler, all of the 360 words are initialised with the constant $(80000000)_{16}$.

The format of each 360 word in pseudo-core is summarised below:

360 WORD IN PSEUDO-CORE



APPENDIX 5

OBJECT DECK FORMAT

By specifying the DECK option, a copy of the object code set up by the Assembler is written out onto the device defined by the SYSPUNCH DD card in a special object deck format which is intended for punched card output.

The object deck for a NOVA or SUPERNOVA program consists of a number of standard object cards followed by a transfer card. The standard object cards contain the assembled machine code, while the transfer card indicates the entry point of the program (see Section 5.11).

The cards have the following format:

Column 1 - contains $(06)_{16}$ for all cards.

Columns 2 to 4 - contain the characters 'TXT' for object cards, and contain the characters 'END' for a transfer card.

Column 5 - contains $(75)_{16}$ for the NOVA or $(76)_{16}$ for the SUPERNOVA.

Columns 6 to 11 - contain the characters 'NOVA' or 'SNOVA' left justified and padded with blanks.

Column 12 - contains $(02)_{16}$ indicating that one word of object code is punched in two card columns.

Column 13 - the number of words of object code punched on this card (word count).

Columns 14 to 16 - contain the address of the first word of object code, or the entry point address for a transfer card.

Columns 17 to 72 - contain object code, 2 columns per word with a maximum of 28 words per card. These columns are blank on a transfer card.

Columns 73 to 75 - blank.

Columns 76 to 80 - contain a five digit sequence number.

At the A.A.E.C. this object deck can be read as data by a program in the PDP-9/L computer, and this punches an object tape using a fast paper tape punch. The object tape is set up in a format acceptable as input to the Block Loader in the NOVA or SUPERNOVA computer.

The information on the transfer card is used to set up the Start Block at the end of the object tape. If the entry point on the transfer card is non-zero, the Start Block is set up to transfer control automatically to this address after the program is loaded into core storage. For an entry point of zero, the Start Block is set up to halt the processor after the program has been loaded.

APPENDIX 6

ERROR MESSAGES AND SEVERITY CODES

All of the possible error messages and the corresponding severity codes that can be generated by the Assembler are listed below:

Severity	Error Message
4	BLANK CARD (TREATED AS COMMENT CARD)
4	CARD WITH EVERY COLUMN PUNCHED
8	LABEL CONTAINS INVALID CHARACTER
8	LABEL TOO LONG (TRUNCATED TO 1st 8 CHARACTERS)
8	SYNTAX ERROR IN EXPRESSION Too many + or - signs etc.
8	NUMBER TOO LARGE (TRUNCATED TO FIRST 8 DIGITS)
8	MULTIPLY DEFINED LABEL The value of the first definition of this label is used for references to this label.
8	INCORRECT USE OF INDIRECT BIT INDICATOR
8	CHARACTER IN WRONG CONTEXT A valid character in the wrong position.
8	ILLEGAL CHARACTER IN NUMBER Illegal character in a decimal number or an 8 or 9 in an octal number.
8	VALUE OF EXPRESSION EXCEEDS 16 BITS (TRUNCATED)
8	A LABEL SHOULD NOT APPEAR ON THIS SOURCE CARD Label used on a USING,DROP,PRINT,EJECT,LIST,NOLIST,END, TRON,TROFF or RDONLY statement.
8	INVALID INSTRUCTION MNEMONIC
8	DEVICE CODE IS INVALID
8	DISPLACEMENT IS INVALID
8	INVALID ACCUMULATOR SPECIFICATION

continued...

APPENDIX 6 (continued)

Severity	Error Message
8	SKIP CONDITION IS NOT VALID
8	LABEL UNDEFINED OR SHOULD BE PREDEFINED
8	ADDRESS OUTSIDE AVAILABLE CORE STORAGE
8	ADDRESS EXCEEDS 15 BITS
8	DOUBLE WORD CONSTANT EXCEEDS 32 BITS
8	INCORRECT EXPRESSION VALUE (SET TO ONE)
8	OPERAND MISSING
8	NO CLOSING BRACKET
8	TOO MANY DECIMAL POINTS IN NUMBER
8	TOO MANY CHARACTERS IN EXPONENT
8	INSTRUCTION VALID ONLY FOR SUPERNOVA COMPUTER
8	INCORRECT FORM OF EXPRESSION
8	CHARACTER STRING NOT TERMINATED BY APOSTROPHE
8	NO CHARACTERS TO BE TRANSLATED TO ASCII CODE
8	A LABEL MUST BE USED ON AN EQU STATEMENT
12	NO END CARD (DUMMY CARD GENERATED)
12	VALUE OF PC OUTSIDE CORE
12	ILLEGAL ENTRY POINT FOR PROGRAM
16	SYNTAX ERROR IN PARM FIELD AT POSITION -- A syntax error has occurred in the PARM field parameter that begins at position --.
16	SYSIN DCB NOT OPENED SYSIN DD card missing
16	SYSUT1 DCB NOT OPENED SYSUT1 DD card missing

continued...

APPENDIX 6 (continued)

Severity	Error Message
16	SYSPUNCH DCB NOT OPENED The parameter DECK was specified, but the SYSPUNCH DD card was missing.
16	UNABLE TO ALLOCATE SPACE FOR XCTL PARM LIST AND PSEUDO-CORE AREA
16	UNABLE TO ALLOCATE SPACE FOR CROSS-REFERENCE LIST AREA
16	UNABLE TO ALLOCATE SPACE FOR SYMBOL TABLE AREA
16	SYMBOL TABLE FULL Exceeded the number of labels specified by the SYMTAB parameter.
16	CROSS-REFERENCE LIST FULL Exceeded the number of cross-reference list entries specified by the XREF parameter.
16	THE DATA SET REFERRED TO ON THE STEPLIB OR JOBLIB OR SYS1.LINKLIB DOES NOT CONTAIN THE MEMBER NOVA---- The program specified by using the LOAD=---- option could not be located.

APPENDIX 7

SAMPLE PROGRAMS

Two sample program listings are included in this Appendix. The first listing gives the Assembler and Simulator output in octal for a source program to be run on the NOVA computer. The second listing gives the Assembler and Simulator output in hexadecimal for the same source program to be run on a SUPERNOVA computer.

