THE UNIVERSITY OF MANITOBA

SIMULATION OF THE

UNIVAC 1108

A Thesis

Presented to

The Faculty of Graduate Studies and Research

The University of Manitoba

In partial fulfillment

of the Requirements for the Degree

Master of Science

in Computer Science

---

by

Robert Macmillan

Winnipeg, Manitoba

October, 1971

ABSTRACT

A simulation of Univac 1108 hardware is written for the IBM 360 Model 65. The simulation includes an 1108 Assembler, Loader and machine code Interpreter.

The objective of the project is to design and develop a system that will assist in the education of Computer Science students.

ACKNOWLEDGEMENTS

I would like to thank the members of the examining committee, Professors A.Macpherson and P. King, for their time spent in reading and suggesting improvements to the thesis.

My thanks also to my wife Carol who typed the work with the assistance of the University of Manitoba Monitor.

A special acknowledgement is made to Dr. C. Abraham, my advisor, who helped considerably in both the development of the simulator, and the writing of the thesis.

TABLE OF CONTENTS.

TABLE OF CONTENTS

LIST OF TABLES.

LIST OF FIGURES.

Chapter I INTRODUCTION.

Computer Science departments of small or medium size universities are usually unable to purchase more than one large scale computer system. Other departments within the school may have special purpose machines for specific applications, but the student of Computer Science rarely has access to these. His experience is limited to one computer and hence his views are significantly influenced by that hardware. How can he be properly trained to make comparisons and evaluations of different computer designs?

Many factors play a role in the selection of computing equipment. For a Computer Science Department this selection can be influenced, not only by student requirements, but also by the needs of other departments, University administration and possibly the business community of the city. In fact it is likely that student requirements are of secondary importance as computers are still expensive items and are usually expected to provide some financial return to the University. The Computer Science Department must develop systems which provide for effective student use of the computer. Also the opportunity for comparison with other equipment should be available.

For example, a student working with an IBM 360/65 may observe inefficient use of core storage. Core fragmentation can render areas unavailable to jobs which would otherwise execute. A solution could be found in improved job scheduling. Other computer designs overcome the problem by

dynamic relocation of addresses or paging. Simulation is an inexpensive method of illustrating such design differences.

The simulator enables students to compile, load and execute programs written in Univac 1108 Basic Assembler Language. Generally the user is unaware that his program is being executed by an IBM 360/65.

The objectives of developing the 1108 simulator are -

(a) to make the student of Computer Science aware of different computer designs.

(b) to familiarize him specifically with the Univac 1108.

Appendix V gives a general description of the 1108.

1108 programs may be written in executive (supervisor) and/or user modes. Using the system gives considerable insight into the functioning of the hardware and the Executive. A suggested student exercise is to compare one specific area of the 1108 with the 360, such as character string manipulation. He can compare ease of programming, number of instructions, main storage requirements etc. The current simulator does not time execution but this could be included to give a more complete comparison.

This presentation includes three sections which describe the simulator. It is suggested the user omit Chapter III and concern himself with the User Guide (Chapter II) and Sample Programs (Chapter IV). 1108 Manuals may be needed to supplement this description (see References). Chapter III is

directed to the Systems Programmer, for purposes of program maintenance and additions. Conclusions and future developments are discussed in Chapter V.

Chapter II USER GUIDE

This chapter contains several sections which describe the simulated Univac 1108 in detail. Special attention is given to the differences between the simulator and the actual 1108.

Differences have been introduced for two reasons.

(1) A different approach enables characteristics of the hardware to be better illustrated.

For example, main storage modules in the simulator are smaller than in the 1108. By reducing their size, it is possible to have multiple storage modules and thus completely represent 1108 addressing.

(2) Some variations help surmount programming problems in representing hardware operations.

One of the major problems in developing the simulator was in representing a hardware interrupt. In the 1108 an interrupt may occur at any point in time. An instruction which executes in more than one machine cycle may be interrupted and re-started later. Programming this into the simulator would have been an impossible task. Hence the simulator handles interrupts a little differently. An instruction cannot be interrupted during execution. Therefore any interrupt that occurs will be handled after the instruction in progress is completed.

Section 2.1 describes the 1108 hardware that is simulated. Examples are provided to clarify the operation of the 1108.

The 1108 instructions implemented in the simulator are

presented in Section 2.2. This section is included for two reasons.

(1) The User Guide is intended as a complete description of the simulated 1108, and is to be used in learning to program the simulator, and for reference. Appendix III provides an alphabetic list of all 1108 instructions. Those that are included in the simulator are cross referenced to Section 2.2.

(2) Each instruction is presented in a notation which is clearer and more concise than that found in any of the manufacturers manuals. Once familiar with the notation (see Appendix III) the student will find this method of presentation facilitates both learning and reference.

Sections 2.3, 2.4 and 2.5 describe the three programs that form the simulator. Of particular interest are the program debugging aids that are available (Section 2.5). None of these is provided in the Univac 1108.

In preparing job control language the user is aware for the first time that his program is executed by an IBM/360. To simplify this process all JCL usually required is listed in Section 2.6.

Errors detected by the simulated Assembler are flagged as in the 1108 Assembler. Unlike the 1108 Assembler, however, the position of the error is marked in the simulated output. Some errors may produce IBM error codes instead of 1108 messages. The reasons for this and a complete list of errors are provided in Section 2.7.

SECTION 2.1 SIMULATED 1108 SYSTEM

Many configurations of individual 1108 components are possible. For the purposes of this simulation a very simple configuration is used. Like the computer itself, the simulator is modular, such that future expansion is possible. Chapter V discusses planned additions that will help to improve and sophisticate the simulator.

The simulated system has a unit-processor (consisting of control, arithmetic and input/output sections), two main storage modules, a card reader and line printer.

1. MAIN STORAGE. [5]

Each 1108 main storage module contains 32,768 36-bit words. The simulator has two modules so the user can familiarize himself with alternate interrupt addresses (see later this section). The modules are reduced in size to 4096 words. Thus valid main storage addresses are:-

0_8 to 17777_8 (0 to 4095)

100000_8 to 107777_8 (32768 to 36863)

Note that addresses 20000_8 to 77777_8 are invalid.

There are 128 control registers whose addresses are 0 to

177_8, i.e. the same addresses as the first 128 words of main storage. Most addresses less than 200_8 refer to control registers and hence the name "Hidden Storage" - the first 128 words of main storage. An address refers to hidden storage rather than a control register when

(a) the address is obtained from the P-register (Program Address Counter) which contains the address of the next sequential instruction (NI).

(b) the address is that of a word referenced during an indirect addressing sequence.

(c) the address is that of a word referenced by the Execute instruction.

Certain locations in main storage are used permanently as interrupt handling routine entry points and status words. The fixed address assignments used in the simulator are listed in Table 2.1.1.

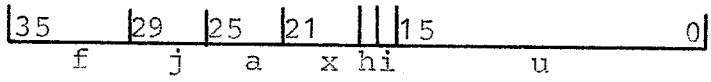
The value x is obtained from the Memory Select Register (MSR). It determines the storage module used. With two storage modules the only allowable values of C(MSR) are 0 or 1. For example, if C(MSR)=1 and a Guard Mode Interrupt occurs the system will trap to absolute location 100243_8. Initially C(MSR)=0. The MSR can be loaded with the SIL (Select Interrupt Locations) instruction.

2. CONTROL SECTION.[6]

The Central Processing Unit (CPU) follows these steps in

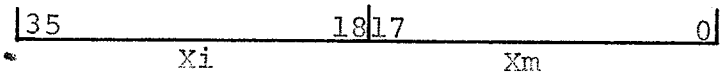
Octal Address	Assignment
x00200	CPU external interrupt status word
x00220	ISI input monitor interrupt
x00221	ISI output monitor interrupt
x00222	ISI function monitor interrupt
x00223	ISI external interrupt
x00241	Illegal instruction fault interrupt
x00242	Executive return interrupt
x00243	Guard mode/Storage protection fault interrupt
x00244	Test and set interrupt
x00247	Divide fault interrupt
x00252	Implementation interrupt

Table 2.1.1. Fixed Address Assignments.



f = function code.
j = operand qualifier, minor function code.
a = control register specification.
x = index register.
h = index register incrementation indicator.
i = indirect addressing bit.
u = operand address.

Figure 2.1.1. Instruction Word Format.



Xi and Xm both halfword signed integers.

Figure 2.1.2. Index Register Format.

j	QW	Main Storage	to	Arithmetic Section
0	0,1	35 0		35 0
1	0,1	//////////17 0		zeros 17 0
2	0,1	35 18//////////		zeros 17 0
3	0,1	//////////17 0		signs 17 0
4	0	35 18//////////		signs 17 0
5	0	////////////11 0		signs 11 0
6	0	////////23 12////////		signs 11 0
7	0	35 24//////////////		signs 11 0
4	1	//////26 18//////////		zeros 8 0
5	1	////////////8 0		zeros 8 0
6	1	//////////17 9//////		zeros 8 0
7	1	35 27//////////////		zeros 8 0
10	0,1	////////////////5 0		zeros 5 0
11	0,1	////////////11 6////		zeros 5 0
12	0,1	//////////17 12//////		zeros 5 0
13	0,1	////////2318//////////		zeros 5 0
14	0,1	////29 24//////////////		zeros 5 0
15	0,1	3530//////////////////		zeros 5 0
16	0,1	//////////17 0		zeros 17 0
17	0,1	//////////17 0		signs 17 0

Table 2.1.2a. Data Transfer Patterns.
Main Storage to Arithmetic Section

j	QW	Arithmetic Section	to	Main Storage
0	0,1	35 0		35 0
1	0,1	////////////17 0		////////////17 0
2	0,1	////////////17 0		35 18////////////
3	0,1	////////////17 0		////////////17 0
4	0	////////////17 0		35 18////////////
5	0	////////////////11 0		////////////////11 0
6	0	////////////////11 0		/////////23 12/////////
7	0	////////////////11 0		35 24////////////////
4	1	/////////////////8 0		//////26 18////////////
5	1	/////////////////8 0		/////////////////8 0
6	1	/////////////////8 0		////////////17 9///////
7	1	/////////////////8 0		35 27//////////////////
10	0,1	////////////////////5 0		////////////////////5 0
11	0,1	////////////////////5 0		/////////////////11 6////
12	0,1	////////////////////5 0		////////////17 12///////
13	0,1	////////////////////5 0		////////2318//////////
14	0,1	////////////////////5 0		///29 24//////////////
15	0,1	////////////////////5 0		3530//////////////////////
16	0,1	NO		TRANSFER
17	0,1	NO		TRANSFER

Table 2.1.2b. Data Transfer Patterns.
Arithmetic Section to Main Storage

the execution of a program:

(a) An instruction is transferred from main storage to the control section of the CPU. The address of this instruction is obtained from the P-register, which is initially set by the Loader and incremented by one until a program jump or CPU interrupt breaks the sequence.

(b) The instruction is then decoded and the instruction effective address computed.

(c) The operation specified by the instruction is then performed.

An 1108 instruction occupies one 36-bit word.

The instruction word format is illustrated in Figure 2.1.1

The following paragraphs discuss the most usual use of each subfield of the instruction. Variations to this are found in the definition of each instruction (Section 2.2).

f-field

The f-field defines completely ($0 < f \leq 70_8$)

partially ($f > 70_8$)

the operation to be performed by the CPU.

For $f > 70_8$, the f and j fields define the operation.

j-field.

If $f < 70_8$, the j-field is used as a operand qualifier. For j such that $0_8 \leq j \leq 3_8$ or $10_8 \leq j \leq 17_8$ the value determines a specific pattern for data transfer from main

storage to the control section or vice-versa. For j such that $4_8 \leq j \leq 7_8$ two different patterns are possible. The Quarter Word Indicator (QW) in the Processor State Register (PSR) determines which pattern is used. Tables 2.1.2a and 2.1.2b give the data transfer patterns for values of the j-field and QW.

When the effective main storage address is less than 200_8 the value of j is assumed equal to zero.

If $f=70_8$, the j-field and a-field together specify a control register ($0_8 \leq j,a \leq 177_8$).

If $f > 70_8$, the j-field is a minor function code. i.e. f and j together define the operation.

a-field

The a-field selects either an accumulator (A), index register (X) or control register (R). If the instruction uses two or three accumulators then the a-field explicitly references Aa and implicitly references Aa+1 and Aa+2.

Table 2.1.3 shows there are two sets of accumulators, index registers and R-registers. D6 (in the PSR) determines which set is used.

If D6=0 then A values are 12 - 27
X " " 0 - 15
R " " 64 - 79

If D6=1 then A values are 108- 123
X " " 97 - 111
R " " 80 - 95

The latter set are called Executive registers.

OCTAL		DECIMAL
0	Processor State Register (temp. storage)	0
1		1
	Fifteen Index Registers (X-Registers)	
13		11
14		12
	(overlap)	
17		15
20		16
	Sixteen Accumulators (A-Registers)	
33		27
34		28
	(not used)	
77		63
100	Real-Time Clock (R0 Register)	64
101	Repeat Count Register (R1 Register)	65
102	Mask Register (R2 Register)	66
103		67
	Thirteen Special Registers (R-Registers)	
117		79
120		80
	Sixteen Executive Special Registers (same assignments as R-Registers)	
137		95
140	Non-Indexing Register (X0 Register)	96
141		97
	Fifteen Executive Index Registers	
153		107
154		108
	(overlap)	
157		111
160		112
	Sixteen Executive Accumulators	
173		123
174		124
	(not used)	
177		127

Table 2.1.3. Control Register Assignments.

x-field

If x=0 then no index register is employed in effective address calculation.

If x≠0, then it selects an index register. The low 18-bits of this register contain a signed number which is used in computing the effective address (see Effective Address Calculation below).

h-field

If x=0, the h-field is not used.

If x≠0, h=0 then the index register remains unchanged during the effective address calculation.

If x≠0, h=1 then the index register is incremented after it has been used for calculating effective address. The incrementation process is defined as

C(Xi) + C(Xm) = C(Xm). (see Figure 2.1.2)

i-field.

If i=0, the value plays no significant role in the execution of the instruction.

For i non-zero, its meaning depends on the Base Register Suppression bit (D7) of the PSR.

If i=1, D7=0 then indirect addressing is specified (see Effective Address Calculation below).

If i=1, D7=1 then the address as calculated is an absolute address and the base register contents are not added. Use of this is best seen by illustration in Sample Problem 2 Chapter IV.

u-field.

The u-field is normally used to address an operand. The 16 bits form an unsigned positive integer which is added into the effective address calculation.

Exceptions are shift instructions and the Executive Return instruction.

PROGRAM ADDRESS COUNTER (P-register)

The P-register is an 18-bit register which contains the absolute address of the next instruction to be executed. Immediately after an instruction is fetched from main storage the value in the P-register is incremented by one. A jump or skip instruction may further alter the contents of the P-register. A jump to E is effected by E = C(P).

When an interrupt occurs control is passed to a fixed location in main storage. The contents of the P-register is not changed. The first instruction of the interrupt handler must save the P-register. Usually a SLJ (store location and jump) or LMJ (load modifier and jump) instruction is executed at the trap location.

Subroutine call sequences:

```
(1)      .              . mainline program
         .
         SLJ   ,RTN . C(P) = C(RTN), jump to RTN+1
         .
         .
      RTN  0
         .
```

```
          .
          J     *RTN . return to mainline

(2)       .          . mainline program
          .
          LMJ   X6, RTN . C(P) = C(X6), jump to RTN
          .
          .
     RTN  0   . the routine must not destroy register 6
          .
          .
          J     ,X6 . return to mainline
```

EFFECTIVE ADDRESS CALCULATION

The following is the sequence used in effective address calculation. Instruction address obtained initially from P-register.

1. resultl = u
2. if x = 0 continue at 6.
3. resultl = resultl + Xx modifier (indexing)
4. if h = 0 continue at 6.
5. C(Xi) + C(Xm) = C(Xm) (index modification)
6. if shift instruction and i = 0 continue at 15.
7. if f $< 70_8$, j = 16_8, 17_8, and x = 0 then continue at 9. (using 18-bit u value).
8. if D7 = 1 and i = 1 then continue at 15. (absolute addressing)

9. if resultl > BS then continue at 12.

10. resultl = resultl + BI (instruction base added)

11. continue at 13

12. resultl = resultl + BD (data base added)

13. if f $< 70_8$, j = 16 , 17 , and x = 0 then continue at 15.

14. if i = 1 then use resultl to address another word and continue at 1. (indirect addressing)

15. E = resultl (final effective address).

The features of the addressing system of the 1108 are

(a) indirect addressing to any level.

(b) index register modification.

(c) absolute addressing option.

(d) base registers enabling dynamic relocation of programs.

(e) 2 base registers (one for instructions, one for data) with separate storage limits checking.

3. ARITHMETIC SECTION [7]

1108 Adder

The 1108 Adder is Ones complement subtractive adder that is used for all fixed point arithmetic operations. It operates with either 36-bit or 72-bit operands.

Fixed point addition and subtraction

Certain instructions can result in an overflow or carry condition. These instructions initialize the overflow and carry indicators (D1 and D0 in PSR) to zero. The indicators may then be set 1 by the arithmetic.

The overflow bit is set 1 when the sign of the result is algebraically inconsistant.

viz.	Augend	Addend	Result
Addition	+	+	-
	-	-	+
	Minuend	Subtrahend	
Add Negative	+	-	-
(Subtraction)	-	+	+

The carry bit is set 1 when an end-around carry occurs during the execution of the instruction.

viz.	Augend	Addend	Result
	+	-	+
Addition	-	+	+
	-	-	+
	-	-	-
	Minuend	Subtrahend	
	+	+	+
Add Negative	-	-	+
(Subtraction)	-	+	+
	-	+	-

Fixed-point multiplication.

There are three fixed point multiply instructions.

(a) Multiply Integer gives a 72-bit result stored in two consecutive accumulators.

(b) Multiply Single Integer stores only the 36 rightmost bits of the product. The high order word is lost.

(c) Multiply Fractional is the same as Multiply Integer until the result is stored. For this instruction the 72-bit product is left shifted circularly one bit position prior to storing in two consecutive accumulators.

For the two multiply integer instructions, if each operand has an implied binary point to the right of the rightmost bit then the product has an implied binary point to the right of the rightmost bit.

For the multiply fractional instruction, if each operand has an implied binary point to the left of the leftmost magnitude bit (i.e. between bits 35 and 34) then the product has an implied binary point to the left of the leftmost magnitude bit.

Scaling operands can be done either up or down. For example to multiply

2.75 by 3.625

i.e. 10.11_2 by 11.101_2

we can proceed in either of the following ways

$(1011)2^{-2} \times (11101)2^{-3}$	$(0.1011)2^{2} \times (0.11101)2^{2}$
$prod=(1011 \times 11101)2^{-5}$	$prod=(0.1011 \times 0.11101)2^{4}$
using mult int	using mult frac
$= (100111111)2^{-5}$	$= (0.100111111)2^{4}$
$= 1001.11111_2$	$= 1001.11111_2$

= 9.96875 = 9.96875

Fixed point division

There are three fixed point divide instructions.

(a) Divide Integer gives a 36-bit quotient and a 36-bit remainder stored in two consecutive accumulators, replacing the original 72-bit dividend.

(b) Divide Fractional left shifts logically the dividend one bit position prior to the divide. A 36-bit quotient and a 36-bit remainder are stored in consecutive accumulators, replacing the original 72-bit dividend.

(c) Divide Single Fractional left shifts logically the dividend one bit position prior to the divide. The original dividend is only 36 bits long. A 36-bit quotient is stored in an accumulator and the remainder is lost.

For the divide integer instruction if both the dividend and divisor have implied binary points to the right of the rightmost bit then the quotient and remainder also have implied binary points to the right of the rightmost bit.

For the divide fractional instructions if both dividend and divisor have implied binary points to the left of the leftmost magnitude bit (i.e. between bits 35 and 34) then the quotient and remainder both have implied binary points to the left of the leftmost magnitude bit.

There is no equivalent in the IBM/360 repertoire to the multiply and divide fractional instructions.

4. THE EXECUTIVE [3]

The 1108 is designed to operate under the control of an executive program whose function includes loading and controlling user programs, handling interrupts and I/O.

The simulated system automatically loads instructions to handle interrupts. For any interrupt a core dump is given and the user program terminated. The user can, however, overwrite those instructions and effectively write his own executive program (see Sample Problem 2, Chapter IV). When doing so consideration must be given the following:

PROCESSOR STATE REGISTER (PSR).

D-field

D8 (bit 35) not used.

D7 (bit 34) - Base Register Suppression Indicator.

When D7 = 1 the i bit of the instruction is used to specify absolute addressing rather than indirect addressing. When computing the effective address the contents of the base registers are not used if i=1.

Consider -

L A2,*01000

location 01000_8 contains 00001_8

location 11000_8 contains 01001_8

location 11001_8 contains 01002_8

Base register contents = 10000_8

If D7 = 0 when the load instruction is executed 01002_8 is loaded into A2. The effective address is computed by:-

(a) base register + u = 11000_8

(b) indirect addressing - get C(11000_8)=1001_8

(c) add base, E=11001_8

If D7=1 when the load instruction is executed, 001 is loaded into A2. The i-bit now indicates absolute addressing so E = 01000_8.

D6 (bit 33) - Control Register Selection Designator.

When D6 = 1, the a and x fields of the instruction word reference the Executive control registers (see Table 2.1.3).

These registers have addresses -

141_8 - 157_8 (index registers)

154_8 - 173_8 (accumulators)

120_8 - 137_8 (R-registers)

When D6 = 0, the a and x fields refer to control registers with addresses

1_8 - 17_8 (index registers)

14_8 - 33_8 (accumulators)

100_8 - 117_8 (R-registers)

D5 (bit 32) - not used.

D4 (bit 31) - not used

D3 (bit 30)/D2(bit 29) - Modified storage protection/Guard mode and storage limit protection.

if D3D2 = 00_2 then guard mode and storage protection are disabled. No guard mode/storage protection fault interrupt can occur.

If D3D2 = 01_2 then guard mode and storage limits protection are fully enabled. A GMSL fault

interrupt occurs if an attempt is made to write into any control register other than user registers, if a priveleged instruction is attempted or if the effective address violates storage limits (from SLR)

If D3D2 = 10_2 then guard mode is disabled but storage limits protection is enabled for writes. i.e. if the effective address violates the storage limits for a write instruction a Storage Limits Fault interrupt occurs.

If D3D2 = 11_2 then guard mode is enabled and storage limits protection is enabled for writes.

D1 (bit 28) - Overflow Designator.

Certain fixed point arithmetic instructions affect D1. These instructions always clear D1 initially and then set D1 = 1 if overflow occurs during the arithmetic.

D0 (bit 27) - Carry Designator.

The same instructions that affect D1 may set D0. Initially D0 = 0 then if end around carry occurs during the arithmetic operation D0 = 1.

BI - Instruction Bank Base Address Register.

BI converts a relative address, of a word that is in the instruction bank of the program, to an absolute address. The value of the base address is formed by placing 9 zero bits to the right of the 9-bit BI field.

QW - Quarter Word Designator.

For some instructions the value QW is used to determine

the data transfer pattern. (see Table 2.1.2)

BS - BI/BD Selection Register.

BS determines whether a relative address references a word in the Instruction bank or the Data bank of the program. Nine zero bits are attached to the right of the 7-bit BS field to form a compare value. If the computed relative address is less than or equal to this compare value, BI is used as base to form the absolute address. Otherwise BD is used to compute the absolute address.

BD - Data Bank Base Address Register.

BD converts a relative address to an absolute address if the word referenced lies in the Data Bank of the program. The process is as for BI.

The PSR is loaded with the Load Processor State instruction. When an interrupt occurs the PSR is automatically stored in control register zero. A new PSR is loaded before the interrupt handler is given control. Only the D-field and QW of the new PSR differ from the old PSR. D7 and D6 are set to 1, and D8, D5 through D0 and QW are set to zero.

STORAGE LIMITS PROTECTION

A user program can be divided into two sections - viz. the instruction bank and the data bank. Each bank is based on a separate register contained in the PSR and each has its own storage limits. When the instruction bank register (BI)

is used to compute the address the instruction bank limits are used to check the validity of the address.

Format of the SLR is given in Figure 2.1.4.

The actual value of each limit is obtained by placing 9 zero bits to the right of each value in the SLR.

If BI is used to compute E then IU ≥ E ≥ IL

If BD is used to compute E then DU ≥ E ≥ DL

If E does not satisfy the appropriate condition and storage protection is enabled (see PSR) then a storage limits protection interrupt occurs.

5. INPUT/OUTPUT SECTION

The design for the I/O section is discussed in Appendix IV. Initially a temporary section consisting of two built-in macro instructions is provided for reading cards and printing lines.

Two instructions directly affect the 1108's preparedness to accept an I/O interrupt. One allows and one prevents interrupts. As the simulator at present does not include the simulation of an 1108 interrupt, these instructions can have no effect.

They are included in the simulator for two reasons:-

(1) I/O interrupt simulation is planned for the future and is discussed in the design of the I/O section (Appendix IV).

(2) Both of the instructions transfer control to the computed effective address and are therefore not "No operation" instructions.

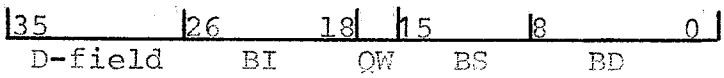


Figure 2.1.3. PSR Format.

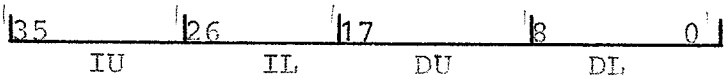


Figure 2.1.4. SLR Format.

2.2 1108 INSTRUCTION SET.[10]

The instruction set is discussed in detail. Instructions are presented in logical groups. Appendix III provides an alphabetic list which cross refers to this section. Factors common to each member of the group are listed first and then individual instructions described.

For an explanation of notation see Appendix II.

LOAD INSTRUCTIONS.

All single precision load instructions

a) transfer data under j control from main storage to the arithmetic section.

b) transfer the fullword to the specified control register.

1. Load Accumulator (L or LA; 10)

C(E)j = C(Aa)

If $j=16_8$, 17_8, x=0, h=i=1, $u=177777_8$ then +0=C(Aa)

2. Load Negative Accumulator (LN or LNA; 11)

-C(E)j = C(Aa)

If $j=16_8$, 17_8, x=h=i=u=0 then -0=C(Aa)

3. Load Magnetude Accumulator (LM or LMA; 12)

   |C(E)j| = C(Aa)

4. Load Negative Magnetude Accumulator (LNMA; 13)

   -|C(E)j| = C(Aa)

   If $j=16_8$, 17_8 , x=h=i=u=0 then -0=C(Aa)

5. Load Register (L or LR; 23)

   C(E)j = C(Ra)

   If Guard Mode is enabled (i.e. D2 of PSR set) then an attempt to load R0 causes a Guard Mode Fault Interrupt.

6. Load Index Modifier (LXM; 26)

   $C(E)j = C(Xa)_{17-0}$; $C(Xa)_{35-18}$ unchanged.

7. Load Index Register (L or LX; 27)

   C(E)j = C(Xa)

8. Load Index Increment (LXI; 46)

   $C(E)j = C(Xa)_{35-18}$; $C(Xa)_{17-0}$ unchanged.

9. Double Load Accumulator (DL; 71,13)

   C(E,E+1) = C(Aa,Aa+1)

10. Double Load Negative Accumulator (DLN; 71,14)

    -C(E,E+1) = C(Aa,Aa+1)

STORE INSTRUCTIONS.

All single precision store instructions

a) transfer a fullword from the specified control register to the arithmetic section.

b) transfer data under j control from the arithmetic section to main storage.

except when

a) $j=16_8$, 17_8 when no data is stored.

b) $E < 200_8$ when only a fullword can be transferred.

1. Store Accumulator (S or SA; 01)

   C(Aa)j = C(E)

2. Store Negative Accumulator (SN or SNA; 02)

   -C(Aa)j = C(E)

3. Store Magnetude Accumulator (SM or SMA; 03)

   |C(Aa)|j = C(E)

4. Store Register (S or SR; 04)

   C(Ra)j = C(E)

5. Store Zero (SZ; 05)

   +0j = C(E)

6. Store Index Register (S or SX; 06)

   C(Xa)j = C(E)

7. Double Store Accumulator (DS; 71,12)

   C(Aa,Aa+1) = C(E,E+1)

8. Block Transfer (BT; 22)

   C(E)j = C(Ea), C(R2) - 1 = C(R2); repeat K times.

   where Ea is effective address computed using Xa instead of Xx.

   If K=0 no data is transferred.

   If $j=16_8$, 17_8 no data is transferred.

   If x=0 no data is transferred and C(Xa) remain unchanged regardless of h.

   If i=0, h is usually non-zero.

   If h=0 no index register modification is performed and hence source and destination addresses remain the same for all K data transfers.

   If the destination address specifies a control register a fullword is always moved from the arithmetic section to the register.

FIXED-POINT ARITHMETIC INSTRUCTIONS.

1. Add to Accumulator(A or AA; 14)

   C(Aa) + C(E)j = C(Aa)

   The overflow (D1) and carry (D0) indicators are cleared at initiation and may be set during this instruction.

   If $j=16_8$, 17_8 , x=0, h=i=1, $u=177777_8$ then

C(Aa)+0=C(Aa)

2. Add Negative to Accumulator (AN or ANA; 15)

C(Aa) - C(E)j = C(Aa)

D1 and/or D0 may be set by this instruction.

If $j=16_8$, 17_8 , x=0, h=i=1, $u=177777_8$ then

C(Aa)-(+0)=C(Aa)

3. Add Magnitude to Accumulator (AM or AMA; 16)

C(Aa) + |C(E)j| = C(Aa)

D1 and/or D0 may be set by this instruction.

If $j=16_8$,17_8 , x=0, h=i=1, $u=177777_8$ then

C(Aa)+0=C(Aa)

4. Add Negative Magnitude to Accumulator (ANM or ANMA; 17)

C(Aa) - |C(E)j| = C(Aa)

D1 and/or D0 may be set by this instruction.

If $j=16_8$, 17_8 , x=0, h=i=1, $u=177777_8$ then

C(Aa)-(+0)=C(Aa)

5. Add Upper (AU; 20)

C(Aa) + C(E)j = C(Aa+1)

D1 and/or D0 may be set by this instruction.

If $j=16_8$, 17_8 , x=0, h=i=1, $u=177777_8$ then

C(Aa)+0=C(Aa+1)

6. Add Negative Upper (ANU; 21)

C(Aa) - C(E)j = C(Aa+1)

D1 and/or D0 may be set by this instruction.

If $j=16_8$, 17_8 , x=0, h=i=1, $u=177777_8$ then

C(Aa)-(+0)=C(Aa+1)

7. Add to Index Register (A or AX; 24)

C(Xa) + C(E)j = C(Xa)

D1 and/or D0 may be set by this instruction.

If $j=16_8$, 17_8 , x=0, h=i=1, $u=177777_8$ then

C(Xa)+0=C(Xa)

8. Add Negative to Index Register (AN or ANX; 25)

C(Xa) - C(E)j = C(Xa)

D1 and/or D0 may be set by this instruction.

