A MICROPROGRAMMED MIX 1009
EMULATOR FOR THE MICRODATA 1600/30 COMPUTER

A Thesis

Presented to

the Faculty of the Department of Computer Science

University of Houston

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

---

by

T. Don Dennis

August, 1975

A MICROPROGRAMMED MIX 1009
EMULATOR FOR THE MICRODATA 1600/30 COMPUTER

A Thesis

Presented to

the Faculty of the Department of Computer Science

University of Houston

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

by

T. Don Dennis

August, 1975

ABSTRACT

The design and implementation of a MIX 1009 emulator for the Microdata 1600/30 are presented. Major design alternatives such as allocation of file registers, allocation of main memory, selection of byte sizes and codes are presented in detail.

Insights from false starts are treated as valuable experiences. The evolution of the system involved one major false start as well as many minor ones. The major false start is discussed in an entire chapter and the minor ones are discussed throughout.

Major firmware logic problems are also discussed in detail. The final system is presented through discussion, a users manual, system flowcharts and listing of the microcode.

TABLE OF CONTENTS

I. INTRODUCTION

Computer Science educators often discuss which computers should be studied in introductory classes involving machine-level programming. Although there is no unanimous agreement, many feel that the computer itself is of no importance so long as it provides a typical example of "machine language".

Donald Knuth has noted the following:

> "There has been some feeling that it is advantageous to have a 'machine-independent machine' which does not change from year to year, and which does not have too many idosyncrasies that tend to waste classroom time." (1)

Knuth calls his machine MIX. MIX is designed to be a computer "which is very much like nearly every computer now in existence (except that it is, perhaps nicer). The language of MIX has been designed to be powerful enough to allow brief programs to be written for most algorithms, yet simple enough so that its operations are easily learned." (1)

The justification for MIX then, is that it satisfies the need for a generalized machine and language to be used as a teaching aid in introductory programming classes.

The step following the design of the computer, is the implementation of the machine. Students can then test their programs and gain a deeper understanding of the problems of computing.

There are at present 3 methods for realizing any machine design:

1. Build the computer.
2. Emulate the computer by means of firmware.
3. Simulate the computer via a software package.

Since MIX is meant to be a teaching tool to be used in an educational environment, a hardware implementation would be difficult to justify in terms of the time and money required to achieve such a computer. Most educational environments have large scale and small scale computer systems readily available for program development, thus either method 2 or method 3 seem to be the proper direction to proceed.

Implementation of MIX via software, either on a large scale computer or a mini-computer, is feasible. This approach has several advantages and disadvantages. If the simulation were done on a large scale system, then the simulator as well as the MIX assembler might be written in a high level language, thus making program development easier. There would be no problem simulating all of MIX memory and the closed shop practices imposed on most large systems might produce faster turn around. However, the simulator would be slow since it must first assemble the MIX assembly language into MIX machine code, and then execute the MIX machine code. The execution of each MIX machine instruction entails the execution of many host machine instructions, the inefficiency of simulation somewhat offsets its advantages, particularly on a large computer, since expensive system resources are tied up for relatively long periods of time while MIX programs are running. The advantages of simulating in a closed shop are also diminished since students are not allowed to touch the machine. Sometimes this fosters the "Big Black Box" concept of computing.

The "Big Black Box" problem is solved by simulating on a mini-computer. Most small computers are batch systems, but many are console-mode or hands-on systems, i.e. the students must operate the machine themselves. Small machines may be easily dedicated to simulating MIX since resources are less expensive. Nevertheless, there are problems. Hands-on operation does improve the students concept of the computer, but through-put is demolished since each student must learn to operate the machine by trial and error. Simulating MIX and its 4K word (31 bit) memory is at least troublesome since most mini-computers have limited main memory. This implies that programming would of necessity be done in assembly language to conserve as much memory as possible. However, assembly language programming of a large program is much harder than coding the same problem in a high level language. Here the simulator would be slower than on a large scale machine since mini-computers usually have longer execution times per instruction than large machines.

Despite these disadvantages MIXAL simulators have been written and used successfully.

The second method seems to be more advantageous if the computer is microprogrammable. There are two main problems in this approach. As noted earlier, program development is most easily accomplished in a high level language. Assembly language programming affords some savings in program size, but requires more effort on the part of the programmer. Microprogramming however, is the worst case with respect to program development. The code is tedious to write and difficult to debug. The microprogrammer must work at the control signal level, armed with a very limited instruction set. If the microprogramming system uses a fixed read-only-memory (ROM), a software simulator must be available for development. In this case, implementation may be costly since a new ROM must be built for the new MIXAL emulator once it is debugged. However, if the mini-computer has an alterable control memory(ACM) the problems of implementation are lessened considerably.

It should be noted that by working on a small machine all the advantages of a mini-computer are retained. By emulating on a small system instead of simulating, many of the problems formerly discussed are resolved. The difficulty concerning the limited memory of mini-computers is eased by emulation since the microprogram resides in control memory leaving the main memory completely free. Thus MIX's 4K words of memory might be emulated if the mini-computer has at least that much main memory. The problem of execution time (per MIX instruction) is also solved since the microcoded MIX instructions will run much faster than MIX macrocoded MIX instructions. Aside from solving these problems, emulation results in possibilities not even considered when simulating. Once implemented, the user has a MIX computer. The machine is as much a MIX 1009 computer as any of the originally announced IBM system 360 computers are IBM system 360 computers. The hardware of the different models of the 360 are in no way alike. They are all microprogrammed, except for the model 70, to execute the same machine language. Once the firmware is coded the natural language of the host machine is MIXAL so the MIX assembler can be written in MIXAL, the loader can be written in MIXAL, in fact a whole operating system can now be written in MIXAL with none of the system degradation that would result if implementation was by simulation.

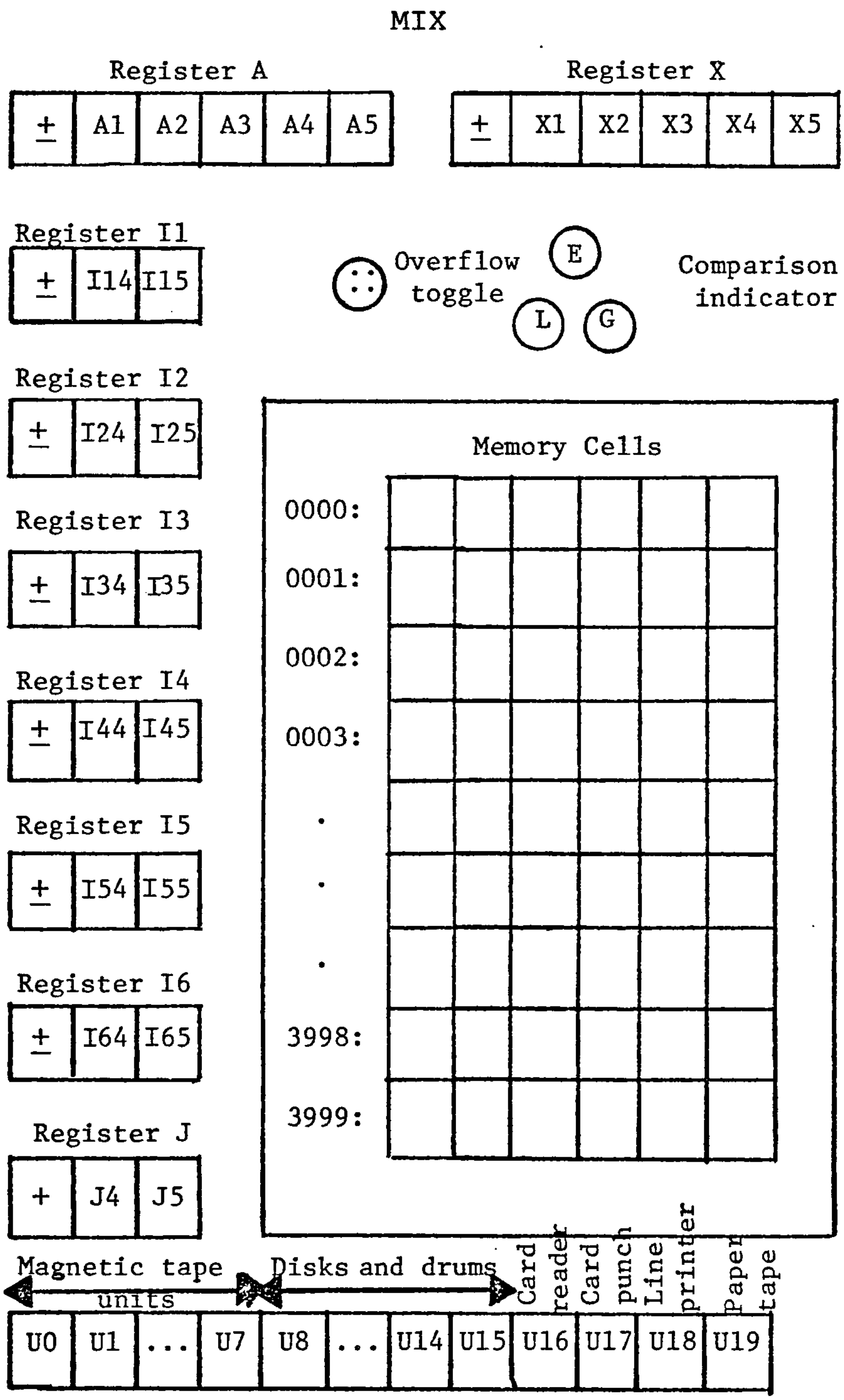
This thesis reports the emulation of the MIX 1009 machine by a Microdata 1600/30. The discussion which follows covers the high points of both machines. If more detailed information is required, see references (1) and (5), for MIX and (3) for the Microdata.

MIX was designed with the "peculiar property..... that it is both binary and decimal at the same time. The programmer does not actually know whether he is programming a machine with base 2 or base 10 arithmetic." (1) This was accomplished by not specifying the amount of information which can be contained in a single byte. The only specifications given is that each byte should be capable of holding at least sixty-four values, and at most 100 values. As long as programs are written "so that no more than sixty-four values are ever assumed for a byte ... An algorithm in MIX should work properly regardless of how big a byte is..." (1) Figure 1.1 presents an overview of the major components of the MIX 1009 computer.

MIX memory consist of 4000 words of storage. Each MIX word is composed of five bytes and a sign, the sign has only two values + or -. Values are stored in sign plus magnitude format instead of the one's complement or two's complement usually found in binary machines or the nine's or ten's complement used on decimal machines.

The 1009 computer has nine registers that are available to the user. The accumulator, (A-register), is a five byte plus sign register used to perform the basic arithmetic operations, add, subtract, multiply, and divide, as well as data manipulation. The X-register is the right hand extension of the A-register and it is also five bytes plus sign. It is used in the multiply and divide instructions in

Figure 1.1


MIX
Register A
± A1 A2 A3 A4 A5
Register X
± X1 X2 X3 X4 X5
Register I1
± I14 I15
Overflow toggle
E
L
G
Comparison indicator
Register I2
± I24 I25
Register I3
± I34 I35
Register I4
± I44 I45
Register I5
± I54 I55
Register I6
± I64 I65
Register J
+ J4 J5
Memory Cells
0000:
0001:
0002:
0003:
3998:
3999:
Magnetic tape units
Disks and drums
Card reader
Card punch
Line printer
Paper tape
U0 U1 ... U7 U8 ... U14 U15 U16 U17 U18 U19

(1)

connection with the A-register to hold the ten byte product or dividend. It is also used in shift commands when ten bytes are to be shifted at once. The X-register can, however, be used separately as a limited accumulator.

I1, I2, I3, I4, I5, and I6 are six index registers. They are used in address modification and in counting. Each index register is two bytes plus sign. The J-register, Jump address register, was designed to provide support for subroutine linkage. It is also a two byte plus sign register and is loaded automatically with the contents of the instruction counter immediately prior to the execution of any Jump instruction, except a JSJ, Jump and Save J instruction.

In addition to these nine registers MIX has an overflow toggle, which is either set or reset, and a comparison indicator which may assume one of three states, representing less, equal, and greater.

MIX was designed to accomodate twenty I/O devices. Units 0-7 are dedicated to magnetic tape, units 8-15 to disks and drums, unit 16 to the card reader, unit 17 to the card punch, unit 18 to the line printer, and unit 19 is reserved for typewriter and a paper tape station.

Most instructions in MIX allow partial fields of words to be selected as the instruction operand. Each word can be broken into six fields as follows:

0	1	2	3	4	5
sign	byte	byte	byte	byte	byte

The particular field or fields which the programmer wishes to use is then encoded in a field specification. Any specification is legal so long as it addresses contiguous fields of the operand. The notation used to express partial fields is (L:R), where L is the number of the left-most field and R is that of the right-most field being specified. Typical examples of MIX's partial fields are:

(0:0), the sign only;
(0:3), the sign and high order 3 bytes;
(0:5), the entire word;
(1:5), the whole word except for the sign;
(2:2), the second byte;
(4:5), the low order 2 bytes.

There are 21 allowable specifications in all, they are:

(0:0)					
(0:1)	(1:1)				
(0:2)	(1:2)	(2:2)			
(0:3)	(1:3)	(2:3)	(3:3)		
(0:4)	(1:4)	(2:4)	(3:4)	(4:4)	
(0:5)	(1:5)	(2:5)	(3:5)	(4:5)	(5:5)

Computer instructions are formated in MIX as follows:

0	1	2	3	4	5
S	A	A	I	F	C

The first three fields, (0:2), of the word form the operand address, the I-field following the address field is used for operand address modification via indexing. If I is zero, no modification occurs and the value in fields (0:2) is the effective memory address of the operand. If I is non-zero it should have a value, i, between 1 and 6. The effective operand address, M, is computed to be the algebraic sum of Index register Ii plus ± AA. The effective address is formed this way on all MIX instructions. It should be noted that in most cases $0 \leq M \leq 3999$, since MIX has 4000 memory locations. However, in some instances M may be outside this range, and indeed be negative. For example, the ENTA instruction, (Enter A), causes the accumulator to be loaded with the value of M.

The right-most two bytes of each instruction explicitly state what operation is to be carried out. The C-field denotes the operation code, while the F-field modifies this opcode. In most cases the F-field contains the partial field designation (L:R) which is encoded as 8L + R. However, the F field has other uses. For example in the Move instruction, F specifies the number of words to transfer. In input-output operators, F is the unit number of the selected device. The F-field is also used as a secondary operation code, which further defines the operation to be performed. Consider opcode 48:

C=48, F=0 is the increment A command, while
C=48, F=1 is the decrement A command.

The following chart, figure 1.2 is a brief description of the MIX instruction set.

The Microdata 1600/30 used to emulate MIX has 32K bytes of main memory. This magnetic core memory has a one microsecond cycle time, is byte addressable, with 8-bit bytes. There are 2K bytes (16 bit/bytes) of semiconductor control memory which have a 200 nanosecond cycle time. I/O devices include

Figure 1.2

General form:

C T
Description OP(F)

C = operation code, (5:5) field of instruction
F = op variant, (4:4) field of instruction
M = address of instruction after indexing
V = F(M) = contents of F field of location M
OP = symbolic name for operation
(F) = standard F setting
t = execution time; T = interlock time

	[*]:		[+]:
rA = register A	JL(4)		N(0)
rX = register X	JE(5)	=	Z(1)
rAX = registers AX as one	JG(6)		P(2)
rIi = index reg. i, $1 \leq i \leq 6$	JGE(7)	=	NN(3)
rJ = register J	JNE(8)	=	NZ(4)
CI = comparison indicator	JLE(9)	=	NP(5)

Figure 1.2 Cont.

00 1	01 2	02 2	03 10
No Operation NOP(0)	rA rA + V ADD(0:5)	rA rA - V SUB(0:5)	rAX rA X V MUL(0:5)
08 2	09 2	10 2	11 2
rA V LDA(0:5)	rI1 V LD1(0:5)	rI2 V LD2(0:5)	rI3 V LD3(0:5)
16 2	17 2	18 2	19 2
rA - V LDAN(0:5)	rI1 - V LD1N(0:5)	rI2 - V LD2N(0:5)	rI3 - V LD3N(0:5)
24 2	25 2	26 2	27 2
F(M) rA STA(0:5)	F(M) rI1 ST1(0:5)	F(M) rI2 ST2(0:5)	F(M) rI3 ST3(0:5)
32 2	33 2	34 1	35 1 + T
F(M) rJ STJ(0:2)	F(M) 0 STZ(0:5)	Unit F Busy? JBUS(0)	Control, Unit F IOC(0)
40 1	41 1	42 1	43 1
rA:0,jump JA[+]	rI1:0,jump J1[+]	rI2:0,jump J2[+]	rI3:0,jump J3[+]
48 1	49 1	50 1	51 1

rA [rA]?±M
INCA(0)DECA(1)
ENTA(2)ENNA(3)
	rI1 [rI1]?±M
	INC1(0)DEC1(1)
	ENT1 (2)ENN1(3)
		rI2 [rI2]?± M
		INC2(0)DEC2(1)
		ENT(2)ENN2(3)
			rI3 [rI3]?± M
			INC3(0)DEC(1)
			ENT(3)ENN(3)

56 2	57 2	58 2	59 2
rA(F):V CI CMPA(0:5)	rI1(F):V CI CMP1(0:5)	rI2(F):V CI CMP2(0:5)	rI3(F):V CI CMP3(0:5)

Figure 1.2 Cont.

04 12	05 1	06 2	07 1+ 2F

rA rAX/V
rX remainder
DIV(0:5)
	Special
	NUM(0)
	CHAR(1)
	HLT(2)
		Shift M bytes
		SLA(0) SRA(1)
		SLAX(2) SRAX(3)
		SLC(4) SRC(5)
			Move F words
			from M to rI1
			MOVE(1)

12 2	13 2	14 2	15 2
rI4 V LD4(0:5)	rI5 V LD5(0:5)	rI6 V LD6(0:5)	r X V LDX(0:5)
20 2	21 2	22 2	23 2
rI4 - V LD4N(0:5)	rI5 - V LD5N(0:5)	rI6 - V LD6N(0:5)	r X - V LDXN(0:5)
28 2	29 2	30 2	31 2
F(M) rI4 ST4(0:5)	F(M) rI5 ST5(0:5)	F(M) rI6 ST6(0:5)	F(M) rX STX(0:5)
36 1+T	37 1	38 1	39 1

Input, unit F
IN(0)
	Output, unit F
	OUT(0)
		Unit F ready?
		JRED(0)
			Jumps
			JMP(0) JSJ(1)
			JOV(2) JNOV(3)
			also [*] above

44 1	45 1	46 1	47 1
rI4:0,jump J4[+]	rI5:0,jump J5[+]	rI6:0,jump J6[+]	rX:0,jump J7[+]
52 1	53 1	54 1	55 1

rI4 [rI4]?±M
INC4(0)DEC4($\overline{1}$)
ENT4(2) ENNA(3)
	rI5 [rI5]?±M
	INC5(0)DEC5($\overline{1}$)
	ENT5(2)ENN5(3)
		rI6 [rI6]?±M
		INC6(0)DEC($\overline{1}$)
		ENT6(2)ENN6(3)
			rX [rX[?± M
			INCX(0)DEC$\overline{X}$(1)
			ENTX(2)ENNX(3)

60 2	61 2	62 2	63 2
rI4(F):V CI CMP4(0:5)	rI5(F):V CI CMP5(0:5)	rI6(F):V CI CMP6(0:5)	rX(F):V CI CMPX(0:5)

(1)

a 500 LPM line printer, a 300 CPM card reader, a magnetic tape unit, two disk drives, a teletype writer, and a paper tape station.

The 1600/30's control memory continuously executes stored microcommands to time and regulate all control and data operations required by the MIX computer. "Using application programming at the micro level, the Micro 1600 can be used directly as a hardwired controller. When the 1600 emulates the operation of a general purpose computer which executes software instructions stored in core memory, macro-instructions are fetched and interpreted by the microprogram with corresponding operations carried out by execution of microprogrammed routines in the control memory." (4)

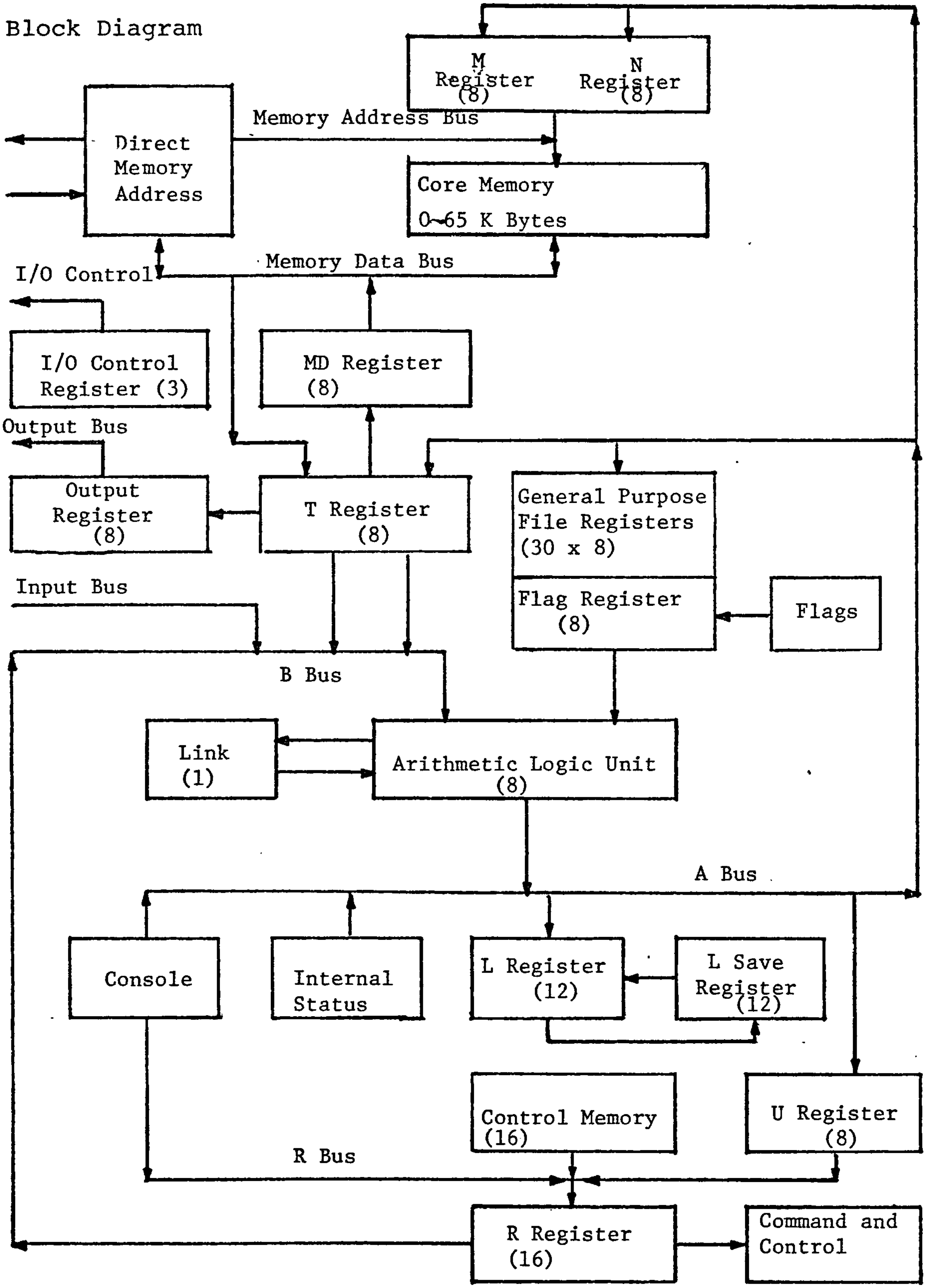
Eight-bit data paths and eight-bit registers are incorporated in the Microdata. A 16-bit micro-instruction is executed every 200 nanoseconds from control memory. Figure 1.3 provides a block diagram of the Microdata 1600/30 at the register level.

Registers

The T-register is one of the main input operands to the eight-bit Arithmetic/Logic Unit (ALU). The T-register is also used in input-output operations and in memory read and memory write operations as a buffer register. Operate type microcommands require the T-register be selected in one of four forms, the mnemonics for these four forms are O, T, F,

Figure 1.3

Micro 1600
Block Diagram



(3)

and F, T. If O is coded then the selected operand transfered to the B-bus will be zero. The mnemonic T indicates the true value of the T-register transfered. F selects the complement of the T-register. Coding both F,T causes the B-bus to be all ones.

The MD register, Memory Data register, is an 8-bit buffer used to hold data being written out to the main memory. It receives input automatically from the T-register 350 nanoseconds after the initiation of a memory write. The MD register is not directly available to the programmer but was designed to free the T-register faster than would be possible otherwise.

The M and N registers, both 8-bits long, hold the 16-bit memory address used in memory read and memory write operations. M holds the 8 most significant bits; N holds the eight least significant bits.

Input-output control signals are regulated, under program control, by the 3-bit IC register. All device controllers are connected to this register via the I/O control bus, allowing device controllers to receive and decode signals from the IC register. Settings of 1, 2, or 3 are decoded as output signals and values of 4,5,6, and 7 are input signals. When an input value is in the IC register, the input bus, rather than the T-register is the operand gated to the B-bus.

The OD register, Output Data register, was designed with a purpose similar to that of the MD register. The OD register automatically copies the T register whenever the IC register is set non-zero, thus freeing the T register for other purposes.

The R register is the Microdata's microinstruction register. It holds the 16 bit microcommand currently being executed. The R register receives input from control memory over the R-bus.

The eight high-order bits of the next microcommand to be executed may be modified through the use of the U register. When selected by the microcommand, the 8 bit U register is ORed with the control memory output prior to input to the R register. This allows the generation of efficient code since routines which differ by only a few instructions may use common subroutines but with different settings of the U register. For example the following code will add,(opcode 8), the T register to file 1:

```
LU     X'00'      Load U with Zeros
ADD    1, T, (S)  Or U with opcode, add T to
                    file 1.
```

By changing the value of U from X'00' to X'90' or X'10' the same add instruction will cause a subtraction since a subtract is opcode 9:

```
LU     X'90'      Load with X'90'
ADD    1, T, (S)  Or  U with opcode, subtract T from
                    file 1.
```

The L register is the 12-bit microinstruction counter. It addresses the next command to be executed and can provide control over 4K of control memory. This register can be altered by executing a Jump instruction, which loads the operand address, or by selecting the L register as the destination for the output from the ALU.

The L Save register is also a 12-bit register, and it provides for one level microsubroutines. It copies the contents of the L register whenever a Jump Extended instruction is executed. After the subroutine has been performed a return instruction causes the L Save register to be copied back into the L register and processing continues.

The Link register is a 2-bit register which holds the high order carry-out from the Arithmetic/Logic Unit. The Arithmetic Link (AL) bit of the Link register is the bit usually selected. The exception occurs when the output from the ALU is directed to the M and N registers, in this case the Memory Link (ML) bit is used.

All the above registers were designed with a specific function in mind. However, the Microdata 1600/30 also provides two files of general purpose registers. These files, denoted the Primary file and the Secondary file, each contain fifteen 8-bit registers. Only one bank of registers may be addressed at any given time and selection of the Primary or Secondary file is under program control. Input to these registers is from the A-bus and output is through the ALU.

Register O is dedicated to ALU condition flags (bits 0, 1, 2) and internal status bits (3-7) and is common to both banks. Register O is a read only file, and readout does not effect its contents. The 8-bits of register 0 are described below:

0----Overflow condition (ALU)
1----Negative condition (ALU)
2----Zero condition (ALU)
3----I/O request flag
4----Internal interrupt flag
5----I/O reply flag
6----Serial TTy
7----External interrupt flag

The remaining "30 general-puropse file registers... are implemented with MSl/LSl semiconductor devices." (3) In the emulation of MIX these file registers are assigned, in groups, the functions of the A register, X register, Index register, Jump register, Instruction register and the Instruction counter as well as providing free work areas.

It should be noted that the Von Neumann concept of memory is absent in microprogramming. In a Von Neumann machine data and instructions are intermixed in memory, in fact instructions can be manipulated as data during one phase of the program and later executed as an instruction. In any case memory is a general purpose storage device containing both instructions and data. In microprogramming, however, control memory is almost always read-only. Thus temporary storage areas (i.e. data) and programming areas (i.e. instructions) are completely separate.

Instructions are confined to an area called control memory, while temporary storage and work areas are located in another, usually very small memory, which is backed up by main memory. In the case of the Microdata this small memory takes the form of these 30 general purpose file registers.

Data Flow:

There are 3 main paths in the Microdata which supply data to the different registers and the ALU. The R-bus provides input to the R register (microinstruction register). Data is gated to the R-bus from three possible sources, control memory, control memory ORed with the U register, and the console panel switches. Only one source may be selected per clock pulse. The B-bus, the second operand to the ALU, is supplied data form either the T register, in true or complemented form, the Input-bus, or the R register. The R register is selected when a literal is gated to the B-bus. The A-bus is the main data bus in the Microdata. It receives input from the ALU primarily, but the internal status or console may also be selected. The data on the A-bus can be transfered to any file register and simultaneously to the L register, U register, T register, M register, or N register.

The Arithmetic/Logic Unit (ALU), an 8-bit unit, is the center of data manipulation in the 1600/30. Its operations include addition, subtraction, and or Exclusive-OR, shifting, and data transfer. The selected file register and the B-bus provide the operands for the ALU and output is placed on the A-bus, which is a common source of input to most registers.

Instruction Repertoire:

The microdata's microcommand repertoire consist of 65 instructions. Each instruction is classified as either a literal command, an operate command or a generic command depending on the commands format. The five possible formats are displayed below along with examples of each format type.

Literal Command

OP- Operation Code
F - File register designator
Literal - 8-bit or 12-bit literal which is
transfered as an operand

Type 1

15 14 13 12	11 10 9 8	7 6 5 4 3 2 1 0
OP	F	Literal

Example:

AF 7,X'04'

3	7	0	4

ADD the value X'04' to file register 7

Type 2

15 14 13 12 11 10 9 8	7 6 5 4 3 2 1 0
OP	Literal

Example:

LT X'56'

1	1	5	6

LOAD T register with X'56'

Type 3

15 14 13 12	11 10 9 8 7 6 5 4 3 2 1 0
OP	Literal

Example:

JE X'621'

0	6	2	1

JUMP to location 621

Operate Commands

OP- Operation Code

F - File Register Designator

C - Control Field Designation

Designator		Definition
L	-	Link Control/ADD Link
C	-	Modify Condition Codes
T	-	Select T Register
F	-	Select T Complement

(continued on the next page)

Designator		Definition
I	-	Increment
D	-	Decrement

* File inhibit - If bit 3 is a one, the file register F is unchanged.

- If bit 3 is a zero, the file register F is loaded with the result of the command (i.e. A-Bus).

R - Distination Register

Designator		Register Designated
blank	-	None
T	-	T Register
M	-	M Register
N	-	N Register
L	-	L Register (even address pages)
K	-	L Register (odd address pages)
U	-	U Register
S	-	U Register is ORed into upper 8 bits of operate command

Type 4

15 14 13 12	11 10 9 8	7 6 5 4	3	2 1 0
OP	F	C	*	R

Example:

ADD* 13, T, L, C, (U)

8	D	B	E

ADD the T register and Link bit to file register 13, Set the condition flags in file zero and place the sum in the U register. File 13 is not updated.

Generic Commands

OP- Operation Code

OP

Example:

SPF Select Primary File

1	0	4	0

II. THE FIRST ATTEMPT

In constructing the MIX emulator some design problems were encountered in meeting Knuth's specifications for MIX. The design problems fall into two groups. The first concerns the allocation of Microdata hardware for the emulation of MIX hardware. The second type of problems involve the development of the firmware logic required by the MIX instructions set.

Two attempts were made to construct a MIX emulator. The first attempt employed a Microdata 1600/30 with 16K bytes (8 bit) of core memory and 2K bytes (16 bit) of Alterable Control Memory (ACM). The first attempt was aborted for reasons discussed in this chapter. The first attempt showed that a complete implementation of a MIX machine with 16K bytes of Microdata memory would require more than 2K of ACM to hold the emulator. The second effort used a Microdata 1600/30 with 32K of core and 2K of Alterable Control Memory. This larger configuration resulted in simpler firmware logic and the successful emulation of the MIX 1009 computer.

Although the first attempt was "scraped", much was learned from previous mistakes which was useful in the second attempt. The purpose of this chapter is to relate this learning experience.

Hardware Allocation Problems

The first porblem encountered in designing the MIX emulator was that of deciding the best way to allocate the model 30's memory in implementing MIX's memory. The memory resources available on the Microdata 1600/30 at this time and the memory requirements of the MIX machine are reflected in figure 2.1.

figure 2.1	Words of Memory	Bits/ Byte	Total # of Bytes	Total # of Bits	Character Code	Numeric Code
Microdata 1600/30	Undefined byte Addressable	8	16,364	130,912	ASCII or EBCDIC	binary 2's comp.
1009	4000	1/sign 6/data	4000 sign 20,000 data	124,000	Knuth's Code	binary sign plus magnitude

According to the MIX specifications a MIX machine is word addressable with 4,000 words of core memory. Each word is composed of a sign byte and five data bytes. The sign byte may contain one of two values, representing plus and minus. Each data byte must be capable of containing at least 64 values but not more than 100 values. The minimum number of bits required for a MIX computer is then,

$$4000 \times 31 = 124{,}000 \text{ bits,}$$

(4000 words, each containg a one bit sign byte and 5 six bit data bytes). In June 1974, the Computer Science Department's 1600/30, which is byte addressable, had 16K bytes (8 bits/byte) or 130,912 bits of core memory. Plainly there existed enough bits to emulate the MIX computer, but not enough addressable units or bytes.

At this point three possible boundary allignments were considered as solutions to the memory allocation problem. MIX memory could be represented as six Microdata bytes per MIX word, or as five Microdata bytes per MIX word, or as 4 bytes per MIX word.

Data in the six Microdata bytes per MIX word solution was to be stored as follows;

figure 2.2

S	N	N	N	N	N	N	N
1_1	1_2	1_3	1_4	1_5	1_6	1_7	1_8
2_1	2_2	2_3	2_4	2_5	2_6	2_7	2_8
3_1	3_2	3_3	3_4	3_5	3_6	3_7	3_8
4_1	4_2	4_3	4_4	4_5	4_6	4_7	4_8
5_1	5_2	5_3	5_4	5_5	5_6	5_7	5_8

S - sign bit

N - not used

K_i- i^{th} bit of K^{th} byte

Using this format and going to an eight bit byte the data was easy to manipulate, however only 2,727 words of MIX memory were available with a 16K host machine. The six byte solution also made MIX word boundaries hard to detect. There was also a related problem due to the nature of the IN and OUT commands, MIX's I/O operators. These instructions handle the sign byte separtely from the data bytes and thus need to sense MIX word boundaries. With six bytes/word this can only be done by dividing the current I/O address, (Microdata address), by six and examining the remainder. This process is too lenghtly for interrupt driven I/O.

Six bytes per word also makes address translation, from MIX addresses to Microdata addresses, and vice versa, involved but not difficult. Given any MIX address M the Microdata address, MD, of the first byte of M is simply,

$$MD = 2M + 4M.$$

Assume M is a 16 bit address, the most significant byte (MSB) residing in Primary file 9, and the least significant byte (LSB) in Primary file 10. Then MD will be in P9 and P10 after the execution of the following nine instructions.

Figure 2.3

```
SFL     10          Shift file 10 to left, multiply by two
SFL     9,L,(T)     Shift file 9 left, inserting the bit just
                    Shifted out of file 10, and put the result
                    in the T register
CPY     11,T        Copy the T register into file 11.
MOV     10,(T)      Move the contents of file 10 to the T register
SFL     10          Shift file 10 left, multiply by two again
SFL     9,L         Shift file 9 left, inserting the bit just
                    Shifted out of file 10.

                    Now M x 2 is in P11 and T
                    M x 4 is in P9 and P10
ADD     10,T        Add file 10 and the T register, put result
                    in file 10
MOV     11,(T)      Move contents of file 11 to T.
ADD     9,L,T       ADD file 9 to T along with the high order
                    Carry of the last add, placing the result
                    in file 9.
```

The need to convert the Microdata address back to the corresponding MIX address also arises when the console step switch is pressed. When the step switch is pressed the next instruction is executed, the machine then HALTS and displays the MIX address of the next instruction. However the MIX Instruction Counter actually contains the Microdata address of the first byte of the next instruction. In order that the MIX address be displayed it must first be computed from the Microdata address and this result placed on the data bus. However, the conversion from a Microdata address back to a MIX address involves a division by six. Naturally, the divide algorithm could be used for this purpose, but the divide routine is usually avoided since it is one of the longest routines in the instruction set. It is possible to divide by six fairly rapidly, given that the dividend is evenly divisible by six (which is the case for memory address). This problem becomes the ability to divide by three, since

$$MD/6 = (MD/3)/2$$

The divide by six algorithm is described in figure 2.4 while the corresponding microprogram is described in figure 2.5.

The 6 byte solution offered the advantage of easy data manipulation, assuming 8 bit bytes were used, but the advantage was offset by three disadvantages, namely:

1) Only 2,727 words of MIX memory could be emulated instead of the specified 4,000 words.
2) Input/Output was severly complicated by the

Figure 2.4

Given: The dividend is evenly divisible by 6, then the quotient may be found by;

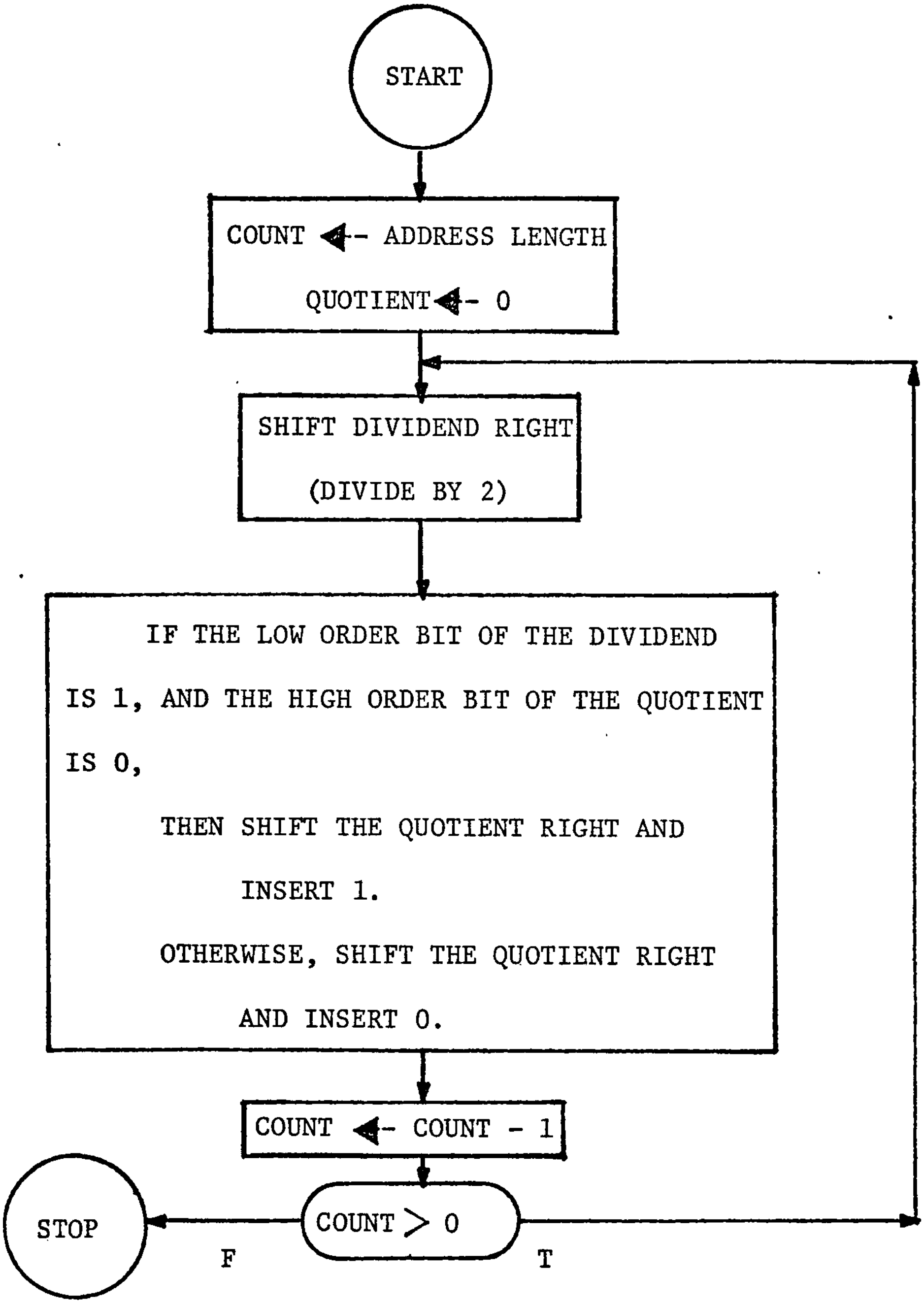

START
COUNT ← ADDRESS LENGTH
QUOTIENT ← 0
SHIFT DIVIDEND RIGHT
(DIVIDE BY 2)
IF THE LOW ORDER BIT OF THE DIVIDEND IS 1, AND THE HIGH ORDER BIT OF THE QUOTIENT IS 0,
THEN SHIFT THE QUOTIENT RIGHT AND INSERT 1.
OTHERWISE, SHIFT THE QUOTIENT RIGHT AND INSERT 0.
COUNT ← COUNT - 1
COUNT > 0
F
T
STOP

Figure 2.5

The corresponding Microprogram follows:

```
*          file 9 contains MSB of address
*          file 10 contains LSB of address
*          result to be placed in file 11 and 12
*          16 bit address

           LF        8,X'10'      LOAD COUNT

           ZOF       11           ZERO QUOTIENT MSB

           ZOF       12           ZERO QUOTIENT LSB

START      SFR        9
                                  DIVIDE DIVIDEND BY TWO

           SFR       10,L

           TN        10,X'01'     TEST LOW ORDER BIT OF
                                       DIVIDEND

           SP        ZERO         JUMP IF ZERO

           TN        11,X'80'     TEST HIGH ORDER BIT OF
                                       QUOTIENT

           JP        ZERO         JUMP IF ONE

ONE        SRI       11           SHIFT QUOTIENT, INSERT 1

           JP        SHIFT

ZERO       SFR       11           SHIFT QUOTIENT, INSERT0

SHIFT      SFR       12,L

           DEC        8           COUNT ——— COUNT - 1

           TZ        8,X'FF'      COUNT  0 ?

           JD        START        COUNT IS  0

           HLT                    COUNT = 0
```

To perform this algorithm on a 16 bit address 211 instructions must be executed.

inability to detect MIX word boundaries.

3) Address translation from MIX address to Microdata address and back again would be time consuming.

Data for the five byte per word solution was to be stored as shown in the illustration below. The MIX sign byte and first MIX data byte were to occupy the first Microdata byte with the remaining four bytes being stored in the next four Microdata bytes.

Figure 2.6

S	1_1	1_2	1_3	1_4	1_5	1_6	1_7
2_1	2_2	2_3	2_4	2_5	2_6	2_7	2_8
3_1	3_2	3_3	3_4	3_5	3_6	3_7	3_8
4_1	4_2	4_3	4_4	4_5	4_6	4_7	4_8
5_1	5_2	5_3	5_4	5_5	5_6	5_7	5_8

S - Sign bit

Ki- i^{th} bit of K^{th} byte

Using this format and again assuming 8 bits per byte, data was easy to manipulate. However, the first Microdata byte must be processed separately, since the Microdata's hardware employee's 2's complement arithmetic and MIX is a sign plus magnitude machine. Aside from this, the 5 byte solution has the same advantages and disadvantages as the 6 byte solution. Here, 3,272 MIX words can be emulated, and addresses are again a problem but data is easy to handle.

The major defect with both the 6 byte and the 5 byte solution was that all 4000 words of MIX memory could not be emulated. In light of this fact, if the entire 4000 words of MIX memory were to be emulated then the 31-bit MIX words must be packed into four Microdata bytes.

1 MIX word = 31 bits 4 Microdata bytes = 32 bits

4000 x 32 = 128,000 bits 130,912 bits = 16K x 8 bits

Using this approach the whole MIX memory could be emulated with 384 Microdata bytes left over. The information was to be packed according to the diagram below.

Figure 2.7

S	N	1_1	1_2	1_3	1_4	1_5	1_6
2_1	2_2	2_3	2_4	2_5	2_6	3_1	3_2
3_3	3_4	3_5	3_6	4_1	4_2	4_3	4_4
4_5	4_6	5_1	5_2	5_3	5_4	5_5	5_6

S - Sign bit
N - Not used
Ki- i^{th} bit of K^{th} byte.

This format solved the 3 problems found with the 5 byte and 6.byte formats. First, all 4000 words of MIX memory could be implemented on the host machine's current 16K memory. Second, word boundaries were easy to identify since any Microdata address whose low order two bits are zero corresponds to the first byte of a MIX word. Finally, address translation, either from MIX addresses to Microdata addresses or vice versa, could be performed by simple register shifts (i.e. multipling or dividing by 4). However in eliminating these three problems the main advantage of the previous two solutions was also eliminated, for data was no longer easy to manipulate. In fact, this packed format caused data manipulation to now become 'the' major firmware logic problem.

The second hardware allocation problem concerned the mapping of MIX's registers into the 30 general purpose file registers that are available on the Microdata 1600/30. There are nine registers available to the user in MIX plus an overflow toggle and a Comparison Indicator. There is also an Instruction Counter and an Instruction Register, although these are not directly accessable to the user. All these registers must be represented by the 30 general purpose file registers. Main memory is accessible, of course, from the microlevel but storage and retrieval is involved. The following example illustrates this point.

Figure 2.8

```
*         File 9 contains MSB of Memory Address
*         File 10 contains LSB of Memory Address
*         File 8 contains data
*
*         Store a byte in Main Memory
          MOV     9,(M)        Load MSB of Memory Ad-
                               dress Register (MAR)
          WMF     10,(N)       Load LSB of MAR, Begin
                               full cycle write
          MOV     8,(T)        Move data to T in time
                               to be written out
*
*         Retrieve a byte from Main Memory
          MOV     9,(M)        Load MSB of MAR
          RMF     10,(N)       Load LSB of MAR, Begin
                               full cycle read
          NOP                  Delay 200 nanoseconds
          CPY     8,T          Copy data from T into
                               file 8
```

Clearly main memory is not a good place to emulate registers of a target machine or to store temporary results, such as firmware loop counters. Recall that these thirty general purpose files compose the only scratch pad available to the microprogrammer, beside main memory, since the Alterable Control Memory (ACM) is read-only when used as a control memory. Thus these thirty files must serve not only as registers for the MIX computer but must also provide the microprogrammer with a fast work area to perform the needed firmware routines.

The MIX Accumulator A, its right hand extension X, and the Instruction Register are the same length as a MIX word, a sign byte plus five data bytes. These six MIX bytes were packed into four Microdata bytes as shown in figure 2.9.

Figure 2.9

A register
X register and
Instruction register

S	N	1_1	1_2	1_3	1_4	1_5	1_6
2_1	2_2	2_3	2_4	2_5	2_6	3_1	3_2
3_3	3_4	3_5	3_6	4_1	4_2	4_3	4_4
4_5	4_6	5_1	5_2	5_3	5_4	5_5	5_6

S- Sign
N- Not used
Ki- i^{th} bit of the K^{th} byte

The Instruction Counter is a two byte register and the remaining seven registers, the Jump register, and the six Index registers are three bytes each in MIX, a sign byte plus two data bytes. Each of these registers was packed into two Microdata bytes as shown in figure 2.10.

Figure 2.10

Instruction
Counter
Jump and
Index registers

S	N	4_1	4_2	4_3	4_4	4_5	4_6
5_1	5_2	5_3	5_4	5_5	5_6	0	0

S - Sign
N - Not used
O - Zero
Ki- i^{th} bit of K^{th} byte

This format was selected for several reasons. First, by carrying the Instruction Counter in this form the Microdata Address, (MIX address times 4), of the next instruction was readily available. Secondly, this format facilitated indexing; Recall that the sign and first two data bytes of an instruction compose the operand address. From figure 2.9 it can be seen that the operand address, in packed form, is in the same format as the Index register. (Figure 2.10). Computing the effective (Microdata) operand address can be accomplished by masking the address field from the Instruction register, zero filling the low order two bits, and adding this result to the specified index register. Thirdly, MIX Jump instructions, which may be indexed, are easy to execute since the address field of the instruction, the Index register, and the Instruction Counter are all packed the same way.

The Overflow toggle is a one bit register in MIX which is either set or reset. The MIX Comparison Indicator can assume one of three values representing greater, less, and equal conditions. These two MIX registers were packed into one Microdata file as shown in figure 2.11

Figure 2.11

N	N	N	O	N	L	E	G

N - not used
O - overflow
LEG - Comparison Indicator

Three bits were used to emulate the MIX Comparison Indicator although only two bits were needed to represent the three possible states. However, a three bit Comparison Indicator allows easier programming of the Jump Less, Jump Equal, Jump Greater, Jump Less or Equal, Jump Greater or Equal, and Jump Not Equal instructions. Using a three bit Comparison Indicator one general microprogram can be written to decide whether the correct conditions exist for each of these six Jump instructions. To test for a less than condition a mask of '0000 0100' is passed to the compare routine, which OR's this mask with the file containing the Comparison Indicator. If the logical result is non-zero, then the Less bit is on indicating a less than condition. The Equal and Greater cases work the same way. The advantage of a three bit indicator is made apparent by the Jump instructions which test for two conditions instead of one. To test for a greater than or equal condition a mask of '0000 0011' is passed to the compare routine. A not equal conditon can be stated as a less than or greater than condition, therefore, a mask of '0000 0101' will test for not equal. Figure 2.12 gives the conditions and the corresponding masks to be used with this method.

Figure 2.12

G - Greater bit
E - Equal bit
L - Less bit
O - Overflow bit

N	N	N	O	N	L	E	G	
0	0	0	0	0	0	0	1	Test Greater
0	0	0	0	0	0	1	0	Test Equal
0	0	0	0	0	1	0	0	Test Less
0	0	0	0	0	0	1	1	Test Greater or Equal
0	0	0	0	0	1	1	0	Test Less or Equal
0	0	0	0	0	1	0	1	Test Not Equal

Using the three formats just described (Figures 2.9, 2.10, and 2.11) twenty-nine microdata files are required to emulate MIX registers (Figures 2.13). This leaves only one free work file to be used by the microprogrammer. However the third byte of a MIX instruction denotes which Index register, if any, is to be used to compute the effective operand address. This computation is done on all instructions immediately following the instruction fetch. Thus by the time the decode routine is executed MIX byte 3 is free to be used by the microprogrammer. MIX byte 5, the instruction operation code, becomes available to the microprogrammer after instruction decode has occured. Also MIX byte 4, the F field (partial word designator), is freed shortly after entering the particular instruction subroutine to be executed. Therefore, though work space is at a premium, enough scratch files are available to perform most computations.

Figure 2.13

A register	4 file registers	Dedicated to MIX
X register	4 file registers	
Instruction register	4 file registers	
Instruction counter	2 file registers	
Jump register	2 file registers	
Index register I1	2 file registers	
Index register I2	2 file registers	
Index register I3	2 file registers	
Index register I4	2 file registers	
Index register I5	2 file registers	
Index register I6	2 file registers	
Overflow toggle and Comparison Indicator	1 file register	Available to Microprogrammer
Free work area	1 file register	
	30 file registers	

The thirty general purpose registers on the Microdata 1600/30 are divided into two files of 15 registers. Each, refered to as the Primary file and the Secondary file. Only one file is available to the microprogrammer at any given time. To get from one file to the other a file select instruction must be executed. Figure 2.14 illustrates the addressing and manipulation of the two sets of file registers.

Figure 2.14

```
*    Transfer the contents of Primary file 1 (P1) to
*        Secondary file 1 (S1)

*    Transfer the contents of Secondary file  15 (S15)
*        to Primary file 15 (P15)
*


     SPF                  Select Primary File
     MOV    1,(T)         Move P1 to T register
     SSF                  Select Secondary files
     CPY    1,T           Copy T register into S1
     MOV    15,(T)        Move S15 to T register
     SPF                  Select Primary files
     CPY    15,T          Copy T register into P15
```

Data transfer between the two sets of files is cumbersome for two reasons. First, transfer must be via the T register since it is the only register common to both files which can be loaded and then read. (The U, M, and N registers can only be loaded). Thus transfers must take place one byte at a time. Secondly, file select commands must be issued each time the file boundary is to be crossed. As a result, MIX registers which are likely to be used together were grouped in the same file to avoid inter-file transfers. The Instruction Counter, the Instruction register, the A register, the X register, and the one free work register were assinged to the Primary file while the Index registers, the Jump register, the Overflow toggle and the Comparison Indicator were assigned to the secondary file.

The Instruction Counter and the Instruction Register were both assigned to the Primary file to facilitate the instruction fetch cycle. Note, to fetch the next MIX instruction four memory reads must take place from consecutive locations in memory starting with the byte addressed by the Instruction Counter. The Memory Address Register (M,N) is loaded with the contents of the Instruction Counter and then a read can be performed, fetching the first of four bytes. Now the Memory Address Register (M,N) must be incremented. However M and N cannot be gated to the Arithmetic Logic Unit, but can only be selected as the destination for the output from the ALU. Instead the Instruction Counter must be incremented, this result can now be selected as the new value of the Memory Address Register (M,N), and the second byte can be read. The fetch routine is then more efficient if both the Instruction Counter and the Instruction register are in the same file, since the fetch routine alternately selects one then the other.

The user registers most frequently selected in MIX are the A register, the X register and the A-X register. The X register is the right hand extension of the A register in multiply, divide, and shift instructions. It is advantageous then to have the A register and the X register in the Primary file with Instruction register to facilitate the execution of A, X and A-X instructions.

The one free work register was placed in the Primary file since this is where the instruction to be executed would reside as well as the registers most likely to be involved with this instruction execution.

The A register was located in Primary file registers P1, P2, P3 and P4 where P1 contains the sign and most significant bits of A and P4 the least significant bits. The X register was assigned registers P5, P6, P7, P8 with P5 holding the most significant bits and P8 the least significant bits. These assignments result in X being the natural right hand extension of A, this of course makes microprogramming the shift, divide, and multiply routines straight forward if not easier. The instruction register was assigned registers P11, P12, P13, and P14. The free work register was located at P15. This helped group the work registers together, recall P14 contains the opcode, MIX byte 5, which is available to the microprogrammer following instruction decode. The Instruction Counter was located at P9 and P10, these being the only remaining registers in the Primary file.

The Index registers, the Jump register, the Overflow and Comparison Indicators occupy all of the Secondary file. The Index registers were grouped into 12 consecutive registers starting with Secondary file 1. Index register Y then resides in Secondary files 2Y-1 and 2Y. This allows microprograms which handle Index register operations to be generalized. Figure 2.15 is a microroutine used to zero the Index register specified (I1-I6) in the Instruction register (MIX byte 3).

Figure 2.15

```
          SPF                   Select Primary files
          LT      X'FO'         Load T register with mask
          OR*     13,T,(T)      Mask off index number
          CPY     15,T          Copy index number x 16 into P15
          SFL     15            Shift P15 left, divide by 2
          SFL     15            Shift P15 left, divide by 2
          SFL     15,(U)        Compute index number x 2,
                                  Put result in U register
          SSF                   Select Secondary file
          ZOF     0,5           Zero file (U), LSB of index
              The U register will be ORed into the upper
              16 bits of the microcommand when the S option
              is included
          SPF                   Select Primary file
          DEC     15,(U)        Compute (Index number x 2) -1
                                  put result in U register
          SSF                   Select Secondary file
          ZOF     O,S           Zero file (U), MSB of index
```

The Jump register was allocated file register S13 and S14. The function of the Jump register is to copy the current contents of the Instruction Counter immediately prior to a Jump Instruction. This provides a one level subroutine linkage for the MIX user. This copy must take place across the file boundary since the Instruction Counter is in the Primary file and the Jump register is in the Secondary file. However, this is only a two byte transfer and Jump instructions are executed less frequently than the fetch routine or even A, X, or A-X register instructions.

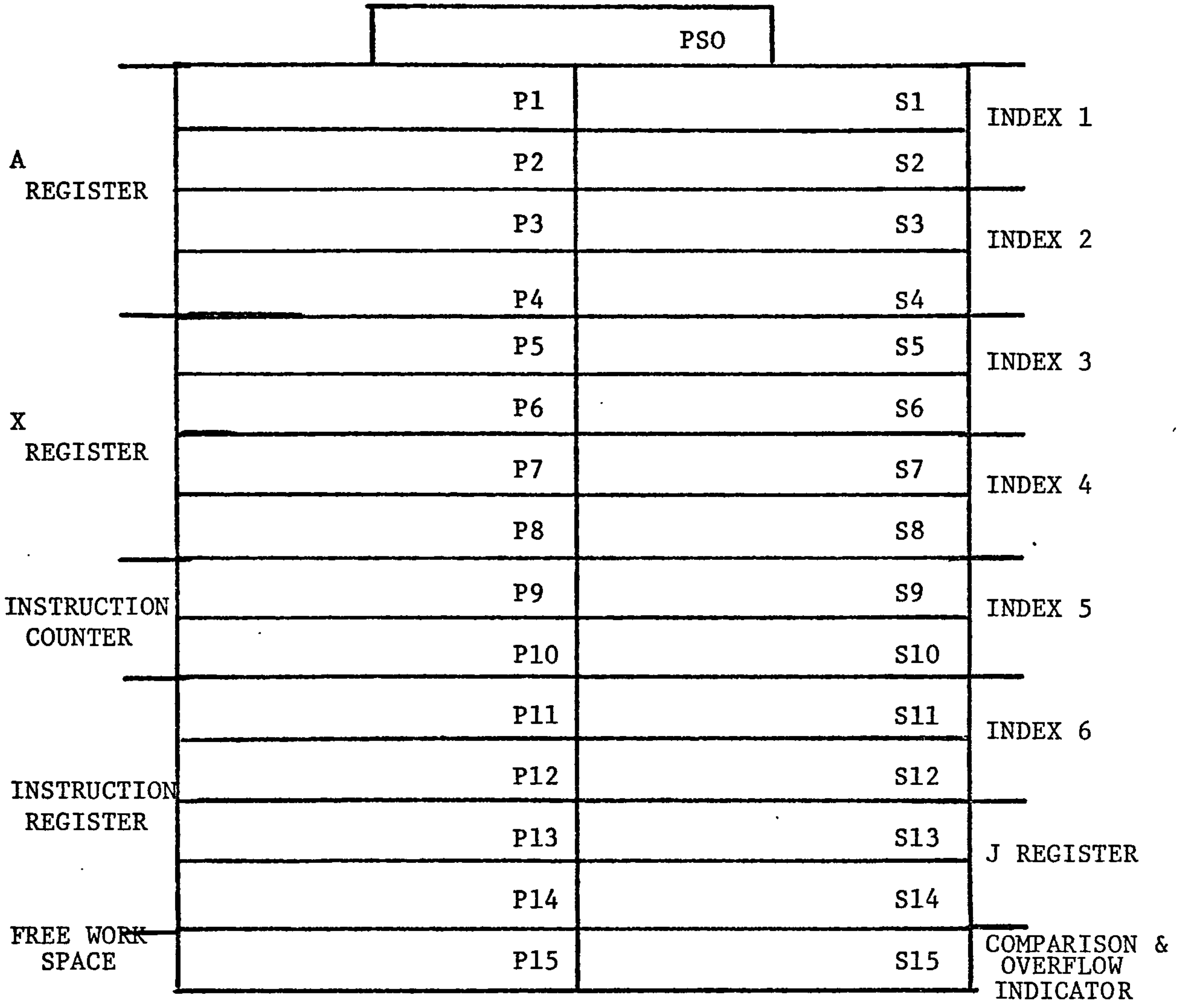
The file containing the overflow toggle and the Comparison Indicator was placed in the Secondary file 15. Note that all the other MIX registers are composed of an even number of file registers, but there are 15 file registers (odd) in each file. Thus the free work file register must be assigned to one file and the Overflow and Comparison Indicator register to the other. Considering the need for a work space in the Primary file, the Overflow and Comparison Indicator was placed in the Secondary file. Figure 2.16 illustrates the file allocation of the MIX registers as discussed.

Firmware Logic Problems:

Solutions to these major hardware allocation problems, memory allocation and register allocation, defined the relation between the host machine and the target machine so that microprogramming could begin. However, the machine organization that was developed resulted in two firmware problems.

The first problem, which had been anticipated, was the lack of sufficient work space to perform the required firmware routines. In the file allocation plan, an attempt was made to keep all MIX registers in either the Primary file or the Secondary file. This resulted in only one free file register to be used by the firmware for counters, temporary

Figure 2.16



S	0	1_1	1_2	1_3	1_4	1_5	1_6
2_1	2_2	2_3	2_4	2_5	2_6	3_1	3_2
3_3	3_4	3_5	3_6	4_1	4_2	4_3	4_4
4_5	4_6	5_1	5_2	5_3	5_4	5_5	5_6

A, X, AND INSTRUCTION REGISTER

S - Sign
0 - Zero
Ki i^{th} bit of K^{th} byte

S	0	4_1	4_2	4_3	4_4	4_5	4_6
5_1	5_2	5_3	5_4	5_5	5_6	0	0

INDEX, JUMP SAVE AND INSTRUCTION COUNTER

o	o	o	O	o	L	E	G

OVERFLOW AND COMPARISON

results and flags. One or two registers were freed after instruction decode occured but in some cases 3 free registers were not enough. For example, the Multiply instruction requires at least one more register than is available. This can only be solved by temporarily writing some portion of a MIX register, not currently being used, out to core memory.

The second firmware problem encountered was caused by the misalignment of MIX bytes and Microdata bytes. Since byte boundaries of the host machine did not correspond to byte boundaries on the target machine, programming the MIX partial field specifications was quite involved. The format used to pack six MIX bytes into four Microdata bytes resulted in each MIX byte being stored in a slightly different position than the other MIX bytes. From figure 2.17 it can be seen that the MIX sign byte and the first data byte occupy the first Microdata byte with one unused bit also present. MIX byte two and the high order two bits of MIX byte three are in Microdata byte 2. The low order four bits of MIX byte 3 and the high order 4 bits of MIX byte four are in Microdata byte 3. The low order 2 bits of MIX byte 4 and MIX byte five are in Microdata byte four. This format defies uniform handling of MIX bytes. As a result the microprogram routines which treated MIX partial word specifications were lengthy. A good example of this problem is the MIX Store A instruction. In this instruction the number of bytes specified by the F field is taken from the right hand side of A and these bytes replace the contents of the effective operand address specified by the F field. The bytes of the operand not mentioned by F and the A register are unchanged. Figure 2.17 illustrates all twenty-one variations of this instruction.

Thirty-four of the sixty-four MIX instructions were microprogrammed using the allocations discussed earlier. It became obvious, however, that the complete MIX emulator would exceed 2048 instructions, the size of the AROM. At this point the following compromises were considered.

1. Emulate a subset of the MIX instructions rather than the complete repertoire.
2. Page in sections of microcode from disk as they are required (7). This would increase MIX instruction execution time but would create a virtual AROM.
3. Reallocate MIX memory avoiding the packing of MIX bytes into Microdata bytes. This makes it possible to write simpler code but impossible to implement all 4000 words of MIX memory.

In the midst of this consideration an additional 16K of memory was acquired for the 1600/30 allowing the adoption of method 3 as well as the implementation of all of MIX memory. This attempt to emulate MIX was then terminated and a new study of hardware allocations was begun taking advantage of the additional memory and the mistakes that had been made during this first attempt.