The source program used in these examples is written to check the hardware multiply-divide instructions against the software multiply and divide routines supplied by Data General Corporation*.

Also note the allocation of work areas by the Assembler for these two runs. In the first case a cross-reference list area and symbol table area were dynamically set up by the Assembler (see Section 7). In the second case the cross-reference list area and symbol table area have been set up on the basis of the values of the XREF and SYMTAB parameters specified in the PARM field of the EXEC card.

---

* A System Reference Manual for the NOVA and SUPERNOVA, DG NM-2. November 1969.

SAMPLE PROGRAM 1

```
//PLS       JOB           ,0034',DR.P.L.SANGER,MSGLEVEL=1
**JST  70.257  19.02.18 00000 R.17
//SAMPLE1  EXEC  NOVASM,PARM='LOAD=SIM'
XXASM    EXEC PGM=NOVASM                                          00000010
XXSTEPLIB  DD DSN=AAELIB.LMODA,DISP=OLD                           00000020
XXSYSPRINT DD SYSOUT=A                                            00000030
XXSYSPUNCH DD  SYSOUT=B,DCB=(MODE=C)                              00000040
XXSYSUT1   DD DSN=&&NOVA,SPACE=(CYL,(1,1)),UNIT=SYSDA             00000050
//ASM.SYSIN DD *
IEF236I ALLOC. FOR PLS      ASM      SAMPLE1
IEF237I STEPLIB  ON 134
IEF237I SYSUT1   ON 130
IEF237I SYSIN    ON 00C


WORK AREA ALLOCATED (HEX BYTES)=013548
                          CORE=12K
                          XREF=3000
                        SYMTAB=1000
```

```
  PC  OBJ CODE  ADDR    ERRORS    STMT     SOURCE STATEMENT

00040                                1          ORG   O(40)
000001                               2  MDV     EQU   1
00040  000012                        3  CHK1C   DC    F(10)
00041  000027                        4  CHK2C   DC    F(23)
00042  074606                        5  CHK3C   DC    F(31110)
00043  000077                        6  CHK4C   DC    F(63)
00044  000011                        7  CHK5C   DC    F(9)
00045  000007                        8  CHK6C   DC    F(7)
00046  000000                        9  CHK7C   DC    #(A55A)
       122532
00050  003000                       10  IMPYU   DC    A(MPYU)
00051  003001                       11  IMPYA   DC    A(MPYA)
00052  003014                       12  IDIVI   DC    A(DIVI)
00053  003015                       13  IDIVU   DC    A(DIVU)
00054  001045                       14  IHMPYU  DC    A(HMPYU)
00055  001046                       15  IHMPYA  DC    A(HMPYA)
00056  001056                       16  IHDIVI  DC    A(HDIVI)
00057  001057                       17  IHDIVU  DC    A(HDIVU)
                                    18         RDONLY 0,1023
01000                               19         ORG   O(1000)
01000  020046  00046                20  CHK     LDA   0,CHK7C
01001  024047  00047                21          LDA   1,CHK7C+1
01002  030040  00040                22          LDA   2,CHK1C
01003  006053  00053                23          JSR   I  IDIVU
01004  006051  00051                24          JSR   I  IMPYA
01005  024040  00040                25          LDA    1,CHK1C
01006  030041  00041                26          LDA    2,CHK2C
01007  006050  00050                27          JSR    I  IMPYU
01010  024042  00042                28          LDA    1,CHK3C
01011  030045  00045                29          LDA    2,CHK6C
01012  006050  00050                30          JSR    I IMPYU
01013  020043  00043                31          LDA    0,CHK4C
01014  126460                       32          SUBC   1,1
01015  030044  00044                33          LDA    2,CHK5C
01016  006053  00053                34          JSR   I  IDIVU
01017  024041  00041                35          LDA    1,CHK2C
01020  030044  00044                36          LDA    2,CHK5C
01021  006052  00052                37          JSR    I  IDIVI
01022  020046  00046                38          LDA    0,CHK7C
01023  024047  00047                39          LDA    1,CHK7C+1
01024  030040  00040                40          LDA    2,CHK1C
01025  006057  00057                41          JSR   I  IHDIVU
01026  006055  00055                42          JSR   I  IHMPYA
01027  024040  00040                43          LDA    1,CHK1C
01030  030041  00041                44          LDA    2,CHK2C
01031  006054  00054                45          JSR   I  IHMPYU
01032  024042  00042                46          LDA    1,CHK3C
01033  030045  00045                47          LDA    2,CHK6C
01034  006054  00054                48          JSR   I  IHMPYU
01035  020043  00043                49          LDA    0,CHK4C
01036  126460                       50          SUBC   1,1
01037  030044  00044                51          LDA    2,CHK5C
01040  006057  00057                52          JSR   I  IHDIVU
01041  024041  00041                53          LDA    1,CHK2C
01042  030044  00044                54          LDA    2,CHK5C
01043  006056  00056                55          JSR    I  IHDIVI
```

```
   PC  OBJ CODE  ADDR    ERRORS    STMT     SOURCE STATEMENT

01044  063077                       56          HALT
01045  102460                       57  HMPYU   SUBC  0,0              CLEAR AC0
01046  061001                       58  HMPYA   DOA   0,MDV            LOAD BUFFER A
01047  066001                       59          DOB   1,MDV            LOAD BUFFER B
01050  073301                       60          DOCP  2,MDV            LOAD BUFFER C AND MULTIPLY
01051  060003                       61          NIO   3                WAIT 6.4 MIC-SECS FOR RESULT
01052  000401  01053                62          JMP   .+1
01053  060401                       63          DIA   0,MDV            PUT HIGH ORDER PART OF RESULT IN AC0
01054  065401                       64          DIB   1,MDV            PUT LOW ORDER PART OF RESULT IN AC1
01055  001400  00000                65          JMP   0,3              RETURN
01056  102400                       66  HDIVI   SUB   0,0              INTEGER DIVIDE, CLEAR HIGH PART
01057  061001                       67  HDIVU   DOA   0,MDV            LOAD BUFFER A
01060  066001                       68          DOB   1,MDV            LOAD BUFFER B
01061  073101                       69          DOCS  2,MDV            LOAD BUFFER C AND DIVIDE
01062  101010                       70          MOV#  0,0              WAIT 7.2 MIC-SECS FOR RESULT
01063  000401  01064                71          JMP   .+1
01064  060401                       72          DIA   0,MDV            PUT REMAINDER IN AC0
01065  065401                       73          DIB   1,MDV            PUT QUOTIENT IN AC1
01066  001400  00000                74          JMP   0,3              RETURN
03000                               75          ORG   0(3000)
                                    76          TROFF
03000  102460                       77  MPYU    SUBC  0,0              CLEAR AC0,DON'T DISTURB CARRY
03001  054411  03012                78  MPYA    STA   3,CB03           SAVE RETURN ADDRESS
03002  034411  03013                79          LDA   3,CB20           GET STEP COUNT
03003  125203                       80  CB99    MOVR  1,1,SNC          CHECK MULTIPLIER BIT
03004  101201                       81          MOVR  0,0,SKP          ZERO - SHIFT
03005  143220                       82          ADDZR 2,0              ONE - ADD MULTIPLICAND AND SHIFT
03006  175404                       83          INC   3,3,SZR          INCREMENT COUNTER,COMPLEMENT CARRY
03007  000774  03003                84          JMP   CB99             REPEAT PROCESS
                                    85          TRON
03010  125260                       86          MOVCR 1,1              SHIFT IN LAST LOW BIT,RESTORE CARRY
03011  002401  03012                87          JMP   I  CB03          RETURN
                                    88          TROFF
03012                               89  CB03    DS    1
03013  177760                       90  CB20    DC    F(-16)
03014  102400                       91  DIVI    SUB   0,0              INTEGER DIVIDE - CLEAR HIGH PART
03015  054416  03033                92  DIVU    STA   3,CC03           SAVE RETURN ADDRESS
03016  142432                       93          SUBZ# 2,0,SZC          OVERFLOW (AC0.GE.AC2)?
03017  000412  03031                94          JMP   CC99             EXIT IF SO
03020  034414  03034                95          LDA   3,CC20           GET STEP COUNT
03021  125120                       96          MOVZL 1,1              SHIFT DIVIDEND LOW PART
03022  101100                       97  CC98    MOVL  0,0              SHIFT DIVIDEND HIGH PART
03023  142412                       98          SUB#  2,0,SZC          DOES DIVISOR GO IN?
03024  142400                       99          SUB   2,0              YES
03025  125100                      100          MOVL  1,1              SHIFT DIVIDEND LOW PART
03026  175404                      101          INC   3,3,SZR          INCREMENT COUNTER
03027  000773  03022               102          JMP   CC98             REPEAT PROCESS
                                   103          TRON
03030  176441                      104          SUBO  3,3,SKP          FINISHED, CLEAR CARRY
03031  176420                      105  CC99    SUBZ  3,3              SET CARRY
03032  002401  03033               106          JMP   I  CC03          RETURN
03033                              107  CC03    DS    1
03034  177760                      108  CC20    DC    F(-16)
01000                              109          END   CHK
```