If $j=16_8$, 17_8 , x=0, h=i=1, $u=177777_8$ then

C(Xa)-(+0)=C(Xa)

9. Multiply Integer (MI; 30)

C(Aa) . C(E)j = C(Aa,Aa+1)

If $j=16_8$, 17_8 , x=0, h=i=1, $u=177777_8$ then

C(Aa).(+0)=C(Aa,Aa+1)

10. Multiply Single Integer (MSI; 31)

C(Aa) . C(E)j = C(Aa)

If $j=16_8$, 17_8 , x=0, h=i=1, $u=177777_8$ then

C(Aa).(+0)=C(Aa)

11. Multiply Fractional (MF; 32)

C(Aa) . C(E)j = C(Aa,Aa+1)

Prior to storing the 72-bit result it is shifted left, circularly, one bit position.

If $j=16_8$, 17_8 , x=0, h=i=1, $u=177777_8$ then

C(Aa).(+0)=C(Aa,Aa+1)

12. Divide Integer (DI; 34)

C(Aa,Aa+1) ÷ C(E)j = C(Aa)

remainder = C(Aa+1)

C(Aa,Aa+1) is left shifted, circularly, one bit position before the division is performed.

The remainder retains the sign of the dividend.

If x=a+n, where n=12 or 13 and h=1 then the value of the dividend is undefined because of the overlap of the index register being incremented and the accumulator.

If $j=16_8$, 17_8 , x=0, h=i=1, $u=177777_8$ then

C(Aa,Aa+1)÷(+0)=C(Aa)

A Divide Fault Interrupt occurs if C(E)j=0

or |C(Aa,Aa+1)| ≥ |C(E)j| . 2e35

13. Divide Single Fractional (DF; 35)

C(Aa) ÷ C(E)j = C(Aa+1)

The remainder is lost.

If $j=16_8$, 17_8 , x=0, h=i=1, $u=177777_8$ then

C(Aa)÷(+0)=C(Aa+1)

A Divide Fault Interrupt occurs if C(E)j=0

or |C(Aa)| ≥ |C(E)j|

14. Divide Fractional (DF; 36)

C(Aa,Aa+1) ÷ C(E)j = C(Aa)

remainder = C(Aa+1)

The remainder retains the sign of the dividend.

If $x=15_8$ and a=0

or $x=16_8$ and a=1

or $x=17_8$ and a=2

and h=1

then the value of the dividend is undefined because of the overlap of the index register being incremented and the accumulator.

If $j=16_8$, 17_8 , x=0, h=i=1, $u=177777_8$ then C(Aa,Aa+1)÷(+0)=C(Aa)

A Divide Fault Interrupt occurs if C(E)j=0

or |C(Aa)| ≥ |C(E)j|

15. Double Precision Fixed Point Add (DA; 71,10)

C(Aa,Aa+1) + C(E,E+1) = C(Aa,Aa+1)

D1 and/or D0 may be set by the instruction.

16. Double Precision Fixed Point Add Negative (DAN; 71,11)

C(Aa,Aa+1) - C(E,E+1) = C(Aa,Aa+1)

D1 and/or D0 may be set by this instruction.

17. Add Halves. (AH; 62,04)

$C(Aa)_{35-18} + C(E)_{35-18} = C(Aa)_{35-18}$

$C(Aa)_{17-0} + C(E)_{17-0} = C(Aa)_{17-0}$

Bits 35 and 17 are sign bits. A carry from bit 35

is propogated to bit 18. Similarly a carry from bit 17 is propogated to bit 0.

Overflow and carry indicators remain unaltered.

18. Add Negative Halves (ANH; 72,05)

$$C(Aa)_{35-18} - C(E)_{35-18} = C(Aa)_{35-18}$$
$$C(Aa)_{17-0} - C(E)_{17-0} = C(Aa)_{17-0}$$

19. Add Thirds (AT; 72,06)

$$C(Aa)_{35-24} + C(E)_{35-24} = C(Aa)_{35-24}$$
$$C(Aa)_{23-12} + C(E)_{23-12} = C(Aa)_{23-12}$$
$$C(Aa)_{11-0} + C(E)_{11-0} = C(Aa)_{11-0}$$

Carries from bits 11, 23, 35 are propogated to bits 0, 12, 24 respectively.

20. Add Negative Thirds (ANT; 72,07)

$$C(Aa)_{35-24} - C(E)_{35-24} = C(Aa)_{35-24}$$
$$C(Aa)_{23-12} - C(E)_{23-12} = C(Aa)_{23-12}$$
$$C(Aa)_{11-0} - C(E)_{11-0} = C(Aa)_{11-0}$$

SEARCH INSTRUCTIONS.

All search instructions consist of three stages of execution, viz.

a) the initial stage

- R_1 is transferred to the index subsection.
- contents of specified accumulators are

transferred to the arithmetic section.

b) repeated test stages. During each test stage the following steps are executed

- the effective address E is computed.
- the contents of E are transferred under j control to the arithmetic section.
- the repeat counter is decremented by one.
- the specified test is made.
- if the condition is satisfied the termination stage is entered immediately.
- else the repeat counter is tested for zero. If non-zero a new test stage is initiated, otherwise the termination stage is entered.

c) the termination stage.

- if entered as a result of a successful test the contents of the P-register is incremented by one, effecting the skip of an instruction.
- the repeat counter is stored in R1.

Note: Unless either h=1 or i=1 i.e. index incrementation or indirect addressing is specified, the effective address computed at each test stage is the same.

Note: The simulator differs in the handling of interrupts from the 1108 and as such the search instructions **cannot be interrupted (see introduction this chapter). For a description of programming considerations for the 1108 see Univac manual [1]**

1. Search Equal (SE; 62)

If C(E)j = C(Aa) then skip NI else repeat up to K times.

If $j=16_8$, 17_8, x=0, h=i=1, $u=177777_8$ then (if C(Aa)=+0 then skip NI, else repeat up to K times)

Note that +0≠-0.

2. Search Not Equal (SNE; 63)

If C(E)j ≠C(Aa) then skip NI, else repeat up to K times.

If $j=16_8$, 17_8, x=0, h=i=1, $u=177777_8$ then (if C(Aa)=+0 then skip NI, else repeat up to K times)

Note that +0≠-0

3. Search Less than or Equal - Search Not Greater (SLE or SNG; 63)

If C(E)j ≤ C(Aa) then skip NI, else repeat up to K times

If $j=16_8$, 17_8, x=0, h=i=1, $u=177777_8$ then (if +0 ≤ C(Aa) then skip NI, else repeat up to K times)

Note that +0> -0

4. Search Greater (SG; 65)

If C(E)j > C(Aa) then skip NI, else repeat up to K times.

If $j=16_8$, 17_8, x=0, h=i=1, $u=177777_8$ then (if +0 > C(Aa) then skip NI, else repeat up to K times).

Note that +0 > -0

5. Search Within Range (SW; 66)

If C(Aa) < C(E)j ≤ C(Aa+1) then skip NI, else repeat up to K times.

If $j=16_8$, 17_8, x=0, h=i=1, $u=177777_8$ then (if C(Aa) < +0 ≤ C(Aa+1) then skip NI, else repeat up to K times)

If C(Aa) ≥ C(Aa+1) there is no value that can satisfy the conditions of the test.

Note that +0 > -0.

If h=1 and x=a+n, where n=12 or 13 then one of the values transferred to the arithmetic section from control registers is undefined.

6. Search Not Within Range (SNW; 67)

If C(E)j ≤ C(Aa) or C(E)j > C(Aa+1) then skip NI, else repeat up to K times.

If $j=16_8$, 17_8, x=0, h=i=1, $u=177777_8$ then (if +0 ≤ C(Aa) or +0> C(Aa+1) then skip NI, else repeat up to K times).

If C(Aa) ≥ C(Aa+1) then all values satisfy the conditions of the test.

Note that +0 > -0.

MASKED SEARCH INSTRUCTIONS.

This group of instructions is similar to the multistage search instructions. The three stages of execution are

a) the initial stage

- as for search instructions except that the contents of R2 (Mask Register) are also transferred to the arithmetic section.

b) repeated test stages. Each test stage is the same as the search instruction test stage except that a fullword is always transferred from main storage to the aritmetic section.

c) termination stage - as for search instructions.

1. Masked Search Equal (MSE; 71,00)

   If [C(E) AND C(R2)] = [C(Aa) AND C(R2)] then skip NI, else repeat up to K times.

   Note: $+0 \neq -0$

2. Masked Search Not Equal (MSNE; 71,01)

   If [C(E) AND C(R2)] $\neq$ [C(Aa) AND C(R2)] then skip NI, else repeat up to K times.

   Note: $+0 \neq -0$

3. Masked Search Less Than or Equal, Masked Search Not Greater (MSLE or MSNG; 71,02)

   If [C(E) AND C(R2)] $\leq$ [C(E) AND C(R2)] then skip NI, else repeat up to K times.

   Note $+0 > -0$

4. Masked Search Greater (MSG; 71,03)

   If [C(E) AND C(R2)] $>$ [C(E) AND C(R2)] then skip NI, else

repeat up to K times.

Note: +0>-0.

5. Masked Search Within Range (MSW; 71,04)

If [C(Aa) AND C(R2)] < [C(E) AND C(R2)] ≤ [C(Aa+1) AND C(R2)] then skip NI, else repeat up to K times.

Note: +0>-0.

If [C(Aa) AND C(R2)] ≥ [C(A_{a+1}) AND C(R2)] then no value can satisfy the conditions of the test.

6. Masked Search Not Within Range (MSNW; 71,05)

If [C(E) AND C(R2)] ≤ [C(Aa) AND C(R2)] or [C(E) AND C(R2)] > [C(Aa+1) AND C(R2)] then skip NI, else repeat up to K times.

Note: +0>-0.

If [C(Aa) AND C(R2)] ≥ [C(Aa+1) AND C(R2)] then all values satisfy the conditions of the test.

7. Masked Alphanumeric Search Less than or Equal (MASL; 71,06)

If [C(E) AND C(R2)] ≤ [C(Aa) AND C(R2)] then skip NI, else repeat K times.

Note: -0 >+0.

8. Masked Alphanumeric Search Greater (MASG; 71,07)

If [C(E) AND C(R2)] > [C(Aa) AND C(R2)] then skip NI else

repeat K times.

Note: -0 ≠+0.

TEST INSTRUCTIONS

1. Test Even Parity (TEP; 44)

   If [C(E)j AND C(Aa)] has even parity then skip NI. The logical product is not saved.

   If $j=16_8$, 17_8, x=0, h=i=1, $u=177777_8$ then (if [+0 AND C(Aa)] has even parity then skip NI).

   If h=1 and a+12=x then the value transferred to the arithmetic section from the control register is undefined.

2. Test Odd Parity (TOP; 45)

   If [C(E)j AND C(Aa)] has odd parity then skip NI. The logical product is not saved.

   If $j=16_8$, 17_8, x=0, h=i=1, $u=177777_8$ then (if [+0 AND C(Aa)] has odd parity then skip NI).

   If h=1 and a+12=x then the value transferred to the arithmetic section from the control register is undefined.

3. Test Less Than or Equal to Modifier, Test Not Greater Than Modifier (TLEM or TNGM; 47)

If $C(E)j_{17-0} \leq C(Xa)_{17-0}$ then skip NI

$C(Xa)_{17-0} + C(Xa)_{35-18} = C(Xa)_{17-0}$

Note: $-0 < +0$.

If $j=16_8$, 17_8, $x=0$, $h=i=1$, $u=177777_8$ then +0 is transferred to the arithmetic section from main storage.

If h=1 and a=x the specified index register is incremented once only.

4. Test Zero (TZ; 50)

If C(E)j = 0 then skip NI.

If $j=16_8$, 17_8, $x=0$, $h=i=1$, $u=177777_8$ then skip NI.

5. Test Non-Zero (TNZ, 51)

If $C(E)j \neq 0$ then skip NI.

If $j=16_8$, 17_8, $x=0$, $h=i=1$, $u=177777_8$ then the NI is executed.

6. Test Equal (TE; 52)

If C(E)j = C(Aa) then skip NI.

Note: $+0 = -0$

If $j=16_8$, 17_8, $x=0$, $h=i=1$, $u=177777_8$ then (if +0=C(Aa) then skip NI).

7. Test Not Equal (TNE; 53)

If $C(E)j \neq C(Aa)$ then skip NI.

Note: $+0 \neq -0$

If $j=16_8$, 17_8, $x=0$, $h=i=1$, $u=177777_8$ then (if $+0 \neq C(Aa)$ then skip NI)

8. Test Less Than or Equal, Test Not Greater (TLE, TNG; 54)

If $C(E)j \leq C(Aa)$ then skip NI.

Note: $+0 > -0$

If $j=16_8$, 17_8, $x=0$, $h=i=1$, $u=177777_8$ then (if $+0 \leq C(Aa)$ then skip NI).

9. Test Greater (TG; 55)

If $C(E)j > C(Aa)$ then skip NI.

Note: $+0 > -0$.

If $j=16_8$, 17_8, $x=0$, $h=i=1$, $u=177777_8$ then (if $+0 > C(Aa)$ then skip NI).

10. Test Within Range (TW; 56)

If $C(Aa) < C(E)j \leq C(Aa+1)$ then skip NI.

Note: $+0 > -0$.

If $C(Aa) \geq C(Aa+1)$ there is no value that can satisfy the conditions of the test.

If $j=16_8$, 17_8, $x=0$, $h=i=1$, $u=177777_8$ then (if $C(Aa) < +0 \leq C(Aa+1)$ then skip NI).

If $h=1$ and $x=a+n$, where $n=12$ or 13 then one of the values transferred to the arithmetic section from the control registers is undefined.

11. Test Not Within Range (TNW; 57)

If C(e)j $\leq$ C(Aa) or C(E)j $>$ C(Aa+1) then skip NI

Note: +0 $>$ -0

If C(Aa) $\geq$ C(Aa+1) then all values satisfy at least one of the conditions of the test.

If $j=16_8$, 17_8, x=0, h=i=1, $u=177777_8$ then (if +0 $\leq$ C(Aa) or 0 $>$ C(Aa+1) then skip NI).

If h=1 and x=a+n, where n=12 or 13 then one of the values transferred to the arithmetic section from the control registers is undefined.

12. Test Positive (TP; 60)

If C(E)j $\geq$ +0 then skip NI.

If $j=16_8$, 17_8, x=0, h=i=1, $u=177777_8$ then the NI is skipped.

13. Test Negative (TN; 61)

If C(E)j $\leq$ -0 then skip NI

If $j=16_8$, 17_8, x=0, h=i=1, $u=177777_8$ then execute NI.

14. Double Precesion Test Equal (DTE; 71,17)

If C(E,E+1) = C(Aa,Aa+1) then skip NI.

Note: +0 $\neq$ -0.

SHIFT INSTRUCTIONS.

The input operands for all shift instructions, except the two load shift and count instructions, are located in one or two accumulators.

The effective address, E, is computed normally and then the value of E is used as the shift count. The shift count can lie between 0 and 72.

Shift types are

- Right circular; bits shifted out of the right end of the register appear at the left end.
- Left circular; bits shifted out of the left end of the register appear at the right end.
- Right logical; zeros fill vacated bits at the left end.
- Left logical; zeros fill vacated bits at the right end.
- Right algebraic; sign bits fill vacated bits at the left end.

1. Single Shift Circular (SSC; 73,00)

Shift C(Aa) right circularly E bit positions.

If $36 \leq E \leq 72$ then a shift of E-36 bit positions is made.

2. Double Shift Circular (DSC; 73,01)

Shift C(Aa,Aa+1) right circularly E bit positions.

If h=1 and x=a+n, where n=12 or 13 then one of the

values transferred to the arithmetic section from the control registers is undefined.

3. Single Shift Logical (SSL; 73,02)

Shift C(Aa) right logically E bit positions.

If $36 \leq E \leq 72$ then the resulting content of Aa is +0.

4. Double Shift Logical (DSL; 73,03)

Shift C(Aa,Aa+1) right logically E bit positions.

If h=1 and x=a+n, where n=12,13 then one of the values transferred to the arithmetic section from the control registers is undefined.

5. Single Shift Algebraic (SSA; 73,04)

Shift C(Aa) algebraically E bit positions.

If $35 \leq E \leq 72$ then all bits of the resulting Aa are equal to the original sign bit.

6. Double Shift Algebraic (DSA; 73,05)

Shift C(Aa,Aa+1) algebraically E bit positions.

If h=1 and x=a+n, where n=12,13 then one of the values transferred to the arithmetic section is undefined.

7. Load Shift and Count (LSC; 73,06)

C(E) = C(Aa); shift C(Aa) left circularly until $C(Aa)_{35} \neq C(Aa)_{34}$; number of bit positions shifted = C(Aa+1).

If C(E) = 0, then C(E)=C(Aa) and 43_8=C(Aa+1).

8. Double Load Shift and Count (DLSC; 73,07)

C(E,E+1) = C(Aa,Aa+1); shift C(Aa,Aa+1) left circularly until $C(Aa,Aa+1)_{71} \neq C(Aa,Aa+1)_{70}$; number of bit positions shifted = C(Aa+2).

If C(E,E+1)=0, the C(E,E+1)= C(Aa,Aa+1) and 107_8 = C(Aa+2).

9. Left Single Shift Circular (LSSC; 73,10)

Shift C(Aa) left circularly E bit positions.

If $36 \leq E \leq 72$ then a shift of E-36 bit positions is made.

10. Left Double Shift Circular (LDSC; 73,11)

Shift C(Aa,Aa+1) left circularly E bit positions.

If h=1 and x=a+n where n=12 or 13 then one of the values transferred to the arithmetic section is undefined.

11. Left Single Shift Logical (LSSL; 73,12)

Shift C(Aa) left logically E bit positions.

If $36 \leq E \leq 72$ resulting C(Aa)=+0.

12. Left Double Shift Logical (LDSL; 73,13)

Shift C(Aa, Aa+1) left logically E bit positions.

If h=1, and x=a+n where n=12,13 then one of the values transferred to the arithmetic section is undefined.

UNCONDITIONAL JUMP INSTRUCTIONS.

1. Store Location and Jump (SLJ; 72,01)

   C(P) - bm = $C(E)_{17-0}$; jump to E+1

   where bm=base address modifier (BI or BD).

   The jump is effected by E+1 = C(P).

   The contents of the P-register (absolute address of NI) is first converted to a relative address by subtracting the base address modifier (in the PSR).

   If E < 200_8 then C(P)-bm = $C(E)_{17-0}$ (refers to a control register); +0=$C(E)_{35-18}$; jump to E+1.

   If the "jump to" address is less than 200_8 then the NI is taken from hidden storage rather than a control register.

2. Load Modifier and Jump (LMJ; 74,13)

   C(P) - bm = $C(Xa)_{17-0}$; jump to E.

   where bm=base address modifier (BI or BD)

   The jump is effected by E+1= C(P).

   The contents of the P-register (absolute address of NI) is first converted to a relative address by subtracting the base address modifier (in the PSR).

   If E <200_8 then the NI is taken from hidden storage rather than a control register.

3. Allow All I/O Interrupts and Jump (AAIJ; 74,07)

Allow all I/O interrupts and jump to E.

This instruction is used to allow I/O interrupts following the handling of an interrupt or following the execution of a Prevent All I/O Interrupts and Jump instruction.

For description of the interrupt handling of the simulator see Section 2.1.

CONDITIONAL JUMP INSTRUCTIONS.

1. Jump Greater and Decrement (JGD; 70)

If C(CRja)> 0 then jump to E.

C(CRja)-1 = C(CRja)

The control register is specified in the right most seven bits of the ja field of the instruction word.

If $40_8 \leq ja \leq 100_8$ or $120_8 \leq ja \leq 177_8$ and D2=1 (in PSR) then a Guard Mode Fault Interrupt occurs.

2. Double Precision Jump Zero (DJZ; 71,16)

If C(Aa,Aa+1)=0 then jump to E.

If h=1 and x=a+n, where n=12 or 13 then one of the values transferred to the arithmetic section from the control registers is undefined.

3. Jump Positive and Shift (JPS; 72,02)

   If C(Aa) $\geq$ +0 then jump to E.

   Always shift C(Aa) left, circularly, one bit position.

4. Jump Negative and Shift (JNS; 72,04)

   If C(Aa) $\leq$ -0 then jump to E.

   Always shift C(Aa) left, circularly, one bit position.

5. Jump Zero (JZ; 74,00)

   If C(Aa) = 0 then jump to E.

6. Jump Nonzero (JNZ; 74,01)

   If C(Aa) $\neq$ 0 then jump to E.

7. Jump Positive (JP; 74,02)

   If C(Aa) $\geq$ +0 then jump to E.

8. Jump Negative (JN; 74,03)

   If C(Aa) $\leq$ -0 then jump to E.

9. Jump, Jump Keys (J,JK; 74,04)

   If a=0 or if SELECT JUMPS(a) = on then jump to E.

   There are 15 SELECT JUMPS indicators in the 1108. These may be set at program load time in the simulator.

10. Halt Jump, Halt Keys and Jump (HJ, HKJ; 74,05)

    If a=0 or (a AND SELECT STOPS indicators)=0 stop; on

restart or continuation jump to E.

The 4 SELECT STOPS indicators may be set at program load time.

If Guard Mode in enabled when a halt condition is satisfied, the condition is ignored and the CPU proceeds immediately with the jump.

11. Jump No Low Bit (JNB; 74,10)

If $C(Aa)_0 = 0$ then jump to E.

Note: Positive even integers have no low bit but negative even integers do.

12. Jump Low Bit (JB; 74,11)

If $C(Aa)_0 = 1$ then jump to E.

Note: Positive even integers have no low bit but negative even integers do.

13. Jump Modifier Greater and Increment (JMGI; 74,12)

If $C(Xa)_{17-0} > 0$ then jump to E.

Always $C(Xa)_{17-0} + C(Xa)_{35-18} = C(Xa)_{17-0}$

Note: the test is made before the addition.

If h=1 and a=x then the specified index register is effectively modified only once.

14. Jump Overflow (JO; 74,14)

If D1 = 1 then jump to E. (D1 is found in the PSR).

The contents of D1 remain unaltered.

15. Jump No Overflow (JNO; 74,15)

If D1 = 0 then jump to E.

The contents of D1 remain unaltered.

16. Jump Carry (JC; 74,16)

If D0 = 1 then jump to E. (D0 is found in the PSR)

The contents of D0 remain unaltered.

17. Jump No Carry (JNC; 74,17)

If D0 = 0 then jump to E.

The contents of D0 remain unaltered.

BOOLEAN INSTRUCTIONS.

1. Logical OR (OR; 40)

C(Aa) OR C(E)j = C(Aa+1)

2. Logical Exclusive OR (XOR; 41)

C(Aa) XOR C(E)j = C(Aa+1)

3. Logical AND (AND; 42)

C(Aa) AND C(E)j = C(Aa+1)

4. Masked Load Upper (MLU; 43)

(C(E)j AND C(R2)) OR (C(Aa) AND C(R2)) = C(Aa+1)

MISCELLANEOUS INSTRUCTIONS.

1. Execute (EX; 72,10)

   Execute the instruction at E.

   If E<200_8 the remote instruction is fetched from hidden storage rather than from a control register.

   Execute instructions may be cascaded to any level.

2. Executive Return (ER; 72,11)

   Interrupt to location x000242_8

   where x=C(MSR)

   The interrupt handler stores the current PSR in control register 0 before the instruction at 242_8 is executed.

   If Guard Mode is enabled (D3D2=01_2) and indirect addressing is specified (i=1,D7=0) and E violates the main storage limits set in the SLR then a Guard Mode/Storage Limits Fault Interrupt occurs.

3. Test and Set (TS; 73,17)

   If $C(E)_{30}$ = 1 then interrupt to location x000244_8, where x=C(MSR).

   Always 01_8 = $C(E)_{35-30}$; $C(E)_{29-0}$ unchanged.

   If E < 200_8 the result of the instruction is undefined.

4. No Operation (NOP; 74,06)

No operation performed.

If x=0 and h=1 index register incrementation will take place.

EXECUTIVE SYSTEM CONTROL INSTRUCTIONS.

All the instructions listed here are privileged instructions. If D2=1 (in PSR) when an attempt is made to execute one of these instructions, a Guard Mode Fault Interrupt occurs.

1. Prevent All I/O Interrupts and Jump (PAIJ; 72,13)

Prevent all I/O interrupts and jump to E.

After the execution of this instruction an Allow All I/O Interrupts and Jump instructions must be executed in order that the CPU will react to any I/O interrupt.

2. Load Processor State (LPS; 72,15)

C(E) = C(PSR)

When programming the 1108 to load the PSR consideration must be given to the timing cycle of the machine [2]. The simulator delays the loading of the PSR for one full instruction. That is, if the following

sequence is used:-

```
        LPS   Ei      (1)
        AAIJ  Ej      (2)
        .
        .
        .
Ej      L     A2,Ek   (3)
```

The old PSR remains in effect until after the execution of instruction (2). Before execution of instruction (3) the new PSR is loaded from location Ei.

When handing control to a User program the final operations of the Executive is normally to load the user PSR and pass control to the user with a sequence similar to that above.

3. Load Storage Limits (LSL; 72,16)

C(E) = C(SLR)

This instruction does not enable storage protection. It redefines the storage protection limits. Enabling and disabling storage protection is under the control of the PSR.

4. Select Interrupt Locations (SIL; 73,15)

a_{2-0} = C(MSR)

Only the three low order bits of the a field are

used. For the simulator the only valid contents for the MSR are 000_2 or 001_2. All fixed address references use the MSR to determine the absolute address (xFFFFF) where x=C(MSR) and FFFFF = fixed address.

INPUT/OUTPUT INSTRUCTIONS.

The design of the simulator's input/output handler is found in Appendix IV. The first version of the system uses a temporary subsection which consists of two assembler macro instructions. Only card reader input and line printer output is permitted. Logically a card contains 72 characters. EBCDIC characters are converted to 1108 code on input. An output line contains 132 characters the first of which specifies carriage control. 1108 code is converted to EBCDIC on output.

1. Get Card (GET; 33)

72 characters from card = C(E)....C(E+11).

Columns 1-72 of the card are converted to 1108 code (6-bit ASCII) which is then packed 6 characters per 1108 word. Storage limits check is done for E only and the user is responsible for seeing that E+i, where i=1,2...11 does not violate storage limits.

2. Put line (PUT; 37)

C(E)....C(E+20) = 132 characters on line printer.

1108 character code is converted to EBCDIC. Six characters are extracted from each 1108 word. The first character of the first word specifies carriage control.

01_8 - write and no space

05_8 - write and space 1 line

11_8 - write and space 2 lines

13_8 - write and space 3 lines

77_8 - skip to channel 1 then write and space 1 line.

E is checked for storage limits violation. No such check is made for E+i, where i=1,2...20

3. Dump core (no mnemonic; 0)

Dump core from location zero to high core. Terminate job.

For a further discussion of debugging aids see Section 2.1

The Assembler reads 1108 Basic Assembler Language statements and produces relocatable code for input to the Loader. Separate assemblies are possible, see Sections 2.4: The Loader, and 2.6: Job Control Language. Free form source statements are coded in EBCDIC using card columns 1-72.

There are three source statement types:

(1) A symbolic representation of a machine instruction.

(2) Data item definition.

(3) An Assembler directive which may or may not produce executable code.

Omitted from the simulated Assembler are :

(1) the macro processor.

(2) label arrays.

(3) some of the built-in directives.

These are features which make the Assembler language programmer's job easier but do not provide any additional machine instructions. They were considered to be beyond the scope of this initial project. Future expansion to the system could include the addition of these features.

All symbolic machine instructions are translated although some are not implemented in the Interpreter.

Assembler Instructions

General form

```
          $(e),label   f,j   a,u,x
or        $(e),label   f     a,u,x,j
```

Where $(e) is a location counter specification, label is a statement tag, f is the function code, j is the partial word designator, a is the accumulator.

u is the main storage address

x is the index register

Some exceptions to this general form are the special mnemonic J (unconditional jump), and the simulator's I/O macros GET and PUT. Each of these instructions has form f u,x.

Example: J LOC rather than J ,LOC

There are 32 location counters available for programmer use (0 through 31). Location counter 0 is assumed in control unless another is specified. Once specified a location counter remains in control until another is encountered. Programs written in logical sections may be regrouped for execution. When specified, the $(e) must commence in column 1.

A relative main storage address is assigned to each location counter using the RES directive (see later this section).

The current location counter may be referenced in the operand field of an instruction by using the symbol $.

A statement label is optional and if present must start either

(a) immediately after "$(e),"

(b) in column 1.

Labels must be no longer than six alphanumeric characters of which the first is alphabetic. If a label is to be used outside the program it must be externalized - by coding an

asterisk immediately after the mnemonic.

The operation field of an instruction contains a symbolic representation of the f or f,j subfields. For instructions where f < 70, if j is not coded it is assumed zero. The j-field may be coded in two ways - see General Form above. The j value can be a decimal constant, an octal constant, a user defined mnemonic or a built-in mnemonic. Table 2.3.1 lists all built-in mnemonics.

The f-field starts with the first non-blank character after the label. It is terminated by a blank except when "f,j" is coded when "f,nj" (where n is any number of blanks) is allowable.

The operand field starts with the first non-blank character after the operation field. Subfields are separated by commas. If one is omitted either ",," or ",0," is coded. If the last subfield is omitted a comma is not required to terminate the previous one.

The a and x subfields may be coded as decimal constants, octal constants, user defined mnemonics or built-in mnemonics.

If the u subfield is immediately preceded by an asterisk the i bit of the machine instruction is set to provide indirect addressing.

If the x-subfield is immediately preceded by an asterisk

INDEX REGISTERS		ARITHMETIC REGISTERS		R REGISTERS		PARTIAL WORD DESIGNATORS	
Mnem	Abs	Mnem	Abs	Mnem	Abs	Mnem	Abs
X0	0	A0	12	R1	65	W	0
X1	1	A1	13	R2	66	H2	1
X2	2	A2	14	R3	67	H1	2
X3	3	A3	15	R4	68	XH2	3
X4	4	A4	16	R5	69	XH1	4
X5	5	A5	17	R6	70	T3	5
X6	6	A6	18	R7	71	T2	6
X7	7	A7	19	R8	72	T1	7
X8	8	A8	20	R9	73	S1	15
X9	9	A9	21	R10	74	S2	14
X10	10	A10	22	R11	75	S3	13
X11	11	A11	23	R12	76	S4	12
		A12	24	R13	77	S5	11
		A13	25	R14	78	S6	10
		A14	26	R15	79	Q1	7
		A15	27			Q2	4
						Q3	6
						Q4	5
						U	16
						XU	17

Table 2.3.1. Control Register Absolute Addresses and Assembler Built-in Mnemonics.

then **the h bit of the** machine instruction is set to provide index register modification.

Comments may follow each source statement. " . ", terminates the assembler scan. An entire card may be used for comments if " . " precedes any other data.

Statements may be continued on two or more cards by coding ";". All data after the ";" are ignored and the assembler scan starts again in column 1 of the next card. Neither mnemonics nor constants may be split across two input records. Comments may be continued for any number of cards.

The Assembler skips the output listing to a new page if "/" is coded in column 1 of an input record. A label or location counter then starts in column 2.