Figure 2.17

$M_S N\ M_1 M_1 M_1 M_1 M_1 M_1$
$M_2 M_2 M_2 M_2 M_2 M_2 M_3 M_3$
$M_3 M_3 M_3 M_3 M_4 M_4 M_4 M_4$
$M_4 M_4 M_5 M_5 M_5 M_5 M_5 M_5$

Initial contents
in Memory
M_K-Some bit of the K^{th} byte
of M

$A_S N\ A_1 A_1 A_1 A_1 A_1 A_1$
$A_2 A_2 A_2 A_2 A_2 A_2 A_3 A_3$
$A_3 A_3 A_3 A_3 A_4 A_4 A_4 A_4$
$A_4 A_4 A_5 A_5 A_5 A_5 A_5 A_5$

Initial Contents
in A Register
A_K-Some bit of the K^{th} byte
of A

$A_S N\ M_1 M_1 M_1 M_1 M_1 M_1$
$M_2 M_2 M_2 M_2 M_2 M_2 M_3 M_3$
$M_3 M_3 M_3 M_3 M_4 M_4 M_4 M_4$
$M_4 M_4 M_5 M_5 M_5 M_5 M_5 M_5$

STA M,(0,0)

$A_S N\ A_5 A_5 A_5 A_5 A_5 A_5$
$M_2 M_2 M_2 M_2 M_2 M_2 M_3 M_3$
$M_3 M_3 M_3 M_3 M_4 M_4 M_4 M_4$
$M_4 M_4 M_5 M_5 M_5 M_5 M_5 M_5$

STA M,(0,1)

$M_S N\ A_5 A_5 A_5 A_5 A_5 A_5$
$M_2 M_2 M_2 M_2 M_2 M_2 M_3 M_3$
$M_3 M_3 M_3 M_3 M_4 M_4 M_4 M_4$
$M_4 M_4 M_5 M_5 M_5 M_5 M_5 M_5$

STA M,(1,1)

$A_S N\ A_4 A_4 A_4 A_4 A_4 A_4$
$A_5 A_5 A_5 A_5 A_5 A_5 M_3 M_3$
$M_3 M_3 M_3 M_3 M_4 M_4 M_4 M_4$
$M_4 M_4 M_5 M_5 M_5 M_5 M_5 M_5$

STA M,(0,2)

$M_S N\ A_4 A_4 A_4 A_4 A_4 A_4$
$A_5 A_5 A_5 A_5 A_5 A_5 M_3 M_3$
$M_3 M_3 M_3 M_3 M_4 M_4 M_4 M_4$
$M_4 M_4 M_5 M_5 M_5 M_5 M_5 M_5$

STA M,(1,2)

$M_S N\ M_1 M_1 M_1 M_1 M_1 M_1$
$A_5 A_5 A_5 A_5 A_5 A_5 M_3 M_3$
$M_3 M_3 M_3 M_3 M_4 M_4 M_4 M_4$
$M_4 M_4 M_5 M_5 M_5 M_5 M_5 M_5$

STA M,(2,2)

$A_SN\ A_3A_3A_3A_3A_3A_3$
$A_4A_4A_4A_4A_4A_4A_5A_5$
$A_5A_5A_5A_5M_4M_4M_4M_4$
$M_4M_4M_5M_5M_5M_5M_5M_5$

STA M,(0,3)

$M_SN\ A_3A_3A_3A_3A_3A_3$
$A_4A_4A_4A_4A_4A_4A_5A_5$
$A_5A_5A_5A_5M_4M_4M_4M_4$
$M_4M_4M_5M_5M_5M_5M_5M_5$

STA M,(1,3)

$M_SN\ M_1M_1M_1M_1M_1M_1$
$A_4A_4A_4A_4A_4A_4A_5A_5$
$A_5A_5A_5A_5M_4M_4M_4M_4$
$M_4M_4M_5M_5M_5M_5M_5M_5$

STA M,(2,3)

$M_SN\ M_1M_1M_1M_1M_1M_1$
$M_2M_2M_2M_2M_2M_2A_5A_5$
$A_5A_5A_5A_5M_4M_4M_4M_4$
$M_4M_4M_5M_5M_5M_5M_5M_5$

STA M,(3,3)

$A_SN\ A_2A_2A_2A_2A_2A_2$
$A_3A_3A_3A_3A_3A_3A_4A_4$
$A_4A_4A_4A_4A_5A_5A_5A_5$
$A_5A_5M_5M_5M_5M_5M_5M_5$

STA M(0,4)

$M_SN\ A_2A_2A_2A_2A_2A_2$
$A_3A_3A_3A_3A_3A_3A_4A_4$
$A_4A_4A_4A_4A_5A_5A_5A_5$
$A_5A_5M_5M_5M_5M_5M_5M_5$

STA M,(1,4)

$M_SN\ M_1M_1M_1M_1M_1M_1$
$A_3A_3A_3A_3A_3A_3A_4A_4$
$A_4A_4A_4A_4A_5A_5A_5A_5$
$A_5A_5M_5M_5M_5M_5M_5M_5$

STA M,(2,4)

$M_SN\ M_1M_1M_1M_1M_1M_1$
$M_2M_2M_2M_2M_2M_2A_4A_4$
$A_4A_4A_4A_4A_4A_4A_5A_5$
$A_5A_5M_5M_5M_5M_5M_5M_5$

STA M,(3,4)

$M_SN\ M_1M_1M_1M_1M_1M_1$
$M_2M_2M_2M_2M_2M_2M_3M_3$
$M_3M_3M_3M_3A_5A_5A_5A_5$
$A_5A_5M_5M_5M_5M_5M_5M_5$

STA M,(4,4)

$A_SN\ A_1A_1A_1A_1A_1A_1$
$A_2A_2A_2A_2A_2A_2A_3A_3$
$A_3A_3A_3A_3A_4A_4A_4A_4$
$A_4A_4A_5A_5A_5A_5A_5A_5$

STA M,(0,5)

$M_SN\ A_1A_1A_1A_1A_1A_1$
$A_2A_2A_2A_2A_2A_2A_3A_3$
$A_3A_3A_3A_3A_4A_4A_4A_4$
$A_4A_4A_5A_5A_5A_5A_5A_5$

STA M,(1,5)

$M_SN\ M_1M_1M_1M_1M_1M_1$
$A_2A_2A_2A_2A_2A_2A_3A_3$
$A_3A_3A_3A_3A_4A_4A_4A_4$
$A_4A_4A_5A_5A_5A_5A_5A_5$

STA M,(2,5)

$M_SN\ M_1M_1M_1M_1M_1M_1$
$M_2M_2M_2M_2M_2M_2A_3A_3$
$A_3A_3A_3A_3A_4A_4A_4A_4$
$A_4A_4A_5A_5A_5A_5A_5A_5$

STA M,(3,5)

$M_SN\ M_1M_1M_1M_1M_1M_1$
$M_2M_2M_2M_2M_2M_2M_3M_3$
$M_3M_3M_3M_3A_4A_4A_4A_4$
$A_4A_4A_5A_5A_5A_5A_5A_5$

STA M,(4,5)

$M_SN\ M_1M_1M_1M_1M_1M_1$
$M_2M_2M_2M_2M_2M_2M_3M_3$
$M_3M_3M_3M_3M_4M_4M_4M_4$
$M_4M_4A_5A_5A_5A_5A_5A_5$

STA M,(5,5)

III. THE SECOND ATTEMPT

In the fall of 1974, the Computer Science Department increased the Microdata's core memory to 32K bytes. At this point a second attempt was initiated to emulate the MIX 1009 computer using this additional memory. Again, the two major design problems concerned the allocation of Microdata hardware for the emulation of MIX hardware, and firmware logic problems.

Hardware Allocation Problems

The first design decision in this second attempt was again how one should emulate MIX's memory. The memory resources now available on the Microdata 1600/30 and the memory requirements of the MIX machine are reflected in Figure 3.1.

	Words of Memory	Bits/ Byte	Total # of Bytes	Total # of Bits	Character Code	Numeric Code
Microdata 1600/30	undefined byte addressable	8	32,728	261,824	ASCII or EBCDIC	binary 2's compliment
MIX 1009	4000	1/sign 6/data	4000 sign 20,000 data	124,000	Knuth's Code	binary sign plus magnitude

With the additional 16K of core memory the Microdata 1600/30 was larger than the MIX 1009. Recall that with 16K of core memory the Microdata had only 16,364 bytes to implement MIX's 24,000 byte memory. In the first attempt this dilemma was solved by packing each 31 bit MIX word into four Microdata 8 bit bytes. However, this design resulted in difficult firmware logic. But with 32K of core memory, packing was no longer necessary since the Microdata had more than enough bytes to implement MIX's memory byte for byte.

This surplus of main memory solved the major problems previously encountered in implementing MIX's memory, but three problems still remainded, namely:

1. How many Microdata bytes should be used to emulate each MIX word?

2. How many bits should be used in each byte?

3. How should any extra Microdata memory be used?

In examining the first of these problems, it appears that either five bytes per MIX word or six bytes per MIX word was the best solution in light of previous experience. The five bytes per MIX word solution required packing the sign and the first MIX data byte together. This packing would result in more available MIX memory but would inhibit uniform handling of all five data bytes. Uniform handling and the ability to generalize the firmware for partial word operations was not possible in the first attempt, it was a primary consideration however in the second attempt. Therefore, the six-bytes-per-MIX-word solution was selected where the sign byte and 5 data bytes would each be assigned to a separate Microdata byte, figure 3.2.

Figure 3.2

Sign Byte
1st Data Byte
2nd Data Byte
3rd Data Byte
4th Data Byte
5th Data Byte

Recall from Chapter II that the three disadvantages of the six byte solution were:

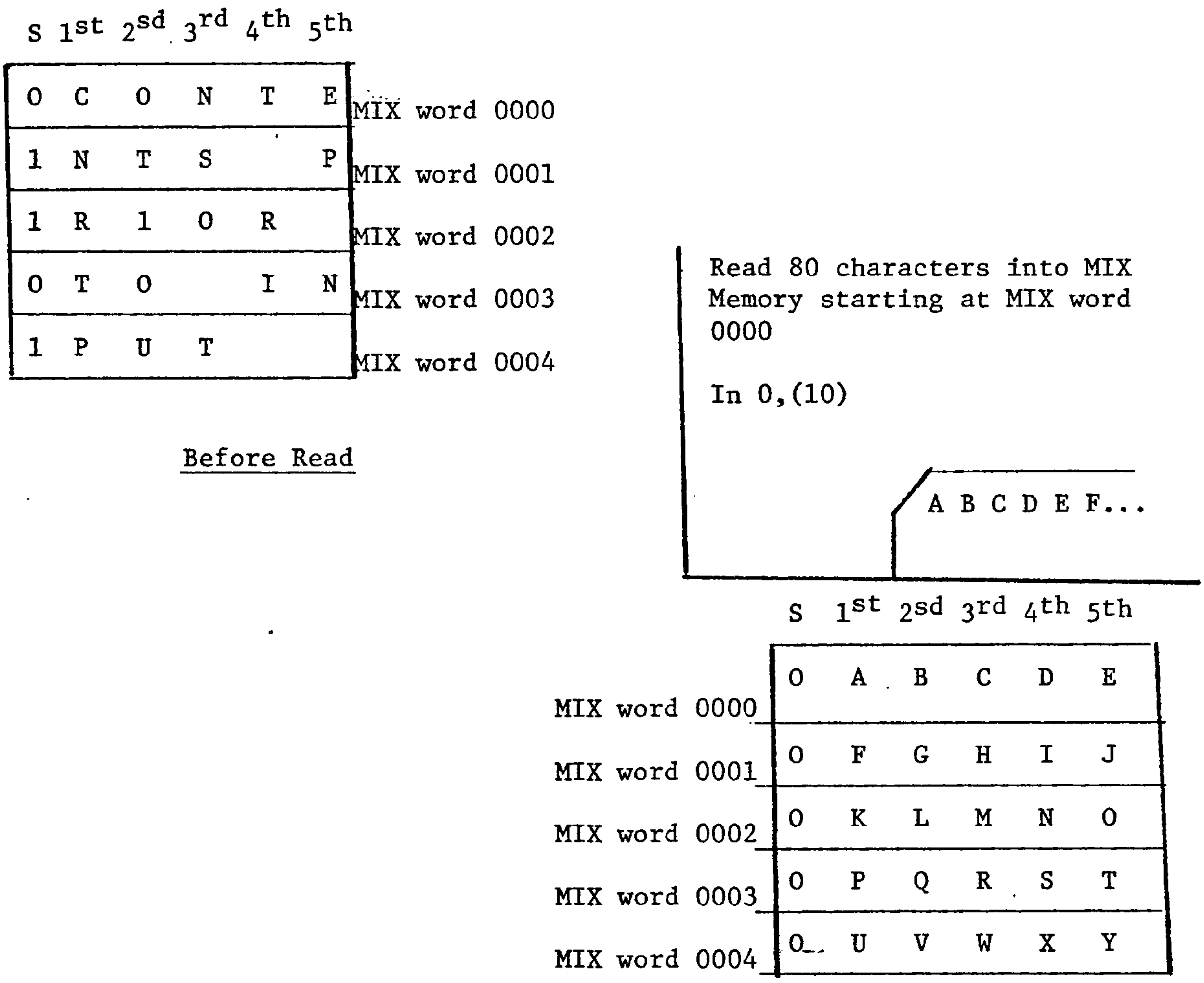
A) Not all 4000 words of MIX memory could be emulated.

B) Address translation from MIX address to Microdata address and vice versa was time consuming.

C) Detection of MIX word boundaries was difficult when given only the corresponding Microdata address.

The increase of main memory to 32K solved problem A, in fact 8,728 bytes of Microdata memory would be still available at six bytes per MIX word. Problem B can be solved, as discussed in Chapter II, however, address translation is still time consuming. Problem C however, is the most difficult. Recall that MIX Input/Output instructions handle the sign byte of each MIX word differently from the data bytes. On Input the sign bytes are set positive and on Output the sign bytes are ignored, figure 3.3 illustrate the Input operation.

Figure 3.3



S	1st	2sd	3rd	4th	5th	
0	C	O	N	T	E	MIX word 0000
1	N	T	S		P	MIX word 0001
1	R	1	O	R		MIX word 0002
0	T	O		I	N	MIX word 0003
1	P	U	T			MIX word 0004

Before Read

Read 80 characters into MIX Memory starting at MIX word 0000

In 0,(10)

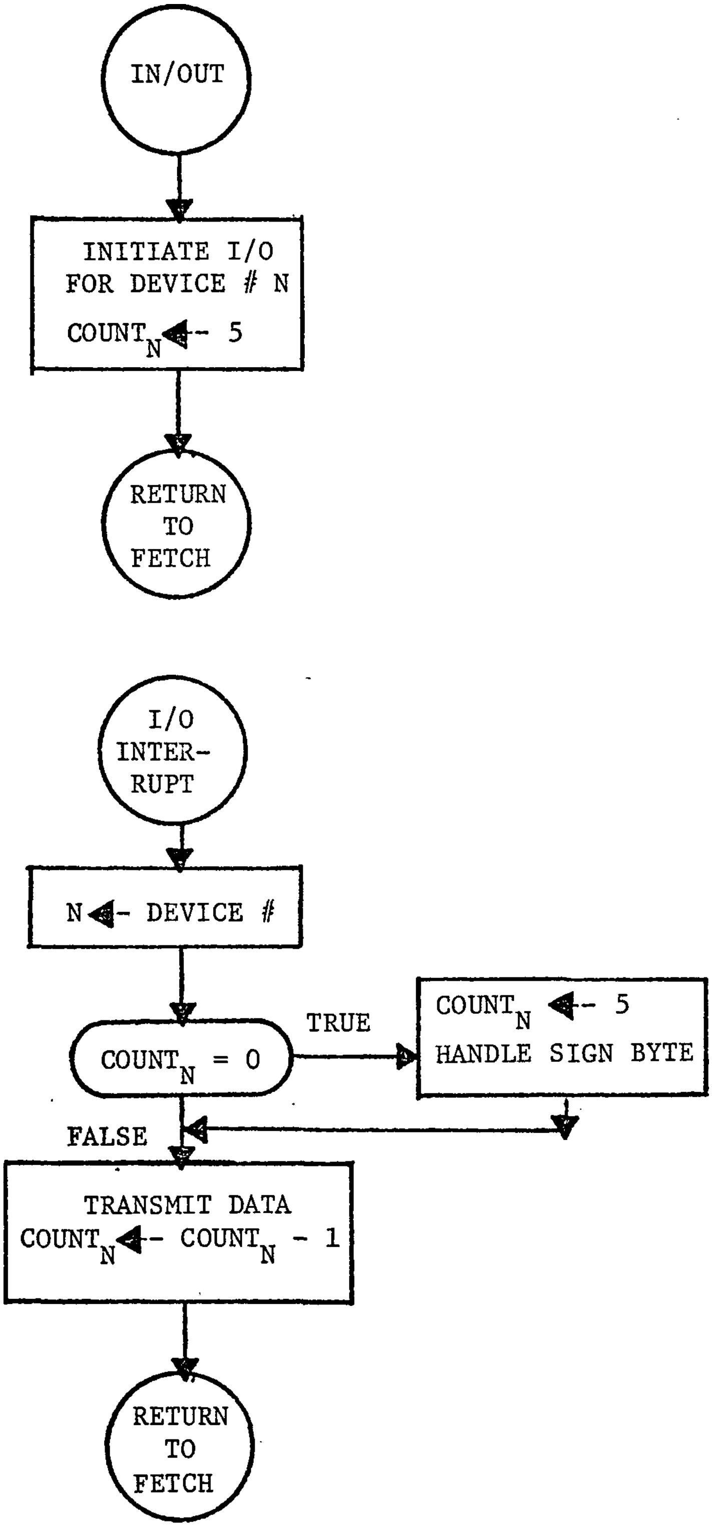
	S	1st	2sd	3rd	4th	5th
MIX word 0000	0	A	B	C	D	E
MIX word 0001	0	F	G	H	I	J
MIX word 0002	0	K	L	M	N	O
MIX word 0003	0	P	Q	R	S	T
MIX word 0004	0	U	V	W	X	Y

After Read

Input/Output on the host machine however occurs one byte at a time, and the microprogram must use Microdata addresses to store each data byte, thus the firmware must be able to sense MIX word boundaries. As mentioned earlier one way to do this is to divide each Microdata address by six. If the remainder is zero then this byte corresponds to a MIX sign byte and should be handled accordingly. Another possible solution is to assign to each I/O device a counter that is set to zero when an I/O operation is initiated on that device, then each time a data byte transmission occurs this counter is tested to see if it is equal to zero. If not the data transmission would take place and the counter would be decremented. But if the counter is zero the Microdata address corresponds to a MIX sign byte and this byte should be either zeroed or ignored, depending on whether the operation involved is input or output. The counter would then be set to 5 and normal handling of data could resume, figure 3.4 presents a flowchart for the above description. Either of these two methods, dividing by six or running a special counter would solve the problem of MIX word boundary detection but neither is easily accomplished.

Having tentatively adopted the six-byte-per-MIX-word solution, the next decision concerned how many bits should be used in each MIX byte. Knuth specifies each MIX word should hold at least 64 values, but a most 100 values. This range allows MIX to be implemented as either a binary or decimal machine. This implies that any binary implementation would have to use 6 bit data bytes. However, the Microdata is an 8 bit machine. The Arithmetic-Logic Unit of the Microdata accepts 8 bit operands and produces an 8 bit result plus an high order carry to be used as a Link bit in multiple byte operations. If MIX was to be emulated with a 6 bit byte

Figure 3.4


IN/OUT
INITIATE I/O FOR DEVICE # N
COUNT$_N$ ← 5
RETURN TO FETCH
I/O INTER-RUPT
N ← DEVICE #
COUNT$_N$ = 0
TRUE
COUNT$_N$ ← 5
HANDLE SIGN BYTE
FALSE
TRANSMIT DATA
COUNT$_N$ ← COUNT$_N$ - 1
RETURN TO FETCH

then the following format would result, figure 3.5.

Figure 3.5

S	N	N	N	N	N	N	N
N	N	1_1	1_2	1_3	1_4	1_5	1_6
N	N	2_1	2_2	2_3	2_4	2_5	2_6
N	N	3_1	3_2	3_3	3_4	3_5	3_6
N	N	4_1	4_2	4_3	4_4	4_5	4_6
N	N	5_1	5_2	5_3	5_4	5_5	5_6

S - Sign Bit
N - Not used
Ki- i^{th} bit of K^{th} byte

This format complicates all arithmetic instructions in MIX. If this format is allowed, the existing Microdata hardware for doing arithmetic operations cannot be used as intended. Incrementing a two byte counter, a very common and usually very simple operation is now fairly involved. The hardware Link bit provided in the Microdata ALU cannot be used to indicate a carry, so firmware logic must be developed to handle the high order carry. Figure 3.6 shows one way of incrementing two 6 bit bytes on the Microdata ALU.

Figure 3.6

```
*    Assume P1 contains the 6 high order bits of the
     counter
*    And P2 contains the 6 low order bits of the
     counter
*    Also assume P1 and P2 are carried in the following
     form
*            00111111     High order 2 bits = zero
*            00222222
             INC  2       Increment P2
             TN   2,X'40' Test for high order carry
             JP   RTN     No high order carry so continue
*    High order carry has occurred
             LT   X'3F'   Load mask into T register
             AND  2,T     Clear carry from P2
             INC  1       Increment P1
*    Now overflow is possible
             TZ   1,X'40' Test for overflow
             JP   OVERFL  Overflow has occurred
*                         Return
```

Using the Microdata's ALU as intended a two byte (8 bit) counter can be incremented as shown in figure 3.7.

Figure 3.7

```
*    Assume P1 and P2 again contain the counter
             INC  2       Increment low order 8 bits
             ADD  1,L,C   Add Link bit to high order 8
                            bits, set condition flags
             TZ   0,X'01" Test for overflow
             JP   OVERFL  Overflow has occurred
*                         Return
```

If the 6 bit format doubles the number of instructions required to increment a two byte counter, it should be clear that involved instructions such as Multiply, Divide, Add, Subtract, Shift, Char, and Num would be considerably longer as well. Recall also that MIX, being a sign plus magnitude machine already conflicts with the Microdata's ALU since it is a two's complement unit.

In light of these complications and the fact that the first attempt failed because the firmware became so involved and lengthly that it would not fit into 2K of AROM, an 8 bit data byte was adopted for the emulation of MIX. The format is shown in figure 3.8.

Figure 3.8

S	N	N	N	N	N	N	N
1_1	1_2	1_3	1_4	1_5	1_6	1_7	1_8
2_1	2_2	2_3	2_4	2_5	2_6	2_7	2_8
3_1	3_2	3_3	3_4	3_5	3_6	3_7	3_8
4_1	4_2	4_3	4_4	4_5	4_6	4_7	4_8
5_1	5_2	5_3	5_4	5_5	5_6	5_7	5_8

S - Sign Bit
N - Not used
Ki- i^{th} bit of K^{th} byte

This eight bit data byte called for the adoption of another character code. Knuth's code, a six bit code, could have been used, however, it was felt that since the teletype, line printer, and disk worked in ASCII, it would be more advantageous to use than forcing Knuth's code onto these devices via firmware.

The remaining memory allocation problem concerned what to do with the 8,728 bytes of surplus Microdata memory. The two choices are obvious:

A. Extend MIX memory by 1,454 words.

B. Provide some sort of system support (temporary storage).

Solution A enhances the MIX machine from the users point of view. However the extra memory is not necessary since MIX is suppose to have only 4000 words of memory. In fact, if programs are to be written according to Knuth's rule "that no more than sixty-four values are ever assumed for a byte" (1). The largest memory location addressable is location 4095 (2^6 -1). It would also seem that 4000 words of memory is more than enough for educational purposes. Thus adding more memory to the MIX machine offers no real advantage.

Solution B offers some advantages that are not obvious at first. All Input/Output in MIX takes place in concurrent mode. That is, an I/O operation is started via an In or Out instruction but a major portion of the I/O operation takes place while the user executes other instructions. Certain information must be available to the microprogram in order to carry out these block data transfers. However this information, memory address, and counters, must be stored somewhere besides the MIX registers available to the user.

This surplus memory provides an ideal, and in fact the only, place to temporarily store this type of information. A decision was also made in this second attempt, for reasons that will be discussed later, to keep MIX's Index registers in main memory instead of the secondary files. Thus with this type of privileged data in main memory it is necessary to have an area of core that the MIX user cannot use. If certain MIX memory locations were dedicated to the above functions then the MIX user could alter concurrent I/O operations as well as the contents of the Index registers. It was thought that this was both dangerous for beginning programmers and unnecessary.

Having adopted solution B one more question arose; where should MIX memory begin and where should the surplus memory reside? Three possible answers were considered, the two which follow are obvious:

1) The surplus resides at Microdata address 0000-8727 and MIX memory resides at 8728-35,728.

2) MIX memory resides at 0000-24,000 and the surplus at 24,001-35,728.

The third possibility, and the one which was chosen was to begin MIX memory at Microdata 0000 and then alternate one MIX word with two surplus bytes throughout Microdata memory. This did not effect the availability of the surplus memory but it did solve two problems previously discussed in this chapter. Figure 3.9 illustrates the above solution.

Figure 3.9

Microdata Address	MIX Address	
00000	00	sign byte
00001		1st data byte
00010		2sd data byte
00011		3rd data byte
00100		4th data byte
00101		5th data byte
00110		Surplus
00111		Surplus
01000	01	sign byte
01001		1st data byte
01010		2sd data byte
01011		3rd data byte
01100		4th data byte
01101		5th data byte
01110		Surplus
01111		Surplus
10000	02	sign byte
10001		1st data byte
10010		2sd data byte

Using this memory layout the addressing problems of the six byte per MIX word solution were solved. Each MIX word is six Microdata bytes long but now each MIX word begins on a Microdata address which is a multiple of eight. Thus conversion from MIX address to Microdata address can be accomplished by shifting the MIX address three places left. Conversion from Microdata address to MIX address involves shifting the Microdata address 3 places right. MIX word boundaries are also easy to sense. Any Microdata address ending in 000 is a word boundary. Surplus bytes are also easy to detect since their addresses all end in either 110 or 111.

In summary, the memory allocation problem was resolved by the following three policies:

1. One MIX word was to be composed of six Microdata bytes.

2. Each MIX byte was to contain 8 bits.

3. MIX memory would begin at Microdata address 0000 but each MIX word would be followed by two surplus data bytes.

It should be noted that these surplus data bytes are invisible to the MIX user. This inleaving of MIX words and surplus data bytes also allowed a minor extension of MIX memory from 4000 words to 4096 words. Some of these extra 96 words were dedicated for purposes not included in Knuth's specifications. For example, one word in high core is trapped to by the microprogram if an illegal address, opcode or I/O device is encountered. Two words are dedicated to each I/O device, one to store the device status byte upon completion of an I/O operation and one as a trap address in case of an I/O error on that device.

The second design decision concerned the mapping of MIX's registers into the hardware available on the Microdata 1600/30. The registers which were to be emulated along with their lenghts' are reflected in figure 3.10. Figure 3.10 appears on the following page.

As noted in chapter II the Microdata provides a 30 register work area for the microprogrammer to use in emulating the registers of target machines. In the first attempt all MIX registers were kept in these 30 file registers, however, this left only one free work register. This constraint

Figure 3.10

A Register	Sign plus 5 bytes
X Register	Sign plus 5 bytes
Instruction Register	Sign plus 5 bytes
Instruction Counter	2 bytes
Jump Save Register	2 bytes
Index Register 1	Sign plus 2 bytes
Index Register 2	Sign plus 2 bytes
Index Register 3	Sign plus 2 bytes
Index Register 4	Sign plus 2 bytes
Index Register 5	Sign plus 2 bytes
Index Register 6	Sign plus 2 bytes
Overflow and Comparison	1 byte

resulted in rather strained firmware logic and in some cases MIX registers had to be read out to main memory temporarily to provide the necessary work space. The idea of storing some of MIX's registers in main memory had been considered, this was avoided in the first attempt since intuitively it would slow the MIX machine. In the second attempt this approach was again considered. It was decided that if simpler firmware logic would result from certain registers being stored in main memory then the speed gained through this simpler and therefore faster logic would make up for the time spent paging registers in and out of main memory. One should also note that the sum of the registers listed in figure 3.10 is 41 file registers. Thus there were more MIX register bytes to emulate than there were file registers.

Initially it was decided to place the X register and the Index registers in main memory and to page them as required into the secondary file. Nine file registers (1-9) were reserved in the secondary file to hold the registers that were currently paged-in. A page map was to designate which registers were in memory and which were in the secondary files as well as which MIX register was in which set of Microdata files. Using this set-up either the X register and one Index register or up to 3 Index registers could be in the secondary files at any given time. The X register could fit into two possible slots either registers 1-6 or registers 3-9, and Index registers could fit into either registers 1-3, 4-6, or 7-9. This paging algorithm plus the other five MIX registers consumed a total of 29 registers leaving one register free.

Although this method did allow the emulation of all of MIX's registers and free work space could be created at almost any time by paging the registers in the secondary file out to memory, the overhead involved was considered very high. Instructions concerning the X registers were complicated since it would be in two positions. Index register routines were complicated, since they could be in any one of three places in the Secondary file. The accounting involved in keeping track of the current location of each MIX register file required three file registers and a considerable amount of AROM. Some sort of scheduling algorithm was also required to determine which register should be paged out in order to make room for the incoming register. It was thus decided that this strategy was too costly both in terms of firmware logic and file registers.

The paging concept was then amended to apply only to the Index registers, with only one Index register allowed in the file registers at any given time. The X register would reside permanently in the secondary files. This simplified the X register instructions considerably, as will be discussed later. The Index register instructions were also simplified since now there were only six different

Index registers instead of thirty-six. For example, there are six load Index instruction, one for each Index register. But all six are effectively represented by one Load Index routine since all the Index registers are loaded into the same place in the secondary file. This routine calls the paging routine to page in the required Index register and then loads this register with the proper contents. The same is true for the Load Index negative, Store Index, Jump On Index, Enter Index and Compare Index instructions. The accounting problem associated with the paging system was also simplified. The page map was now 3 bits long; these bits contained the number of the Index register currently rolled in from memory or the value zero if all the registers were currently rolled out. This ability to page all the Index registers out to memory provided an easy way to create free work files when the need arose. By allowing only one Index register in the Secondary files at any time, the need for a scheduling algorithm was eliminated. If Index register 1 is in the Secondary files and Index register 2 is required, either for indexing or by an Index instruction, Index register 1 must be paged out and then Index register 2 paged in.

The detailed flowchart of the paging mechanism, called the Index Register Supervisor can be seen in Chapter V and the microcode for the routine is found in Chapter VI.

Having solved the problem of too-many-MIX-registers-and-not-enough-files, the allocation of MIX registers to Microdata files was begun. The A register, the Instruction register and the Instruction Counter were assigned to the Primary file while the X register, Overflow and Comparison Indicators, Index registers, Index Map and Jump register were assigned to the Secondary file.

The Instruction Counter and Instruction register were again placed together to facilitate the fetch routine. The A register was placed in the same file with the Instruction register for the same reasons discussed in Chapter II, the main one being that the A register is the register most likely to be involved in the next instruction fetched. Although only one free work file remained in the Primary file, the three files in the Instruction Counter containing the Operation Code, the Index register and the sign of the Memory Address normally become available following the execution of the Instruction Decode Routine. These four free work areas in the Primary file are available in most cases.

The remaining MIX registers, the X register, the Home position for the Index registers, the Index Map, the Overflow and Comparison Indicators and the Jump Save register were grouped together out of necessity since the secondary file was the only place left to put them.

Figure 3.11 illustrates the file mapping that was finally selected. The A register was allocated Primary

Figure 3.11

E T $\overset{I}{0}$ I $\overset{I}{0}$ Z N O PSO

A Register	sign	P1	sign	S1	X Register
	1st byte	P2	1st byte	S2	
	2sd byte	P3	2sd byte	S3	
	3rd byte	P4	3rd byte	S4	
	4th byte	P5	4th byte	S5	
	5th byte	P6	5th byte	S6	
Free work Register		P7	OLEG x iii	S7	Overflow, Comp., & Index Map
Instruction Register	sign	P8	sign	S8	Index Register i
	A1 address-ho	P9	1st byte	S9	
	A2 address-lo	P10	2sd byte	S10	
	1 index spec.	P11		S11	Free work Registers
	F field spec.	P12		S12	
	C op code	P13		S13	
Instruction Counter	1st byte-ho	P14	1st byte-ho	S14	Jump Register
	2sd byte-lo	P15	2sd byte-lo	S15	

files P1, P2, P3, P4, P5, and P6. The X register was allocated the matching registers in the Secondary file. This allignment simplified the different A and X instructions in much the same way that paging simplified the Index instructions. The only difference between a Load A and a Load X instruction is the perodic selection of the Secondary files instead of the Primary files.

The Instruction register was placed in P8-P13 with P7 being the one free work file. This placed P7 next to the sign byte of the instruction address, which is one of the first files in the Instruction register to become free. The Instruction Counter was then assigned to P14 and P15, the remaining Primary files.

The Overflow and Comparison Indicators, a 4 bit register, and the Index Map, a 3 bit register, were combined into Secondary file S7. The Home position for the Index registers was assigned S8, S9, and S10. The Jump Save register was alligned with the Instruction Counter in S14 and S15. This left S11, S12, and S13 as free work registers.

It should be noted that eleven file registers can be freed after instruction decode if they are needed. P7, S11, S12, and S13 are always free. P8, P11, and P13 are free after instruction decode. S8, S9, S10 can be freed by paging the current Index register out to memory after the effective operand address has been computed. Finally, the seven bits of S7 can be packed into S1 with the sign of the X register if necessary, freeing S7.

The mapping allowed simplified coding of Index instructions and of A and X instructions as well as ample work space and result in the successful emulation of the MIX 1009 computer.

Firmware Logic Design:

Having defined the MIX Machine in terms of the Microdata's hardware, firmware design could begin. First a general overview of the system was composed. The following is an explanation of the overview as presented in figure 3.12.

Start Routine:

- performed following cold start and prior to the execution of any MIX instructions;

- enables external interrupts;
- enables the real time clock;
- initializes the teletype;
- loads the Instruction Counter from a dedicated high core address.

Fetch Routine:

- fetches the next instruction into the Instruction register from the address contained in the Instruction Counter.

Addressing Routine:

- computes the effective operand address

Decode Routine:

- examines the Instruction Operation Code and transfers control to the corresponding firmware instruction module;

Instruction Modules:

- firmware routines that execute the individual MIX instructions;

Interrupt Handler:

- executed after the execution of the last instruction and prior to fetching the next instruction;
- acknowledges and handles external interrupts from I/O devices;
- acknowledges and handles internal interrupts from the real time clock and console pannel.

Subroutine Packet:

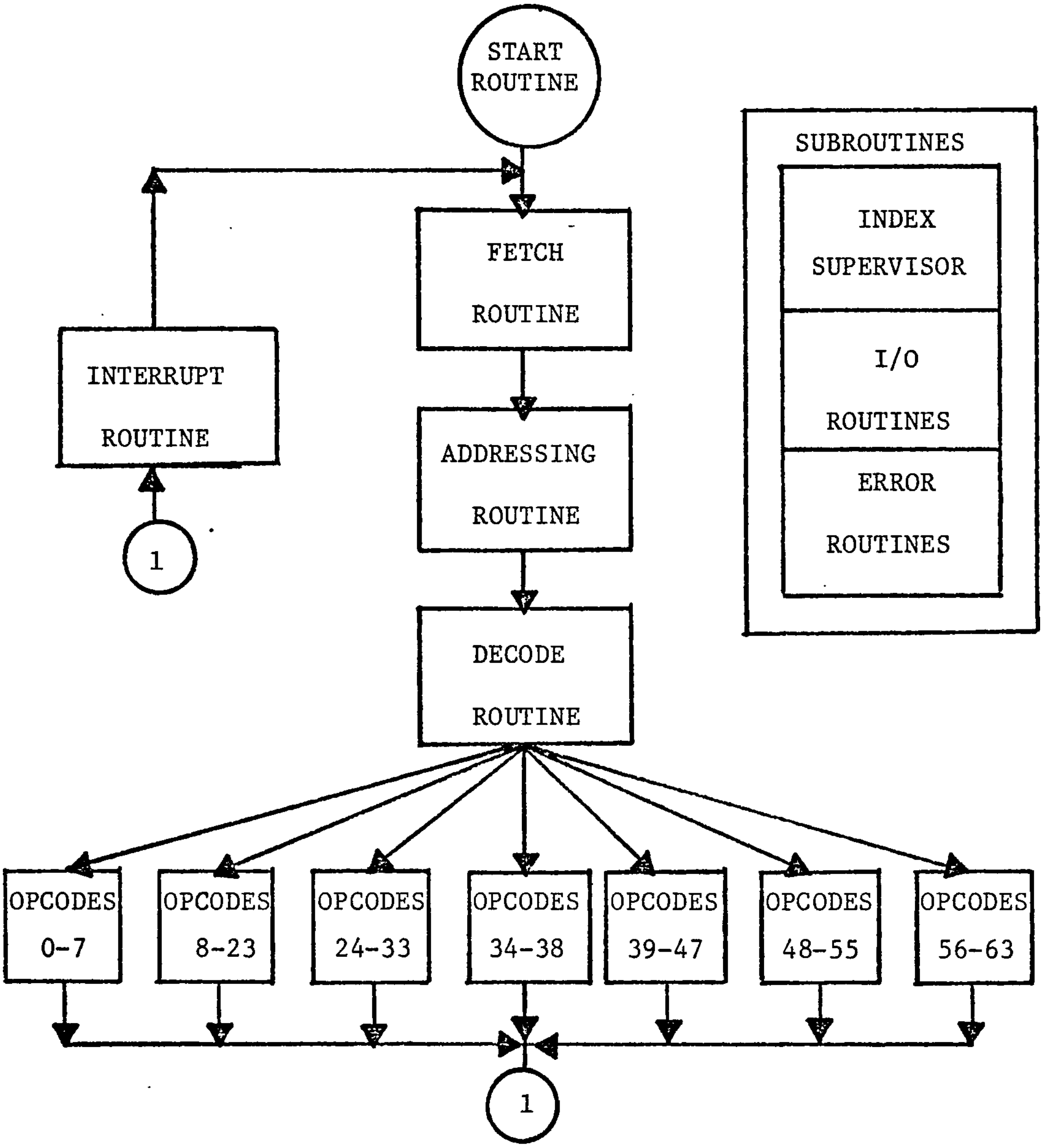
- Index register Supervisor: handles the paging of MIX index registers.

- I/O Routines: used by the Input/Output instructions as well as the interrupt handler.

- Error Routines: handles user errors such as illegal addresses, illegal opcodes, illegal I/O device numbers, and I/O errors.

An attempt was made to keep the MIX emulator as modular as possible. This facilitated program development and debugging as well as simplifying the decode routine. The decode routine divides the MIX instruction set into seven groups. Each group representing a type of MIX instruction.

Figure 3.12 SYSTEM OVERVIEW



The seven groups are:

1. Opcodes 0-7 Arithmetic-Logic instructions
2. Opcodes 8-23 Load instructions
3. Opcodes 24-33 Store instructions
4. Opcodes 34-38 Input/Output instructions
5. Opcodes 39-47 Jump instructions
6. Opcodes 48-55 Enter and Increment instructions
7. Opcodes 56-63 Compare Instructions

The remainder of this Chapter has been devoted to a discussion of the major logic problems and their solutions encountered in microprogramming each of these modules.

Opcode 0-7 Arithmetic-Logic Instructions

The Arithmetic Logic instructions are composed of ten instructions. Four of these instructions are relatively straight-forward and nothing will be said here concerning these instructions. If more information is required about these routines, consult the corresponding flowcharts in Chapter V and microcode in Chapter VI. These four instructions are NOP, HLT, SHIFT, and MOVE.

The remaing six instructions are the various arithmetic operations, ADD, SUB, MUL, DIV, CHAR, and NUM. The first problem encountered in microprogramming these routines was that of representing sign plus magnitude arithmetic on a two's complement machine. This problem was also found when coding the increment immediate and decrement immediate portions of opcodes 48-55 and in the Compare instructions, opcodes 56-63. The major difficulty involved adding or subtracting two sign plus magnitude numbers, since no hardware mechanism was available to;

1) determine the sign of the result;
2) determine if the result was in true form or two's complement form;
3) determine if overflow had occurred.

The Microdata's ALU does, of course, provide this type of hardware support, but only for two's complement arithmetic. Therefore, the existing negative result indicator and overflow indicator could not be used. The Microdata's negative result indicator is turned on when the high order bit of the result is a one, this high order bit being the sign bit in two's complement. However, in the case of MIX's sign plus magnitude format the presence of a one bit in the high order position indicates the presence of a large magnitude and says nothing about the sign of the mumber involved. The Microdata's overflow indicator is turned on "when the carry out of the high bit of the adder differs from the carry into it" (3). But for sign plus magnitude operations overflow occurs when the signs of the two operands are the same and the high order carry out of the address (i.e. the contents of the Link register) is a one.

When performing sign plus magnitude addition four cases arise, namely:

Case 1. (+A) + (+B);

Case 2. (-A) + (-B);

Case 3. (+A) + (-B);

Case 4. (-A) + (+B).

The following two rules were employed to perform sign plus magnitude addition using the Microdata's ALU.

Rule 1: If the signs of the operands are the same, case 1 and 2, then add the magnitudes together, giving the result the sign of A. Overflow has occurred if the Microdata's Link register contains a one.

Rule 2: If the signs of the two operands are different, cases 3 and 4, then the two's complement of B is formed, the addition occurs, and the Link register is examined. If the Link register contains a one the result is in true form and the sign of the result is the sign of A. But if the Link register contains a zero, then the magnitude of the result must be two's complement and the sign is the complement of the sign of A, (or simply the sign of B).

This algorithm for sign plus magnitude addition also works for sign plus magnitude subtraction with one modification, namely the sign of B must first be complemented then the addition algorithm can be employed, figure 3.13.

Figure 3.13

(+A) - (-B) = (+A) + (+B)   Case 1

(-A) - (+B) = (-A) + (-B)   Case 2

(+A) - (+B) = (+A) + (-B)   Case 3

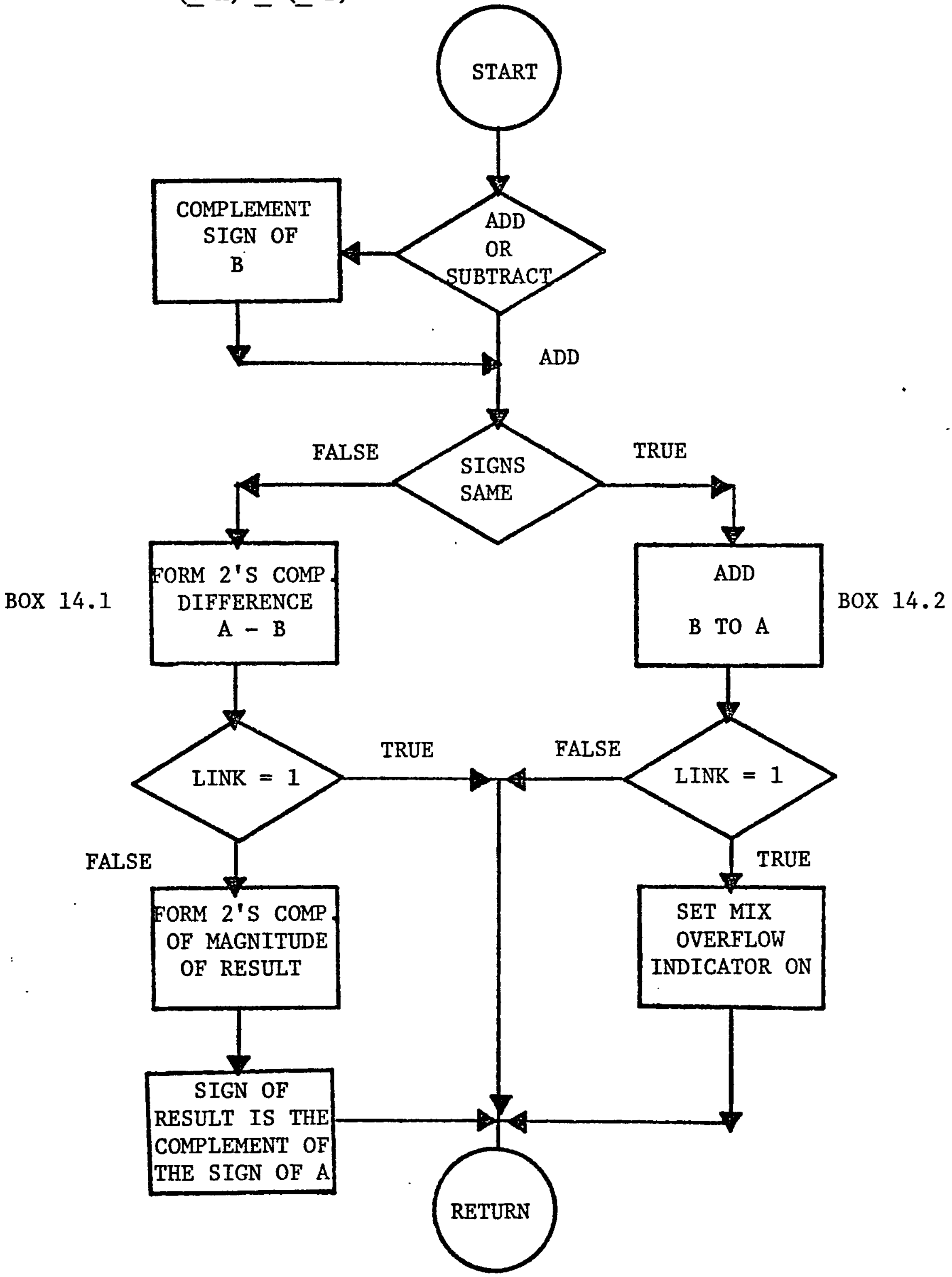
(-A) - (-B) = (-A) + (+B)   Case 4

The flowchart of the algorithm for two's complement addition and subtraction on the Microdata 1600/30 appears on the following page in figure 3.14.

In the actual microprogramming of this algorithm one major inefficiency was discovered. In figure 3.14, Box 14.1 and Box 14.2 represent the bulk of the required microcode. Since they involve multiple byte addition or subtraction operations they are quite long. The other unlabeled flowchart symbol's are represented in microcode by two or three instructions. However, the subtract section, Box 14.1, is identical to the add section, Box 14.2 with the exception that all add instructions (opcode 8) are replaced by two's complement subtract instructions (opcode 9). To avoid this repetition of code figure 3.14 was modified to take advantage of the Microdata's U register. The microprogrammer can select the U register, an eight bit register, to be ORed into the high order eight bits of the next instruction. For example, the instruction in figure 3.15 adds the contents of the T register to Primary file 6 if the U register contains

Figure 3.14

(+̲ A) +̲ (+̲ B)


START
COMPLEMENT SIGN OF B
ADD OR SUBTRACT
ADD
FALSE
SIGNS SAME
TRUE
BOX 14.1
FORM 2'S COMP. DIFFERENCE A - B
ADD B TO A
BOX 14.2
LINK = 1
TRUE
FALSE
LINK = 1
FALSE
TRUE
FORM 2'S COMP. OF MAGNITUDE OF RESULT
SET MIX OVERFLOW INDICATOR ON
SIGN OF RESULT IS THE COMPLEMENT OF THE SIGN OF A
RETURN

X'00'. However, it forms the two's complement difference of Primary file 6 and the T register (P6←P6-T) if the U register contains X'10'.

Figure 3.15

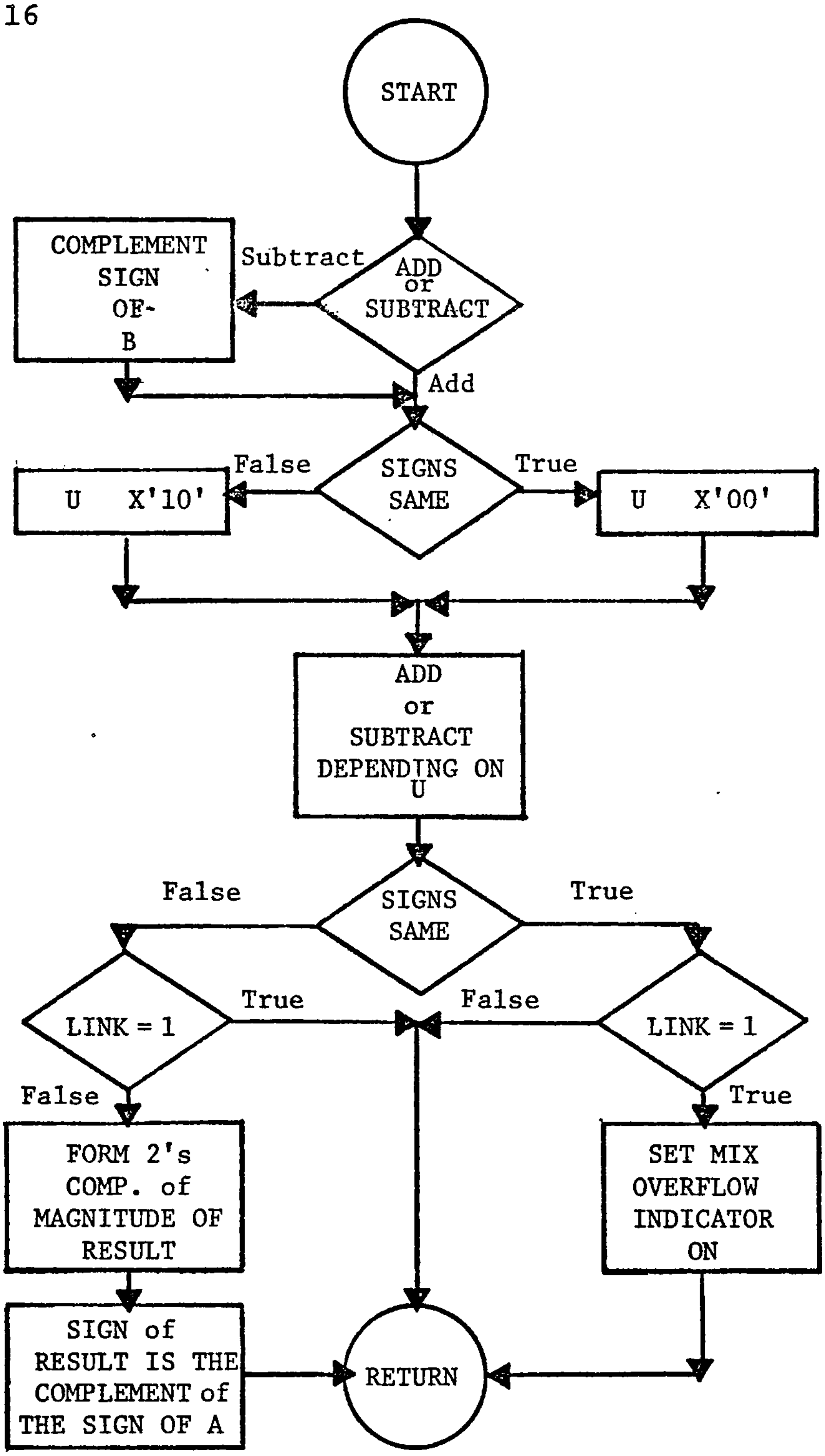
```
                ADD     6,T,(S) or U with X'86' and execute
     *     ADD ⟹opcode 8 = X'8' V X'0'
     *     SUBTRACT ⟹opcode 9 = X'8' V X'1'
```

Figure 3.16 gives the revised general logic flow used in performing MIX sign plus magnitude addition and subtraction.

The next problem encountered in microprogramming the Arithmetic operators was that of performing two's complement multiplication and division. The Microdata's ALU does not have a multiply or divide function available. However, multiplication and division are actually easier in sign plus magnitude format than in two's complement format. After the mechanism for testing overflow and negative results were developed for the add and subtract algorithm. The multiplication routine used a typical shift and add algorithm while the division employed one of the non-restoring techniques. These algorithms are common and are not discussed in this chapter. Detailed flowcharts and the corresponding microprograms can be found in Chapters V and VI.

The remaining two Arithmetic Operators NUM and CHAR proved to be the most difficult of the Arithmetic-Logic instructions. The MIX instruction NUM takes the ten digit

Figure 3.16


START
ADD or SUBTRACT
Subtract
COMPLEMENT SIGN OF B
Add
SIGNS SAME
False
True
U X'10'
U X'00'
ADD or SUBTRACT DEPENDING ON U
SIGNS SAME
False
True
LINK = 1
True
False
LINK = 1
False
True
FORM 2's COMP. of MAGNITUDE OF RESULT
SET MIX OVERFLOW INDICATOR ON
SIGN of RESULT IS THE COMPLEMENT of THE SIGN OF A
RETURN

decimal number is ASCII code loaded in the A and X registers and converts them into the corresponding binary number, placing the result in the A register. It is assumed that these ten characters are numeric and not alpha-numeric. CHAR provides the opposite function. It takes the 40 bit binary number in the A register and converts it into the equivalent decimal number, encoded in ASCII.

The NUM routine works by stripping the four high order zone bits off of each ASCII character. This leaves a ten digit Binary Coded Decimal number which was then expanded according to figure 3.17.

Figure 3.17

a_0 a_1 a_2 a_3 a_4	a_5 a_6 a_7 a_8 a_9
A register	B register

$$(((((((((a_0*10+a_1)*10+a_2)*10+a_3)*10+a_4)*10+a_5)*10+a_6)*10+a_7)*10+a_8)*10+a_9)$$

This part of the expansion was done with a 3 byte multiply.

This part of the expansion was done by a 5 byte multiply.

As noted above two special multiply routines were written to perform this expansion. One multiplied a three byte operand by 10 and the other multiplied a five byte operand by 10. This was done to speed up the execution of this instruction. Figure 3.18 points out that multiplying a number X by 10 is straight forward.

Figure 3.18

$$X * 10 = (X * 5) * 2 = (X * 2^2 + X) * 2$$

The CHAR routine is essentially a reversal of the NUM algorithm operand, initially the 40 bit contents of the A register is divided by 10. The remainder after the division was ORed with hexidecimal X'BO' to produce the corresponding ASCII character. This division occurred ten times constructing the ten digit decimal number from right to left.

Opcodes 8-23 Load instructions and
Opcodes 24-33 Store instructions:

The microprogramming of the Load instructions and the Store instructions was greatly facilitated by the way MIX hardware was emulated on the Microdata. No logic problems were encountered in programming these routines. However, it is interesting to note the difference in AROM utilization on these routines between this attempt and the first attempt. The Load routine for the first attempt, which handled all 16 Load commands took a total of 166 microinstructions. However, by reorganizing memory and file allocations the Load routine in the second attempt took only 86 instructions. In fact the number of microinstructions required to code the Load routine and the Store routine, 26 MIX instructions, totaled only 136 words of AROM. Details of these routines can be found in Chapters V and VI.

Opcodes 39-63 Jump Instructions Enter and Increment Instructions and Compare Instructions

The only major problem encountered with these instructions was the problem previously discussed concerning sign plus magnitude addition and subtraction. After this problem was solved these routines proved trival. Chapters V and VI contain details in these instructions.

Opcodes 34-38 Input/Output Instructions and the Interrupt Handler

The MIX I/O routines are by far the most involved in the MIX emulator. MIX is designed so that the user need not worry about details of Input/Output. All MIX Input/Output occurs in concurrent mode; the user initiates the operation and then is free to perform other work. At some later time the user checks if the operation has completed via a Jump Busy (J-Bus) or a Jump Ready (J-Red) instruction. Figure 3.19 gives the complete table of MIX Input/Output equipment, all of which is optional. (1).

Figure 3.19

Unit number	Peripheral Device	Record Size
t	tape unit no. t ($0 \leq t \leq 7$)	100 words
d	disk or drum unit no. d ($8 \leq d \leq 15$)	100 words
16	Card Reader	16 words
17	Card Punch	16 words
18	Printer	24 words
19	typewriter and paper tape	14 words

Each device has an associated fixed length record size. Transfers to and from the magnetic units involve full MIX words, sign and five bytes. Input or Output to the other devices is by character code, thus on Input signs are set to zero, and on Output signs are ignored.

The format of the Input/Output instructions is a little different from the rest of the MIX instruction repitoire. The opcode is, of course, used to indicate which I/O instruction the user wishes to execute. The F-field is used to denote which device is to be activated, and the memory address, which may be indexed, points to the first word of the buffer area to be used. The length of this buffer area is determined by the device chosen, (F-field) and by Figure 3.19.

Emulation of MIX I/O instructions was difficult because all the work associated with Input/Output must be performed by the microprogram. The MIX user provides the emulator with three parameters; (1) the direction of transfer, (2) the device number, and (3) the buffer address. After this presentation of parameters the microprogram is responsible for the remainder of the operation. The problem of writing these device handlers was complicated by the way the Microdata 1600/30 devices work. Of the MIX devices described in figure 3.19 the following were implemented.

Figure 3.20

Unit Number	Peripheral Device	Record Size
14	Disk Drive	32 words
15	Disk Drive	32 words
16	Card Reader	16 words
18	Printer	24 words
19	typewriter and paper tape	14 words

Due to differences in the peripheral controllers, these four devices employ three different types of Input/Output. The card reader and printer work in concurrent interrupt mode. In this mode the device controller generates a concurrent I/O request (interrupt) each time it is ready to perform a data transfer. Each data transfer involves sending or receiving a single byte of information. The typewriter and paper tape (teletype) work in byte mode. This mode of Input/ Output is the simplest of all I/O schemes. No interrupts are available in byte mode operations. The microprogram must repeatedly sample the teletype controller status byte and test if the controller is ready to transfer a byte of data. When the controller is finally ready, a single byte of data can be transmitted either to or from the teletype. The disks employ a mode resembling the I/O described for MIX. A Direct Memory Access (DMA) port is used by the disk controller to handle disk I/O. This port provides a direct path between main memory and the specified disk drive.

Therefore, no microprogram intervention is required once the operation has been initiated. The microprogram starts a disk operation by sending four parameters to the disk controller as follows:

(1) Device Address;
(2) Section Address;
(3) Starting Memory Buffer Address;
(4) Ending Memory Buffer Address.

Once this information is received the transfer takes place automatically.

The remainder of the Chapter is divided into three sections. Section one presents the card reader and printer handlers. Section two explains the teletype handler and section three deals with the disk handler.

Card Reader and Printer

Three major problems were encountered in writing the I/O handlers for the card reader and printer. The first problem was to develope a scheme to perform block transfers. This can be accomplished by setting a counter equal to the number of bytes to be transfered and saving the address of the buffer area. Then when a concurrent request is recognized the microprogram adjusts the counter and the memory address and performs the transfer. However, if the I/O is to be concurrent, the user must be allowed to execute MIX instructions between data transfers. This implies that the concurrent counter and buffer address cannot be saved in the file registers, since some MIX instructions use all of the files in the course of their execution. Thus these concurrent I/O values were stored in dedicated surplus memory bytes. As a result these counters are invisible to the MIX user.

As shown in figure 3.21, the execution of an In command to the card reader or an Out command to the printer occurs in three steps. First the status of the unit involved is polled to see if the controller is currently performing an I/O operation. If the unit is busy then the microprogram loops through the interrupt subroutine until the unit is ready. At this point a command is issued to the device controller to arm the concurrent interrupts and to begin an I/O operation. The concurrent count is then assigned its proper value, either 80 for the card reader or 120 for the printer, and this value along with the instruction memory address, files P9 and P10 are read out to the corresponding dedicated memory locations.

Recall from figure 3.12 that the interrupt routine is executed immediately prior to fetching the next MIX instruction. It is this routine that actually performs the data transfers once a concurrent operation has begun on the card reader or printer. Figure 3.22 illustrates the handling of concurrent interrupts by the interrupt subroutine. If a concurrent request has occurred, the request is acknowledged

Figure 3.21

Concurrent I/O Initiation

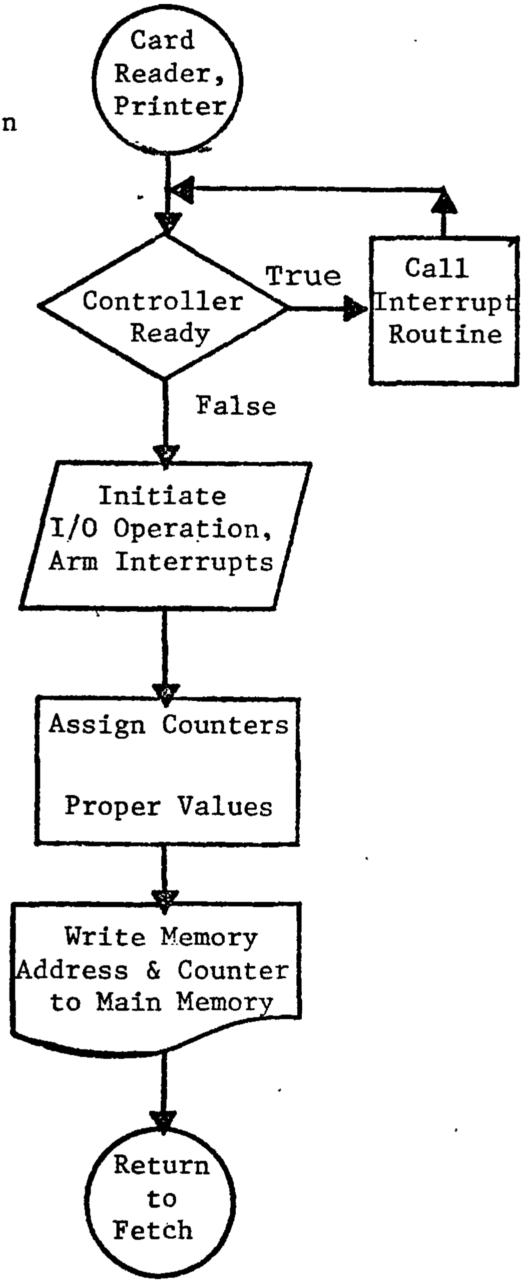
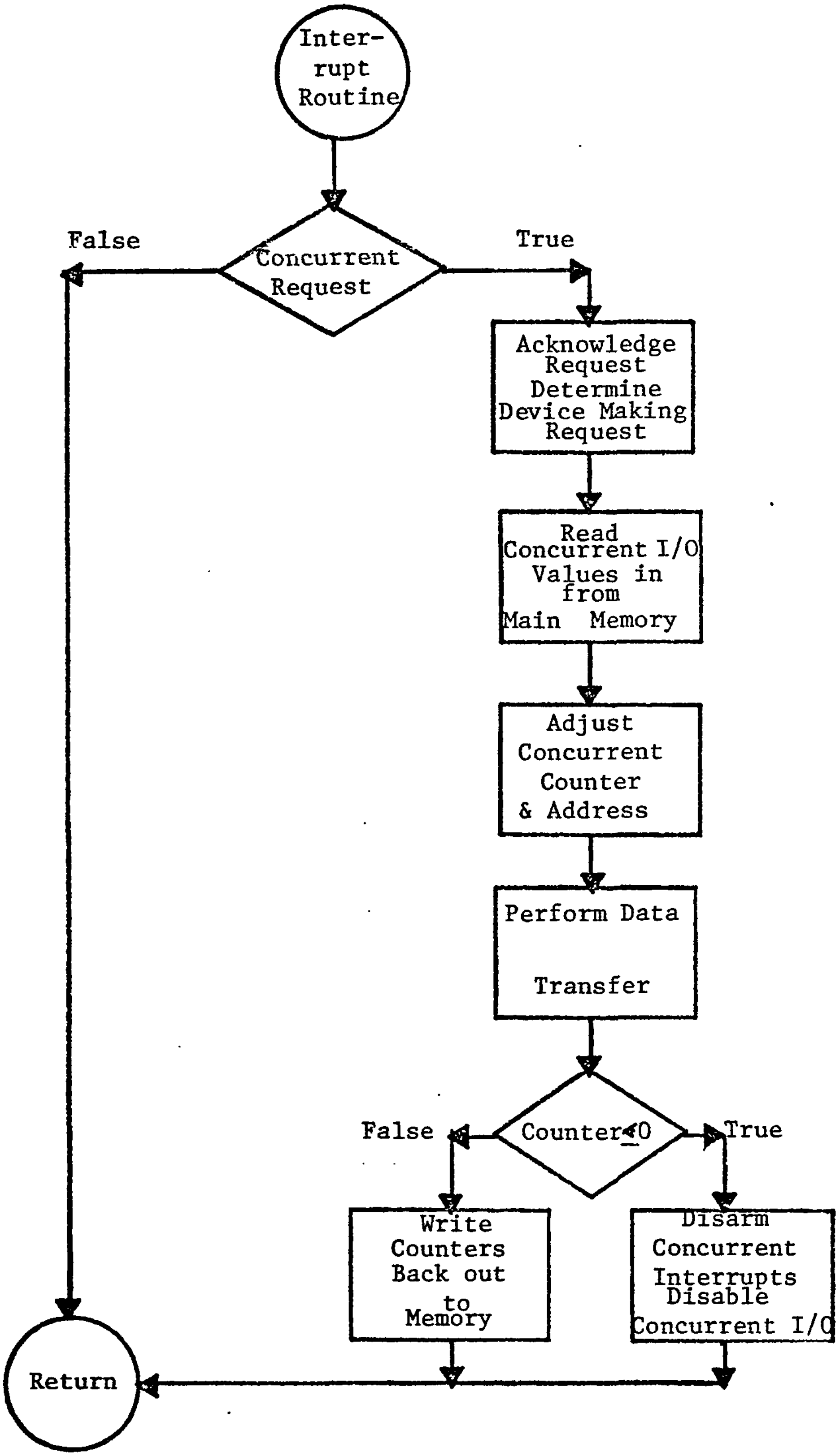


Figure 3.22


Inter-rupt Routine
Concurrent Request
False
True
Acknowledge Request Determine Device Making Request
Read Concurrent I/O Values in from Main Memory
Adjust Concurrent Counter & Address
Perform Data Transfer
Counter≤0
False
True
Write Counters Back out to Memory
Disarm Concurrent Interrupts Disable Concurrent I/O
Return

and the requesting device responds by supplying its device address. The interrupt routine examines this device address to determine which device is requesting service. Once this has been determined the devices concurrent values are fetched from main memory. These counters are adjusted and the specified transfer occurs. The concurrent counter is then tested if it is still positive the counters are written back out to main memory. However, if the counter is zero or negative the interrupting device is disabled and interrupts for this device are disarmed.