CROSS-REFERENCE

SYMBOL	VALUE	DEFN	REFERENCES								
CB03	003012	0089	0078	0087							
CB20	003013	0090	0079								
CB99	003003	0080	0084								
CC03	003033	0107	0092	0106							
CC20	003034	0108	0095								
CC98	003022	0097	0102								
CC99	003031	0105	0094								
CHK	001000	0020	0109								
CHK1C	000040	0003	0022	0025	0040	0043					
CHK2C	000041	0004	0026	0035	0044	0053					
CHK3C	000042	0005	0028	0046							
CHK4C	000043	0006	0031	0049							
CHK5C	000044	0007	0033	0036	0051	0054					
CHK6C	000045	0008	0029	0047							
CHK7C	000046	0009	0020	0021	0038	0039					
DIVI	003014	0091	0012								
DIVU	003015	0092	0013								
HDIVI	001056	0066	0016								
HDIVU	001057	0067	0017								
HMPYA	001046	0058	0015								
HMPYU	001045	0057	0014								
IDIVI	000052	0012	0037								
IDIVU	000053	0013	0023	0034							
IHDIVI	000056	0016	0055								
IHDIVU	000057	0017	0041	0052							
IHMPYA	000055	0015	0042								
IHMPYU	000054	0014	0045	0048							
IMPYA	000051	0011	0024								
IMPYU	000050	0010	0027	0030							
MOV	000001	0002	0058	0059	0060	0063	0064	0067	0068	0069	0072	0073
MPYA	003001	0078	0011								
MPYU	003000	0077	0010								