Examples.

```
(1)   LA,6     A2,LOC,3
(2)   LA       A2,LOC,X3,6

(3)   LA,T2    A2,LOC,X3

(4)   L,T2     A2,LOC,X3

(5)   L,T2     14,LOC,X3

(6)   L,T2     016,LOC,X3

(7)   L,T2     016,01023,X3 .  COMMENT
```

If LOC is relative main storage address 1023_8 then each statement (1) through (7) generates the same object code with

$f = 10_8$

$j = 6_8$

a = 2

u = 1023_8

x = 3

h = 0

i = 0

(8) AN,H2 A6,*ABLE,*X5

This statement generates a machine instruction with

f = 15_8

j = 1

a = 6

u = relative address of ABLE

x = 5

h = 1 - specifies index register modification

i = 1 - specifies indirect addressing

(9) $(3),SIX L A3, $+3

Location counter 3 is in control for this and all subsequent statements until another location counter specification is encountered. "SIX" is a statement label. The operand u subfield refers to the relative location of this instruction plus 3.

Data generation

Data items may be numeric (octal or decimal) or alphanumeric (right or left justified), single or double word. A data item is identified by a sign, a digit or an apostrophe starting the operation field of a source statment.

Numeric items may be signed. An octal value has a zero to the left of the most significant digit. A double word is generated if either the value is too large for a single word or a "D" is coded to the immediate right of the least significant digit.

Examples

+013	generates octal	000000000013
+13	generates octal	000000000015
13D	generates octal	000000000000 000000000015
-027	generates octal	777777777750

Alphanumeric data items are enclosed in apostrophes and may be signed or unsigned. A signed item is right justified and zero filled to the left. Unsigned constants are left justified and filled to the right with blanks. If an item consists of six alphanumeric characters or less one data word is generated. Seven to twelve characters are packed into a double word. A "D" after the closing apostrophe specifies that a doubleword is to be generated.

1108 internal character code is 6-bit ASCII. Table 2.3.2 lists the code.

Examples

'ABC'	generates octal	060710050505
+'ABC'	generates octal	000000060710
'ABC'D	generates octal	060710050505 050505050505
'ABCDEFG'	generates octal	060710111213 140505050505

Int Code	Graphic Symbol	Function Code	Int Code	Graphic Symbol	Function Code	Int Code	Graphic Symbol	Function Code
00	@	-	26	Q	LXM	53	:	TNE
01	(	S,SA	27	R	L,LX	54	?	TLE,TNG
02	)	SN,SNA	30	S	MI	55	!	TG
03	#	SM,SMA	31	T	MSI	56	,	TW
04	⌜	S,SR	32	U	MF	57	>	TNV
05	blank	SZ	33	V	GET	60	0	TP
06	A	S,SX	34	W	DI	61	1	TN
07	B	-	35	X	DSF	62	2	SE
10	C	L,LA	36	Y	DF	63	3	SNE
11	D	LN,LNA	37	Z	PUT	64	4	SLE,SNG
12	E	LM,LMA	40	)	OR	65	5	SG
13	F	LNMA	41	-	XOR	66	6	SW
14	G	A,AA	42	+	AND	67	7	SNW
15	H	AN,ANA	43	<	MLU	70	8	JGD
16	I	AM,AMA	44	=	TEP	71	9	
17	J	ANM,ANMA	45	>	TOP	72	'	
20	K	AU	46	&	LXI	73	;	
21	L	ANU	47	$	TLEM,TNGM	74	/	
22	M	BT	50	*	TZ	75	.	
23	N	L,LR	51	(	TNZ	76	<	
24	O	A,AX	52	%	TE	77	?	
25	P	AN,ANX						

Table 2.3.2a. Character Code and Function Codes.

f	j	Function	f	j	Function	f	j	Function
71	00	MSE	72	00	-	73	00	SSC
	01	MSNE		01	SLJ		01	DSC
	02	MSLE,MSNG		02	JPS		02	SSL
	03	MSG		03	JNS		03	DSL
	04	MSW		04	AH		04	SSA
	05	MSNW		05	ANH		05	DSA
	06	MASL		06	AT		06	LSC
	07	MASG		07	ANT		07	DLSC
	10	DA		10	EX		10	LSSC
	11	DAN		11	ER		11	LDSC
	12	DS		12	-		12	LSSL
	13	DL		13	PAIJ		13	LDSL
	14	DLN		14	SCN		14	III,ALRM
	15	DLM		15	LPS		15	SIL
	16	DJZ		16	LSL		16	LCR,LLA
	17	DTE		17	-		17	TS

Table 2.3.2b. Function Codes cont.

f	j	Function
74	00	JZ
	01	JNZ
	02	JP
	03	JN
	04	JK,J
	05	HJK,HJ
	06	NOP
	07	AAIJ
	10	JNB
	11	JB
	12	JMGI
	13	LMJ
	14	JO
	15	JNO
	16	JC
	17	JNC
77	00-17	-

f	j	Function
75	00	LIC
	01	LICM
	02	JIC
	03	DIC
	04	LOC
	05	LOCM
	06	JOC
	07	DOC
	10	LFC
	11	LFCM
	12	JFC
	13	-
	14	AACI
	15	PACI
	16	-
	17	-

f	j	Function
76	00	FA
	01	FAN
	02	FM
	03	FD
	04	LUF
	05	LCF
	06	MCDU
	07	CDU
	10	DFA
	11	DFAN
	12	DFM
	13	DFD
	14	DFU
	15	DFP
	16	FFL
	17	FCL

Table 2.3.2c. Function Codes cont.

Expressions

Wherever an elementary data item is valid, an expression of elementary items is also valid. An expression is defined as one or more elementary items connected by arithmetic or boolean operators. Expressions are evaluated according to a heirachy of operators and use of parentheses. Tables 2.3.3 and 2.3.4 summarize rules for evaluation of assembly expressions.

Examples

(1) L A3,x+6

x relocatable, 6 absolute, result relocatable

(2) +(ALPHA + (BETA++GAMMA))

evaluation BETA OR GAMMA gives RESULT1 absolute

ALPHA + RESULT1 gives RESULT relocatable if ALPHA is relocatable

(3) + ALPHA -- (BETA GAMMA)

evaluation - if BETA GAMMA RESULT1 = 1 else RESULT1 = 0

ALPHA XOR RESULT1 gives RESULT absolute.

Line items

A line item is a line of symbolic code, with no label, enclosed in parentheses. A prefix operator distinguishes a line item from a literal.

A literal is a line of symbolic code completely enclosed in parentheses. The word or double word generated by a line item is placed "in line", i.e. at the current value of the location counter. A literal generates the word, or

	Operator	Operation
Highest Priority	+	Arithmetic sum
	-	Arithmetic difference
	**	Logical product
	++	Logical sum
	--	Logical difference
	=	a=b has value 1 if true, 0 if false
		a b " " 1 if true, 0 if false
Lowest Priority		a b " " 1 if true, 0 if false

Table 2.3.3. Heirarchy of Operators.

First Item	Operator	Second Item	Result
Any	, =,	Any	Not relocatable
Any	++, --	Any	Not relocatable
Any	**	Any	Not relocatable
Not relocatable	+, -	Not relocatable	Not relocatable
Relocatable	+, -	Not relocatable	Relocatable
Not relocatable	+, -	Relocatable	Relocatable
Relocatable	+, -	Relocatable	Relocatable

Table 2.3.4. Rules for Determining Result of Operations.

doubleword, "out of line" in a literal pool. The address of this word is inserted at the current value of the location counter.

There is a literal pool for each of the 32 location counters. Duplicate literals appear only once in each pool but the same literal may be in several pools.

Examples

(1) L A2,(0734)

Machine code generated is

```
     L   A2,LP1
      .
      .
LP1  0734
```

The literal 0734 is inserted into the literal pool for the current location counter. The address of the literal is placed in the instruction.

(2) L A2,(L A3,(L A4,(67)))

Machine code generated is

```
     L   A2,LP1
      .
      .
LP1  L   A3,LP2
LP2  L   A4,LP3
LP3  67
```

Literals within literals are permitted up to eight levels.

(3) L A2,+(67)

+(67) is not a literal. Code generated is

```
L    A2,67
```

Assembler Directives

The general form for directives is:-

label directive expression

EQU directive

The EQU directive equates the label to the value of the expression. Depending on the expression so the value of the label is relocatable or not. When an absolute value is defined, the definition must precede the first use of the variable.

Examples

(1)
```
ALPHA   EQu   6
        L     21,BETA + ALPHA
```

The absolute value 6 is assigned to ALPHA so the statment becomes

```
        L     21,BETA + 6
```

(2)
```
ALPHA   EQu   $ + 2
```

The current value of the location counter plus 2 is assigned to ALPHA. ALPHA is relocatible.

END directive

The END directive instructs the assembler that the assembly is complete. There is one and only one end statement in each program. The value of the expression is relocatable and is passed to the Loader as the execution

starting address.

RES directive

The RES directive is used to

(1) reserve a block of words

(2) assign a relocatible address to a location counter

To reserve a block of words the expression reduces to an absolute value equal to the number of words required. There is no initialization. This operation is effected by adding the value of the expression to the current value of the location counter.

To assign a relocatable address to a location counter, code $(e) starting in column 1. The value of the expression is added to the location counter. Thus to initialize a location counter the current value must be subtracted from the expression.

Examples

(1) BLK RES 12

Twelve words reserved at address BLK.

(2) $(2) RES 01000 - $

Operation performed by assembler is

(01000 - $) + $ = $

for location counter 2.

Refer also to Sample Problems, Chapter IV.

SECTION 2.4 THE LOADER

The Loader loads relocatable machine code into main storage assigning absolute addresses and initializing certain fields used in program execution. The input to the loader is one **file, or several concatenated files** of output from the assembler. The initial version of the loader treats the relocatable addresses as absolute values. Base registers should therefore contain zeros. Subsequent loaders will allow the user to relocate his program during program execution.

If an address is less than 200 the word is loaded into hidden storage. The user should be aware of the fixed address assignments (see Section 2.1). These locations (200_8 to 247_8 and 252_8) are all initially loaded with the instruction

```
SLJ   ,*250
```

Note: D7 = 1 when interrupt occurs, hence * specifies absolute addressing rather than indirect addressing

i.e. store the contents of P-register at absolute location 250_8 and jump to absolute location 251_8. A dump core instruction is loaded at address 251_8.

The user program may override any of these instructions. Care should be taken to capture the contents of the P-register during the handling of an interrupt. This implies coding either a SLJ or LMJ instruction at the interrupt location. D7 and D6 are set to 1 automatically when an interrupt occurs.

Note: The loader does not check for duplicate addresses. When assembling use unique addresses for each program section.

The loader initializes several internal registers and switches. See paragraph on Initialization Section 2.5.

SECTION 2.5 THE INTERPRETER.[11]

The interpreter simulates 1108 hardware. The initial contents of the P-register is set by the loader. Logically the interpreter functions as follows:

1. Obtain absolute address of instruction from P-register.
2. Fetch instruction from main storage into control section.
3. Decode the instruction.
4. Compute the effective address.
5. Perform the operation specified in the instruction.
6. Continue at 1, except if the operation just performed was either

a) terminate the job or

b) dump core and terminate.

The terminate instruction is ER ,077.

The dump core instruction has f = 0

Each instruction is described fully in Section 2.2. Any not in that section is not implemented in the simulation. Floating point arithmetic and some I/O instructions fall into this category. If an attempt is made to perform one of these instructions an implementation interrupt occurs. This interrupt is unique to the simulator and traps to absolute location x00252 , where x = C(MSR).

The sequence of operation described above is broken if an

interrupt occurs. The contents of the PSR are stored at absolute location zero prior to executing the instruction at the interrupt location. Note that the contents of the P-register is unchanged - i.e. contains the address of the instruction being executed when the interrupt occured plus one.

DEBUG AIDS.

To assist the user in debugging 1108 assembler programs certain features have been included in the interpreter. A program trace facility is provided. This prints the address of each instruction executed. That is, the contents of the P-register prior to incrementation. When an interrupt occurs the address of the interrupt location is not captured by the trace. For example, consider the following sequence

Address	Instruction
00247_8	SLJ ,*0260
00260_8	0
00261_8	L A1,*0260
	.
	.
01004_8	DI A2,DSOR

Suppose a divide fault interrupt occurs while executing the instruction at location 01004_8. The system traps to location 00247_8 which transfers control to location 00261_8. A trace would produce the following output

001004

000261

.

.

The core dump is another debug facility available. The program is terminated after the dump. For core dump format refer to Figure 2.5.1.

Description of fields.

PPPPPP contents of the P-register.

BBBBBBBBBBBB contents of the PSR.

CCCCCCCCCCCC contents of the SLR.

EEEEEE absolute value of the last effective address computed.

IIIIII address of the last interrupt trap location. (The handling of this interrupt may or may not be complete.)

FF channel number of most recent I/O interrupt.

G CPU associated with most recent I/O interrupt.

HH channel number of most recent I/O interrupt for CPU #0.

KK channel number of most recent I/O interrupt for CPU #1.

L number of CPU performing this instruction.

M contents of the MSR.

NNNN halt key setting (binary).

DD contents of the channel select register (CSR).

J contents of the last address register (LAR).

S interrupt switch. Zero indicates that the CPU will

```
INTERNAL REGISTERS
 P-REGISTER PPPPPP    PSR  BBBBBBBBBBBB    SLR  CCCCCCCCCCCC    EFA  EEEEE
INT ADDRESS IIIIII   CHAN  FF-G-HH-KK-L    MSR  M                HK  MNNN
        CSR DD        LAR  J               IS  S               LPS  T

ADDRESS              STORAGE CONTENTS

REGISTER CONTENTS
 AAAAAA  XXXXXXXXXXXX  XXXXXXXXXXXX  XXXXXXXXXXXX  ............XXXXXXXXXXXX
                       (eight 1108 words per line)
 AAAAAA  XXXXXXXXXXXX  XXXXXXXXXXXX  XXXXXXXXXXXX  ............XXXXXXXXXXXX
   ..........................................................................
   ..........................................................................
 AAAAAA  XXXXXXXXXXXX  XXXXXXXXXXXX  XXXXXXXXXXXX  ............XXXXXXXXXXXX

HIDDEN STORAGE
 AAAAAA  XXXXXXXXXXXX  XXXXXXXXXXXX  XXXXXXXXXXXX  ............XXXXXXXXXXXX
 AAAAAA  XXXXXXXXXXXX  XXXXXXXXXXXX  XXXXXXXXXXXX  ............XXXXXXXXXXXX
   ..........................................................................
   ..........................................................................
 AAAAAA  XXXXXXXXXXXX  XXXXXXXXXXXX  XXXXXXXXXXXX  ............XXXXXXXXXXXX

MAIN STORAGE
 AAAAAA  XXXXXXXXXXXX  XXXXXXXXXXXX  XXXXXXXXXXXX  ............XXXXXXXXXXXX
 AAAAAA  XXXXXXXXXXXX  XXXXXXXXXXXX  XXXXXXXXXXXX  ............XXXXXXXXXXXX
   ..........................................................................
   ..........................................................................
 AAAAAA  XXXXXXXXXXXX  XXXXXXXXXXXX  XXXXXXXXXXXX  ............XXXXXXXXXXXX
```

Figure 2.5.1. Core Dump Format.

currently allow all I/O interrupts. If 1 then a prevent all I/O interrupts instruction has been executed. No I/O interrupts are currently allowed.

T load processor state switch. If set 1 it indicates a LPS instruction has been executed but the new PSR has not yet been loaded. Refer to Section 2.2 for a description of the delay in the LPS instruction.

AAAAAA address of the first word output this line of dump. It refers to either a control register, hidden storage or main storage address. Eight words are output per line. The addresses are AAAAAA,AAAAA+1..AAAAAA+7.

XXXXXXXXXXXX is the contents of the control register, hidden storage or main storage word at the address indicated by AAAAAA.

A core dump may be requested by the program or may be the result of a program error. To request a core dump code the following:

```
        .
        .
    L   A2,VAL1
    MI  A2,VAL2
    S   A2,VAL3
        0
```

The constant zero is programmed where the dump is required. After executing the store instruction the interpreter proceeds to the next sequential instruction, which dumps **core.**

A special version of the interpreter is available to provide "mini-dumps" before the execution of each instruction. Use LEY1108 for this feature. The normal interpreter is LEX1108 (see Section 2.5 Job Control Language).

A "mini-dump" prints the contents of eight words - 4 control registers and 4 main storage locations. The control registers 14 - 17 inclusive (i.e. A2 - A5, 16_8 - 21_8) and the main storage words at addresses 514 - 517 inclusive (i.e. 1002_8 - 1005) are output. Note that control registers 14 and 15 may be either accumulators or index registers.

If a program is written to use these control registers and four main storage locations, the mini-dump can be of considerable assistance. When the program is debugged remove the mini-dump by executing LEX1108. The mini-dump is illustrated in Section D: Sample Problems.

Subroutines are often useful for program testing. One such routine is Sample Problem 3, Section D, which prints, in both display and octal, the contents of one to seven words of main storage. When testing is complete remove the subroutine calls.

INITIALIZATION.

The following are initialized at program execution time:
P-register: value taken from END statement.
PSR: 000000177000_8

i.e. D8 - D0,QW = 0

BI = 0

BS = 177_8

BD = 0

SLR: 110000110001_8

i.e. IU = 110_8

IL = 0

DU = 110_8

DL = 1

All other internal registers are initialized at zero. Switches, such as SELECT JUMPS and SELECT STOPS, are all zero.

Note: there is no initialization of control registers or main storage.

SECTION 2.6 JOB CONTROL LANGUAGE.

Job Control Language discussed herein is for OS/360 MVT. The most commonly used JCL is listed below.

To compile an 1108 Assembler language program

```
//NAME JOB ' a/c information ',programmer
//JOBLIB DD DSN=RLMACL.A0224.RACLIB,DISP=SHR,
//       VOLUME=SER=UM1404,UNIT=SYSDA
//    EXEC PGM=ASM1108,REGION=42K
//SYSUT1 DD DSN=filename1,UNIT=SYSDA,
//       VOLUME=REF=ONE.MONTH,DISP=(NEW,DELETE),
//         SPACE=(TRK,(1,1),RLSE)
//SYSUT2 DD DSN=filename2,UNIT=SYSDA,
//      VOLUME=REF=ONE.MONTH,DISP=(NEW, DELETE),
                                        KEEP
//     SPACE=(TRK,(1,1),RLSE)
//SYSUT3 DD DSN=filename3,UNIT=SYSDA,
//      VOLUME=REF=ONE.MONTH,DISP=(NEW,DELETE),
//     SPACE=(TRK,(1,1),RLSE)
//SYSPRINT DD SYSOUT=A
//SYSUDUMP DD SYSOUT=A
//SYSIN DD *
   source statements
   .
   .
/*
```

The partitioned data set RLMACL.A0224.RACLIB (on permanently mounted volume Um1404) contains the load module ASM1108. Two temporary work files SYSUT1 and SYSUT3 should be deleted after the assembly. The relocatable module produced from the assembler is written to the file SYSUT2.

To compile, load and execute an 1108 Assembler language program

```
//NAME JOB ' a/c information ',programmer
//JOBLIB DD DSN=RLMACL.A0224.RACLIB,DISP=SHR,
//       VOLUME=SER=UM1404,UNIT=SYSDA
//    EXEC PGM=ASM1108
//SYSUT1 DD DSN=filename1,UNIT=SYSDA,
//       VOLUME=REF=ONE.MONTH,DISP=(NEW,DELETE),
//         SPACE=(TRK,(1,1),RLSE)
//SYSUT2 DD DSN=filename2,UNIT=SYSDA,
//      VOLUME=REF=ONE.MONTH,DISP=(NEW,PASS),
//     SPACE=(TRK,(1,1),RLSE)
//SYSUT3 DD DSN=filename3,UNIT=SYSDA,
//      VOLUME=REF=ONE.MONTH,DISP=(NEW,DELETE),
//     SPACE=(TRK,(1,1),RLSE)
//SYSPRINT DD SYSOUT=A
//SYSUDUMP DD SYSOUT=A
//SYSIN DD *
   source statements
   .
   .
/*
```

```
// EXEC PGM=LEXl108,PARM='MS=size,TRACE= NO',REGION=78K
            LEYl108                          YES
//SYSUT2 DD DSN=filename2,DISP=(, DELETE),
                                        KEEP
//        UNIT=SYSDA,VOLUME=REF=ONE.MONTH
//SYSPRINT DD SYSOUT=A
//SYSUDUMP DD SYSOUT=A
//SYSIN DD *
   program data
   .
   .
/*
```

For an explanation of the compile JCL see above. There are two parameters input to the Loader/Interpreter.

(a)MS=XXXXX where XXXXX is a decimal integer equal to the 1108 high main storage address required. XXXXX should lie on the range 256 to 4095, or 32768 to 36863. Default is 36863.

(b)TRACE= YES

NO (default value)

The trace prints the value of the P-register prior to execution of each instruction.

The relocatable module is loaded from the file SYSUT2. Separate assembly is possible by concatenating data sets to form the logical file SYSUT2.

To execute a program previously compiled.

```
//NAME JOB ' a/c information ',programmer
//JOBLIB DD DSN=RLMACL.A0224.RACLIB,DISP=SHR,
```

```
//       VOLUME=SER=UM1404,UNIT=SYSDA
// EXEC PGM= LEX1108,PARM='MS=size,TRACE= NO',REGION=78K
             LEY1108                          YES
//SYSUT2 DD DSN=filename2,DISP=(, DELETE),
                                    KEEP
//         UNIT=SYSDA,VOLUME=REF=ONE.MONTH
//SYSPRINT DD SYSOUT=A
//SYSUDUMP DD SYSOUT=A
//SYSIN DD *
   program data
   .
   .
```

/* Note: Use LEY1108 instead of LEX1108 to get the "mini-dump" debug feature.

Core Requirements:

Assembler 42K bytes.

Loader/Interpreter 38K+5(1108 size) bytes. For example 78K is required for a 8K 1108 system.

Section 2.7 ERROR CODES.

For the most part the user is unaware that his program is executed by an IBM 360/65. Most errors detected give appropriate 1108 error messages. There are, however, instances where IBM error codes are output. This may occur when

(a) The user steps beyond the scope of the system and attempts to use an 1108 feature not implemented.

(b) The system fails.

To correct an (a) type error the user should consult the User Guide, modify his program and re-submit the job.

Type (b) errors should be reported to a Systems Programmer.

The Assembler.

Source statement errors are listed in the Assembler print. The type of error is represented by an alphabetic code which is printed under the heading "Error Flags". A "#" is placed under the character where the error is detected. If the error occurs in an arithmetic expression the last element of the expression is flagged.

Error Flag	Description	User Action
D	Duplicate label.	Correct and re-submit.
E	Invalid Expession.	Alter expression and

		re-submit.
I	Invalid instruction.	Correct the mnemonic and re-submit.
L	Level - line item or literal nested to a depth greater than 8.	Change the line item such that level is 8 or less and re-submit.
R	Relocation error. A relocatable item loses its relocatability in evaluation of expression.	Correct the expression and re-submit.
S	Syntax error. Statement format invalid or a,x, or j value outside allowable range.	Correct statement and re-submit.
T	Truncation error. Data item too long, eg. character string of length 13.	Reduce field size by coding additional statements.
U	Undefined label. An absolute assignment may follow the label reference.	Define item or re-position the definition.
W	I/O error.	Re-submit. If error persists see Systems Programmer.
X	System error.	See Systems Programmer
Y	Implementation error. Feature not included in current version of Assembler.	Alter to avoid feature and re-submit.
Z	End-of-file error. Unexpected EOF condition, eg. during	Correct, ensure an END statement completes

statement continuation. program.

The assembler may abnormally terminate with one of the following /360 user abend codes.

User Abend Code	Description	User Action
99	Error on system work file write.	Re-run. If error persists see Systems Programmer.
199	Error on print file write.	As for 99.
299	Error on output file write.	As for 99.
2222	Error on source file open.	Check source file DCB and re-run.
2223	Error on system work file open.	Check system work file DCB and re-run.
3000	Label table overflow. Maximum number of labels approx. 500.	Reduce number of labels and re-run.
3002	Procedure statement used.	Alter program to avoid feature and re-run.
3003	Constant table overflow. Maximum approx. 100.	Reduce the number of "equate" statements.
3004	Directive not implemented.	Replace directive and re-run.
3103	Floating point constant.	Remove floating point item and re-compile.
3107	Invalid operation in Assembler expression.	Replace invalid operator and re-run.
6010	Assembler internal push down	Correct level error

	list overflow. May be a level error or system error.	and re-compile, or see Systems Programmer.
6011	System error.	See Systems Programmer
6088	Open system work file error.	See Systems Programmer
8001	Partial word constant.	Replace partial word constants with single octal value.
8021	Literal table overflow. Maximum approx. 150.	Reduce number of literals and re-run.

The Loader

The loader may terminate abnormally with the following /360 error code:

User Abend Code	Description	User Action
101	Error on input file read.	Re-submit. If error persists consult Systems Programmer.

The Interpreter

An error in program execution causes an interrupt to a fixed main storage address. Unless overriden, this instruction requests an 1108 core dump.

The interpreter may abnormally terminate with one of the following /360 error codes:

User Abend Code	Description	User Action
102	I/O interrupt detected.	Replace the I/O instruction that caused the interrupt.
103	Invalid effective address. E must satisfy $0 \leq E \leq 7777_8$ or $100000_8 \leq E \leq 107777_8$	Correct instruction and re-submit.
104	Invalid parameter string.	Correct(see Section 2.6) and re-submit.

Chapter III SYSTEM DESCRIPTION

The system is described from the Systems Programmer's viewpoint. Each source program is discussed and flow charts are provided for the main subprograms. Included in the discussion of each section are mathematical formulae and the more important data structures used.

SECTION 3.1 SYSTEM STRUCTURE.

This section outlines system stucture and program logic. All programming is in /360 Assembler Language. Sections 3.2 and 3.3 discuss individual modules, their logic and linkage.

The two pass assembler reads Univac 1108 Basic Assembler **Language from a source file, usually the card reader. Pass 1 assigns relocatable values to labels and creates a table** of user assignments. The length of each program segment is measured to determine origins of literal tables. Some error checking is done. Two system files are created - one a duplicate of the input file, the other a record of errors detected. Pass 2 reads these two files producing 1108 machine code which is written, together with relocatable addresses to the system output file. Pictorially the operation of the assembler is illustrated in Figure 3.1.1.

File attributes and record formats are as follows:

Source file Record length = 80 bytes. Block size = 80 bytes. The first 72 bytes of each record contain source data.

SYSUT1 Record length = 84 bytes. Block size = 84 bytes. The first 80 bytes are copied from the source file record. The remaining 4 bytes are error flags set during pass 1.

SYSUT3 Record length = 80 bytes. Block size = 800 bytes. The position of errors in the corresponding source record is marked.

SYSUT2 Record length = 16 bytes. Block size = 160

bytes. Record format is shown in Figure 3.1.2. Only when the source statement generates a double word, are bytes 9-13 used.

The assembler produces a list of the source statements together with machine code generated and errors detected.

Version 1 of the Assembler requires a 42K region. Limitations imposed on a single assembly are:

maximum number of labels approx. 500.

maximum number of absolute assignments approx. 100.

maximum number of operators in assembler expression = 100.

The loader reads the file SYSUT2. Machine code is loaded into 1108 Main Storage. If the relocatable address in the input record is 777777_8 then the record contains the entry address of the program (bytes 4-8). This value is loaded into the P-register.

When the loading is completed, the interpreter executes the user program. The system files are used for reading user data cards and printing user reports. Only the first 72 characters of each 80 character input record is given to the user program. These 72 characters are placed in 12 consecutive 1108 words (6 characters per word).

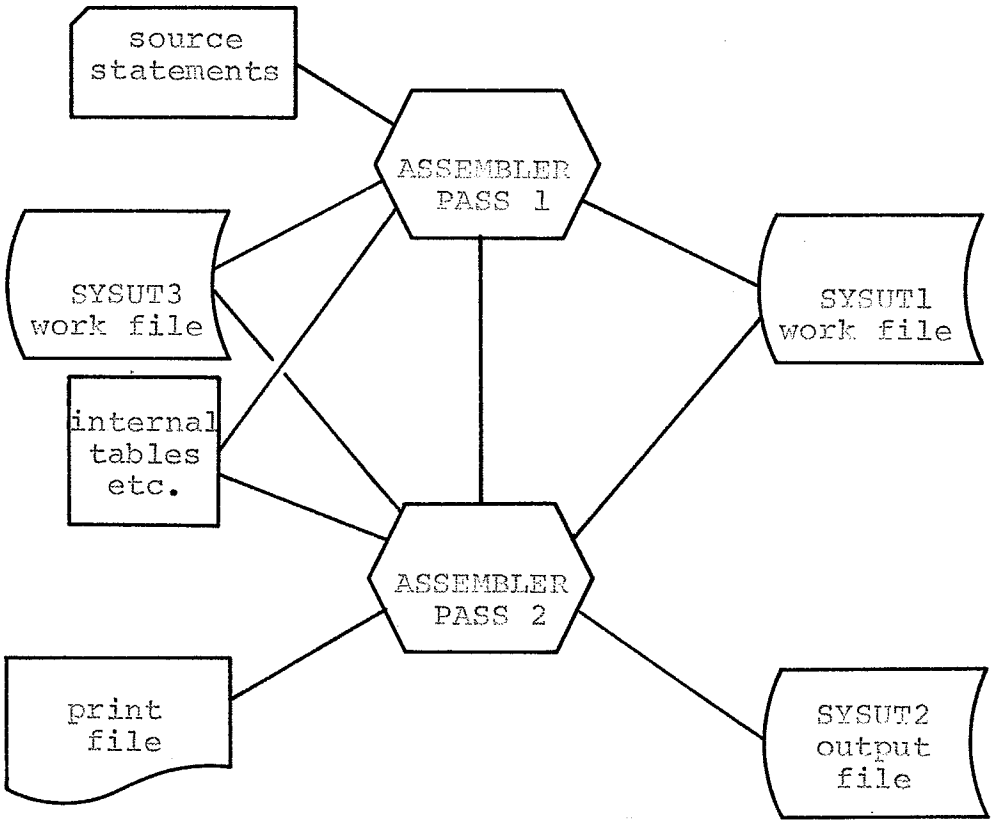
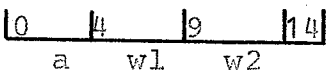


Figure 3.1.1. The Assembler.



where a = relocatable address of generated word
w1 = generated word
w2 = second generated word, if applicable
bytes 14, 15 are unused

Figure 3.1.2. Assembler Output Record Format.