The second problem in designing the concurrent I/O routines concerned the card reader's character code. The card reader uses the EBCDIC character code while the other devices use ASCII. Therefore, translation from EBCDID to ASCII must occur before a card image can, for example, be listed on the printer. This conversion involves a simple table look up. The high order two bits of the incomming EBCDIC character are masked off and the low order bits are then used as an index into the ASCII table to retrieve the corresponding character. Since MIX provides no logical operators this masking operation is rather involved if the MIX user must perform the conversion. Therefore, the EBCDIC to ASCII conversion was performed in the microprogram. This was rather expensive in terms of AROM utilization. There are 64 characters in the ASCII code. However 128 AROM locations were required to hold the table. This resulted from the lack of a microinstruction of the form;

Figure 3.23

Opcode	file register	AROM address

Load the specified file register with the contents of the given address.

The only Load instructions are Load Immediate instructions, where the literal in the low order 8 bits of the instruction are loaded into the specified register. As a result each entry in the ASCII table was composed of two microcommands. The first was a Load immediate instruction, the second was a Jump instruction. The microprogram that converts from EBCDIC to ASCII works as follows:

1. AND off the high order two bits of the EBCDIC character;
2. Shift the remaining 6 bits one place left;
3. Move the register containing these bits to the L register (This creates a multiply way branch, for now the low order 8 bits of the microlocation counter have been altered).
4. A branch to the ASCII table occurs where the correct ASCII character is loaded into a predetermined file register;

5. A Jump back to the card reader routine occurs;
6. At this point the conversion process has been accomplished.

The third problem encountered also concerned the card reader. The Microdata 1600/30 and the card reader controller were designed to recognize certain error conditions such as, pick failure, hopper empty, and illegal Hollerith Codes. When such an error is detected an external interrupt is generated. However, MIX does not specify how these interrupts should be handled. Therefore, two high core MIX words were dedicated to the card reader to allow the user to handle these I/O errors. If an external interrupt from the card reader is encountered the interrupts routine stores the card reader status byte and the contents of the MIX location counter in one of these dedicated locations and then loads the MIX location counter with the address of the other dedicated location. Thus the next instruction will be fetched from this interrupt address. If the user wishes to halt anytime a card reader error occurs, he simply loads this location, prior to run time with a MIX Halt instruction. If thes user wants to handle the error, then a Jump instruction should be placed at this location which will cause control to be transfered to the users error routines. Following the execution of the error routine, the user can load the old contents of the MIX Location Counter into the address portion of a MIX Jump instruction and jump back to section of MIX code being executed when the error occurred.

Typewriter and Paper Tape

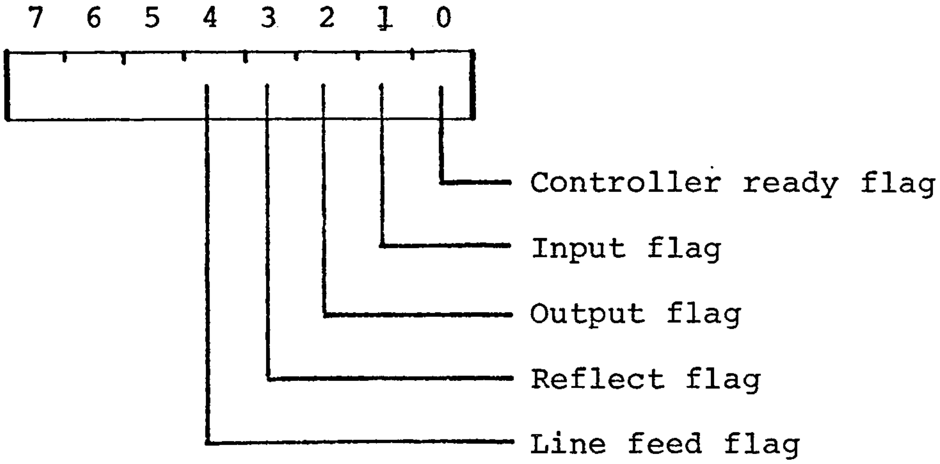
The typewriter and paper tape station available on the Microdàta 1600/30 is a 10 baud full duplex teletype. This device works in byte mode and no interrupts are set by the controller. This presented several problems. First, if teletype Input/Output was to be performed concurrently with the execution of MIX instructions the interrupt routine must handle all but the I/O intialization. However, since no data ready interrupts were available, some other mechanism had to be developed to time data transmissions due to the slowness of the teletype. The teletype could be polled on each execution of the interrupt routine, however, this routine is executed approximately once every 40 microseconds and the teletype transmits only 10 characters per second. Thus polling each time the interrupt routine was entered seemed somewhat extreme. The only other timing device available was the Microdata real time clock. This hardware clock generates an internal interrupt once every millisecond. This is still 100 times faster than the teletype; however, it was an improvement over a few cycle times. This interrupt scheme along with four dedicated bytes of surplus memory solved the teletype problem. Three of these dedicated bytes were used to provide the memory buffer address (16 bits) and the byte counter, as with the card reader and printer. The fourth byte was used as an internal MIX status byte for the teletype. One bit of this byte was labeled the "controller ready bit". If this bit was a one the device was ready to begin an I/O operation. If this bit was zero then the teletype was still involved in an I/O operation started previously. This bit was used both in the In and Out routines as well as in the interrupt routine. The In and Out routines loop through the interrupt routine until this bit becomes a one. The interrupt routine test this bit upon receiving a real time clock interrupt. If this bit is one then the teletype routine is skipped. If it is a zero an attempt is made to transfer a byte of data.

Four other bits of this internal MIX status bytes were assigned functions also. One bit was dedicated as an input flag and one as an output flag. This was necessary in order to determine which operation was to be perfomred.

One bit was used to remember when to send a line feed. In case of either teletype Input or Output, once 14 words have been transmitted a carriage return and line feed must be sent. The interrupt routine recognizes the need to send a carriage return when the counter goes to zero. Upon sending the carriage return the line feed flag is set to 1. The next time the teletype is polled, the interrupt routine finds the counter zero and the line feed flag turned on. The interrupt routine then sends a line feed and resets the internal status controller ready bit to a one.

The other dedicated bit in the internal status bit of the teletype is used to reflect input characters back to the typewriter (teletype is full duplex). When in Input mode, the interrupt routine must receive characters from the keyboard and then send this character back to the printing ball, so the characters being input will be written on the teletype paper. However, this reflection cannot be done immediately. The reflection must occur the next time the teletype is ready to receive data. Thus when a character is input from the keyboard, the interrupt routine receives the byte and stores it in the location specified by the memory address counter. However, the memory address counter and byte counter are not updated at this time. The interrupt routine turns the input bit off and turns on the reflection and output bits. The next time the teletype is ready the interrupt routine finds the output bit on so the data byte pointed to by the memory address counter is output and the counters are adjusted and read back to memory. The interrupt routine then tests to see if the reflection flag is on. If it is not, the interrupt routine leaves the teletype handler and proceeds to find other interrupts. But if the reflection flag is on, the interrupt routine turns this bit and the output bit off and turns the input bit on. Figure 3.24 illustrates the format if the teletype internal status byte.

Figure 3.24



Figures 3.25 and 3.26 present a general flowchart of the teletype internal status byte.

At this point a word concerning Input/Output timing seems appropriate. What quarentee is there that the interrupt routine will process I/O interrupts fast enough to avoid losing any information? This problem is most serious on the card reader since it is faster than the teletype. If there exist a MIX instruction whose execution time is longer than the time between card reader interrupts, then it is possible to miss information, for example, the routine might only receive every other byte. The printer is faster than the card reader, but once the printer is ready to receive a data byte, it will stay ready as long as no outside intervention occurs. However, when the card reader interrupts to request a transfer, this interrupt must be answered within a certain time frame or else the next byte of information will

Figure 3.25

Teletype In and Out Instructions

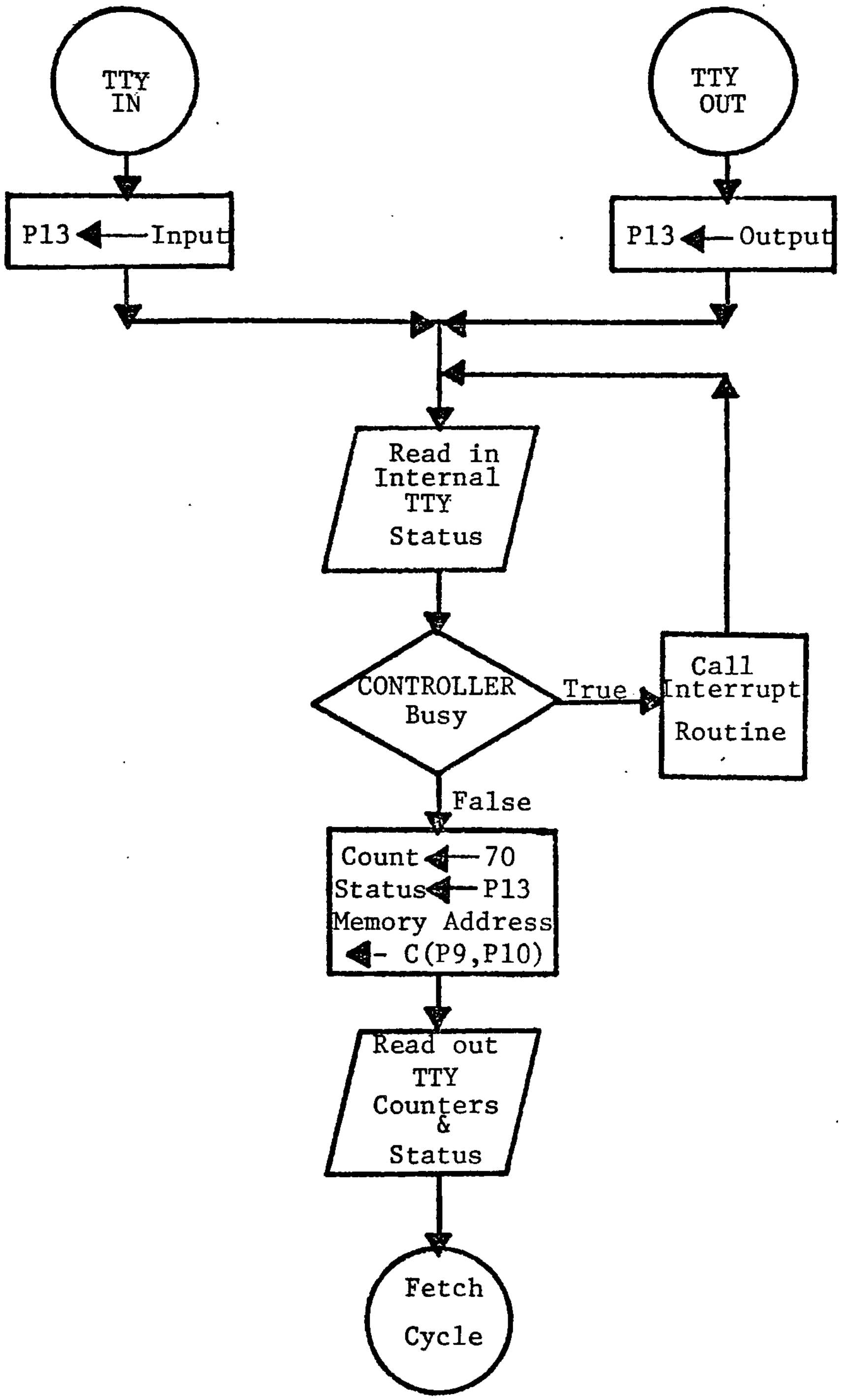
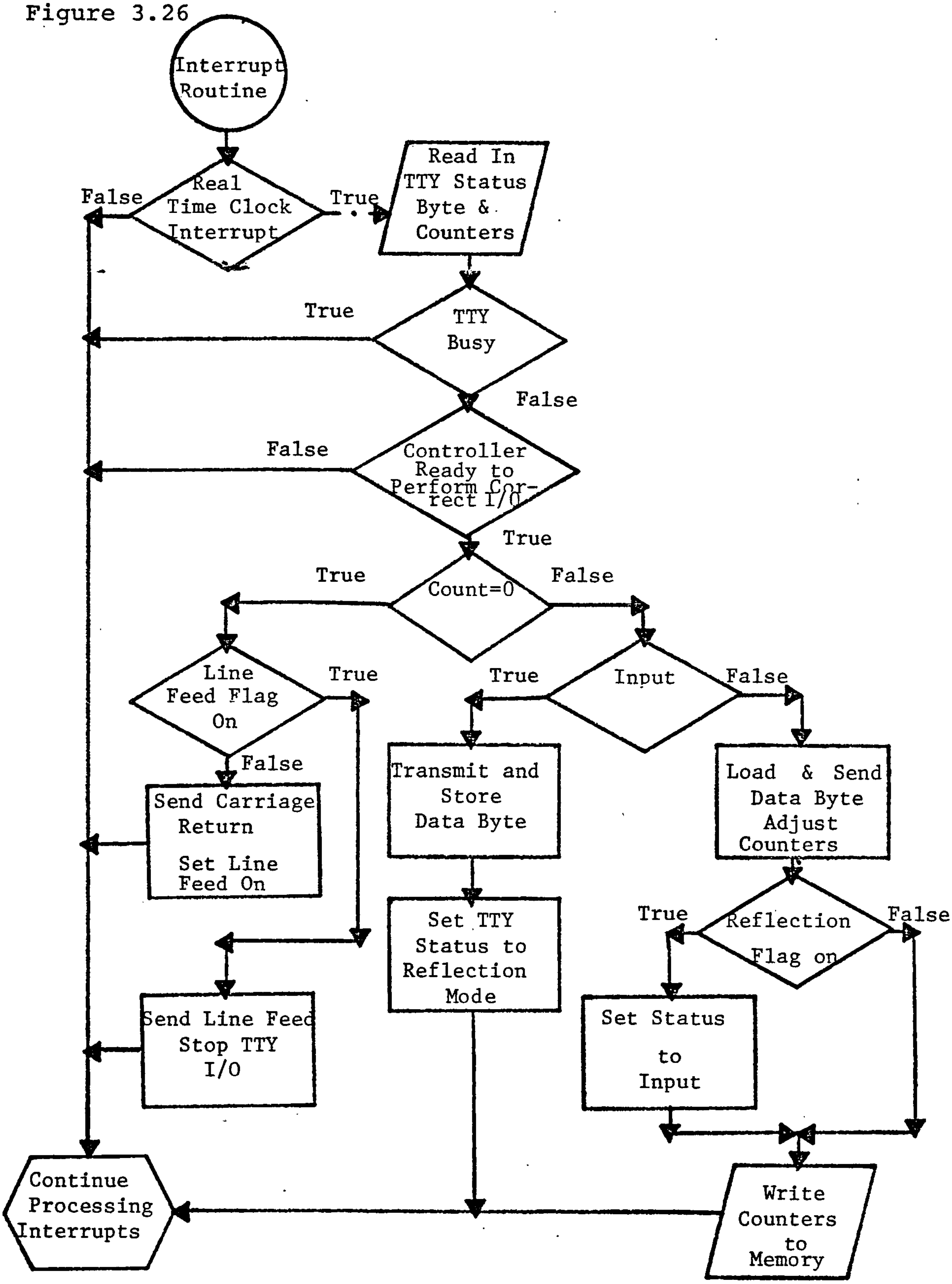


Figure 3.26


Interrupt Routine
Real Time Clock Interrupt
False
True
Read In TTY Status Byte & Counters
True
TTY Busy
False
False
Controller Ready to Perform Cor-rect I/O
True
True
Count=0
False
Line Feed Flag On
True
True
Input
False
False
Send Carriage Return Set Line Feed On
Transmit and Store Data Byte
Load & Send Data Byte Adjust Counters
True
Reflection Flag on
False
Set TTY Status to Reflection Mode
Send Line Feed Stop TTY I/O
Set Status to Input
Continue Processing Interrupts
Write Counters to Memory

arrive, overlaging the previous byte.

To show that the microroutine is fast enough to handle the card reader, the longest possible MIX instruction cycle time must be shown to be shorter than the shortest possible interval between data signals. Figure 3.27 shows how an upper bound on the longest path through the MIX emulation can be found.

Figure 3.27

1. Execution of the fetch routine requires 44 clock pules (200 nanosecond pulses).
2. The memory address and decode routines contain no loops and the total number of instructions involved in both routines is 127. This includes all possible paths.
3. The interrupt routine is 215 instructions long. Note: Card reader interrupts are handled first by this routine.
4. The longest MIX instruction is the Char instruction which is 116 commands long, some sections of which are executed 10 times.
5. An upper bound on the worst case is thus 44 + 127 + 215 + 116 * 10 = 1546 clock pulses.

The data signal from the card reader lasts, in the worst case (allowing for skewness and taking only the highest light signal), at least .5 milliseconds or 500,000 nanoseconds

($.5 * 10^{-3} = 500{,}000 * 10^{-9}$). One clock pulse occurs every 200 nanoseconds on the Microdata, so each data signal lasts 2,500 clock pulses, safely more than the 1544 required.

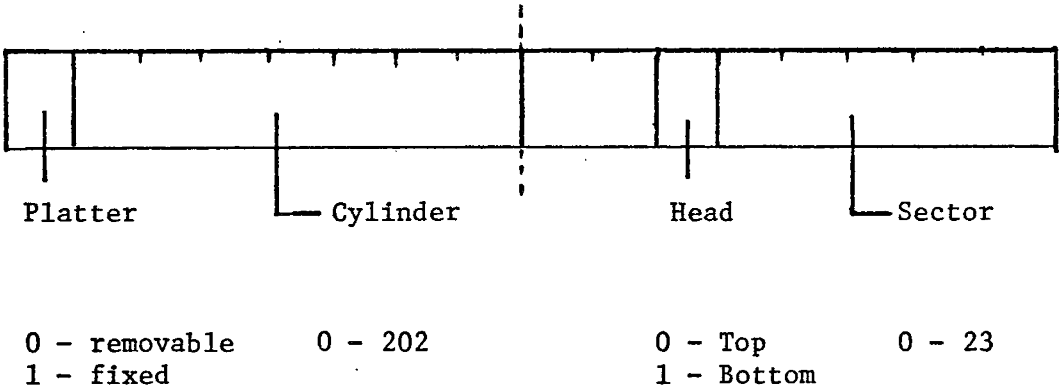
Disks

The disk drivers were the simplest I/O handlers written due to the hardware available on the Microdata. However, several problems were encountered. Knuth specified that each disk record should consist of 100 MIX words. This works out to 800 Microdata bytes. However, the smallest addressable block on the Microdata disk is 256 bytes long. Thus to accommodate 800 bytes, three full sectors plus 32 bytes of a fourth sector are required. This wastes the remaining 224 bytes of the fourth sector. The disk record size was therefore made to be variable in length, with the hope that MIX users would use records sizes that are multiples of 32 MIX words. The length of the disk record in MIX words to be read or written is placed in bytes 2 and 3 of the X register.

The X register is also used to select the beginning sector address, (bytes 4 and 5); however, the MIX user must use the Microdata's scheme for addressing different sectors of the disk. Figure 3.28 gives the addressing format as well as the format of the X register. Figure 3.28 appears on the following page.

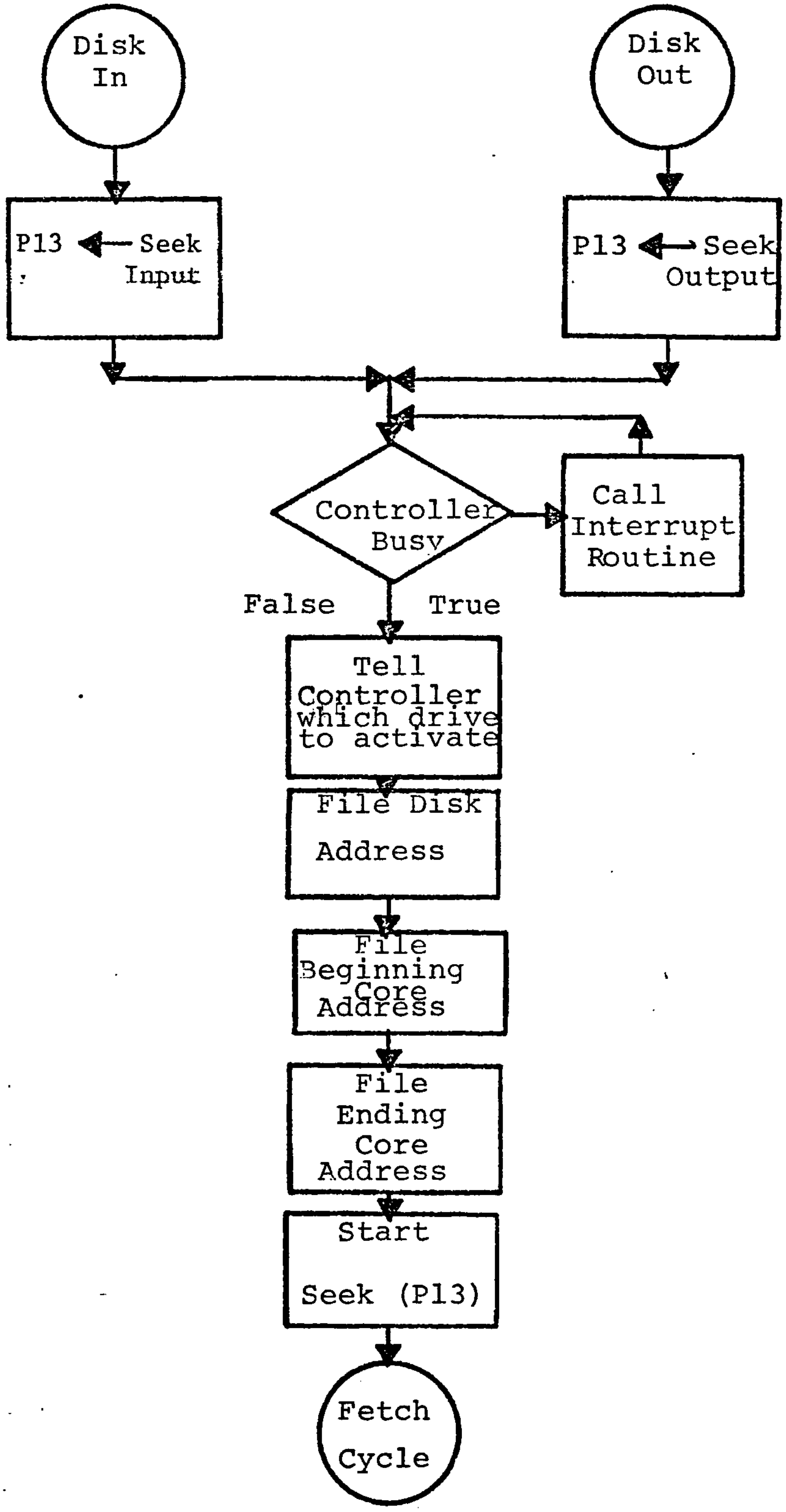
Figure 3.28

Sign	
1st data byte	Number of MIX words to transfer
2sd data byte	
3rd data byte	
4th data byte	Sector Address (16 bit)
5th data byte	



The other problem encountered concerned the IOC instructions when this instruction is executed with the disk as the selected device, the disk read/write heads are to be positioned to the cylinder address found in the X register. This is not possible on the Microdata since once the Seek command is given the heads are positioned and the I/O must follow immediately. Figure 3.29 gives a general overview of the disk operation.

Figure 3.29


Disk In
Disk Out
P13 ← Seek Input
P13 ↔ Seek Output
Controller Busy
False
True
Call Interrupt Routine
Tell Controller which drive to activate
File Disk Address
File Beginning Core Address
File Ending Core Address
Start Seek (P13)
Fetch Cycle

IV. USER'S GUIDE

This Chapter provides a user's guide for the MIX 1009 machine as emulated on the Microdata 1600/30. Two sections are presented; the first explains the functions of the sys~~tem console as they pertain to the~~ MIX 1009 machine. For more information on the 1600/30 console see Microdata (3); the second section explains the ten dedicated MIX memory locations.

System Console

Figure 4.1 presents a frontal view of the Microdata 1600/30 console. This console provides the user with the hardware to cold start the MIX machine and to execute and debug MIX programs.

1. Key-Lock Switch

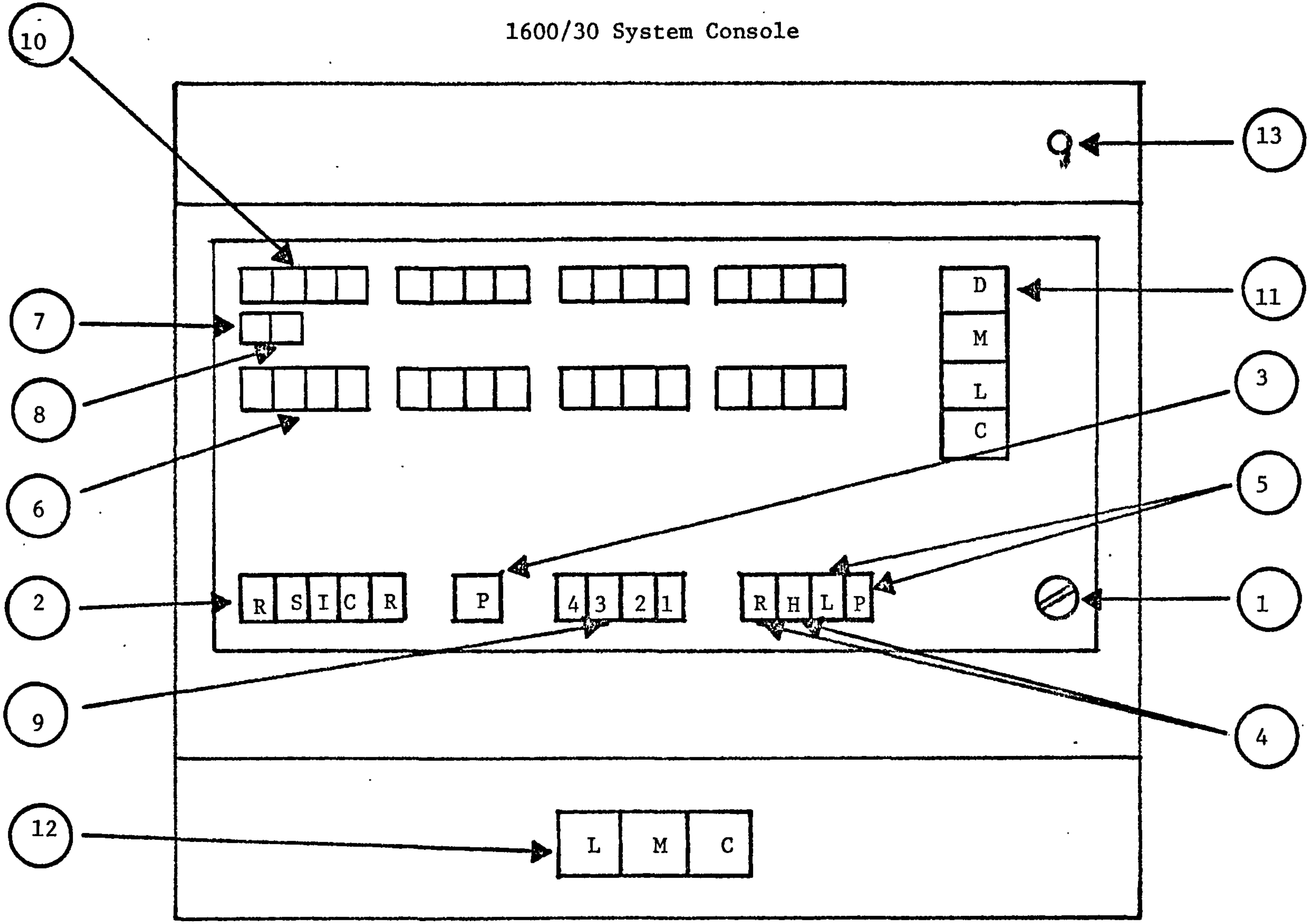
   This switch can be turned to one of three positions. The key-lock switch should be set to the ON position when using the MIX 1009 machine. This supplies power to the CPU and enables the PANEL mode switch.

2. Machine State Control Switches

   This group of five momentary contact switches are activated when pressed down. Each switch is described below:

Figure 4.1 (2)

1600/30 System Console



A. RESET Switch

When pressed this switch clears and halts the CPU. It is used only during COLD START (Refer to MIX COLD START Procedure).

B. CLOCK Switch

When the CPU is the RUN state, pressing the CLOCK switch forces the CPU to halt after executing the current microcommand. When the CPU is in the HALT state, pressing the CLOCK Switch forces the CPU to fetch and execute the next microcommand and then halt.

C. INT Switch

When the INT switch is pressed, a console interrupt is recognized by the MIX 1009 machine. This invokes the MIX initialization sequence (Refer to MIX COLD START Procedure).

D. STEP Switch

Pressing the STEP switch forces the MIX processor to execute the next MIX instruction and HALT. When the HALT occurs the M display may be selected and the address of the next MIX instruction, in binary will appear on the 16 data lights.

E. RUN Switch

Pressing the RUN switch places the CPU in the RUN state. The CPU remains in this state until a halt instruction is executed or until the CLOCK, STEP, INT or RESET switch is pressed.

3. PANEL MODE Switch

When the key-lock switch is in the ON position and the PANEL MODE switch is in the UP position the MIX processor will execute MIX instructions when placed in the RUN state. When the key-lock switches in the ON position and the PANEL MODE switch is in the DOWN position most of the MIX registers can be displayed and modified via the DATA switches and the CLOCK switch.

4. Machine State Indicator Lights

When the RUN indicator is lit the CPU is in the RUN state. When the HALT indicator is lit the CPU is in the HALT state.

5. Panel STATUS Indicator Lights

When the key-lock switch is set to LOCK, the LOCK indicator comes on. When the key-lock switch is set to On and the PANEL MODE switch is down, the PANEL light comes on.

6. DATA Switches

When the PANEL INDICATOR is off, all 16 DATA switches should be in the UP position. When the PANEL INDICATOR is on the 16, DATA switches can be used to display and modify most of the MIX registers. The DATA switches are numbered from 0 to 15, starting from the RIGHT. A binary 1 is indicated when a DATA switch is down, a binary zero is indicated when the switch is up.

7. Address Stop Indicator

This indicator is not used by the MIX 1009 machine.

8. Scan Indicator

This indicator is not used by the MIX 1009 machine.

9. SENSE Switches

These four switches are used by the MIX 1009 machine during COLD START and BOOTSTRAP operations.

10. DATA Display

The 16 DATA display lights allow the MIX user to view, in binary, the address of the next MIX instruction or the current contents of a selected MIX register. A binary 1 is indicated when a data light is on, a binary 0 if the light is off.

11. DISPLAY SELECT Push Buttons

These push buttons select the source of the data displayed on the 16 DATA display indicators. The L and C push buttons are not used on the MIX machine.

M-Core Memory Address

This button is depressed when the MIX user is executing a program via the STEP switch. When the M button is selected and the STEP switch is depressed the address of the next MIX instruction will appear, in binary, on the DATA display indicators.

D-DATA

The D push button is selected when the MIX user is examining and /or changing the contents of MIX registers via the PANEL, DATA, and CLOCK switches.

12. AROM Control Switches

The LINK CONTROL switch, MANUAL OPERATION switch and CONTROL MODE switch control the operating environment of the Alterable Control Memory (ACM). All three switches should be DOWN for proper operation of the MIX 1009 machine.

13. COLD START Button

The disk controller allows the Microdata 1600/30 to cold start from a loader program stored on a disk drive. Pressing the COLD START Button causes the first 256 bytes of the removable platten on drive 0 to be loaded into the first 256 bytes of core memory.

MIX Console Procedures

The MIX user has available five major Console Procedures. The MIX user can COLD START the MIX 1009 computer, BOOTSTRAP an object program into memory, STEP through MIX programs one instruction at a time, DISPLAY most MIX registers, and MODIFY most MIX registers. The following is a detailed discussion of each procedure.

MIX COLD START Procedure

The microprogram that defines the MIX 1009 program on the Microdata 1600/30 resides in the Microdata's Alterable Control Memory. This ACM is a volatile memory, the contents of which are filled with binary ones each time the AROM CONTROL MODE switch is placed in the UP position. Each time this occurs and before a MIX program can be loaded the MIX emulator must be reloaded into the AROM. This procedure is called a MIX COLD START. The COLD START hardware and loader program are the same as the ones used to perform initial program loads for the operating systems on the Microdata 1600/30. Pressing the COLD START Button causes the first sector (256 bytes) of the removable disk of drive zero to be loaded into the low 256 bytes of memory. Pressing the RUN switch causes this loader program to begin execution. This program can load any of four systems. The user specifies which system is to be loaded by setting the system CONSOLE SENSE switches. To specify the MIX emulator, SENSE switches 4 and 3 should be UP and SENSE Switches 2 and 1 should be DOWN. The COLD START load routine, upon testing these switches, will read into memory the AROM Load Program and the MIX eumulator and jump to the beginning of the AROM Load Program. The AROM Load Program then loads the AROM with the MIX emulator.

The AROM Load Program will inform the user of any errors that occur during transmission. When the AROM has been successfully loaded the CPU will halt. Then, when the RUN switch is pressed the MIX 1009 computer will begin executing MIX instructions. The MIX COLD START Procedure is as follows:

1. Turn the Key-Lock switch to the On position;
2. Set the DATA switches and the PANEL MODE switch UP;
3. Set the AROM CONTROL switches DOWN;
4. Flip SENSE Switches 3 and 4 UP, and SENSE switches 1 and 2 DOWN;
5. Press the CLOCK SWITCH;
6. Press the RESET switch;
7. Press the COLD START Button;
8. Press the RUN switch.

If an AROM load error occurs the user can attempt a reload as follows;

9. Set SENSE switches 2, 3, and 4 UP. Set SENSE switch 1 DOWN:
10. Press the RUN switch;

If an AROM load error does not occur or occurs but the user wishes to ignore the error, then:

9. Set SENSE switches 1, 3, and 4 UP; set SENSE switch 2 DOWN;
10. Press the RUN switch.

BOOT STRAP Procedures

The BOOTSTRAP procedure allows the MIX user to load the first MIX program into memory, after the MIX emulator has been loaded. The BOOTSTRAP Procedure is emulations's implementation of Knuth's GO button. When Knuth's GO button is pressed, a single card is read from the card reader into MIX locations 0000-000F. When the card reader is no longer busy, the MIX computer begins executing the program just read from this card starting at location 0000. The BOOTSTRAP Procedure corresponding to Knuth's GO button is as follows:

1. COLD START the MIX 1009 computer;
2. Place the load program card in the card reader and ready the card reader;
3. Set SENSE switch 4 DOWN and the other SENSE switches UP;
4. Press the INT switch;
5. Press the RUN switch.

STEP Procedure

The STEP Procedure may be invoked at any time the user wishes to execute a single MIX instruction at a time. This procedure is as follows:

1. PRESS the M DISPLAY SELECT push button;
2. Press the STEP Switch.

Each time the STEP switch is pressed one MIX instruction is executed and the address of the next MIX instruction to be executed will be displayed in binary on the DATA DISPLAY INDICATORS. The user should note the I/O instructions will not work correctly if they are executed by pressing the STEP switch, since the I/O devices work separately from the CPU.

DISPLAY MIX Registers Procedures

It is possible for the MIX user to display most of the MIX registers whenever the MIX 1009 computer is halted, (i.e. when the HALT Machine Indicator Light is ON). The MIX A, X, J. Instruction Counter, Overflow Toggle and Comparison Indicator can be displayed one byte at a time whenever the HALT light is on. However only one Index register is available for display at any given time. The MIX registers emulated in the Microdata's 30 general puropse registers as shown in figure 4.2. These 30 general purpose registers are divided into two files, the Primary files P1-P15 and the Secondary files S1-S15. Only one set of files may be addressed at any one time. When the STEP switch is pressed, the Primary files are selected and any byte of the A register or of the Instruction Counter can be displayed. If information in the Secondary files is to be displayed the Secondary file must first be selected. Once this has been done, any byte in the Secondary files, S1-S15, can be displayed. If

Figure 4.2

E T $\overset{I}{O}$ I $\overset{I}{O}$ Z N O PSO

A Register	sign	P1	sign	S1	X Register
	1^{st} byte	P2	1^{st} byte	S2	
	2^{nd} byte	P3	2^{nd} byte	S3	
	3^{rd} byte	P4	3^{rd} byte	S4	
	4^{th} byte	P5	4^{th} byte	S5	
	5^{th} byte	P6	5^{th} byte	S6	
Free work Register		P7	OLEG x iii	S7	Overflow, Comp., & Index Map
Instruction Register	sign	P8	sign	S8	Index Register i
	A1 address-ho	P9	1^{st} byte	S9	
	A2 address-lo	P10	2^{sd} byte	S10	
	I index spec.	P11		S11	Free work Registers
	F field spec.	P12		S12	
	C opcode	P13		S13	
Instruction Counter	1^{st} byte-ho	P14	1^{st} byte-ho	S14	Jump Register
	2^{nd} byte-lo	P15	2^{sd} byte-lo	S15	

the user wishes to check again some information in the Primary files, the Primary files must first be selected since the Secondary files are currently the ones available for display. The procedures for displaying information, and selecting the Secondary and Primary file follow:

Select Secondary Files Procedure

1. Load the following bit pattern into the DATA switches: 0001 0000 1000 0000 (1080_{16})
2. Set the PANEL MODE switch to the DOWN position;
3. Press the CLOCK switch.

Select Primary Files Procedure:

Note: The Primary files are automatically selected when the STEP switch is pressed. This procedure need be involked only when the user has pressed the STEP switch and then performed the Select Secondary Files Procedure and now wishes to select the Primary files once again.

1. Load the following bit pattern into the DATA switches: 0001 0000 0100 0000 (1040_{16});
2. Set the PANEL MODE switch to the DOWN position;
3. Press the CLOCK switch.

Display Primary File Register

1. Select Primary files either by pressing the STEP switch or by performing the Select Primary Files Pro.;

2. Press the D DISPLAY SELECT Push button;

3. Set the PANEL MODE Switch to the DOWN position;

4. Load the following bit pattern into the DATA switches 1100 XXXX 0000 0000 ($CX00_{16}$) where XXXX is the binary equivalent of the file number to be diaplayed. For example to display P11 enter 1100 1011 0000 0000 ($CB00_{16}$).

5. Press the CLOCK switch.

The contents of the file register selected will appear on the eight right hand DATA DISPLAY lights.

Display Secondary File Register Procedures

This procedure is the same as the Display Primary File Register Procedure, only in Step 1 the Secondary files must be selected instead of the Primary files.

Note from figure 4.2 that Secondary file registers S8, S9, S10 are dedicated to Index register i. This is the only one of the six MIX Index registers that can be displayed. The user can determine which of the six Index registers is occupying these locations by examining the low order three bits of Secondary file register S7.

Modify MIX Registers Procedures

MIX registers can be modified one byte at a time. The procedure is similar to the Display Procedures. The Procedure is as follows;

1. Determine from figure 4.2 the location of the MIX register to be modified.
2. Using the Select Files Procedure select the proper set of file registers.
3. If required the user may now display the contents of the chosen MIX register;
4. Set the PANEL MODE switch DOWN;
5. Enter the following bit pattern into the DATA switches: 0010 XXXX YYYY YYYY, where XXXX is the binary equivalent of the file number to be loaded, and YYYY YYYY is the 8 bit binary value to be loaded. For example, to load file 3 with a decimal 18 set the DATA switches to 0010 0011 0001 0010.
6. Press the CLOCK Switch.
7. The user may now display the new contents, change them again, or change some other register.

Note: After Displaying or Modifying registers the user must raise the PANEL MODE switch before pressing the STEP 62 RUN switch. The user need not select the Primary files before executing the next instruction, as the MIX emulator will reset the CPU to the Primary files upon leaving the STEP Sequence.

Dedicated MIX Memory Locations

The last ten words of MIX memory have been dedicated to specific functions of the MIX machine figure 4.3 and the description that follows explains these functions.

Note: MIX users should never use the last 18 locations of memory as buffer area for disk operations as certain values which are transparent to the user would be destroyed if the disk were allowed to input data to this area.

MIX Words OFF6-OFF7

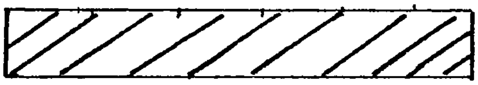
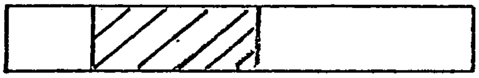
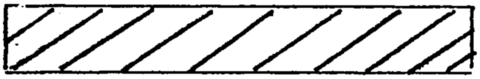
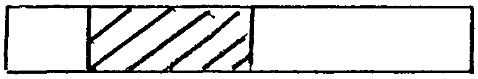
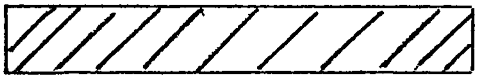
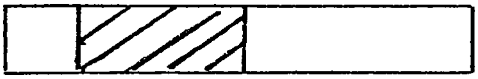
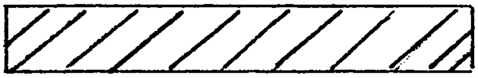
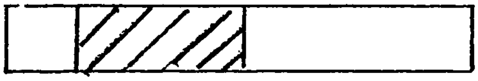
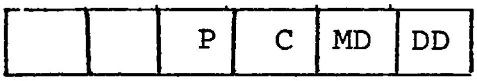
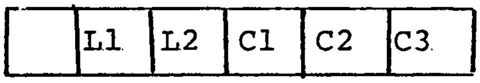
Illegal address, Illegal Opcode, Illegal Device,
Trap Locations

An attempt by the MIX user to execute an instruction containing an illegal address, opcode, or device number, invokes the MIX User Abort Sequence. This sequence causes the execution of the following two steps:

1. The current contents of the MIX Instruction Counter are written into the (1:2) field of MIX word OFF 7.

Note: The address of the illegal instruction is the contents of the Instruction Counter minus one.

Figure 4.3

MIX Address			
Hexidecimal	Decimal	Format	Function
OFF6	4086		Abort Instruction
OFF7	4087		Abort Return Address
OFF8	4088		Hopper Empty Instruction
OFF9	4089		Hopper Empty Return Address
OFFA	4090		Card Reader Error Instruction
OFFB	4091		Card Reader Error Return Address
OFFC	4092		Disk Error Instruction
OFFD	4093		Disk Error Return Address
OFFE	4094	 P C MD DD	P - Printer Status C - Card Reader Status MD - Major Disk Status DD - Diagnostic Disk Status
OFFF	4095	 L1 L2 C1 C2 C3	L1 - MSB } of Load Address L2 - LSB } C1 - MSB } of MIX Clock C2 - C3 - LSB }

2. The Instruction Counter is loaded with OFF 6 and execution continues, beginning with the instruction found at OFF 6.

To use this facility the user should load MIX location OFF 8 with either a Halt instruction or a Jump instruction. If the Halt instruction is used the MIX processer will halt when an illegal instruction is used. The MIX processer will halt when an illegal instruction is encountered. However, the iser may wish to weite his own routine to handle illegal instructions, for example, a core dump routine. In which case OFF 6 would contain a Jump to this routine. The user can also choose to continue processing by jumping to the address found in the (1:2) field of location OFF 7.

MIX WORDS OFF8 - OFFD

Card Reader Trap Locations

A. Hopper Empty Trap Location

An attempt to execute an IN instruction directed to the cardreader when the hopper of the card reader is empty invokes the Hopper Empty Trap Sequence. This sequence causes the execution of the following two steps:

1. The contents of the MIX instruction Counter are written into the (1:2) field of MIX word OFF 9.

Note: This is the instruction immediately following the interrupting IN instruction. The IN instruction has not been executed

2. The Instruction Counter is loaded with OFF8, and execution continues, beginning with the instruction found at OFF8.

B. Illegal Character Code or Mechanical Failure Trap Location

Failure of the card reader to recognize a character punched on the card currently being read or failure to correctly pick the next card invokes the Illegal Character Code or Mechanical Failure Trap Sequence. This sequence causes the execution of the following three steps:

1. The contents of the MIX Instruction Counter are written into the (1:2) field of MIX word OFF B.

Note: This is the next instruction to be executed.

2. The card reader status byte is stored in the (3:3) field of MIX word OFF E.

3. The Instruction Counter is loaded with OFF A, and execution continues, beginning with the instruction found at OFF8.

MIX Words OFFC - OFFD

Disk Trap Locations

If a disk error is sensed by the disk controller, the Disk Trap Sequence is invoked. This sequence occurs in three steps.

1. The contents of the MIX Instruction Counter are written into the (1:2) field of MIX word OFFD.

   Note: This is the next instruction to be executed.

2. The Disk Major Status is written into the (5:5) field of MIX word OFFE. The Disk Diagonistic Status is written into the (4:4) field of MIX word OFFE.

3. The Instruction Counter is loaded with OFFC, and execution continues.

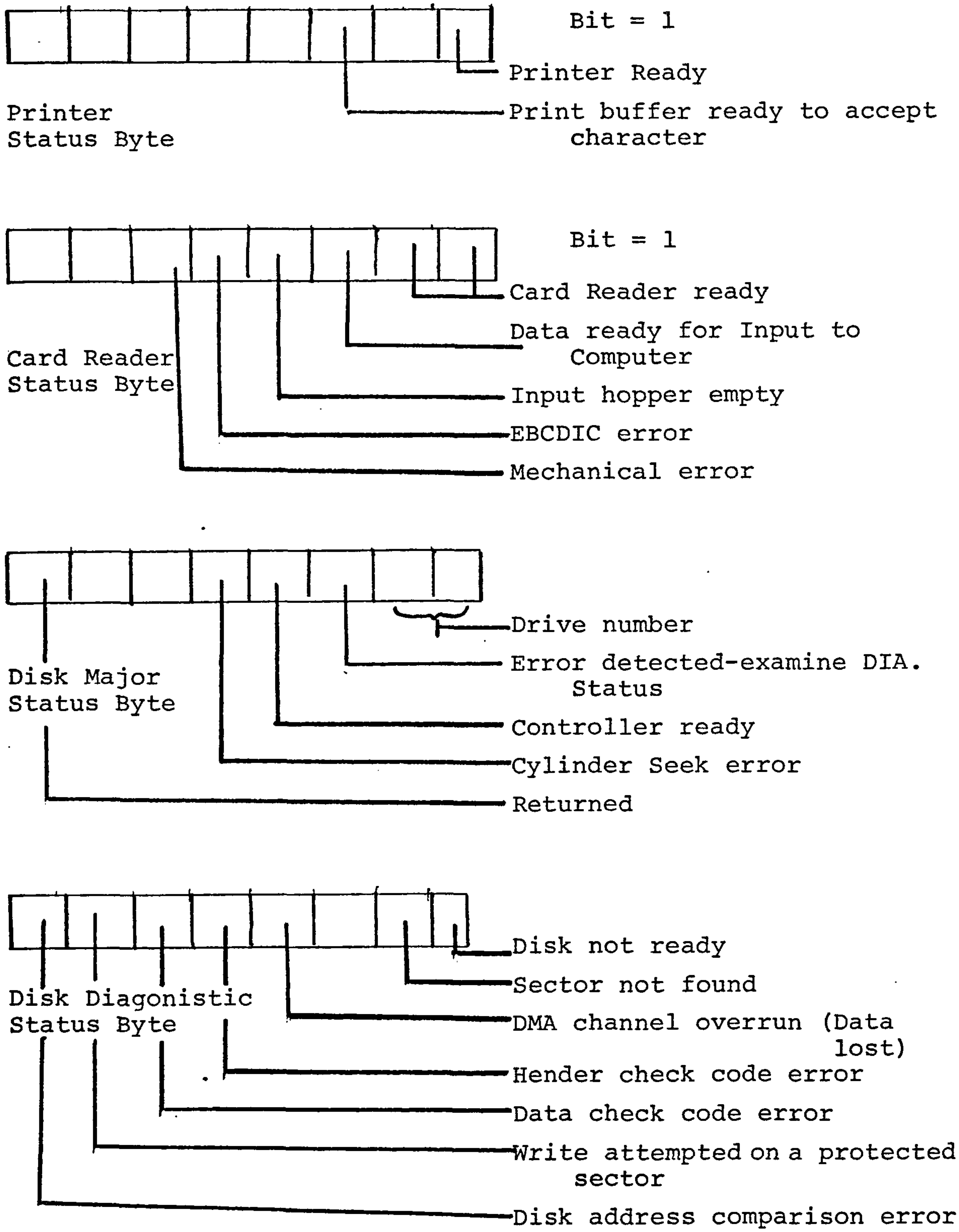
MIX Word OFFE

MIX I/O Status word

Following the completion of card reader, printer or disk I/O operation the status of the device involved is stored in MIX word OFFE. The user can sample this word to determine the outcome of the last operation. Figure 4.4 gives the format for each status word.

MIX Word OFFE

Figure 4.4


Bit = 1
Printer Ready
Print buffer ready to accept character
Printer Status Byte
Bit = 1
Card Reader ready
Data ready for Input to Computer
Input hopper empty
EBCDIC error
Mechanical error
Card Reader Status Byte
Drive number
Error detected-examine DIA. Status
Controller ready
Cylinder Seek error
Returned
Disk Major Status Byte
Disk not ready
Sector not found
DMA channel overrun (Data lost)
Hender check code error
Data check code error
Write attempted on a protected sector
Disk address comparison error
Disk Diagonistic Status Byte

MIX Word OFFF

MIX Load Address and MIX Real Time Clock

When one cold starts the MIX processer or when the PANEL Interrupt Switch is depressed, the MIX processer is reset and the Instruction Counter is loaded from the (1:2) field of MIX word OFFF. Also at these times the MIX Real Time Clock counter, field (3:5) of this word, is set to zero. This field is incremented by one every millisecond while the processor is running.

V. FLOWCHARTS

I. Driver Routines

- A. GO
- B. FETCH
- C. ADDRESS
- D: DECODE

II. Instruction Repertoire

- A. ADD and SUB
- B. MUL
- C. DIV
- D. NUM
- E. CHAR
- F. SHIFT
- G. MOVE
- H. LOAD
- I. STORE
- J. INPUT/OUTPUT
- K. JUMP
- L. ENTER
- M. COMPARE

III. Subroutines

- A. ABORT
- B. ERROR
- C. Z11, Z6, Z4

D. L.R

E. ID.IX, IDIX, .IDIX

F. ICOX

G. I.DOX, IDOX

H. .OUT, O.UT, I.OUT

I. .IN

J. ISKIP

K. INDEX Supervisor, PAGE, VIA, VINDEX

L. INTERRUPT

M. PRINTER, T.OUT

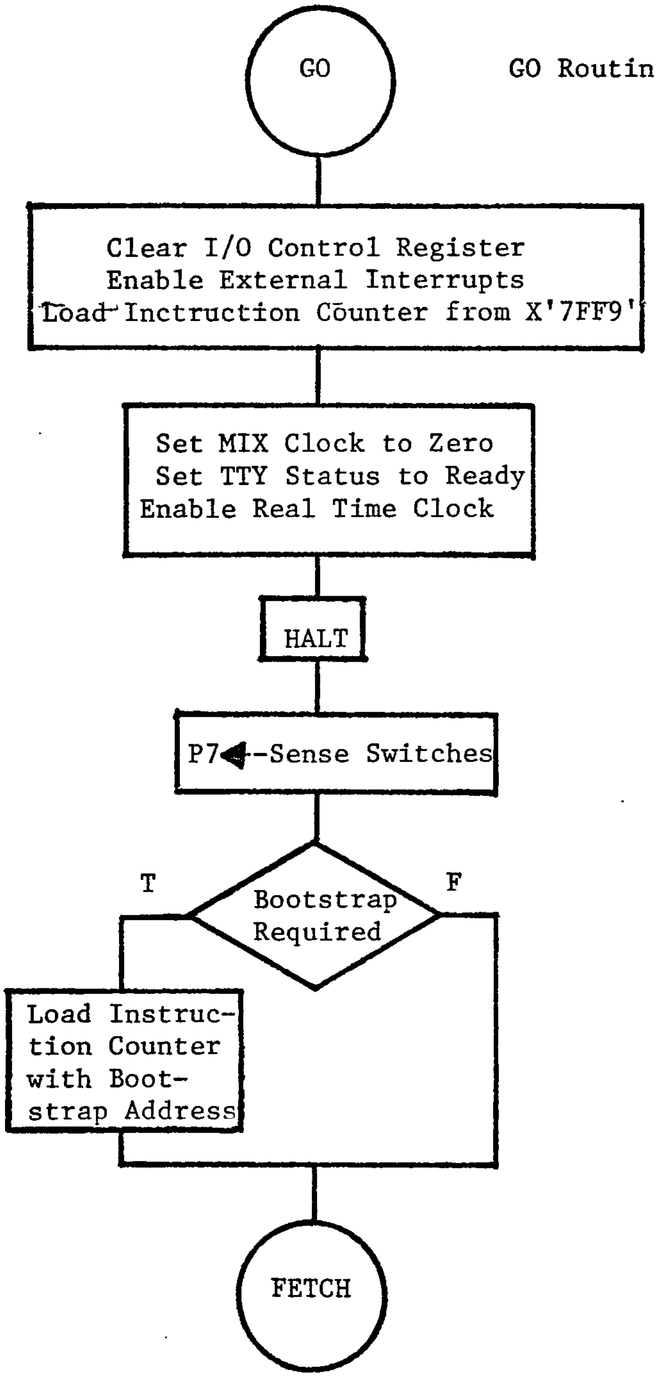
N. READER, T.IN

O. IREADER

P. IDSK

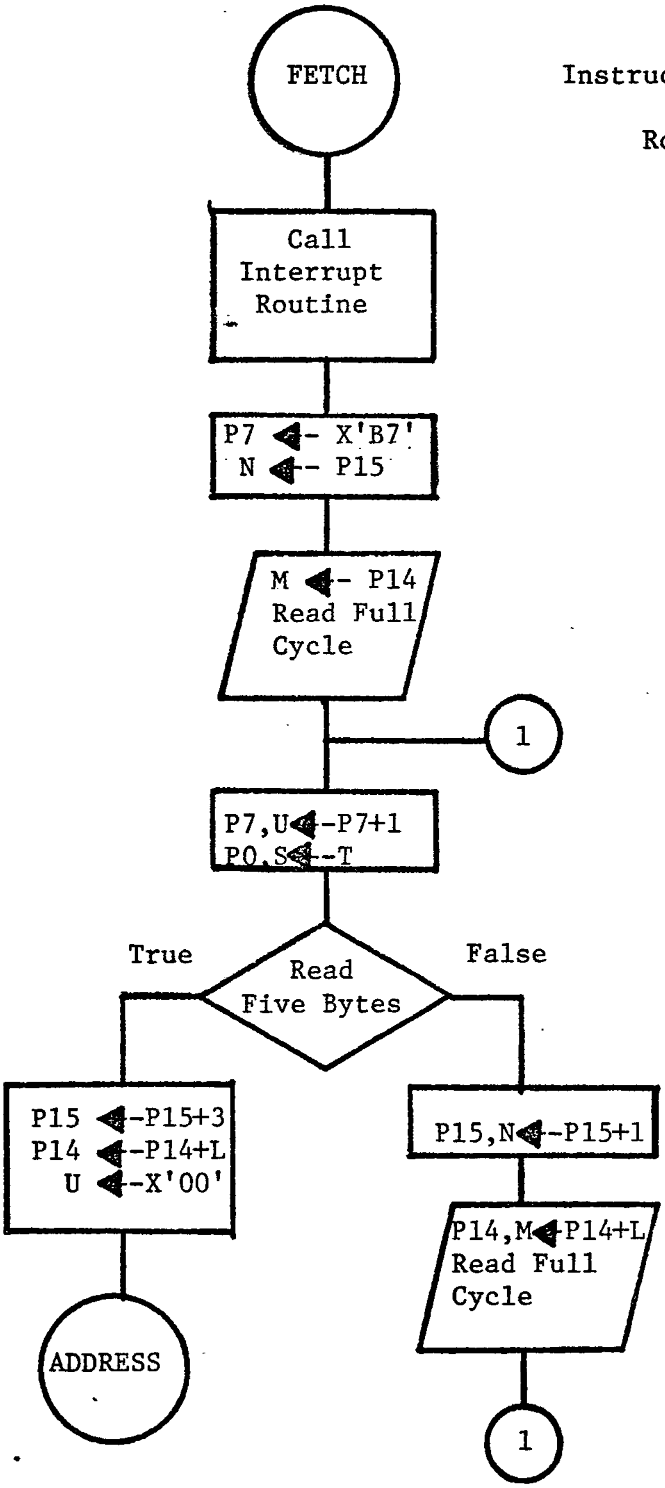
I. Driver Routines

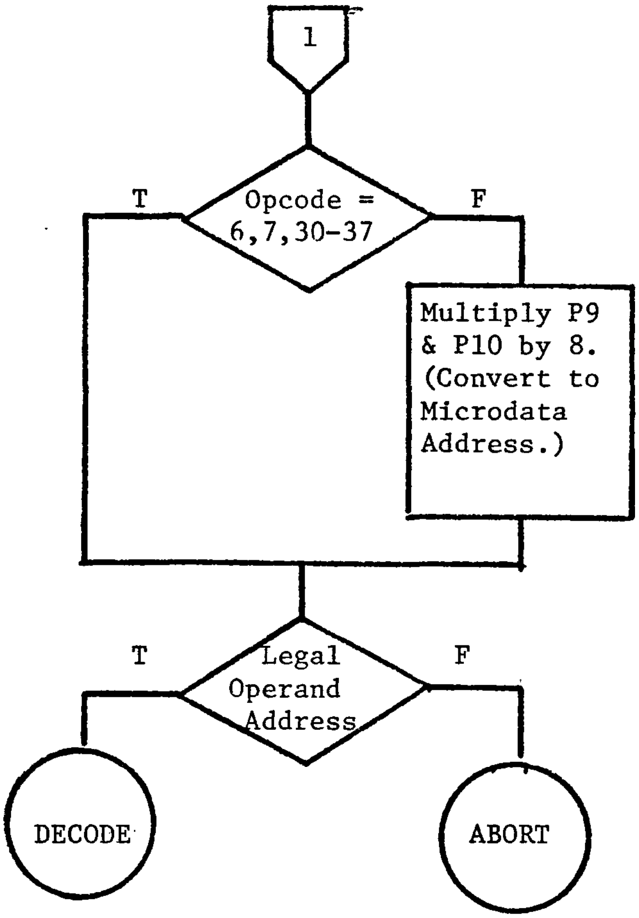
GO Routine

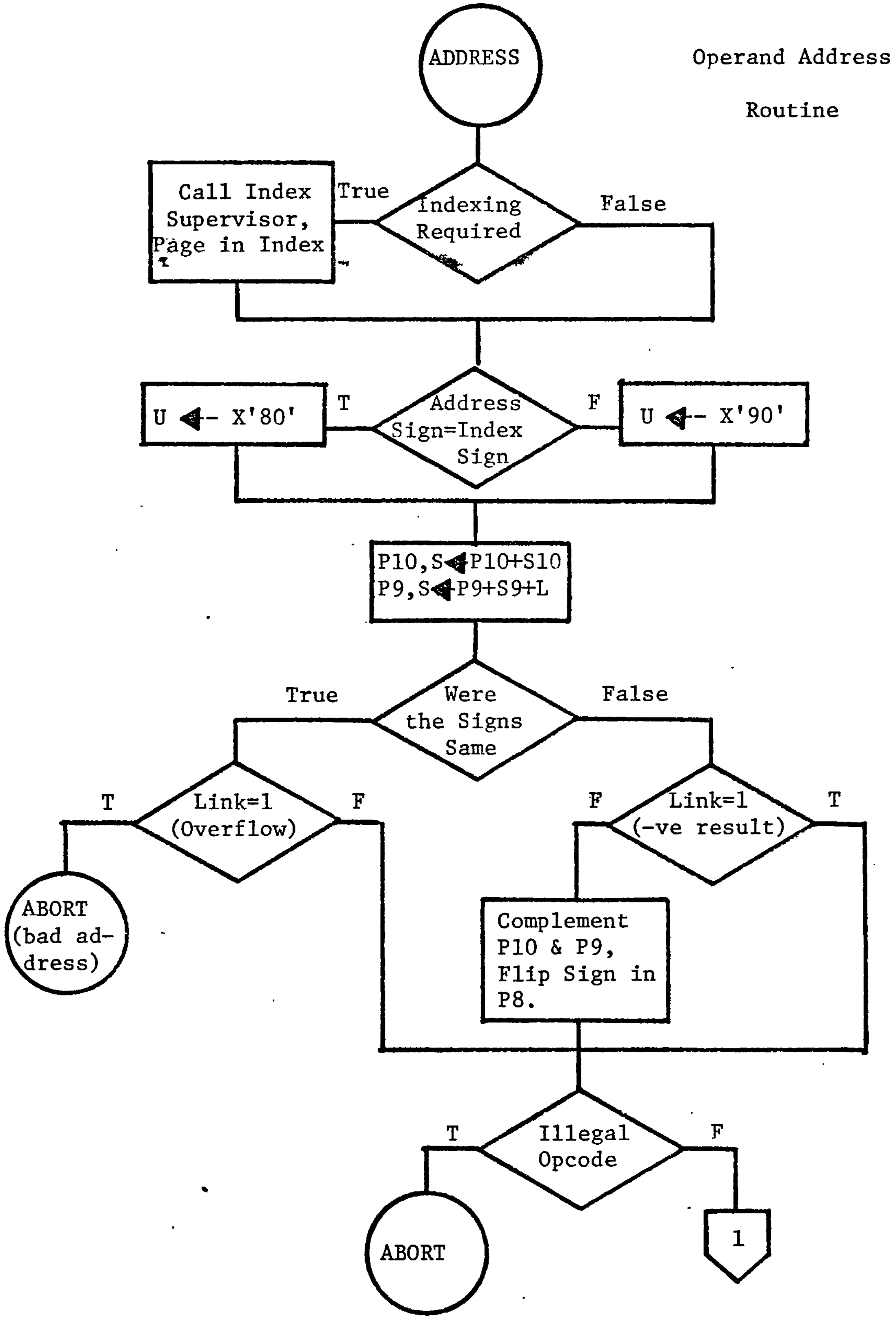

GO
Clear I/O Control Register
Enable External Interrupts
Load Inctruction Counter from X'7FF9'
Set MIX Clock to Zero
Set TTY Status to Ready
Enable Real Time Clock
HALT
P7←-Sense Switches
T
F
Bootstrap Required
Load Instruction Counter with Bootstrap Address
FETCH