NO STATEMENTS FLAGGED IN THIS ASSEMBLY

NOVA SIMULATOR

NOVA TIME		AC SAME	PC	INSTRUCTION MNEMONIC			CORE ADDR	CORE BEFORE	CORE AFTER	CARRY,SOURCE AND DESTN ACS INTO AU	CARRY AND AC FROM AU	CARRY,AC FROM SHIFTER	CARRY AND ACCUMULATORS 0, 1, 2, 3 AFTER INSTRUCTION
4.8	*	0	01000	LDA		0,38	00046	000000					0 000000 000000 000000 000000
9.6		0	01001	LDA		1,39	00047	122532					0 000000 122532 000000 000000
14.4		0	01002	LDA		2,32	00040	000012					0 000000 122532 000012 000000
20.1		0	01003	JSR	1	43	03015	054416					0 000000 122532 000012 001004
474.0		0	03030	SUB0		3,3,SKP				1 000000	000000	0 000000	0 000000 010211 000012 000000
479.2		0	03032	JMP	1	*+1	01004	006051					0 000000 010211 000012 000000
484.9		0	01004	JSR	1	41	03001	054411					0 000000 010211 000012 001005
810.2		0	03010	MOVCR		1,1				1 045264	045264	1 045264 0 122532	0 000000 122532 000012 000000
815.4		0	03011	JMP	1	*+1	01005	024040					0 000000 122532 000012 000000
820.2		0	01005	LDA		1,32	00040	000012					0 000000 000012 000012 000000
825.0		0	01006	LDA		2,33	00041	000027					0 000000 000012 000027 000000
830.7		0	01007	JSR	1	40	03000	102460					0 000000 000012 000027 001010
1161.3		0	03010	MOVCR		1,1				0 000714	000714	0 000714 0 000346	0 000000 000346 000027 000000
1166.5		0	03011	JMP	1	*+1	01010	024042					0 000000 000346 000027 000000
1171.3		0	01010	LDA		1,34	00042	074606					0 000000 074606 000027 000000
1176.1		0	01011	LDA		2,37	00045	000007					0 000000 074606 000007 000000
1181.8		0	01012	JSR	1	40	03000	102460					0 000000 074606 000007 001013
1514.2		0	03010	MOVCR		1,1				0 122524	122524	0 122524 0 051252	0 000003 051252 000007 000000
1519.4		0	03011	JMP	1	*+1	01013	020043					0 000003 051252 000007 000000
1524.2		0	01013	LDA		0,35	00043	000077					0 000077 051252 000007 000000
1529.9		0	01014	SUBC		1,1				1 051252	051252	0 000000 0 000000	0 000077 000000 000007 000000
1534.7		0	01015	LDA		2,36	00044	000011					0 000077 000000 000011 000000
1540.4		0	01016	JSR	1	43	03015	054416					0 000077 000000 000011 001017
1560.3		0	03031	SUBZ		3,3				0 001017	001017	1 000000 1 000000	1 000077 000000 000011 000000
1565.5		0	03032	JMP	1	*+1	01017	024041					1 000077 000000 000011 000000
1570.3		0	01017	LDA		1,33	00041	000027					1 000077 000027 000011 000000
1575.1	*	0	01020	LDA		2,36	00044	000011					1 000077 000027 000011 000000
1580.8		0	01021	JSR	1	42	03014	102400					1 000077 000027 000011 001022
2022.9		0	03030	SUB0		3,3,SKP				1 000000	000000	0 000000 0 000000	0 000005 000002 000011 000000
2028.1		0	03032	JMP	1	*+1	01022	020046					0 000005 000002 000011 000000
2032.9		0	01022	LDA		0,38	00046	000000					0 000000 000002 000011 000000
2037.7		0	01023	LDA		1,39	00047	122532					0 000000 122532 000011 000000
2042.5		0	01024	LDA		2,32	00040	000012					0 000000 122532 000012 000000
2048.2		0	01025	JSR	1	47	01057	061001					0 000000 122532 000012 001026
2052.7		0	01057	DOA		0,MDV	BUFF A	000000	000000				0 000000 122532 000012 001026
2057.2		0	01060	DOB		1,MDV	BUFF B	000000	122532				0 000000 122532 000012 001026
2061.7		0	01061	DOCS		2,MDV	BUFF C	000000	000012				0 000000 122532 000012 001026
2067.1		0	01062	MOV#		0,0				0 000000	000000	0 000000 0 000000	0 000000 122532 000012 001026
2069.5		0	01063	JMP		*+1	01064	060401					0 000000 122532 000012 001026
2073.7		0	01064	DIA		0,MDV	BUFF A	000000	000000				0 000000 122532 000012 001026
2077.9		0	01065	DIB		1,MDV	BUFF B	010211	010211				0 000000 010211 000012 001026
2080.6		0	01066	JMP		0(3)	01026	006055					0 000000 010211 000012 001026
2086.3		0	01026	JSR	1	45	01046	061001					0 000000 010211 000012 001027
2090.8		0	01046	DOA		0,MDV	BUFF A	000000	000000				0 000000 010211 000012 001027
2095.3		0	01047	DOB		1,MDV	BUFF B	010211	010211				0 000000 010211 000012 001027
2099.8		0	01050	DOCP		2,MDV	BUFF C	000012	000012				0 000000 010211 000012 001027
2104.0		0	01051	NIO		3	NOP						0 000000 010211 000012 001027
2106.4		0	01052	JMP		*+1	01053	060401					0 000000 010211 000012 001027
2110.6		0	01053	DIA		0,MDV	BUFF A	000000	000000				0 000000 010211 000012 001027
2114.8		0	01054	DIB		1,MDV	BUFF B	122532	122532				0 000000 122532 000012 001027
2117.5		0	01055	JMP		0(3)	01027	024040					0 000000 122532 000012 001027
2122.3		0	01027	LDA		1,32	00040	000012					0 000000 000012 000012 001027
2127.1		0	01030	LDA		2,33	00041	000027					0 000000 000012 000027 001027
2132.8		0	01031	JSR	1	44	01045	102460					0 000000 000012 000027 001032