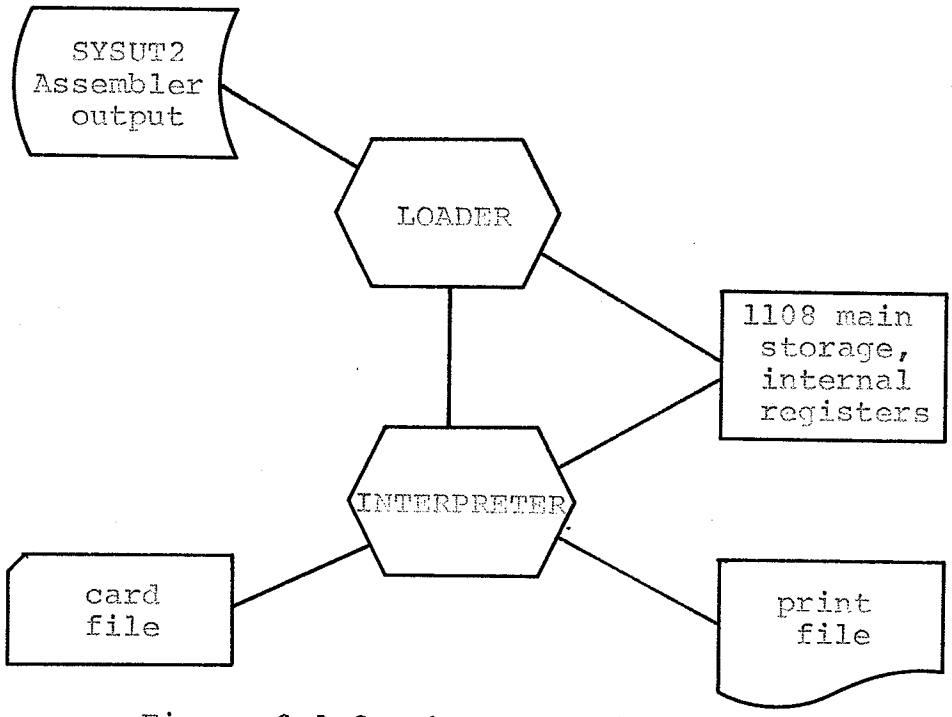


Figure 3.1.3. The Loader/Interpreter.

SECTION 3.2 THE ASSEMBLER.

The Assembler structure is presented in Figure 3.2.1. A brief discussion of each Assembler module is found later in this section. Module Linkage is done according to standard /360 practice. A communication fullword is used by many programs to indicate entry and exit conditions. Each character string passed between modules has a PL/I type "dope vector" which contains the address and length of the string.

LABELS

Labels are inserted into the label table during pass 1. Each element of the table is 10 bytes long.

Bytes 0-5 contain the LABEL MNEMONIC

Byte 6 contains FLAGS

bit 0 - element occupied.

1 - label externalized.

2 - mnemonic value is absolute.

3-7 - an integer between zero and 31 representing the location counter under which mnemonic is defined.

Bytes 7-9 contain VALUE. If bit 2 of the flag byte is zero then "value" contains the relocatable value of the mnemonic. If bit 2 of the flag byte is 1 then "value" serves as a pointer to a table of constants (see below).

The position of an entry in the label table is determined by the "hash" value (see glossary) of the mnemonic. If this location is occupied (flag bit 0) the assembler scans

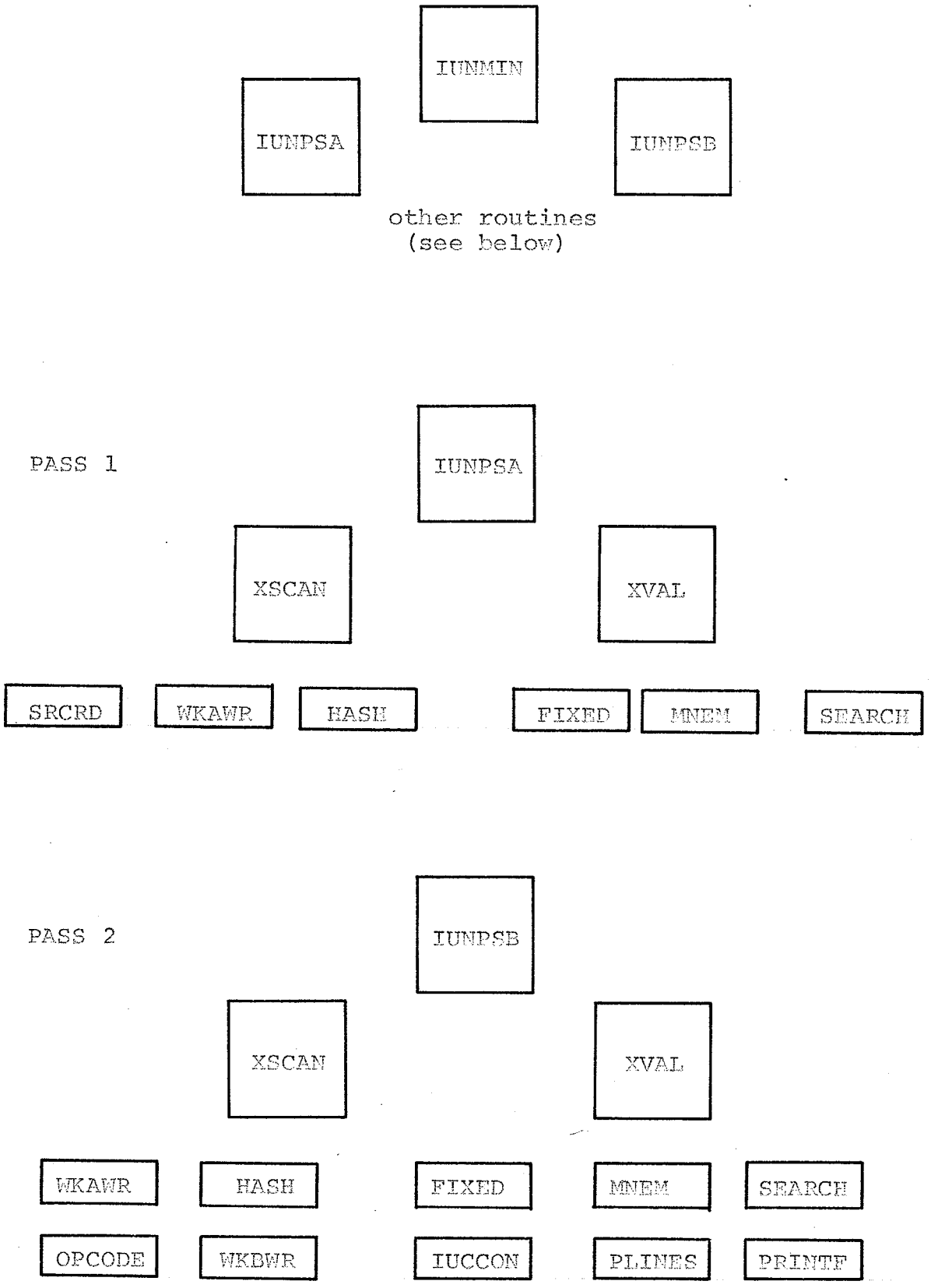


Figure 3.2.1. Assembler Structure.

for the next free location. At the end of pass 1 each "value" is incremented by the location counter base address.

An extension to the label table is a table of absolute constants. When a mnemonic is assigned an absolute value, it is inserted into the label table and "value" points to the element in the constant table.

Elements in this table are 10 bytes long, and contain two 1108 words. The leftmost 4 bits give attributes of the item, e.g. single or double word, binary or floating.

EXPRESSIONS

In scanning an arithmetic expression the assembler first forms the reverse polish equivalent. This expression is then evaluated. [12]

The reverse polish string has the following form:

Indicator bytes are intermixed with arithmetic values. An indicator byte may contain a mumeric code representing an operation (operation byte) or, the attributes of a numeric value in the string (attribute byte).

ASSEMBLER MODULES

IUNMIN - Mainline program.

Invokes passes 1 and 2.

IUNPSA - Pass 1.

Flow of pass 1 in Figure 3.2.2.

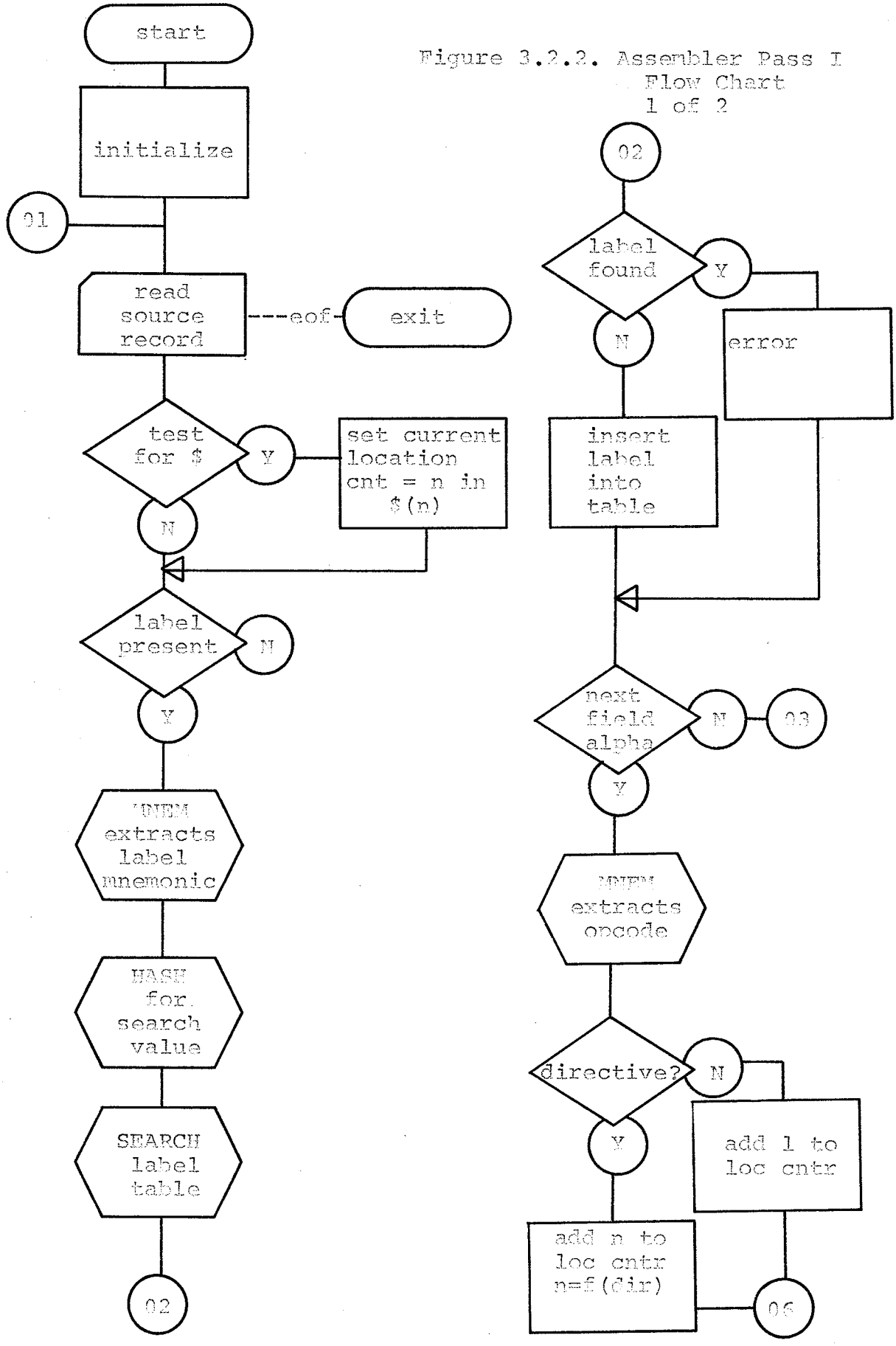


Figure 3.2.2. Assembler Pass I Flow Chart 1 of 2

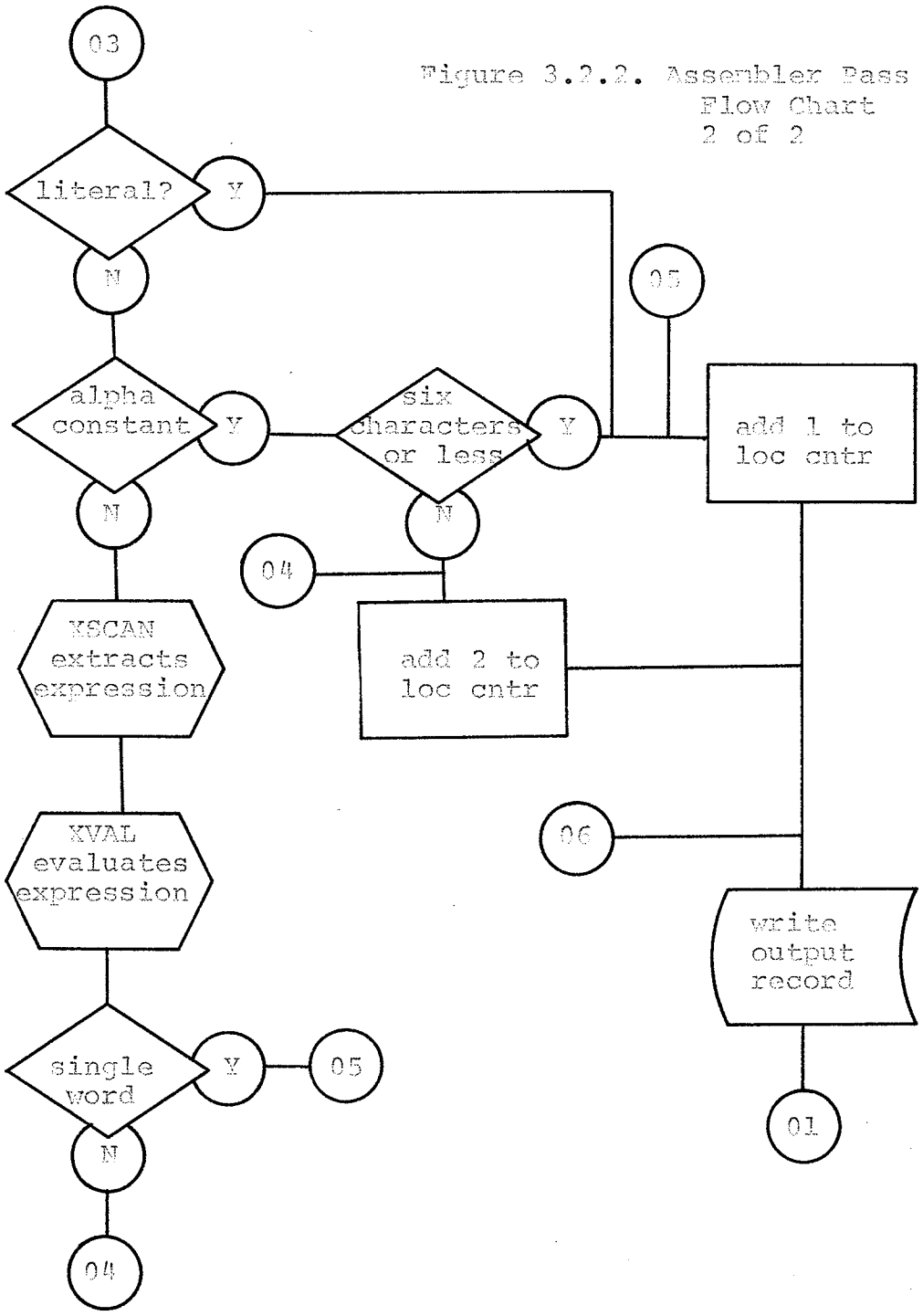


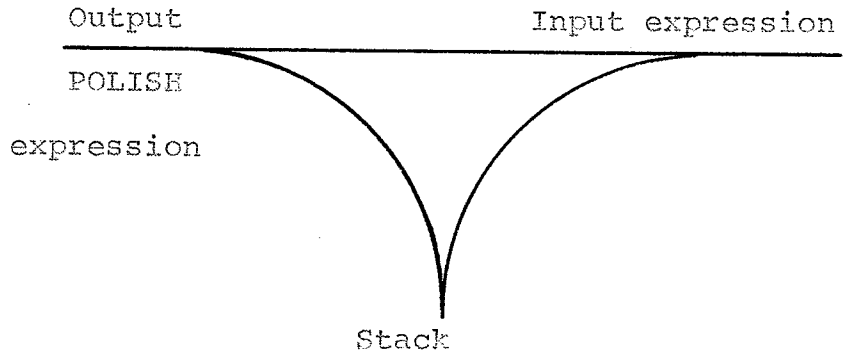
Figure 3.2.2. Assembler Pass I
Flow Chart
2 of 2

IUNPSB - Pass 2.

Flow of pass 2 in Figure 3.2.3.

XSCAN - Scans a source arithmetic expression and constructs the equivalent reverse polish expression.

The input string is scanned and the polish notation formed according to the following algorithm.



Each operator has a stack priority and a compare priority.

1. if nextsymbol (input) = operand
   then pass it through to the output
   else
2. if stackpriority (operator at top of stack)
   comparepriority (incoming operator)
   then pass stack operator to output
   else move incoming operator to top of stack.

Operator priorities are listed in Table 3.2.1.

XVAL - Evaluates a reverse polish expression.

XVAL reduces the reverse polish string to "arg" "terminator".

The string is scanned for the combination

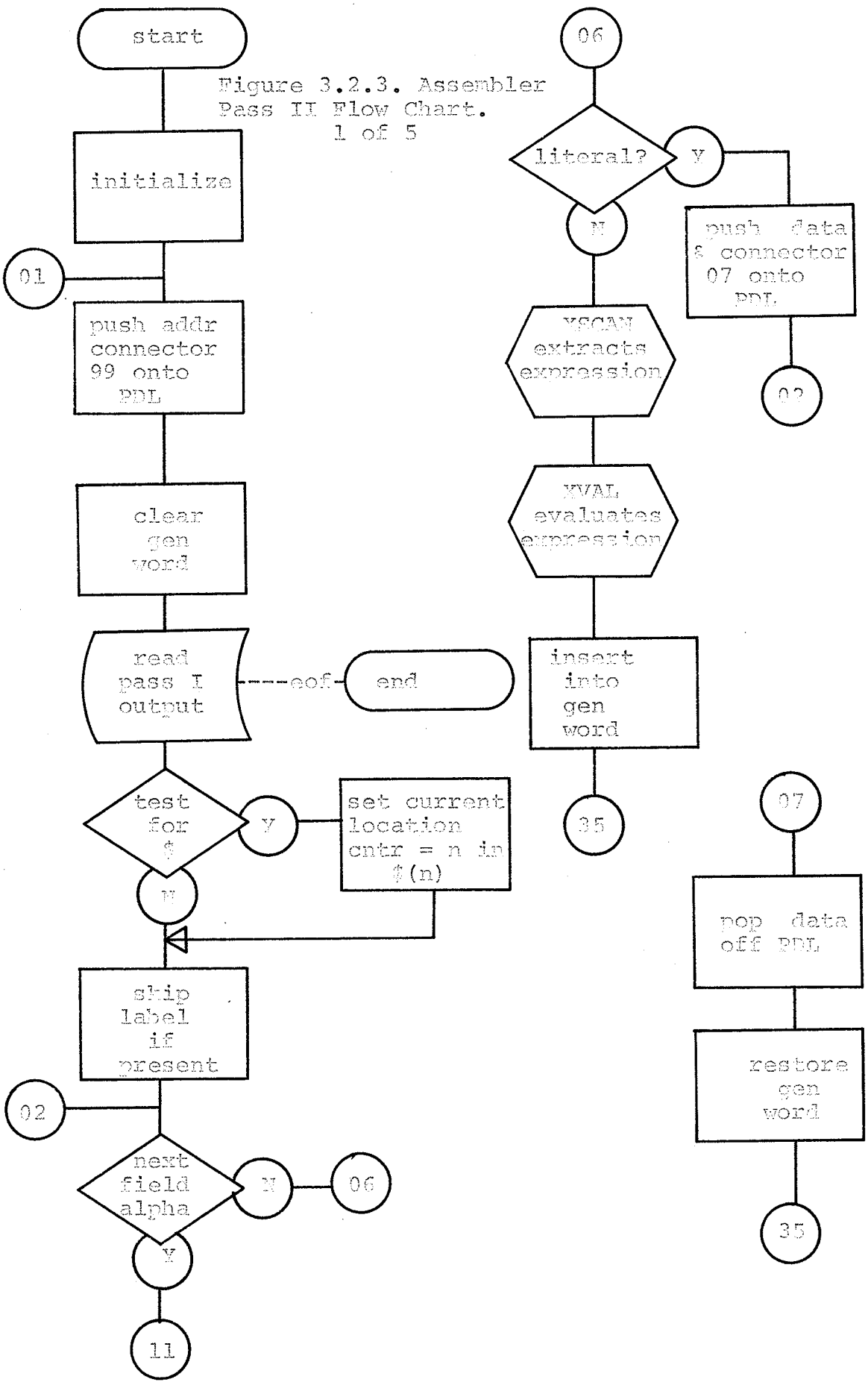
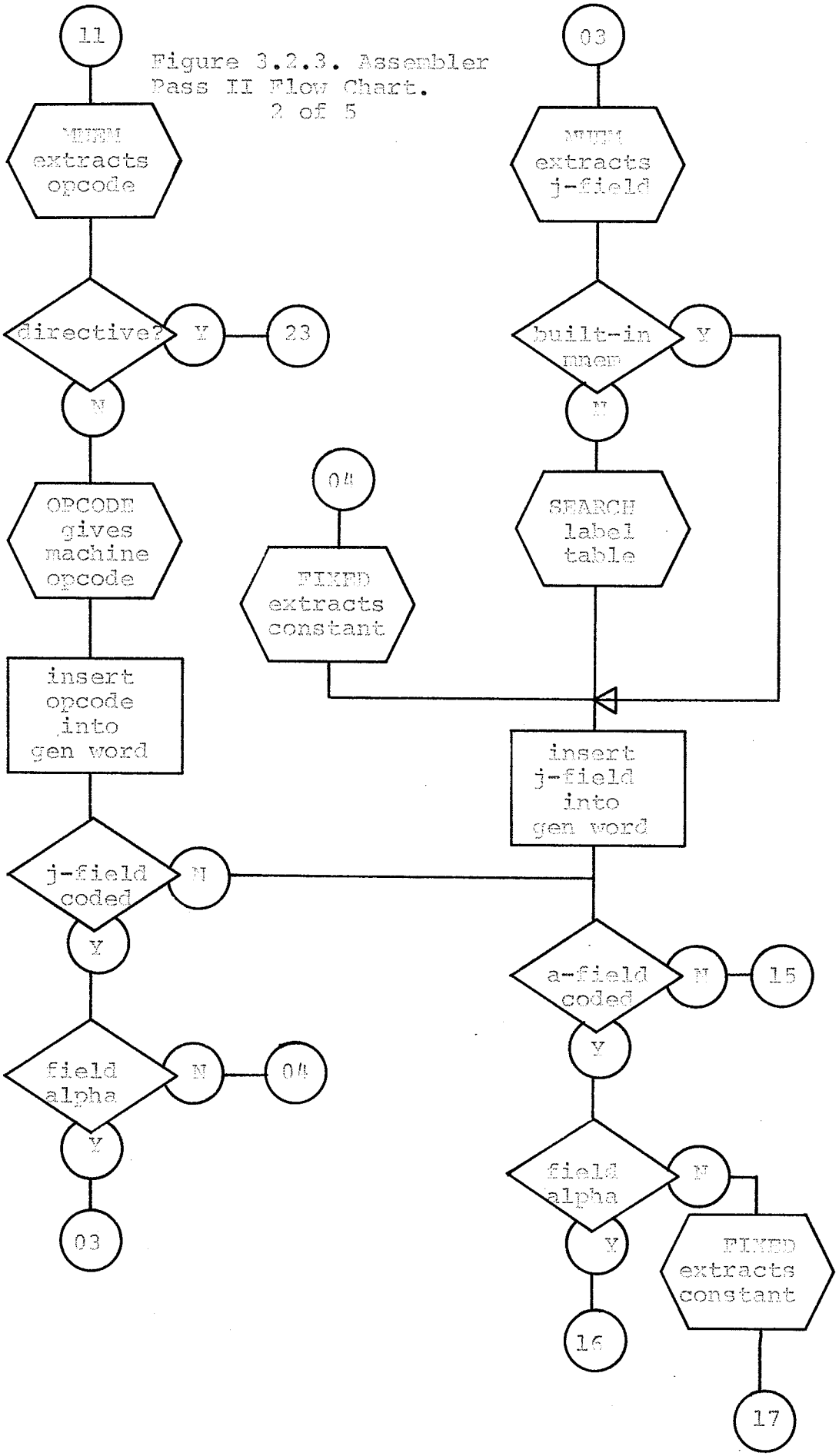


Figure 3.2.3. Assembler Pass II Flow Chart. 1 of 5

Figure 3.2.3. Assembler Pass II Flow Chart. 2 of 5



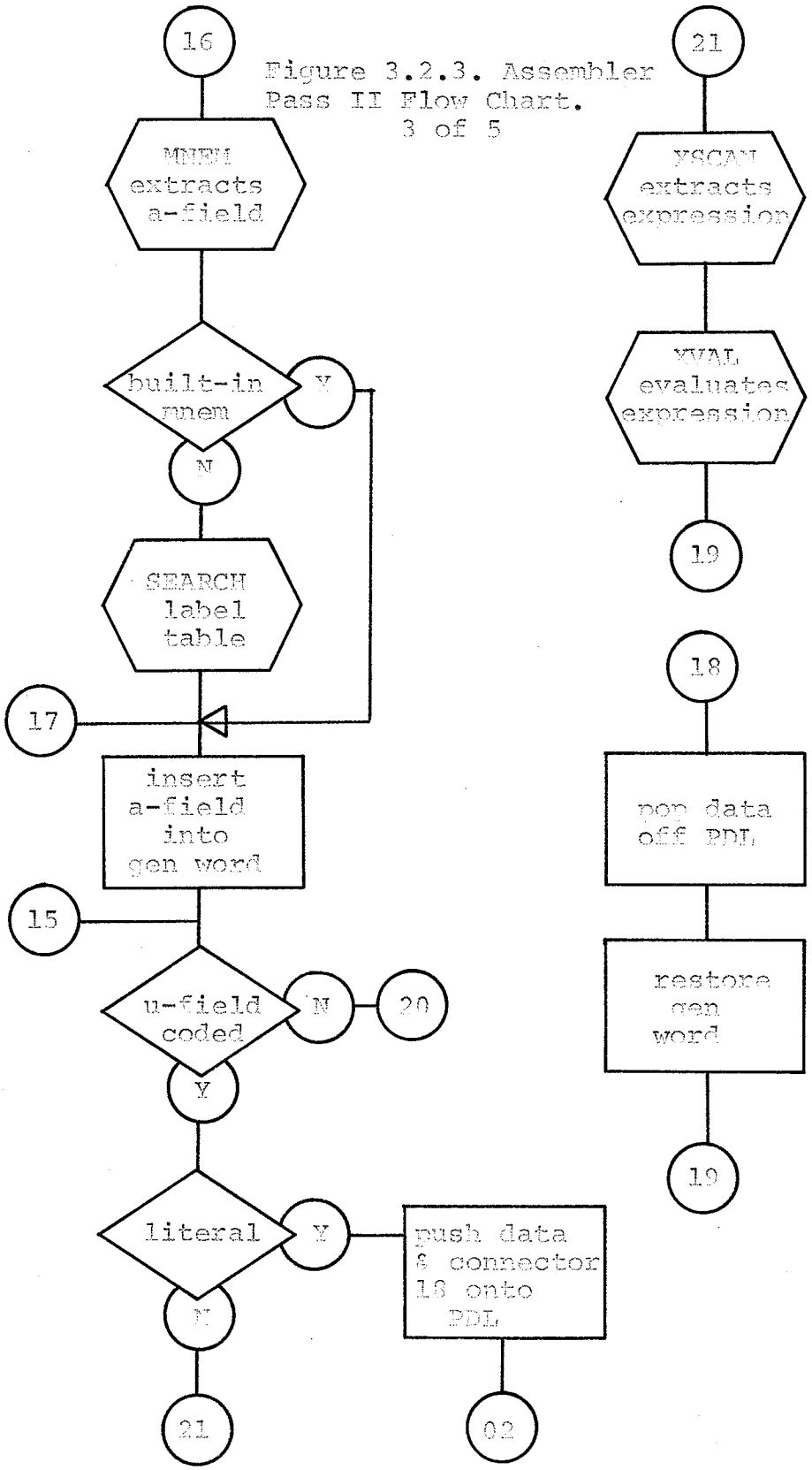


Figure 3.2.3. Assembler Pass II Flow Chart. 3 of 5

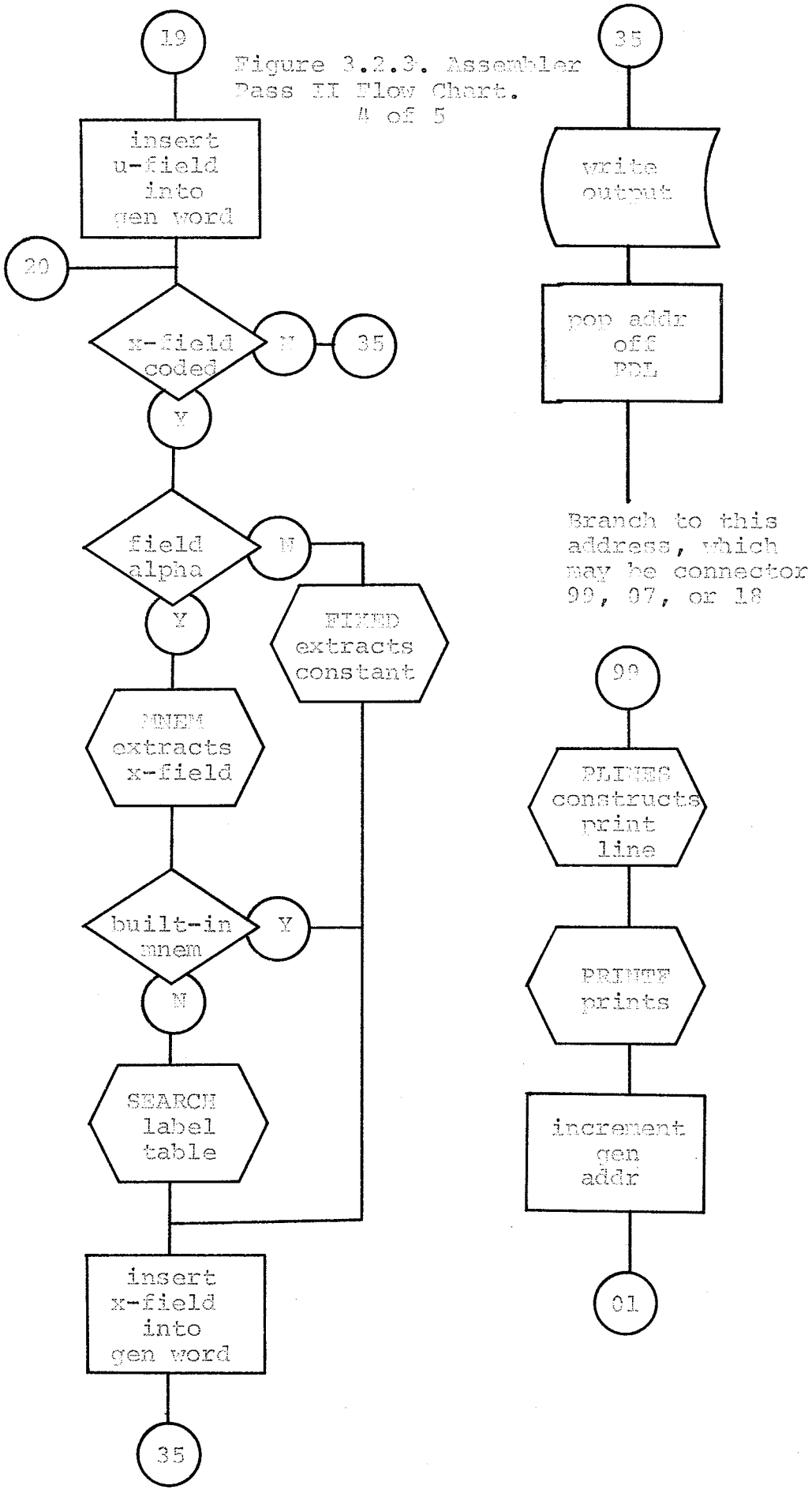


Figure 3.2.3. Assembler Pass II Flow Chart. 4 of 5

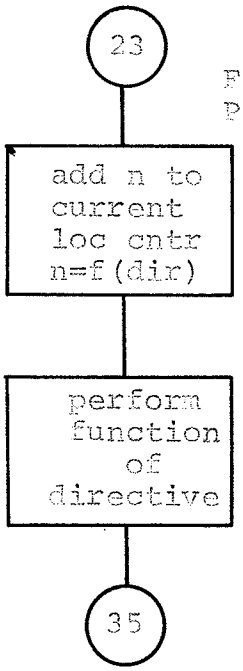


Figure 3.2.3. Assembler Pass II Flow Chart. 5 of 5

Operator	Code (hex)	Stack prio	Compare prio
(	38	01	08
<	00	02	02
>	04	02	02
=	08	02	02
--	0C	03	03
++	10	03	03
**	14	04	04
-	18	05	05
+	1C	05	05
//	20	06	06
/	24	06	06
*	28	06	06
*/	2C	07	07
*-	30	07	07
*+	34	07	07
)	44	—	02
⊥	3C	00	00

Note: i) Close parentheses are never stacked.

ii) ⊥ is string terminator.