Instruction Fetch

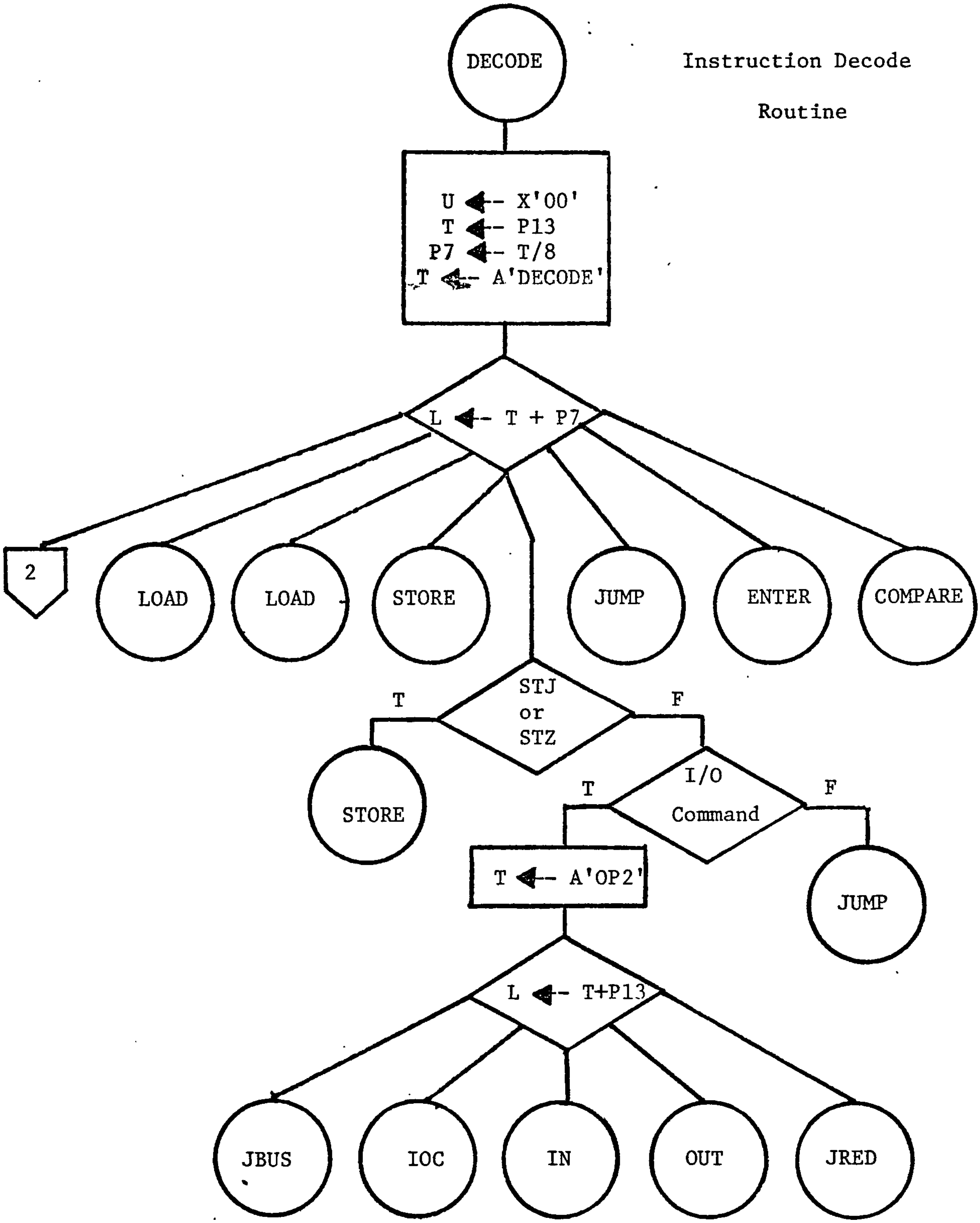
Routine

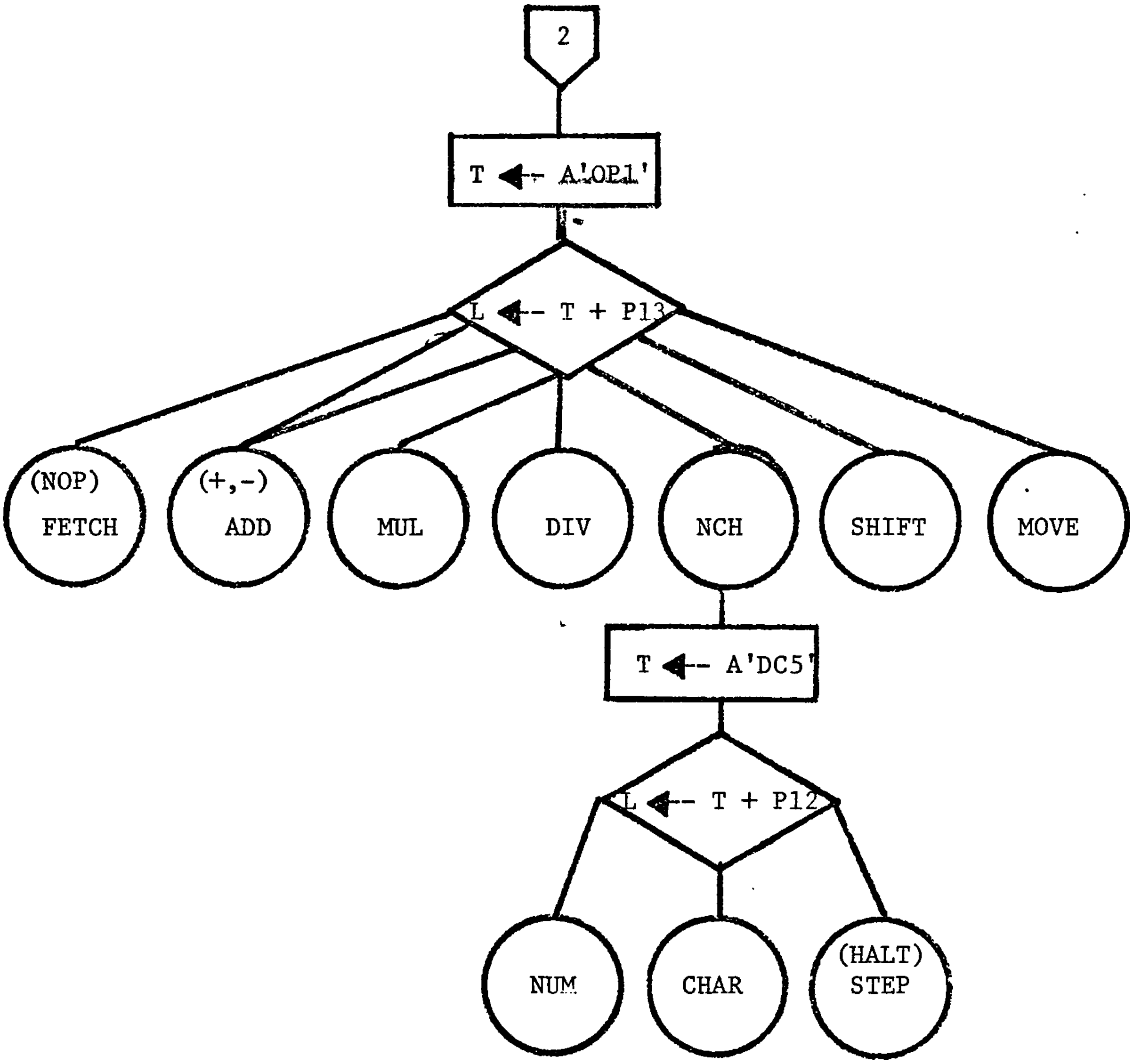



1
Opcode = 6,7,30-37
T
F
Multiply P9 & P10 by 8. (Convert to Microdata Address.)
Legal Operand Address
T
F
DECODE
ABORT

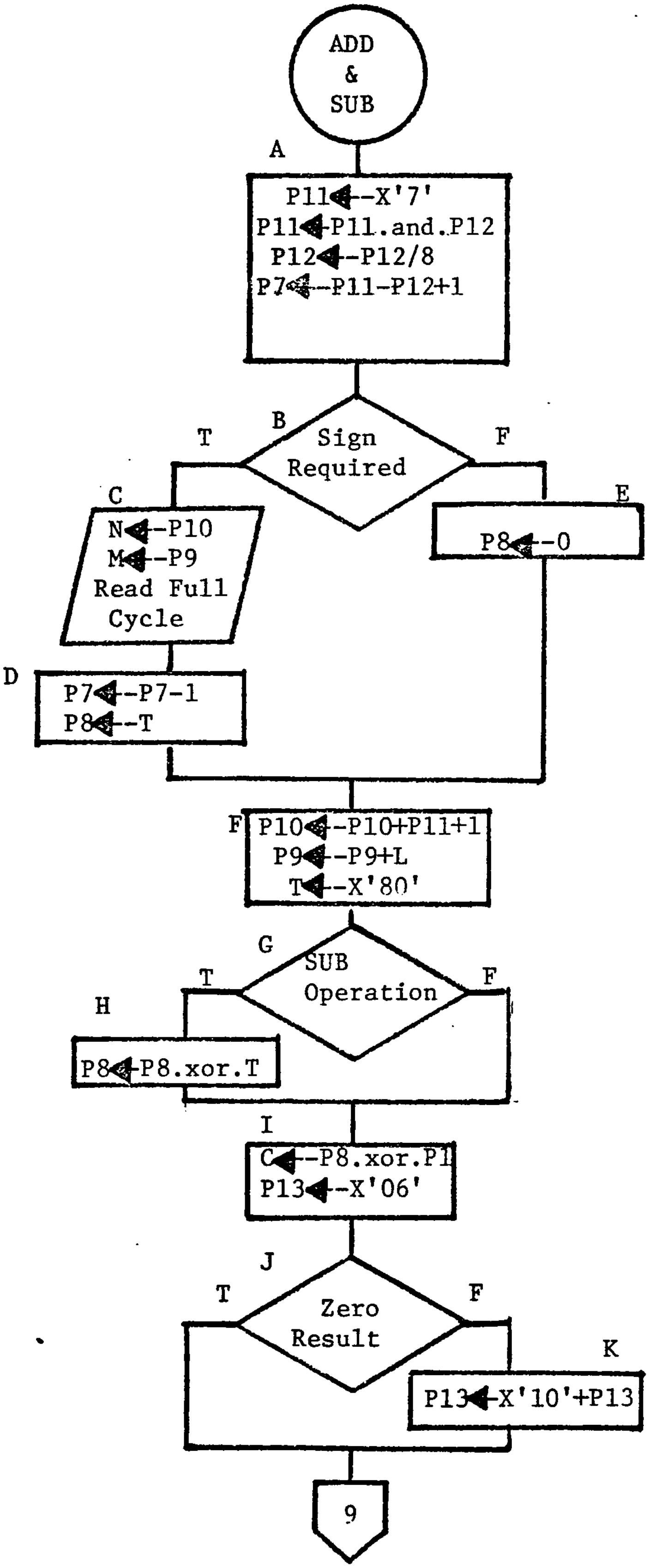

ADDRESS
Operand Address Routine
Call Index Supervisor, Page in Index
True
Indexing Required
False
U ◀- X'80'
T
Address Sign=Index Sign
F
U ◀- X'90'
P10,S◀P10+S10
P9,S◀P9+S9+L
True
Were the Signs Same
False
T
Link=1 (Overflow)
F
ABORT (bad address)
F
Link=1 (-ve result)
T
Complement P10 & P9, Flip Sign in P8.
T
Illegal Opcode
F
ABORT
1

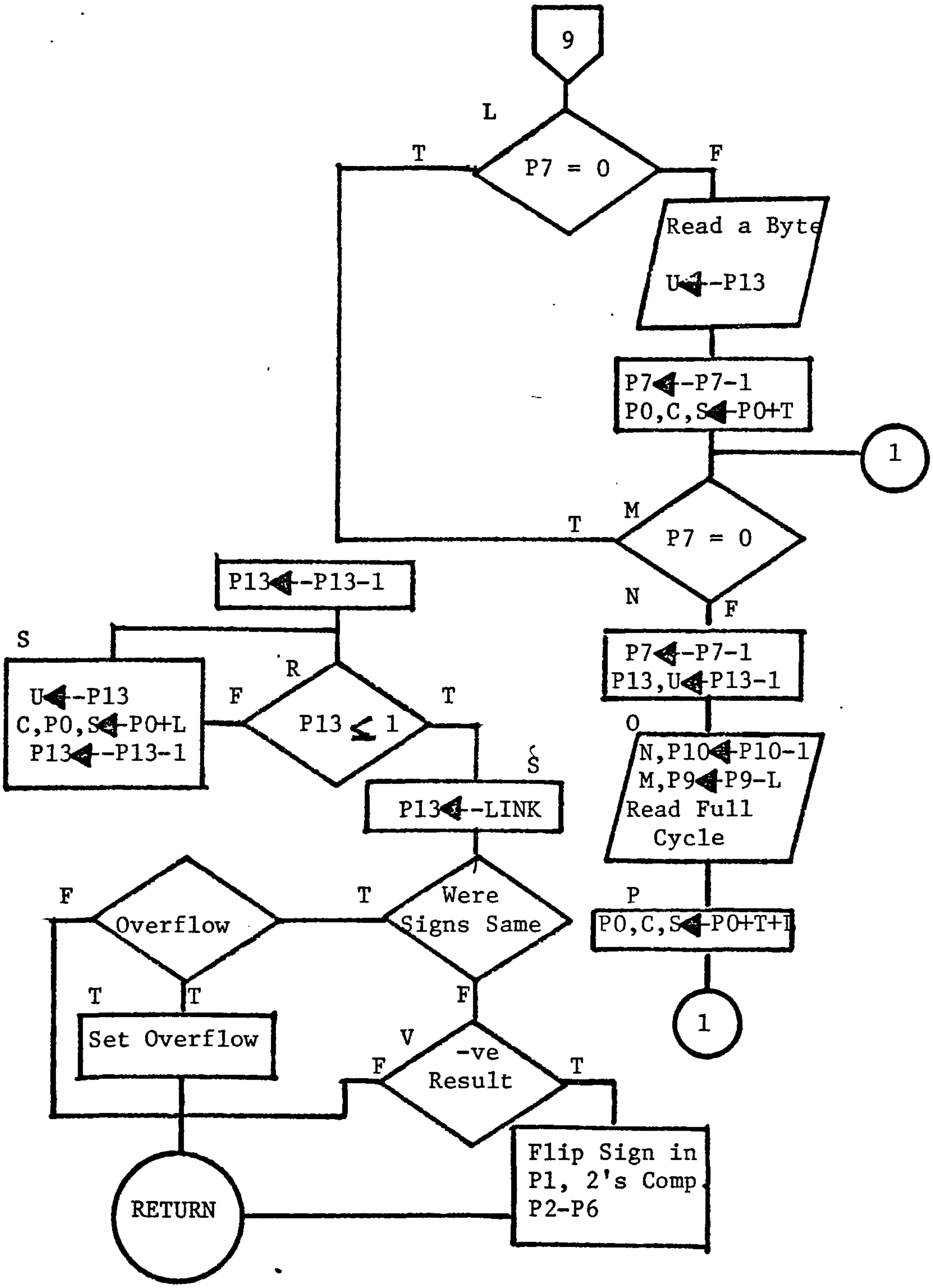
Instruction Decode Routine

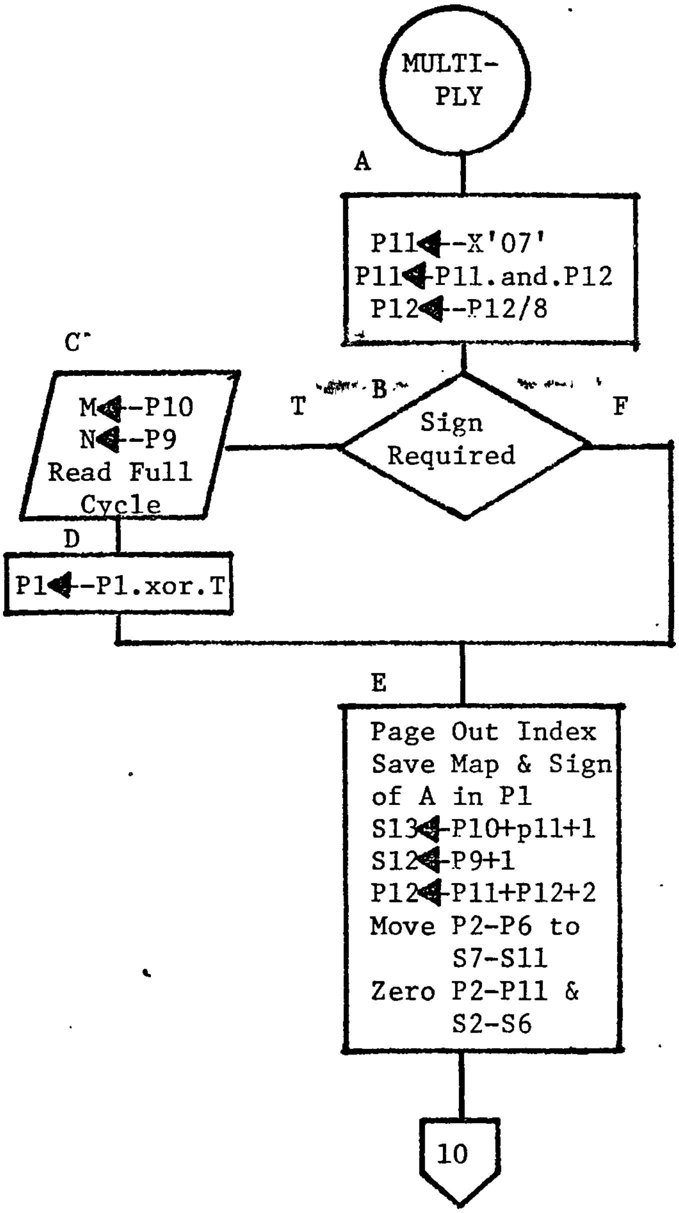


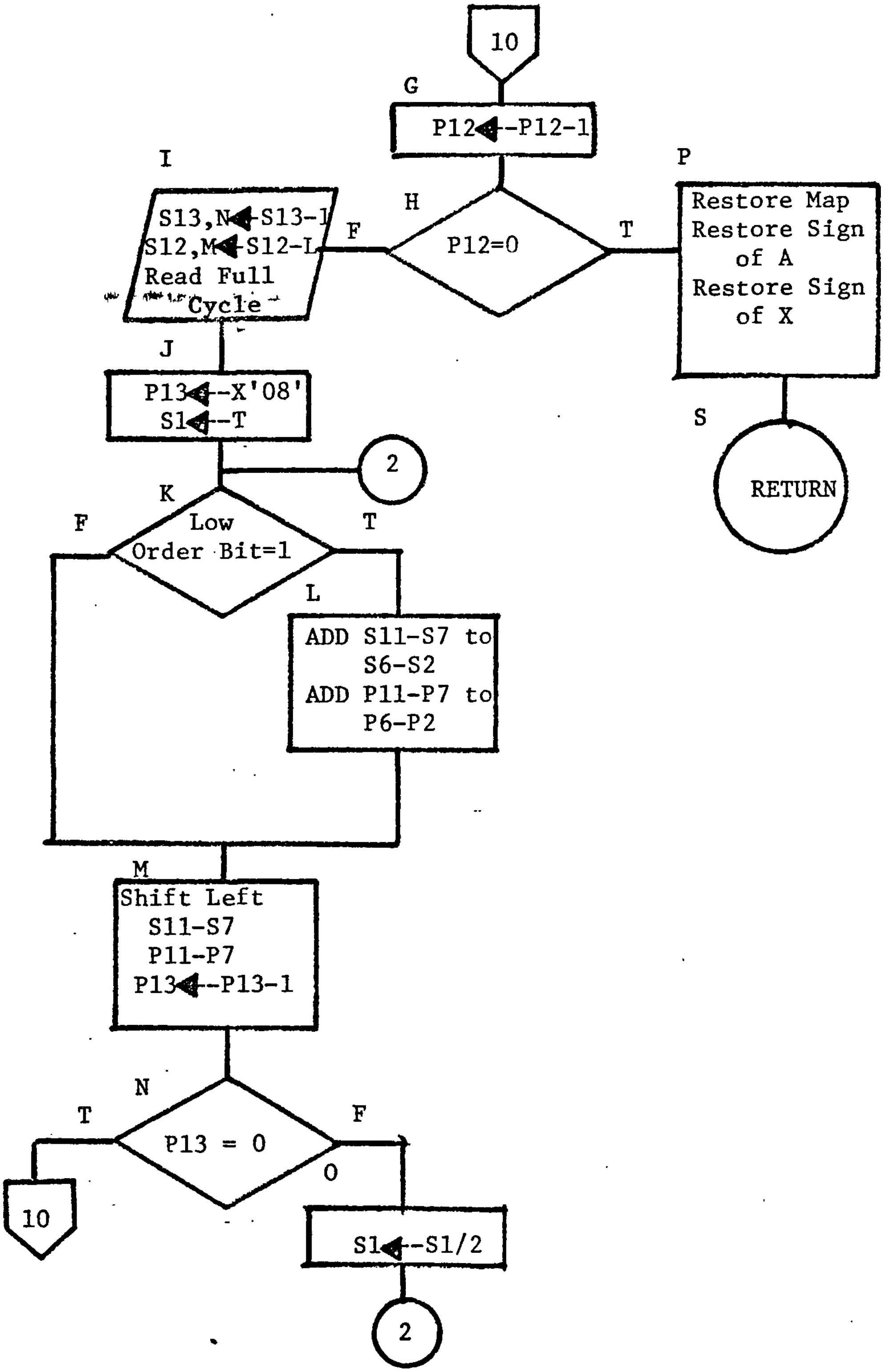

2
T ◀-- A'OP1'
L ◀-- T + P13
(NOP)
FETCH
(+,-)
ADD
MUL
DIV
NCH
SHIFT
MOVE
T ◀-- A'DC5'
L ◀-- T + P12
NUM
CHAR
(HALT)
STEP

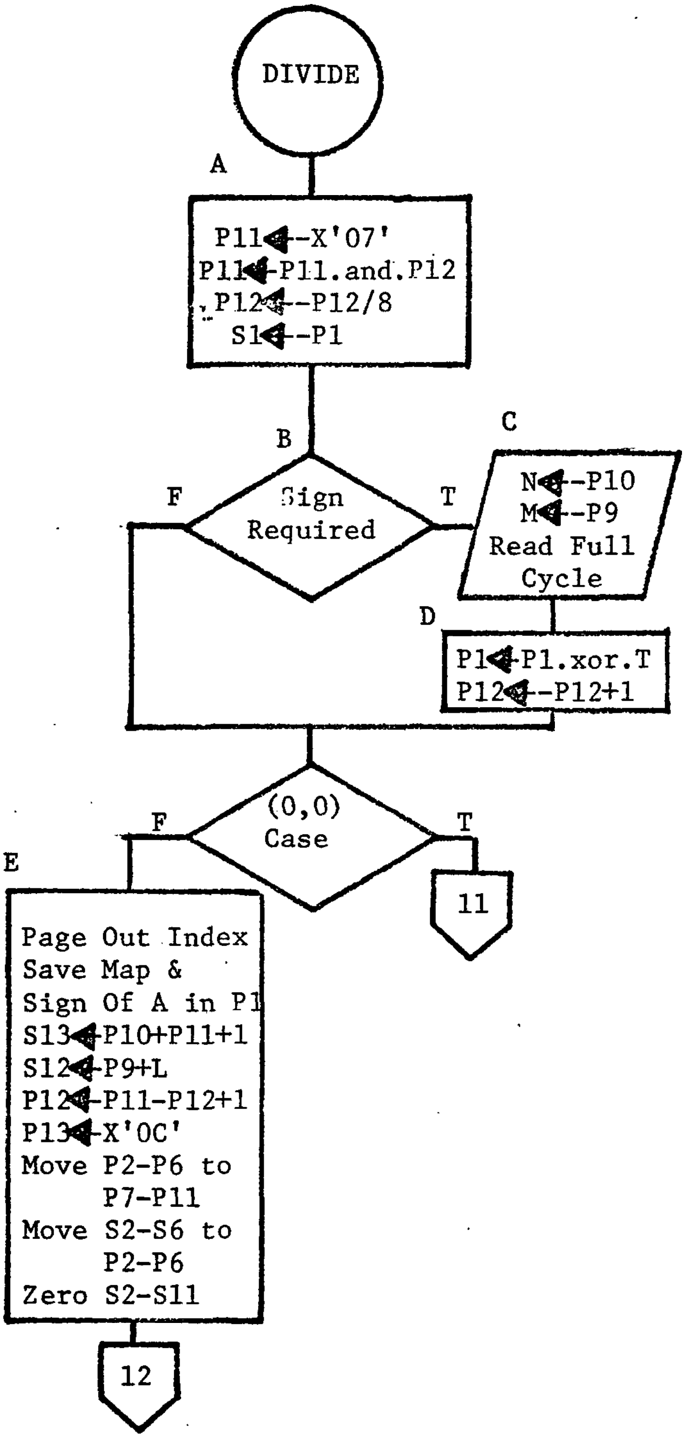
II. Instruction Repertoire

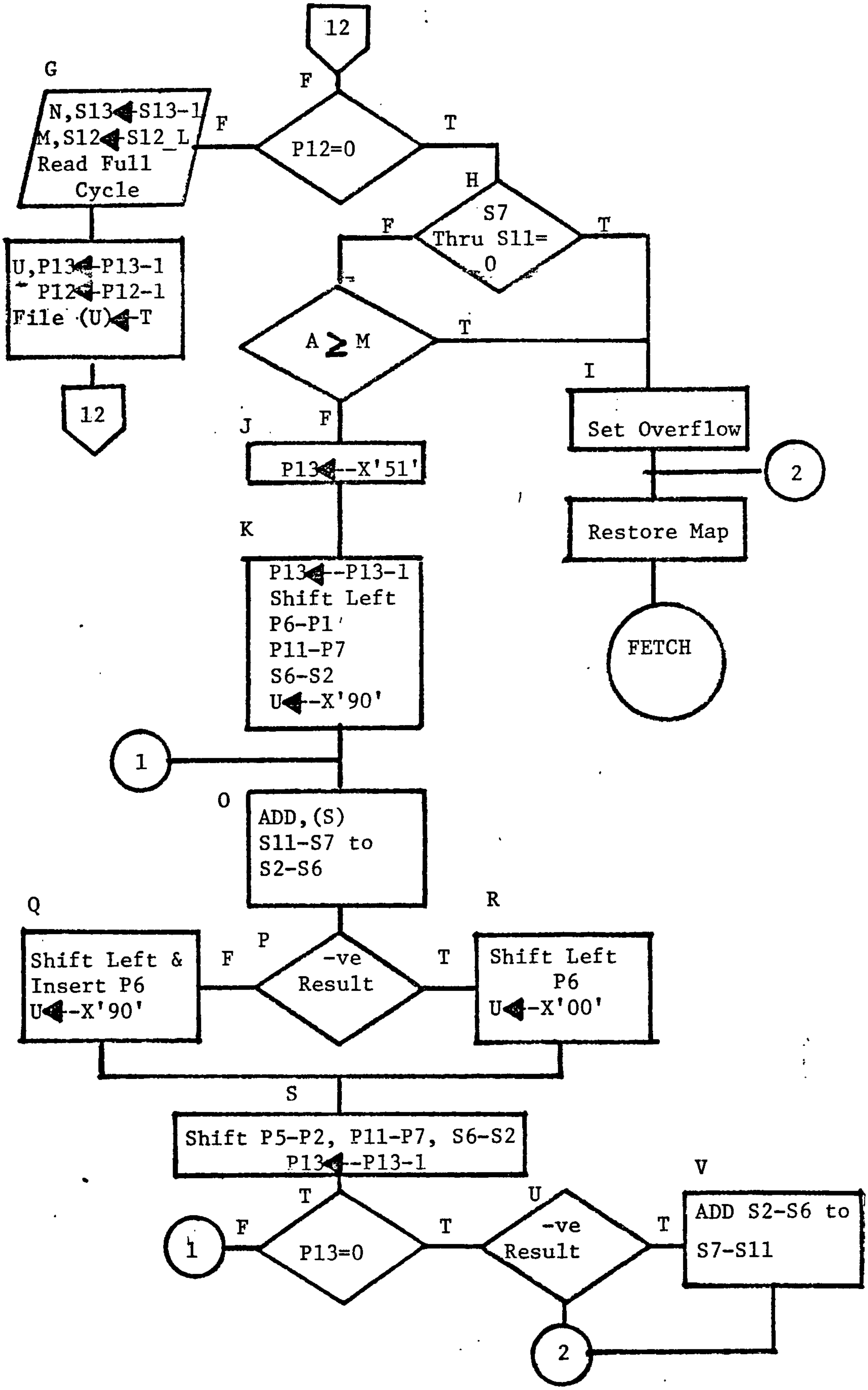

ADD
&
SUB
A
P11←-X'7'
P11←P11.and.P12
P12←-P12/8
P7←-P11-P12+1
B
T
Sign
Required
F
C
N←-P10
M←-P9
Read Full
Cycle
E
P8←-0
D
P7←-P7-1
P8←-T
F
P10←-P10+P11+1
P9←-P9+L
T←-X'80'
G
T
SUB
Operation
F
H
P8←P8.xor.T
I
C←-P8.xor.P1
P13←-X'06'
J
T
Zero
Result
F
K
P13←X'10'+P13
9

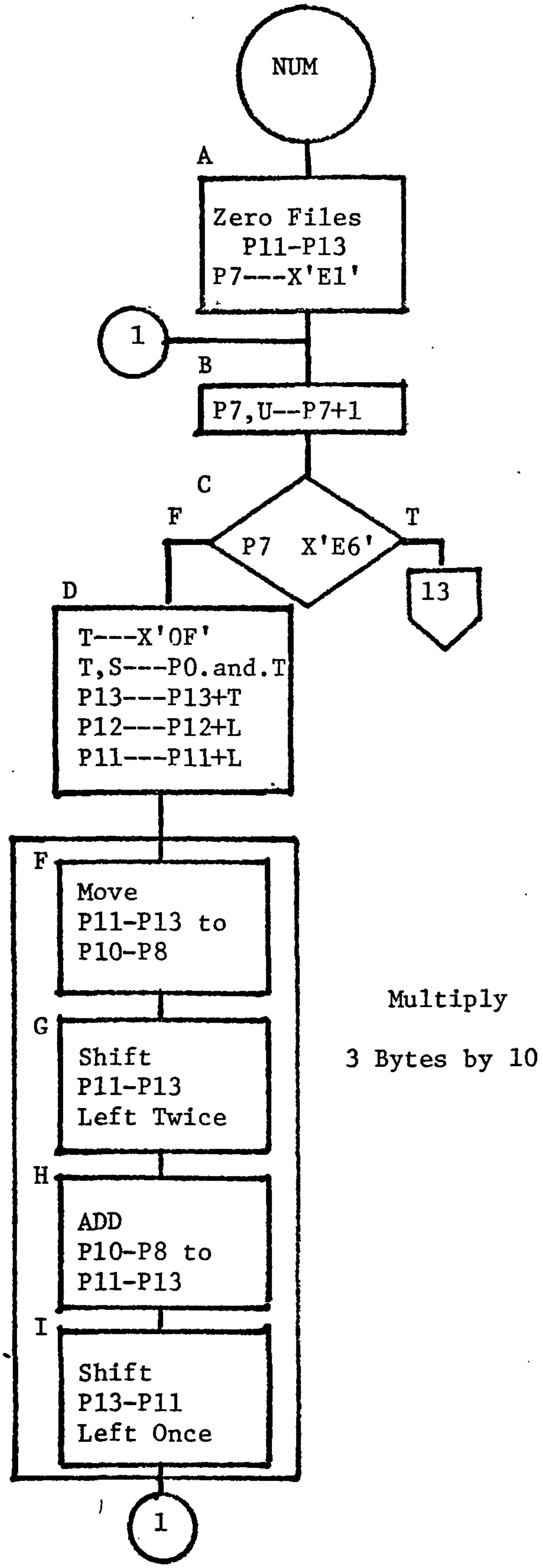

9
L
P7 = 0
T
F
Read a Byte
U←--P13
P7←-P7-1
P0,C,S←P0+T
1
M
P7 = 0
T
N
F
P7←-P7-1
P13,U←P13-1
O
N,P10←P10-1
M,P9←P9-L
Read Full
Cycle
P
P0,C,S←P0+T+1
1
P13←-P13-1
S
R
U←-P13
C,P0,S←P0+L
P13←-P13-1
F
P13 ≤ 1
T
§
P13←-LINK
F
Overflow
T
Were
Signs Same
T
T
F
Set Overflow
V
F
-ve
Result
T
Flip Sign in
P1, 2's Comp.
P2-P6
RETURN

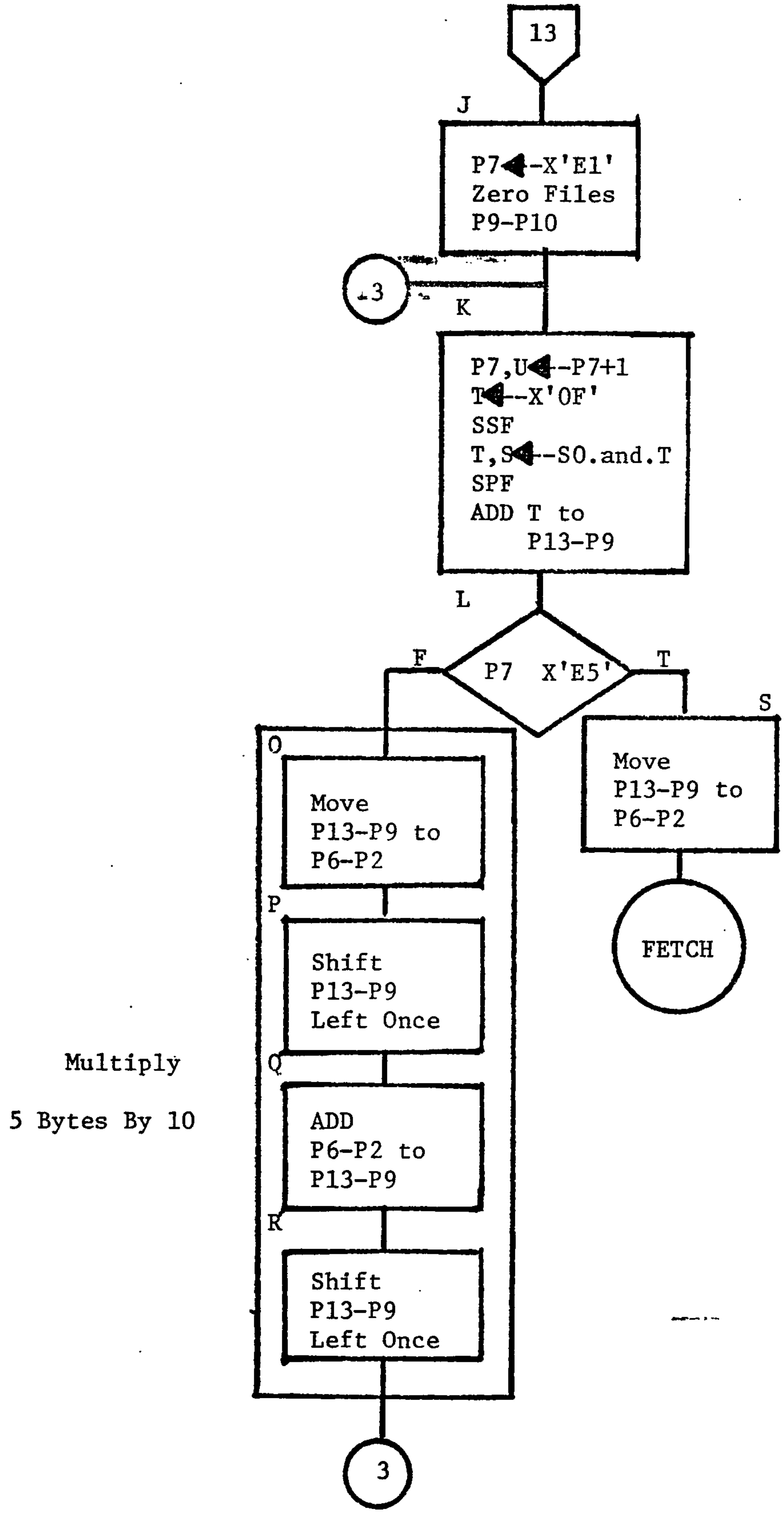

MULTI-
PLY
A
P11◀--X'07'
P11◀-P11.and.P12
P12◀--P12/8
C
M◀--P10
N◀--P9
Read Full
Cycle
T
B
Sign
Required
F
D
P1◀--P1.xor.T
E
Page Out Index
Save Map & Sign
of A in P1
S13◀-P10+p11+1
S12◀-P9+1
P12◀-P11+P12+2
Move P2-P6 to
S7-S11
Zero P2-P11 &
S2-S6
10

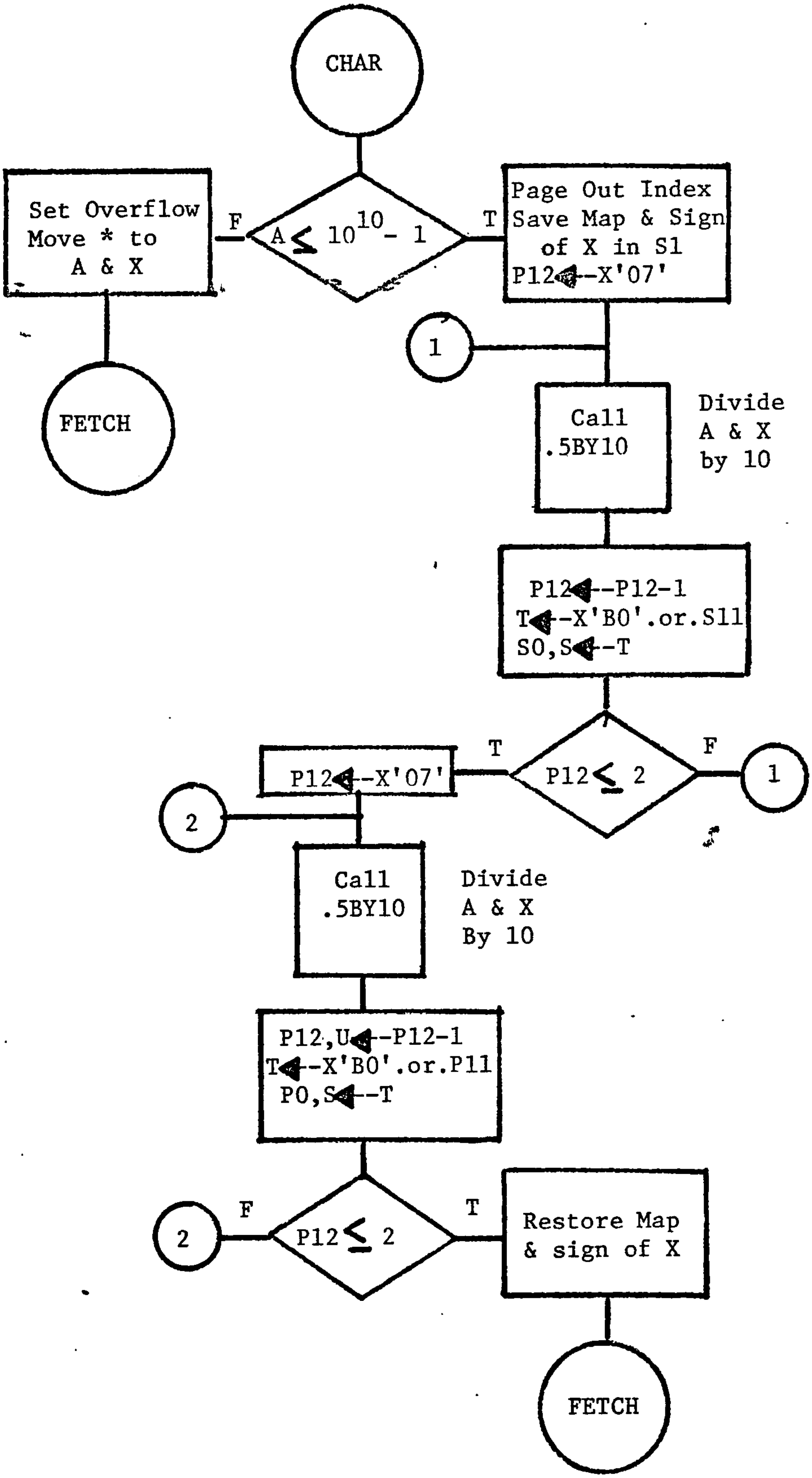

10
G
P12←P12-1
H
P12=0
T
P
Restore Map
Restore Sign of A
Restore Sign of X
S
RETURN
F
I
S13,N←S13-1
S12,M←S12-L
Read Full Cycle
J
P13←X'08'
S1←T
2
K
Low Order Bit=1
F
T
L
ADD S11-S7 to S6-S2
ADD P11-P7 to P6-P2
M
Shift Left
S11-S7
P11-P7
P13←P13-1
N
P13 = 0
T
10
F
O
S1←S1/2
2


DIVIDE
A
P11←-X'07'
P11←-P11.and.P12
P12←-P12/8
S1←-P1
B
Sign Required
F
T
C
N←-P10
M←-P9
Read Full Cycle
D
P1←-P1.xor.T
P12←-P12+1
(0,0) Case
F
T
11
E
Page Out Index
Save Map &
Sign Of A in P1
S13←-P10+P11+1
S12←-P9+L
P12←-P11-P12+1
P13←-X'0C'
Move P2-P6 to P7-P11
Move S2-S6 to P2-P6
Zero S2-S11
12


12
G
F
N,S13←S13-1
M,S12←S12_L
Read Full
Cycle
F
P12=0
T
H
S7
Thru S11=
0
F
T
U,P13←P13-1
P12←P12-1
File (U)←T
A ≥ M
T
12
I
J
F
Set Overflow
P13←-X'51'
2
K
Restore Map
P13←-P13-1
Shift Left
P6-P1
P11-P7
S6-S2
U←-X'90'
FETCH
1
O
ADD,(S)
S11-S7 to
S2-S6
Q
R
P
Shift Left &
Insert P6
U←-X'90'
F
-ve
Result
T
Shift Left
P6
U←-X'00'
S
Shift P5-P2, P11-P7, S6-S2
P13←-P13-1
V
T
U
1
F
P13=0
T
-ve
Result
T
ADD S2-S6 to
S7-S11
2

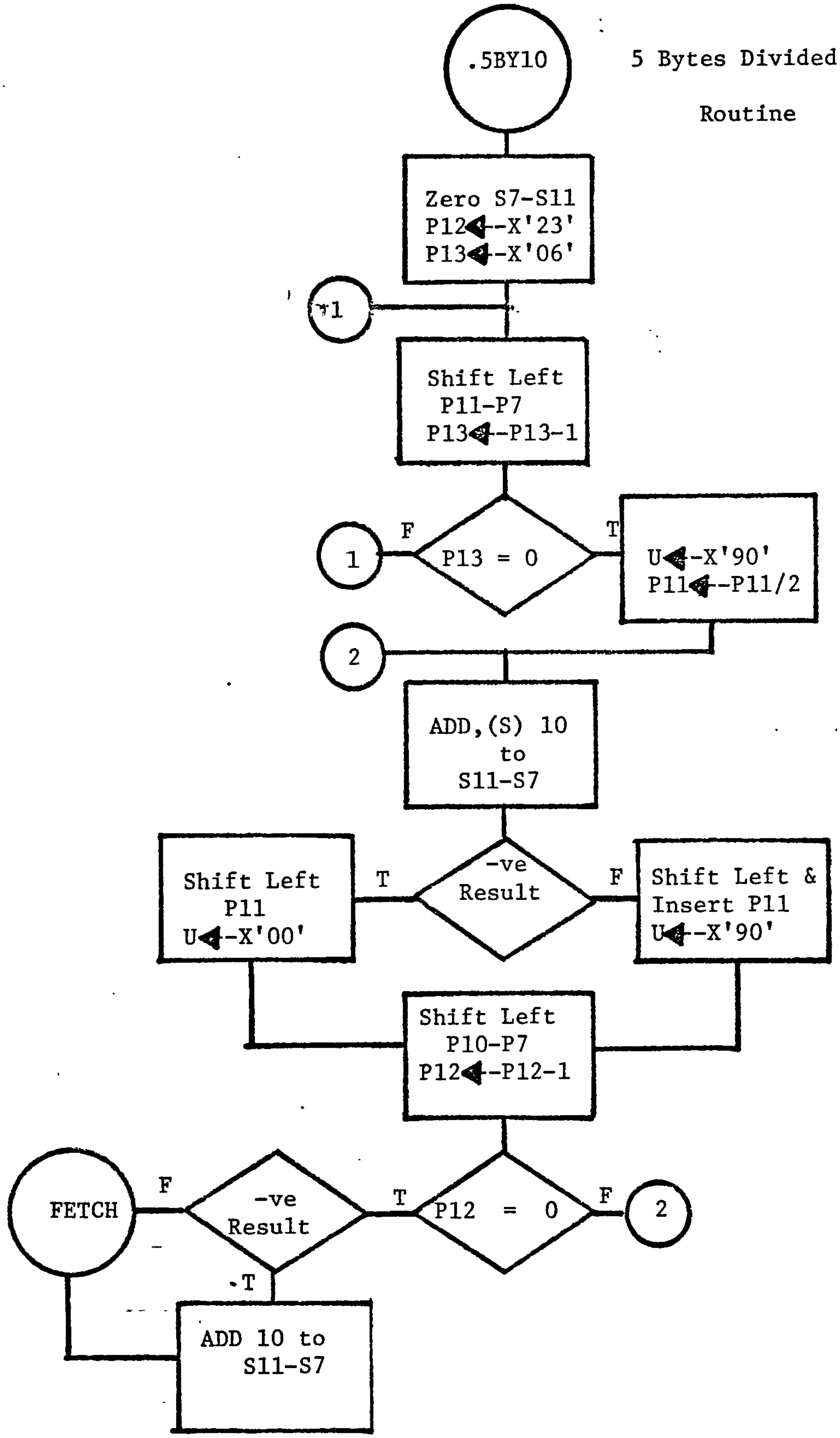

NUM
A
Zero Files
P11-P13
P7---X'E1'
1
B
P7,U--P7+1
C
F
T
P7 X'E6'
13
D
T---X'OF'
T,S---P0.and.T
P13---P13+T
P12---P12+L
P11---P11+L
F
Move
P11-P13 to
P10-P8
G
Shift
P11-P13
Left Twice
H
ADD
P10-P8 to
P11-P13
I
Shift
P13-P11
Left Once
1
Multiply
3 Bytes by 10

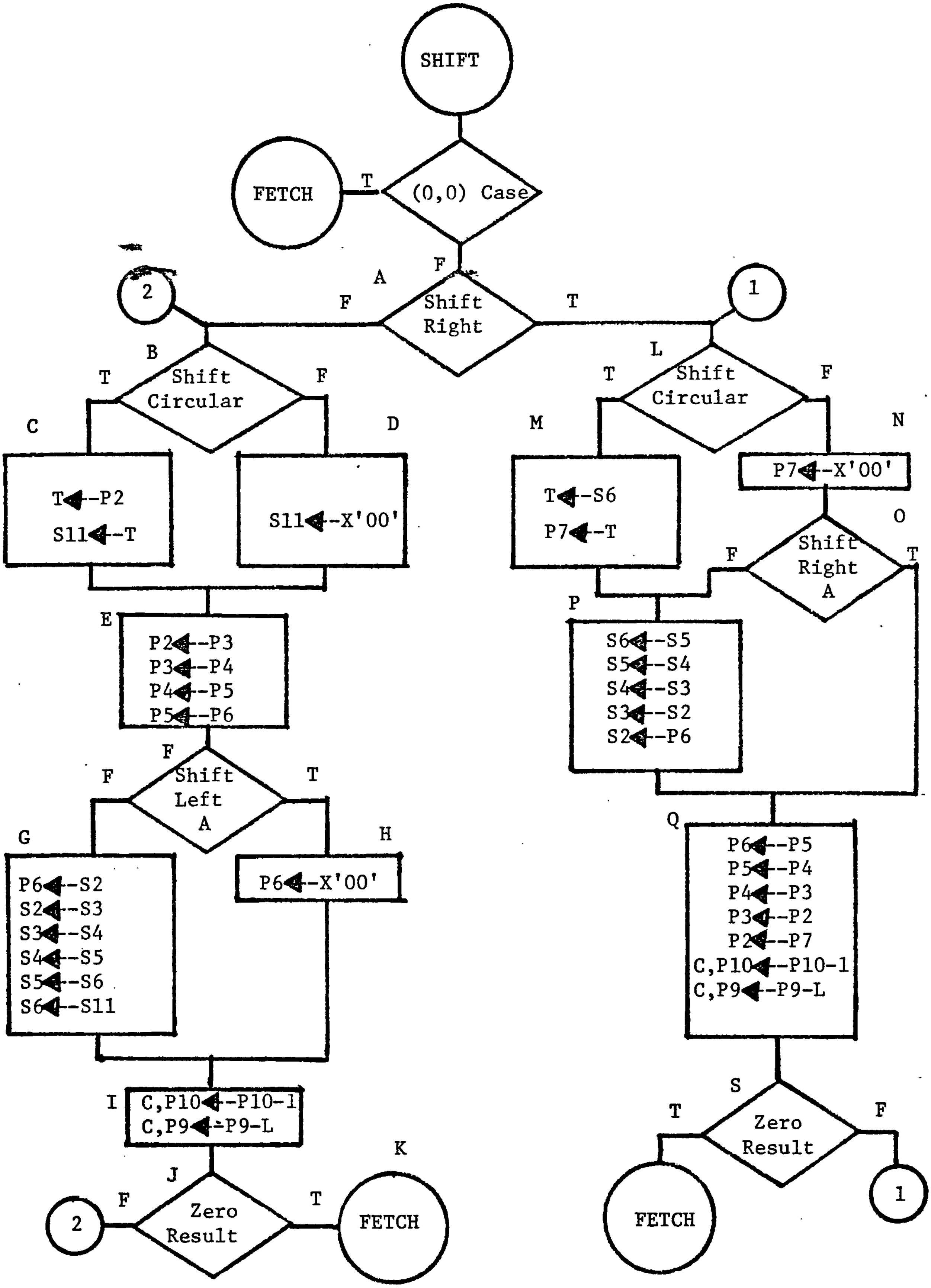

13
J
P7←-X'E1'
Zero Files
P9-P10
3
K
P7,U←-P7+1
T←-X'0F'
SSF
T,S←-S0.and.T
SPF
ADD T to
P13-P9
L
F
P7 X'E5'
T
S
Move
P13-P9 to
P6-P2
FETCH
O
Move
P13-P9 to
P6-P2
P
Shift
P13-P9
Left Once
Multiply
5 Bytes By 10
Q
ADD
P6-P2 to
P13-P9
R
Shift
P13-P9
Left Once
3

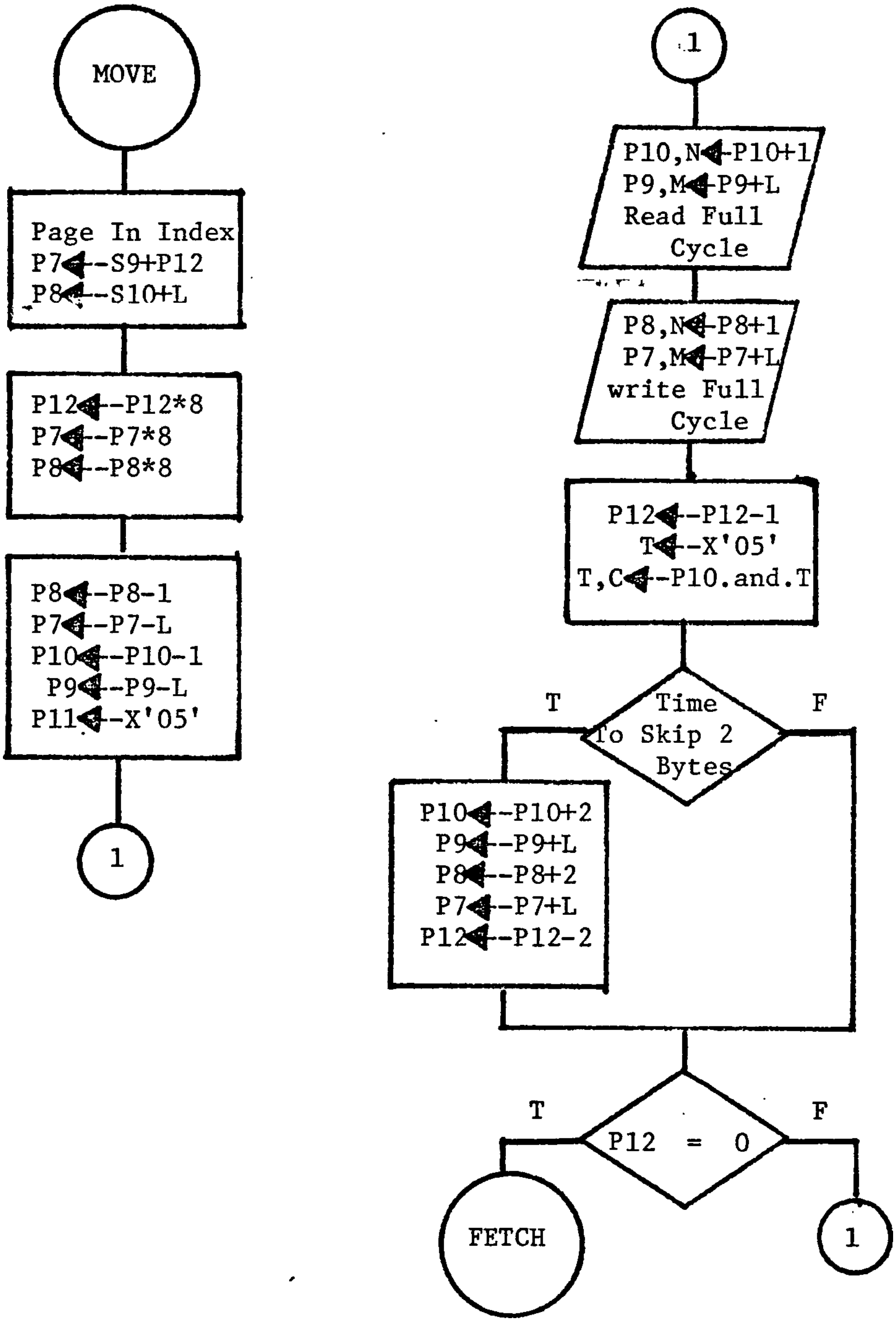

CHAR
F
$A \leq 10^{10} - 1$
T
Set Overflow
Move * to
A & X
FETCH
Page Out Index
Save Map & Sign
of X in S1
P12←-X'07'
1
Call
.5BY10
Divide
A & X
by 10
P12←-P12-1
T←-X'B0'.or.S11
S0,S←-T
T
F
P12←-X'07'
P12 ≤ 2
1
2
Call
.5BY10
Divide
A & X
By 10
P12,U←-P12-1
T←-X'B0'.or.P11
P0,S←-T
2
F
P12 ≤ 2
T
Restore Map
& sign of X
FETCH

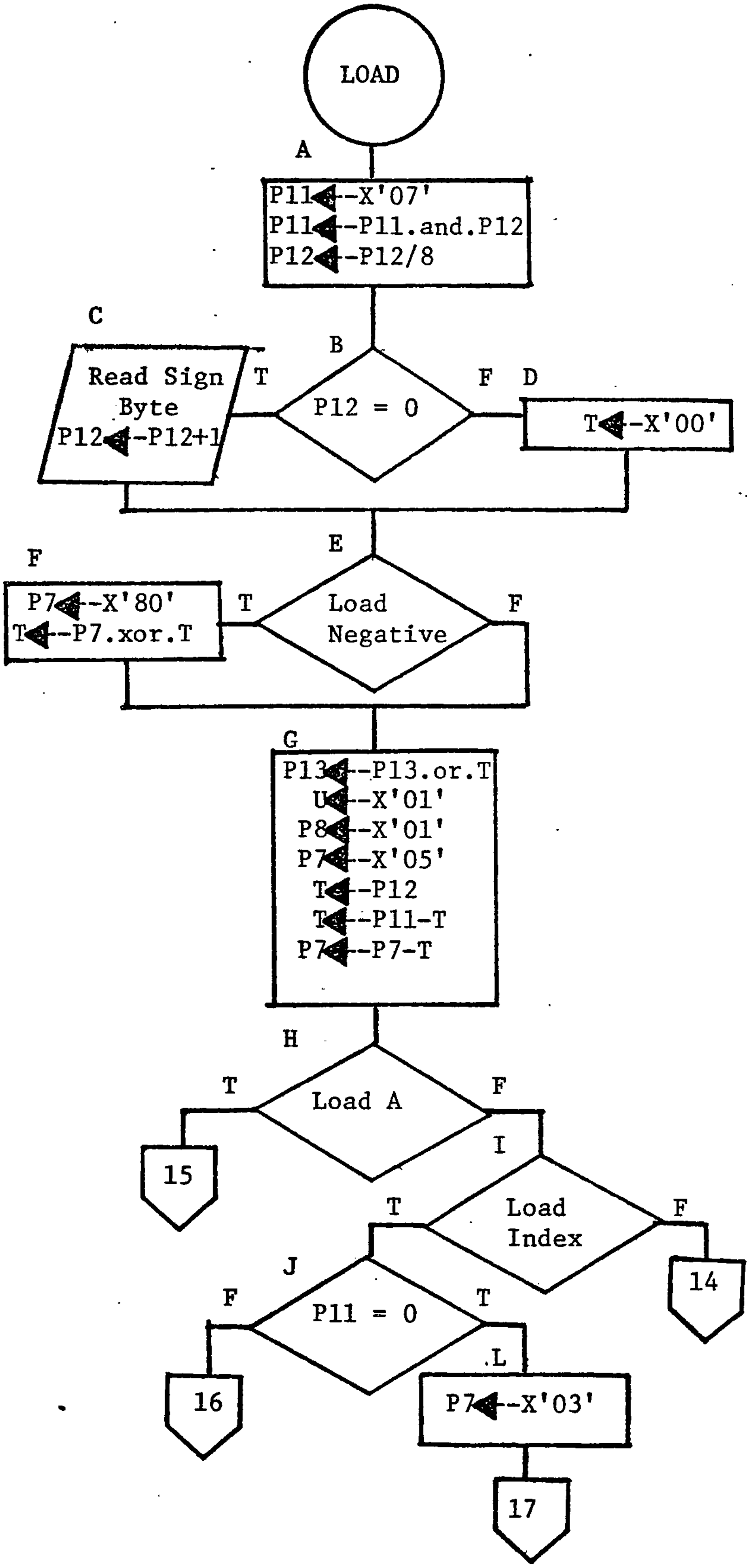
5 Bytes Divided By 10

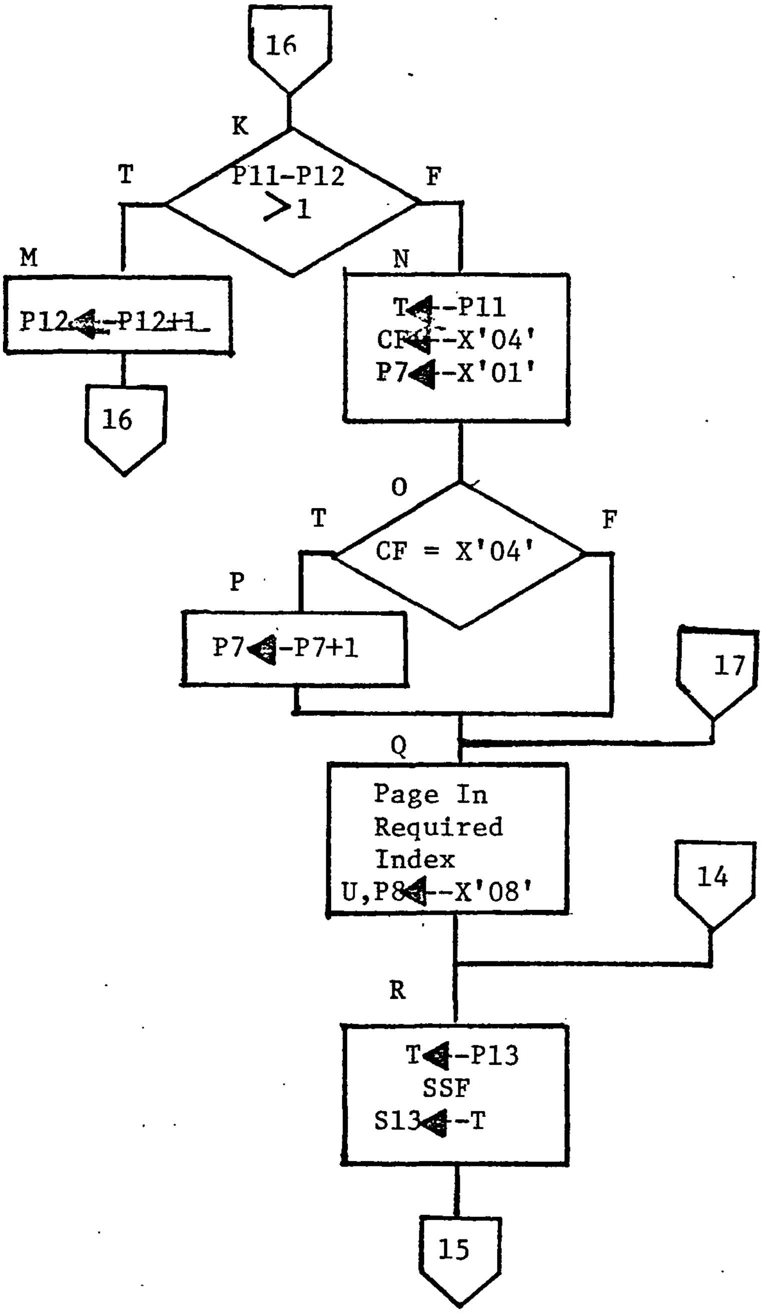
Routine

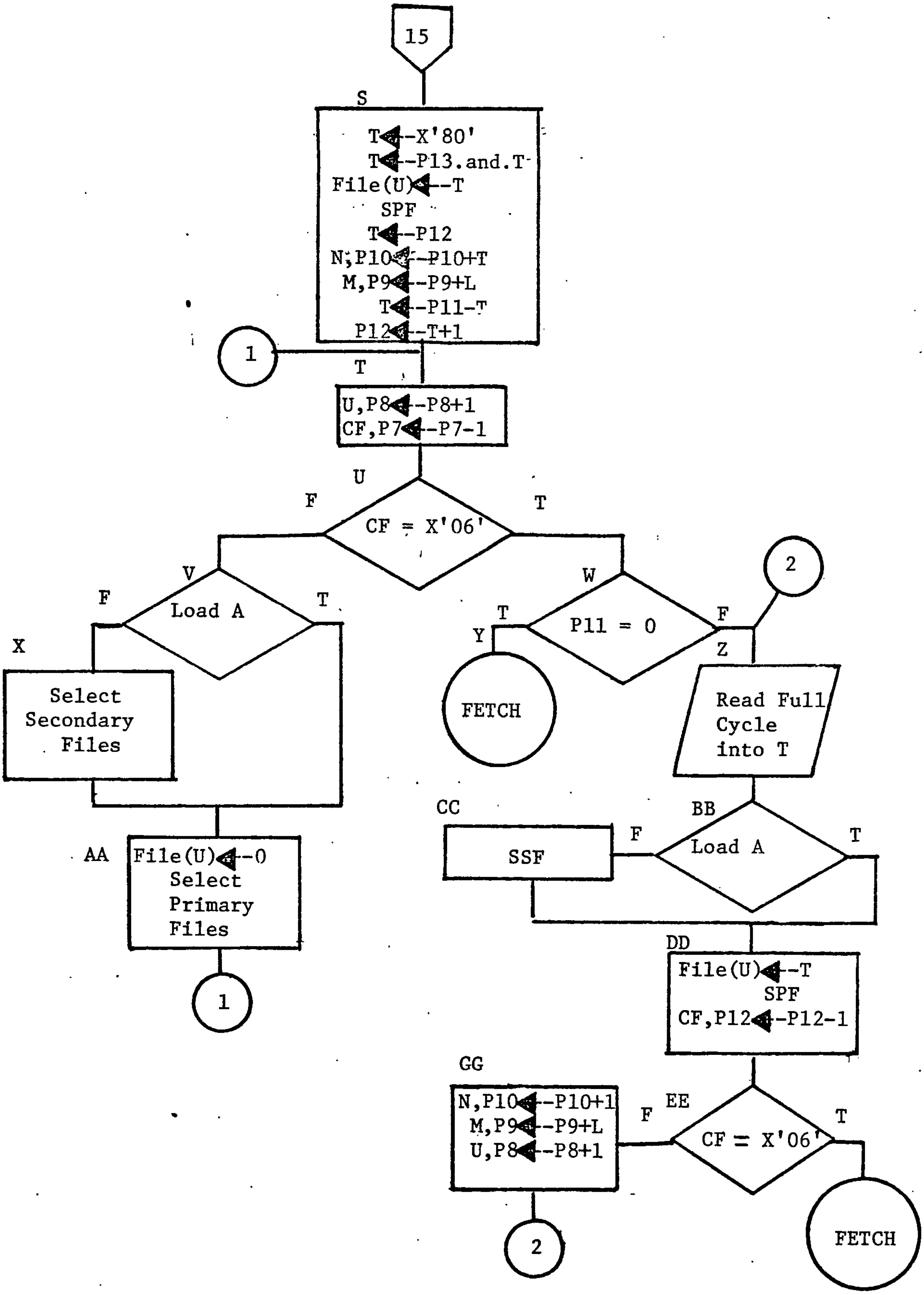


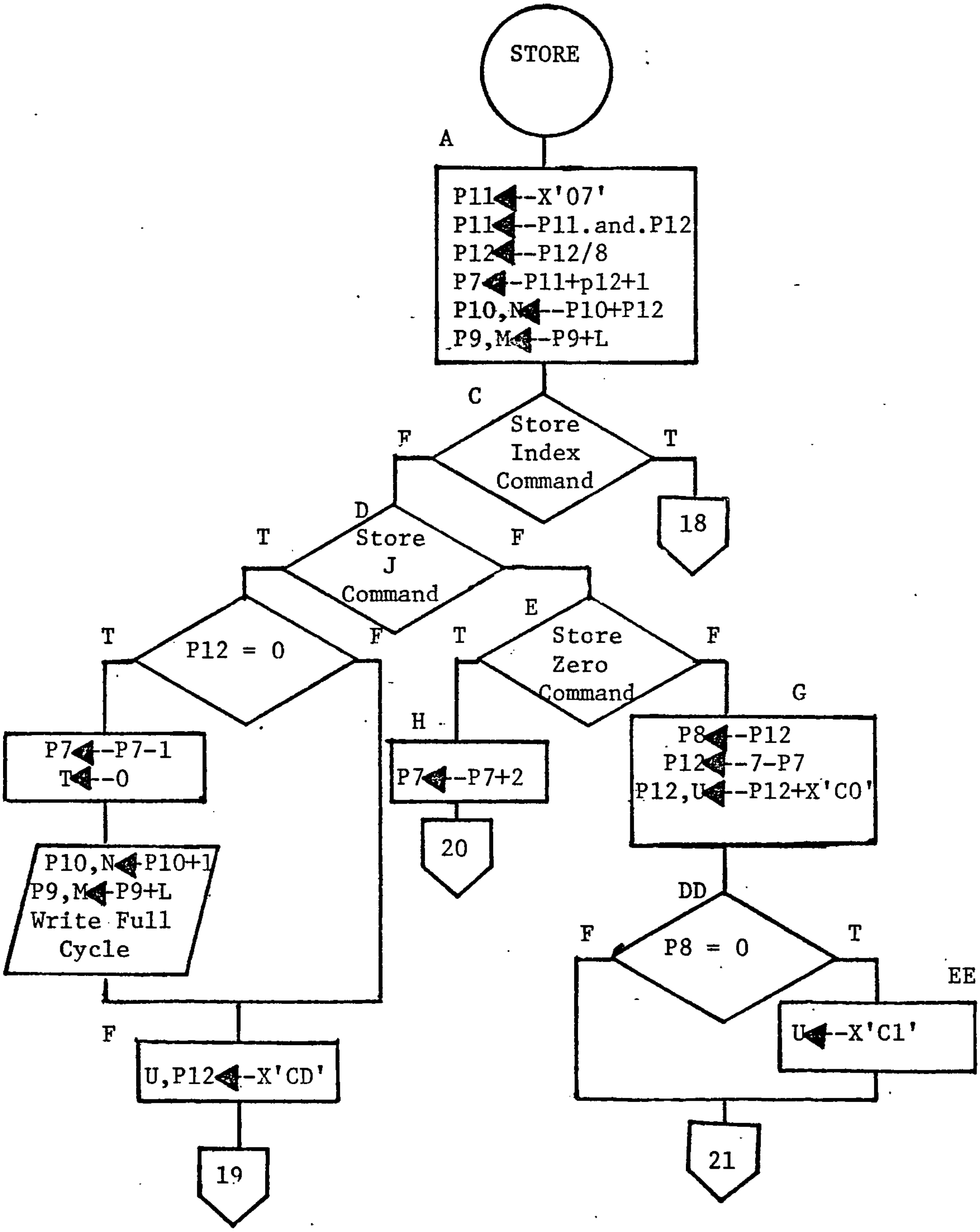

SHIFT
FETCH
T
(0,0) Case
F
A
Shift Right
F
T
2
1
B
Shift Circular
T
F
C
T◀--P2
S11◀--T
D
S11◀--X'00'
E
P2◀--P3
P3◀--P4
P4◀--P5
P5◀--P6
F
Shift Left A
F
T
G
P6◀--S2
S2◀--S3
S3◀--S4
S4◀--S5
S5◀--S6
S6◀--S11
H
P6◀--X'00'
I
C,P10◀--P10-1
C,P9◀--P9-L
J
Zero Result
F
2
T
K
FETCH
L
Shift Circular
T
F
M
T◀--S6
P7◀--T
N
P7◀--X'00'
O
Shift Right A
F
T
P
S6◀--S5
S5◀--S4
S4◀--S3
S3◀--S2
S2◀--P6
Q
P6◀--P5
P5◀--P4
P4◀--P3
P3◀--P2
P2◀--P7
C,P10◀--P10-1
C,P9◀--P9-L
S
Zero Result
T
F
FETCH
1