NOVA SIMULATOR

NOVA TIME	AC SAME	PC	INSTRUCTION MNEMONIC		CORE ADDR	CORE BEFORE	CORE AFTER	CARRY,SOURCE AND DESTN ACS INTO AU	CARRY AND AC FROM AU	CARRY,AC FROM SHIFTER	CARRY AND ACCUMULATORS 0, 1, 2, 3 AFTER INSTRUCTION
2138.5	*	01045	SUBC	0,0				1 000000	000000	0 000000	0 000000 000012 000027 001032
2143.0		01046	DOA	0,MDV	BUFF A	000000	000000				0 000000 000012 000027 001032
2147.5		01047	DOB	1,MDV	BUFF B	122532	000012				0 000000 000012 000027 001032
2152.0		01050	DOCP	2,MDV	BUFF C	000012	000027				0 000000 000012 000027 001032
2156.2		01051	NIO	3		NOP					0 000000 000012 000027 001032
2158.6		01052	JMP	*+1		01053	060401				0 000000 000012 000027 001032
2162.8		01053	DIA	0,MDV	BUFF A	000000	000000				0 000000 000346 000027 001032
2167.0		01054	DIB	1,MDV	BUFF B	000346	000346				0 000000 000346 000027 001032
2169.7		01055	JMP	0(3)		01032	024042				0 000000 074606 000027 001032
2174.5		01032	LDA	1,34		00042	074606				0 000000 074606 000007 001032
2179.3		01033	LDA	2,37		00045	000007				0 000000 074606 000007 001035
2185.0		01034	JSR	1 44		01045	102460				0 000000 074606 000007 001035
2190.7	*	01045	SUBC	0,0				1 000000	000000	0 000000	0 000000 074606 000007 001035
2195.2		01046	DOA	0,MDV	BUFF A	000000	000000				0 000000 074606 000007 001035
2199.7		01047	DOB	1,MDV	BUFF B	000346	074606				0 000000 074606 000007 001035
2204.2		01050	DOCP	2,MDV	BUFF C	000027	000007				0 000000 074606 000007 001035
2208.4		01051	NIO	3		NOP					0 000000 074606 000007 001035
2210.8		01052	JMP	*+1		01053	060401				0 000003 074606 000007 001035
2215.0		01053	DIA	0,MDV	BUFF A	000003	000003				0 000003 051252 000007 001035
2219.2		01054	DIB	1,MDV	BUFF B	051252	051252				0 000003 051252 000007 001035
2221.9		01055	JMP	0(3)		01035	020043				0 000077 051252 000007 001035
2226.7		01035	LDA	0,35		00043	000077				0 000077 000000 000007 001035
2232.4		01036	SUBC	1,1				1 051252	051252	0 000000	0 000000 0 000077 000000 000011 001035
2237.2		01037	LDA	2,36		00044	000011				0 000077 000000 000011 001041
2242.9		01040	JSR	1 47		01057	061001				0 000077 000000 000011 001041
2247.4		01057	DOA	0,MDV	BUFF A	000003	000077				0 000077 000000 000011 001041
2251.9		01060	DOB	1,MDV	BUFF B	051252	000000				1 000077 000000 000011 001041
2256.4		01061	DOCS	2,MDV	BUFF C	000007	000011				1 000077 000000 000011 001041
2261.8		01062	MOV#	0,0				1 000077	000077	1 000077	1 000077 000000 000011 001041
2264.2		01063	JMP	*+1		01064	060401				1 000077 000000 000011 001041
2268.4		01064	DIA	0,MDV	BUFF A	000077	000077				1 000077 000000 000011 001041
2272.6		01065	DIB	1,MDV	BUFF B	000000	000000				1 000077 000000 000011 001041
2275.3		01066	JMP	0(3)		01041	024041				1 000077 000027 000011 001041
2280.1		01041	LDA	1,33		00041	000027				1 000077 000027 000011 001041
2284.9	*	01042	LDA	2,36		00044	000011				1 000077 000027 000011 001044
2290.6		01043	JSR	1 46		01056	102400				0 000000 000027 000011 001044
2296.3		01056	SUB	0,0				1 000077	000077	0 000000	0 000000 000027 000011 001044
2300.8		01057	DOA	0,MDV	BUFF A	000077	000000				0 000000 000027 000011 001044
2305.3		01060	DOB	1,MDV	BUFF B	000000	000027				0 000000 000027 000011 001044
2309.8		01061	DOCS	2,MDV	BUFF C	000011	000011				0 000000 000027 000011 001044
2315.2		01062	MOV#	0,0				0 000000	000000	0 000000	0 000000 000027 000011 001044
2317.6		01063	JMP	*+1		01064	060401				0 000000 000027 000011 001044
2321.8		01064	DIA	0,MDV	BUFF A	000005	000005				0 000005 000027 000011 001044
2326.0		01065	DIB	1,MDV	BUFF B	000002	000002				0 000005 000002 000011 001044
2328.7		01066	JMP	0(3)		01044	063077				0 000005 000002 000011 001044

PC=01044 HALT INSTRUCTION ENCOUNTERED

END OF SIMULATION

```
IEF285I   AAELIB.LMODA                                  KEPT
IEF285I   VOL SER NOS= AAE004.
IEF285I   SYSOUT                                        SYSOUT
IEF285I   VOL SER NOS= OUTPUT.
IEF285I   SYS70257.T163503.RP001.PLS.NOVA               DELETED
IEF285I   VOL SER NOS= AAE008.
**EOS ASM       19.02.49 00031 SECS 0000
//
**EOJ PLS       19.02.51 0.01 HOURS
```