Table 3.2.1 Operator Stack and Compare Priorities.

"arg" "arg" "operator"

or "arg" "prefix operator"

The operation is performed and the above reduced to "arg". The string is compressed. The scan process then continues. If neither of the above forms can be located then the evaluation is terminated abnormally. When the "arg" "terminator" string is reached the argument attributes are validated.

SRCRD - All source file operations.

The module opens, closes and reads the source file.

PRINTF - All print file operations.

Opens, closes and outputs the print file.

WKAWR - All system work file operations.

Performs all I/O for files SYSUT1 and SYSUT3.

WKBWR - All system output operations.

Opens, closes, and writes file SYSUT2.

This module is also used by the Loader to read SYSUT2.

HASH - hashes a mnemonic into an integer value.

The six characters of the mnemonic are V1,V2,....V6.

for all i, $0 \leq Vi \leq 24_{16}$.

A - Z give 0 - 25

0 - 9 give 26 - 35

blank gives 36

The hash value h0 is computed as -

$$h0=\underset{704}{MOD}\ (I(27V1+27V2/37)+I(19(V3+V4)/2)+I(19(V5+V6)/74))$$

where I means "integer part of".

Note that $0 \leq h0 < 704$

The mnemonic is placed at location h0 of the label table unless occupied (see SEARCH). (Refer "hash" Glossary)

FIXED - Reads decimal or octal integers from source string and converts to binary values.

Numbers are treated according to certain characteristics - viz.

(a) An Octal number is manipulated logically to extract the binary equivalent.

(b) A decimal number of magnitude less than or equal to $2^{31}-1$ is converted as a /360 fullword value.

(c) A decimal number of magnitude greater than $2^{31}-1$ but less than or equal to $2^{63}-1$ is converted as a /360 doubleword value. This is then converted to an 1108 fullword or doubleword.

(d) A decimal number of magnitude greater than $2^{63}-1$ (that does not cause 1108 overflow) is converted to octal by decimal division taking successive remainders.

MNEM - Extracts mnemonic from source string.

The first character must be alphabetic. Subsequently any non-alphanumeric terminates the scan.

*SEARCH - Searches the label table for a mnemonic and returns the table offset.

The label table search commences at entry h0, where h0 is the hash value of the mnemonic. A seek can result in -

(a) label found - return table offset.

(b) label not found, table entry occupied - continue search at next entry.

(c) label not found, table entry unoccupied - return table offset of expected entry.

The search continues only if

$$ha - h0 \leq 500d/n$$

is satisfied.

where ha = actual table entry this seek

h0 = original hash value

d = tolerance (currently 5)

n = number of elements in table

The constant 500 is used to give a table "fill" factor of 500/704. That is, of the 704 possible table entries it is hoped to use 500, without any search going beyond the tolerance. (Refer "hash" Glossary)

OPCODE - Translates a mnemonic opcode into its machine code equivalent. The opcode table is searched comparing the input mnemonic with those in the table. When found, the f and j values are returned to the calling program.

IUCCON - Translates EBCDIC character code to 6-bit ASCII.

Either six or twelve byte strings are converted and

packed into an 1108 single or double word.

This module is also used by the Interpreter.

PLINES - Constructs print lines for pass 2.

The print line has the format:

Position 1	control character
2 - 17	error flags
18 - 23	generation address in octal
25 - 36	generated word in octal
38 - 58	generated word divided into separate fields
60 - 131	source code

If a doubleword is generated a second call to PLINES is made. The second word is printed in octal from column 25 to column 36.

SECTION 3.3 THE LOADER/INTERPRETER

The interpreter structure is designed for easy maintainability and expansion. Each 1108 machine instruction is handled by a separate CSECT which is named

IUNFxx ($0 < f \leq 70_8$) or IUxxJyy ($f > 70_8$)

where xx = octal value of f

yy = octal value of j

Table 3.3.1 cross-refers CSECT to module in system library.

Figure 3.3.1 illustrates the three-level subprogram structure which gives optimum use of core storage.

The contents of the Internal Register Block (IRB), which is described fully below, show the status of the system at any time. All subprograms have access to the IRB.

A 240 byte work storage area is obtained at initialization time and is used by all subprograms.

Main storage for the 1108 is obtained at initialization time and is a function of the user assigned MS parameter.

System function subroutines are available to both mainline and machine instruction programs. They perform tasks such as effective address calculation and production of the 1108 core dump.

The logic of each module is discussed in three sections

(a) Mainline Program.

(b) Machine Instruction Subprograms.

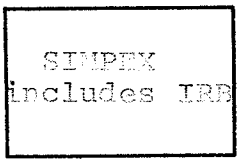
(c) System Function Subprograms.

Csect	Module	Csect	Module	Csect	Module
IUNF01	IUNF01	IUNF35	DIVIDE	IUNF70	IU74J14
IUNF02	IUNF01	IUNF36	DIVIDE	IU71J00	IU71J00
IUNF03	IUNF01	IUNF37	IUNF33	IU71J01	IU71J00
IUNF04	IUNF01	IUNF40	IUNF40	IU71J02	IU71J00
IUNF05	IUNF01	IUNF41	IUNF40	IU71J03	IU71J00
IUNF06	IUNF01	IUNF42	IUNF40	IU71J04	IU71J04
IUNF10	IUNF10	IUNF43	IUNF10	IU71J05	IU71J04
IUNF11	IUNF10	IUNF44	IUNF44	IU71J06	IU71J04
IUNF12	IUNF10	IUNF45	IUNF44	IU71J07	IU71J04
IUNF13	IUNF10	IUNF46	IUNF10	IU71J10	IU71J10
IUNF14	IUNF14	IUNF47	IUNF44	IU71J11	IU71J10
IUNF15	IUNF14	IUNF50	IUNF44	IU71J12	IUNF01
IUNF16	IUNF14	IUNF51	IUNF44	IU71J13	IUNF10
IUNF17	IUNF14	IUNF52	IUNF44	IU71J14	IUNF10
IUNF20	IUNF14	IUNF53	IUNF44	IU71J15	IUNF10
IUNF21	IUNF14	IUNF54	IUNF54	IU71J16	IU74J14
IUNF22	IUNF22	IUNF55	IUNF54	IU71J17	IU74J14
IUNF23	IUNF10	IUNF56	IUNF54	IU72J01	IU72J01
IUNF24	IUNF14	IUNF57	IUNF54	IU72J02	IU74J14
IUNF25	IUNF14	IUNF60	IUNF54	IU72J03	IU74J14
IUNF26	IUNF10	IUNF61	IUNF54	IU72J04	IU72J04
IUNF27	IUNF10	IUNF62	IUNF62	IU72J05	IU72J04
IUNF30	MULT	IUNF63	IUNF62	IU72J06	IU72J04
IUNF31	MULT	IUNF64	IUNF62	IU72J07	IU72J04
IUNF32	MULT	IUNF65	IUNF62	IU72J10	IU72J01
IUNF33	IUNF33	IUNF66	IUNF62	IU72J11	IU72J01
IUNF34	DIVIDE	IUNF67	IUNF62	IU72J13	IU72J01

Table 3.3.1a Csects/Library Modules.

Csect	Module	Csect	Module	Csect	Module
IU72J14	IU72J01	IU73J11	IU72J06	IU74J05	IU74J00
IU72J15	IU72J15	IU73J12	IU72J06	IU74J06	IU74J00
IU72J16	IU72J15	IU73J13	IU72J06	IU74J07	IU72J15
IU73J00	IU73J00	IU73J15	IU72J15	IU74J10	IU74J00
IU73J01	IU73J00	IU73J16	IU72J15	IU74J11	IU74J00
IU73J02	IU73J00	IU73J17	IUNF54	IU74J12	IU74J00
IU73J03	IU73J00	IU74J00	IU74J00	IU74J13	IU72J15
IU73J04	IU73J00	IU74J01	IU74J00	IU74J14	IU74J14
IU73J05	IU73J00	IU74J02	IU74J00	IU74J15	IU74J14
IU73J06	IU72J06	IU74J03	IU74J00	IU74J16	IU74J14
IU73J07	IU72J06	IU74J04	IU74J00	IU74J17	IU74J14
IU73J10	IU72J06				

Table 3.3.1b Csects/Library Modules.



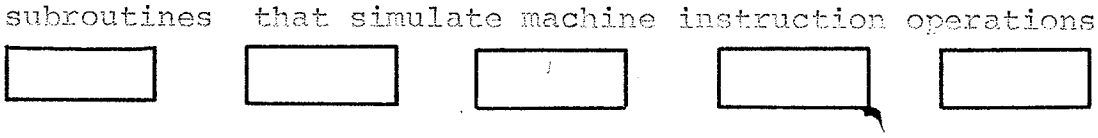


Figure 3.3.1. Loader/Interpreter Structure.

(a) Mainline Program (Module name SIMPEX)

ASSIGNMENTS AND DEFINITIONS.

200 bytes of main storage are reserved for system internal registers. This area is called the Internal Register Block (IRB). Throughout execution, register 3 addresses the IRB.

Contents of the IRB

Offset	Length (bytes)	Mnemonic	Contents
0	4	PAC	P-register - absolute 1108 address of next instruction.
4	4	IAR	Instruction address register. The /360 address of the instruction being executed.
8	5	SLR	Storage limits Register. The leftmost 36 bits contain the 1108 SLR.
16	4	EFA	Effective address (1108) computed for the current instruction.
20	5	PSR	Processor State Register. The leftmost 36 bits contain the 1108 PSR.
28	16	ACU	Arithmetic/Control unit. 16 bytes used as the 1108 ACU.
44	2	F	The integer value of the f-field in the current instruction.

46	2	J	The integer value of the j-field in the current instruction.
48	72	DSA	An 18 word save area used by most subroutines.
120	4	A	/360 offset of the control register referenced by the a-field in the current instruction. If D6=0 in PSR then to get the actual address of the register: - add 0 if a-field refers to X-register. - add 60 if a-field refers to A-register. - add 320 if a-field refers to R-register. If D6=1 in PSR then to get the actual address of the register - add 0 if a-field refers to X-register. - add 60 if a-field refers to A-register. - subtract 80 if a-field refers to R-register.
124	2	SJK	Select jump keys. Rightmost 15 bits used for keys.
126	1	HK	Halt keys. Rightmost 4 bits

			used for halt keys.
128	4	IA	Interrupt Address. 1108 absolute address from which next instruction is to be taken - because of an interrupt. This field is cleared as soon as the interrupt is handled.
132	4		Unused.
136	4	CHAN	Byte 0 contains the channel number of the most recent I/O interrupt (4 bits) and the associated CPU number (2 bits) - right justified. Byte 1 contains the channel number of the most recent I/O interrupt (4 bits) for CPU #0 - right justified. Byte 2 contains the channel number of the most recent I/O interrupt (4 bits) for CPU #1 - right justified. Byte 3 contains the number of the CPU performing this instruction (2 bits) - right justified.
140	1	MSR	Memory Select Register. Contained in bits 1-3 of the byte. All other bits zero.

148	1	CSR	Channel Select Register. Contained in bits 0-3 of the byte. Other bits zero.
149	1	LAR	Last Address Register. Contained in bits 1-3 of the byte.
150	1	IS	Interrupt switch. Set zero initially and subsequently by any Allow All I/O Interrupts instruction. Set 1 by a Prevent All I/O Interrupts instruction.
151	1	LPS	LPS Switch. Set 1 by the Load Processor State instruction. Set zero when the load is completed, after the execution of the next 1108 instruction.
152	5	new PSR	New PSR. Loaded by the LPS instruction. Transferred to PSR after execution of next instruction.
157	1	SL	Storage Limits violate flag. Bit 0 is set 1 if a storage limits violation is detected during effective address calculation. This does not imply that an interrupt will occur as storage protection may

			be disabled. Bit 1 is set zero if BI is used in effective address calculation set 1 if BD used. Bit 3 is set 1 if an invalid address is computed - that is, outside the range of the simulated machine.
160	4	EFA/360	The /360 address equivalent to EFA.
164	2	ISI/ESI	16 flags indicating corresponding channel is either ISI (bit set 0) or ESI (bit set 1).
168	4	HS	The /360 address of the hidden storage origin.
172	4	UL	Machine upper limit. The highest valid 1108 address.
176	4	IU	I/O interrupt count. An integer count of the outstanding I/O interrupts - i.e. those still to be serviced.
180	4	LI	Last interrupt. Trap location (1108) of last interrupt handled.
184	16		Not used.

NB. The IRB contains some fields which are not used in the present version of the simulator.

When core storage is obtained for the simulated 1108, register 4 is loaded with the /360 address of location zero. To find the /360 address of an 1108 word whose address is E:

If $E < 200_8$ and refers to a control register then

$I = E*5 + LOC0$

If $E < 200_8$ and refers to hidden storage then

$I = E*5 + HS$

If $200_8 \leq E \leq 7777_8$ then

$I = E*5 + LOC0$

If $100000_8 \leq E \leq 107777_8$ then

$I = E*5 - 28672 + LOC0$

where I = /360 address

E = 1108 address

LOC0 = /360 address of location zero of the 1108

HS = /360 address of location zero of 1108 hidden storage.

240 bytes of work storage are permanently addressed by register 5. This area is used by subroutines for scratch work.

PROGRAM EXECUTION. The flow chart in Figure 3.3.2 provides a general outline of SIMPEX logic.

The program is terminated when an ER ,077 is executed. "NORMAL END" is printed.

(b) Machine Instruction Subprograms.

These subroutines are not discussed individually. Each

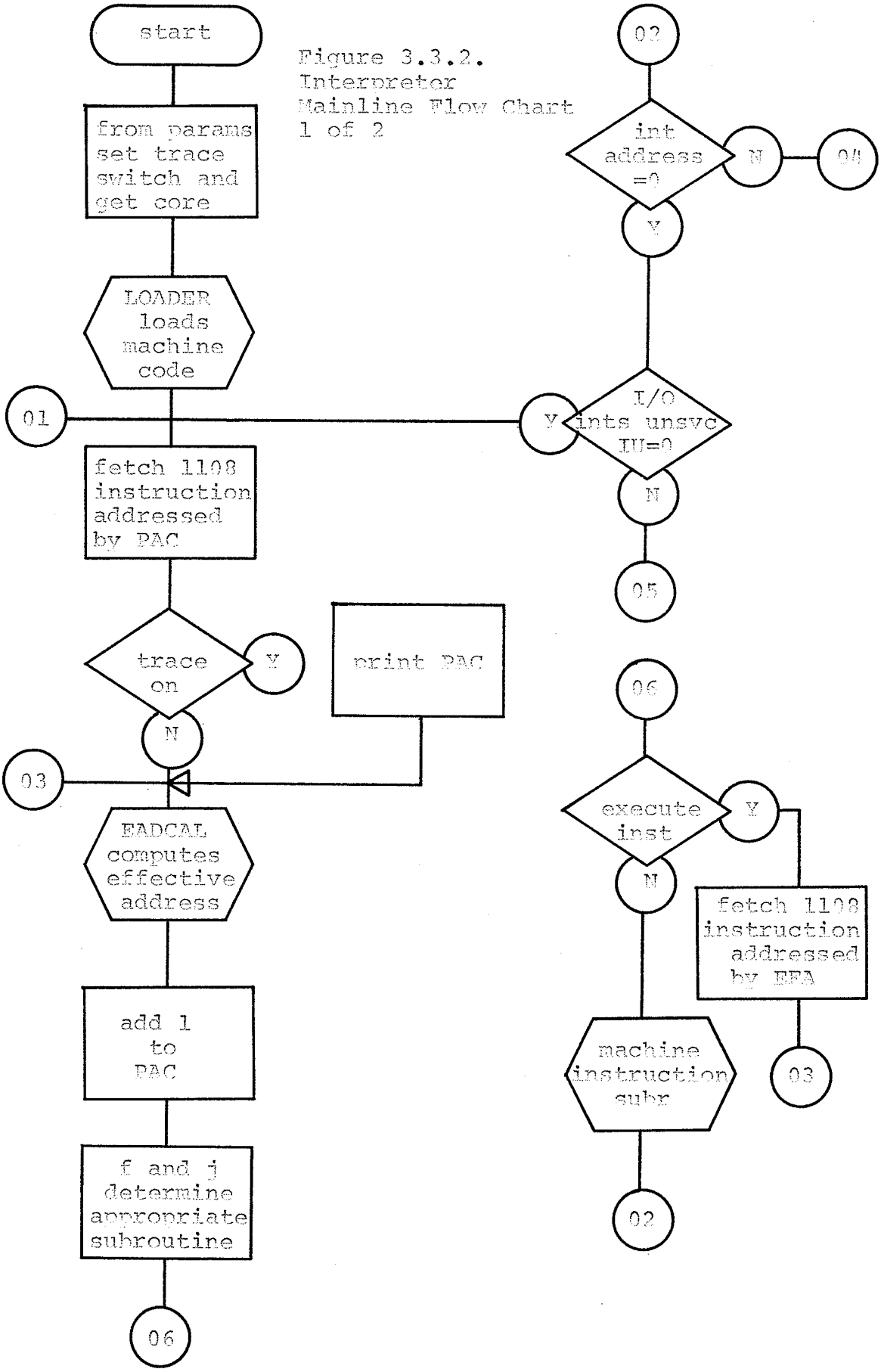


Figure 3.3.2.
Interpreter
Mainline Flow Chart
1 of 2

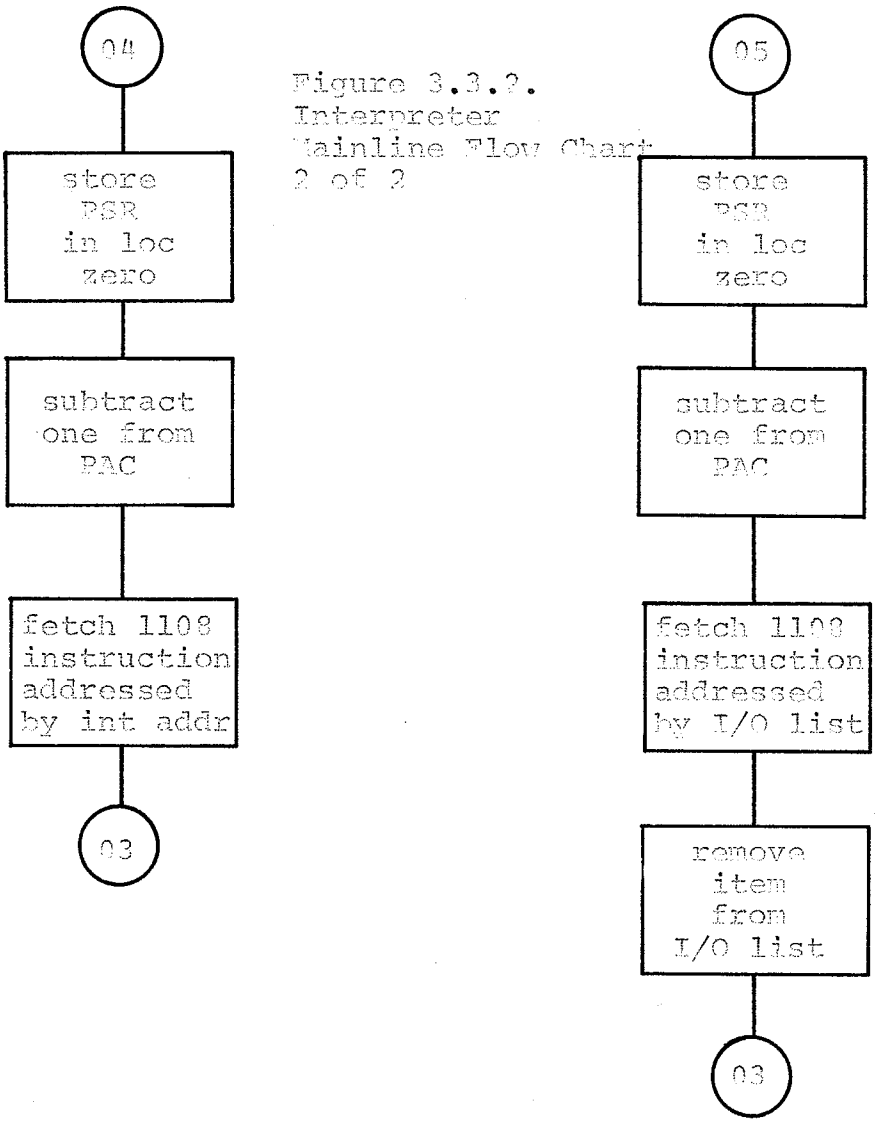


Figure 3.3.2.
Interpreter
Mainline Flow Chart
2 of 2

is unique in that it performs a certain 1108 function such as loading, storing or shifting. The routines are written to a general pattern (Figure 3.3.3).

To alter any single instruction routine

i) refer to Table 3.3.1 to find the library module that contains the routine.

ii) make the necessary alterations.

iii) re-compile the library module, linking into the system library RLMACL.A0224.RACLIB.

iv) link edit the system into the same library. Remember that two versions of the system exist - LEX1108 and LEY1108.

(c) System Function Subprograms.

Each system function subprogram may be called from either the mainline or machine instruction subprogram. No parameters are passed between these routines as the IRB and WSA are used as common areas.

LOADER - Loads relocatable code from a system data set, assigns absolute addresses and establishes program entry point.

Initially 1108 locations 200_8-247_8 and 252_8 are loaded with the instruction

```
SLJ ,*0250
```

Location 251_8 is loaded with a dump instruction.

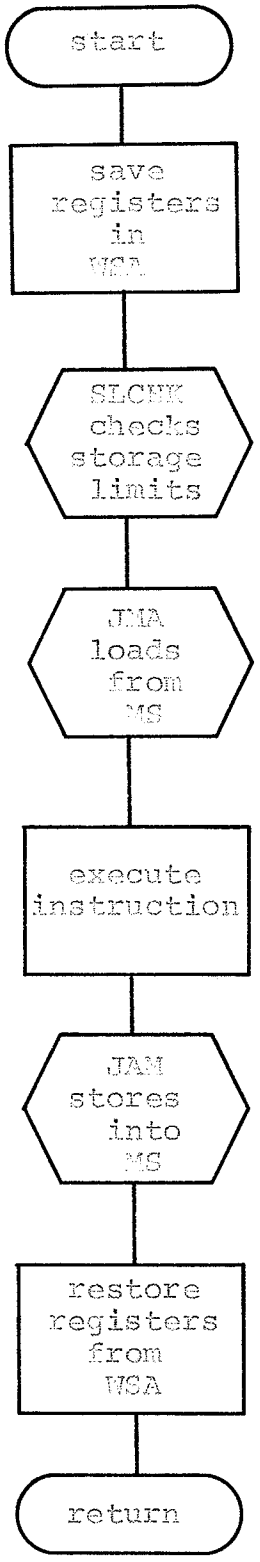


Figure 3.3.3.
Machine Instruction
Subprogram Flow Chart

Relocatable code is read from file SYSUT2. Absolute addresses are assigned equal to the relocatable addresses in the file. Figure 3.1.2 shows SYSUT2 record format. An entry address record loads the P-register.

EADCAL - Computes the effective address in the current instruction.

The Flow Chart of effective address calculation is in Figure 3.3.4.

SLCHK - Tests the status of the Guard Mode/Storage Limits indicators in the PSR and if enabled tests for a violation.

EADCAL has previously set violation flags.

There are two entry points to SLCHK.

i) For instructions that read only from main storage.

ii) For instructions that write into main storage.

Violation flags in the IRB and D3,D2 settings in the PSR are examined. A storage limits interrupt is effected by loading 243 into IA in the IRB.

MDUMP - Constructs and prints one line of data - being the contents of control registers 14 - 17 and absolute locations 512 - 515 (mini-dump).

PRINTX - Performs all I/O for system print file.

Opens, closes and writes to print file.

DUMPX - Constructs and prints an 1108 core dump.

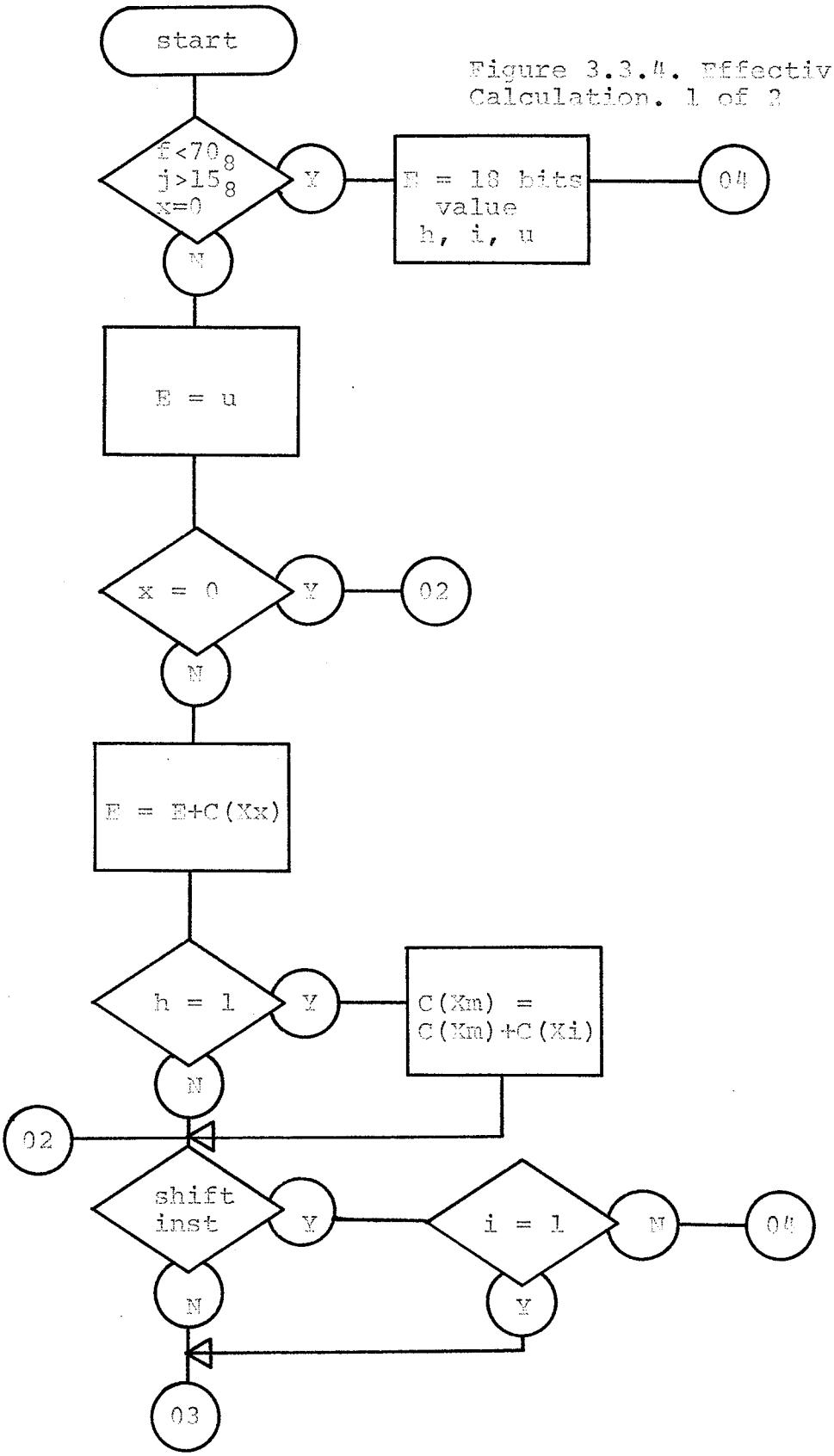


Figure 3.3.4. Effective Address Calculation. 1 of 2

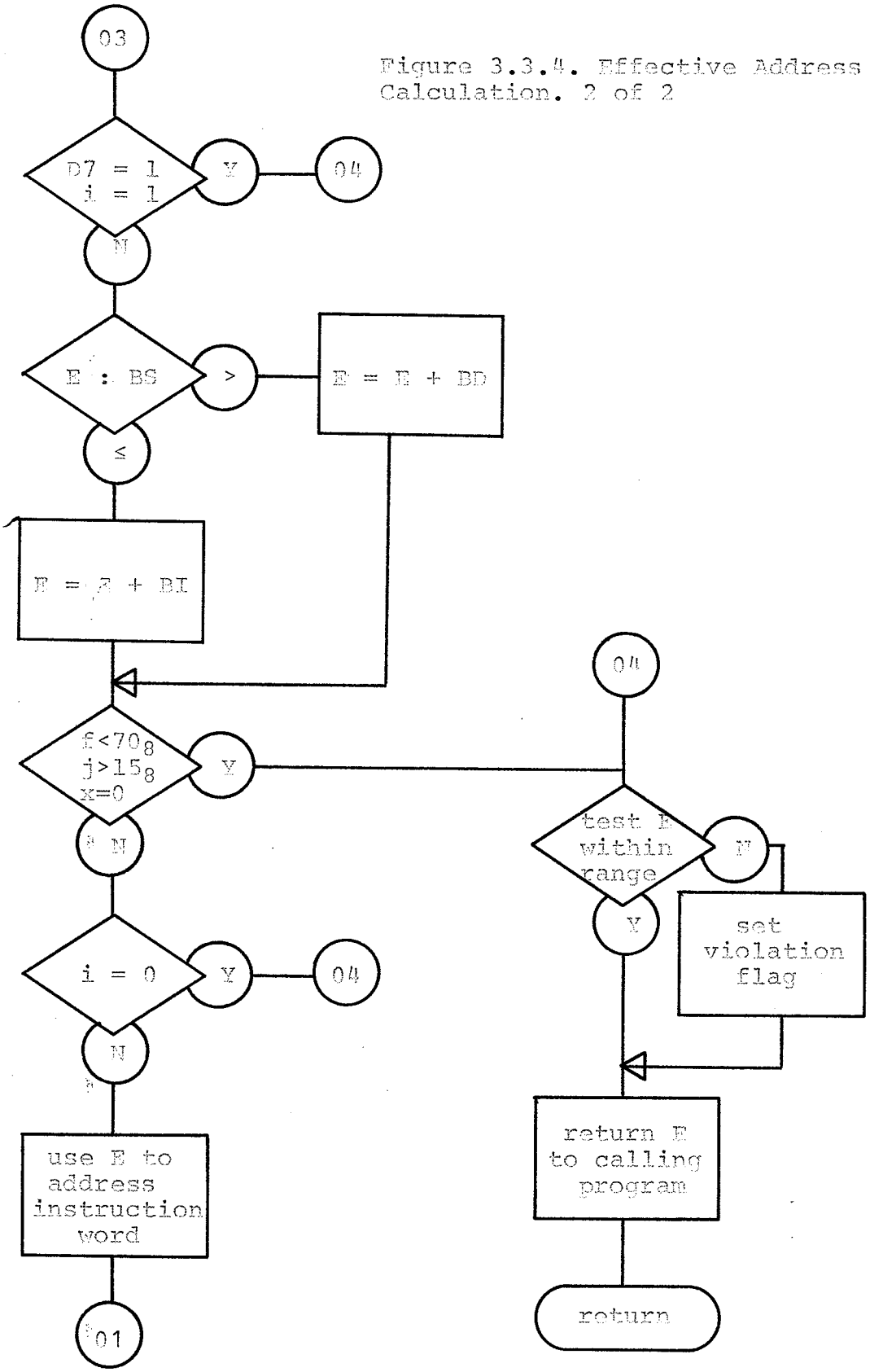


Figure 3.3.4. Effective Address Calculation. 2 of 2

Data from 1108 internal registers and main storage are formatted as shown in Figure 2.5.1.