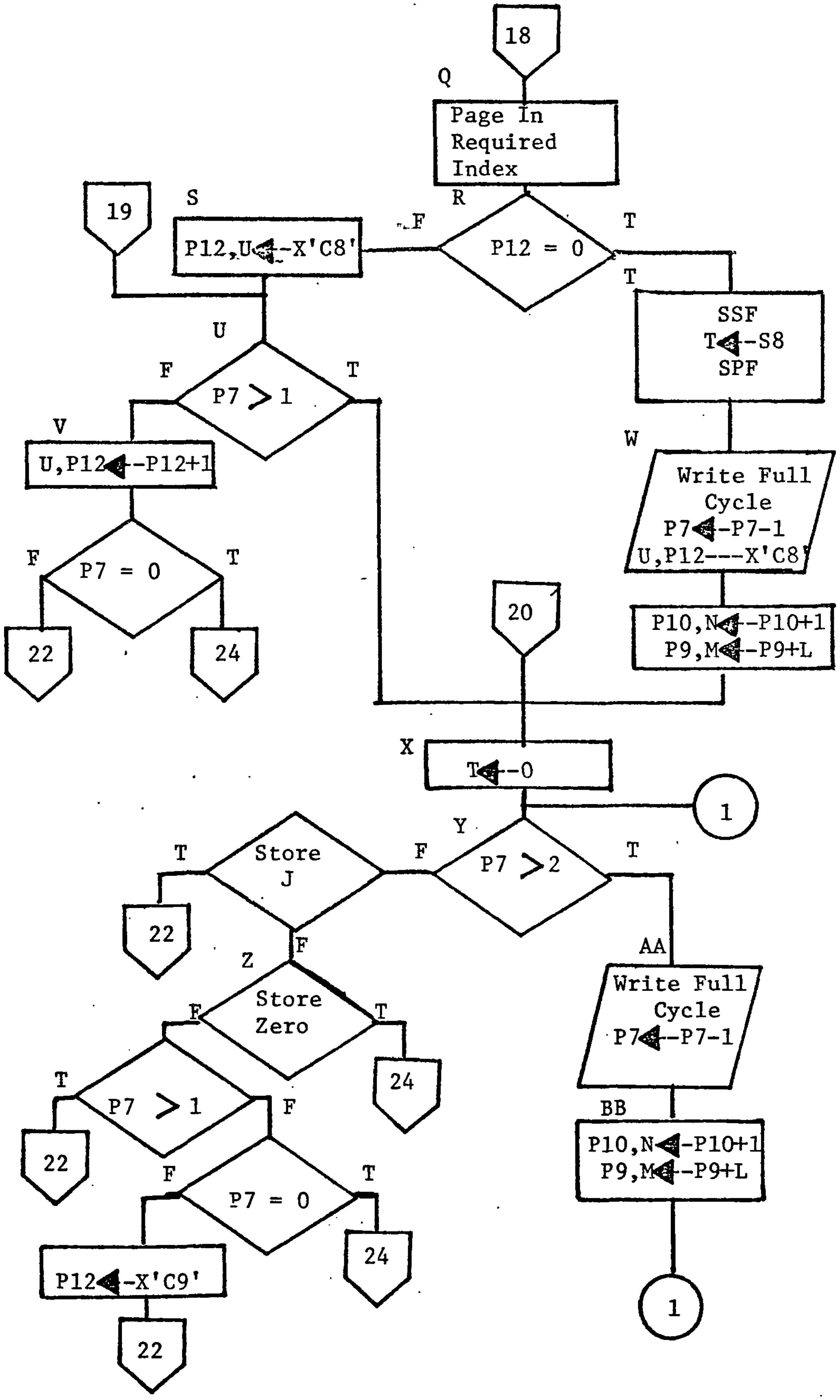

MOVE
Page In Index
P7◄--S9+P12
P8◄--S10+L
P12◄--P12*8
P7◄--P7*8
P8◄--P8*8
P8◄--P8-1
P7◄--P7-L
P10◄--P10-1
P9◄--P9-L
P11◄--X'05'
1
1
P10,N◄-P10+1
P9,M◄-P9+L
Read Full
Cycle
P8,N◄-P8+1
P7,M◄-P7+L
write Full
Cycle
P12◄--P12-1
T◄--X'05'
T,C◄-P10.and.T
T
Time
To Skip 2
Bytes
F
P10◄--P10+2
P9◄--P9+L
P8◄--P8+2
P7◄--P7+L
P12◄--P12-2
T
P12 = 0
F
FETCH
1

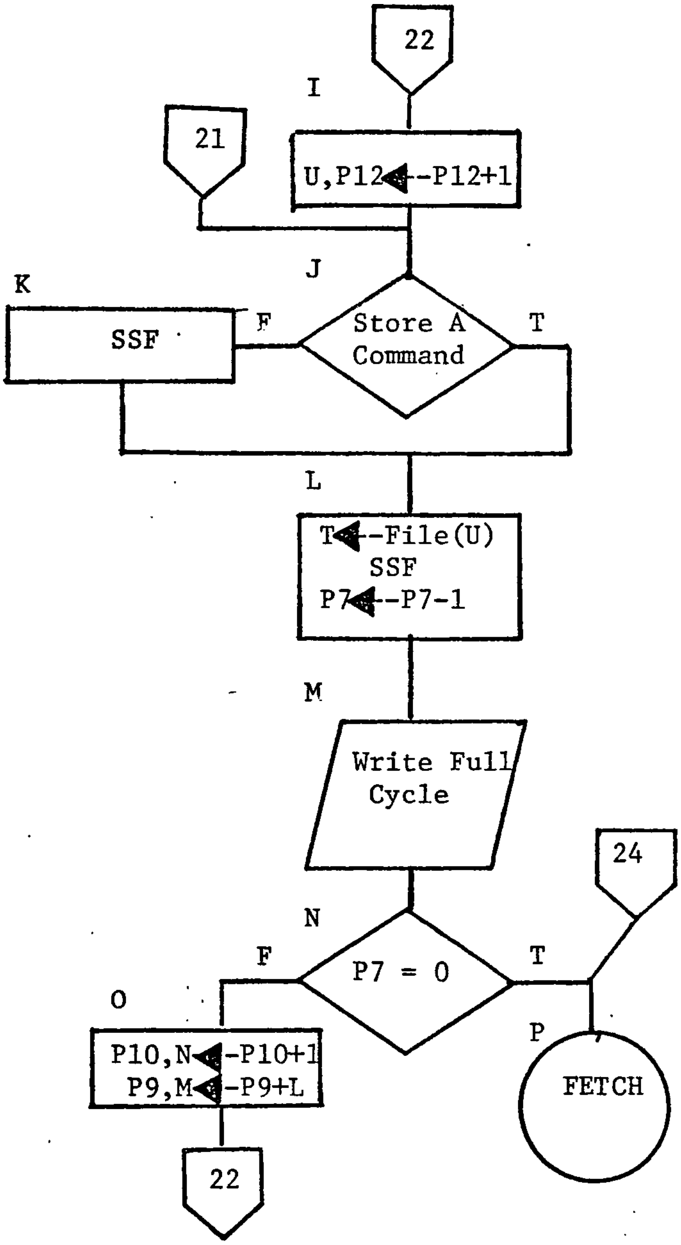

LOAD
A
P11◄--X'07'
P11◄--P11.and.P12
P12◄--P12/8
C
Read Sign Byte
P12◄--P12+1
T
B
P12 = 0
F
D
T◄--X'00'
E
F
P7◄--X'80'
T◄--P7.xor.T
T
Load Negative
F
G
P13◄--P13.or.T
U◄--X'01'
P8◄--X'01'
P7◄--X'05'
T◄--P12
T◄--P11-T
P7◄--P7-T
H
T
Load A
F
15
I
T
Load Index
F
14
J
F
P11 = 0
T
L
16
P7◄--X'03'
17

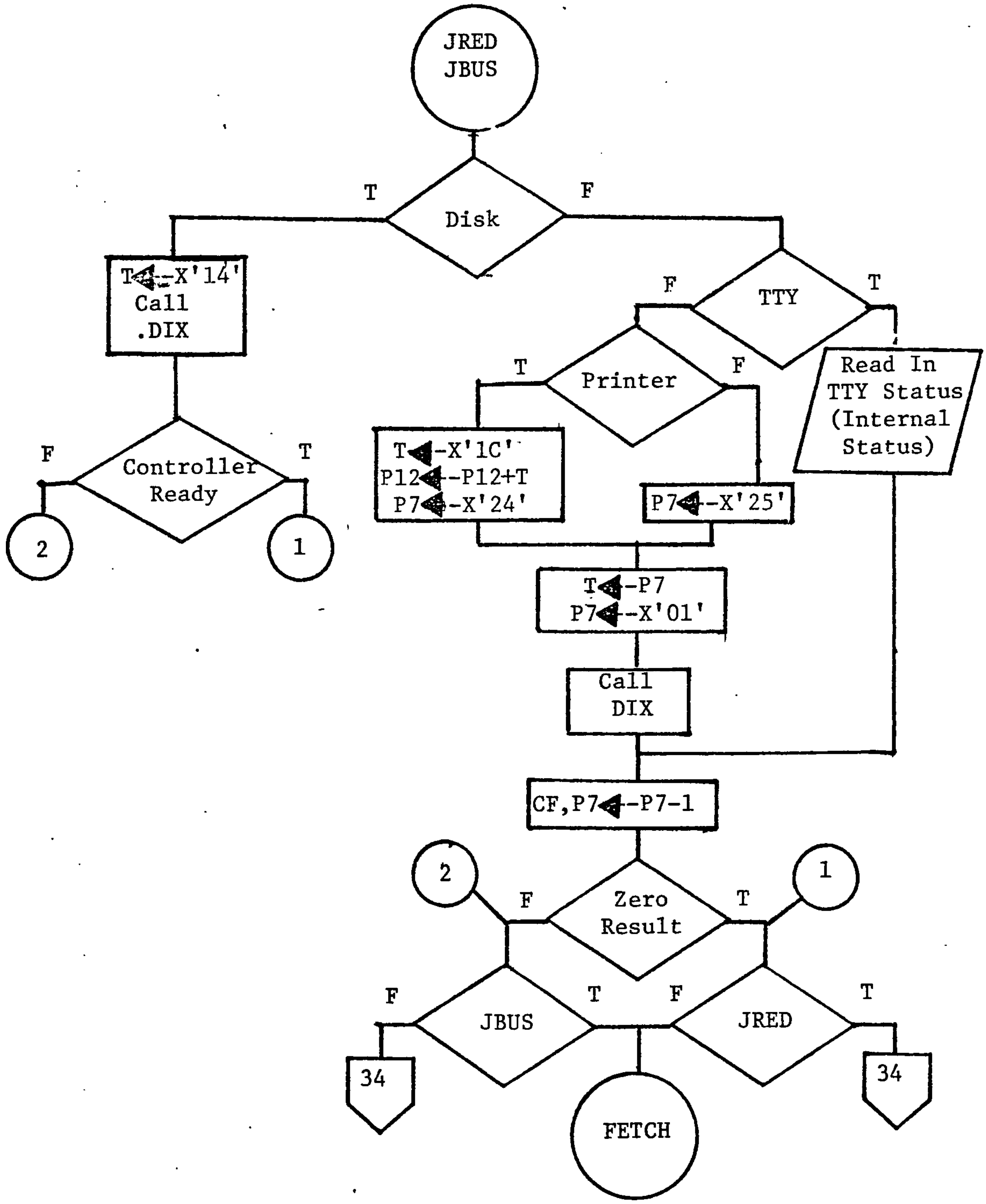

16
K
T
P11-P12
>1
F
M
P12◄-P12+1
16
N
T◄-P11
CF◄-X'04'
P7◄-X'01'
O
T
CF = X'04'
F
P
P7◄-P7+1
17
Q
Page In
Required
Index
U,P8◄-X'08'
14
R
T◄-P13
SSF
S13◄-T
15

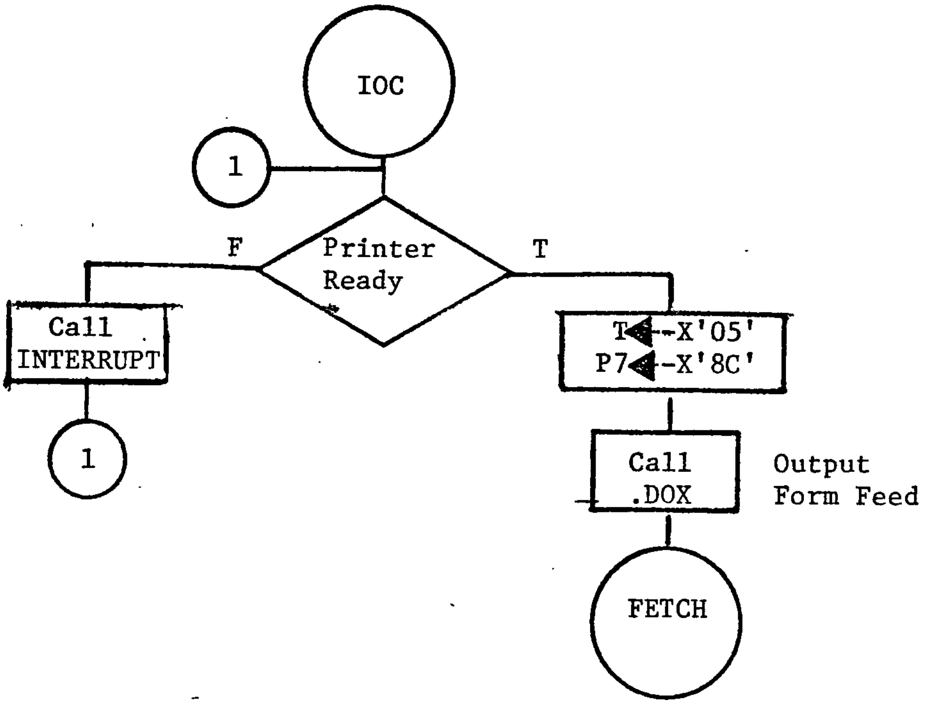

15
S
T←X'80'
T←P13.and.T
File(U)←T
SPF
T←P12
N,P10←P10+T
M,P9←P9+L
T←P11-T
P12←T+1
1
T
U,P8←P8+1
CF,P7←P7-1
U
F
CF = X'06'
T
V
F
Load A
T
W
2
T
Y
P11 = 0
F
Z
X
Select Secondary Files
FETCH
Read Full Cycle into T
CC
BB
F
Load A
T
SSF
AA
File(U)←0
Select Primary Files
DD
File(U)←T
SPF
CF,P12←P12-1
1
GG
N,P10←P10+1
M,P9←P9+L
U,P8←P8+1
F
EE
CF = X'06'
T
2
FETCH

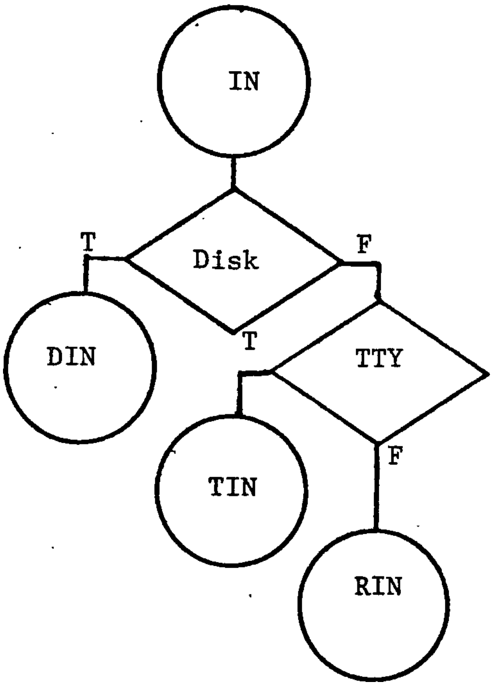

STORE
A
P11◀--X'07'
P11◀--P11.and.P12
P12◀--P12/8
P7◀--P11+p12+1
P10,N◀--P10+P12
P9,M◀--P9+L
C
Store Index Command
F
T
18
D
Store J Command
T
F
E
Store Zero Command
T
F
P12 = 0
T
F
P7◀--P7-1
T◀--0
P10,N◀-P10+1
P9,M◀-P9+L
Write Full Cycle
F
U,P12◀-X'CD'
19
H
P7◀--P7+2
20
G
P8◀--P12
P12◀--7-P7
P12,U◀--P12+X'C0'
DD
P8 = 0
F
T
EE
U◀--X'C1'
21

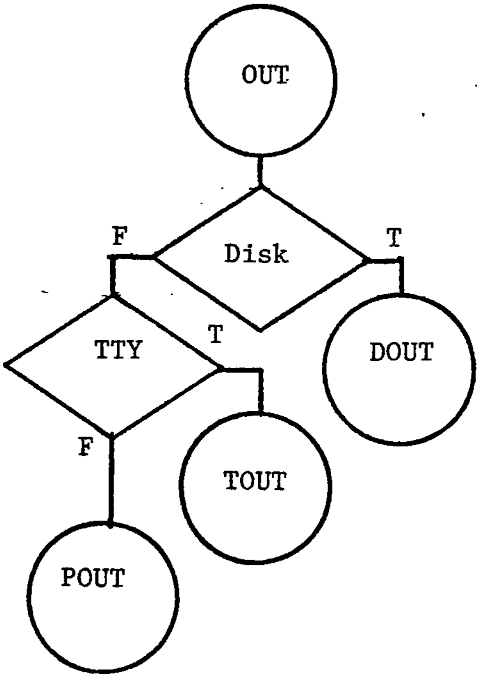

18
Q
Page In Required Index
R
P12 = 0
F
T
S
P12,U◀--X'C8'
19
T
SSF
T◀--S8
SPF
U
P7 > 1
F
T
V
U,P12◀--P12+1
P7 = 0
F
T
22
24
W
Write Full Cycle
P7◀--P7-1
U,P12---X'C8'
P10,N◀--P10+1
P9,M◀--P9+L
20
X
T◀--0
1
Y
P7 >2
F
T
Store J
T
22
F
Z
Store Zero
F
T
24
P7 > 1
T
F
22
P7 = 0
F
T
24
P12◀-X'C9'
22
AA
Write Full Cycle
P7◀--P7-1
BB
P10,N◀-P10+1
P9,M◀--P9+L
1

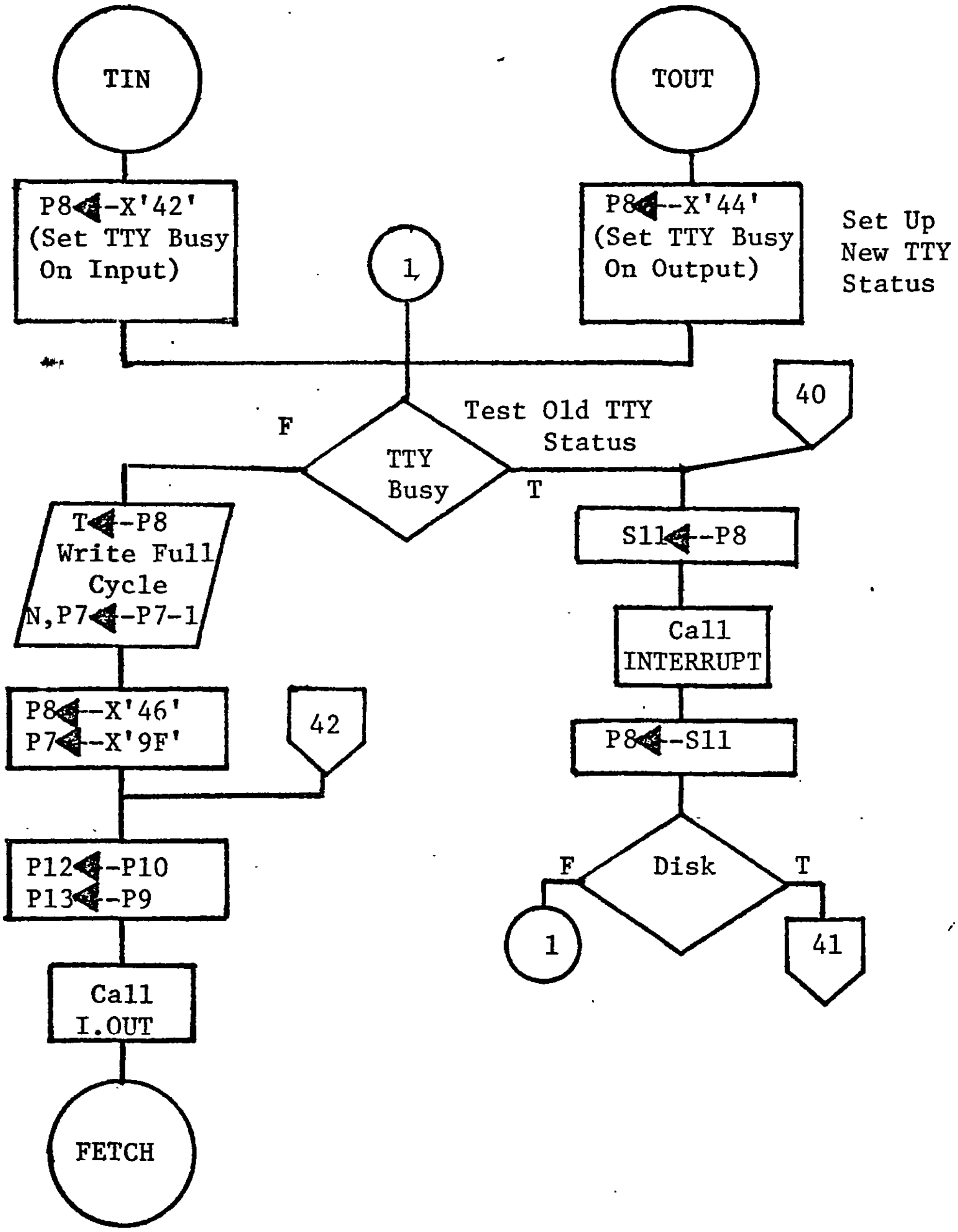

22
I
U,P12←-P12+1
21
J
K
SSF
F
Store A Command
T
L
T←-File(U)
SSF
P7←-P7-1
M
Write Full Cycle
24
N
F
P7 = 0
T
O
P
P10,N←-P10+1
P9,M←-P9+L
FETCH
22

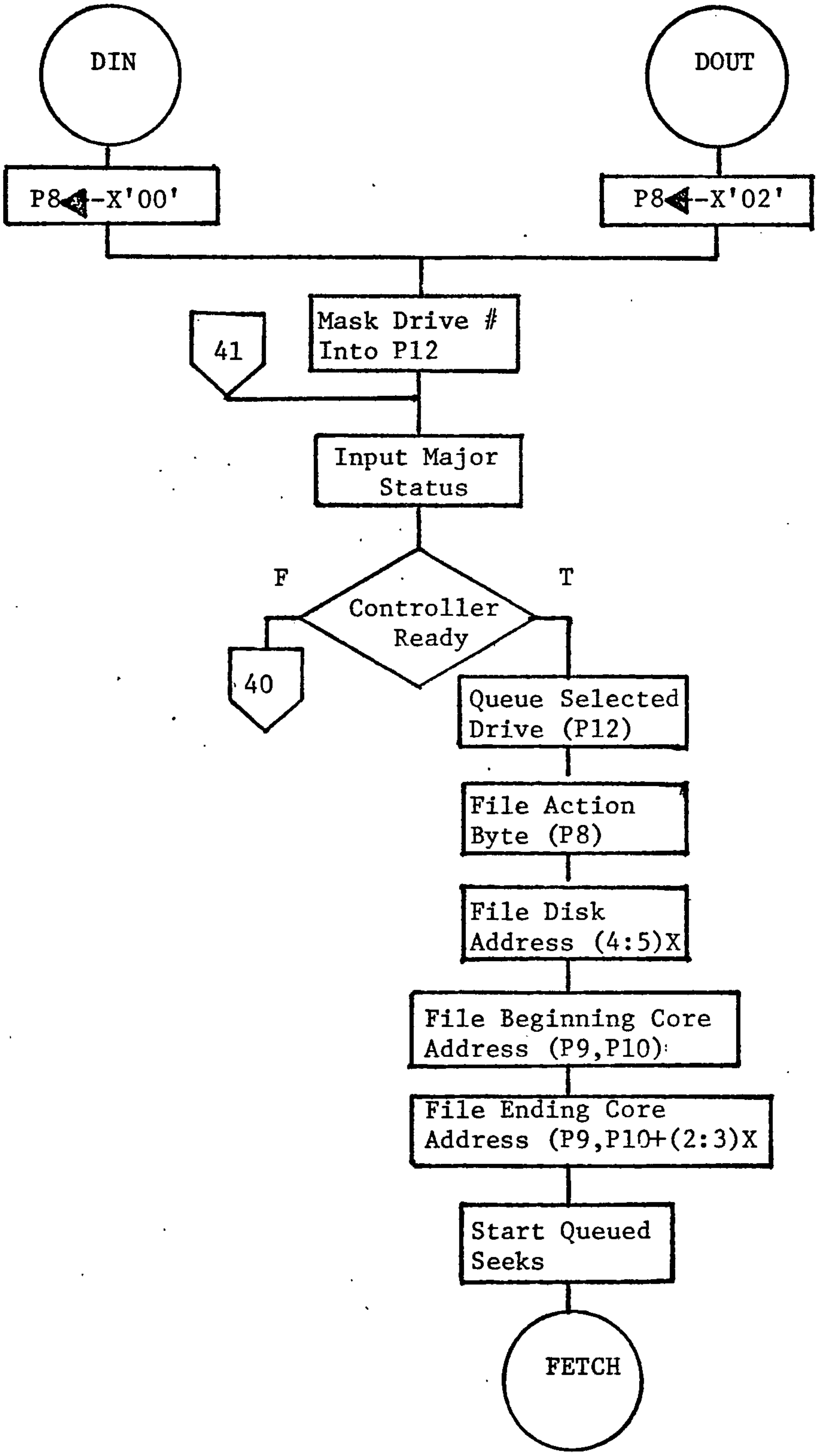

JRED
JBUS
T
Disk
F
T←-X'14'
Call
.DIX
F
TTY
T
T
Printer
F
Read In
TTY Status
(Internal
Status)
F
Controller
Ready
T
2
1
T←-X'1C'
P12←-P12+T
P7←-X'24'
P7←-X'25'
T←-P7
P7←-X'01'
Call
DIX
CF,P7←-P7-1
2
F
Zero
Result
T
1
F
JBUS
T
F
JRED
T
34
FETCH
34

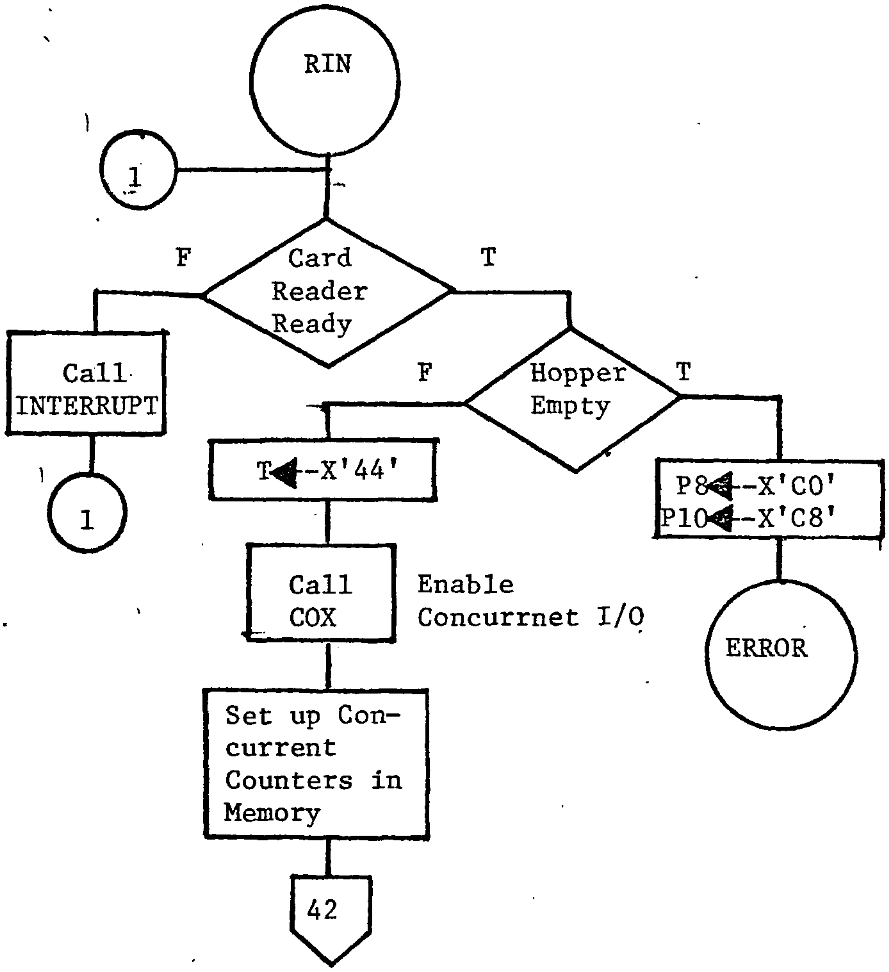

IOC
1
F
Printer Ready
T
Call INTERRUPT
1
T←X'05'
P7←X'8C'
Call .DOX
Output Form Feed
FETCH

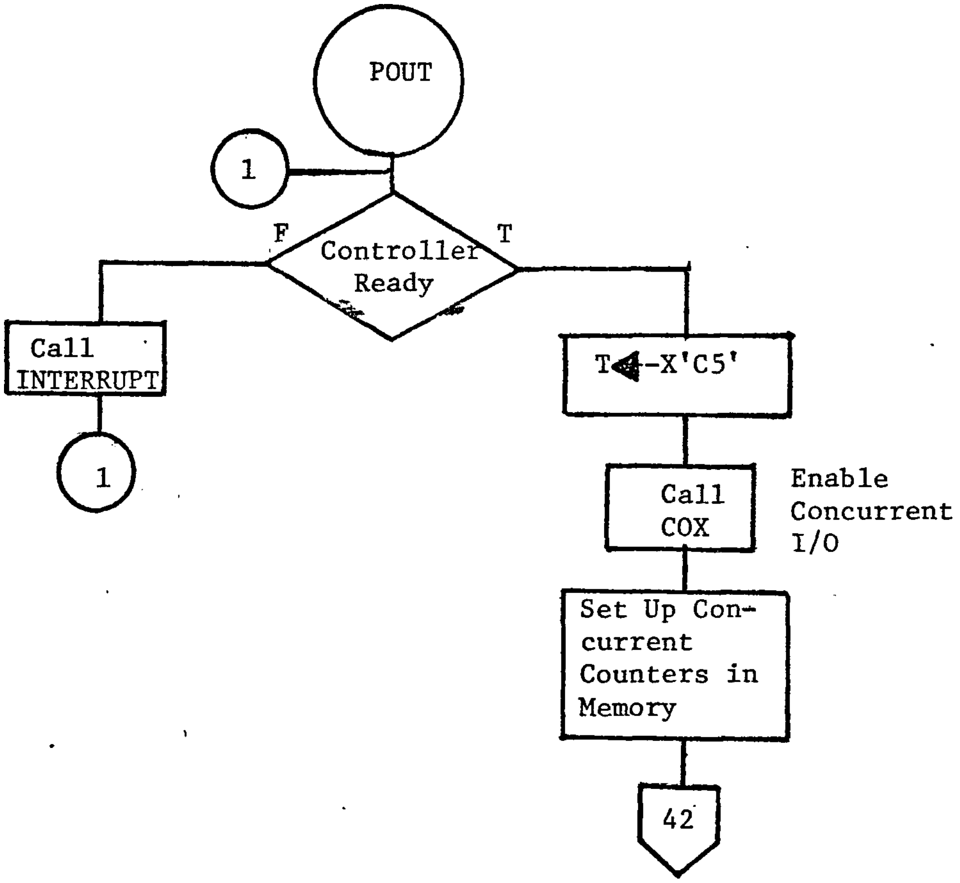

IN
T
Disk
F
DIN
T
TTY
TIN
F
RIN

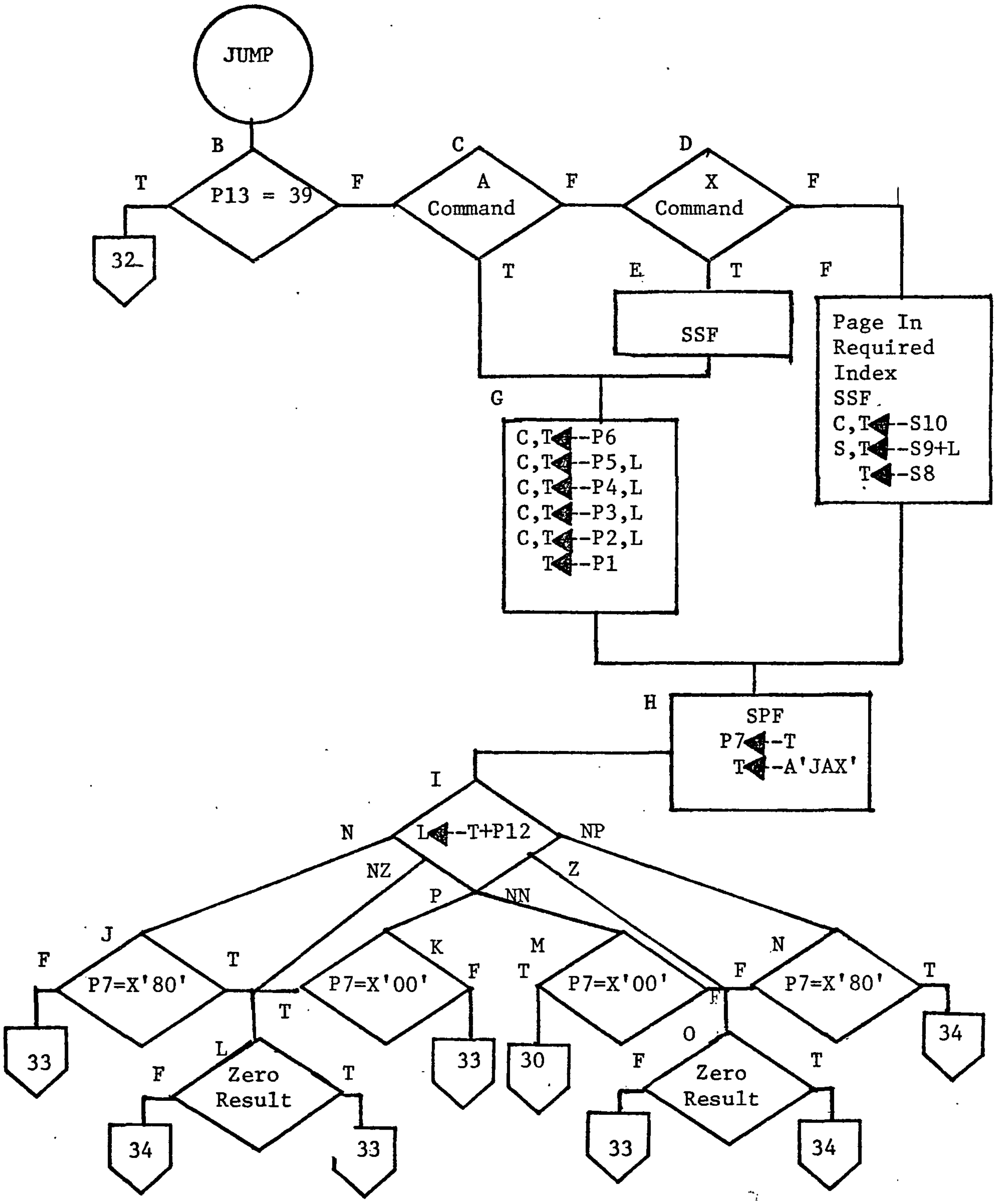

OUT
F
Disk
T
TTY
T
DOUT
F
TOUT
POUT

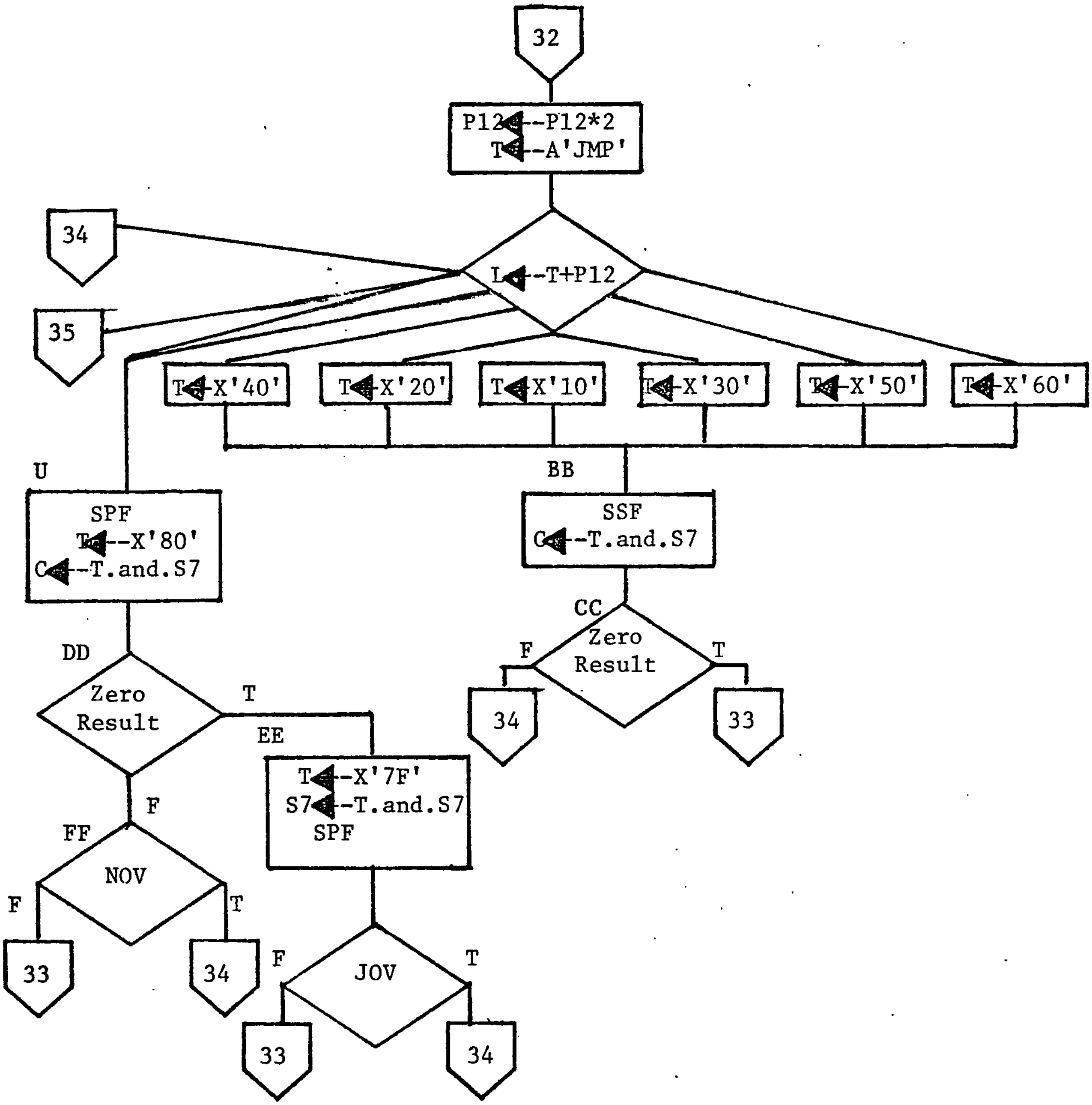

TIN
TOUT
P8◄--X'42'
(Set TTY Busy
On Input)
1
P8◄--X'44'
(Set TTY Busy
On Output)
Set Up
New TTY
Status
40
F
Test Old TTY
Status
TTY
Busy
T
T◄--P8
Write Full
Cycle
N,P7◄--P7-1
S11◄--P8
Call
INTERRUPT
P8◄--X'46'
P7◄--X'9F'
42
P8◄--S11
P12◄--P10
P13◄--P9
F
Disk
T
1
41
Call
I.OUT
FETCH

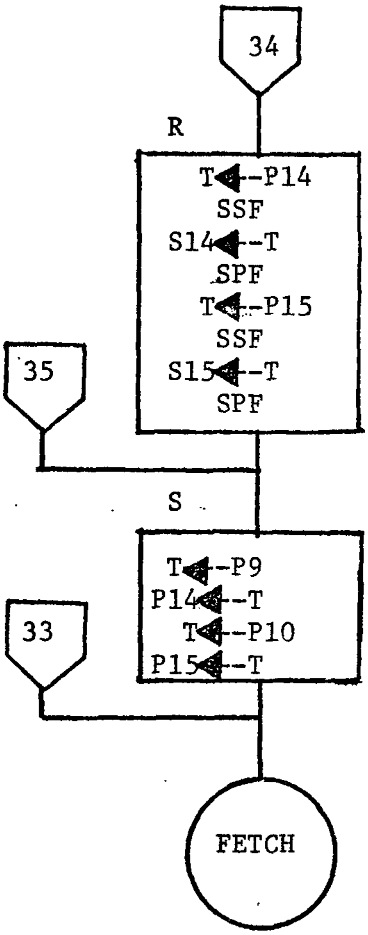

DIN
DOUT
P8←X'00'
P8←X'02'
Mask Drive #
Into P12
41
Input Major
Status
F
T
Controller
Ready
40
Queue Selected
Drive (P12)
File Action
Byte (P8)
File Disk
Address (4:5)X
File Beginning Core
Address (P9,P10)
File Ending Core
Address (P9,P10+(2:3)X
Start Queued
Seeks
FETCH

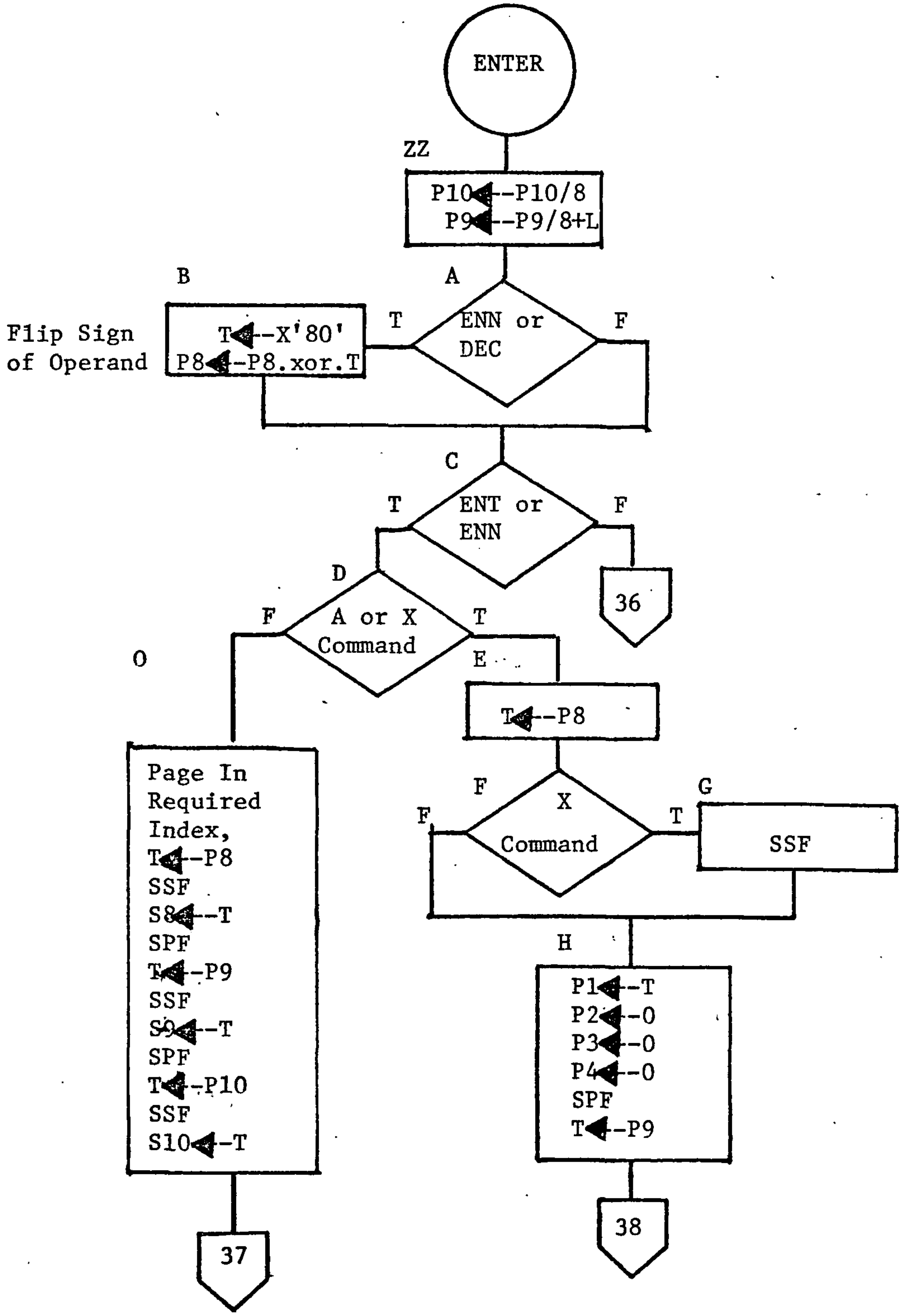

RIN
1
F
Card Reader Ready
T
Call INTERRUPT
1
F
Hopper Empty
T
T◀--X'44'
Call COX
Enable Concurrnet I/O
Set up Concurrent Counters in Memory
42
P8◀--X'C0'
P10◀--X'C8'
ERROR

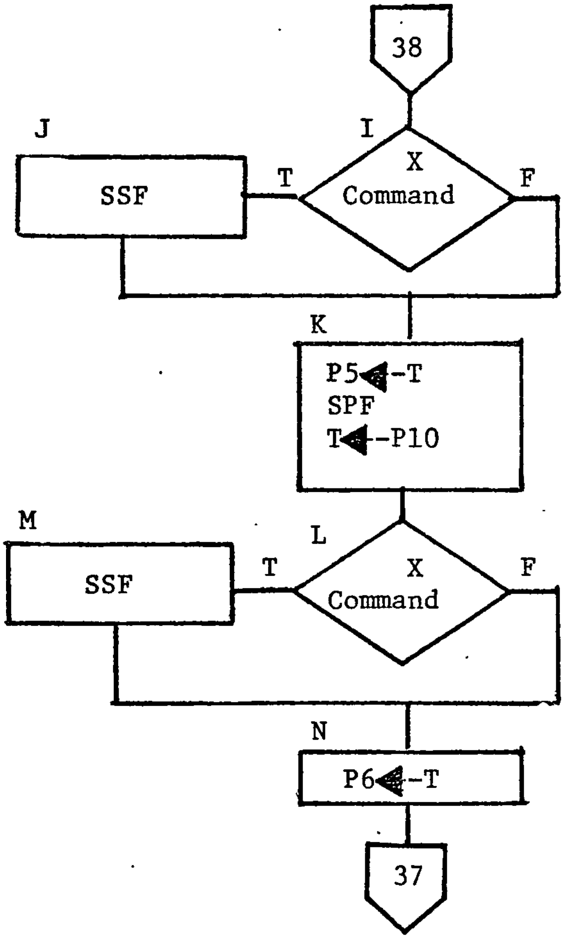

POUT
1
F
T
Controller Ready
Call INTERRUPT
1
T←X'C5'
Call COX
Enable Concurrent I/O
Set Up Concurrent Counters in Memory
42

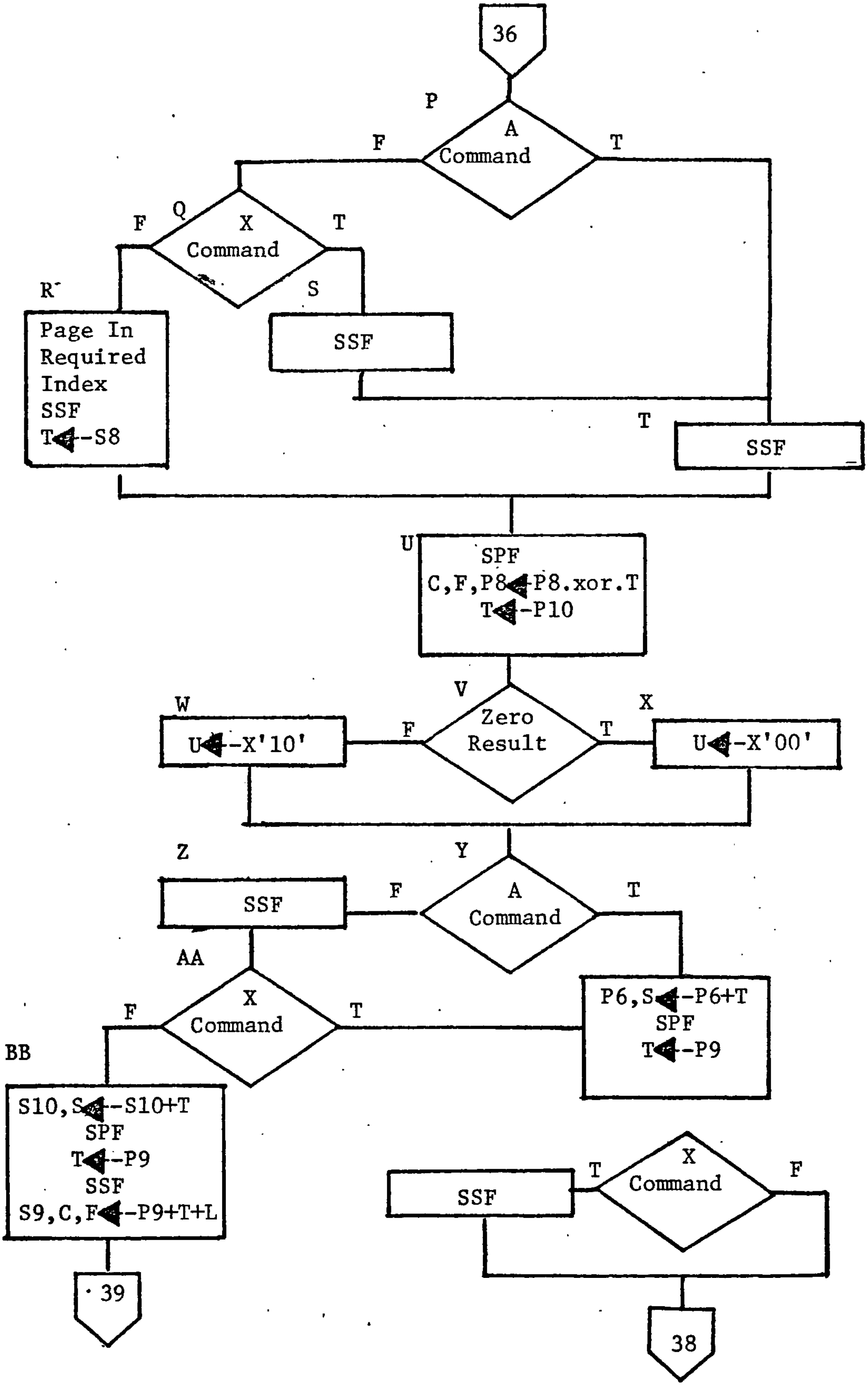

JUMP
B
P13 = 39
T
F
32
C
A
Command
F
T
D
X
Command
F
T
E
SSF
F
Page In
Required
Index
SSF
C,T◀--S10
S,T◀--S9+L
T◀--S8
G
C,T◀--P6
C,T◀--P5,L
C,T◀--P4,L
C,T◀--P3,L
C,T◀--P2,L
T◀--P1
H
SPF
P7◀--T
T◀--A'JAX'
I
L◀--T+P12
N
NZ
P
NN
Z
NP
J
P7=X'80'
F
T
33
K
P7=X'00'
T
F
33
L
Zero
Result
F
T
34
33
M
P7=X'00'
T
F
30
O
Zero
Result
F
T
33
34
N
P7=X'80'
F
T
34

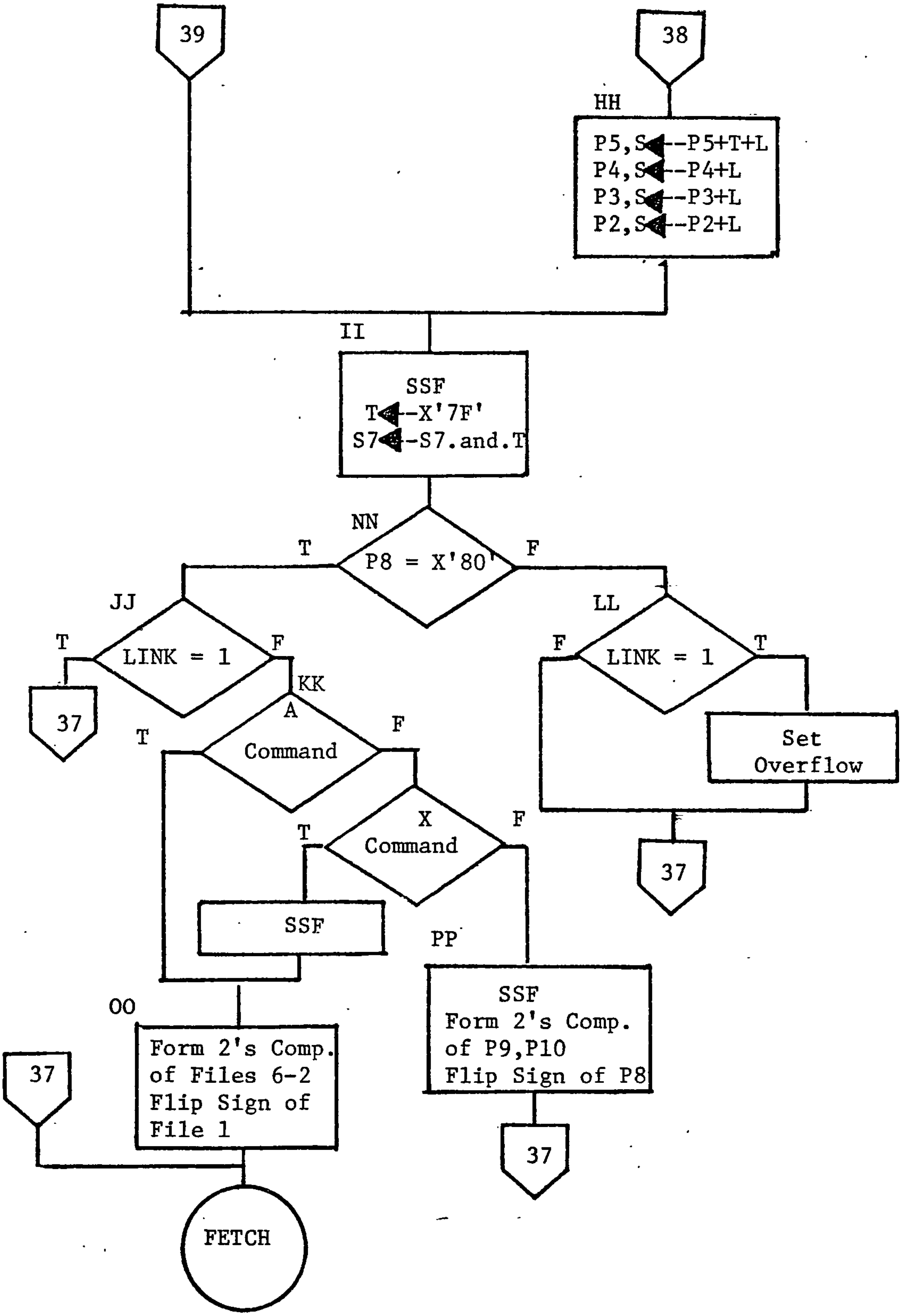

32
P12←-P12*2
T←-A'JMP'
34
L←-T+P12
35
T←-X'40'
T←-X'20'
T←-X'10'
T←-X'30'
T←-X'50'
T←-X'60'
U
SPF
T←-X'80'
C←-T.and.S7
BB
SSF
C←-T.and.S7
CC
Zero Result
F
T
34
33
DD
Zero Result
T
EE
T←-X'7F'
S7←-T.and.S7
SPF
F
FF
NOV
F
T
33
34
JOV
F
T
33
34