SAMPLE PROGRAM 2

```
//PLS       JOB           .0034',DR.P.L.SANGER,MSGLEVEL=1
**JST  70.257  18.54.40 00000 9.17
//SAMPLE2 EXEC NOVASM,PARM='LOAD=SIM,SNOVA,CORE=8K,XREF=800,SYMTAB=300'
XXASM    EXEC PGM=NOVASM                                                  00000010
XXSTEPLIB  DD DSN=AAELIB.LMODA,DISP=OLD                                   00000020
XXSYSPRINT DD SYSOUT=A                                                    00000030
XXSYSPUNCH DD  SYSOUT=B,DCB=(MODE=C)                                      00000040
XXSYSUT1   DD DSN=&&NOVA,SPACE=(CYL,(1,1)),UNIT=SYSDA                     00000050
//ASM.SYSIN DD *
IEF236I ALLOC. FOR PLS       ASM       SAMPLE2
IEF237I STEPLIB  ON 134
IEF237I SYSUT1   ON 130
IEF237I SYSIN    ON 00C


WORK AREA ALLOCATED (HEX BYTES)=00A1B0
                          CORE=8K
                          XREF=600
                         SYMTAB=300
```

```
  PC  OBJ CODE  ADDR    ERRORS    STMT     SOURCE STATEMENT

                                     1            PRINT HEXADECIMAL
0020                                 2            ORG   0(40)
0001                                 3  MDV       EQU   1
0020  000A                         4  CHK1C     DC    F(10)
0021  0017                         5  CHK2C     DC    F(23)
0022  7986                         6  CHK3C     DC    F(31110)
0023  003F                         7  CHK4C     DC    F(63)
0024  0009                         8  CHK5C     DC    F(9)
0025  0007                         9  CHK6C     DC    F(7)
0026  0000A55A                    10  CHK7C     DC    Z(A55A)
0028  0600                        11  IMPYU     DC    A(MPYU)
0029  0601                        12  IMPYA     DC    A(MPYA)
002A  060C                        13  IDIVI     DC    A(DIVI)
002B  060D                        14  IDIVU     DC    A(DIVU)
002C  0225                        15  IHMPYU    DC    A(HMPYU)
002D  0226                        16  IHMPYA    DC    A(HMPYA)
002E  022E                        17  IHDIVI    DC    A(HDIVI)
002F  022F                        18  IHDIVU    DC    A(HDIVU)
                                    19            RDONLY 0,1023
0200                                20            ORG   0(1000)
0200  2026        0026            21  CHK       LDA   0,CHK7C
0201  2827        0027            22            LDA   1,CHK7C+1
0202  3020        0020            23            LDA   2,CHK1C
0203  0C28        0028            24            JSR  I  IDIVU
0204  0C29        0029            25            JSR  I  IMPYA
0205  2820        0020            26            LDA   1,CHK1C
0206  3021        0021            27            LDA   2,CHK2C
0207  0C28        0028            28            JSR   I  IMPYU
0208  2822        0022            29            LDA   1,CHK3C
0209  3025        0025            30            LDA   2,CHK6C
020A  0C28        0028            31            JSR   I IMPYU
020B  2023        0023            32            LDA   0,CHK4C
020C  AD30                        33            SUBC  1,1
020D  3024        0024            34            LDA   2,CHK5C
020E  0C2B        002B            35            JSR  I  IDIVU
020F  2821        0021            36            LDA   1,CHK2C
0210  3024        0024            37            LDA   2,CHK5C
0211  0C2A        002A            38            JSR   I  IDIVI
0212  2026        0026            39            LDA   0,CHK7C
0213  2827        0027            40            LDA   1,CHK7C+1
0214  3020        0020            41            LDA   2,CHK1C
0215  0C2F        002F            42            JSR  I  IHDIVU
0216  0C2D        002D            43            JSR  I  IHMPYA
0217  2820        0020            44            LDA   1,CHK1C
0218  3021        0021            45            LDA   2,CHK2C
0219  0C2C        002C            46            JSR  I  IHMPYU
021A  2822        0022            47            LDA   1,CHK3C
021B  3025        0025            48            LDA   2,CHK6C
021C  0C2C        002C            49            JSR  I  IHMPYU
021D  2023        0023            50            LDA   0,CHK4C
021E  AD30                        51            SUBC  1,1
021F  3024        0024            52            LDA   2,CHK5C
0220  0C2F        002F            53            JSR  I  IHDIVU
0221  2821        0021            54            LDA   1,CHK2C
0222  3024        0024            55            LDA   2,CHK5C
0223  0C2E        002E            56            JSR   I  IHDIVI
```

```
  PC  OBJ CODE  ADDR     ERRORS    STMT     SOURCE STATEMENT

0224  663F                          57          HALT
0225  8530                          58  HMPYU   SUBC  0,0          CLEAR AC0
0226  6201                          59  HMPYA   DOA   0,MDV        LOAD BUFFER A
0227  6C01                          60          DOB   1,MDV        LOAD BUFFER B
0228  76C1                          61          DOCP  2,MDV        LOAD BUFFER C AND MULTIPLY
0229  6003                          62          NIO   3            WAIT 6.4 MIC-SECS FOR RESULT
022A  0101      022B                63          JMP   .+1
022B  6101                          64          DIA   0,MDV        PUT HIGH ORDER PART OF RESULT IN AC0
022C  6B01                          65          DIB   1,MDV        PUT LOW ORDER PART OF RESULT IN AC1
022D  0300      0000                66          JMP   0,3          RETURN
022E  8500                          67  HDIVI   SUB   0,0          INTEGER DIVIDE, CLEAR HIGH PART
022F  6201                          68  HDIVU   DOA   0,MDV        LOAD BUFFER A
0230  6C01                          69          DOB   1,MDV        LOAD BUFFER B
0231  7641                          70          DOCS  2,MDV        LOAD BUFFER C AND DIVIDE
0232  8208                          71          MOV#  0,0          WAIT 7.2 MIC-SECS FOR RESULT
0233  0101      0234                72          JMP   .+1
0234  6101                          73          DIA   0,MDV        PUT REMAINDER IN AC0
0235  6B01                          74          DIB   1,MDV        PUT QUOTIENT IN AC1
0236  0300      0000                75          JMP   0,3          RETURN
0600                                76          ORG   0(3000)
                                    77          TROFF
0600  8530                          78  MPYU    SUBC  0,0          CLEAR AC0,DON'T DISTURB CARRY
0601  5909      060A                79  MPYA    STA   3,CB03       SAVE RETURN ADDRESS
0602  3909      060B                80          LDA   3,CB20       GET STEP COUNT
0603  AA83                          81  CB99    MOVR  1,1,SNC      CHECK MULTIPLIER BIT
0604  8281                          82          MOVR  0,0,SKP      ZERO - SHIFT
0605  C690                          83          ADDZR 2,0          ONE - ADD MULTIPLICAND AND SHIFT
0606  FB04                          84          INC   3,3,SZR      INCREMENT COUNTER,COMPLEMENT CARRY
0607  01FC      0603                85          JMP   CB99         REPEAT PROCESS
                                    86          TRON
0608  AAB0                          87          MOVCR 1,1          SHIFT IN LAST LOW BIT,RESTORE CARRY
0609  0501      060A                88          JMP   I  CB03      RETURN
                                    89          TROFF
060A                                90  CB03    DS    1
060B  FFF0                          91  CB20    DC    F(-16)
060C  8500                          92  DIVI    SUB   0,0          INTEGER DIVIDE - CLEAR HIGH PART
060D  590E      061B                93  DIVU    STA   3,CC03       SAVE RETURN ADDRESS
060E  C51A                          94          SUBZ# 2,0,SZC      OVERFLOW (AC0.GE.AC2)?
060F  010A      0619                95          JMP   CC99         EXIT IF SO
0610  390C      061C                96          LDA   3,CC20       GET STEP COUNT
0611  AA50                          97          MOVZL 1,1          SHIFT DIVIDEND LOW PART
0612  8240                          98  CC98    MOVL  0,0          SHIFT DIVIDEND HIGH PART
0613  C50A                          99          SUB#  2,0,SZC      DOES DIVISOR GO IN?
0614  C500                         100          SUB   2,0          YES
0615  AA40                         101          MOVL  1,1          SHIFT DIVIDEND LOW PART
0616  FB04                         102          INC   3,3,SZR      INCREMENT COUNTER
0617  01FB      0612               103          JMP   CC98         REPEAT PROCESS
                                   104          TRON
0618  FD21                         105          SUBO  3,3,SKP      FINISHED, CLEAR CARRY
0619  FD10                         106  CC99    SUBZ  3,3          SET CARRY
061A  0501      061B               107          JMP   I  CC03      RETURN
061B                               108  CC03    DS    1
061C  FFF0                         109  CC20    DC    F(-16)
0200                               110          END   CHK
```