JAM - Transfers data from the 1108 arithmetic control unit to main storage under j-control.

The data transfer (see Table 2.1.2) is effected by

(a) AND-ing off the bits of the main storage word which are unchanged by the transfer.

then (b) OR-ing in the specified bits from the control unit word.

JMA - Transfers data from the 1108 main storage to the arithmetic control unit under j-control.

The transfer is accomplished by

(a) AND-ing off the bits of the main storage word which are to be transferred.

(b) Moving the word to the ACU, shifting right if necessary and left filling with either zeros or signs.

SUBT - Perform subtraction and addition, i.e. to simulate the action of the 1108 ones complement subtractive adder.

For subtraction: The ones complement of the subtrahend is taken and then the logic proceeds as for addition.

For addition: The overflow and carry indicators in the PSR (D1 and D0) are cleared.

If either the augend or the addend is negative then it is converted to a twos complement negative value.

The addition is performed using /360 multiple register arithmetic.

If the result is negative then it is converted to a ones complement negative value.

The signs of the augend, addend and result are then examined and the overflow and/or carry indicators set if necessary.

DIVIDE - Performs integer division. Division takes place from left to right and thus makes both divide integer and divide fractional possible.

If either of the arguments is negative then it is complemented.

Divide exception checks are made.

The quotient is set to +0.

The following operations are performed 35 times.

(a) the 72-bit dividend is left shifted logically 1 bit position.

(b) the quotient is left shifted logically 1 bit position.

(c) the leftmost 36 bits of the dividend are treated as an integer value and this value is compared with the divisor. If it is greater than or equal to the divisor then the divisor is subtracted from it and a 1 bit is stored in the rightmost bit of the quotient. Otherwise neither this value nor the quotient is altered.

The quotient is then complemented if the signs of the

original dividend and divisor are unequal. The remainder is given the sign of the dividend.

MULT - Performs integer ones complement multiplication.

If either argument is negative it is complemented and the arithmetic performed with only positive values.

Each argument is treated as:

Multiplicand = $a1*2^{18} + a2$

Multiplier = $b1*2^{18} + b2$

The result is then computed by:-

If a1 = b1 = 0 then result = a2*b2

If a1 = 0 then result = $a2*b1*2^{18} + a2*b2$

If b1 = 0 then result = $a1*b2*2^{18} + a2*b2$

Otherwise result = $(a1*b1*2^{18} + a1*b2 + a2*b1)2^{18} + a2*b2$

The result is then complemented if the signs of the multiplicand and multiplier are unequal.

UICCON - Translates 6-bit ASCII into EBCDIC.

Either a single or double 1108 word is converted to a 6 or 12 byte character string.

Chapter IV SAMPLE PROBLEMS.

1. Integer Random Number Generator.

A program is developed to generate integer random numbers between 0 and 9. It is tested separately for use later as a subroutine. The algorithm requires 36-bit integer arithmetic and hence is not normally run on the /360.

The starting value is an odd integer in location STP. The value in this word should not be altered between random number generations. Each random number is stored in RAND.

Note that the routine is written to make use of the mini-dump feature at execution time. Accumulators A2-A5 are used for arithmetic and main storage locations 512-514 contain STP, RAND and a counter. The print illustrates how the trace and mini-dump together provide an excellent record of program execution.

```
//RLM92 JOB '0299,XXX,1,T=20,I=20,C=0,L=2,R=42','R MACMILLAN'          JOB 229
//JOBLIB DD DSN=RLMACL.A0224.RACLIB,UNIT=SYSDA,
//     DISP=SHR,VOLUME=SER=UM1404
//CM EXEC PGM=ASM11D8,REGION=42K
//SYSUT1 DD DSN=RLMACL.A0224.SYSUT1,UNIT=SYSDA,
//     VOLUME=REF=ONE.MONTH,DISP=(NEW,DELETE),SPACE=(TRK,(1,1),RLSE),
//    DCB=(RECFM=F,BLKSIZE=84,LRECL=84)
//SYSUT2 DD DSN=RLMACL.A0224.SYSUT21,UNIT=SYSDA,
//      VOLUME=REF=ONE.MONTH,DISP=(NEW,KEEP),SPACE=(TRK,(1,1),RLSE),
//     DCB=(RECFM=FB,BLKSIZE=160,LRECL=16)
//SYSUT3 DD DSN=RLMACL.A0224.SYSUT3,UNIT=SYSDA,
//      VOLUME=REF=ONE.MONTH,SPACE=(TRK,(1,1),RLSE),
//      DISP=(NEW,DELETE),DCB=(RECFM=FB,BLKSIZE=800,LRECL=80)
//SYSPRINT DD SYSOUT=A
//SYSUDUMP DD SYSOUT=A
//SYSIN DD *
IEF373I STEP /CM       / START 71188.0903
IEF374I STEP /CM       / STOP  71188.0904 CPU    0MIN 00.65SEC MAIN  42K LCS   0K
CM           17.26 SEC EXEC TIME     0.65 SEC CPU TIME      290 I/O COUNTS  42K REGION  42K USED

IEF375I  JOB /RLM92    / START 71188.0903
IEF376I  JOB /RLM92    / STOP  71188.0904 CPU    0MIN 00.65SEC
RLM92        17.26 SEC EXEC TIME     0.65 SEC CPU TIME       DATE 71.188 UNIVERSITY OF MANITOBA
```

```
 HASP-II JOB STATISTICS --      44 CARDS READ --        55 LINES PRINTED --       0 CARDS PUNCHED --    0.28 MINUTES EXECUTION TIME
 UNIVERSITY OF MANITOBA --     307 I/O COUNTS --       342 K BYTE-SECONDS -     0.62 UNITS         --    0.01 MINUTES CPU TIME
 ONLINE ACCOUNTING INFO --     556 JOB(S) RUN --
```

```
ERROR FLAGS    ADDR  OBJECT CODE  F  J  A  X  H1   U        SOURCE CODE

                                                    $(1)      RES   0500-$  . RELATIVE LOCATION IN MAIN STORAGE
                                                    LP        EQU   $
000500 100040001002 10 00 02 00 00 001002                     LA    A2,STP . LOAD GENERATOR STARTING VALUE
000501 300040000522 30 00 02 00 00 000522                     MI    A2,(3125) . MULTIPLY BY 3125 AND DIVIDE BY 67108864
000502 340040001006 34 00 02 00 00 001006                     DI    A2,LIMIT
000503 010060001002 01 00 03 00 00 001002                     SA    A3,STP . STORE NEW GENERATOR VALUE
000504 100100000520 10 00 04 00 00 000520                     L     A4,TOR . LOAD HIGH LIMIT OF RANGE
000505 150100000521 15 00 04 00 00 000521                     AN    A4,BOR . SUBTRACT LOW LIMIT
000506 300100001002 30 00 04 00 00 001002                     MI    A4,STP . MAP ONTO RANGE
000507 340100001006 34 00 04 00 00 001006                     DI    A4,LIMIT
000510 140100000521 14 00 04 00 00 000521                     A     A4,BOR . ADD LOW LIMIT
000511 010100001003 01 00 04 00 00 001003                     S     A4,RAND . STORE RANDOM NUMBER
000512 100140001004 10 00 06 00 00 001004                     L     A6,CNT . COUNTER
000513 140140000523 14 00 06 00 00 000523                     A     A6,(1) . INCREMENT
000514 010140001004 01 00 06 00 00 001004                     S     A6,CNT
000515 520140001005 52 00 06 00 00 001005                     TE    A6,NN . TEST FOR END OF EXECUTION
000516 742000000500 74 04 00 00 00 000500                     J     LP
000517 724400000077 72 11 00 00 00 000077                     ER    ,077 . HALT
000520 000000000012 00 00 00 00 00 000012 TOR                 10 . UPPER LIMIT ON RANGE
000521 000000000000 00 00 00 00 00 000000 BOR                 0 . LOWER LIMIT ON RANGE
                                                    $(3)      RES   01002-$
001002 000000363171 00 00 00 00 01 163171 STP                 124537 . STARTING VALUE FOR GEN
001003 000000000000 00 00 00 00 00 000000 RAND                RES   1
001004 000000000000 00 00 00 00 00 000000 CNT                 0
001005 000000000012 00 00 00 00 00 000012 NN                  10  . NUMBER OF RANDOM NUMBERS REQUIRED
001006 000400000000 00 01 00 00 00 000000 LIMIT               67108864
777777 000000000500 00 00 00 00 00 000500                   END 0500
```

```
//RLM92 JOB '0299,XXX,1,T=20,I=20,C=0,L=2,R=78','R MACMILLAN'          JOB 357
//JOBLIB DD DSN=RLMACL.A0224.RACLIB,UNIT=SYSDA,
//     DISP=SHR,VOLUME=SER=UM1404
//GO EXEC PGM=LEY11J8,PARM='MS=36863,TRACE=YES',REGION=78K
//GO.SYSUT2 DD DSN=RLMACL.A0224.SYSUT21,DISP=SHR,UNIT=SYSDA,
//    VOLUME=REF=ONE.MONTH,DCB=(RECFM=FB,LRECL=16,BLKSIZE=160)
//SYSPRINT DD SYSOUT=A
//SYSUDUMP DD SYSOUT=A
//SYSIN DD *
IEF373I STEP /GO       / START 71188.1210
IEF374I STEP /GO       / STOP  71188.1210 CPU    0MIN 00.84SEC MAIN  78K LCS   0K
GO          11.38 SEC EXEC TIME      0.84 SEC CPU TIME      118 I/O COUNTS  78K REGION  78K USED

IEF375I  JOB /RLM92    / START 71188.1210
IEF376I  JOB /RLM92    / STOP  71188.1210 CPU    0MIN 00.84SEC
RLM92       11.38 SEC EXEC TIME      0.84 SEC CPU TIME        DATE 71.188 UNIVERSITY OF MANITOBA


 HASP-II JOB STATISTICS --      10 CARDS READ --       321 LINES PRINTED --       0 CARDS PUNCHED --    0.18 MINUTES EXECUTION TIME
 UNIVERSITY OF MANITOBA --     135 I/O COUNTS --       456 K BYTE-SECONDS -     0.56 UNITS        --    0.01 MINUTES CPU TIME
 ONLINE ACCOUNTING INFO --     573 JOB(S) RUN --
```

```
000500
000000363171  041510030400  600123220061  031400446560  000000363171  000000000000  000000000000  000000000012
000501
000000000000  002714461415  600123220061  031400446560  000000363171  000000000000  000000000000  000000000012
000502
000000000005  000314461415  600123220061  031400446560  000000363171  000000000000  000000000000  000000000012
000503
000000000005  000314461415  600123220061  031400446560  000314461415  000000000000  000000000000  000000000012
000504
000000000005  000314461415  000000000012  031400446560  000314461415  000000000000  000000000000  000000000012
000505
000000000005  000314461415  000000000012  031400446560  000314461415  000000000000  000000000000  000000000012
000506
000000000005  000314461415  000000000000  003775757202  000314461415  000000000000  000000000000  000000000012
000507
000000000005  000314461415  000000000007  000375757202  000314461415  000000000000  000000000000  000000000012
000510
000000000005  000314461415  000000000007  000375757202  000314461415  000000000000  000000000000  000000000012
000511
000000000005  000314461415  000000000007  000375757202  000314461415  000000000007  000000000000  000000000012
000512
000000000005  000314461415  000000000007  000375757202  000314461415  000000000007  000000000000  000000000012
000513
000000000005  000314461415  000000000007  000375757202  000314461415  000000000007  000000000000  000000000012
000514
000000000005  000314461415  000000000007  000375757202  000314461415  000000000007  000000000001  000000000012
000515
000000000005  000314461415  000000000007  000375757202  000314461415  000000000007  000000000001  000000000012
000516
000000000005  000314461415  000000000007  000375757202  000314461415  000000000007  000000000001  000000000012
000500
000314461415  000314461415  000000000007  000375757202  000314461415  000000000007  000000000001  000000000012
000501
000000000002  340304616661  000000000007  000375757202  000314461415  000000000007  000000000001  000000000012
000502
000000004701  000204616661  000000000007  000375757202  000314461415  000000000007  000000000001  000000000012
000503
000000004701  000204616661  000000000007  000375757202  000204616661  000000000007  000000000001  000000000012
000504
000000004701  000204616661  000000000012  000375757202  000204616661  000000000007  000000000001  000000000012
000505
000000004701  000204616661  000000000012  000375757202  000204616661  000000000007  000000000001  000000000012
000506
000000004701  000204616661  000000000000  002457624352  000204616661  000000000007  000000000001  000000000012
000507
000000004701  000204616661  000000000005  000057624352  000204616661  000000000007  000000000001  000000000012
000510
000000004701  000204616661  000000000005  000057624352  000204616661  000000000007  000000000001  000000000012
000511
```

Locations dumped in "mini-dump" are octal addresses:-

16	17	20	21	1002	1003	1004	1005
i.e							
A2	A3	A4	A5	STP	RAND	CNT	MN

000000004701 000512	000204616661	000000000005	000057624352	000204616661	000000000005	000000000001	000000000012
000000004701 000513	000204616661	000000000005	000057624352	000204616661	000000000005	000000000001	000000000012
000000004701 000514	000204616661	000000000005	000057624352	000204616661	000000000005	000000000001	000000000012
000000004701 000515	000204616661	000000000005	000057624352	000204616661	000000000005	000000000002	000000000012
000000004701 000516	000204616661	000000000005	000057624352	000204616661	000000000005	000000000002	000000000012
000000004701 000500	000204616661	000000000005	000057624352	000204616661	000000000005	000000000002	000000000012
000204616661 000501	000204616661	000000000005	000057624352	000204616661	000000000005	000000000002	000000000012
000000000001 000502	452326270645	000000000005	000057624352	000204616661	000000000005	000000000002	000000000012
000000003124 000503	000326270645	000000000005	000057624352	000204616661	000000000005	000000000002	000000000012
000000003124 000504	000326270645	000000000005	000057624352	000326270645	000000000005	000000000002	000000000012
000000003124 000505	000326270645	000000000012	000057624352	000326270645	000000000005	000000000002	000000000012
000000003124 000506	000326270645	000000000012	000057624352	000326270645	000000000005	000000000002	000000000012
000000003124 000507	000326270645	000000000000	004137470162	000326270645	000000000005	000000000002	000000000012
000000003124 000510	000326270645	000000000010	000137470162	000326270645	000000000005	000000000002	000000000012
000000003124 000511	000326270645	000000000010	000137470162	000326270645	000000000005	000000000002	000000000012
000000003124 000512	000326270645	000000000010	000137470162	000326270645	000000000010	000000000002	000000000012
000000003124 000513	000326270645	000000000010	000137470162	000326270645	000000000010	000000000002	000000000012
000000003124 000514	000326270645	000000000010	000137470162	000326270645	000000000010	000000000002	000000000012
000000003124 000515	000326270645	000000000010	000137470162	000326270645	000000000010	000000000003	000000000012
000000003124 000516	000326270645	000000000010	000137470162	000326270645	000000000010	000000000003	000000000012
000000003124 000500	000326270645	000000000010	000137470162	000326270645	000000000010	000000000003	000000000012
000326270645 000501	000326270645	000000000010	000137470162	000326270645	000000000010	000000000003	000000000012
000000000002 000502	434266041451	000000000010	000137470162	000326270645	000000000010	000000000003	000000000012
000000005070 000503	000266041451	000000000010	000137470162	000326270645	000000000010	000000000003	000000000012
000000005070	000266041451	000000000010	000137470162	000266041451	000000000010	000000000003	000000000012

A new random number is stored in 1003 after the execution of the instruction at Main Storage location 511 octal.

000504							
000000005070	000266041451	000000000012	000137470162	000266041451	000000000010	000000000003	000000000012
000505							
000000005070	000266041451	000000000012	000137470162	000266041451	000000000010	000000000003	000000000012
000506							
000000005070	000266041451	000000000000	003434517632	000266041451	000000000010	000000000003	000000000012
000507							
000000005070	000266041451	000000000007	000034517632	000266041451	000000000010	000000000003	000000000012
000510							
000000005070	000266041451	000000000007	000034517632	000266041451	000000000010	000000000003	000000000012
000511							
000000005070	000266041451	000000000007	000034517632	000266041451	000000000007	000000000003	000000000012
000512							
000000005070	000266041451	000000000007	000034517632	000266041451	000000000007	000000000003	000000000012
000513							
000000005070	000266041451	000000000007	000034517632	000266041451	000000000007	000000000003	000000000012
000514							
000000005070	000266041451	000000000007	000034517632	000266041451	000000000007	000000000004	000000000012
000515							
000000005070	000266041451	000000000007	000034517632	000266041451	000000000007	000000000004	000000000012
000516							
000000005070	000266041451	000000000007	000034517632	000266041451	000000000007	000000000004	000000000012
000500							
000266041451	000266041451	000000000007	000034517632	000266041451	000000000007	000000000004	000000000012
000501							
000000000002	127172751575	000000000007	000034517632	000266041451	000000000007	000000000004	000000000012
000502							
000000004256	000172751575	000000000007	000034517632	000266041451	000000000007	000000000004	000000000012
000503							
000000004256	000172751575	000000000007	000034517632	000172751575	000000000007	000000000004	000000000012
000504							
000000004256	000172751575	000000000012	000034517632	000172751575	000000000007	000000000004	000000000012
000505							
000000004256	000172751575	000000000012	000034517632	000172751575	000000000007	000000000004	000000000012
000506							
000000004256	000172751575	000000000000	002315441342	000172751575	000000000007	000000000004	000000000012
000507							
000000004256	000172751575	000000000004	000315441342	000172751575	000000000007	000000000004	000000000012
000510							
000000004256	000172751575	000000000004	000315441342	000172751575	000000000007	000000000004	000000000012
000511							
000000004256	000172751575	000000000004	000315441342	000172751575	000000000004	000000000004	000000000012
000512							
000000004256	000172751575	000000000004	000315441342	000172751575	000000000004	000000000004	000000000012
000513							
000000004256	000172751575	000000000004	000315441342	000172751575	000000000004	000000000004	000000000012
000514							
000000004256	000172751575	000000000004	000315441342	000172751575	000000000004	000000000005	000000000012
000515							

000000004256 000516	000172751575	000000000004	000315441342	000172751575	000000000004	000000000005	000000000012
000000004256 000500	000172751575	000000000004	000315441342	000172751575	000000000004	000000000005	000000000012
000172751575 000501	000172751575	000000000004	000315441342	000172751575	000000000004	000000000005	000000000012
000000000001 000502	356357122341	000000000004	000315441342	000172751575	000000000004	000000000005	000000000012
000000002734 000503	000357122341	000000000004	000315441342	000172751575	000000000004	000000000005	000000000012
000000002734 000504	000357122341	000000000004	000315441342	000357122341	000000000004	000000000005	000000000012
000000002734 000505	000357122341	000000000012	000315441342	000357122341	000000000004	000000000005	000000000012
000000002734 000506	000357122341	000000000012	000315441342	000357122341	000000000004	000000000005	000000000012
000000002734 000507	000357122341	000000000000	004527470312	000357122341	000000000004	000000000005	000000000012
000000002734 000510	000357122341	000000000011	000127470312	000357122341	000000000004	000000000005	000000000012
000000002734 000511	000357122341	000000000011	000127470312	000357122341	000000000004	000000000005	000000000012
000000002734 000512	000357122341	000000000011	000127470312	000357122341	000000000011	000000000005	000000000012
000000002734 000513	000357122341	000000000011	000127470312	000357122341	000000000011	000000000005	000000000012
000000002734 000514	000357122341	000000000011	000127470312	000357122341	000000000011	000000000005	000000000012
000000002734 000515	000357122341	000000000011	000127470312	000357122341	000000000011	000000000006	000000000012
000000002734 000516	000357122341	000000000011	000127470312	000357122341	000000000011	000000000006	000000000012
000000002734 000500	000357122341	000000000011	000127470312	000357122341	000000000011	000000000006	000000000012
000357122341 000501	000357122341	000000000011	000127470312	000357122341	000000000011	000000000006	000000000012
000000000002 000502	663562127225	000000000011	000127470312	000357122341	000000000011	000000000006	000000000012
000000005547 000503	000162127225	000000000011	000127470312	000357122341	000000000011	000000000006	000000000012
000000005547 000504	000162127225	000000000011	000127470312	000162127225	000000000011	000000000006	000000000012
000000005547 000505	000162127225	000000000012	000127470312	000162127225	000000000011	000000000006	000000000012
000000005547 000506	000162127225	000000000012	000127470312	000162127225	000000000011	000000000006	000000000012
000000005547 000507	000162127225	000000000000	002165550722	000162127225	000000000011	000000000006	000000000012
000000005547	000162127225	000000000004	000165550722	000162127225	000000000011	000000000006	000000000012

000510
000000005547 000162127225 000000000004 000165550722 000162127225 000000000011 000000000006 000000000012
000511
000000005547 000162127225 000000000004 000165550722 000162127225 000000000004 000000000006 000000000012
000512
000000005547 000162127225 000000000004 000165550722 000162127225 000000000004 000000000006 000000000012
000513
000000005547 000162127225 000000000004 000165550722 000162127225 000000000004 000000000006 000000000012
000514
000000005547 000162127225 000000000004 000165550722 000162127225 000000000004 000000000007 000000000012
000515
000000005547 000162127225 000000000004 000165550722 000162127225 000000000004 000000000007 000000000012
000516
000000005547 000162127225 000000000004 000165550722 000162127225 000000000004 000000000007 000000000012
000500
000162127225 000162127225 000000000004 000165550722 000162127225 000000000004 000000000007 000000000012
000501
000000000001 270656620331 000000000004 000165550722 000162127225 000000000004 000000000007 000000000012
000502
000000002561 000256620331 000000000004 000165550722 000162127225 000000000004 000000000007 000000000012
000503
000000002561 000256620331 000000000004 000165550722 000256620331 000000000004 000000000007 000000000012
000504
000000002561 000256620331 000000000012 000165550722 000256620331 000000000004 000000000007 000000000012
000505
000000002561 000256620331 000000000012 000165550722 000256620331 000000000004 000000000007 000000000012
000506
000000002561 000256620331 000000000000 003323644172 000256620331 000000000004 000000000007 000000000012
000507
000000002561 000256620331 000000000006 000323644172 000256620331 000000000004 000000000007 000000000012
000510
000000002561 000256620331 000000000006 000323644172 000256620331 000000000004 000000000007 000000000012
000511
000000002561 000256620331 000000000006 000323644172 000256620331 000000000006 000000000007 000000000012
000512
000000002561 000256620331 000000000006 000323644172 000256620331 000000000006 000000000007 000000000012
000513
000000002561 000256620331 000000000006 000323644172 000256620331 000000000006 000000000007 000000000012
000514
000000002561 000256620331 000000000006 000323644172 000256620331 000000000006 000000000010 000000000012
000515
000000002561 000256620331 000000000006 000323644172 000256620331 000000000006 000000000010 000000000012
000516
000000002561 000256620331 000000000006 000323644172 000256620331 000000000006 000000000010 000000000012
000500
000256620331 000256620331 000000000006 000323644172 000256620331 000000000006 000000000010 000000000012
000501
000000000002 052621774355 000000000006 000323644172 000256620331 000000000006 000000000010 000000000012
000502

000000004125 000503	000221774355	000000000006	000323644172	000256620331	000000000005	000000000010	000000000012
000000004125 000504	000221774355	000000000006	000323644172	000221774355	000000000006	000000000010	000000000012
000000004125 000505	000221774355	000000000012	000323644172	000221774355	000000000006	000000000010	000000000012
000000004125 000506	000221774355	000000000012	000323644172	000221774355	000000000006	000000000010	000000000012
000000004125 000507	000221774355	000000000000	002663734502	000221774355	000000000006	000000000010	000000000012
000000004125 000510	000221774355	000000000005	000263734502	000221774355	000000000006	000000000010	000000000012
000000004125 000511	000221774355	000000000005	000263734502	000221774355	000000000006	000000000010	000000000012
000000004125 000512	000221774355	000000000005	000263734502	000221774355	000000000005	000000000010	000000000012
000000004125 000513	000221774355	000000000005	000263734502	000221774355	000000000005	000000000010	000000000012
000000004125 000514	000221774355	000000000005	000263734502	000221774355	000000000005	000000000010	000000000012
000000004125 000515	000221774355	000000000005	000263734502	000221774355	000000000005	000000000011	000000000012
000000004125 000516	000221774355	000000000005	000263734502	000221774355	000000000005	000000000011	000000000012
000000004125 000500	000221774355	000000000005	000263734502	000221774355	000000000005	000000000011	000000000012
000221774355 000501	000221774355	000000000005	000263734502	000221774355	000000000005	000000000011	000000000012
000000000001 000502	573044322421	000000000005	000263734502	000221774355	000000000005	000000000011	000000000012
000000003366 000503	000044322421	000000000005	000263734502	000221774355	000000000005	000000000011	000000000012
000000003366 000504	000044322421	000000000005	000263734502	000044322421	000000000005	000000000011	000000000012
000000003366 000505	000044322421	000000000012	000263734502	000044322421	000000000005	000000000011	000000000012
000000003366 000506	000044322421	000000000012	000263734502	000044322421	000000000005	000000000011	000000000012
000000003366 000507	000044322421	000000000000	000554071252	000044322421	000000000005	000000000011	000000000012
000000003366 000510	000044322421	000000000001	000154071252	000044322421	000000000005	000000000011	000000000012
000000003366 000511	000044322421	000000000001	000154071252	000044322421	000000000005	000000000011	000000000012
000000003366 000512	000044322421	000000000001	000154071252	000044322421	000000000001	000000000011	000000000012
000000003366 000513	000044322421	000000000001	000154071252	000044322421	000000000001	000000000011	000000000012
000000003366	000044322421	000000000001	000154071252	000044322421	000000000001	000000000011	000000000012

```
000514
000000003366  000044322421  000000000001  000154071252  000044322421  000000000001  000000000012  000000000012
000515
000000003366  000044322421  000000000001  000154071252  000044322421  000000000001  000000000012  000000000012
000517
000000003366  000044322421  000000000001  000154071252  000044322421  000000000001  000000000012  000000000012
NORMAL END
```

2. Executive.

This sample illustrates how an Executive system may be written. The program is divided into two sections -

(a) the executive part which handles ER interrupts and does all I/O.

(b) the user part which requests I/O, and performs some elementary character manipulations.

The instrucion normally loaded at 242_8 is replaced with a Store Location and Jump to ERRTN. This means that whenever an Executive Return interrupt occurs control is handed to the ERRTN routine. The u-field of the ER instruction that caused the interrupt is examined. $u=10_8$ requests a read operation, 17_8 a write, 77_8 - terminate job. User index register 1 contains the address of the I/O field.

Note: ER ,077 gives normal end of job. However, as the usual ER handler has been overridden, it is necessary to code an instruction with $f = 72_8$, $j = 0$ (at location 310_8) to terminate the job. There is no mnemonic that gives this instruction.

The "User" program requests a card read. The data input are then examined for the string "ARG" on a halfword boundary. Each card is printed and a message indicates cards in which the search is successful.