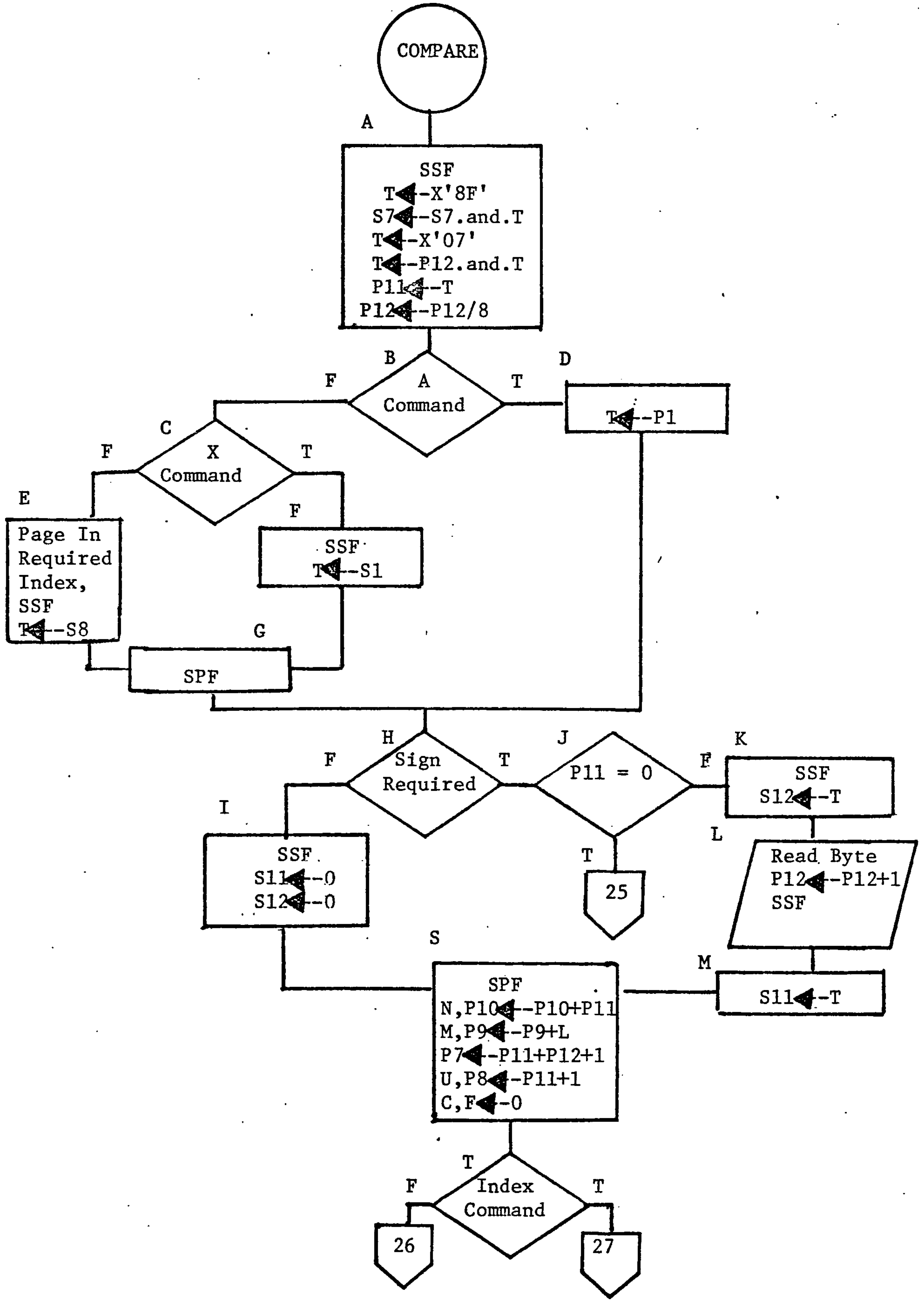

34
R
T◀--P14
SSF
S14◀--T
SPF
T◀--P15
SSF
S15◀--T
SPF
35
S
T◀--P9
P14◀--T
T◀--P10
P15◀--T
33
FETCH

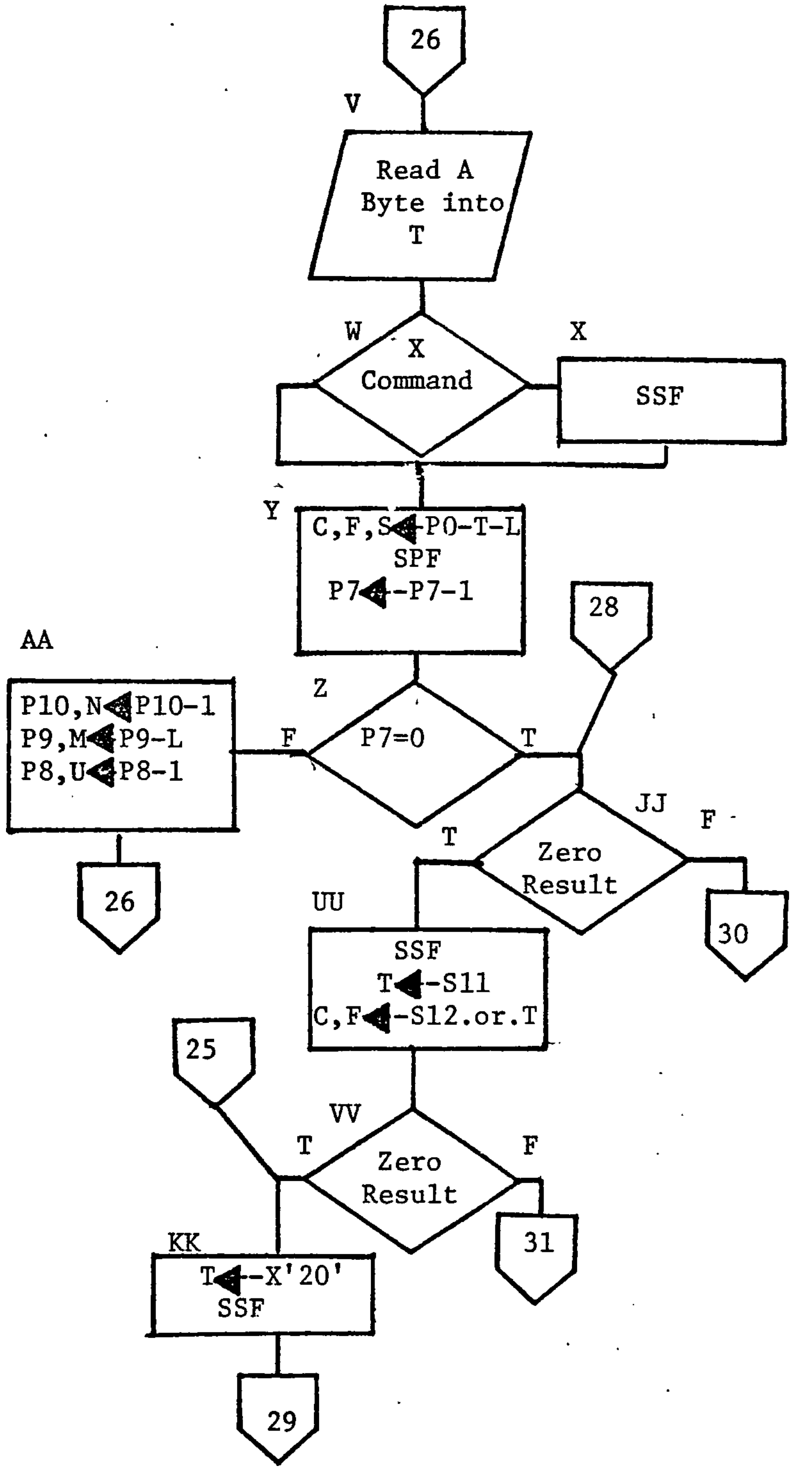

ENTER
ZZ
P10◀--P10/8
P9◀--P9/8+L
B
A
Flip Sign
of Operand
T◀--X'80'
P8◀--P8.xor.T
T
ENN or
DEC
F
C
T
ENT or
ENN
F
36
D
F
A or X
Command
T
O
E
T◀--P8
Page In
Required
Index,
T◀--P8
SSF
S8◀--T
SPF
T◀--P9
SSF
S9◀--T
SPF
T◀--P10
SSF
S10◀--T
37
F
F
X
Command
T
G
SSF
H
P1◀--T
P2◀--0
P3◀--0
P4◀--0
SPF
T◀--P9
38

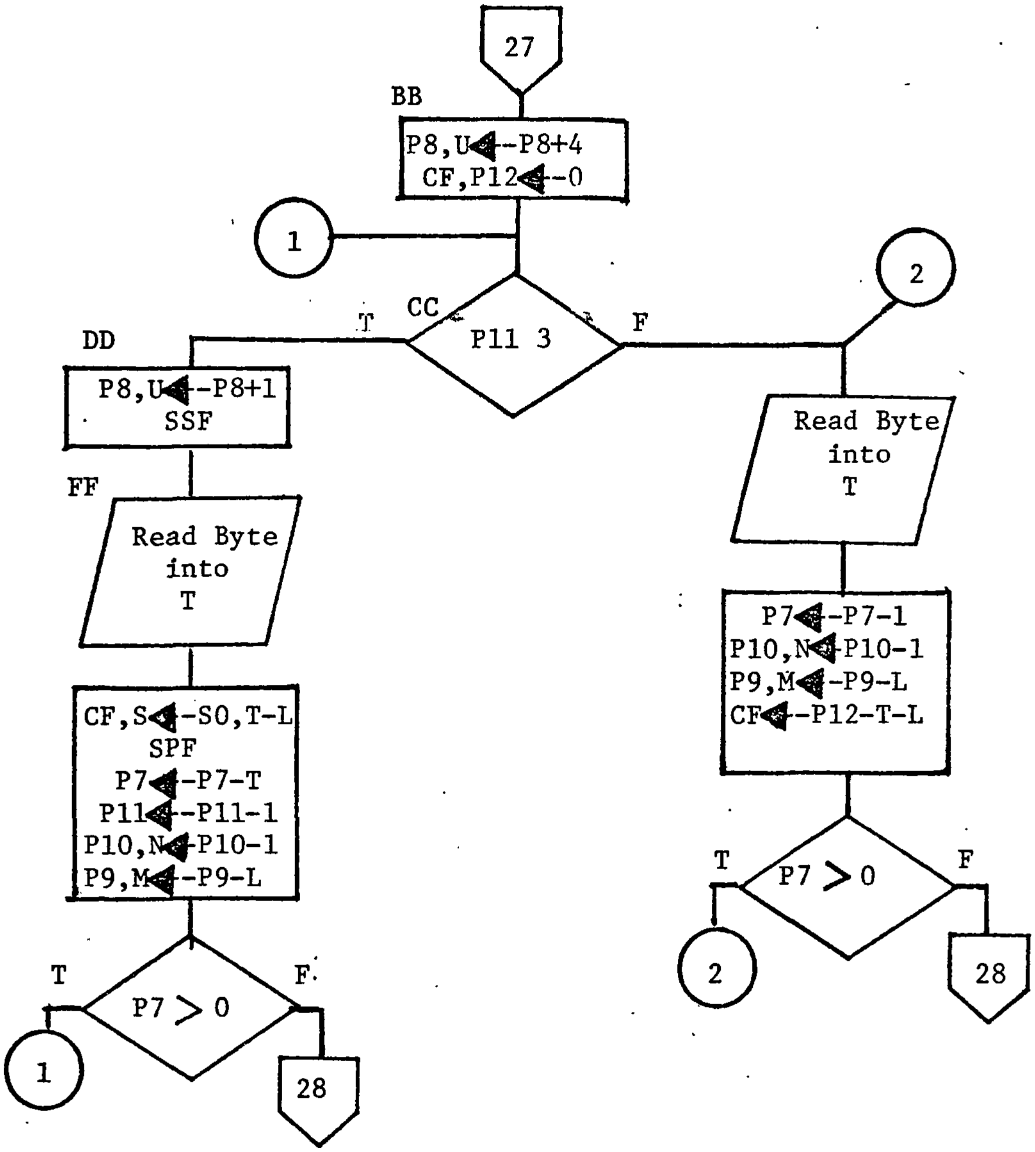

38
J
I
X
Command
T
F
SSF
K
P5◄-T
SPF
T◄-P10
M
L
T
X
Command
F
SSF
N
P6◄-T
37

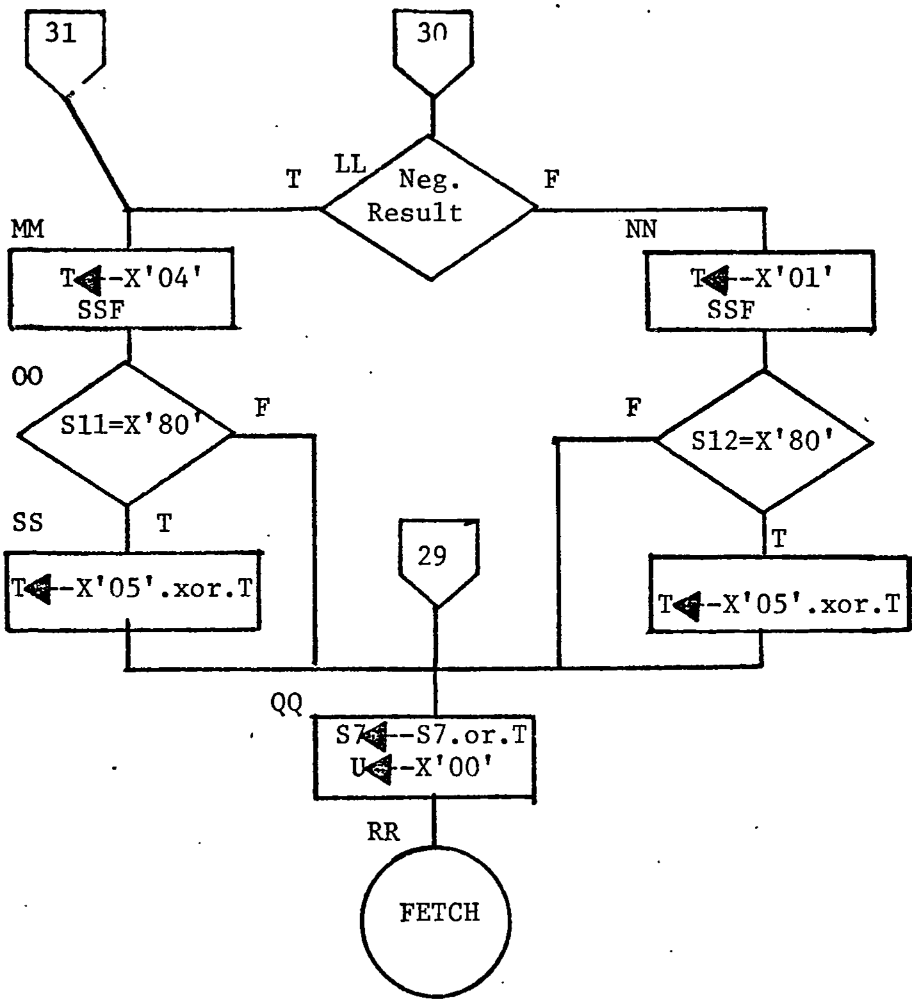

36
P
A Command
F
T
Q
X Command
F
T
R
Page In Required Index SSF T←S8
S
SSF
T
SSF
U
SPF
C,F,P8←P8.xor.T
T←P10
V
Zero Result
W
U←X'10'
F
T
X
U←X'00'
Z
Y
SSF
F
A Command
T
AA
X Command
F
T
P6,S←P6+T
SPF
T←P9
BB
S10,S←S10+T
SPF
T←P9
SSF
S9,C,F←P9+T+L
SSF
T
X Command
F
39
38


39
38
HH
P5,S←P5+T+L
P4,S←P4+L
P3,S←P3+L
P2,S←P2+L
II
SSF
T←X'7F'
S7←S7.and.T
NN
T
P8 = X'80'
F
JJ
T
LINK = 1
F
37
KK
A
Command
T
F
X
Command
T
F
SSF
PP
SSF
Form 2's Comp.
of P9,P10
Flip Sign of P8
37
OO
Form 2's Comp.
of Files 6-2
Flip Sign of
File 1
37
FETCH
LL
F
LINK = 1
T
Set
Overflow
37

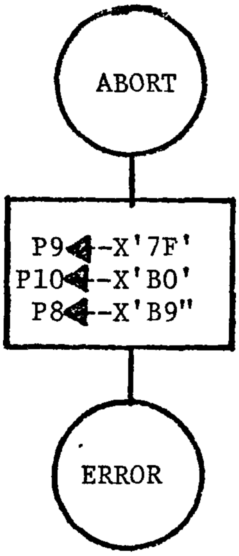

COMPARE
A
SSF
T◀--X'8F'
S7◀--S7.and.T
T◀--X'07'
T◀--P12.and.T
P11◀--T
P12◀--P12/8
B
F
A Command
T
D
T◀--P1
C
F
X Command
T
E
Page In Required Index, SSF
T◀--S8
F
SSF
T◀--S1
G
SPF
H
F
Sign Required
T
J
P11 = 0
F
K
SSF
S12◀--T
I
SSF
S11◀--0
S12◀--0
T
25
L
Read Byte
P12◀--P12+1
SSF
S
SPF
N,P10◀--P10+P11
M,P9◀--P9+L
P7◀--P11+P12+1
U,P8◀--P11+1
C,F◀--0
M
S11◀--T
T
F
Index Command
T
26
27

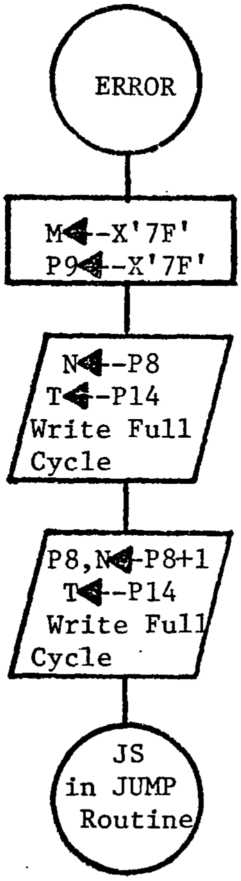

26
V
Read A Byte into T
W
X Command
X
SSF
Y
C,F,S←P0-T-L
SPF
P7←P7-1
28
AA
P10,N←P10-1
P9,M←P9-L
P8,U←P8-1
Z
F
P7=0
T
JJ
F
T
Zero Result
26
UU
30
SSF
T←S11
C,F←S12.or.T
25
VV
T
Zero Result
F
31
KK
T←X'20'
SSF
29

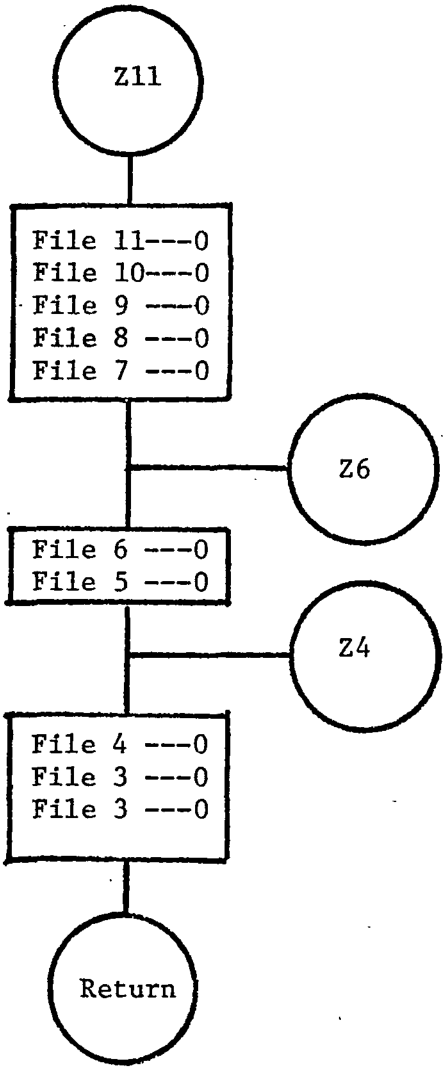

27
BB
P8,U◄--P8+4
CF,P12◄--0
1
2
CC
T
P11 3
F
DD
P8,U◄--P8+1
SSF
FF
Read Byte into T
CF,S◄--S0,T-L
SPF
P7◄--P7-T
P11◄--P11-1
P10,N◄-P10-1
P9,M◄--P9-L
T
P7 > 0
F
1
28
Read Byte into T
P7◄--P7-1
P10,N◄-P10-1
P9,M◄--P9-L
CF◄--P12-T-L
T
P7 > 0
F
2
28

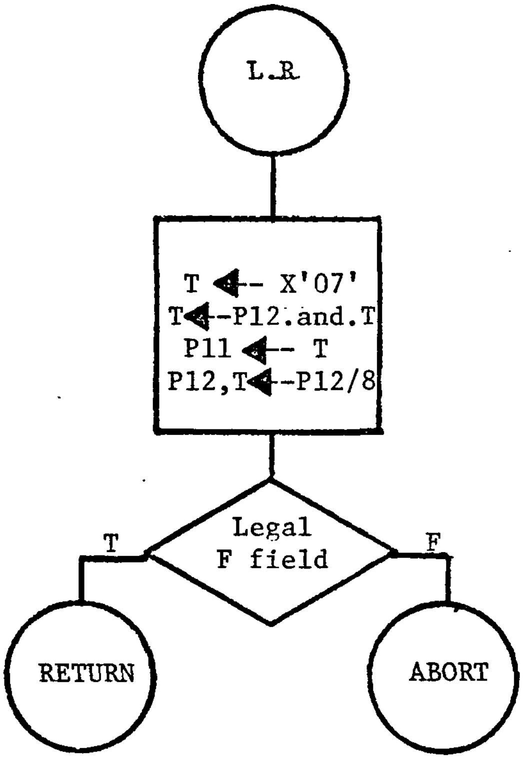

31
30
LL
Neg. Result
T
F
MM
T←X'04'
SSF
NN
T←X'01'
SSF
OO
S11=X'80'
F
S12=X'80'
F
SS
T
T←X'05'.xor.T
29
T
T←X'05'.xor.T
QQ
S7←S7.or.T
U←X'00'
RR
FETCH

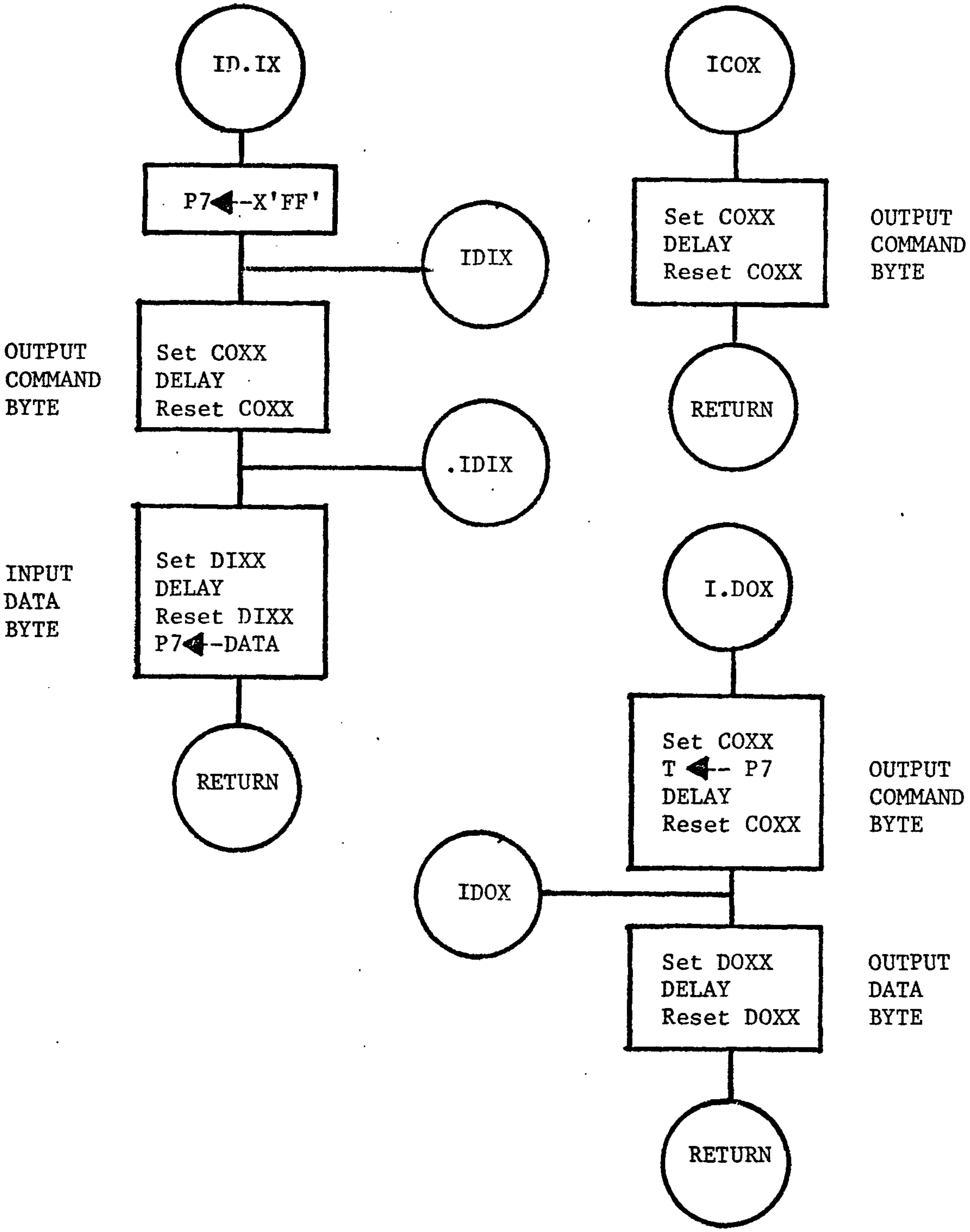
III. Subroutines

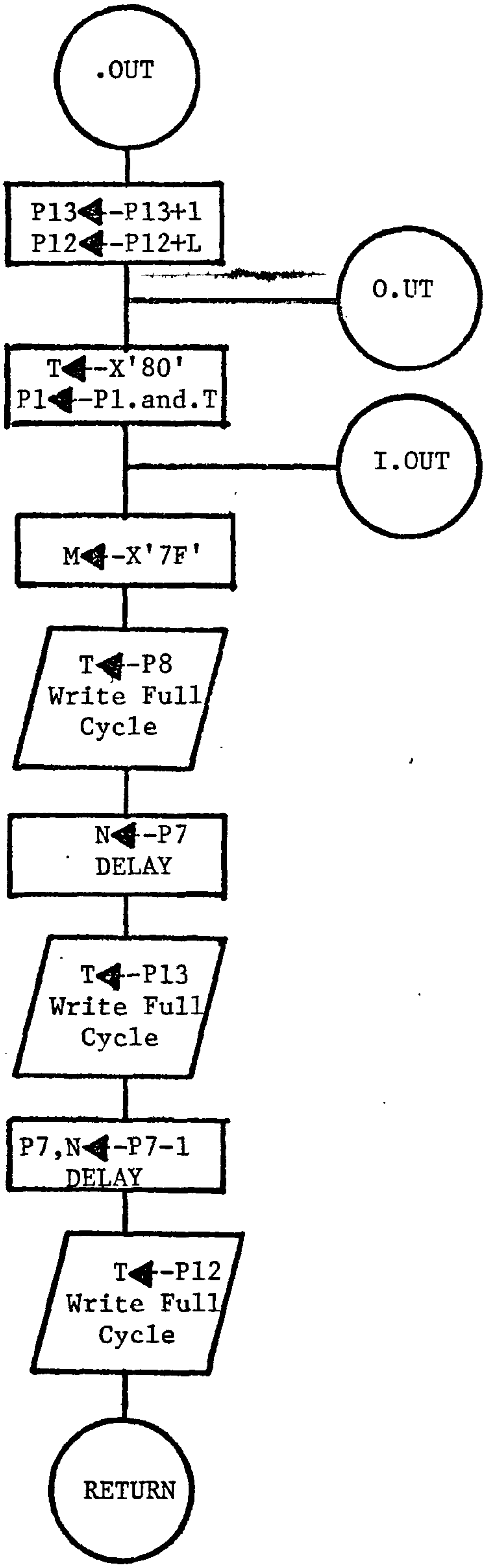

ABORT
P9←X'7F'
P10←X'B0'
P8←X'B9"
ERROR

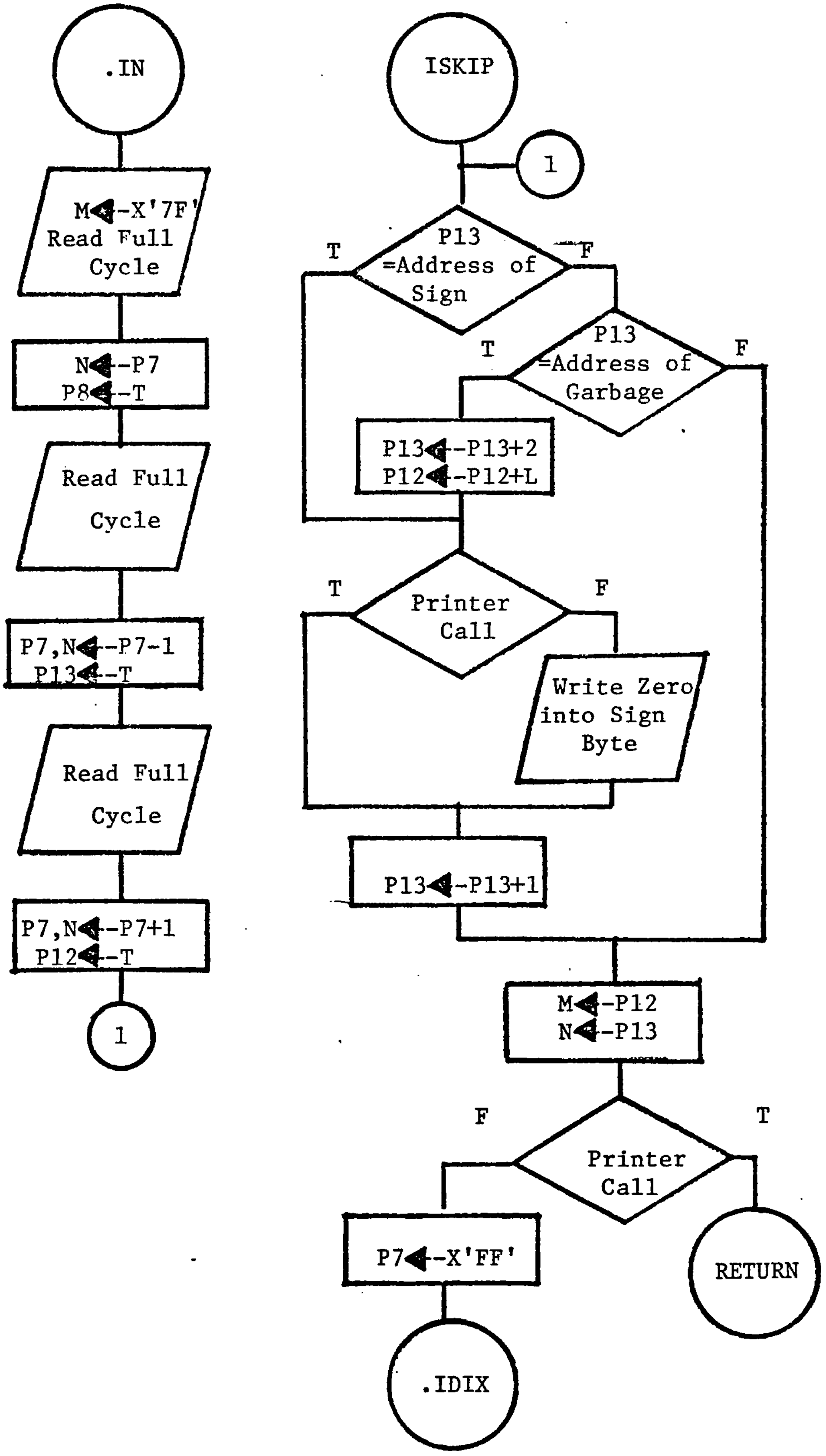

ERROR
M←X'7F'
P9←X'7F'
N←P8
T←P14
Write Full Cycle
P8,N←P8+1
T←P14
Write Full Cycle
JS in JUMP Routine

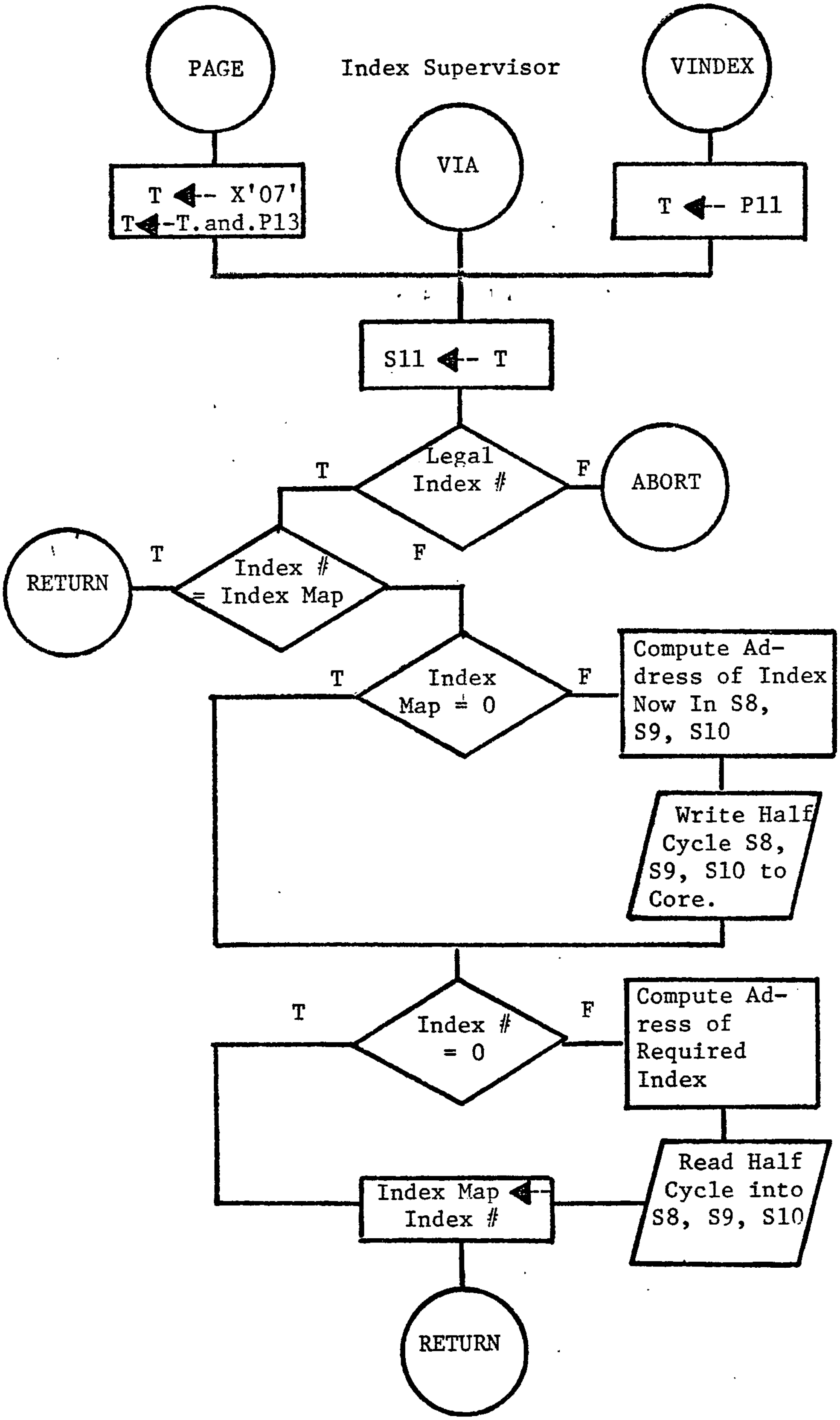

Z11
File 11---0
File 10---0
File 9 ---0
File 8 ---0
File 7 ---0
Z6
File 6 ---0
File 5 ---0
Z4
File 4 ---0
File 3 ---0
File 3 ---0
Return

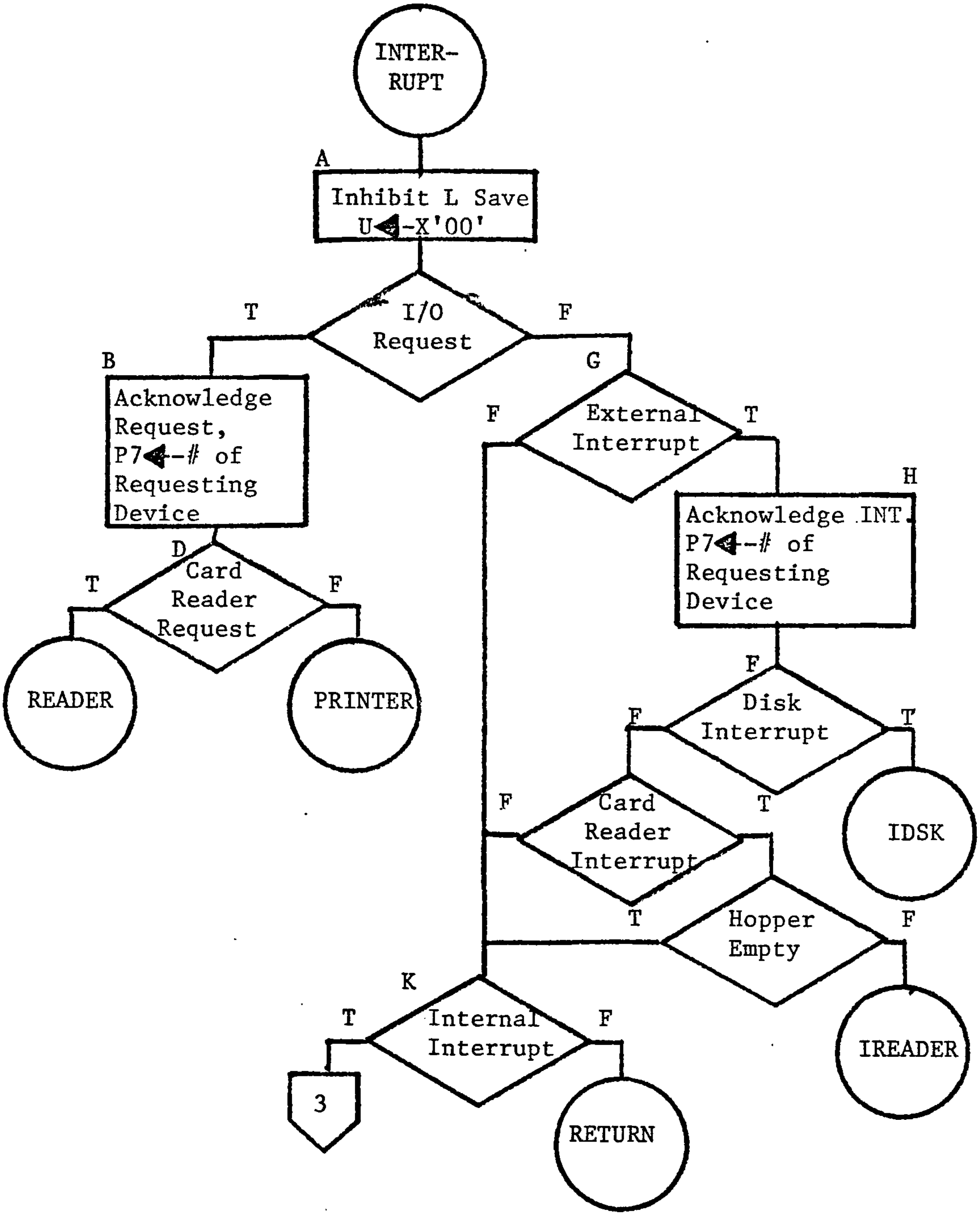

L.R
T ←-- X'07'
T←-P12.and.T
P11 ←-- T
P12,T←-P12/8
Legal
F field
T
F
RETURN
ABORT

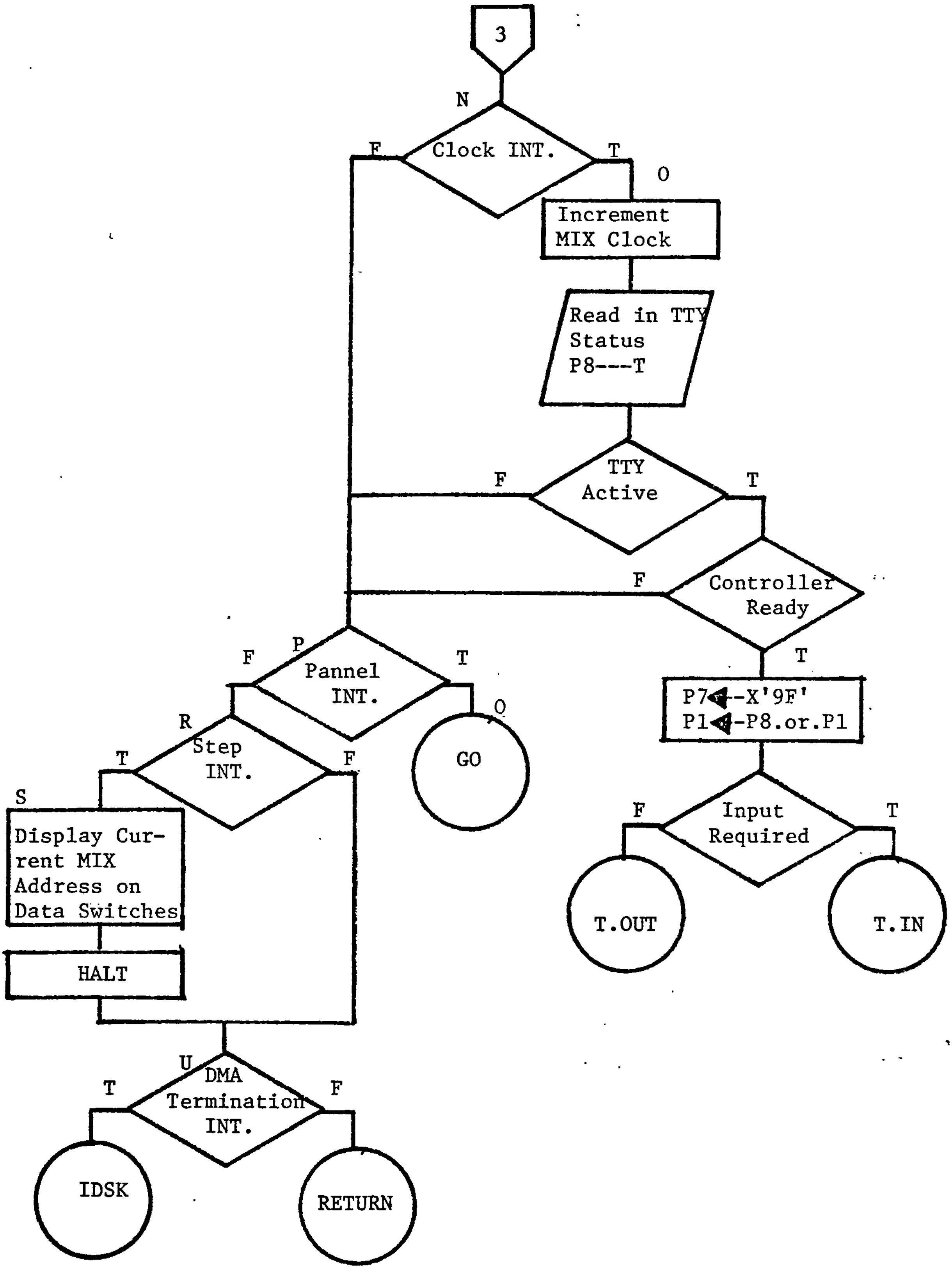

ID.IX
P7←X'FF'
IDIX
OUTPUT
COMMAND
BYTE
Set COXX
DELAY
Reset COXX
.IDIX
INPUT
DATA
BYTE
Set DIXX
DELAY
Reset DIXX
P7←DATA
RETURN
ICOX
Set COXX
DELAY
Reset COXX
OUTPUT
COMMAND
BYTE
RETURN
I.DOX
Set COXX
T ← P7
DELAY
Reset COXX
OUTPUT
COMMAND
BYTE
IDOX
Set DOXX
DELAY
Reset DOXX
OUTPUT
DATA
BYTE
RETURN

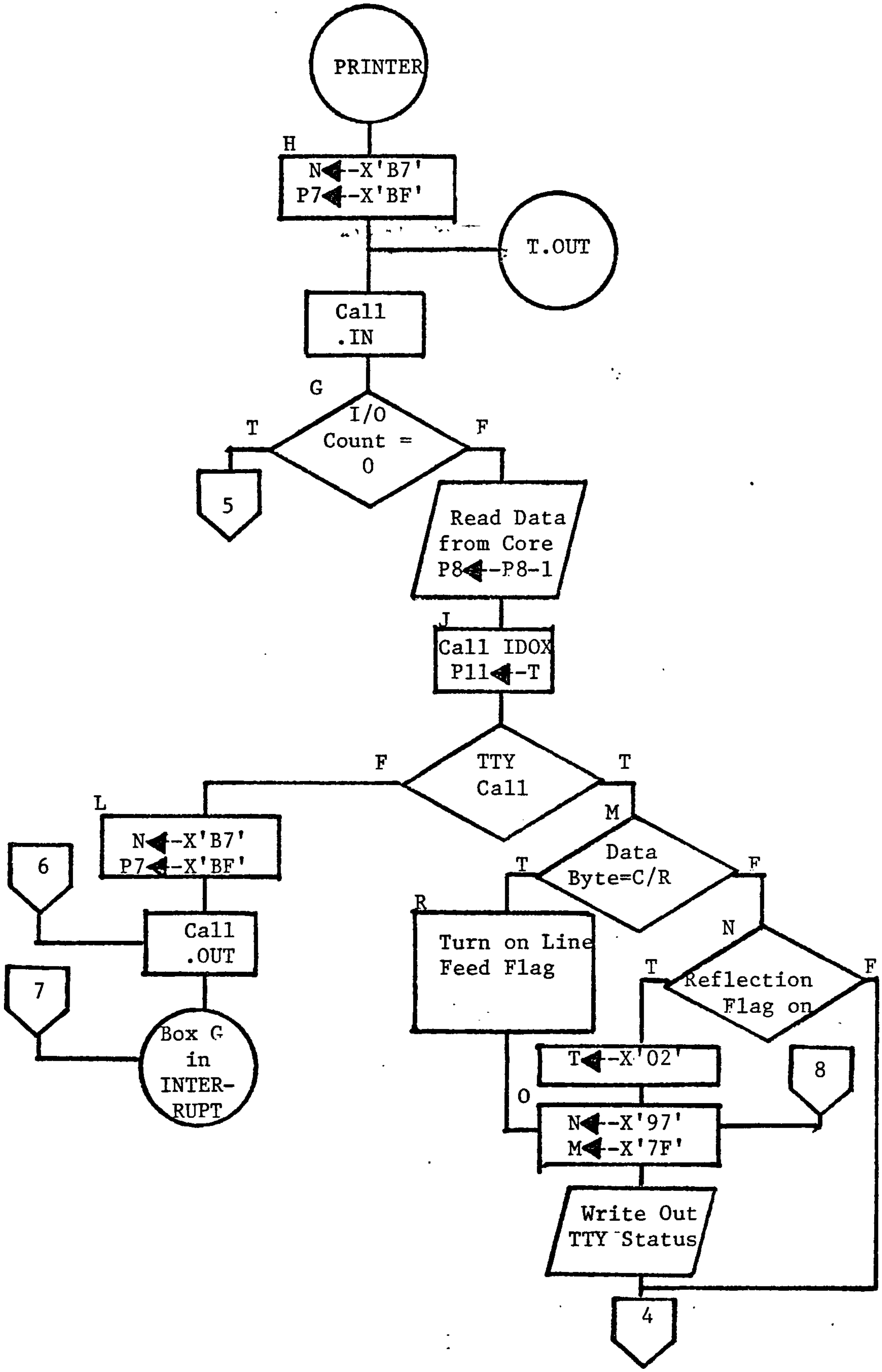

.OUT
P13◄-P13+1
P12◄-P12+L
O.UT
T◄-X'80'
P1◄-P1.and.T
I.OUT
M◄-X'7F'
T◄-P8
Write Full
Cycle
N◄-P7
DELAY
T◄-P13
Write Full
Cycle
P7,N◄-P7-1
DELAY
T◄-P12
Write Full
Cycle
RETURN

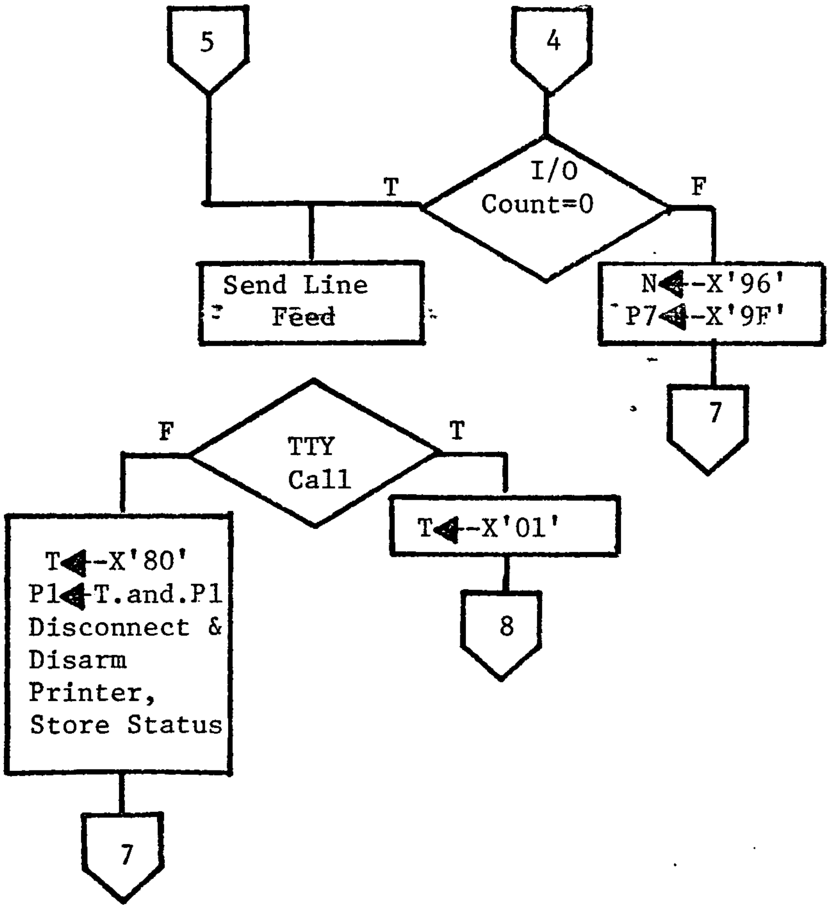

.IN
M◀-X'7F'
Read Full
Cycle
N◀-P7
P8◀-T
Read Full
Cycle
P7,N◀-P7-1
P13◀-T
Read Full
Cycle
P7,N◀-P7+1
P12◀-T
1
ISKIP
1
T
P13
=Address of
Sign
F
T
P13
=Address of
Garbage
F
P13◀-P13+2
P12◀-P12+L
T
Printer
Call
F
Write Zero
into Sign
Byte
P13◀-P13+1
M◀-P12
N◀-P13
F
Printer
Call
T
P7◀-X'FF'
RETURN
.IDIX

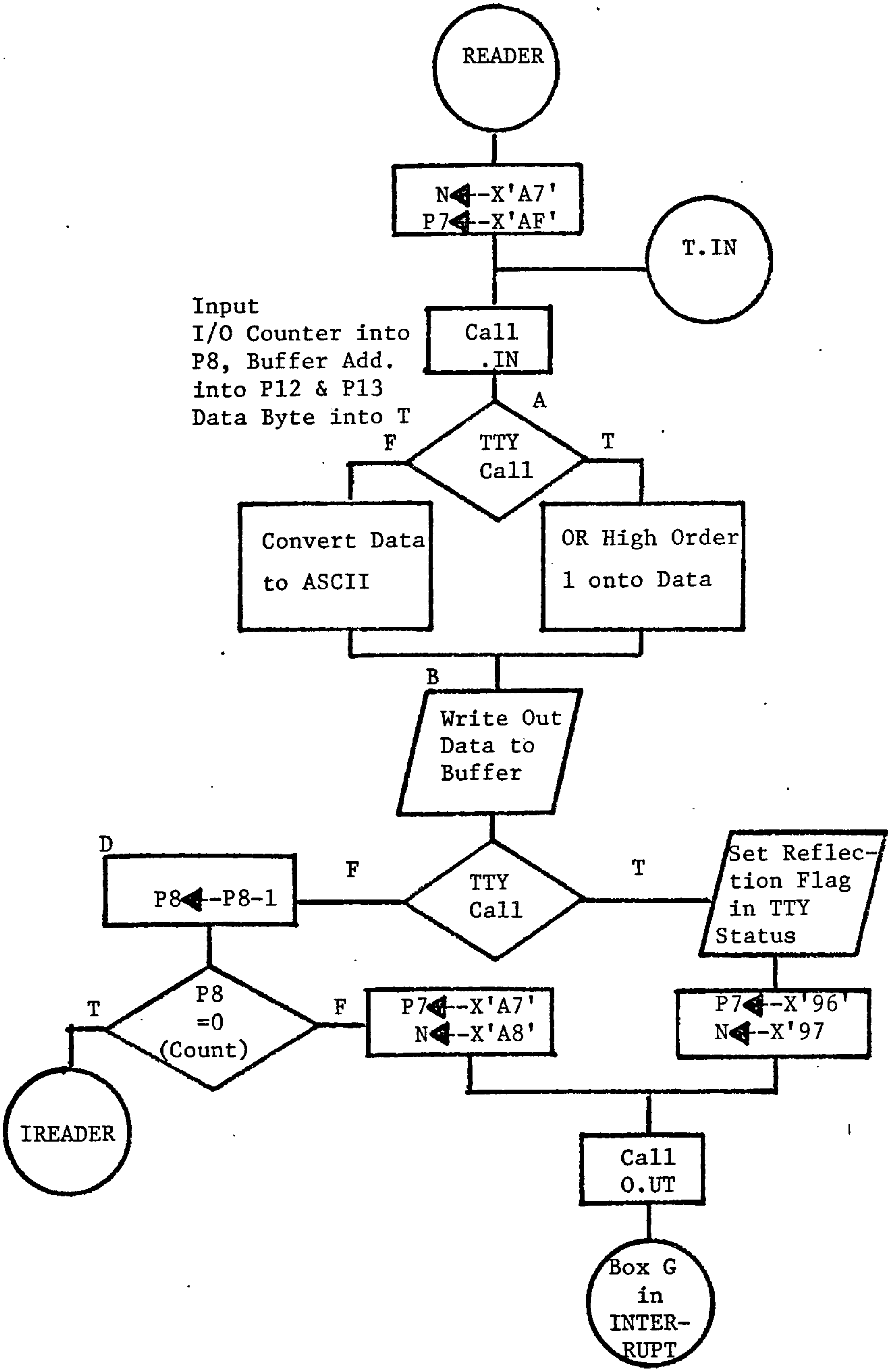

PAGE
Index Supervisor
VINDEX
VIA
T ←- X'07'
T←-T.and.P13
T ←- P11
S11 ←- T
Legal Index #
T
F
ABORT
RETURN
T
Index # = Index Map
F
Index Map ≠ 0
T
F
Compute Address of Index Now In S8, S9, S10
Write Half Cycle S8, S9, S10 to Core.
Index # = 0
T
F
Compute Adress of Required Index
Read Half Cycle into S8, S9, S10
Index Map ←- Index #
RETURN

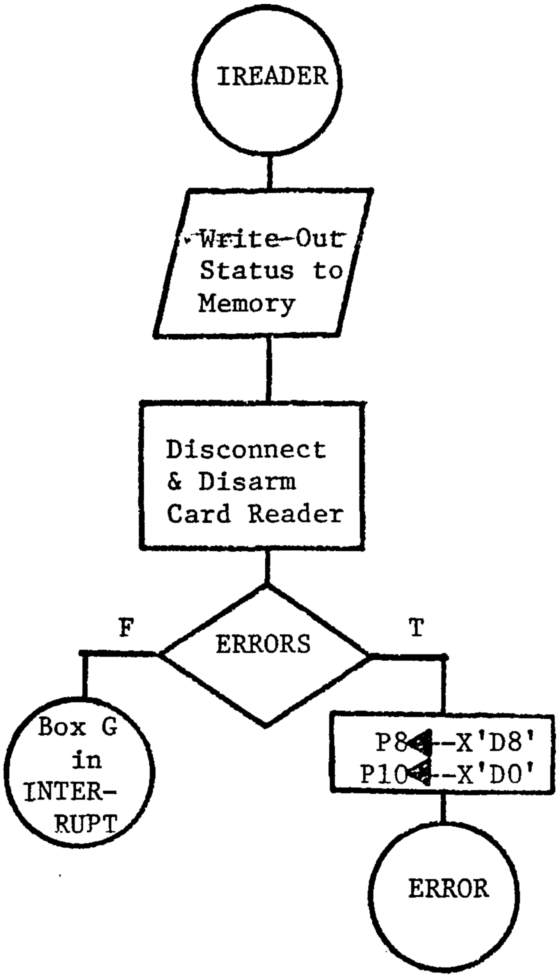

INTER-
RUPT
A
Inhibit L Save
U←X'00'
T
I/O
Request
F
B
Acknowledge
Request,
P7←# of
Requesting
Device
G
F
External
Interrupt
T
H
Acknowledge INT.
P7←# of
Requesting
Device
D
T
Card
Reader
Request
F
READER
PRINTER
F
Disk
Interrupt
F
T
IDSK
F
Card
Reader
Interrupt
T
T
Hopper
Empty
F
K
T
Internal
Interrupt
F
3
RETURN
IREADER

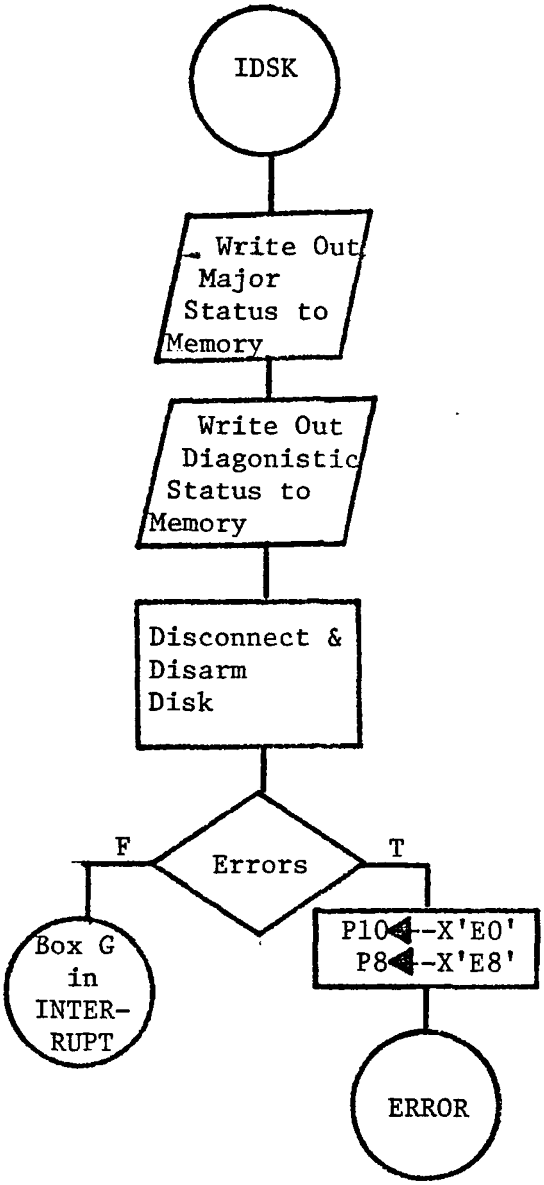

3
N
Clock INT.
F
T
O
Increment MIX Clock
Read in TTY Status P8---T
TTY Active
F
T
Controller Ready
F
T
P7◀-X'9F'
P1◀-P8.or.P1
Input Required
F
T
T.OUT
T.IN
P
Pannel INT.
F
T
Q
GO
R
Step INT.
T
F
S
Display Current MIX Address on Data Switches
HALT
U
DMA Termination INT.
T
F
IDSK
RETURN


PRINTER
H
N◀--X'B7'
P7◀--X'BF'
T.OUT
Call
.IN
G
I/O
Count =
0
T
F
5
Read Data
from Core
P8◀--P8-1
J
Call IDOX
P11◀-T
TTY
Call
F
T
L
N◀-X'B7'
P7◀-X'BF'
6
Call
.OUT
7
Box G
in
INTER-
RUPT
M
Data
Byte=C/R
T
F
R
Turn on Line
Feed Flag
N
T
Reflection
Flag on
F
T◀-X'02'
8
O
N◀-X'97'
M◀-X'7F'
Write Out
TTY Status
4


5
4
I/O Count=0
T
F
Send Line Feed
N◀-X'96'
P7◀-X'9F'
7
TTY Call
F
T
T◀-X'80'
P1◀T.and.P1
Disconnect &
Disarm
Printer,
Store Status
T◀-X'01'
8
7


READER
N←-X'A7'
P7←-X'AF'
T.IN
Input
I/O Counter into
P8, Buffer Add.
into P12 & P13
Data Byte into T
Call
.IN
A
F
TTY
Call
T
Convert Data
to ASCII
OR High Order
1 onto Data
B
Write Out
Data to
Buffer
D
P8←-P8-1
F
TTY
Call
T
Set Reflection Flag
in TTY
Status
T
P8
=0
(Count)
F
P7←-X'A7'
N←-X'A8'
P7←-X'96'
N←-X'97
IREADER
Call
O.UT
Box G
in
INTER-
RUPT


IREADER
Write-Out Status to Memory
Disconnect & Disarm Card Reader
ERRORS
F
T
Box G in INTER-RUPT
P8←X'D8'
P10←X'D0'
ERROR


IDSK
Write Out Major Status to Memory
Write Out Diagonistic Status to Memory
Disconnect & Disarm Disk
Errors
F
T
Box G in INTER-RUPT
P10←-X'E0'
P8←-X'E8'
ERROR

CHAPTER VI. MICROPROGRAM LISTINGS

Unused Names...