CROSS-REFERENCE

SYMBOL	VALUE	DEFN	REFERENCES									
CB03	060A	0090	0079	0088								
CB20	060B	0091	0080									
CB99	0603	0081	0085									
CC03	061B	0108	0093	0107								
CC20	061C	0109	0096									
CC98	0612	0098	0103									
CC99	0619	0106	0095									
CHK	0200	0021	0110									
CHK1C	0020	0004	0023	0026	0041	0044						
CHK2C	0021	0005	0027	0036	0045	0054						
CHK3C	0022	0006	0029	0047								
CHK4C	0023	0007	0032	0050								
CHK5C	0024	0008	0034	0037	0052	0055						
CHK6C	0025	0009	0030	0048								
CHK7C	0026	0010	0021	0022	0039	0040						
DIVI	060C	0092	0013									
DIVU	060D	0093	0014									
HDIVI	022E	0067	0017									
HDIVU	022F	0068	0018									
HMPYA	0226	0059	0016									
HMPYU	0225	0058	0015									
IDIVI	002A	0013	0038									
IDIVU	002B	0014	0024	0035								
IHDIVI	002E	0017	0056									
IHDIVU	002F	0018	0042	0053								
IHMPYA	002D	0016	0043									
IHMPYU	002C	0015	0046	0049								
IMPYA	0029	0012	0025									
IMPYU	0028	0011	0028	0031								
MDV	0001	0003	0059	0060	0061	0064	0065	0068	0069	0070	0073	0074
MPYA	0601	0079	0012									
MPYU	0600	0078	0011									

NO STATEMENTS FLAGGED IN THIS ASSEMBLY

SNOVA TIME	AC SAME	PC	INSTRUCTION MNEMONIC	CORE ADDR	CORE BEFORE	CORE AFTER	CARRY,SOURCE AND DESTN ACS INTO AU		CARRY AND AC FROM AU	CARRY,AC FROM SHIFTER	CARRY AND ACCUMULATORS 0, 1, 2, 3 AFTER INSTRUCTION
1.2	*	0200	LDA 0,38	0026	0000						0 0000 0000 0000 0000
2.4		0201	LDA 1,39	0027	A55A						0 0000 A55A 0000 0000
3.6		0202	LDA 2,32	0020	000A						0 0000 A55A 000A 0000
5.4		0203	JSR 1 43	060D	590E						0 0000 A55A 000A 0204
89.4		0618	SUB0 3,3,SKP				1 0000	0000	0 0000	0 0000	0 0000 1089 000A 0000
91.0		061A	JMP 1 *+1	0204	0C29						0 0000 1089 000A 0000
92.8		0204	JSR 1 41	0601	5909						0 0000 1089 000A 0205
160.8		0608	MOVCR 1,1				1 4AB4	4AB4	1 4AB4	0 A55A	0 0000 A55A 000A 0000
162.4		0609	JMP 1 *+1	0205	2820						0 0000 A55A 000A 0000
163.6		0205	LDA 1,32	0020	000A						0 0000 000A 000A 0000
164.8		0206	LDA 2,33	0021	0017						0 0000 000A 0017 0000
166.6		0207	JSR 1 40	0600	8530						0 0000 000A 0017 0208
235.4		0608	MOVCR 1,1				0 01CC	01CC	0 01CC	0 00E6	0 0000 00E6 0017 0000
237.0		0609	JMP 1 *+1	0208	2822						0 0000 00E6 0017 0000
238.2		0208	LDA 1,34	0022	7986						0 0000 7986 0017 0000
239.4		0209	LDA 2,37	0025	0007						0 0000 7986 0007 0000
241.2		020A	JSR 1 40	0600	8530						0 0000 7986 0007 020B
310.0		0608	MOVCR 1,1				0 A554	A554	0 A554	0 52AA	0 0003 52AA 0007 0000
311.6		0609	JMP 1 *+1	020B	2023						0 0003 52AA 0007 0000
312.8		020B	LDA 0,35	0023	003F						0 003F 52AA 0007 0000
313.4		020C	SUBC 1,1				1 52AA	52AA	0 0000	0 0000	0 003F 0000 0007 0000
314.6		020D	LDA 2,36	0024	0009						0 003F 0000 0009 0000
316.4		020E	JSR 1 43	060D	590E						0 003F 0000 0009 020F
320.4		0619	SUBZ 3,3				0 020F	020F	1 0000	1 0000	1 003F 0000 0009 0000
322.0		061A	JMP 1 *+1	020F	2821						1 003F 0000 0009 0000
323.2		020F	LDA 1,33	0021	0017						1 003F 0017 0009 0000
324.4	*	0210	LDA 2,36	0024	0009						1 003F 0017 0009 0000
326.2		0211	JSR 1 42	060C	8500						1 003F 0017 0009 0212
411.0		0618	SUB0 3,3,SKP				1 0000	0000	0 0000	0 0000	0 0005 0002 0009 0000
412.6		061A	JMP 1 *+1	0212	2026						0 0005 0002 0009 0000
413.8		0212	LDA 0,38	0026	0000						0 0000 0002 0009 0000
415.0		0213	LDA 1,39	0027	A55A						0 0000 A55A 0009 0000
416.2		0214	LDA 2,32	0020	000A						0 0000 A55A 000A 0000
418.0		0215	JSR 1 47	022F	6201						0 0000 A55A 000A 0216
421.2		022F	DOA 0,MDV	NOP							0 0000 A55A 000A 0216
424.4		0230	DOB 1,MDV	NOP							0 0000 A55A 000A 0216
431.2		0231	DIV								0 0000 1089 000A 0216
431.8		0232	MOV# 0,0				0 0000	0000	0 0000	0 0000	0 0000 1089 000A 0216
432.4		0233	JMP *+1	0234	6101						0 0000 1089 000A 0216
435.2		0234	DIA 0,MDV	NOP							0 0000 1089 000A 0216
438.0		0235	DIB 1,MDV	NOP							0 0000 1089 000A 0216
438.6		0236	JMP 0(3)	0216	0C2D						0 0000 1089 000A 0216
440.4		0216	JSR 1 45	0226	6201						0 0000 1089 000A 0217
443.6		0226	DOA 0,MDV	NOP							0 0000 1089 000A 0217
446.8		0227	DOB 1,MDV	NOP							0 0000 1089 000A 0217
452.1		0228	MUL								0 0000 A55A 000A 0217
455.3		0229	NIO 3	NOP							0 0000 A55A 000A 0217
455.9		022A	JMP *+1	022B	6101						0 0000 A55A 000A 0217
458.7		022B	DIA 0,MDV	NOP							0 0000 A55A 000A 0217
461.5		022C	DIB 1,MDV	NOP							0 0000 A55A 000A 0217
462.1		022D	JMP 0(3)	0217	2820						0 0000 A55A 000A 0217
463.3		0217	LDA 1,32	0020	000A						0 0000 000A 000A 0217
464.5		0218	LDA 2,33	0021	0017						0 0000 000A 0017 0217
466.3		0219	JSR 1 44	0225	8530						0 0000 000A 0017 021A