The trace feature is employed to show how control passes back and forth from "Executive" to "User".

```
//RLM92 JOB '0299,XXX,1,T=20,I=20,C=0,L=2,R=78','R MACMILLAN'          JOB 235
//JOBLIB DD DSN=RLMACL.A0224.RACLIB,UNIT=SYSDA,
//    DISP=SHR,VOLUME=SER=UM1404
//CM EXEC PGM=ASM1108,REGION=42K
//SYSUT1 DD DSN=RLMACL.A0224.SYSUT1,UNIT=SYSDA,
//    VOLUME=REF=ONE.MONTH,DISP=(NEW,DELETE),SPACE=(TRK,(1,1),RLSE),
//   DCB=(RECFM=F,BLKSIZE=84,LRECL=84)
//SYSUT2 DD DSN=RLMACL.A0224.SYSUT22,UNIT=SYSDA,
//     VOLUME=REF=ONE.MONTH,DISP=(NEW,PASS),SPACE=(TRK,(1,1),RLSE),
//    DCB=(RECFM=FB,BLKSIZE=160,LRECL=16)
//SYSUT3 DD DSN=RLMACL.A0224.SYSUT3,UNIT=SYSDA,
//     VOLUME=REF=ONE.MONTH,SPACE=(TRK,(1,1),RLSE),
//     DISP=(NEW,DELETE),DCB=(RECFM=FB,BLKSIZE=800,LRECL=80)
//SYSPRINT DD SYSOUT=A
//SYSUDUMP DD SYSOUT=A
//SYSIN DD *
IEF373I STEP /CM       / START 71188.0913
IEF374I STEP /CM       / STOP  71188.0914 CPU    0MIN 01.44SEC MAIN  42K LCS    0K
CM           22.60 SEC EXEC TIME      1.44 SEC CPU TIME      489 I/O COUNTS  42K REGION  42K USED

//GO EXEC PGM=LEX1108,PARM='MS=36863,TRACE=YES',REGION=78K
//GO.SYSUT2 DD DSN=RLMACL.A0224.SYSUT22,DISP=(OLD,DELETE),UNIT=2314,
//   VOLUME=REF=ONE.MONTH,DCB=(RECFM=FB,LRECL=16,BLKSIZE=160)
//SYSPRINT DD SYSOUT=A
//SYSUDUMP DD SYSOUT=A
//SYSIN DD *
IEF373I STEP /GO       / START 71188.0914
IEF374I STEP /GO       / STOP  71188.0914 CPU    0MIN 00.85SEC MAIN  78K LCS    0K
GO            6.02 SEC EXEC TIME      0.85 SEC CPU TIME      138 I/O COUNTS  78K REGION  78K USED

IEF375I  JOB /RLM92    / START 71188.0913
IEF376I  JOB /RLM92    / STOP  71188.0914 CPU    0MIN 02.29SEC
RLM92        28.62 SEC EXEC TIME      2.29 SEC CPU TIME        DATE 71.188 UNIVERSITY OF MANITOBA

 HASP-II JOB STATISTICS --     121 CARDS READ --       493 LINES PRINTED --       0 CARDS PUNCHED --    0.47 MINUTES EXECUTION TIME
 UNIVERSITY OF MANITOBA --     649 I/O COUNTS --     1,129 K BYTE-SECONDS -     1.74 UNITS         --    0.03 MINUTES CPU TIME
 CNLINE ACCOUNTING INFO --     561 JOB(S) RUN --
```

```
ERROR FLAGS    ADDR  OBJECT CODE  F  J  A  X  HI   U        SOURCE CODE

                                                    $(1)     RES   0242-$ . ER TRAPS TO LOCATION 242
                000242 720400200300 72 01 00 00 01 000300           SLJ   ,*ERRTN . JUMP TO ER HANDLER
                                                    $(2)     RES   0270-$ . INITIAL EXEC START LOCATION
                                                             LSL   ,(010410011010 4) . USER SLR LOAD ;
                                                    INSTRUCTION 100000-103777;
                000270 727000000273 72 16 00 00 00 000273 DATA       104000-107777
                000271 726400000274 72 15 00 00 00 000274           LPS   ,(004000177004) . LOAD USER PSR
                000272 743400100000 74 07 00 00 00 100000           AAIJ   ,0100000 . CONTROL TO USER
                                                    $(3)     RES   0300-$ . EXEC RETURN HANDLER
                000300 000000000000 00 00 00 00 00 000000 ERRTN     0 . P-REGISTER SAVE AREA
                000301 270060200300 27 00 03 00 01 000300           L     X3,*ERRTN . LOCATE USER ER INSTRUCTION
                000302 240060200326 24 00 03 00 01 000326           AX    X3,*USER . ADD USER START ADDRESS
                000303 703060200304 70 06 03 00 01 000304           JGD   99,*$+1 . C(EXEC X3) = A(USER ER INSTRUCTION)
                000304 270040200322 27 00 02 00 01 000322           L     X2,*INDEX . SET INDEX REGISTER
                000305 230020200323 23 00 01 00 01 000323           L     R1,*COUNT . REPEAT COUNTER
                000306 104043200000 10 10 02 03 01 000000           L,010 A2,*0,X3 . LOAD LOW CHAR OF USER ER INST
                000307 620042600324 62 00 02 02 11 000324           SE    A2,*ELIST,*X2 . SEARCH ER LIST
                000310 720000000000 72 00 00 00 00 000000           0720000000000 . END OF JOB
                000311 270100200001 27 00 04 00 01 000001           L     X4,*1 . LOAD USER STRING ADDRESS
                000312 702420200315 70 05 01 00 01 000315           JGD   81,*READ . A POSITIVE VALUE INDICATES READ
                000313 370004200000 37 00 00 04 01 000000           PUT   *0,X4 . EXECUTE WRITE
                000314 742000200316 74 04 00 00 01 000316           J     *END
                000315 330004200000 33 00 00 04 01 000000 READ      GET   *0,X4 . EXECUTE READ
                000316 270040200300 27 00 02 00 01 000300 END       L     X2,*ERRTN . GET RETURN ADDRESS
                000317 240040200326 24 00 02 00 01 000326           AX    X2,*USER . ADD USER START ADDRESS
                000320 726400200000 72 15 00 00 01 000000           LPS   ,*0 . RESTORE USER PSR
                000321 743402200000 74 07 00 02 01 000000           AAIJ  ,*0,X2 . RETURN TO USER
                000322 000001000000 00 00 00 01 00 000000 INDEX     01000000 . INITIAL VALUE INDEX REGISTER
                000323 000000000002 00 00 00 00 00 000002 COUNT     2 . NUMBER OF ELEMENTS ON ELIST
                000324 000000000010 00 00 00 00 00 000010 ELIST     010 . READ IDENTIFIER
                000325 000000000017 00 00 00 00 00 000017           017 . WRITE IDENTIFIER
                000326 000000000000 00 00 00 00 00 000000 USER      00000000 . USER START ADDRESS
```

```
ERROR FLAGS    ADDR  OBJECT CODE  F  J  A  X  HI   U        SOURCE CODE

                                                            /$(6)    RES   0100000-$ . USER PROGRAM;
                                                            LOGIC;
                                                            READS 6 CARDS AND SEARCHES EACH FOR THE ALPHABETIC STRING ARG;
                                                            ON ALL HALFWORD (THREE CHARACTER) BOUNDARIES;
                                                            WHEN FOUND A MESSAGE IS PRINTED
                100000 100040100112 10 00 C2 00 00 100112           L     A2,(6) . NUMBER OF CARDS TO BE READ
                100001 700340100003 70 00 16 00 00 100003 CLP       JGD   A2,$+2 . TEST FOR END OF JOB
                100002 7244C0000C77 72 11 C0 00 00 C00077           ER    ,077 . REQUEST EXEC TERMINATE JOB
                100003 230040100113 23 00 C2 00 00 100113           L     R2,(0777777000000) . SET MASK REGISTER
                100004 260020100114 26 00 01 00 00 100114           LXM   X1,(L A1,INAREA) . SET INDEX REGISTER FOR READ
                100005 724400000010 72 11 00 00 00 000010           ER    ,010 . REQUEST READ OPERATION FROM EXECUTIVE
                100006 260020100115 26 00 01 00 00 100115           LXM   X1,(L A1,INAREA-4) . SET INDEX REGISTER FOR WRITE
                100007 724400000017 72 11 00 00 00 000017           ER    ,017 . WRITE INPUT RECORD
                100010 100100100116 10 00 C4 00 00 100116           L     A4,('ARGARG') . SEARCH ARGUMENT
                                                          . NOTE THAT WHEN AN ARGUMENT IS FOUND THE SEARCH IS DISCONTINUED
                100011 230020100117 23 00 01 00 00 100117 ILP       L     R1,(12) . REPEAT COUNTER
                100012 270040100120 27 00 02 00 00 100120           LX    X2,(01000000) . INITIAL VALUE FOR INDEX REGISTER
                100013 710102500042 71 00 04 02 10 100042           MSE   A4,INAREA,*X2 . SEARCH INPUT RECORD
                100014 742000100017 74 04 00 00 00 100017           J     ILP2 . NOT FOUND
                100015 720400100026 72 01 00 00 00 100026           SLJ   ,ORTN . GIVE CONTROL TO OUTPUT ROUTINE
                100016 742000100001 74 04 00 00 00 100001           J     CLP  . ONLY ONE SUCCESSFUL SEEK PER CARD
                100017 100140000102 10 00 C6 00 00 000102 ILP2      L     A6,66 . LOAD MASK REGISTER
                100020 731140000022 73 02 C6 00 00 000022           SSL   A6,18 . SHIFT MASK TO RIGHTMOST HALFWORD
                100021 230040000022 23 00 02 00 00 000022           L     R2,18 . RESTORE MASK REGISTER
                100022 710102500042 71 00 04 02 10 100042           MSE   A4,INAREA,*X2 . SEARCH A SECOND TIME
                100023 742000100025 74 04 00 00 00 100025           J     $+2 . UNSUCCESSFUL
                100024 720400100026 72 01 00 00 00 100026           SLJ   ,ORTN . OUTPUT
                100025 742000100001 74 04 00 00 00 100001 CONT      J     CLP .  LOOP SIX TIMES
                100026 000000000000 00 00 00 00 00 000000 ORTN      0 . OUTPUT ROUTINE
                100027 100160100112 10 00 C7 00 00 100112           L     A7,(6)
                100030 150160000016 15 00 07 00 00 000016           AN    A7,14 . FIND NUMBER OF CURRENT CARD
                100031 404160100121 40 10 07 00 00 100121           OR,8  A7,(060) . CONVERT VALUE TO DISPLAY CHARACTER
                100032 014200100C72 01 10 10 00 00 100C72           SA,8  A8,NUM . STORE IN PRINT LINE
                100033 260020100122 26 00 01 00 00 100122           LXM   X1,(L A1,OUTA) . SET X1 FOR ER WRITE REQUEST
                100034 724400000017 72 11 00 00 00 000017           ER    ,017 . REQUEST WRITE OF EXECUTIVE
                100035 742000300026 74 04 00 00 01 100026           J     *ORTN . RETURN TO MAINLINE
                100036 110505050505 11 01 04 05 00 050505           0110505050505 . FOR PRINTING INPUT RECORD
                100037 110631060510 11 01 11 11 00 060510           'DATA CARD RE'
                       062711052712
                100041 061105050505 06 02 04 05 00 C50505           'AD'
                100042 000000000000 00 00 00 00 00 000000 INAREA    RES   12 . 12 WORDS FOR CARD INPUT
                100056 050505050505 05 01 04 05 00 050505           ' 'D
                       050505050505
                100060 050505050505 05 01 04 05 00 050505           ' 'D
                       050505050505
                100062 050505050505 05 01 04 05 00 050505           ' 'D
                       050505050505
                100064 110505050505 11 01 04 05 00 050505 OUTA      0110505050505 . 22 WORDS FOR LINE PRINTER OUTPUT
                100065 301206271015 30 02 10 06 01 071015           'SEARCH'
```

```
ERROR FLAGS    ADDR  OBJECT CODE  F  J  A  X  HI   U        SOURCE CODE

               100066 050627143222 05 01 11 07 00 143222          ' ARGUM'
               100067 122331051324 12 C4 15 11 00 051324          'ENT FC'
               100070 322311051623 32 C4 14 11 00 C51623          'UND IN'
               100071 C5100627110S 05 02 00 06 01 071105          ' CARD'
               100072 05C505050505 C5 01 04 C5 00 C50505 NUM      ' 'D
                      050505050505
               100074 050505050505 C5 01 04 05 00 050505          ' 'D
                      C50505050505
               100076 050505050505 C5 01 04 05 00 C505C5          ' 'D
                      050505050505
               100100 C50505C5C5C5 C5 C1 04 C5 00 050505          ' 'D
                      050505050505
               100102 050505050505 05 01 04 05 00 C50505          ' 'D
                      C50505050505
               100104 050505050505 C5 01 04 C5 00 C5C5C5          ' 'D
                      050505C50505
               100106 050505050505 C5 01 04 05 00 050505          ' 'D
                      050505050505
               100110 050505050505 05 01 04 05 00 C50505          ' 'D
                      C50505050505
               777777 000000000270 00 00 00 00 00 000270          END   0270 . INITIAL EXECUTIVE ENTRY
```

```
000270
000271
000272
100000
100001
100003
100004
100005
000301
000302
000303
000304
000305
000306
000307
000311
000312
000315
000316
000317
000320
000321
100006
100007
000301
000302
000303
000304
000305
000306
000307
000311
000312
000313
DATA CARD READ    A TEST DATA CARD WITH SEVERAL POSSIBILITIESARG ARG ARG ARG ARG ARG
000314
000316
000317
000320
000321
100010
100011
100012
100013
100015
100027
100030
100031
100032
```

```
100033
100034
C00301
000302
000303
C00304
000305
000306
000307
000311
000312
000313
SEARCH ARGUMENT FOUND IN CARD      1
000314
000316
000317
000320
000321
100035
100016
100001
100003
100004
100005
000301
000302
000303
000304
000305
000306
000307
000311
000312
000315
000316
000317
000320
000321
100006
100007
000301
000302
000303
000304
000305
000306
000307
000311
000312
```

```
000315
DATA CARD READ    AND ANOTHER ARG THAT IS CERTAIN
000314
000316
000317
000320
000321
100010
100011
100012
100013
100015
100027
100030
100031
100032
100033
100034
000301
000302
000303
000304
000305
000306
000307
000311
000312
000313
SEARCH ARGUMENT FOUND IN CARD      2
000314
000316
000317
000320
000321
100035
100016
100001
100003
100004
100005
000301
000302
000303
000304
000305
000306
000307
000311
000312
```

```
000315
000316
000317
000320
000321
100006
100007
000301
000302
000303
000304
000305
000306
000307
000311
000312
000313
DATA CARD READ    ARG AGAIN
000314
000316
000317
000320
000321
100010
100011
100012
100013
100015
100027
100030
100031
100032
100033
100034
000301
000302
000303
000304
000305
000306
000307
000311
000312
000313
SEARCH ARGUMENT FOUND IN CARD      3
000314
000316
000317
000320
```

```
000321
100035
100C16
100C01
100C03
100C04
100C05
000301
000302
000303
000304
000305
000306
000307
000311
000312
000315
000316
000317
000320
000321
100006
100007
000301
000302
000303
000304
000305
000306
000307
000311
000312
000313
DATA CARD READ    NOT HERE
000314
000316
000317
000320
000321
100010
100011
100012
100013
100014
100017
100020
100021
100022
100023
```

```
100025
100C01
100C03
100C04
100C05
000301
C0C302
00C3C3
000304
C00305
C00306
C00307
000311
000312
000315
000316
000317
000320
C0C321
100C06
100C07
000301
C0C302
C0C3C3
000304
000305
C00306
000307
000311
000312
000313
DATA CARD READ     GR HERE
000314
000316
000317
000320
000321
1CCC10
100C11
100012
100013
100014
100017
100C20
100021
100022
100023
100025
100C01
```

```
100003
100004
100005
000301
000302
000303
000304
000305
000306
000307
000311
000312
000315
000316
000317
000320
000321
100006
100007
000301
000302
000303
000304
000305
000306
000307
000311
000312
000313
DATA CARD READ      ARG ALMOST
000314
000316
000317
000320
000321
100010
100011
100012
100013
100014
100017
100020
100021
100022
100023
100025
100001
100002
000301
```

```
000302
000303
000304
000305
000306
000307
000310
NORMAL END
```

3. Program Test Subroutine.

A small mainline program performs some half and third word arithmetic. It invokes a dump subroutine for program testing.

The subroutine TDUMP prints between 1 and 7 words in both display and octal. It may be called any number of times. The arguments are -

(i) Number of words to dump - an integer between 1 and 7 in user Accumulator A0.

(ii) The address of the first word to dump - in user Index Register X1.

TDUMP does not alter the contents of any Index Registers or Accumulators.

```
//RLM92 JOB '0299,XXX,1,T=20,I=20,C=0,L=2,R=78','R MACMILLAN'          JOB 358
//JOBLIB DD DSN=RLMACL.A0224.RACLIB,UNIT=SYSDA,
//    DISP=SHR,VOLUME=SER=UM1404
//CM EXEC PGM=ASM11C8,REGION=42K
//SYSUT1 DD DSN=RLMACL.A0224.SYSUT1,UNIT=SYSDA,
//    VOLUME=REF=ONE.MONTH,DISP=(NEW,DELETE),SPACE=(TRK,(1,1),RLSE),
//    DCB=(RECFM=F,BLKSIZE=84,LRECL=84)
//SYSUT2 DD DSN=RLMACL.A0224.SYSUT23,UNIT=SYSDA,
//    VOLUME=REF=ONE.MONTH,DISP=(NEW,PASS),SPACE=(TRK,(1,1),RLSE),
//    DCB=(RECFM=FB,BLKSIZE=160,LRECL=16)
//SYSUT3 DD DSN=RLMACL.A0224.SYSUT3,UNIT=SYSDA,
//    VOLUME=REF=ONE.MONTH,SPACE=(TRK,(1,1),RLSE),
//    DISP=(NEW,DELETE),DCB=(RECFM=FB,BLKSIZE=800,LRECL=80)
//SYSPRINT DD SYSOUT=A
//SYSUDUMP DD SYSOUT=A
//SYSIN DD *
IEF373I STEP /CM      / START 71188.1210
IEF374I STEP /CM      / STOP  71188.1211 CPU    0MIN 01.77SEC MAIN  42K LCS    0K
CM           57.80 SEC EXEC TIME     1.77 SEC CPU TIME     509 I/O COUNTS  42K REGION  42K USED

//GO EXEC PGM=LEX1108,PARM='MS=36863,TRACE=NC',REGION=78K
//GO.SYSUT2 DD DSN=RLMACL.A0224.SYSUT23,DISP=(OLD,DELETE),UNIT=2314,
//    VOLUME=REF=ONE.MONTH,DCB=(RECFM=FB,LRECL=16,BLKSIZE=160)
//SYSPRINT DD SYSOUT=A
//SYSUDUMP DD SYSOUT=A
//SYSIN DD *
IEF373I STEP /GO      / START 71188.1211
IEF374I STEP /GO      / STOP  71188.1211 CPU    0MIN 01.09SEC MAIN  78K LCS    0K
GO            8.78 SEC EXEC TIME     1.09 SEC CPU TIME     138 I/O COUNTS  78K REGION  78K USED

IEF375I  JOB /RLM92   / START 71188.1210
IEF376I  JOB /RLM92   / STOP  71188.1211 CPU    0MIN 02.86SEC
RLM92        66.58 SEC EXEC TIME     2.86 SEC CPU TIME        DATE 71.188 UNIVERSITY OF MANITOBA

 HASP-II JOB STATISTICS --     123 CARDS READ --       177 LINES PRINTED --       0 CARDS PUNCHED --    1.10 MINUTES EXECUTION TIME
 UNIVERSITY OF MANITOBA --     669 I/O COUNTS --     1,020 K BYTE-SECONDS -     1.55 UNITS         --    0.04 MINUTES CPU TIME
 CALINE ACCOUNTING INFO --     574 JOB(S) RUN --
```

```
ERROR FLAGS    ADDR  OBJECT CODE  F  J  A  X  H1  U          SOURCE CODE

                                                    $(1),MAIN RES   01000-$          . LOAD AT 1000 OCTAL
001000 100240001022 10 00 12 00 00 001022                     L     A10,(7)          . NUMBER OF ITERATIONS REQD
001001 100120001014 10 00 05 00 00 001014                     L     A5,A
001002 100140001016 10 00 06 00 00 001016                     L     A6,C
001003 700540001005 70 01 06 00 00 001005 ST2                 JGD   A10,$+2          . TEST FOR EOJ, DECR COUNTER
001004 7244C0000077 72 11 00 00 00 000077 END                 ER    ,077             . TERMINATE JOB
001005 722120001015 72 04 05 00 00 001015                     AH    A5,B
001006 723540001017 72 07 06 00 00 001017                     ANT   A6,D
001007 715120001020 71 12 05 00 00 001020                     DS    A5,E             . STORE RESULTS OF ARITHMETIC
001010 100000001023 10 00 00 00 00 001023                     L     A0,(2)           . NUMBER OF WORDS TO DUMP
001011 270020001007 27 00 01 00 00 001007                     LX    X1,$-2           . LOAD ADDRESS OF E FOR TDUMP
001012 72C430000270 72 C1 C0 C0 00 000270                     SLJ   ,TDUMP           . CALL DUMP SUBROUTINE
001013 742000001003 74 04 00 00 00 001003                     J     ST2              . LOOP
001014 C0000300C032 CC C0 CC C3 00 C00032 A                   003000032
001015 000007000061 C0 C0 00 C7 00 C00061 B                   007000061
001016 000100077776 00 00 04 00 00 077776 C                   00100077776
001017 000700010031 C0 01 14 00 00 010031 D                   00700010031
001020 000000000000 00 00 00 00 00 000000 E                   RES   2
```

```
ERROR FLAGS    ADDR  OBJECT CODE  F  J  A  X  HI   U        SOURCE CODE

                                                    /$(31)   RES   0270-$                . DUMP SUBROUTINE
                000270 000000000000 00 00 00 00 00 000000 TDUMP  0                      . ENTRY POINT
                000271 060040000406 06 00 02 00 00 000406        S     X2,SX2           . SAVE INDEX REGISTERS AND
                000272 060060000407 06 00 03 00 00 000407        S     X3,SX3           . ACCUMULATORS
                000273 270040000464 27 00 02 00 00 000464        L     X2,(NOP ,SVA)    . ADDRESS OF SAVE AREA
                000274 460040000465 46 00 02 00 00 000465        LXI   X2,(01)          . INDEX INCREMENT
                000275 270060000355 27 00 03 00 00 000355        LX    X3,CONST1        . REGISTER ADDRESS
                000276 230020000466 23 00 01 00 00 000466        L     R1,(24)          . NUMBER OF WORDS TO MOVE
                000277 220043400000 22 00 02 03 10 000000        BT    X2,0,*X3         . MOVE TO SAVE AREA
                000300 550000000467 55 00 00 00 00 000467        TG    A0,(8)           . MAXIMUM NUMBER OF WORDS = 7
                000301 742000000345 74 04 00 00 00 000345        J     RETX             . RETURN NO PRINT
                000302 270060000470 27 00 03 00 00 000470        L     X3,(01000001)    . PLINO INDEX
                000303 270100000471 27 00 04 00 00 000471        L     X4,(03000001)    . PLIND INDEX
                000304 460020000472 46 00 01 00 00 000472        LXI   X1,(01)          . INDEX INCR FOR INPUT ADDRESS
                000305 700300000307 70 00 14 00 00 000307 LP     JGD   A0,$+2           . TEST AND DECREMENT
                000306 742000000330 74 04 00 00 00 000330        J     RET
                000307 100121400000 10 00 05 01 10 000000        L     A5,0,*X1         . GET INPUT WORD
                000310 010124400410 01 00 05 04 10 000410        S     A5,PLIND,*X4     . INSERT INTO DISPLAY LINE
                000311 100160000473 10 00 07 00 00 000473        L     A7,(2)           . COUNTER
                000312 700460000314 70 01 03 00 00 000314 LP1    JGD   19,$+2           . LOOP THROUGH TWICE
                000313 742000000326 74 04 00 00 00 000326        J     FINX
                000314 100140000474 10 00 06 00 00 000474        L     A6,(6)           . COUNTER
                000315 700440000317 70 01 02 00 00 000317 LP2    JGD   18,$+2           . LOOP THROUGH 6 TIMES
                000316 742000000322 74 04 00 00 00 000322        J     FIN
                000317 735100000003 73 12 04 00 00 000003        LSSL  A4,3             . SHIFT LEFT ACC 4
                000320 735300000003 73 13 04 00 00 000003        LDSL  A4,3             . SHIFT LEFT ACCS 4-5
                000321 742000000315 74 04 00 00 00 000315        J     LP2
                000322 100220000020 10 00 11 00 00 000020 FIN    L     A9,16            . COPY ACC 4
                000323 400220000475 40 00 11 00 00 000475        OR    A9,(0606060606060) . OR IN ZONE BITS FOR EACH OCTAL
                000324 010243400436 01 00 12 03 10 000436        S     A10,PLINO,*X3    . STORE IN OUTPUT LINE
                000325 742000000312 74 04 00 00 00 000312        J     LP1
                000326 240060000476 24 00 03 00 00 000476 FINX   A     X3,(01)          . ADDITIONAL INCR FOR X3
                000327 742000000305 74 04 00 00 00 000305        J     LP
                000330 370000000410 37 00 00 00 00 000410 RET    PUT   PLIND            . OUTPUT DISPLAY LINE
                000331 370000000436 37 00 00 00 00 000436        PUT   PLINO            . OCTAL LINE
                000332 100100000477 10 00 04 00 00 000477        L     A4,(' ')         . CLEAR OUTPUT AREA
                000333 010100000411 01 00 04 00 00 000411        S     A4,PLIND+1
                000334 010100000437 01 00 04 00 00 000437        S     A4,PLINO+1
                000335 270040000500 27 00 02 00 00 000500        L     X2,(01000000)
                000336 270060000501 27 00 03 00 00 000501        L     X3,(01000001)
                000337 230020000502 23 00 01 00 00 000502        L     R1,(20)
                000340 220062400411 22 00 03 02 10 000411        BT    X3,PLIND+1,*X2
                000341 230020000503 23 00 01 00 00 000503        L     R1,(20)
                000342 270040000504 27 00 02 00 00 000504        L     X2,(01000000)
                000343 270060000505 27 00 03 00 00 000505        L     X3,(01000001)
                000344 220062400437 22 00 03 02 10 000437        BT    X3,PLINO+1,*X2
                000345 270040000506 27 00 02 00 00 000506 RETX   L     X2,(NOP ,SVA)    . RESTORE REGS
                000346 460040000507 46 00 02 00 00 000507        LXI   X2,(01)
                000347 270060000355 27 00 03 00 00 000355        L     X3,CONST1
```

```
ERROR FLAGS    ADDR  OBJECT CODE  F  J  A  X  HI   U        SOURCE CODE

                000350 230020000510 23 00 01 00 00 000510          L     R1,(24)
                000351 220062400C00 22 00 03 C2 10 C00000          BT    X3,0,*X2
                000352 27C040000406 27 00 02 00 00 C00406          L     X2,SX2
                000353 270C60000407 27 00 03 00 00 000407          L     X3,SX3
                000354 742000200270 74 C4 00 00 01 000270          J     *TDUMP                    . RETURN TO CALLING PROGRAM.
                000355 000001000004 00 00 00 01 00 C00004 CONST1   D1000004
                000356 CC0000000000 00 00 00 00 00 000000 SVA      RES   24
                000406 CC0000000000 CC 00 00 00 00 000000 SX2      RES   1
                000407 000000000000 00 00 00 00 00 000000 SX3      RES   1
                000410 050505050505 05 01 04 C5 00 C50505 PLIND    ' 'D
                       050505050505
                000412 050505050505 05 01 04 C5 00 C50505          ' 'D
                       C50505050505
                000414 050505050505 C5 C1 04 C5 0C C5C5C5          ' 'D
                       050505050505
                000416 C50505050505 C5 01 04 C5 00 050505          ' 'D
                       050505050505
                000420 050505050505 05 01 04 C5 00 C5C5C5          ' 'D
                       C50505050505
                000422 050505050505 C5 C1 04 C5 0C C5C5C5          ' 'D
                       050505050505
                000424 C50505050505 C5 01 04 C5 00 050505          ' 'D
                       050505050505
                000426 050505050505 05 01 04 C5 00 C5C5C5          ' 'D
                       050505050505
                000430 050505050505 C5 01 04 C5 0C C5C5C5          ' 'D
                       050505050505
                000432 C50505050505 05 01 04 05 00 050505          ' 'D
                       050505050505
                000434 050505050505 05 01 04 C5 00 C50505          ' 'D
                       C505C5050505
                000436 110505050505 11 C1 04 C5 0C C5C5C5 PLIND    D11050505050505050505050505
                       050505050505
                000440 050505050505 C5 01 04 05 00 C5C5C5          ' 'D
                       050505050505
                000442 050505050505 05 01 04 C5 00 C50505          ' 'D
                       050505050505
                000444 050505050505 C5 C1 04 C5 00 C5C5C5          ' 'D
                       050505050505
                000446 C50505050505 C5 01 04 05 00 050505          ' 'D
                       050505050505
                000450 050505050505 05 01 04 C5 00 050505          ' 'D
                       C50505050505
                000452 050505050505 C5 C1 04 C5 00 C5C5C5          ' 'D
                       050505050505
                000454 C50505050505 05 01 04 05 00 050505          ' 'D
                       050505050505
                000456 050505050505 05 01 04 05 00 C50505          ' 'D
                       050505050505
```

```
ERROR FLAGS    ADDR  OBJECT CODE  F  J  A  X  HI   U        SOURCE CODE

               000460 050505050505 C5 01 04 C5 00 050505      ' 'D
                      050505050505
               000462 050505050505 05 01 04 05 00 050505      ' 'D
                      050505050505
               777777 000000001000 00 00 00 00 00 001000      END   01000
```

```
@@E@(F              ?9@^?>
CCCC12C00113        777100067745
@@L@(/              ?2@ ?G
CCCC21C00174        776200057714
@@S@)               ?:@¬<3
000030000255        775300047663
@@Z@#Y              ?=@#<U
CCCC37C00336        774400037632
@@&@¬J              ?X@)<(
CCCC46C00417        773500027601
@@ @ @              ?Q@(.*
CCCC55C00500        772600017550
@@4@ 1              ?J@@.J
CCCC64C00561        771700007517
NORMAL END
```

Two lines of output printed each time TDUMP is invoked. The first is the display dump, the second the same words in octal.