TMS

CMS

PMS

LDL

ISK

VNN

ITT

IPR

ADI

ADK

SN

JPR

```
0600                      ORG    X'0600'
     003F          HICORE EQU    X'3F'
     0002          TOS    EQU    X'02'
     007B          .TOS   EQU    X'7B'
     007C          BOOT   EQU    X'7C'     ADDRESS OF BOOT STRAP PROGRAM
     00F8          .BOOT  EQU    X'F8'
     0076          TCNT   EQU    X'76'
     0077          TSTAT  EQU    X'77'
     007E          TMSB   EQU    X'7E'
     007F          TLSB   EQU    X'7F'
     0087          CCNT   EQU    X'87'
     008E          CMSB   EQU    X'8E'
     008F          CLSB   EQU    X'8F'
     0097          BASE   EQU    X'97'      BASE ADDRESS FOR INDEX£REG.
     0097          PCNT   EQU    X'97'
     009E          PMSB   EQU    X'9E'
     009F          PLSB   EQU    X'9F'
     00C0          HSAVE  EQU    X'C0'
     00C8          HADD   EQU    X'C8'
     00B0          ABTADD EQU    X'B0'      ABORT ADDRESS
     00B9          ABSAVE EQU    X'B9'      ABORT RETURN ADDRESS
     00D0          CIADD  EQU    X'D0'
     00D8          CISAVE EQU    X'D8'
     00E0          DADD   EQU    X'E0'
     00E8          DSAVE  EQU    X'E8'
     00F2          PSTAT  EQU    X'F2'      PRINTER STATUS
     00F3          CSTAT  EQU    X'F3'      CARD READER£STATUS
     00F4          DSTATD EQU    X'F4'      DISK DIAGONSTIC STATUS
                   *             DSTATD + 1 IS THE DISK MAJOR STATUS
     00F9          LDMSB  EQU    X'F9'      MSB MIX£START ADDRESS
     00FA          LDLSB  EQU    X'FA'      LSB MIX£START ADDRESS
     00FB          CLOCK  EQU    X'FB'      MSB OF CLOCK (3 BYTES)
     00FD          .CLOCK EQU    X'FD'      LSB OF CLOCK (3 BYTES)
```

```
                         * GO = THIS IS THE INITIALIZATION ROUTINE   *******
                         *
                         ********
                         *BOX£M  *
                         ********
0600 7080                GO      CIO   0          CLEAR I/O CONTROL REGISTER
0601 1708                        EEI            ENABLE EXTERNAL INTERRUPTS
                         *********-
                         *BOX£N *
                         ********
0602 1720                        ERT            ENABLE REAL=TIME CLOCK
                         ********
                         *BOX£O *      ZERO CLOCK
                         ********
0603 28FB                        LF    8,CLOCK
0604 123F                        LM    HICORE
0605 A813                        WMF   8,(N)
0606 1100                        LT    X'00'
0607 A8D3                        WMF   8,I,(N)
0608 1000                        NOP
0609 A8D3                        WMF   8,I,(N)
                         *             SET TTY TO READY
060A 2877                        LF    8,TSTAT
060B A813                        WMF   8,(N)
060C 1101                        LT    X'01'
                         ********
                         *BOX£P *
                         ********
060D 1080                        SSF
060E B700                        ZOF   7
060F 1040                        SPF
0610 28F9                        LF    8,LDMSB    LOAD STARTING ADDRESS MSB
0611 A803                        RMF   8,(N)
0612 1000                        NOP
0613 BE20                        CPY   14,T
0614 A8C3                        RMF   8,I,(N)
0615 1000                        NOP
0616 BF20                        CPY   15,T
0617 FF00                        SFL   15
0618 FE80                        SFL   14,L
0619 FF00                        SFL   15
061A FE80                        SFL   14,L
061B FF00                        SFL   15
061C FE80                        SFL   14,L
061D 1780                        HLT
061E 1600                        LU    X'00'
061F 7710                        ESS   7          ENTER SENSE SWITCHES
0620 5780                        TN    7,X'80'    TEST FOR CARD READER BOOTSTRA
```

```
0621 1C24                JP    *+3
0622 2E7C                LF    14,BOOT
0623 2FF8                LF    15,,BOOT
0624 07E1                JE    FETCH
```

```
                          *  THIS IS THE EBCDIC TO ASCII CONVERSION
                          *  TABLE USED BY THE CARD READER£ROUTINE
0625 27A0                 ASCII  LF     7,X'A0'
0626 1DA0                        JP     RB
                          *          THIS ROUTINE CONVERTS EBCDIC CHARACTERS INTO
                          *  ASCII CHARACTERS
                          *-         FIRST THE EBCDIC CHARACTERS ARE BROKEN
                          *          INTO FIVE GROUPS
                          *          GROUP 1 - SPECIAL CHARACTERS CODES 40-7F
                          *                    THESE CODES MUST£BE LOOKED UP
                          *                    IN THE ASCII TABLE
                          *          GROUP 2 - LETTERS A - I, THESE ARE CORRECT
                          *          GROUP 3 - LETTERS J - R, TO CONVERT THESE
                          *                    TO ASCII SUBTRACT£X'07'
                          *          GROUP 4 - LETTERS S - Z, TO CONVERT THESE
                          *                    SUBTRACT X'0F'
                          *          GROUP 5  NUMBERS, TO CONVERT£THESE
                          *                    SUBTRACT X'40'
0627 677F                 EBCDIC CP     7,X'7F'   TEST FOR£SPECIAL CHARACTERS
0628 1C47                        JP     SPCHAR    SPECIAL CHARACTER
0629 6736                        CP     7,X'36'   LETTERS A - I
062A 1DA0                        JP     RB        A - I
062B 6726                        CP     7,X'26'   TEST FOR J - R
062C 1C31                        JP     D1,D9     CHARACTERS J - R
062D 6716                        CP     7,X'16'   TEST FOR LETTERS S - Z
062E 1C33                        JP     E2,E9     S - Z SUBTRACT X'0F'
                          *                MUST BE A NUMBER
062F 37C0                        AF     7,X'C0'   SUBTRACT£X'40'
0630 1DA0                        JP     RB
0631 37F9                 D1,D9  AF     7,X'F9'   SUBTRACT£X'07'
0632 1DA0                        JP     RB
0633 37F1                 E2,E9  AF     7,X'F1'   SUBTRACT£X'0F'
0634 1DA0                        JP     RB
0635 1000                        NOP              FREE LOCATION
0636 1000                        NOP              FREE LOCATION
0637 1000                        NOP              FREE LOCATION
0638 1000                        NOP              FREE LOCATION
0639 27DC                        LF     7,X'DC'
063A 1DA0                        JP     RB
063B 27AE                        LF     7,X'AE'
063C 1DA0                        JP     RB
063D 27BC                        LF     7,X'BC'
063E 1DA0                        JP     RB
063F 27A8                        LF     7,X'A8'
0640 1DA0                        JP     RB
0641 27AB                        LF     7,X'AB'
0642 1DA0                        JP     RB
0643 27DB                        LF     7,X'DB'
```

```
0644 1DA0              JP     RB
0645 27A6              LF     7,X'A6'
0646 1DA0              JP     RB
                *             SPECIAL CHARACTER, THESE MUST BE
                *             LOOKED UP IN THE ASCII TABLE
0647 F700       SPCHAR SFL    7
~~0648 F700              SFL    7~~
0649 F720              SFR    7          MULTIPLY BY 2
064A 1125              LT     ASCII
064B 8724              ADD    7,T,(L)    JUMP TO CORRECT£ASCII CHARACT
                *   THESE ARE SUBROUTINES USED BY THE I/O DRIVERS
                *  THIS CODE PROVIDES SUBROUTINE LINKAGE VIA THE
                *  JE AND RTN INSTRUCTIONS FOR CALLS MORE THAN 256
                *  WORDS AWAY
064C 2B58       ,DIX£  LF     11,RETURN
064D 1C69              JP     ID,IX
064E 2B58       DIX£   LF     11,RETURN
064F 1C6A              JP     IDIX
0650 2B58       ,DOX£  LF     11,RETURN
0651 1C85              JP     I,DOX
0652 2B58       DOX£   LF     11,RETURN
0653 1C89              JP     IDOX
0654 2B58       COX£   LF     11,RETURN
0655 1C73              JP     ICOX
0656 2BE1       W,OUT  LF     11,FETCH
0657 1C92              JP     I,OUT
0658 1000              NOP               FREE LOCATION
0659 27A1              LF     7,X'A1'
065A 1DA0              JP     RB
065B 27A4              LF     7,X'A4'
065C 1DA0              JP     RB
065D 27AA              LF     7,X'AA'
065E 1DA0              JP     RB
065F 27A9              LF     7,X'A9'
0660 1DA0              JP     RB
0661 27BB              LF     7,X'BB'
0662 1DA0              JP     RB
0663 27DD              LF     7,X'DD'
0664 1DA0              JP     RB
0665 27AD              LF     7,X'AD'
0666 1DA0              JP     RB
0667 27AF              LF     7,X'AF'
0668 1DA0              JP     RB
                ** THESE ROUTINES DO THE DEVICE INPUT/OUTPUT
                *  THEY EXPECT THE DEVICE ADDRESS AND FUNCTION CODE
                *  IN THE T REGISTER, AND DATA IN P7
                *  RETURN ADDRESS IN P11
                *** INPUT£A BYTE
```

```
0669 27FF          ID,IX£ LF     7,X'FF'
066A 7090          IDIX£  COX£   0         SET COXX, SEND COMMAND BYTE
066B 1000                 NOP
066C 1C6D                 JP     *+1       DELAY 3 CLOCK PULSES
066D 7080                 CIO    0         CLEAR I/O REG
066E 70E0          ,IDIX£ DIX£   0         SET DIXX
066F 1000                 NOP
0670 1C71                 JP     *+1
0671 7781                 CIO    7,(T)
0672 1CC1                 JP     GOBACK    RETURN
                   *** OUTPUT£COMMAND BYTE
0673 7090          ICOX£  COX£   0         SET COXX=OUTPUT COMMAND BYTE
0674 1000                 NOP
0675 1C76                 JP     *+1       DELAY 3 CLOCK PULSES
0676 7080                 CIO    0         RESET COXX
0677 1CC1                 JP     GOBACK    RETURN
0678 1000                 NOP              FREE LOCATION
0679 27A0                 LF     7,X'A0'
067A 1DA0                 JP     RB
067B 27AC                 LF     7,X'AC'
067C 1DA0                 JP     RB
067D 27A5                 LF     7,X'A5'
067E 1DA0                 JP     RB
067F 27DE                 LF     7,X'DE'
0680 1DA0                 JP     RB
0681 27BE                 LF     7,X'BE'
0682 1DA0                 JP     RB
0683 27BF                 LF     7,X'BF'
0684 1DA0                 JP     RB
                   ** THESE ROUTINES DO THE DEVICE INPUT/OUTPUT
                   *  THEY EXPECT THE DEVICE ADDRESS AND FUNCTION CODE
                   *  IN THE T REGISTER, AND DATA IN P7
                   *  RETURN ADDRESS IN P11
                   *** OUTPUT£A BYTE
0685 7090          I,DOX£ COX£   0         SET COXX=OUTPUT£COMMAND BYTE
0686 1C87                 JP     *+1       DELAT 3 CLOCK PULSES
0687 C701                 MOV    7,(T)     MOVE DATA TO T REG,
0688 7080                 CIO    0         CLEAR I/O REG
0689 70A0          IDOX£  DOX£   0         SET DOXX=OUTPUT DATA IN T REG,
068A 1000                 NOP
068B 1C8C                 JP     *+1       DELAY 3 CLOCK PULSES
068C 7080                 CIO    0         RESET COXX
068D 1CC1                 JP     GOBACK    RETURN
                   *****************
                   *****************
                   *****************
                   *  THIS ROUTINE WRITE OUT TO MEMORY THE UPDATED
                   *  CONCURRENT£I/O VALUES
```

```
                         *  N<-- ADDRESS OF THE COUNTER
                         *  P7<-- ADDRESS OF LSB OF C-I/O ADDRESS
                         *  P8<-- COUNTER VALUE
                         * P11<-- RETURN ADDRESS
                         *  P12<--MSB OF C-I/O ADDRESS
                         *  P13<--LSB OF C-I/O ADDRESS
                         *
                         *
068E 8D40                ,OUT   INC    13
068F 8C80                       ADD    12,L       ADJUST C-I/O ADDRESS
0690 1180                O,UT   LT     X'80'
0691 E120                       AND    1,T        CLEAR P1
0692 123F                I,OUT  LM     HICORE     LOAD MAR(MSB)
0693 A811                       WMF    8,(T)      WRITE COUNTER VALUE
0694 C703                       MOV    7,(N)      ADJUST MAR(LSB)
0695 AD11                       WMF    13,(T)     WRITE LSB OF C-I/O ADDRESS
0696 9743                       DEC    7,(N)      ADJUST MAR(LSB)
0697 AC11                       WMF    12,(T)     WRITE MSB OF C-I/O ADDRESS
0698 1CC1                       JP     GOBACK     RETURN
0699 27BA                       LF     7,X'BA'
069A 1DA0                       JP     RB
069B 27A3                       LF     7,X'A3'
069C 1DA0                       JP     RB
069D 27C0                       LF     7,X'C0'
069E 1DA0                       JP     RB
069F 27A7                       LF     7,X'A7'
06A0 1DA0                       JP     RB
06A1 27BD                       LF     7,X'BD'
06A2 1DA0                       JP     RB
06A3 27A2                       LF     7,X'A2'
06A4 1DA0                       JP     RB
                         ***********
                         ***********
                         ***********
                         ***********
                         *  THIS ROUTINE READS IN FROM MEMORY THE CONCURRENT
                         *  I/O VALUES
                         *  N<-- ADDRESS OF THE COUNTER
                         *  P7<-- ADDRESS OF THE LSB OF THE C-I/O ADDRESS
                         *  P11<--RETURN ADDRESS
                         *  P8<-- COUNTER VALUE
                         *  P12<-- MSB OF C-I/O ADDRESS
                         *  P13<-- LSB OF THE C-I/O ADDRESS
                         *
06A5 123F                ,IN    LM     HICORE     LOAD MAR(MSB)
06A6 A020                       RMH    0          READ IN COUNTER VALUE
06A7 C703                       MOV    7,(N)      ADJUST MAR(LSB)-DELAY
06A8 B820                       CPY    8,T        COPY COUNTER£INTO P8
```

```
06A9 A020              RMH    0          READ LSB OF C-I/O ADDRESS
06AA 9743              DEC    7,(N)      ADJUST MAR(LSB)-DELAY
06AB 8D20              CPY    13,T        COPY LSB OF C-I/O ADDRESS
06AC A020              RMH    0          READ MSB OF C-I/O ADDRESS
06AD 8743              INC    7,(N)      ADJUST MAR(LSB)-DELAY
06AE BC20              CPY    12,T       COPY MSB OF C-I/O INTO P12
                 ***********
                 ***********
                 ***********
                 *  THIS ROUTINE IS USED BY THE PRINTER,
                 *  CARD READER, AND TTY INTERRUPT£ROUTINES TO SKIP
                 *  SIGN AND GARBAGE BYTES
                 * P11<== RETURN ADDRESS
                 *  P12<== MSB OF C-I/O ADDRESS
                 *  P13<== LSB OD C-I/O ADDRESS
                 *
                 *
06AF 5D07        ISKIP TN     13,X'07'   IS THIS A SIGN BYTE ADDRESS
06B0 1CB8              JP     I6         YES
06B1 5D02              TN     13,X'02'   IS THIS A GARBAGE BYTE ADDRES
06B2 1CBE              JP     I8         NO
06B3 5D04              TN     13,X'04'   IS THIS A GARBAGE BYTE ADDRES
06B4 1CBE              JP     I8         NO
                 *  TIME TO SKIP 2 BYTES
06B5 1102              LT     X'02'      GARBAGE BYTES
06B6 8D20              ADD     13,T
06B7 8C80              ADD    12,L
06B8 5B80        I6    TN     11,X'80'   PRINTER OR TTY-OUT CALL
06B9 1CBD              JP     I7         YES
                 *  CARD READER OR TTY-IN CALLED THIS ROUTINE
                 *  MUST£ZERO SIGN BYTE
06BA CC02              MOV    12,(M)     LOAD MAR(MSB)
06BB AD13              WMF    13,(N)     WRITE, LOAD MAR(LSB)
06BC 1100              LT     X'00'      SET SIGN TO ZERO
06BD 8D40        I7    INC    13
06BE CC02        I8    MOV    12,(M)     LOAD MAR(MSB)
06BF CD03              MOV    13,(N)     LOAD MAR(LSB)
06C0 5B80              TN     11,X'80'   PRINTER OR CARD READER??
06C1 CB05        GOBACK MOV   11,(K)
06C2 27FF              LF     7,X'FF'
06C3 1C6E              JP     .IDIX
```

```
                         * REGISTER£SUPERVISOR
                         *
                         * THIS IS THE INDEX£SUPERVISOR ROUTINE
                         * IT TAKES CARE OF ROLLING THE INDEX£REGISTER
                         * WHICH IS NEEDED INTO S8, S9, S10
                         * S7 IS THE INDEX£MAP, IT CONTAINS THE NUMBER OF
                         * THE REGISTER CURRENTLY IN S8 = S10
                         *
06C4 1107                PAGE   LT    X'07'
06C5 ED29                       AND*  13,T,(T)  GET INDEX£# FROM OPCODE
06C6 1CC8                       JP    VIA
                         ********
                         *BOX£DD*
                         ********
06C7 CB01                VINDEX£MOV   11,(T)
                         ********
                         *BOX£EE*
                         ********
06C8 1080                VIA    SSF
06C9 BB20                       CPY   11,T
                         **Q*****
                         *            TEST FOR ILLEGAL INDEX£NUMBER
                         ********
06CA 6BF9                       CP    11,X'F9'
06CB 1CCD                       JP    *+2        INDEX£BETWEEN 0 & 6 = OK
06CC 06F9                       JE    ABORT      INDEX£> 6 ABORT USER RUN
                         ********
                         *BOX£FF*
                         ********
06CD 1107                       LT    X'07'   LOAD MASK
06CE E729                       AND*  7,T,(T)   PICK UP INDEX£MAP
06CF BD20                       CPY   13,T    SAVE INDEX£MAP
06D0 DB38                       XOR*  11,T,C    TEST EQUAL
06D1 4004                       TZ£   0,X'04'
06D2 1060                       RSP           REQUIRED INDEX£IS PRESENT
                         *
                         * THE REQUIRED INDEX£IS NOT IN THE HOME POSITION
                         *  6 STEPS MUST BE TAKEN
                         *
                         *  STEP 1 IS ANY INDEX£HOME? IF NOT GOTO STEP 4
                         *  STEP 2  COMPUTE CORE STORAGE ADDRESS OF THE INDEX
                         *  STEP 3  WRITE HALF CYCLE THIS INDEX£OUT TO CORE
                         *  STEP 4  COMPUTE THE CORE ADDRESS OF THE REQUIRED
                         *  STEP 5  READ HALF CYCLE THIS INDEX£INTO FILES S8 -
                         * STEP 6  UPDATE INDEX£MAP IN S7
                         *
                         ********
                         *BOX£HH*   STEP 1
```

```
                   ********
06D3 123F                   LM     HICORE
06D4 5D07                   TN     13,X'07'
06D5 1CE7                   JP     VKK      INDEX£MAP = 0
                   ********
                   *BOX£II*    STEP 2
                   ********
06D6 FD00                   SFL    13            MULTIPLY OLD INDEX£# BY 16
06D7 FD00                   SFL    13
06D8 FD00                   SFL    13
06D9 FD00                   SFL    13
06DA 1197                   LT     BASE          LOAD T WITH BASE ADDRESS
06DB 8D23                   ADD    13,T,(N)      COMPUTE ADDRESS OF SIGN
                   ********
                   *BOX£JJ*    STEP 3
                   ********
06DC A831                   WMH    8,(T)         WRITE OUT SIGN BYTE
06DD 3D07                   AF     13,X'07'      COMPUTE SIGN OF MSB
06DE AD33                   WMH    13,(N)        WRITE MSB MOVE ADDRESS TO N
06DF C901                   MOV    9,(T)         MOVE WRITE OPERAND TO T
06E0 3D01                   AF     13,X'01'      COMPUTE ADDRESS OF LSB
06E1 AD33                   WMH    13,(N)        WRITE LSB MOVE ADDRESS TO N
06E2 CA01                   MOV    10,(T)        MOVE WRITE OPERAND TO T
                   ********
                   *BOX£NN*    STEP 6
                   ********
06E3 11F8          VNN      LT     X'F8'
06E4 E720                   AND    7,T
06E5 CB01                   MOV    11,(T)
06E6 C720                   LOR    7,T
06E7 5B07          VKK      TN     11,X'07'
06E8 1060                   RSP             INDEX£# = 0
                   ********
                   *BOX£KK*    STEP 4
                   ********
                   ********
                   *BOX£LL*
                   ********
06E9 FB00                   SFL    11            MULTIPLY NEW INDEX£# BY 16
06EA FB00                   SFL    11
06EB FB00                   SFL    11
06EC FB00                   SFL    11
06ED 1197                   LT     BASE          LOAD T WITH BASE ADDRESS
06EE 8B23                   ADD    11,T,(N)      COMPUTE ADDRESS OF SIGN
                   ********
                   *BOX£MM*    STEP 5
                   ********
06EF A020                   RMH    0             READ IN SIGN BYTE
```

```
06F0 3807               AF    11,X'07'  COMPUTE ADDRESS OF MSB
06F1 B820               CPY   8,T       COPY SIGN OF INDEX£INTO S8
06F2 AB23               RMH   11,(N)    READ MSB MOVE ADDRESS TO N
06F3 3801               AF    11,X'01'  COMPUTE ADDRESS OF LSB
06F4 B920               CPY   9,T       COPY MSB INTO 9
06F5 AB23               RMH   11,(N)    READ LSB MOVE ADDRESS TO N
~~06F6 1000~~               ~~NOP~~             ~~DELAY~~
06F7 BA20               CPY   10,T      COPY LSB INTO S10
              ********
              *BOX£00*
              *******
06F8 1060               RSP
```

```
                    **  ABORT£ROUTINE
06F9 293F           ABORT  LF    9,HICORE
06FA 2AB0                  LF    10,ABTADD
06FB 28B9                  LF    8,ABSAVE
06FC 07DA                  JE    ERROR
```

```
                        *  INTERRUPT£ROUTINE
                        ********
                        *BOX£A *
                        ********
06FD 1800               INT    ILS              INHIBIT L SAVE UNTIL RETURN OCC
06FE 1600                      LU     X'00'
06FF 1040                      SPF
0700 5008                      TN     0,X'08' TEST FOR CONCURRENT£I/O REQUEST
0701 1D08                      JP     IHG
                        ********
                        *BOX£B *                CONCURRENT£REQUEST£HAS OCCURED
                        ********
0702 70C0                      CAK    0         ACKNOWLEDGE REQUEST
0703 27FF                      LF     7,X'FF'
0704 1D05                      JP     IH1
                        *******
                        *BOX£C *
                        *******
0705 7780               IH1    CIO    7         RESET, AND INPUT BUS WITH P7
                        ********
                        *BOX£D *
                        ********
0706 1180                      LT     X'80'
0707 E120                      AND    1,T         CLEAR P1
0708 5702                      TN     7,X'02'     INPUT OR OUTPUT
0709 0798                      JE     READER
070A 075A                      JE     PRNTER
                        ********
                        * BOX£G*       TEST FOR EXTERNAL INTERRUPT=ERROR
                        ********
070B 5080               IHG    TN     0,X'80'     TEST EXTERNAL FLAG
070C 071D                      JE     IHK         NO EXTERNAL INTERRUPT
                        *********
                        *BOX£H *       EXTERNAL INTERRUPT£HAS OCCURRED
                        ********
070D 70D0                      IAK    0           ACKNOWLEDGE INTERRUPT
070E 27FF                      LF     7,X'FF'     P7<==ALL ONES
070F 1D10                      JP     IH2         DELAY
0710 7780               IH2    CIO    7           RESET, INPUT BUS ANDED WITH P7
                        ********
                        * BOX£I*
                        ********
                        *              FIND OUT WHICH DEVICE CAUSED INTERRUPT
0711 1128                      LT     X'28'       DISK ADDRESS * 2
0712 D738                      XOR*   7,T,C       WAS IT THE DISK
0713 4004                      TZ£    0,X'04'
0714 07C4                      JE     IDSK        DISK INTERRUPT
0715 1108                      LT     X'08'       CARD READER ADDRESS * 2
```

```
0716 D738                  XOR*  7,T,C
0717 1D1D                  JP    IHK       MUST BE TTY OR MAG TAPE=IGNOR
0718 1124                  LT    X'24'     INPUT STATUS BYTE
0719 281B                  LF    11,*+2    LOAD RETURN ADDRESS
071A 1C69                  JP    ID,IX£    GET CARD READER STATUS
071B 5704                  TN    7,X'04'   IS THIS A HOPPER EMPTY INT,
~~071C 07B1~~                  ~~JE~~    IREADR    NO REAL ERROR
                    ********
                    *BOX£K *
                    ********
071D 5010           IHK    TN    0,X'10'  TEST FOR INTERNAL INTERRUPTS
                    ********
                    *BOX£L *
                    ********
071E 1060                  RSP             RETURN
                    ********
                    *BOX£M *
                    ********
071F 7B40                  EIS   11       ENTER INTERNAL STATUS
                    ********
                    *BOX£N *
                    ********
0720 5B04                  TN    11,X'04'   REAL TIME CLOCK INTERRUPT?
0721 1D48                  JP    IHP
                    ********
                    *BOX£O *        ADJUST CLOCK
                    ********
0722 123F                  LM    HICORE
0723 28FD                  LF    8,,CLOCK  LOAD P8 WITH A(C1)
0724 A823                  RMH   8,(N)     READ IN C1
0725 9843                  DEC   8,(N)     DECREMENT£P8 TO A(C2)=DELAY
0726 BD20                  CPY   13,T      COPY C1 INTO P13
0727 ADE0                  RMH   13,I      READ C2, INCREMENT£C1
0728 8843                  INC   8,(N)     INCREMENT£P8 TO A(C1)=DELAY
0729 BC20                  CPY   12,T      COPY C2 INTO P12
072A AD31                  WMH   13,(T)    WRITE OUT UPDATED C1
072B 9843                  DEC   8,(N)     DECREMENT£P8 TO A(C2)=DELAY
072C ACB1                  WMH   12,L,(T)  WRITE OUT UPDATED C2
072D 9843                  DEC   8,(N)     DECREMENT£P8 TO A(C3)=DELAY
072E A020                  RMH   0         READ C3
072F 9843                  DEC   8,(N)     DELAY
0730 BD20                  CPY   13,T      COPY C3 INTO P13
0731 8D81                  ADD   13,L,(T)  UPDATE C3
0732 A8F3                  WMH   8,I,(N)   WRITE OUT C3
                    * TTY INTERRUPT£ROUTINE
                    *            THIS IS THE TELETYPE ROUTINE
                    *            NO INTERRUPTS ARE AVAILABLE ON THE
                    *            TTY, THUS THE I/O IS RUN BY INTERRUPTS
```

```
                   *                FROM THE REAL TIME CLOCK,
                   *
0733 2877          ITTY   LF    8,TSTAT
0734 A803                 RMF   8,(N)     READ IN INTERNAL STATUS
0735 9843                 DEC   8,(N)     LOAD TCNT-DELAY
0736 B820                 CPY   8,T       COPY STATUS
0737 4801                 TZ    8,X'01'   TEST BUSY BIT
0738 1D48                 JP    IHP       NO TTY TRANSFER IN PROGRESS
                   *            GET STATUS FROM TTY CONTROLLER
0739 1120                 LT    X'20'     LOAD COMMAND BYTE
073A 283C                 LF    11,*+2    LOAD RETURN ADDRESS
073B 1C69                 JP    ID,IX£    INPUT STATUS BYTE
073C E838                 AND*   8,T,C    TEST STATUS
073D 4004                 TZ£   0,X'04'   TEST IF READY
073E 1D48                 JP    IHP       NOT READY, CONTINUE
                   *
                   *            CONTROLLER READY TO DO I/O
                   *            SEND COMMAND BYTE TO DO I/O
                   *
073F 1100                 LT    X'00'     LOAD COMMAND BYTE
0740 2842                 LF    11,*+2    LOAD RETURN ADDR5SS
0741 1C73                 JP    ICOX£     REQUEST£I/O
                   *
                   *            DETERMINE IF INPUT£OR OUTPUT
                   *
0742 277F                 LF    7,TLSB
0743 C801                 MOV   8,(T)
0744 C120                 LOR   1,T       SAVE STATUS IN P1
0745 5802                 TN    8,X'02'
0746 1D5C                 JP    T,OUT     OUTPUT
0747 1D9A                 JP    T,IN      INPUT
                   ********
                   *BOX£P *
                   ********
0748 5801          IHP    TN    11,X'01'  PANNEL INTERRUPTS
0749 1D4C                 JP    IHR
                   ********
                   *BOX£Q *
                   ********
074A 1600                 LU    X'00'
074B 0600                 JE    GO
                   ********
                   *BOX£R *
                   ********
074C 5B40          IHR    TN    11,X'40'  STEP INTERRUPT
                   ********
                   *BOX£T *
                   ********
```

```
                         *********
                         *BOX£S *
074D 1D57                      JP     IHU
                         ********
                         *                  DIVIDE LOCATION COUNTER BY 8 TO GET
                         *                    MIX£ADDRESS, THEN MOVE THIS ADDRESS
                         *                    TO THE DATA BUS SO IT CAN BE DISPLAYE
074E FE29                STEP  SFR*   14,(T)
074F BC20                      CPY    12,T
0750 FFA9                      SFR*   15,L,(T)
0751 BD20                      CPY    13,T
0752 FC20                      SFR    12
0753 FDA0                      SFR    13,L
0754 FC22                      SFR    12,(M)
0755 FDA3                      SFR    13,L,(N)
0756 1780                      HLT
                         *********
                         * BOX£U *
                         *********
0757 5B02                IHU   TN     11,X'02'
0758 1060                RETURN RSP
0759 07C4                      JE     IDSK
```

```
                     * PRINTER INTERRUPT£ROUTINE
                     ********
                     *BOX£H *
                     ********
075A 1397            PRNTER LN    PCNT
075B 279F                   LF    7,PLSB
                     *            THIS ROUTINE IS COMMON TO BOTH THE
                     *            PRINTER AND THE TTY-OUT
075C 285E            T.OUT  LF    11,*+2      LOAD RETURN ADDRESS
075D 1CA5                   JP    .IN         GET C-I/O COUNTERS &
                     *                        ADJUST C-I/O ADDRESSES
                     ********
                     *BOX£G *
                     ********
075E 587F                   TN    8,X'7F'     TEST IF COUNT = ZERO
075F 1D79                   JP    PRK         COUNT = ZERO
0760 A840                   RMF   8,D         GET DATA, DECREMENT£COUNTER
                     ********
                     * BOX£J*
                     ********
0761 2863                   LF    11,*+2
0762 1C89                   JP    IDOX£       OUTPUT DATA BYTE
0763 B820                   CPY   11,T        SAVE DATA BYTE
0764 410F                   TZ£   1,X'0F'
0765 1D6A                   JP    T1          TTY R6UTINE
                     ********
                     *BOX£L *
                     ********
0766 1397                   LN    PCNT
0767 279F                   LF    7,PLSB
0768 280B            T.P    LF    11,IHG
0769 1C8E                   JP    .OUT
                     ********
                     *BOX£M *
                     ********
076A 118D            T1     LT    X'8D'       LOAD CARRIAGE RETURN
076B DB30                   XOR   11,T,C      TEST IF DATA WAS A C/R
076C 4004                   TZ£   0,X'04'
076D 1D80                   JP    C.R         DATA BYTE WAS C/R!!
                     ********
                     * BOX£N*
                     ********
                     *            TEST REFLECT FLAG
076E 5108                   TN    1,X'08'
076F 1D74                   JP    T2          NOT ON WRITE OUT COUNTERS
                     ********
                     *BOX£O *     REFLECT FLAG IS ON,THIS MEANS THAT
                     *            A TTY-IN OPERATION IS IN PROGRESS
```

```
                    *            THIS IS THE SECOND PHASE OF HANDLING
                    *            ONE DATA BYTE
                    *            MUST FIXETHE STATUS TO ALLOW THE
                    *            NEXT OPERATION TO BE A TTY=IN,
                    ********
0770 1102                  LT    X'02'
0771 1377           R,F    LN    TSTAT
0772 123F                  LM    HICORE
0773 A010                  WMF   0         WRITE OUT STATUS
                    ********
                    *BOXEP *
                    ********
0774 58FF           T2     TN    8,X'FF'
0775 1D79                  JP    PRK       COUNT = ZERO STOP TRANSFER
0776 1376                  LN    TCNT
0777 277F                  LF    7,TLSB
0778 1D68                  JP    T,P       WRITE OUT C=I/O COUNTERS
                    ********
                    *BOXEK *
                    ********
0779 510F           PRK    TN    1,X'0F'   TTY??
077A 1D85                  JP    ,PRK      PRINTER
077B 4110                  TZE   1,X'10'   LINE FEED FLAG ON??
077C 1D85                  JP    ,PRK      FALG IN ON
                    ********
                    *BOXEQ *
                    ********
077D 118D                  LT    X'8D'
077E 2B80                  LF    11,*+2
077F 1C89                  JP    IDOXE     SEND C/R
                    ********
                    *BOXER *
                    ********
0780 1114           C,R    LT    X'14'     TURN LINE FEED FLAG ON
0781 2880                  LF    8,X'80'
0782 1D71                  JP    R,F
                    *********
                    *BOXEV *
                    *********
0783 1101           TLF    LT    X'01'     STOP TYY
0784 1D71                  JP    R,F
                    ********
                    * BOXES*
                    ********
0785 118A           ,PRK   LT    X'8A'     SEND LINE FEED
0786 2B88                  LF    11,*+2
0787 1C89                  JP    IDOX
                    ********
```

```
                   *BOX£T *
                   ********
0788 410F                 TZ    1,X'0F'
0789 1D83                 JP    TLF        STOP TTY TRANSFER
                   ********
                   *BOX£U*
                   ********
078A 1180                 LT    X'80'
078B E120                 AND   1,T
                   * IPRNTR ROUTINE
                   * THIS IS THE EXTERNAL INTERRUPT£ROUTINE FOR THE
                   * PRINTER, IT STORES THE STATUS BYTE IN ADDRESS 7FFB
078C 1125          IPRNTR LT    X'25'      INPUT THE STATUS BYTE
078D 288F                 LF    11,*+2
078E 1C69                 JP    ID,IX£     GET STATUS IN P7 AND T
078F 13F2                 LN    PSTST
0790 123F                 LM    HICORE
0791 A711                 WMP   7,(T)      WRITE OUT STATUS
                   * NOW SEND DISCONNECT
0792 11A5                 LT    X'A5'
0793 2895                 LF    11,*+2
0794 1C73                 JP    ICOX£      SEND C6MMAND BYTE
                   * NOW SEND DISARM TO PRINTER CONTROLLER
0795 1185                 LT    X'85'
0796 280B                 LF    11,IHG     SET RETURN TO INT LABEL
0797 1C73                 JP    ICOX£      SEND COMMAND BYTE
```

```
                         *CARD READER INTERRUPT ROUTINE
                         *
                         *********
                         *BOX£B *
                         *********
0798 1387                READER LN     CCNT
0799-278F                       LF-    7,CLSB
                         *             READ IN C-I/O COUNTERS
                         * MUST£SKIP SIGN BYTE AND TWO HI ORDER BYTES OF EACH
                         *             GET DATA FROM DEVICE
                         *   THIS CODE IS COMMON TO THE TTY & CARD READER
079A 2B9C                T,IN   LF     11,*+2
079B 1CA5                       JP     ,IN
                         ********
                         *BOX£A *
                         ********
079C 510F                       TN     1,X'0F'   TTY OR CARD READER
                         *********
                         *             CONVERT EBCDIC TO ASCII
                         *********
079D 1C27                       JP     EBCDIC    CONVERT£CHARACTERS TO ASCII
                         *             MUST OR HIGH ORDER 1 ONTO EACH TTY
                         *             CHARACTER TO GET THE CORRECT ASCII CODE
079E 1180                       LT     X'80'
079F C720                       LOR    7,T
                         *********
                         *BOX£B *
                         ********
07A0 A711                RB     WMF    7,(T)     WRITE OUT DATA BYTE
07A1 2B0B                       LF     11,IHG
07A2 510F                       TN     1,X'0F'   TTY OR CARD READER
07A3 1DAB                       JP     RD        CARD READER
07A4 1377                       LN     TSTAT
07A5 123F                       LM     HICORE
07A6 A010                       WMF    0
07A7 110C                       LT     X'0C'     CHANGE STATUS TO REFLECTION
07A8 277F                       LF     7,TLSB
07A9 1376                       LN     TCNT
07AA 1C90                       JP     O,UT
                         ********
                         *BOX£D *
                         ********
07AB 9840                RD     DEC    8
07AC 58FF                       TN     8,X'FF'
07AD 07B1                       JE     IREADR
07AE 1387                       LN     CCNT
07AF 278F                       LF     7,CLSB
07B0 1C8E                       JP     ,OUT
```

```
07B0 1C8E          * IREDR ROUTINE ****************************************
                   * THIS IS THE EXTERNAL INTERRUPT£ROUTINE FOR
                   * THE CARD READER, IT STORES THE STATUS BYTE IN
                   * ADDRESS 7FFC
07B1 1124          IREADR LT    X'24'      INPUT STATUS BYTE
07B2 2BB4                 LF    11,*+2
07B3 1C69                 JP    ID,IX£     GET CARD READER STATUS
07B4 13F3                 LN    CSTST
07B5 123F                 LM    HICORE
07B6 A010                 WMF   0          WRITE STATUS OUT TO MEMORY
                   * NOW SEND DISCONNECT£TO READER CONTROLLER
07B7 1184                 LT    X'84'
07B8 2BBA                 LF    11,*+2
07B9 1C73                 JP    ICOX
                   * NOW SEND DISARM
07BA 11A4                 LT    X'A4'
07BB 2BBD                 LF    11,*+2
07BC 1C73                 JP    ICOX
07BD 1118                 LT    X'18'      TEST FOR£ERRORS
07BE E730                 AND   7,T,C
07BF 4004                 TZ    0,X'04'    TEST FOR ERROR
07C0 1D0B                 JP    IHG        NO ERROR
07C1 28D8                 LF    8,CISAVE
07C2 2AD0                 LF    10,CIADD
07C3 07DA                 JE    ERROR
```

```
                     * IDISK ROUTINE
                     * THIS IS THE EXTERNAL INTERRUPT£ROUTINE FOR THE
                     * DISK, IT STORES THE MAJOR STATUS BYTE IN ADDRESS
                     * 7FDFA AND STORES THE DIAGONISTIC STATUS BYTE IN
                     * ADDRESS 7FF9
07C4 1114            IDSK    LT     X'14'       INPUT MAJOR STATUS
07C5 28C7                    LF     11,*+2
07C6 1C69                    JP     ID,IX£      GET MAJOR DISK STATUS
07C7 B820                    CPY    8,T         COPY MAJOR STATUS INTO P8
07C8 1134                    LT     X'34'       INPUT DIAGONOSTIC STATUS
07C9 28C8                    LF     11,*+2
07CA 1C69                    JP     ID,IX£      GET DIAGONISTIC STATUS IN P7
07CB 123F                    LM     HICORE
07CC 2CF4                    LF     12,DSTAT
07CD AC13                    WMF    12,(N)      WRITE DIAGONSITIC STATUS OUT
07CE 8C43                    INC    12,(N)      ADJUST MAR(LSB)=DELAY
07CF A811                    WMF    8,(T)       WRITE OUT MAJOR STATUS
                     * NOW SEND DISARM
07D0 1114                    LT     X'14'       OUTPUT COMMAND BYTE
07D1 2720                    LF     7,X'20'
07D2 28D4                    LF     11,*+2
07D3 1C85                    JP     I,DOX
07D4 1114                    LT     X'14'
07D5 E830                    AND    8,T,C
07D6 4004                    TZ£    0,X'04'
07D7 071D                    JE     IHK
07D8 2AE0                    LF     10,DADD
07D9 28E8                    LF     8,DSAVE
                     *********
                     * ERROR£ROUTINE
                     *********
07DA 123F            ERROR   LM     HICORE
07DB 293F                    LF     9,HICORE
07DC A813                    WMF    8,(N)
07DD CE01                    MOV    14,(T)
07DE A8D3                    WMF    8,I,(N)
07DF CF01                    MOV    15,(T)
07E0 0CDC                    JE     JS
```

```
                   * FETCH ROUTINE
                   * THIS ROUTINE FETCHES THE NEXT£INSTRUCTION FROM THE
                   * FOUND IN THE INSTRUCTION COUNTER
                   *
                   ********
                   *BOX£A1*
                   ********
07E1 06F0          FETCH  JE    INT     GO CHECK INTERRUPTS
                   ********
                   *BOX£A *
                   ********
                   ********
                   *BOX£B *
                   ********
07E2 27B7                 LF    7,X'B7'  LOAD COUNTER£AND MASK
07E3 CF03                 MOV   15,(N)
07E4 AE02                 RMF   14,(M)
07E5 1DE8                 JP    FC+2
                   ********
                   *BOX£C *
                   ********
07E6 8F43          FC     INC   15,(N)  ADJUST MEMORY ADDRESS
07E7 AE82                 RMF   14,L,(M)  READ A BYTE
                   ********
                   *BOX£D *
                   ********
07E8 8746                 INC   7,(U)   INCREMENT£COUNTER
07E9 0820                 EOT   8,T     COPY T INTO CORRECT£FILE REGIST
                   ********
                   *BOX£E *
                   ********
07EA 6743                 CP    7,X'43'    TEST FOR END OF READ LOOP
07EB 1DE6                 JP    FC
                   ********
                   *BOX£F *
                   ********
07EC 1103                 LT    X'03'
07ED 8F20                 ADD   15,T
07EE 8E80                 ADD   14,L
07EF 1600                 LU    X'00'     CLEAR U REGISTER
```

```
                    * ADDRESSING ROUTINE
                    *
                    * THIS ROUTINE COMPUTES THE EFFECTIVE MIXAL MEMORY A
                    * FOR THE INSTRUCTION CURRENTLY HELD IN THE INSTRUCT
                    * COUNTER
                    *
                    *********
                    *BOX£I *            TEST FOR INDEXING
                    ********
07F0 5B07           ADI     TN     11,X'07'
07F1 0811                   JE     ADJ
                    *
                    * INDEXING HAS OCCURED
                    *
                    ********
                    *BOX£K *
                    ********
07F2 1600           ADK     LU     X'00'
07F3 06C7                   JE     VINDEX£   JUMP TO INDEXING ROUTINE
07F4 1080                   SSF
                    *
                    * THE INDEX£REGISTER REFERENCED IN P11 IS NOW AVAILA
                    * IN S8,S9,S10
                    ********
                    *BOX£L *         TEST SIGN OF INDEX
                    ********
                    *
                    *              TEST IF SIGNS ARE SAME
                    *
07F5 1680                   LU     X'80'     ASSUME SIGNS ARE SAME-ADD
07F6 C801                   MOV    8,(T)
07F7 1040                   SPF
07F8 D838                   XOR*   8,T,C     X-/R SIGNS ; SET CONDITION FL
07F9 5004                   TN     0,X'04'   TEST FOR ZERO RESULT
07FA 1690                   LU     X'90'     SIGNS DIFFERENT£SUBTRACT
07FB 1080                   SSF
07FC CA01                   MOV    10,(T)
07FD 1040                   SPF
07FE 8A27                   ADD    10,T,(S)
07FF 1080                   SSF
0800 C901                   MOV    9,(T)
0801 1040                   SPF
0802 89A7                   ADD    9,T,L,(S)
0803 FB00                   SFL    11        SHIFT LINK BIT INTO P11-INDEX
0804 5004                   TN     0,X'04'   WERE SIGNS DIFFERENT???
0805 1409                   JP     ADL       SINGS DIFFERENT
                    *              SIGNS SAME TEST FOR OVERFLOW
0806 4B01                   TZ£    11,X'01'  LINK = 1 ==> OVERFLOW
```

```
0807 06F9                JE    ABORT      BAD ADDRESS
0808 1411                JP    ADJ
                 *
                 *              SIGNS WERE DIFFERENT, MUST GET ADDRESS
                 *             BACK INTO SIGN + MAGNITUDE FORMAT
0809 4B01_       ADL_    TZ_   11,X'01'  TEST SIGN OF RESULT
080A 1411                JP    ADJ       RESULT IN TRUE FORM
                 *              RESULT IN COMPLEMENT£FORM
080B 1180                LT    X'80'
080C D820                XOR   8,T       FLIP SIGN
080D D960                XOR   9,T,F
080E DA60                XOR   10,T,F
080F 8A40                INC   10
0810 8980                ADD   9,L       COMPLEMENT£ADDRESS
                 *
                 * THIS ROUTINE FORMS THE MICRODATA EFFECTIVE MEMORY
                 ********
                 *BOX£P *
                 ********
                 *       ILLEGAL INSTRUCTION TRAP
0811 6DBF        ADJ     CP    13,X'BF'
0812 1414                JP    *+2
0813 06F9                JE    ABORT     ILLEGAL INSTRUCTION
0814 6DFA                CP    13,X'FA'
0815 141C                JP    ,ADJ      OPCODES 00-05
0816 6DF9                CP    13,X'F9'
0817 1422                JP    ,DECOD    OPCODE 06 SHIFT COMMAND
0818 6DD0                CP    13,X'D0'
0819 141C                JP    ,ADJ      OPCODES 00 - 2F
081A 6DC8                CP    13,X'C8'
081B 1422                JP    ,DECOD    OPCODE 30 - 37
081C FA00        ,ADJ    SFL   10
081D F980                SFL   9,L
081E FA00                SFL   10
081F F980                SFL   9,L
0820 FA03                SFL   10,(N)
0821 F982                SFL   9,L,(M)
0822 4980        ,DECOD  TZ£   9,X'80'
0823 06F9                JE    ABORT     ILLEGAL OPERAND ADDRESS
```

```
                * DECODE ROUTINE
0824 1600             LU    X'00'
0825 CD01             MOV   13,(T)
0826 B720             CPY   7,T
0827 F720             SFR   7
0828 F720             SFR   7
0829 F720             SFR   7
                *******
                *BOXEQ *     DECODE
                *******
082A 112C             LT    DECODE
082B 872C             ADD*  7,T,(L)
                *******
                *BOXER *
                *******
082C 083C       DECODE JE   MISC
                *******
                *BOXES *
                *******
082D 0827             JE    LOAD
                *******
                *BOXET *
                *******
082E 0827             JE    LOAD
                *******
                *BOXEU *
                *******
082F 0880             JE    STORE
                *******
                *BOXEY1*    JUMP TO BOXEY
                *******
0830 0837             JE    DY1
                *******
                *BOXEV *
                *******
0831 0C8D             JE    JUMP
                *******
                *BOXEW *
                *******
0832 0CF0             JE    ENTER
                *******
                *BOXEX *
                *******
0833 0D64             JE    COMP
                *     TRAP BACK TO THE ROM ON OPCODE 40 HEX,
0834 2802             LF    8,TOS       LOAD MSB OF TOS ADDRESS
0835 297B             LF    9,.TOS      LOAD LSB OF TOS ADDRESS
0836 0104             DC    X'0104'     JUMP TO THE ROM FETCH ROUTINE
```

```
0836 0104          ********
                   *BOX£Y *
                   ********
0837 6DDE          DY1     CP     13,X'DE'
                   ********
                   *BOX£Z *
                   ********
0838 0B80                  JE     STORE
                   ********
                   *BOX£AA*
                   ********
0839 6DD9                  CP     13,X'D9'
                   ********
                   *BOX£BB*
                   ********
083A 084D                  JE     INPUT
                   ********
                   *BOX£CC*
                   ********
083B 0C8D                  JE     JUMP
                   ********
                   *BOX£PP*
                   ********
083C 113E          MISC    LT     OP1
083D 8D2C                  ADD*   13,T,(L)
                   ********
                   *BOX£RR*       MIXAL NOP
                   ********
083E 07E1          OP1     JE     FETCH
                   ********
                   *BOX£SS*
                   ********
083F 0877                  JE     ADD
0840 0877                  JE     ADD
                   ********
                   *BOX£TT*
                   ********
0841 08AD                  JE     MUL
                   ********
                   *BOX£UU*
                   ********
0842 091C                  JE     DIV
                   ********
                   *BOX£VV*
                   ********
0843 0846                  JE     NCH
                   ********
                   *BOX£WW*
```

```
                    ********
0844 0AA0                  JE     SHIFT
                    ********
                    *BOX£XX*
                    ********
0845 0AF6                  JE     MOVE
                    *********
                    *BOX£YY*
                    ********
0846 1148           NCH    LT     DC5
                    ********
                    *BOX£ZZ*
                    ********
0847 8C2C                  ADD*   12,T,(L)
                    ********
                    * BOX£A1*
                    ********
0848 09D1           DC5    JE     NUM
                    ********
                    *BOX£B1*
                    ********
0849 0A28                  JE     CHAR
                    ********
                    *BOX£C1*
                    ********
084A 06FD                  JE     INT        TEST INTERRUPTS BEFORE HALTIN
084B 074E                  JE     STEP
                    ********
                    *BOX£D1*
                    ********
084C 07E1                  JE     FETCH
                    ********
                    *BOX£E1*
                    ********
084D 1122           INPUT  LT     X'22'
                    *       TEST FOR ILLEGAL DEVICE CODES, THE ONLY
                    *             DEVICES ALLOWED ARE AS FOLLOWS:
                    *               0E = DISK
                    *               0F = DISK
                    *               10 = CARD READER
                    *               12 = PRINTER
                    *               13 = TTY
084E 6CF2                  CP     12,X'F2'
084F 06F9                  JE     ABORT      ILLEGAL CODES 0 = D
0850 6CEF                  CP     12,X'EF'
0851 1457                  JP     *+6        CODES E = 10
0852 6CEE                  CP     12,X'EE'
0853 06F9                  JE     ABORT      CODE 11
```

```
0854 6CEC                    CP     12,X'EC'
0855 1457                    JP     *+2
0856 06F9                    JE     ABORT     CODES  > 13
0857 9D29                    SBT*   13,T,(T)
0858 275A                    LF     7,OP2
0859 872C                    ADD*   7,T,(L)
085A 0BD5              OP2   JE     JBUS
085B 0BF9                    JE     IOC
085C 0C03                    JE     IN
085D 0C1B                    JE     OUT
085E 0BD5                    JE     JRED
```

```
                    *                  THIS ROUTINE PUTS THE LEFT HALF OF
                    *                  THE F SPECIFICATION IN P11
                    *                  AND PUTS THE RIGHT£HALF IN P12
085F 1107           L,R     LT         X'07'
0860 EC29                   AND*       12,T,(T)
0861 BB20                   CPY        11,T
0862 FC20                   SFR        12
0863 FC20                   SFR        12
0864 FC21                   SFR        12,(T)
                    *                  TEST FOR ILLEGAL F FIELD VALUES
0865 6BFA                   CP         11,X'FA'
0866 1468                   JP         *+2
0867 06F9                   JE         ABORT
0868 6CFA                   CP         12,X'FA'
0869 146B                   JP         *+2
086A 06F9                   JE         ABORT
086B 1020                   RTN
```

```
                         *    ZERO OUT FILES SUB ROUTINE
                         *            THIS ROUTINE IS USED BY THE ENTER,
                         *            DIVIDE AND MULTIPLY ROUTINES TO
                         *            ZERO OUT CONSECUTIVE FILES
086C BB00                Z11    ZOF   11
086D BA00                       ZOF   10
086E B900                       ZOF   9
086F B800                       ZOF   8
0870 B700                       ZOF   7
0871 B600                Z6     ZOF   6
0872 B500                       ZOF   5
0873 B400                Z4     ZOF   4
0874 B300                       ZOF   3
0875 B200                       ZOF   2
0876 1020                       RTN
```

```
                     * ADD AND SUBTRACT ROUTINE    ***********************
                     * THIS IS THE ADD AND SUBTRACT£ROUTINES
                     * OPCODES 01 & 02
                     *********
                     * BOX£A *
                     *********
0877 085F            ADD     JE    L,R        SEPERATE LEFT AND RIGHT FIELD
0878 9829                    SBT*  11,T,(T)
0879 8760                    CPY   7,I,T
                     *******
                     * BOX£B *
                     *********
087A 4C07                    TZ    12,X'07'   S9GN REQUIRED ??
087B 149A                    JP    AE         NO SIGN REQUIRED
                     *********
                     * BOX£C *
                     *********
087C CA03                    MOV   10,(N)
087D A902                    RMF   9,(M)
                     *********
                     * BOX£D *
                     *********
087E 9740                    DEC   7
087F B820                    CPY   8,T
                     *********
                     * BOX£F *
                     *********
0880 CB01            AF      MOV   11,(T)
0881 8A23                    ADD   10,T,(N)
0882 8982                    ADD   9,L,(M)
                     *********
                     * BOX£G *
                     *********
0883 1180                    LT    X'80'
0884 5D01                    TN    13,X'01'   TEST OPCODE
                     *********
                     * BOX£H *                SUBTRACT£COMMAND
                     *********
0885 D820                    XOR   8,T        FLIP SIGN BIT OF M
                     *********
                     * BOX£I *
                     *********
0886 C801                    MOV   8,(T)
0887 D138                    XOR*  1,C,T      EXCLUSIVE OR SIGN BITS
0888 2D06                    LF    13,X'06'
                     *********
                     * BOX£J *
                     *********
```

```
0889 5004                 TN    0,X'04'   TEST IF SIGNS SAME
                   *********
                   * BOX£K *
                   *********
088A 3D10                 AF    13,X'10'  SIGNS NOT SAME, SUBTRACT
                   *********
                   * BOX£L *
                   *********
088B F128                 SFR*  1         CLEAR LINK REGISTER TO 0
088C 5707                 TN    7,X'07'   TEST P7 = 0
088D 149C                 JP    AR
088E AD06                 RMF   13,(U)    U<==P13, READ 1ST BYTE
088F 37FF                 AF    7,X'FF'   DECREMENT£P7 BY 1
0890 8037                 ADD   0,T,C,(S) DO FIRST£ADD
                   *********
                   * BOX£M *
                   *********
0891 5707          AM     TN    7,X'07'
0892 149C                 JP    AR
                   *********
                   * BOX£N *
                   *********
0893 3DFF                 AF    13,X'FF'
                   *********
                   * BOX£O *
                   *********
0894 9A43                 DEC   10,(N)
0895 9982                 SBT   9,L,(M)
0896 AD06                 RMF   13,(U)
0897 37FF                 AF    7,X'FF'
                   *********
                   * BOX£P *
                   *********
0898 80B7                 ADD   0,T,L,C,(S)
0899 1491                 JP    AM
                   *********
                   * BOX£E *
                   *********
089A B800          AE     ZOF   8
089B 1480                 JP    AF
                   *********
                   * BOX£R *
                   *********
089C 3DFF          AR     AF    13,X'FF'  SUBTRACT£1-FROM P13
089D 5D06                 TN    13,X'06'  TEST P13 > 1
089E 14A3                 JP    AS
                   *********
                   * BOX£Y *
                   *********
089F CD06                 MOV   13,(U)
08A0 8097                 ADD   0,L,C,(S)
08A1 3DFF                 AF    13,X'FF'
08A2 149D                 JP    AR+1
                   *********
                   * BOXEU *
                   *********
08A3 FD80          AS     SFL   13,L      SAVE LINK BIT FOR TESTING
08A4 1600                 LU    X'00'
08A5 4D20                 TZE   13,X'20'  WERE SIGNS SAME ???
08A6 14AA                 JP    AV        SIGNS NOT SAME
                   *********
                   * BOXES *
                   *********
08A7 4D01                 TZ    13,X'01'  LINK = 1 ==> OVERFLOW
                   *********
                   * BOXET *    OVERFLOW HAS OCCURED
                   *********
08A8 0D53                 JE    OVERFL    GO SET OVERFLOW FLAG
08A9 07E1                 JE    FETCH
                   *********
                   * BOXEV *
                   *********
08AA 4D01          AV     TZ    13,X'01'  LINK = 1 ???
08AB 07E1                 JE    FETCH     LINK=1 • RESULT IN TRUE FORM
08AC 0D3C                 JE    E00       MUST COMPLEMENTERESULT & SIGN
```

```
                         * MULTOPLY ROUTINE     ******************************
                         * OPCODE 03
                         **********
                         * BOX£A *      SEPARATE F FIELD INTO P11 & P12
                         **********
08AD 085F                MUL    JE    L,R         SEPERATE (L/R)
                         **********
                         * BOX£B *
                         **********
08AE 4C07                       TZ    12,X'07'    TEST IF SIGN REQUIRED
08AF 14B5                       JP    ME          SIGN NOT£REQUIRED
                         **********
                         * BOX£C *
                         **********
08B0 CA03                       MOV   10,(N)
08B1 A902                       RMF   9,(M)
08B2 8C40                       INC   12
                         **********
                         * BOX£D *    COMPUTE ALGEBRAIC SIGN OF RESULT
                         **********
08B3 1000                       NOP
08B4 D120                       XOR   1,T
                         **********
                         * BOX£E *
                         **********
08B5 1100                ME     LT    X'00'       PAGE OUT£INDEX
08B6 06C8                       JE    VIA
08B7 1080                       SSF
08B8 F721                       SFR   7,(T)
08B9 1040                       SPF
08BA C120                       LOR   1,T         SMMMMXXX£- P1
                         *                        SIGN & MAP PACKED INTO P1
08BB 8B41                       INC   11,(T)
08BC 8A29                       ADD*  10,T,(T)
08BD 1080                       SSF
08BE BD20                       CPY   13,T
08BF 1040                       SPF
08C0 8981                       ADD   9,L,(T)
08C1 1080                       SSF
08C2 BC20                       CPY   12,T
08C3 1040                       SPF
08C4 9C41                       DEC   12,(T)      P12 - 1 --> T
08C5 9B21                       SBT   11,T,(T)    P11 + 1 -(P12 - 1) --> T
08C6 BC20                       CPY   12,T        SET UP MAJOR COUNTER
                         *      MOVE A REG, (P2 -P6) TO S7 - S 11
08C7 C201                       MOV   2,(T)
08C8 1080                       SSF
08C9 B720                       CPY   7,T
```

```
08CA 1040            SPF
08CB C301            MOV  3,(T)
08CC 1080            SSF
08CD B820            CPY  8,T
08CE 1040            SPF
08CF C401            MOV  4,(T)
08D0 1080            SSF
08D1 B920            CPY  9,T
08D2 1040            SPF
08D3 C501            MOV  5,(T)
08D4 1080            SSF
08D5 BA20            CPY  10,T
08D6 1040            SPF
08D7 C601            MOV  6,(T)
08D8 1080            SSF
08D9 BB20            CPY  11,T
08DA 0871            JE   Z6      ZERO FILES 6-2
08DB 1040            SPF
08DC 086C            JE   Z11     ZERO FILES 11 - 2
               *********
               * BOX£G *
               *********
08DD 9C40      MG    DEC   12
               *********
               * BOX£H *   TEST MAJOR COUNTER£= ZERO
               *********
08DE 5CFF            TN   12,X'FF'
08DF 1511            JP   MP      MAJOR COUNTER IS ZERO
               *********
               * BOX£I *    READ IN A BYTE OF MULTIPLIER
               *********
08E0 1080            SSF
08E1 9D43            DEC  13,(N)
08E2 9C82            SBT  12,L,(M)
08E3 A000            RMF  0
               *********
               * BOX£J *
               *********
08E4 1040            SPF
08E5 2D08            LF   13,X'08'
08E6 1080            SSF
08E7 B120            CPY  1,T
               *********
               * BOX£K * TEST LOW ORDER BIT£OF S1
               *********
08E8 5101      MK    TN   1,X'01'
08E9 1500            JP   MM      ZERO BIT; SHIFT
               *********
```

```
                       * BOX£L *      ONE BIT; ADD & SHIFT
                       *********
08EA CB01                    MOV    11,(T)
08EB 8630                    ADD    6,T,C
08EC CA01                    MOV    10,(T)
08ED 85B0                    ADD    5,T,C,L
08EE C901                    MOV    9;(T)
08EF 84B0                    ADD    4,T,C,L
08F0 C801                    MOV    8,(T)
08F1 83B0                    ADD    3,T,C,L
08F2 C701                    MOV    7,(T)
08F3 82B0                    ADD    2,T,C,L
08F4 1040                    SPF
08F5 CB01                    MOV    11,(T)
08F6 86B0                    ADD    6,T,C,L
08F7 CA01                    MOV    10,(T)
08F8 85B0                    ADD    5,T,C,L
08F9 C901                    MOV    9,(T)
08FA 84B0                    ADD    4,T,C,L
08FB C801                    MOV    8,(T)
08FC 83B0                    ADD    3,T,C,L
08FD C701                    MOV    7,(T)
08FE 82B0                    ADD    2,T,C,L
08FF 1080                    SSF
                       *********
                       * BOX£M *
                       *********
0900 FB00              MM    SFL    11
0901 FA80                    SFL    10,L
0902 F980                    SFL    9,L
0903 F880                    SFL    8,L
0904 F780                    SFL    7,L
0905 1040                    SPF
0906 FB80                    SFL    11,L
0907 FA80                    SFL    10,L
0908 F980                    SFL    9,L
0909 F880                    SFL    8,L
090A F780                    SFL    7,L
090B 9D40                    DEC    13
                       *********
                       * BOX£N *
                       *********
090C 5D0F                    TN     13,X'0F'
090D 14DD                    JP     MG
                       *********
                       * BOX£O *
                       *********
090E 1080                    SSF
```

```
090F F120                   SFR    1
0910 14E8                   JP     MK
                  *********
                  * BOX£P *
                  *********
0911 C101         MP        MOV    1,(T)
0912 1080                   SSF
0913 B720                   CPY    7,T
0914 F700                   SFL    7              RESTORE MAP
0915 2180                   LF     1,X'80'
0916 E120                   AND    1,T            RESTORE SIGN OF X
0917 1040                   SPF
0918 2180                   LF     1,X'80'
0919 E120                   AND    1,T            RESTORE SIGN OF A
                  *********
                  * BOX£S *
                  *********
091A 1600                   LU     X'00'
091B 07E1                   JE     FETCH
```

```
                        * DIVIDE ROUTINE ************************************
                        * OPCODE 04
                        *********
                        * BOX£A *
                        *********
091C 085F               DIV    JE    L,R        SEPERATE (L;R)
091D C101                      MOV   1,(T)
091E 1080                      SSF
091F B120                      CPY   1,T
0920 1040                      SPF
                        *********
                        * BOX£B *    TEST IF SIGN REQUIRED
                        *********
0921 4C0F                      TZ    12,X'0F'
0922 1529                      JP    DE
                        *********
                        * BOX£C *    SIGN IS REQUIRED
                        *********
0923 CA03                      MOV   10,(N)
0924 A902                      RMF   9,(M)
                        *********
                        * BOX£D *    COMPUTE ALGEBRAIC SIGN OF RESULT
                        *********
0925 8C40                      INC   12
0926 D120                      XOR   1,T
0927 5B07                      TN    11,X'07'   TEST FOR£(0,0) CASE
0928 156F                      JP    DI         DIVIDE BY ZERO ATTEMPTD
                        *********
                        * BOX£E *
                        *********
0929 1100               DE     LT    X'00'
092A 06C8                      JE    VIA        PAGE OUT£INDEX
092B 8B49                      INC*  11,(T)
092C 8A29                      ADD*  10,T,(T)
092D 1080                      SSF
092E BD20                      CPY   13,T
092F 1040                      SPF
0930 8981                      ADD   9,L,(T)
0931 1080                      SSF
0932 BC20                      CPY   12,T
0933 F721                      SFR   7,(T)
0934 C120                      LOR   1,T
0935 1040                      SPF
0936 CC01                      MOV   12,(T)
0937 9B21                      SBT   11,T,(T)
0938 BC60                      CPY   12,I,T
0939 2D0C                      LF    13,X'0C'
093A C201                      MOV   2,(T)
```

```
 PAGE 225

093B B720                 CPY   7,T
093C C301                 MOV   3,(T)
093D B820                 CPY   8,T
093E C401                 MOV   4,(T)
093F B920                 CPY   9,T
0940 C501                 MOV   5,(T)
0941 BA20                 CPY   10,T
0942 C601                 MOV   6,(T)
0943 BB20                 CPY   11,T
0944 1080                 SSF
0945 C201                 MOV   2,(T)
0946 1040                 SPF
0947 B220                 CPY   2,T
0948 1080                 SSF
0949 C301                 MOV   3,(T)
094A 1040                 SPF
094B B320                 CPY   3,T
094C 1080                 SSF
094D C401                 MOV   4,(T)
094E 1040                 SPF
094F B420                 CPY   4,T
0950 1080                 SSF
0951 C501                 MOV   5,(T)
0952 1040                 SPF
0953 B520                 CPY   5,T
0954 1080                 SSF
0955 C601                 MOV   6,(T)
0956 1040                 SPF
0957 B620                 CPY   6,T
0958 1080                 SSF
0959 086C                 JE    Z11        ZERO FILES 11 - 2
095A 1040                 SPF
                   *********
                   * BOX£F *
                   *********
095B 5C0F          DF     TN    12,X'0F'
095C 1567                 JP    DH
                   *********
                   * BOX£G *
                   *********
095D 1080                 SSF
095E 9D43                 DEC   13,(N)
095F 9C82                 SBT   12,L,(M)
0960 1040                 SPF
0961 AD46                 RMF   13,D,(U)
0962 9C40                 DEC   12
0963 1080                 SSF
0964 B027                 CPY   0,T,(S)
```

```
0965 1040                 SPF
0966 155B                 JP     DF
                   *********
                   * BOXEH *     TEST IF DIVISOR IS ZERO
                   *********
0967 1080          DH     SSF
0968 CB11                 MOV    11,C,(T)
0969 CA91                 MOV    10,C,L,(T)
096A C991                 MOV    9,C,L,(T)
096B C891                 MOV    8,C,L,(T)
096C C791                 MOV    7,C,L,(T)
096D 5004                 TN     0,X'04'
096E 1576                 JP     DJ1
                   *********
                   * BOXEI *        DIVIDE BY ZERO ATTEMPTED
                   *********
096F C101          DI     MOV    1,(T)
0970 B720                 CPY    7,T
0971 F700                 SFL    7
0972 1180                 LT     X'80'
0973 E120                 AND    1,T         RESTORE SIGN
0974 0D53                 JE     OVERFL      GO SET OVERFLOW FLAG
0975 1600                 LU     X'00'
                   **********
                   * BOXEDJ1*    TEST IF A > M OR A = M
                   **********
0976 CB01          DJ1    MOV    11,(T)
0977 1040                 SPF
0978 9B38                 SBT*   11,C,T
0979 1080                 SSF
097A CA01                 MOV    10,(T)
097B 1040                 SPF
097C 9AB8                 SBT*   10,C,L,T
097D 1080                 SSF
097E C901                 MOV    9,(T)
097F 1040                 SPF
0980 99B8                 SBT*   9,C,L,T
0981 1080                 SSF
0982 C801                 MOV    8,(T)
0983 1040                 SPF
0984 98B8                 SBT*   8,L,C,T
0985 1080                 SSF
0986 C701                 MOV    7,(T)
0987 1040                 SPF
0988 97B8                 SBT*   7,L,C,T
0989 1080                 SSF
098A 4004                 TZE    0,X'04'     A = M ???
098B 156F                 JP     DI          OVERFLOW WILL OCCUR
```

```
098C FC00                SFL   12         GET LINK BIT
098D 4C01                TZE  12,X'01'    LINK = 1 ==> POSITIVE RESULT
098E 156F                JP    DI         POSITIVE RESULT ==> A > M
                 **********
                 * BOXEJ *
                 **********
098F 1040                SPF
0990 2D51                LF    13,X'51'
                 **********
                 * BOXEN *
                 **********
0991 1690                LU   X'90'
0992 F600                SFL   6
0993 15A8                JP    DS
                 **********
                 * BOXEO *
                 **********
0994 1080        DO      SSF
0995 C801                MOV   11,(T)
0996 8627                ADD   6,T,(S)
0997 CA01                MOV   10,(T)
0998 85A7                ADD   5,T,L,(S)
0999 C901                MOV   9,(T)
099A 84A7                ADD   4,T,L,(S)
099B C801                MOV   8,(T)
099C 83A7                ADD   3,T,L,(S)
099D C701                MOV   7,(T)
099E 82B7                ADD   2,T,L,C,(S)
                 **********
                 * BOXEP *
                 **********
099F FC80                SFL   12,L
09A0 CC01                MOV   12,(T)
09A1 BD20                CPY   13,T       S13 <== LINK BIT
09A2 1600                LU    X'00'      ASSUME NEGATIVE RESULT
09A3 4C01                TZE   12,X'01'   LINK = 1 ==> POSITIVE RESULT
09A4 1690                LU    X'90'      POSITIVE RESULT
09A5 FC20                SFR   12         PUT VALUE BACK IN LINK REG.
09A6 1040                SPF