SUPERNOVA SIMULATOR

SNOVA TIME		AC SAME	PC	INSTRUCTION MNEMONIC		CORE ADDR	CORE BEFORE	CORE AFTER	CARRY,SOURCE AND DESTN ACS INTO AU			CARRY AND AC FROM AU	CARRY,AC FROM SHIFTER	CARRY AND ACCUMULATORS 0, 1, 2, 3 AFTER INSTRUCTION			
466.9	*		0225	SUBC	0,0				1	0000	0000	0 0000	0 0000	0	0000	000A	0017	021A
470.1			0226	DOA	0,MDV	NOP								0	0000	000A	0017	021A
473.3			0227	DOB	1,MDV	NOP								0	0000	000A	0017	021A
478.6			0228	MUL										0	0000	00E6	0017	021A
481.8			0229	NIO	3	NOP								0	0000	00E6	0017	021A
482.4			022A	JMP	*+1	022B	6101							0	0000	00E6	0017	021A
485.2			022B	DIA	0,MDV	NOP								0	0000	00E6	0017	021A
488.0			022C	DIB	1,MDV	NOP								0	0000	00E6	0017	021A
488.6			022D	JMP	0(3)	021A	2822							0	0000	00E6	0017	021A
489.8			021A	LDA	1,34	0022	7986							0	0000	7986	0017	021A
491.0			021B	LDA	2,37	0025	0007							0	0000	7986	0007	021A
492.8			021C	JSR	1 44	0225	8530							0	0000	7986	0007	021D
493.4	*		0225	SUBC	0,0				1	0000	0000	0 0000	0 0000	0	0000	7986	0007	021D
496.6			0226	DOA	0,MDV	NOP								0	0000	7986	0007	021D
499.8			0227	DOB	1,MDV	NOP								0	0000	7986	0007	021D
505.1			0228	MUL										0	0003	52AA	0007	021D
508.3			0229	NIO	3	NOP								0	0003	52AA	0007	021D
508.9			022A	JMP	*+1	022B	6101							0	0003	52AA	0007	021D
511.7			022B	DIA	0,MDV	NOP								0	0003	52AA	0007	021D
514.5			022C	DIB	1,MDV	NOP								0	0003	52AA	0007	021D
515.1			022D	JMP	0(3)	021D	2023							0	003F	52AA	0007	021D
516.3			021D	LDA	0,35	0023	003F							0	003F	0000	0007	021D
516.9			021E	SUBC	1,1				1	52AA	52AA	0 0000	0 0000	0	003F	0000	0009	021D
518.1			021F	LDA	2,36	0024	0009							0	003F	0000	0009	0221
519.9			0220	JSR	1 47	022F	6201							0	003F	0000	0009	0221
523.1			022F	DOA	0,MDV	NOP								0	003F	0000	0009	0221
526.3			0230	DOB	1,MDV	NOP								1	003F	0000	0009	0221
527.8			0231	DIV										1	003F	0000	0009	0221
528.4			0232	MOV#	0,0				1	003F	003F	1 003F	1 003F	1	003F	0000	0009	0221
529.0			0233	JMP	*+1	0234	6101							1	003F	0000	0009	0221
531.8			0234	DIA	0,MDV	NOP								1	003F	0000	0009	0221
534.6			0235	DIB	1,MDV	NOP								1	003F	0000	0009	0221
535.2			0236	JMP	0(3)	0221	2821							1	003F	0000	0009	0221
536.4			0221	LDA	1,33	0021	0017							1	003F	0017	0009	0221
537.6	*		0222	LDA	2,36	0024	0009							1	003F	0017	0009	0221
539.4			0223	JSR	1 46	022E	8500							1	003F	0017	0009	0224
540.0			022E	SUB	0,0				1	003F	003F	0 0000	0 0000	0	0000	0017	0009	0224
543.2			022F	DOA	0,MDV	NOP								0	0000	0017	0009	0224
546.4			0230	DOB	1,MDV	NOP								0	0000	0017	0009	0224
553.2			0231	DIV										0	0005	0002	0009	0224
553.8			0232	MOV#	0,0				0	0005	0005	0 0005	0 0005	0	0005	0002	0009	0224
554.4			0233	JMP	*+1	0234	6101							0	0005	0002	0009	0224
557.2			0234	DIA	0,MDV	NOP								0	0005	0002	0009	0224
560.0			0235	DIB	1,MDV	NOP								0	0005	0002	0009	0224
560.6			0236	JMP	0(3)	0224	663F							0	0005	0002	0009	0224

PC=0224 HALT INSTRUCTION ENCOUNTERED

END OF SIMULATION

```
IEF285I   AAELIB.LMODA                                    KEPT
IEF285I   VOL SER NOS= AAE004.
IEF285I   SYSOUT                                          SYSOUT
IEF285I   VOL SER NOS= OUTPUT.
IEF285I   SYS70257.T163503.RP001.PLS.NOVA                 DELETED
IEF285I   VOL SER NOS= AAE008.
**EOS ASM       18.55.12 00031 SECS 0000
//
**EOJ PLS       18.55.13 0.01 HOURS
```