4. Errors and Core Dump.

This sample program has Assembler errors. The execution phase is attempted and an 1108 dump results.

```
//RLM92 JOB '0299,XXX,1,T=20,I=20,C=0,L=2,R=78','R MACMILLAN'          JOB 237
//JOBLIB DD DSN=RLMACL.A0224.RACLIB,UNIT=SYSDA,
//    DISP=SHR,VOLUME=SER=UM1404
//CM EXEC PGM=AS41108,REGION=42K
//SYSUT1 DD DSN=RLMACL.A0224.SYSUT1,UNIT=SYSDA,
//    VOLUME=REF=ONE.MONTH,DISP=(NEW,DELETE),SPACE=(TRK,(1,1),RLSE),
//    DCB=(RECFM=F,BLKSIZE=84,LRECL=84)
//SYSUT2 DD DSN=RLMACL.A0224.SYSUT24,UNIT=SYSDA,
//    VOLUME=REF=ONE.MONTH,DISP=(NEW,PASS),SPACE=(TRK,(1,1),RLSE),
//    DCB=(RECFM=FB,BLKSIZE=160,LRECL=16)
//SYSUT3 DD DSN=RLMACL.A0224.SYSUT3,UNIT=SYSDA,
//    VOLUME=REF=ONE.MONTH,SPACE=(TRK,(1,1),RLSE),
//    DISP=(NEW,DELETE),DCB=(RECFM=FB,BLKSIZE=800,LRECL=80)
//SYSPRINT DD SYSOUT=A
//SYSUDUMP DD SYSOUT=A
//SYSIN DD *
IEF373I STEP /CM       / START 71188.0913
IEF374I STEP /CM       / STOP  71188.0913 CPU    0MIN 00.57SEC MAIN  42K LCS    0K
CM           12.53 SEC EXEC TIME      0.57 SEC CPU TIME      267 I/O COUNTS  42K REGION  42K USED

//GO EXEC PGM=LEXI108,PARM='MS=4095,TRACE=YES',REGION=58K
//GO.SYSUT2 DD DSN=RLMACL.A0224.SYSUT24,DISP=(OLD,DELETE),UNIT=2314,
//    VOLUME=REF=ONE.MONTH,DCB=(RECFM=FB,LRECL=16,BLKSIZE=160)
//SYSPRINT DD SYSOUT=A
//SYSUDUMP DD SYSOUT=A
//SYSIN DD *
IEF373I STEP /GO       / START 71188.0913
IEF374I STEP /GO       / STOP  71188.0913 CPU    0MIN 01.22SEC MAIN  58K LCS    0K
GO           8.46 SEC EXEC TIME      1.22 SEC CPU TIME      132 I/O COUNTS  58K REGION  58K USED

IEF375I  JOB /RLM92    / START 71188.0913
IEF376I  JOB /RLM92    / STOP  71188.0913 CPU    0MIN 01.79SEC
RLM92        20.99 SEC EXEC TIME      1.79 SEC CPU TIME        DATE 71.188 UNIVERSITY OF MANITOBA

 HASP-II JOB STATISTICS --      37 CARDS READ --       597 LINES PRINTED --        0 CARDS PUNCHED --    0.34 MINUTES EXECUTION TIME
 UNIVERSITY OF MANITOBA --     421 I/O COUNTS --       792 K BYTE-SECONDS -      1.32 UNITS          --    0.02 MINUTES CPU TIME
 ONLINE ACCOUNTING INFO --     560 JOB(S) RUN --
```

```
ERROR FLAGS    ADDR  OBJECT CODE  F  J  A  X  HI   U        SOURCE CODE

                                                    $(1)      RES   0500-$  . RELATIVE LOCATION IN MAIN STORAGE
                                                              STC   A2,FLD . INVALID INSTRUCTION      ; CONTINUATION
                     000500 000040000000 00 00 02 00 00 000000  THIS INSTRUCTION WILL GIVE DUMP AT EXECUTION TIME
       S             000501 100030000507 10 00 C0 00 00 000507           LA    X2,STP . X SPECIFICATION WHERE A EXPECTED
                                                                                    #
       UE            000502 300040C00000 30 00 02 00 00 C00000           MI    A2,FLD . UNDEFINED FLD
                                                                                   ##
       S             000503 340040000506 34 00 02 00 00 000506           DI    A2,LIM,23 . ILLEGAL INDEX REG
                                                                                    #
       UE            000504 010060000000 01 00 03 00 00 000000           SA    A3,FIN+23N . ILLEGAL EXPRESSION
                                                                                   ##
       UE            000505 270040000000 27 00 02 00 00 000000 FIN       L     X2,A2 . ACC MNEMONIC INVALID IN U-FIELD
                                                                                  ##
                     000506 000000000000 00 00 00 00 00 000000 LIM       RES   1
                     000507 000000000077 00 00 00 00 00 000077 STP 077
                     000510 000000000024 00 00 00 00 00 000024 T22       024
       D             000511 000000000000 00 00 00 00 00 000000 T22       RES 1    . MULTIPLY DEFINED LABEL

                     777777 000000000500 00 00 00 00 00 000500           END   0500
```

Note: S - Syntax error
U - Undefined variable name
E - Error in expression
D - Duplicate label

INTERNAL REGISTERS

P-REGISTER	000501	PSR	000000177000	SLR	110000110001	EFA	000000
INT ADDRESS	000000	CHAN	00-0-00-00-0	MSR	0	HK	0000
CSR	00	LAR	0	IS	0	LPS	0

ADDRESS	STORAGE CONTENTS							
REGISTER CONTENTS								
000000	216402235055	776100001000	064120321101	150437604470	044315074011	534030602103	450530301001	116500020000
000010	240710000105	101400442027	000040300161	054404210000	024051300414	310503010000	644014200114	041510030400
000020	600123220061	031400446560	040100200411	241200052167	450023260074	101400402167	450120335664	741120306520
000030	600202243344	401120246560	600203001004	540000040604	260500000411	100502662170	450321302405	350541202400
000040	656015202205	220404204507	024301010411	424050602615	600061205720	040501150001	461663200311	777500140177
000050	644000200002	000000000000	676700000153	400300067230	705617276074	613617246534	663546536320	001774022020
000060	000033510200	015643706617	217003770044	000000067220	000033510416	721613116134	705423307500	000000654000
000070	623427067437	741445006604	651437312014	303531006135	662403066555	200401002004	200401002004	663427437115
000100	652403036554	612401002002	000400003006	200000000017	452221010400	440407604400	024034000156	320000000000
000110	623427067175	741703602137	740120000156	370000000000	623427067137	753637230240	417461057417	050000067231
000120	000000000014	613433456517	200400120165	071103546400	013000301425	500300302612	000001304412	230542600001
000130	261302600304	000000702121	400170120241	342541201402	066631033120	014031402605	000022102400	040455205407
000140	217302002304	200000012266	541020332524	741001262204	540770342104	201302060645	064531013000	002141062244
000150	541000306141	610145342170	401001071010	364157040665	217702004101	330437604006	060660332521	113044600000
000160	430200000204	314000010445	216702010244	241100042167	401241003011	010437604031	262502200004	401002702611
000170	440001077410	461106004400	217002005004	241302002024	500541011011	020500402001	202420001004	104002602145
HIDDEN STORAGE								
000000	000000000000	000000000000	000000000000	000000000000	000000000000	000000000000	000000000000	000000000000
000010	000000000000	000000000000	000000000000	000000000000	000000000000	000000000000	000000000000	000000000000
000020	000000000000	000000000000	000000000000	000000000000	000000000000	000000000000	000000000000	000000000000
000030	000000000000	000000000000	000000000000	000000000000	000000000000	000000000000	000000000000	000000000000
000040	000000000000	000000000000	000000000000	000000000000	000000000000	000000000000	000000000000	000000000000
000050	000000000000	000000000000	000000000000	000000000000	000000000000	000000000000	000000000000	000000000000
000060	000000000000	000000000000	000000000000	000000000000	000000000000	000000000000	000000000000	000000000000
000070	000000000000	000000000000	000000000000	000000000000	000000000000	000000000000	000000000000	000000000000
000100	000000000000	000000000000	000000000000	000000000000	000000000000	000000000000	000000000000	000000000000
000110	000000000000	000000000000	000000000000	000000000000	000000000000	000000000000	000000000000	000000000000
000120	000000000000	000000000000	000000000000	000000000000	000000000000	000000000000	000000000000	000000000000
000130	000000000000	000000000000	000000000000	000000000000	000000000000	000000000000	000000000000	000000000000
000140	000000000000	000000000000	000000000000	000000000000	000000000000	000000000000	000000000000	000000000000
000150	000000000000	000000000000	000000000000	000000000000	000000000000	000000000000	000000000000	000000000000
000160	000000000000	000000000000	000000000000	000000000000	000000000000	000000000000	000000000000	000000000000
000170	000000000000	000000000000	000000000000	000000000000	000000000000	000000000000	000000000000	000000000000
MAIN STORAGE								
000200	720400200250	720400200250	720400200250	720400200250	720400200250	720400200250	720400200250	720400200250
000210	720400200250	720400200250	720400200250	720400200250	720400200250	720400200250	720400200250	720400200250
000220	720400200250	720400200250	720400200250	720400200250	720400200250	720400200250	720400200250	720400200250
000230	720400200250	720400200250	720400200250	720400200250	720400200250	720400200250	720400200250	720400200250
000240	720400200250	720400200250	720400200250	720400200250	720400200250	720400200250	720400200250	720400200250
000250	000000000000	000000000000	720400200250	341006506440	200002200015	007100005004	241102402305	300300402604

000260	000000331050	100000104203	000040221044	401010060142	203444000204	040002600700	055406250105	020436004040	
000270	645277000004	001132004400	217702020155	536400006000	067775207720	040140272020	000340120251	731500140177	
000300	000033536200	220050120130	220602604005	716000604430	440031074010	061110053401	217702000711	014700202025	
000310	500540306144	640000400714	065525206560	050141154520	340101073410	200443204400	217702002601	105064400000	
000320	202100000021	045060400000	217002004544	656000422202	540751136413	414435004007	067461100413	374167010655	
000330	423500000204	671500012177	401051126413	414453205410	014202111000	020153222020	000002110000	020437604011	
000340	260100601005	002000042021	540321200160	150140472613	000101307413	070027572600	000321300403	071510070402	
000350	340003220104	000700036440	240101600404	210000052025	000201063010	670437604035	430000000410	040000302021	
000360	000005305400	040542605400	220202603641	004436604020	220202604104	101302062262	404631077410	034440405407	
000370	062111071010	140417670000	203774000024	756000074221	000142150000	041121772400	217702006004	040000010640	
000400	422100000610	300000202021	540321200160	150404470000	261300000405	741300340136	260034001515	002700033400	
000410	216702013415	004700053400	455001200004	741003006440	340001600204	000000302121	402770120244	102000042265	
000420	404653220161	040500302402	240142220265	060407000000	444401200704	600500362034	240141122410	324503010000	
000430	6560542 00001	001510010400	340015306160	050457004046	060261306300	000113542167	403411200440	000540070003	
000440	217702006005	040300342602	000121201160	040542470002	202202402605	102000242602	000101301002	030404401000	
000450	240204000604	042000040240	260100600704	000000010240	260234000405	102000242614	000121136010	320143342614	
000460	000000221304	401011102614	000001300440	031510070400	100100300444	000000300240	217702021215	007500006400	
000470	067775300403	071125770400	217002023004	740000142616	000300077740	002000100002	000004001430	001447056157	
000500	000040000000	100000000507	300040000000	340040000506	010060000000	270040000000	000000000000	000000000077	
000510	000000000024	000000000000	400003116134	745717676224	721643502004	201407236475	607407436134	201617262014	
000520	611527016514	717634040004	001360002170	001447056157	751743112015	655413256035	613635007056	613537256035	
000530	612403046115	603523057176	201427316635	662403016475	655417017074	653435006634	705427317134	610403456555	
000540	711523050400	240000400002	400003116134	745723706224	721643502004	201443017044	663427426135	713427042016	
000550	655517446514	201613427056	705610200005	000000000000	000000000000	117000006234	615713637614	201547056575	
000560	720403046136	623417052015	603523052015	662403036035	607427232000	240000740002	400003116134	745727616224	
000570	643533026534	651427472015	655411006074	653417056474	610020002000	000003116134	753703666224	611423256035	
000600	613637472015	654403427075	663407076124	713533237115	613611132016	655517446514	375537316236	603617052014	
000610	705613446514	610000100137	016001057417	054000067374	000000000014	613433037135	647704120160	740427607401	
000620	000033576600	000000006234	615417456555	744050070137	213703600500	015677600000	000003116134	607627266477	
000630	024035200402	110404200000	304100000146	000000002400	000000000000	000000000000	000000000000	000000000000	
000640	000000000000	000000000000	000000000000	000000000000	000000000000	000000000000	000000000000	000000000000	
000650	000000000000	000000000000	000000000000	000000000000	000000000000	000000000000	000000000000	000000000000	
000660	000000000000	000000000000	000000000000	000000000000	000000000000	000000000000	000000000000	000000000000	
000670	000000000000	000000000000	000000000000	000000000000	000000000000	000000000000	000000000000	000000000000	
000700	000000000000	000000000000	000000000000	000000000000	000000000000	000000000000	000000000000	000000000000	
000710	000000000000	000000000000	000000000000	000000000000	000000000000	000000000000	000000000000	000000000000	
000720	000000000000	000000000000	000000000000	000000000000	000000000000	000000000000	000000000000	000000000000	
000730	000000000000	000000000000	000000000000	000000000000	000000000000	000000000000	000000000000	000000000000	
000740	000000000000	000000000000	000000000000	000000000000	000000000000	000000000000	000000000000	000000000000	
000750	000000000000	000000000000	000000000000	000000000000	000000000000	000000000000	000000000000	000000000000	
000760	000000000000	000000000000	000000000000	000000000000	000000000000	000000000000	000000000000	000000000000	
000770	000000000000	000000000000	000000000000	000000000000	000000000000	000000000000	000000000000	000000000000	
001000	000000000000	000000000000	000000000000	000000000000	000000000000	000000000000	000000000000	000000000000	
001010	000000000000	000000000000	000000000000	000000000000	000000000000	000000000000	000000000000	000000000000	
001020	000000000000	000000000000	000000000000	000000000000	000000000000	000000000000	000000000000	000000000000	
001030	000000000000	000000000000	000000000000	000000000000	000000000000	000000000000	000000000000	000000000000	
001040	000000000000	000000000000	000000000000	000000000000	000000000000	000000000000	000000000000	000000000000	
001050	000000000000	000000000000	000000000000	000000000000	000000000000	000000000000	000000000000	000000000000	
001060	000000000000	000000000000	000000000000	000000000000	000000000000	000000000000	000000000000	000000000000	

Note: Compiled program has been loaded at location 500_8.
Core dump format - Figure 2.5.1.

CHAPTER V CONCLUSIONS.

This chapter discusses project development, problems encountered, and plans for future expansion of the 1108 simulator.

Most project time was spent in design and development of the Assembler and Interpreter and less on the Loader. The latter was considered of lesser importance in illustrating 1108 capablities. An improved Loader which may be called from the executing 1108 program is planned. This will enable users to better simulate the operation of an Executive system in loading and relocating "user" programs.

During the writing of the 1108 Assembler, the absence of a certain feature in both the /360 and 1108 became apparent. The Assembler recurses to translate statements which nest literals and line items. Pushdown stack handling routines were written to perform this function. Neither the /360 nor the 1108 has a hardware pushdown stack. The stack feature facilitates programming this type of re-entrant routine.

Due to the magnitude of the project, some sections of the 1108 Assembler were omitted. The system is structured to allow these to be included at a future date. Procedures, further directives and partial word constants are in this category.

Some problems were encountered because of the difference

in the arithmetic sections of the two machines. The 1108 word length differs from that of the /360 and all arithmetic is ones complement binary. The arithmetic section of the Interpreter therefore does repeated conversions and alignments. The 1108 word is stored in the leftmost 36 bits of a five byte block. Negative values are converted to their twos complement equivalent before each arithmetic operation and a negative result is converted back to ones complement. Addition, subtraction, and multiplication are done using multiple register /360 arithmetic. A step by step division is necessary to provide the divide fractional instructions. Obviously the simulated arithmetic involves multiple steps for each operation. As the entire simulator is coded in /360 Assembler it is as efficient as possible in core usage and execution time. The following indicates approximate number of /360 Assembler instructions required to simulate 1108 arithmetic operations:

Operation	/360 instructions executed
Divide	50 - 150
Multiply	15 - 60
Add	25 - 40
Shift Circular	8 - 20
Repeated Search	20 - 30 times number of cycles
Jump	3 - 7

Note that these figures do not include instructions to calculate effective addresses, j-control loads and stores etc.

Future additions include -

- 1108 instruction timing.
- 1108 I/O section (see Appendix IV). Simulated I/O devices could then be "interfaced" with the 1108.

While the simulator does not illustrate all 1108 characteristics, most have been included, for example -

- indirect addressing to any level
- indexing at each level of indirect addressing
- index register modification
- storage protection
- guard mode
- partial word arithmetic
- partial word data transfer
- program relocation
- repeated instructions such as repeated search and repeated masked search
- multiply and divide fractional instructions
- interrupt handling

The aim of the project was to illustrate that the development of an 1108 simulator for use in student teaching was possible. It has been noted that problem areas exist where the simulator does not exactly represent the actual machine operation. These variations are so few that they do not detract fron the simulators use as a student teaching aid.

In other cases a true representation was obtained at the cost of the efficiency of the simulator. For example, one

hundred and fifty /360 instructions may be required to perform a single 1108 division.

The sample problems contained in Chapter IV illustrate typical student programs. Many 1108 features are used in these programs and the total /360 execution time for compile, load and execute is less than 3 seconds for each example. Therefore, it is concluded that the simulator's inefficiencies are not critical when being used for student programs.

The advantages to the student using the simulator are:-

(1) First hand experience on a machine with many design differences from the University computer.

(2) Additional aids built into the simulator to assist in familiarizing himself with the 1108 and writing and debugging his programs.

The development of simulators of this nature is a relatively inexpensive task. Approximately 1 to 1½ man years would be sufficient to produce a comprehensive system. The simulation of computers will soon assist in the teaching of Computer Science at the University of Manitoba.

Appendix I GLOSSARY

a-field. Four bit field in instruction word (bits 25-22). References a control register, usually an accumulator.

A-registers or Accumulators. Control register locations 12 to 27 (User A-registers) or 108 to 123 (Executive A-registers). Normally used for arithmetic.

Assembler. The set of programs that translate Univac 1108 Assembler Language into relocatable machine code.

Base Register. The Processor State Register contains two base registers (BI and BD). BI is the base address for the program instruction bank, BD the base address for the data bank. In effective address calculation base address plus relocatable address gives absolute address.

Carry Indicator. Bit 27 of Processor State Register indicates carry in most recent arithmetic operation. In ones complement arithmetic a carry from bit position 35 is transferred to bit 0.

Channel Select Register. A four bit internal register used to select channel for I/O operation [4].

Control Registers. 128 program addressable words of high speed storage that provide multiple index registers, accumulators and special registers.

Effective Address. The absolute address obtained from instruction after considering u-field, index register, base register and indirect addressing.

Executive. In the 1108 the Executive program assigns absolute addresses to User programs and handles all I/O transfers. Built in to the simulator is a simplified Executive. The User is encouraged to write his own Executive systems.

Externally Specified Index. One of the two possible modes of operation of each I/O channel. Used for communications equipment. Transfers are either quarter or half word. Not implemented in simulator.

f-field. Six bit field in instruction word (bits 35-30). Contains the function code.

Fixed Address Assignments. Interrupt handling subroutines have entry points and status words assigned fixed locations in main storage. See Table 2.1.1.

Hash value. Pass I of the Assembler reads the 1108 source code and creates a table of user labels. Each mnemonic is positioned in the table according to an integer value which is called the hash value. This integer is obtained by hashing the characters of the mnemonic. Each component character is assigned a value and this value substituted in an equation (see HASH, Section 3.2 The Assembler) that gives an integer result. The equation is designed to give an even spread across the 704 elements of the label table. If a required position is already occupied the Assembler searches sequentially for a vacancy in the table. The search continues only for a specified number of positions. If no vacancy is found the table is considered full and a new one created.

When a label is referenced a similar hash and search process is executed. Here, however, the search continues until the label is either found or established as undefined.

This technique allows the label table to be searched rapidly. A maximum of five seeks is required to locate or position any label. This 5 is called tolerance and is an arbitrary value which has worked very well to date.

h-field. Bit 17 of instruction word. Indicates whether

or not the index register is incremented. Sometimes the h-field, i-field and u-field form an 18-bit absolute address.

Hidden Storage. Main storage locations 0 to 127. Called "hidden" because an address between 0 and 127 normally refers to a control register.

i-field. Bit 16 of instruction word. Specifies indirect addressing. Sometimes the h-field, i-field and u-field form an 18-bit absolute address.

Index Register or I-Register. Control register locations 1 to 15 (User I-Registers) or 97 to 111 (Executive I-Registers). The low order halfword contains an 18-bit modifier used in effective address calculation. The high halfword may be used to increment the index modifier.

Indirect Addressing. When indirect addressing is specified the rightmost 22 bits of the C(E) are loaded into the instruction register and used to compute a new E. This procedure continues until i=0.

Internally Specified Index. One of the two possible modes of operation of each I/O channel. Used for peripheral equipment such as magnetic tapes, drums,

printers and card readers. See Appendix IV.

Interpreter. The set of programs that simulates the execution of an 1108 machine language program.

j-control. Data may be transferred from Main Storage to the Arithmetic Control Unit (or vice-versa) according to a certain pattern. Fullword, halfword, third word, quarter word and single character transfers are possible. See Table 2.1.2. All instructions with $f < 70_8$ transfer data under j-control.

j-field. Four bit field in instruction word (bits 29-26). Is used as an extension to the function code or to specify a data transfer pattern.

Last Address Register. Used in the event of a Main Storage parity error [3]. Not implemented in simulator.

Loader. The program which loads relocatable 1108 machine code into Main Storage, prior to execution. Absolute addresses are assigned and the initial value of the P-register loaded.

Mask Register. Control register location 66 (User Mask Register) or 82 (Executive Mask Register). Used to

"AND-off" bits within words referenced by repeated masked search instructions and some logical comparisons. The Mask Register must be loaded prior to execution of the instruction that references it.

Memory Select Register. When an interrupt occurs control is transferred to a fixed main storage location (see Table 2.1.1). The address of this location is computed using the contents of the 3-bit MSR. The C(MSR) select the fixed address in one of the main storage modules. The simulator has two modules, hence C(MSR) must be zero or one.

Ones complement binary. This is the 1108 internal representation of all binary integers. The negative of an integer is obtained by complementing each binary digit. i.e. changing all zeros to ones and all ones to zeros. This means there are two representations of zero, viz. all binary zeros and all binary ones. Throughout this presentation the former is called +0, the latter -0.

Overflow Indicator. Bit 28 of Processor State Register indicates overflow in most recent arithmetic operation. Overflow is defined as the resultant sign of the arithmetic being inconsistent with the argument signs,

e.g. + x + (+y) gives - z.

P-register or Program Address Counter. An 18-bit internal register that contains the absolute address of the next sequential instruction. The instruction fetch is made from the word addressed by the P-register.

Privileged Mode. A mode of operation established in the PSR which protects only against out-of-bounds writes. The entire instruction set is available to a program in Privileged Mode. Each program run in the simulator commences in this mode.

Processor State Register. A 36-bit internal register which contains the two base registers, and indicators which define the state of the CPU at any time.

R-Registers. Control register locations 64 to 79 (User R-Registers) or 80 to 95 (Executive R-Registers). Includes the Repeat Count and Mask Registers in addition to 13 others for User or Executive use.

Repeat Count Register. Control register location 65 (User Repeat Count) or 81 (Executive Repeat Count). Used to specify number of cycles in repeated instructions such as search, masked search and block transfer instructions. The repeat Count Register must be loaded prior to execution of the instruction

that references it.

Storage Limits Register. A 36-bit internal register that contains upper and lower bounds for both instruction and data banks. The loading of the SLR does not enable storage protection, it merely sets the bounds.

u-field. 15-bit field in instruction word (bits 15-0). Specifies the operand address.

User Program Mode. A mode of operation established in the PSR which provides full protection against out-of-bounds reads or writes and use of certain priveleged instructions.

x-field. Four bit field in instruction word (bits 21-18). References an index register which is used in address calculation.

Appendix II NOTATION

a Bits 25-22 of instruction.

Aa Accumulator specified by a-field of instruction.

Aa+1 Accumulator whose address is one greater than that specified by a-field of instruction.

BD Data bank base register. Bits 8-0 of PSR.

BI Instruction bank base register. Bits 26-18 of PSR.

bm Base address modifier (either BI or BD).

BS Base Selection register. Bits 15-9 of PSR.

C( ) Contents of.

C( )j Contents of, under j-control.

$C( )_{17-0}$ Contents of, specified bits only.

CPU Central Processor Unit.

CR Control register.

CSR Channel Select Register.

D0 Carry indicator (bit 27 PSR).

D1 Overflow indicator (bit 28 PSR).

D2 Guard Mode/Storage Limits Protection indicator (bit 29 PSR).

D3 Modified Storage Protection indicator (bit 30 PSR).

D4 Bit 31 PSR.

D5 Bit 32 PSR.

D6 Control Register Selection indicator (bit 33 PSR).

D7 Base Register Suppression indicator (bit 34 PSR).

D8 Bit 35 PSR.

E Effective address.

E+1 Effective address plus one.

ESI Externally Specified Index.

f Bits 35-30 of instruction.

h Bit 17 of instruction.

HS Hidden Storage.

i Bit 16 of instruction.

IRB Internal Register Block.

ISI Internally Specified Index.

j Bits 29-26 of instruction.

ja Bits 29-22 of JGD instruction.

K Initial value in the Repeat Count Register for Block Transfer, search or masked search instructions.

LAR Last Address Register.

MSR Memory Select Register.

NI Next instruction.

OR Logical sum.

P-Register Program address counter

PSR Processor State Register.

QW Quarter Word designator (bit 17 PSR).

Ra R-Register specified by a-field of instruction.

R1 Repeat Count Register.

R2 Mask Register.

SLR Storage Limits Register.

u Bits 15-0 of instruction.

WSA Work Storage Area.

x Bits 21-18 of instruction.

Xa Index register specified by a-field of instruction.

Xi Bits 35-18 of index register. Index increment - may be positive or negative.

Xm Bits 17-0 of index register. Used in effective address calculation to modify address.

Xx Index register specified by x-field of instruction.

XOR Logical difference.

= In mathematical context means "equals". Otherwise indicates a data transfer e.g. C(F)j = C(Aa), contents of effective address transferred under j-control to contents of accumulator specified by a.

≠ Not equal.

+0 All zeros.

-0 All ones.

0 Either +0 or -0.

| | Magnetude of.

. Arithmetic product.

÷ Arithmetic division.

Appendix III 1108 INSTRUCTION SET.

Mnemonic	Function f	j	Page	Mnemonic	Function f	j	Page
A	14		30	BT	22		30
A	24		32	CDU	76	07	-
AA	14		30	DA	71	10	34
AACI	75	14	-	DAN	71	11	34
AAIJ	74	07	49	DDC	73	14	-
AH	72	04	34	DF	36		34
ALR1	73	14	-	DFA	76	10	-
AM	16		31	DFAN	76	11	-
AMA	16		31	DFD	76	13	-
AN	15		31	DFM	76	12	-
AN	25		32	DJZ	76	15	-
ANA	15		31	DFU	76	14	-
AND	42		52	DI	34		33
ANH	72	05	35	DIC	75	03	-
ANM	17		31	DJZ	71	16	49
ANMA	17		31	DL	71	13	28
ANT	72	07	35	DLM	71	15	-
ANU	21		31	DLN	71	14	28
ANX	25		32	DLSC	73	07	47
AT	72	06	35	DOC	75	07	-
AU	20		31	DS	71	12	30
AX	24		32	DSA	73	05	46

Mnemonic	Function		Page	Mnemonic	Function		Page
	f	j			f	j	
LNA	11		27	MSUM	71	05	40
LNMA	13		28	MSU	71	04	40
LOC	75	04	-	MOR	74	06	54
LOCM	75	05	-	OR	40		52
LPS	72	15	54	PACI	75	15	-
LR	23		28	PAIJ	72	13	54
LSC	73	06	46	S	01		29
LSL	72	16	55	S	04		29
LSSC	73	10	47	S	06		29
LSSL	73	12	47	SA	01		29
LUF	76	04	-	SCM	72	14	-
LX	27		28	SE	62		36
LXI	46		28	SG	65		37
LXM	26		28	SIL	73	15	55
MASG	71	07	40	SLE	64		-
MASL	71	06	40	SLJ	72	01	48
MCDU	76	06	-	SM	03		29
MF	32		32	SMA	03		29
MI	30		32	SN	02		29
MLU	43		52	SNA	02		29
MSE	71	00	39	SNE	63		37
MSG	71	03	39	SNG	64		37
MSI	31		32	SNM	67		38
MSLE	71	02	39	SR	04		29
MSNE	71	01	39	SSA	73	04	46
MSNG	71	02	39	SSC	73	00	45

Appendix IV. I/O SECTION DESIGN [4]

The following is an outline of the I/O Section that is to be implemented at a future date.

When an I/O instruction is executed the system

(a) performs the operation as specified by the instruction.

(b) sets the /360 interval timer to interrupt the interpreter after a small time period.

(c) loads details of I/O operation onto I/O list.

The use of the timer enables the "1108 CPU" to continue processing while the "I/O section" is performing the necessary data transfer. When the time interval expires the interpreter sets values so that an 1108 interrupt, as specified on the I/O list, will occur before the execution of the next sequential instruction. Figure 3.3.2 shows the testing of the IU field to detect an I/O interrupt.

In general I/O instructions are treated according to Figure IV.1. When the current time interval expires the program represented in Figure IV.2 is executed. IP is a count of interrupts pending, i.e. await expiration of timer. IU is a count of outstanding interrupts.

1108 INPUT/OUTPUT

ISI Function Mode.

LFC of LFCM activate three circuits -

(a) force external function (deactivated during cycle)

(b) external function (deactivated by LOC,LOCM,DOC)

(c) output active control (deactivated by terminal condition or DOC)

The output channel is said to be active in function mode when the external function and output active control circuits are simultaneously active.

LFCM also activates -

output monitor control circuit (deactivated by function mode interrupt or by LFC,LOC or DOC).

ISI Output Mode.

LOC or LOCM activate -

output active control (deactivated by terminal condition or DOC).

The output channel is said to be active in output mode when the output active control circuit is active and the external function control circuit is simultaneously inactive.

LOCM also activates -

output monitor control circuit (deactivated when output monitor interrupt occurs or by LFC,LOC or DOC).

ISI Input Mode.

LIC or LICM activate -

input active control circuit (deactivated by terminal condition or by DIC).

An input channel is in input mode when the input active control circuit is active.

LICM also activates -

input monitor control circuit (deactivated when input monitor interrupt occurs or when a DIC is executed).

ISI External Interrupt Mode.

A subsystem reports either on abnormal condition or a normal completion of an operation by assembling a status word and then turning on the correspondin status word signals and external interrupt signal on the input word/status word lines and the external interrupt line respectively.

DIC and DOC deactivate channels.
JIC,JOC and JFC test channel status.

ESI Mode is not considered.

Data Transfer is controlled by Access Control Words (IACR for input, OACR output). For ISI channels these control words are control registers 32-47(IACR) and 48-63(OACR) - one for each channel.

The simulator keeps flags denoting activity of the following circuits -

force external function
external function
output active control
input active control
output monitor control
input monitor control

When an I/O instruction is executed 1 or more items are placed on the I/O list. The element of this list contains

(a) channel number

(b) interrupt address - identifies interrupt type.

(c) identifier (input, output, function, external interrupt)

(d) status word

(e) flag to indicate if time interval has elapsed.

Elements remain on the list until the interrupt is handled.

Interrupts

An external interrupt indicates end of operation. Function external interrupts, and output external interrupts reset output active control. Input external interrupts reset input active control.

The external function control circuit activates the EF control signal as the first word is received by the subsystem.

A monitor interrupt indicates end of operation and resets the appropriate circuit.

APPENDIX V GENERAL DESCRIPTION - 1108

The Univac 1108 Unit - and Multi-Processor system is a general purpose computer, designed to attract both scientific and business oriented users. Since its introduction in 1968 it has developed an impressive sales record. The 1108 has excelled in a real-time environment. Several transportation companies in North America and Europe have installed 1108 computers to handle passenger reservations, the most recent of these being Air Canada, whose centre for reservations is Toronto. On-line terminals enable passenger agents in all parts of Canada, U.S.A. and Europe to communicate directly with the system.

The modularity of the 1108 applies not only to main storage and I/O subsystems but also to central Processors. Single processor systems can be expanded to multiple processor systems just as main storage modules can be added, where user requirements exceed the current configuration.

Some of the features of the 1108 are listed below:

- All Central Processors are equal. Each can perform all functions for execution of instructions, including arithmetic, I/O and Executive control.
- Multiple I/O Controllers. Each is an independant processor which expands the I/O capabilities of the system.
- Modular parity checked high speed main storage. Up to four banks each of 65536 36-bit words. Cycle time

750 nanoseconds.

- Overlapping and interleaving of main storage access. Overlapping gives effective cycle time of 375 nanoseconds. Two-way interleaving of storage modules reduces the probability of access conflicts.
- Redundancy amongst system components.
- Program address relocation.
- Storage protection.
- Partial word addressability in 6, 9, 12, and 18 bit bytes as well as fullword and doubleword addressing.
- Priveleged mode for Executive system.
- Guard mode for user programs.

REFERENCES

1. Univac Processor and Storage, Programmers Reference. Section 6.6. Search and Masked Search Instructions, page 6-30.
2. Ibid. Section 6.14.3. Load Processor State, page 6-85.
3. Ibid. Section 6.14.11. Load Last Address Register, page 6-92.
4. Ibid. Section 7. Input/Output, page 7-1.
5. Ibid. Section 3.2. Main Storage, page 3-1.
6. Ibid. Section 3.3. Control Section, page 3-10.
7. Ibid. Section 4. CPU Arithmetic Section, page4-1.
8. Univac 1108 Multi-processor System, Operating System Exec 8, Programmers Reference.
9. Univac 1106/1108 Multi-processor System, Assembler Language, Programmers Reference.
10. Univac 1108 Processor and Storage, Programmers Reference, Section 6. Instruction Repertoire, page 6-1.
11. Univac 1108 Multi-processor System, System Description.
12. Shaw. A.C. Lecture Notes on a Course in Systems Programming, Computer Science Department, Stanford University.