                 **********
                 * BOXES *
                 **********
09A7 F680                SFL   6,L        LINK BITECONTAINS RIGHT VALUE
09A8 F580        DS      SFL   5,L
09A9 F480                SFL   4,L
09AA F380                SFL   3,L
09AB F280                SFL   2,L
09AC FB80                SFL   11,L
```

```
09AD FA80                  SFL   10,L
09AE F980                  SFL   9,L
09AF F880                  SFL   8,L
09B0 F780                  SFL   7,L
                    *********
                    * BOX£T *
                    *********
09B1 3DFF                  AF    13,X'FF'
09B2 5DFF                  TN    13,X'FF'
09B3 15BB                  JP    DU
09B4 1080                  SSF
09B5 F680                  SFL   6,L
09B6 F580                  SFL   5,L
09B7 F480                  SFL   4,L
09B8 F380                  SFL   3,L
09B9 F280                  SFL   2,L
09BA 1594                  JP    DO
                    *********
                    * BOX£U *
                    *********
09BB 1080           DU     SSF
09BC 4D01           BU     TZ£   13,X'01'  LINK = 1 ==> POSITIVE RESULT
09BD 15CA                  JP    DW        POSITIVE RESULT
                    *********
                    * BOX£V *    NEGATIVE RESULT
                    *********
09BE CB01                  MOV   11,(T)
09BF 8620                  ADD   6,T
09C0 CA01                  MOV   10,(T)
09C1 85A0                  ADD   5,L,T
09C2 C901                  MOV   9,(T)
09C3 84A0                  ADD   4,L,T
09C4 C801                  MOV   8,(T)
09C5 83A0                  ADD   3,L,T
09C6 C701                  MOV   7,(T)
09C7 82B0                  ADD   2,L,C,T
09C8 FD80                  SFL   13,L      S13 <== LINK BIT
09C9 15BC                  JP    BU
                    *********
                    * BOX£W *    POSITIVE RESULT
                    *********
09CA C101           DW     MOV   1,(T)
09CB 8720                  CPY   7,T
09CC F700                  SFL   7
09CD 1180                  LT    X'80'
09CE E120                  AND   1,T
09CF 1600                  LU    X'00'
09D0 07E1                  JE    FETCH
```

```
09D0 07E1           * NUM ROUTINE ****************************************
                    * THIS ROUTINE TAKES 10 CHARACTERS IN THE A & X£REG,
                    * AND CONVERTS THEM TO THEIR BINARY EQUIVALENT
                    *********
                    * BOX£A *
                    *********
09D1 BD00           NUM     ZOF    13
09D2 BC00                   ZOF    12
09D3 BB00                   ZOF    11
09D4 27E1                   LF     7,X'E1'
                    *********
                    * BOX£B *
                    *********
09D5 8746           NB      INC    7,(U)
                    *********
                    * BOX£C *
                    *********
09D6 6719                   CP     7,X'19'
09D7 1C13                   JP     ND
                    *********
                    * BOX£J *
                    *********
09D8 27E1                   LF     7,X'E1'
09D9 B900                   ZOF     9
09DA BA00                   ZOF     10
                    *********
                    * BOX£K *
                    *********
09DB 8746           NK      INC    7,(U)
                    *********
                    * BOX£M *
                    *********
09DC 110F                   LT     X'0F'
09DD 1080                   SSF
09DE 0029                   EOT*   0,T,(T)
09DF 1040                   SPF
                    *********
                    * BOX£N *
                    *********
09E0 8D20                   ADD    13,T
09E1 8C80                   ADD    12,L
09E2 8B80                   ADD    11,L
09E3 8A80                   ADD    10,L
09E4 8981                   ADD    9,L,(T)
                    *********
                    * BOX£L *
                    *********
09E5 671A                   CP     7,X'1A'
```

```
09E6 15F2                JP     NO
                   *********
                   * BOX£S *
                   *********
09E7 B220                CPY    2,T
09E8 CA01                MOV    10,(T)
09E9 B320                CPY    3,T
09EA CB01                MOV    11,(T)
09EB B420                CPY    4,T
09EC CC01                MOV    12,(T)
09ED B520                CPY    5,T
09EE CD01                MOV    13,(T)
09EF B620                CPY    6,T
                   *********
                   * BOX£U *
                   *********
09F0 1600                LU     X'00'
09F1 07E1                JE     FETCH
                   *********
                   * BOX£O *
                   *********
09F2 B220          NO    CPY    2,T
09F3 CA01                MOV    10,(T)
09F4 B320                CPY    3,T
09F5 CB01                MOV    11,(T)
09F6 B420                CPY    4,T
09F7 CC01                MOV    12,(T)
09F8 B520                CPY    5,T
09F9 CD01                MOV    13,(T)
                   *********
                   * BOX£P *
                   *********
09FA FD00                SFL    13
09FB FC80                SFL    12,L
09FC FB80                SFL    11,L
09FD FA80                SFL    10,L
09FE F980                SFL    9,L
09FF FD00                SFL    13
0A00 FC80                SFL    12,L
0A01 FB80                SFL    11,L
0A02 FA80                SFL    10,L
0A03 F980                SFL    9,L
                   *********
                   * BOX£Q *
                   *********
0A04 8D20                ADD    13,T
0A05 C501                MOV    5,(T)
0A06 8CA0                ADD    12,T,L
```

```
0A07 C401                   MOV    4,(T)
0A08 8BA0                   ADD    11,T,L
0A09 C301                   MOV    3,(T)
0A0A 8AA0                   ADD    10,T,L
0A0B C201                   MOV    2,(T)
0A0C 89A0                   ADD    9,T,L
                     *********
                     * BOX£R *
                     *********
0A0D FD00                   SFL    13
0A0E FC80                   SFL    12,L
0A0F FB80                   SFL    11,L
0A10 FA80                   SFL    10,L
0A11 F981                   SFL    9,L,(T)
0A12 15DB                   JP     NK
                     *********
                     * BOX£D *
                     *********
0A13 110F            ND     LT     X'0F'
0A14 0029                   EOT*   0,T,(T)
                     *********
                     * BOX£E *
                     *********
0A15 8D20                   ADD    13,T
0A16 8C80                   ADD    12,L
0A17 8B81                   ADD    11,L,(T)
                     *********
                     * BOX£F *
                     *********
0A18 B820                   CPY    8,T
0A19 CC01                   MOV    12,(T)
0A1A B920                   CPY    9,T
0A1B CD01                   MOV    13,(T)
                     *********
                     * BOX£G *
                     *********
0A1C FD00                   SFL    13
0A1D FC80                   SFL    12,L
0A1E FB80                   SFL    11,L
0A1F FD00                   SFL    13
0A20 FC80                   SFL    12,L
0A21 FB80                   SFL    11,L
                     *********
                     * BOX£H *
                     *********
0A22 8D20                   ADD    13,T
0A23 C901                   MOV    9,(T)
0A24 8CA0                   ADD    12,L,T
```

```
0A25 C801                    MOV    8,(T)
0A26 8BA0                    ADD    11,T,L
                        *********
                        * BOXEI *
                        *********
0A27 FD00                    SFL    13
0A28 FC80                    SFL    12,L
0A29 FB80                    SFL    11,L
0A2A 15D5                    JP     NB
```

```
                         * CHAR ROUTINE ***************************************
                         * THIS ROUTINE TAKES A 40 BIT BINARY NUMBER IN THE
                         * A REG, AND CONVERTS IT INTO 10 ASCII CHARACTERS
                         *  OPCODE 05
                         *********
                         * BOX£A *    SUBTRACT 254,0BE,3FF FROM A REG,
                         *-NOTE THIS IS-THE HEX£NUMBER = 9,999,999,999 DEC,
                         *********
0A2B 11FF                CHAR   LT    X'FF'
0A2C 9638                       SBT*  6,T,C
0A2D 11E3                       LT    X'E3'
0A2E 95B8                       SBT*  5,T,L,C
0A2F 110B                       LT    X'0B'
0A30 94B8                       SBT*  4,L,T,C
0A31 1154                       LT    X'54'
0A32 93B8                       SBT*  3,L,T,C
0A33 1102                       LT    X'02'
0A34 92B8                       SBT*  2,T,L,C
                         *********
                         * BOX£B *
                         *********
0A35 F780                       SFL   7,L
0A36 5701                       TN    7,X'01'   TEST LINK BIT
0A37 1C3A                       JP    HD
0A38 5004                       TN    0,X'04'   TEST ZERO
0A39 0D53                       JE    OVERFL    GO SET OVERFLOW FLAG
                         *********
                         * BOX£D *
                         *********
0A3A C201                HD     MOV   2,(T)
0A3B B720                       CPY   7,T
0A3C C301                       MOV   3,(T)
0A3D B820                       CPY   8,T
0A3E C401                       MOV   4,(T)
0A3F B920                       CPY   9,T
0A40 C501                       MOV   5,(T)
0A41 BA20                       CPY   10,T
0A42 C601                       MOV   6,(T)
0A43 BB20                       CPY   11,T
                         *********
                         * BOX£E *
                         *********
0A44 1100                       LT    X'00'
0A45 06C8                       JE    VIA
0A46 1080                       SSF
0A47 F701                       SFL   7,(T)
0A48 C120                       LOR   1,T
0A49 2C07                       LF    12,X'07'  LOAD CHARACTER£COUNTER
```

```
0A49 2C07          *********
                   * BOX£F *
                   *********
0A4A 1600          HF     LU    X'00'
0A4B 0A66                 JE    ,5BY10
0A4C 9C46                 DEC   12,(U)    LOAD MASK, DECREMENT£COUNTER
0A4D 1180                 LT    X'B0'
0A4E CB21                 LOR   11,T,(T)  PICK UP NEXT ASCII CHARACTER
0A4F B027                 CPY   0,T,(S)
0A50 6CFD                 CP    12,X'FD'
0A51 1C53                 JP    HH
0A52 1C4A                 JP    HF
                   *********
                   * BOX£H *
                   *********
0A53 2C07          HH     LF    12,X'07'
0A54 1600                 LU    X'00'
0A55 0A66                 JE    ,5BY10
0A56 9C46                 DEC   12,(U)
0A57 1180                 LT    X'B0'
0A58 CB21                 LOR   11,T,(T)
0A59 1040                 SPF
0A5A B027                 CPY   0,T,(S)
0A5B 1080                 SSF
0A5C 6CFD                 CP    12,X'FD'
0A5D 1C5F                 JP    HI
0A5E 1C54                 JP    HH+1
                   *********
                   * BOX£I *
                   *********
0A5F 1080          HI     SSF
0A60 F129                 SFR*  1,(T)
0A61 8720                 CPY   7,T
0A62 1180                 LT    X'80'
0A63 E120                 AND   1,T
                   *********
                   * BOX£J *
                   *********
0A64 1600                 LU    X'00'
0A65 07E1                 JE    FETCH
```

```
                        *     THIS IS A 5 BYTE DIVIDE ROUTINE
                        *********
                        * BOX£A *
                        *********
0A66 1080               ,5BY10 SSF
0A67 B700                     ZOF   7
0A68 B800                     ZOF   8
0A69 B900                     ZOF   9
0A6A BA00                     ZOF   10
0A6B BB00                     ZOF   11
0A6C 1040                     SPF
0A6D 2C23                     LF    12,X'23'  # OF TIMES THRU ROUTINE
                        *********
                        * BOX£AA*
                        *********
0A6E 2D06                     LF    13,X'06'
0A6F FB00               TAA   SFL   11
0A70 FA80                     SFL   10,L
0A71 F980                     SFL   9,L
0A72 F880                     SFL   8,L
0A73 F780                     SFL   7,L
0A74 9D40                     DEC   13
0A75 4DFF                     TZ   13,X'FF'
0A76 1C6F                     JP    TAA
                        *********
                        * BOX£B *     INITIALIZE FIRST PASS
                        *********
0A77 1690                     LU    X'90'     SET UP SUBTRACT
0A78 FB00                     SFL   11
0A79 110A                     LT    X'0A'     LOAD T WITH SUBTREND
0A7A 1C87                     JP    TE        JUMP TO SHIFT£ROUTINE
                        *********
                        * BOX£C *
                        *********
0A7B 8B27               TC    ADD   11,T,(S)
0A7C 8A87                     ADD   10,L,(S)
0A7D 8987                     ADD   9,L,(S)
0A7E 8887                     ADD   8,L,(S)
0A7F 8787                     ADD   7,L,(S)
                        *********
                        * BOX£D *     TEST RESULTS
                        *********
                        *             LINK = 0 ==> NEGATIVE RESULT
                        *             LINK = 1 ==> POSITIVE RESULT
0A80 FD80                     SFL   13,L
0A81 FD28                     SFR*  13
0A82 1600                     LU    X'00'     ASSUME NEGATIVE RESULT
0A83 4D01                     TZ    13,X'01'  LINK = 1 ==> POSITIVE
```

```
0A84 1690                  LU    X'90'     POSITIVE RESULT
0A85 1040                  SPF
                    *********
                    * BOX£E *
                    *********
0A86 FB80                  SFL   11,L      LINK BIT£CONTAINS RIGHT£VALUE
0A87 FA80           TE     SFL   10,L
0A88 F980                  SFL   9,L
0A89 F880                  SFL   8,L
0A8A F780                  SFL   7,L
                    *********
                    * BOX£F *
                    *********
0A8B 3CFF                  AP   12,X'FF'   DECREMENT£COUNTER
0A8C 5CFF                  TN   12,X'FF'   FINISHED???
0A8D 1C95                  JP    TG        RETURN
0A8E 1080                  SSF
0A8F FB80                  SFL   11,L
0A90 FA80                  SFL   10,L
0A91 F980                  SFL   9,L
0A92 F880                  SFL   8,L
0A93 F780                  SFL   7,L
0A94 1C78                  JP    TC
                    *********
                    * BOX£G *    FINISHED DIVIDING, GET REMAINDER +VE
                    *********
0A95 1080           TG     SSF
0A96 4D01                  TZ    13,X'01'  LINK = 1 ==> POSITIVE RESULT
0A97 1020                  RTN
                    *********
                    * BOX£H *    REMAINDER IS NEGATIVE
                    *********
0A98 110A                  LT    X'0A'
0A99 8B20                  ADD   11,T
0A9A 8A80                  ADD   10,L
0A9B 8980                  ADD   9,L
0A9C 8880                  ADD   8,L
0A9D 8780                  ADD   7,L
0A9E 1600                  LU    X'00'
0A9F 1C97                  JP    TG+2
```

```
                       * SHIFT£ROUTINE ************************************
                       * THIS ROUTINE HANDLES THE SHIFT£INSTRUCTIONS
                       * OPCODE 06
                       *********
                       * BOX£A1*
                       *********
~~0AA0 C911             SHIFT  MOV   9,C,(T)~~
0AA1 CA91                     MOV   10,C,L,(T)
0AA2 4004                     TZ    0,X'04'   TEST IF 0 BYTES TO SHIFT
0AA3 07E1                     JE    FETCH     NO OPERATION, RETURN
                       *********
                       * BOX£A *
                       *********
0AA4 4C01                     TZ£   12,X'01'  TEST FOR£SHIFT£RIGHT£COMMAND
0AA5 1CCF                     JP    SL,       SHIFT RIGHT
                       *********
                       * BOX£B *    SHIFT LEFT ROUTINE
                       *********
0AA6 5C04              SB     TN    12,X'04'  TEST FOR CIRCULAR SHIFT
0AA7 1CAC                     JP    SD        NON=CIRCULAR SHIFT
                       *********
                       * BOX£C *    CIRCULAR SHIFT
                       *********
0AA8 C201                     MOV   2,(T)
0AA9 1080                     SSF
0AAA BB20                     CPY   11,T
0AAB 1CAE                     JP    SE
                       *********
                       * BOX£D *    NON=CIRCULAR SHIFT
                       *********
0AAC 1080              SD     SSF
0AAD BB00                     ZOF   11
                       *********
                       * BOX£E *
                       *********
0AAE 1040              SE     SPF
0AAF C301                     MOV   3,(T)
0AB0 B220                     CPY   2,T
0AB1 C401                     MOV   4,(T)
0AB2 B320                     CPY   3,T
0AB3 C501                     MOV   5,(T)
0AB4 B420                     CPY   4,T
0AB5 C601                     MOV   6,(T)
0AB6 B520                     CPY   5,T
                       *********
                       * BOX£H *    TEST FOR SLA COMMAND
                       *********
0AB7 B600                     ZOF   6         ASSUME SLA COMMAND
```

```
0AB7 B600           **********
                    * BOXEF *
                    **********
0AB8 5C06                   TN     12,X'06'  TEST IF SLA COMMAND
0AB9 1CCA                   JP     SI,       SLA COMMAND
                    **********
                    * BOXEG *      SLAXEOR SLC COMMAND
                    **********
0ABA 1080                   SSF
0ABB C201                   MOV    2,(T)
0ABC 1040                   SPF
0ABD B620                   CPY    6,T
0ABE 1080                   SSF
0ABF C301                   MOV    3,(T)
0AC0 B220                   CPY    2,T
0AC1 C401                   MOV    4,(T)
0AC2 B320                   CPY    3,T
0AC3 C501                   MOV    5,(T)
0AC4 B420                   CPY    4,T
0AC5 C601                   MOV    6,(T)
0AC6 B520                   CPY    5,T
0AC7 CB01                   MOV    11,(T)
0AC8 B620                   CPY    6,T
0AC9 1040                   SPF
                    **********
                    * BOXEI *      DECREMENT COUNTER
                    **********
0ACA 9A50           SI,     DEC    10,C
0ACB 9990                   SBT    9,L,C
                    **********
                    * BOXEJ *
                    **********
0ACC 5004                   TN     0,X'04'   TEST RESULT
0ACD 1CA6                   JP     SB        NON-ZERO RESULT
                    **********
                    * BOXEK *      ZERO RESULT
                    **********
0ACE 07E1                   JE     FETCH
                    **********
                    * BOXEL *      RIGHT SHIFT ROUTINE
                    **********
0ACF 5C04           SL,     TN     12,X'04'  TEST IF CIRCULAR COMMAND
0AD0 1CF2                   JP     SN,       NON-CIRCULAR COMMAND
                    **********
                    * BOXEM *      CIRCULAR COMMAND
                    **********
0AD1 1080                   SSF
0AD2 C601                   MOV    6,(T)
```

```
0AD3 1040                 SPF
0AD4 B720                 CPY    7,T
                    *********
                    * BOX£P *    SHIFT SECONDARY FILES (X£REG.)
                    *********
0AD5 1080           SP    SSF
0AD6 C501                 MOV    5,(T)
0AD7 B620                 CPY    6,T
0AD8 C401                 MOV    4,(T)
0AD9 B520                 CPY    5,T
0ADA C301                 MOV    3,(T)
0ADB B420                 CPY    4,T
0ADC C201                 MOV    2,(T)
0ADD B320                 CPY    3,T
0ADE 1040                 SPF
0ADF C601                 MOV    6,(T)
0AE0 1080                 SSF
0AE1 B220                 CPY    2,T
0AE2 1040                 SPF
                    *********
                    * BOX£Q *    SHIFT PRIMARY FILES ( A REG.)
                    *********
0AE3 C501           SQ.   MOV    5,(T)
0AE4 B620                 CPY    6,T
0AE5 C401                 MOV    4,(T)
0AE6 B520                 CPY    5,T
0AE7 C301                 MOV    3,(T)
0AE8 B420                 CPY    4,T
0AE9 C201                 MOV    2,(T)
0AEA B320                 CPY    3,T
0AEB C701                 MOV    7,(T)
0AEC B220                 CPY    2,T
                    *********
                    * BOX£R *    DECREMENT£COUNTER
                    *********
0AED 9A50                 DEC    10,C
0AEE 9990                 SBT    9,L,C
                    *********
                    * BOX£R *    TEST IF RESULT ZERO
                    *********
0AEF 4004                 TZ     0,X'04'
0AF0 07E1                 JE     FETCH      ZERO RESULT
0AF1 1CCF                 JP     SL.        NON-ZERO RESULT
                    *********
                    * BOX£N *    NON-CIRCULAR SHIFT
                    *********
0AF2 B700           SN.   ZOF    7
                    *********
```

```
                         * BOX£O *    TEST IF SRA OR SRAX£COMMAND
                         **********
0AF3 5C02                       TN    12,X'02'
0AF4 1CE3                       JP    SQ,          SRA COMMAND
0AF5 1CD5                       JP    SP           SRAX£COMMAND
```

```
                   * MOV ROUTINE****************************************
                   * OPCODE 07
                   *********
                   * BOXEA *
                   *********
0AF6 1101          MOV    LT     X'01'
0AF7 06C0                 JE     VIA      PAGE IN INDEXE#1
                   *             MOVE INDEXE1 (S9,S10) TO P7, P8
0AF8 1080                 SSF
0AF9 C901                 MOV    9,(T)
0AFA 1040                 SPF
0AFB B720                 CPY    7,T
0AFC 1080                 SSF
0AFD CA01                 MOV    10,(T)
0AFE 1040                 SPF
0AFF B820                 CPY    8,T
                   *             ADJUST CONTENTS OF INDEXE1
                   *             INDEX1<==INDEX1 + F (# OF WORDS MOVED)
0B00 CC01                 MOV    12,(T)
0B01 1080                 SSF
0B02 8A20                 ADD    10,T
0B03 8980                 ADD    9,L
0B04 1040                 SPF
                   *             CONVERT # OF WORDS (MIXAL) TO TRANSFER
                   *             TO # OF BYTES (MICRODATA) TO TRANSFER
                   *             F <== F * 8
0B05 FC00                 SFL    12
0B06 FC00                 SFL    12
0B07 FC00                 SFL    12
                   *             CONVERTEMIXAL ADDRESS FROM INDEXE1
                   *             TO MICRODATA ADDRESS
                   *             MOCRODATA ADDRESS = MIXAL ADDRESS * 8
0B08 F800                 SFL    8
0B09 F780                 SFL    7,L
0B0A F800                 SFL    8
0B0B F780                 SFL    7,L
0B0C F800                 SFL    8
0B0D F780                 SFL    7,L
                   *             SET UP ADDRESSES FOR READ WRITE LOOP
0B0E 9840                 DEC    8
0B0F 9780                 SBT    7,L
0B10 9A40                 DEC    10
0B11 9980                 SBT    9,L
0B12 2B05                 LF     11,X'05'  SKIP MASK
                   *             SENDING ADDRESS IN P9, P10
                   *             RECIEVING ADDRESS IN P7, P8
                   *             BYTE COUNTER IN P12
                   *********
                   * BOX£C *    READ WRITE LOOP
                   *********
0B13 8A43          R,W    INC   10,(N)
0B14 A982                 RMF   9,L,(M)   READ A BYTE
                   *********
                   * BOX£E *
                   *********
0B15 8843                 INC   8,(N)
0B16 A792                 WMF   7,L,(M)   WRITE A BYTE
                   *********
                   * BOX£D *
                   *********
0B17 9C40                 DEC   12        ADJUST BYTE COUNTER
                   *********
                   * BOX£B1*
                   *********
0B18 1105                 LT    X'05'     LOAD T WITH SKIP MASK
0B19 EA29                 AND*  10,T,(T)
0B1A DB38                 XOR*  11,T,C    TEST IF TIME TO SKIP 2 BYTES
0B1B 4004                 TZ    0,X'04'   TIME TO SKIP TWO BYTES??
0B1C 1D20                 JP    ,SKIP     YES
                   *********
                   * BOX£B *
                   *********
0B1D 4CFF          MB     TZ    12,X'FF'    TRANSFER£DONE??
0B1E 1D13                 JP    R,W
                   *********
                   * BOX£F *
                   *********
0B1F 07E1                 JE    FETCH
                   *********
                   * BOX£G *    SKIP TWO BYTES
                   *********
0B20 1102          ,SKIP  LT    X'02'
0B21 8A20                 ADD   10,T      INCREMENT£SOURCE ADDRESS
0B22 8980                 ADD   9,L
0B23 8820                 ADD   8,T       INCREMENT£TARGET ADDRESS
0B24 8780                 ADD   7,L
0B25 9C20                 SBT   12,T      INCREMENT£COUNTER
0B26 1D1D                 JP    MB
```

```
                    *    LOAD ROUTINE
                    *    THIS ROUTINE HANDLES ALL 16 LOAD COMMANDS
                    *
                    *******
                    *BOX£A*     PUT RIGHT FIELD SPEC IN P11, LEFT FIELD
                    *******
0B27 085F           LOAD   JE    L,R        SEPERATE (L;R)
                    ********
                    *BOX£B *      TEST=IS SIGN BYTE REQUIRED?
                    ********
0B28 2780                  LF    7,X'80'
0B29 4C07                  TZ    12,X'07'
0B2A 1D6C                  JP    LD         JUMP TO BOX£D
                    ********
                    *BOX£C *  READ IN SIGN BYTE
                    ********
0B2B CA03                  MOV   10,(N)
0B2C C902                  MOV   9,(M)
0B2D ACC0                  RMF   12,I
                    ********
                    *BOX£E *     TEST=LAOD NEGATIVE COMMAND?
                    ********
0B2E 5D10           LE     TN    13,X'10'
0B2F 1D31                  JP    LG         JUMP TO BOX£G
                    ********
                    *BOX£F *     FLIP SIGN BIT IN T
                    ********
0B30 D729                  XOR*  7,T,(T)
                    ********
                    *BOX£G *
                    ********
0B31 E729           LG     AND*  7,T,(T)   AND OFF SIGN
0B32 CD20                  LOR   13,T
0B33 1601                  LU    X'01'
0B34 2801                  LF    8,X'01'
0B35 2705                  LF    7,X'05'
0B36 CC01                  MOV   12,(T)
0B37 9B29                  SBT*  11,T,(T)
0B38 9720                  SBT   7,T
                    ********
                    *BOX£H * TEST=LOAD A COMMAND?
                    ********
0B39 5D07                  TN    13,X'07'
0B3A 1D59                  JP    LS         JUMP TO BOX£S
                    ********
                    *BOX£I *     TEST= LOAD INDEX£COMMAND?
                    ********
0B3B 1107                  LT    X'07'
```

```
0B3C ED29              AND*  13,T,(T)
0B3D 1080              SSF
0B3E 2D07              LF    13,X'07'
0B3F DD38              XOR*  13,T,C
0B40 1040              SPF
0B41 4004              TZ    0,X'04'
0B42 1D56              JP    LR          JUMP TO BOXER= LOAD X£INSTRUC
                 ********
                 *BOX£J**
                 ********
0B43 5807              TN    11,X'07'
0B44 1D6E              JP    LL          JUMP TO BOX£L
                 ********
                 *BOX£K *
                 ********
0B45 CC01              MOV   12,(T)
0B46 9829        LK    SBT*  11,T,(T)
0B47 8720              CPY   7,T
0B48 67FE              CP    7,X'FE'
0B49 1D4C              JP    LN
                 ********
                 *BOX£M *
                 ********
0B4A 8C41              INC   12,(T)
0B4B 1D46              JP    LK
                 ********
                 *BOX£N *
                 ********
0B4C CB01        LN    MOV   11,(T)
0B4D DC38              XOR*  12,T,C
0B4E 2701              LF    7,X'01'
                 ********
                 *BOX£O *
                 ********
0B4F 4004              TZ    0,X'04'
                 ********
                 *BOX£P *
                 ********
0B50 8740              INC   7
                 ********
                 *BOX£Q *
                 ********
0B51 1600        LQ    LU    X'00'
0B52 06C4              JE    PAGE        GO PAGE IN INDEX£REGISTER
0B53 1040              SPF
0B54 2808              LF    8,X'08'
0B55 1608              LU    X'08'
                 ********
```

```
                         *BOX£R *
                         ********
0B56 CD01                LR      MOV    13,(T)
0B57 1080                        SSF
0B58 8D20                        CPY    13,T
                         ********
                         *BOX£S *
                         ********
0B59 1180                LS      LT     X'80'
0B5A ED29                        AND*   13,T,(T)
0B5B B027                        CPY    0,T,(S)
0B5C 1040                        SPF
0B5D CC01                        MOV    12,(T)
0B5E 8A23                        ADD    10,T,(N)
0B5F 8982                        ADD    9,L,(M)
0B60 9B29                        SBT*   11,T,(T)
0B61 BC60                        CPY    12,I,T
0B62 1100                        LT     X'00'    ZERO T
                         ********
                         * BOX£T *
                         ********
0B63 8846                LT      INC    8,(U)
0B64 9750                        DEC    7,C
                         ********
                         *BOX£U*    TEST LOOP VARIBLE
                         ********
0B65 4006                        TZ     0,X'06'
0B66 1D70                        JP     LW          JUMP TO BOX£W
                         ********
                         *BOX£V *   TEST= LOAD A COMMAND?
                         ********
0B67 4D07                        TZ£    13,X'07'
                         ********
                         *BOX£X *
                         ********
0B68 1080                        SSF
                         ********
                         *BOX£AA*
                         ********
0B69 B027                        CPY    0,T,(S)
0B6A 1040                        SPF
0B6B 1D63                        JP     LT
                         ********
                         *BOX£D *
                         ********
0B6C 1100                LD      LT     X'00'
0B6D 1D2E                        JP     LE
                         ********
```

```
                    *BOX£L *
                    ********
0B6E 2703           LL      LP     7,X'03'
0B6F 1D51                   JP     LQ
                    ********
                    *BOX£W *
                    ********
0B70 5B07           LW      TN     11,X'07'
                    ********
                    *BOX£Y *
                    ********
0B71 1D7E                   JP     LFF
                    ********
                    *BOX£Z *
                    ********
0B72 A001           LZ      RMF    0,(T)
                    ********
                    *BOX£BB*      TEST FOR LOAD A COMMAND
                    ********
0B73 4D07                   TZ     13,X'07'
                    ********
                    *BOX£CC*
                    ********
0B74 1080                   SSF
                    ********
                    *BOX£DD*
                    ********
0B75 B027                   CPY    0,T,(S)
0B76 1040                   SPF
0B77 9C50                   DEC    12,C
                    ********
                    ***
                    *BOX£EE*         TEST LOOP VARIBLE P12
                    *********
0B78 4006                   TZ     0,X'06'
0B79 1D7E                   JP     LFF          RETURN
                    ********
                    *BOX£GG*
                    ********
0B7A 8A43                   INC    10,(N)
0B7B 8982                   ADD    9,L,(M)
0B7C 8846                   INC    8,(U)
0B7D 1D72                   JP     LZ
                    ********
                    *BOX£FF*
                    ********
0B7E 1600           LFF     LU     X'00'
0B7F 07E1                   JE     FETCH
```

```
0B7F 07E1          *STORE ROUTINE  *************************************
                   *
                   * THIS IS THE STORE ROUTINE IT£HANDLES ALL STORE COM
                   *
                   *********
                   * BOX£A *
                   *********
0B80 085F          STORE  JE    L,R        SEPERATE (L;R)
                   *********
                   * BOX£B *
                   *********
0B81 9829                 SBT*  11,T,(T)
0B82 B760                 CPY   7,T,I
0B83 CC01                 MOV   12,(T)
0B84 8A23                 ADD   10,T,(N)
0B85 8982                 ADD   9,L,(M)
                   *********
                   * BOX£C *
                   *********
0B86 6DE7                 CP    13,X'E7'
0B87 1D99                 JP    SG      JUMP TO BOX£O: STORE A
0B88 6DE1                 CP    13,X'E1'
0B89 1DAF                 JP    SQ      JUMP TO BOX£Q: STORE INDEX
                   ************
                   * BOX£D&E   *
                   ************
0B8A 6DE0                 CP    13,X'E0'
0B8B 1D99                 JP    SG      JUMP TP BOX£G: STORE X
0B8C 6DDF                 CP    13,X'DF'
0B8D 1D91                 JP    SF      JUMP TO BOX£F: STORE J
                   *                    NO JUMP IMPLIES; STORE ZERO
                   *********
                   * BOX£H *
                   *********
0B8E 8740                 INC   7
0B8F 8740                 INC   7
0B90 1DB6                 JP    SX£     JUMP TO BOX£X
                   *********
                   * BOX£F *
                   *********
0B91 4CFF          SF     TZ    12,X'FF'
0B92 1D97                 JP    *+5
0B93 A750                 WMF   7,D
0B94 1100                 LT    X'00'
0B95 8A43                 INC   10,(N)
0B96 8982                 ADD   9,L,(M)
0B97 2CCD                 LF    12,X'CD'
0B98 1DB4                 JP    SU
```

```
0B98 1DB4           **********
                    * BOX£G *
                    **********
0B99 CC01           SG      MOV   12,(T)
0B9A B820                   CPY   8,T
0B9B 2CC7                   LF    12,X'C7'
0B9C C701                   MOV   7,(T)
0B9D 9C26                   SBT   12,T,(U)
                    **********
                    * BOX£DD*
                    **********
0B9E 5807                   TN   8,X'07'
0B9F 16C1                   LU    X'C1'
0BA0 1DA2                   JP    SJ
                    **********
                    * BOX£I *
                    **********
0BA1 8C46           SI      INC   12,(U)
                    **********
                    * BOX£J *
                    **********
0BA2 6DE7           SJ      CP    13,X'E7'
0BA3 1DA5                   JP    SL        JUMP TO BOX£L:STORE A
                    **********
                    * BOX£K *
                    **********
0BA4 1080                   SSF
                    **********
                    * BOX£L *
                    **********
0BA5 0001           SL      EOT   0,(T)
0BA6 1040                   SPF
                    **********
                    * BOX£M *
                    **********
0BA7 A750                   WMF   7,D
                    **********
                    * BOX£N *
                    **********
0BA8 570F           SN      TN    7,X'0F'
                    **********
                    * BOX£P *
                    **********
0BA9 1DAD                   JP    SEN
                    **********
                    * BOX£O *
                    **********
0BAA 8A43                   INC   10,(N)
```

```
0BAB 8982                        ADD    9,L,(M)
0BAC 1DA1                        JP     SI
                      **********
                      * RETURN TO FETCH ROUTINE
                      **********
0BAD 1600             SEND       LU     X'00'
~~0BAE 07E1~~                        JE     FETCH
                      **********
                      * BOX£Q *
                      **********
0BAF 06C4             SQ         JE     PAGE          GO PAGE IN INDEX£REGISTER
0BB0 1040                        SPF
                      **********
                      * BOX£R *
                      **********
0BB1 5C07                        TN     12,X'07'
0BB2 1DC9                        JP     ST
                      * BOX£S *
                      **********
0BB3 2CC8                        LF     12,X'C8'
                      **********
                      * BOX£U *
                      **********
0BB4 67FE             SU         CP     7,X'FE'
0BB5 1DD1                        JP     SV          JUMP TO BOX£X! P7< OR = 1
                      **********
                      * BOX£X *
                      **********
0BB6 1100             SX£        LT     X'00'
                      **********
                      * BOX£Y *
                      **********
0BB7 67FD             SY         CP     7,X'FD'
0BB8 1DBD                        JP     SZ£         JUMP TO BOX£Z! P7 , OR = 2
                      **********
                      * BOX£AA*
                      **********
0BB9 A750                        WMF    7,D
                      **********
                      * BOX£BB*
                      **********
0BBA 8A43                        INC    10,(N)
0BBB 8982                        ADD    9,L,(M)
0BBC 1DB7                        JP     SY
                      **********
                      * BOX£Z *
                      **********
0BBD 6DDF             SZ£        CP     13,X'DF'
```

```
0BBE 1DC0                     JP     *+2
0BBF 1DAD                     JP     SEND
0BC0 6DE0                     CP     13,X'E0'
0BC1 1DC3                     JP     *+2
0BC2 1DA1                     JP     SI
0BC3 4702                     TZE    7,X'02'
0BC4 1DA1                     JP     SI
0BC5 5701                     TN     7,X'01'
0BC6 1DAD                     JP     SEND
0BC7 2CC9                     LF     12,X'C9'
0BC8 1DA1                     JP     SI
                    *********
                    * BOXET *
                    *********
0BC9 1080           ST        SSF
0BCA C801                     MOV    8,(T)
0BCB 1040                     SPF
                    *********
                    * BOXEW *
                    *********
0BCC A750                     WMF    7,D
0BCD 2CC8                     LF     12,X'C8'
0BCE 8A43                     INC    10,(N)
0BCF 8982                     ADD    9,L,(M)
0BD0 1D86                     JP     SX
                    *********
                    * BOXEV *
                    *********
0BD1 57FF           SV        TN     7,X'FF'
0BD2 1DAD                     JP     SEND
0BD3 8C46                     INC    12,(U)
0BD4 1DA1                     JP     SI
```

```
                          * JRED DECODE ROUTINE *********************************
                          **********
                          * BOX£D *
                          **********
0BD5 4C08                 JRED    TZ     12,X'08'
     0BD5                 JBUS    EQU    JRED
0BD6-0BED                         JE     DRED
0BD7 4C01                         TZ£    12,X'01'  TTY???
0BD8 0BF2                         JE     TBUS
0BD9 5C02                         TN     12,X'02'
0BDA 1DE7                         JP     JCRD
                          **********
                          * BOX£A *
                          **********
0BDB 1125                 JPRNT   LT     X'25'
                          **********
                          * BOX£B *
                          **********
0BDC 2701                         LF     7,X'01'
0BDD 064E                         JE     DIX
                          **********
                          * BOX£D *
                          **********
0BDE 9750                 ,JTEST£DEC     7,C
0BDF 5004                         TN     0,X'04'
0BE0 1DE4                         JP     JBG
                          **********
                          * BOX£E *
                          **********
0BE1 5D04                 JBE     TN    13,X'04'
                          **********
                          * BOX£F *
                          **********
0BE2 07E1                         JE     FETCH
                          **********
                          * BOX£H *
                          **********
0BE3 0CD4                         JE     JR
                          **********
                          * BOX£G *
                          **********
0BE4 5D04                 JBG     TN    13,X'04'
                          **********
                          * BOX£H *
                          **********
0BE5 0CD4                         JE     JR
                          **********
                          * BOX£I *
```

```
                   **********
0BE6 07E1                 JE    FETCH
                   *********
                   * JCRD*
                   *********
0BE7 1387          JCRD   LN    CCNT
0BE8-123F*                LM    HICORE
0BE9 A000                 RMF   0
0BEA 1DEB                 JP    *+1
0BEB B760                 CPY   7,T,I     P7<---CCNT+1
0BEC 1DDE                 JP    ,JTEST
                   **********
                   * DBUS *
                   **********
0BED 1114          DBUS   LT    X'14'     INPUT MAJOR STATUS
     0BED          DRED   EQU   DBUS
0BEE 064C                 JE    ,DIX
                   *   TEST£IF CONTROLLER IS READY
0BEF 5708                 TN    7,X'08'
0BF0 1DE4                 JP    JBG       NOT READY
0BF1 1DE1                 JP    JBE       READY
                   *********
                   * TBUS *
                   *********
0BF2 1377          TBUS   LN    TSTAT
0BF3 123F                 LM    HICORE
0BF4 A000                 RMF   0         READ IN INTERNAL TTY STATUS
0BF5 2701                 LP    7,X'01'   LOAD MASK
0BF6 E720                 AND   7,T       STRIP OFF READY BIT
0BF7 0BDE                 JE    ,JTEST    TEST IF READY OR NOT
```

```
                         * IOC DECODE ROUTINE *********************************
                         *********
                         * BOX£S *
                         *********
0BF8 06FD                      JE    INT
0BF9 1125                IOC   LT    X'25'      INPUT MAJOR STATUS
0BFA 064C                      JE    ,DIX
                         *********
                         * BOX£K *
                         *********
0BFB 5701                      TN    7,X'01'     PRINTER£READY
0BFC 1DF8                      JP    IOC-1
0BFD 5704                      TN    7,X'04'    PRINTER READY
0BFE 1DF8                      JP    IOC-1
                         *********
                         * BOX£M *
                         *********
0BFF 1105                      LT    X'05'      OUTPUT DATA BYTE
0C00 278C                      LF    7,X'8C'    FORM FEED
0C01 0650                      JE    ,DOX
                         *********
                         * BOX£N *
                         *********
0C02 07E1                      JE    FETCH
```

```
                         * IN DECODE ROUTINE **********************************
                         *********
                         * BOXEV *
                         *********
0C03 5C10                IN      TN    12,X'10'
0C04 0C4C                        JE    DIN
0C05 4C01                        TZ    12,X'01'  TTY???
0C06 0C2D                        JE    TIN
0C07 0C09                        JE    RIN
```

```
                    * CARD READER INPUT ROUTINE *************************
                    * RIN INITIATES A READ FROM THE CARD READER
                    * ONE CARD IS READ IN CONCURRENT£MODE
                    *********
                    * BOX£A *
                    *********
0C08 06F0                  JE     INT
0C09 1124           RIN    LT     X'24'      INPUT STATUS BYTE
0C0A 271D                  LF     7,X'1D'
0C0B 064E                  JE     DIX
0C0C 123F                  LM     HICORE
0C0D 4708                  TZ     7,X'08'    IS THE HOPPER EMPTY
0C0E 0C18                  JE     HOPPER     HOPPER IS EMPTY
                    *********
                    * BOX£B *
                    *********
0C0F 9750                  DEC    7,C        SUBTRACT
0C10 5004                  TN     0,X'04'    ZERO RESULT£???
0C11 0C08                  JE     RIN-1
                    *********
                    * BOX£C *
                    *********
0C12 1144                  LT     X'44'      ENABLE CON-CURRENT I/O& INT,
0C13 0654                  JE     COX
                    *********
                    * BOX£E *
                    *********
0C14 278F                  LF     7,CLSB
0C15 2850                  LF     8,80
0C16 1387                  LN     CCNT
0C17 0C47                  JE     CTP
0C18 28C0           HOPPER LF     8,HSAVE
0C19 2AC8                  LF     10,HADD
0C1A 07DA                  JE     ERROR
```

```
                         * OUT DECODE ROUTINE ********************************
                         **********
                         * BOXEXE*
                         **********
0C1B 5C10                OUT      TN     12,X'10'
0C1C 0C4E                         JE     DOUT
0C1D 4E01                         TZ     12,X'01'  TTY???
0C1E 0C2F                         JE     TOUT
0C1F 0C21                         JE     POUT
```

```
                   * PRINTER£OUTPUT£ROUTINE ****************************
                   * THIS ROUTINE INITIATES CONCURRENT£I/O TO THE
                   *   PRINTER
                   *********
                   * BOX£L *
                   *********
0C20 06FD                 JE     INT
0C21 1125          POUT   LT     X'25'     INPUT STATUS BYTE
0C22 277F                 LF     7,X'7F'
0C23 064E                 JE     DIX
                   *********
                   * BOX£M *
                   *********
0C24 37FB                 AF     7,X'FB'   SUBTRACT£5
0C25 47FF                 TZ£    7,X'FF'   TEST READY
0C26 0C20                 JE     POUT-1    NOT READY SO WAIT
                   *********
                   * BOX£N *
                   *********
0C27 11C5                 LT     X'C5'     ENABLE CON-CURRENT I/O
0C28 0654                 JE     COX
                   *******
                   * BOX£P *
                   *********
0C29 2878                 LF     8,120
0C2A 279F                 LF     7,PLSB
0C2B 1397                 LN     PCNT
0C2C 0C47                 JE     CTP
```

```
                         * TTY INPUT / OUTPUT HANDLERS
0C2D 2842                TIN    LP    8,X'42'    SET STATUS TO BUSY ON INPUT
0C2E 1430                       JP    TTY
0C2F 2844                TOUT   LF    8,X'44'    SET STATUS TO BUSY ON OUTPUT
0C30 2777                TTY    LF    7,TSTAT
0C31 123F                       LM    HICORE
0C32 A703                       RMF   7,(N)      INPUT INTERNAL STATUS
0C33 9743                       DEC   7,(N)      DELAY
0C34 BB20                       CPY   11,T       COPY STATUS INTO P11
0C35 4B01                       TZ£   11,X'01'   TEST BUSY BIT,1<=READY
0C36 1442                       JP    TOK        NOT BUSY
                         *            TTY OR DISK IS BUSY MUST WAIT FOR
                         *            PREVIOUS I/O TO FINISH
                         *            THIS CODE IS USED BY BOTH TTY & DISK
0C37 C801                WAIT   MOV   8,(T)
0C38 1080                       SSF
0C39 BB20                       CPY   11,T       SAVE NEW STATUS VALUE IN S11
0C3A 06FD                       JE    INT        GO SERVICE INTERRUPTS
0C3B 1080                       SSF
0C3C CB01                       MOV   11,(T)
0C3D 1040                       SPF
0C3E B820                       CPY   8,T
0C3F 5840                       TN    8,X'40'    TTY OR DISK ???
0C40 1452                       JP    DISK       DISK INSTRUCTION
0C41 1430                       JP    TTY        TTY INSTRUCTION
0C42 C801                TOK    MOV   8,(T)      WRITE OUT NEW STATUS
0C43 A7D3                       WMF   7,I,(N)
0C44 9743                       DEC   7,(N)      LOAD COUNTER£ADDRESS*DELAY
0C45 2846                       LP    8,70       SET UP COUNTER VALUE
0C46 277F                       LF    7,TLSB     LOAD C*I/O ADDRESS(LSB)
0C47 C901                CTP    MOV   9,(T)
0C48 BC20                       CPY   12,T
0C49 CA01                       MOV   10,(T)
0C4A BD20                       CPY   13,T
0C4B 0656                       JE    W,OUT
```

```
                         * DISK INPUT & OUTPUT ROUTINE ***********************
                         * THIS ROUTINE INITIATES DISK I/O IN THE CON-CURRENT
                         * MODE.
                         *********
                         * BOX£L *
                         *********
0C4C 2800                DIN      LF     8,X'00'
0C4D 144F                         JP     DIK
0C4E 2802                DOUT     LF     8,X'02'
                         *********
                         * BOX£K *
                         *********
0C4F 1101                DIK      LT     X'01'      MASK OFF DRIVE NUMBER = 1 OR
0C50 EC20                         AND    12,T
0C51 3C04                         AF     12,X'04'   SET UP QUEUE SEEK BYTE
                         *********
                         * BOX£A *    INPUT MAJOR STATUS
                         *********
0C52 1114                DISK     LT     X'14'
0C53 064C                         JE     ,DIX
                         *********
                         * BOX£B *     TEST MAJOT STATUS TO SEE IF CONTROLLER
                         *             IS READY
                         *********
0C54 5708                         TN     7,X'08'
0C55 1437                         JP     WAIT       NOT READY SO WAIT
                         *********
                         * BOX£C *    QUEUE SELECTED DRIVE
                         *********
0C56 CC01                         MOV    12,(T)
0C57 B720                         CPY    7,T
0C58 1114                         LT     X'14'
0C59 0650                         JE     ,DOX
                         *********
                         * BOX£D *     FILE ACTION BYTE
                         *********
0C5A C801                         MOV    8,(T)
0C5B B720                         CPY    7,T
0C5C 1134                         LT     X'34'
0C5D 0650                         JE     ,DOX
                         *********
                         * BOX£E *    FILE DISK ADDRESS BYTES
                         *********
0C5E 1154                         LT     X'54'
0C5F 0654                         JE     COX
0C60 1080                         SSF
0C61 C501                         MOV    5,(T)
0C62 0652                         JE     DOX
```

```
0C63 1174               LT    X'74'
0C64 0654               JE    COX
0C65 C601               MOV   6,(T)
0C66 0652               JE    DOX
                   *********
                   * BOX£F *   FILE BEGINNING CORE ADDRESS
                   *********-
0C67 C401               MOV   4,(T)
0C68 1040               SPF
0C69 BD20               CPY   13,T
0C6A 1080               SSF
0C6B C301               MOV   3,(T)
0C6C 1040               SPF
0C6D BC20               CPY   12,T
0C6E C901               MOV   9,(T)
0C6F B720               CPY   7,T
0C70 1194               LT    X'94'
0C71 0650               JE    ,DOX
0C72 CA01               MOV   10,(T)
0C73 B720               CPY   7,T
0C74 11B4               LT    X'B4'
0C75 0650               JE    ,DOX
                   *********
                   * BOX£G *   FILE ENDING CORE ADDRESS
                   *********
0C76 FD00               SFL   13         MULTIPLY NUMBER OF MIXAL
                   *                        WORDS TO TRANSFER BY 8
0C77 FC80               SFL   12,L       THIS GIVES THE NUMBER OF
                   *                       MICRODATA BYTES
0C78 FD00               SFL   13
0C79 FC80               SFL   12,L
0C7A FD00               SFL   13
0C7B FC80               SFL   12,L
0C7C CD01               MOV   13,(T)
0C7D 8A20               ADD   10,T       COMPUTE ENDING CORE ADDRESS
0C7E CC01               MOV   12,(T)
0C7F 89A0               ADD   9,T,L      COMPUTE ENDING ADDRESS (MSB)
0C80 9A40               DEC   10
0C81 9981               SBT   9,L,(T)
0C82 B720               CPY   7,T
0C83 11D4               LT    X'D4'
0C84 0650               JE    ,DOX
0C85 CA01               MOV   10,(T)
0C86 B720               CPY   7,T
0C87 11F4               LT    X'F4'
0C88 0650               JE    ,DOX
                   *********
                   * BOX£H *  START QUEUED SEEKS
```

```
                           **********
0C89 1114                        LT     X'14'
0C8A 2790                        LF     7,X'90'
0C8B 0650                        JE     ,DOX
0C8C 07E1                        JE     FETCH      RETURN
```

```
                   * JUMP ROUTINE ****************************************
                   * THIS ROUTINE HANDLES THE ARITHMETIC JUMP INST.
                   * OPCODES 39 - 47
                   *********
                   * BOXEB *
                   *********
0C8D 6DD8          JUMP   CP    13,X'D8'  DECODE
0C8E 14BA                 JP    JP        OPCODE 39
                   *********
                   * BOXEC *
                   *********
0C8F 6DD7                 CP    13,X'D7'  DECODE
0C90 1494                 JP    JG        OPCODE 40 - A COMMAND
                   *********
                   * BOXED *
                   *********
0C91 6DD1                 CP    13,X'D1'  DECODE
0C92 14A4                 JP    JF        OPCODES 41 - 46
                   *********
                   * BOXEE *
                   *********
0C93 1080                 SSF             OPCODE 47
                   *********
                   * BOXEG *     TEST FOR ZERO CONDITION
                   *********
0C94 C611          JG     MOV   6,C,(T)
0C95 C591                 MOV   5,C,L,(T)
0C96 C491                 MOV   4,C,L,(T)
0C97 C391                 MOV   3,C,L,(T)
0C98 C291                 MOV   2,C,L,(T)
0C99 C101                 MOV   1,(T)     MOVE SIGN TO T
                   *********
                   * BOXEH *    THIS SECTION IS COMMON TO OPCODES 40-4
                   *********    THE CONDITION FLAGS HAVE BEEN SET
                   *********        BY A TEST FOR ZERO
                   *********    THE T REGISTER CONTAINS THE SIGN
                   *********        OF THE REGISTER BEING TESTED
                   *********
0C9A 1040          JH     SPF
0C9B B720                 CPY   7,T       PUT SIGN OF REG. IN P7
0C9C 119E                 LT    JAX
                   *********
                   * BOXEI *
                   *********
0C9D 8C2D                 ADD*  12,T,(K)  MULTIPLE WAY BRANCH ON F FIEL
0C9E 14AA          JAXE   JP    JJ        JUMP NEGATIVE
0C9F 14B4                 JP    JO        JUMP ZERO
0CA0 14AF                 JP    JK        JUMP POSITIVE
```

```
0CA1 14B2                JP    JM         JUMP NON-NEGATIVE
0CA2 14AC                JP    JL         JUMP NON-ZERO
0CA3 14B7                JP    JN         JUMP NON-POSITIVE
                *********
                * BOX£F *
                *********
~~0CA4 06C4~~       JF        JE    PAGE       GO PAGE IN INDEX£REGISTER
0CA5 1080                SSF
0CA6 CA11                MOV   10,C,(T)   TEST FOR£ZERO CONDITION
0CA7 C991                MOV   9,C,L,(T)
0CA8 C801                MOV   8,(T)      MOVE SIGN TO T
0CA9 149A                JP    JH
                *********
                * BOX£J *     JUMP NEGATIVE
                *********
0CAA 5780       JJ       TN    7,X'80'
0CAB 07E1                JE    FETCH      POSITIVE - RETURN
                *********
                * BOX£L *       JUMP NON-ZERO
                *********
0CAC 5004       JL       TN    0,X'04'
0CAD 14D4                JP    JR         NON-ZERO RESULT - JUMP
0CAE 07E1                JE    FETCH      ZERO RESULT£- RETURN
                *********
                * BOX£K *       JUMP POSITIVE
                *********
0CAF 5780       JK       TN    7,X'80'
0CB0 14AC                JP    JL
0CB1 07E1                JE    FETCH      NEGATIVE - RETURN
                *********
                * BOX£M *        JUMP NON-NEGATIVE
                *********
0CB2 5780       JM       TN    7,X'80'
0CB3 14D4                JP    JR         POSITIVE - JUMP
                *********
                * BOX£O *       JUMP ZERO
                *********
0CB4 5004       JO       TN    0,X'04'
0CB5 07E1                JE    FETCH      NON-ZERO - RETURN
0CB6 14D4                JP    JR         ZERO - JUMP
                *********
                * BOX£N *      JUMP NON-POSITIVE
                *********
0CB7 5780       JN       TN    7,X'80'
0CB8 14B4                JP    JO
0CB9 14D4                JP    JR         NEGATIVE - JUMP
                *********
                * BOX£P *
```

```
                   *********
0CBA FC00          JP     SFL   12        MULTIPLY P12 BY 2
0CBB 11BD                 LT    JMP
                   *********
                   * BOX£Q *     MULTIPLE WAY BRANCH ON F FIELD
                   *********
~~0CBC 8C2D                 ADD*  12,T,(K)~~
0CBD 14D4          JMP    JP    JR        JUMP
0CBE 1000                 NOP
0CBF 14DC                 JP    JS        JSJ
0CC0 1000                 NOP
0CC1 1080                 SSF             JOV
0CC2 14E2                 JP    JU
0CC3 1080                 SSF             JNOV
0CC4 14E2                 JP    JU
                   *********
                   * BOX£V *
                   *********
0CC5 1140                 LT    X'40'     JL
0CC6 14D0                 JP    JBB
                   *********
                   * BOX£W *
                   *********
0CC7 1120                 LT    X'20'     JE
0CC8 14D0                 JP    JBB
                   *********
                   * BOX£X *
                   *********
0CC9 1110                 LT    X'10'     JG
0CCA 14D0                 JP    JBB
                   *********
                   * BOX£Y *
                   *********
0CCB 1130                 LT    X'30'     JGE
0CCC 14D0                 JP    JBB
                   *********
                   * BOX£Z *
                   *********
0CCD 1150                 LT    X'50'     JNE
0CCE 14D0                 JP    JBB
                   *********
                   * BOX£AA*
                   *********
0CCF 1160                 LT    X'60'     JLE
                   *********
                   * BOX£BB*
                   *********
0CD0 1080          JBB    SSF
```

```
0CD1 E738                 AND*  7,T,C
                   *********
                   * BOX£CC*
                   *********
0CD2 4004                 TZ    0,X'04'   TEST FOR ZERO RESULT
0CD3 07E1                 JE    FETCH     RETURN
                   *********
                   * BOX£R *        JUMP ROUTINE
                   *********
0CD4 1040          JR     SPF               SAVE NEXT£MEMORY ADDRESS
0CD5 CE01                 MOV   14,(T)
0CD6 1080                 SSF
0CD7 BE20                 CPY   14,T
0CD8 1040                 SPF
0CD9 CF01                 MOV   15,(T)
0CDA 1080                 SSF
0CDB BF20                 CPY   15,T
                   *********
                   * BOX£S *       JUMP SAVE J ROUTINE
                   *********
0CDC 1040          JS     SPF
0CDD C901                 MOV   9,(T)
0CDE BE20                 CPY   14,T
0CDF CA01                 MOV   10,(T)
0CE0 BF20                 CPY   15,T
                   *********
                   * BOX£T *
                   *********
0CE1 07E1                 JE    FETCH
                   *********
                   * BOX£U *
                   *********
0CE2 1180          JU     LT    X'80'
0CE3 E738                 AND*  7,T,C
                   *********
                   * BOX£DD*
                   *********
0CE4 4004                 TZ£   0,X'04'
0CE5 14EC                 JP    JFF       OVERFLOW IS OFF
                   *********
                   * BOX£EE*       OVERFLOW FLAG IS SET
                   *********
0CE6 117F                 LT    X'7F'
0CE7 E720                 AND   7,T       RESET OVERFLOW FLAG
0CE8 1040                 SPF
                   *********
                   * BOX£GG*
                   *********
```

```
0CE9 5C02                   TN     12,X'02'
0CEA 14D4                   JP     JR          JOV = JUMP
0CEB 07E1                   JE     FETCH       JNOV = RETURN
                  *********
                  * BOX£FF*
                  *********
0CEC 1040         JFF       SPF
0CED 5C02                   TN     12,X'02'
0CEE 07E1                   JE     FETCH       JOV = RETURN
0CEF 14D4                   JP     JR          JNOV = RETURN
```

```
                    * ENTER ROUTINE     ********************************
                    * THIS ROUTINE HANDLES THE FOLLOWING INSTRUCTIONS;
                    *      INCREMENT, DECREMENT ENTER, & ENTER NEGATIVE,
                    *    OPCODES 48 - 53
                    *********
                    * BOX£A *
                    *********
0CF0 5C01           ENTER  TN    12,X'01'  TEST FOR ENN OR DEC
0CF1 14F4                  JP    EC        ENTA OR INCA COMMAND
                    *********
                    * BOX£B *
                    *********
0CF2 1180                  LT    X'80'     DECA OR ENNA COMMAND
0CF3 D820                  XOR   8,T       FLIP SIGN OF M
                    *********
                    * BOX£C *
                    *********
0CF4 6CFE           EC     CP    12,X'FE'  TEST FOR£ENT OR ENN COMMAND
0CF5 1517                  JP    EP        INC OR DEC COMMAND
                    *********
                    * BOX£D *       ENTA OR ENNA ROUTINE
                    *********
0CF6 6DCF                  CP    13,X'CF'  TEST FOR£A COMMAND
0CF7 14FA                  JP    EE        A COMMAND
0CF8 6DC9                  CP    13,X'C9'  X£COMMAND
0CF9 150A                  JP    EO        INDEX£COMMAND
                    *********
                    * BOX£E *
                    *********
0CFA C801           EE     MOV   8,(T)     X£COMMAND OR A COMMAND
                    *********
                    * BOX£F *
                    *********
0CFB 4D01                  TZ£   13,X'01'
                    *********
                    * BOX£G *
                    *********
0CFC 1080                  SSF             X COMMAND - SELECT SEC, FILES
                    *********
                    * BOX£H *
                    *********
0CFD B120                  CPY   1,T
0CFE 0873                  JE    Z4        ZERO FILES 4 -2
0CFF 1040                  SPF
0D00 C901                  MOV   9,(T)
                    *********
                    * BOX£I *
                    *********
```

```
0D01 4D01                 TZ£    13,X'01'
                    *********
                    * BOX£J *
                    *********
0D02 1080                 SSF              X COMMAND
                    *********
                    ~~* BOX£K *~~
                    *********
0D03 B520                 CPY    5,T
0D04 1040                 SPF
0D05 CA01                 MOV    10,(T)
                    *********
                    * BOX£L *
                    *********
0D06 4D01                 TZ     13,X'01'
                    *********
                    * BOX£M *
                    *********
0D07 1080                 SSF              X COMMAND
                    *********
                    * BOX£N *
                    *********
0D08 B620                 CPY    6,T
0D09 1556                 JP     EQQ
                    *********
                    * BOX£O *    INDEX£COMMAND
                    *********
0D0A 06C4           EO    JE     PAGE      GO PAGE IN INDEX£REGISTER
0D0B C801                 MOV    8,(T)
0D0C 1080                 SSF
0D0D B820                 CPY    8,T
0D0E 1040                 SPF
0D0F C901                 MOV    9,(T)
0D10 1080                 SSF
0D11 B920                 CPY    9,T
0D12 1040                 SPF
0D13 CA01                 MOV    10,(T)
0D14 1080                 SSF
0D15 BA20                 CPY    10,T
0D16 1556                 JP     EQQ
                    *********
                    * BOX£P *              INCA OR DECA ROUTINE
                    *********
0D17 6DCF           EP    CP     13,X'CF'
0D18 151C                 JP     ET        A COMMAND
                    *********
                    * BOX£Q *
                    *********
```

```
0D19 6DC9                  CP    13,X'C9'
0D1A 1558                  JP    ER        INDEX£COMMAND
                    *********
                    * BOX£S *
                    *********
0D1B 1080                  SSF             X£COMMAND
                    *********
                    * BOX£T *
                    *********
0D1C C101           ET     MOV   1,(T)     MOVE SIGN TO T
                    *********
                    * BOX£U *
                    *********
0D1D 1040           EU     SPF
0D1E D830                  XOR   8,T,C     TEST SIGNS, SET C=FLAGS
0D1F CA01                  MOV   10,(T)
                    *********
                    * BOX£X£*
                    *********
0D20 1600                  LU    X'00'     ASSUME SIGNS SAME=SET UP ADD
                    *********
                    * BOX£V *
                    *********
0D21 5004                  TN    0,X'04'   TEST SIGNS
                    *********
                    * BOX£W *
                    *********
0D22 1610                  LU    X'10'     SIGNS NOT SAME=SET UP SUBTRAC
                    *********
                    * BOX£Y *
                    *********
0D23 6DCF                  CP    13,X'CF'
0D24 1528                  JP    EEE       A COMMAND
                    *********
                    * BOX£AA*
                    *********
0D25 6DC9                  CP    13,X'C9'
0D26 155C                  JP    EBB       INDEX£COMMAND
0D27 1080                  SSF             X£COMMAND
                    *********
                    * BOX£EE*
                    *********
0D28 8627           EEE    ADD   6,T,(S)   ADD OR SUBTRACT£DEPENDING ON
0D29 1040                  SPF
0D2A C901                  MOV   9,(T)
                    *********
                    * BOX£FF*
                    *********
```

```
0D2B 4D01                 TZ    13,X'01'
                   *********
                   * BOX£GG*
                   *********
0D2C 1080                 SSF             X COMMAND
                   *********
                   * BOX£HH*
                   *********
0D2D 85A7                 ADD   5,T,L,(S) ADD OR SUBTRACT DEPENDING ON
0D2E 8487                 ADD   4,L,(S)
0D2F 8387                 ADD   3,L,(S)
0D30 8297                 ADD   2,L,C,(S)
                   *********
                   * BOX£NN*
                   *********
0D31 FC80          EII    SFL   12,L      SHIFT LINK BIT INTO S13
0D32 5880                 TN    8,X'80'   TEST SIGNS
0D33 1551                 JP    ELL       SIGNS SAME
                   *********
                   * BOX£JJ*      SIGNS DIFFERENT
                   *********
0D34 4C01                 TZ    12,X'01'  LINK = 1
0D35 1556                 JP    EQQ       RETURN
                   *********
                   * BOX£KK*      LINK = 0, FORM 2'S COMP, FLIP SIGN
                   *********
0D36 1040                 SPF
0D37 5D0F                 TN    13,X'0F'
0D38 153C                 JP    EOO       A COMMAND
0D39 5D08                 TN    13,X'08'
0D3A 1549                 JP    EPP       INDEX£COMMAND
                   *********
                   * BOX£OO*      X£OR A COMMAND
                   *********
0D3B 1080                 SSF
0D3C D660          EOO    XOR   6,T,F
0D3D D560                 XOR   5,T,F
0D3E D460                 XOR   4,T,F
0D3F D360                 XOR   3,T,F
0D40 D260                 XOR   2,T,F
0D41 8640                 INC   6
0D42 8580                 ADD   5,L
0D43 8480                 ADD   4,L
0D44 8380                 ADD   3,L
0D45 8280                 ADD   2,L
0D46 1180                 LT    X'80'
0D47 D120                 XOR   1,T       FLIP SIGN
0D48 1556                 JP    EQQ
```

```
0D48 1556          *********
                   * BOX£PP*    INDEX£COMMAND
                   *********
0D49 1080          EPP    SSF
0D4A D960                 XOR   9,T,F
0D4B DA60                 XOR   10,T,F
0D4C 8A40                 INC   10
0D4D 8980                 ADD   9,L
0D4E 1180                 LT    X'80'
0D4F D820                 XOR   8,T        FLIP SIGN
0D50 1556                 JP    EQQ
                   *********
                   * BOX£LL*    TEST FOR OVERFLOW
                   *********
0D51 5C01          ELL    TN    12,X'01'   LINK = 1 ==> OVERFLOW
0D52 1556                 JP    EQQ        NO OVERFLOW
0D53 1080          OVERFL SSF              OVERFLOW HAS OCCURED
0D54 1180                 LT    X'80'      T <== OVERFLOW BIT
0D55 C720                 LOR   7,T        SET OVERFLOW
                   *********
                   * BOX£QQ*
                   *********
0D56 1600          EQQ    LU    X'00'
0D57 07E1                 JE    FETCH
                   *********
                   * BOX£R *    INDEX£COMMAND
                   *********
0D58 06C4          ER     JE    PAGE       GO PAGE IN INDEX£REGISTER
0D59 1080                 SSF
0D5A C801                 MOV   8,(T)      MOVE SIGN TO T
0D5B 151D                 JP    EU
                   *********
                   * BOX£BB*     INDEX£COMMAND - INC OT DEC
                   *********
0D5C 1080          EBB    SSF
0D5D 8A27                 ADD   10,T,(S)  ADD OR SUBTRACT£DEPENDING ON
0D5E 1040                 SPF
0D5F C901                 MOV   9,(T)
0D60 1080                 SSF
0D61 89B7                 ADD   9,T,C,L,(S) ADD OR SUB DEPENDING ON U
0D62 1040                 SPF
0D63 1531                 JP    EII
```

```
                   * COMPARE ROUTINE *****************************
                   * THIS ROUTINE HANDLES THE COMPARE INSTRUCTIONS
                   * OPCODES 56 - 63
                   *    COMPARE IS DONE BY LOOKING AT THE RESULT
                   *    OF SUBTRACTING M FROM R
                   *         IF NEGATIVE RESLUT ==> [M[ > [R[
                   *         IF-EQUAL RESULT    ==> [M[ = [R[
                   *         IF POSITIVE RESULT ==> [R[ > [M[
                   *********
                   * BOX£A *      CLEAR LEG FLAGS
                   *********
0D64 1080          COMP   SSF
0D65 118F                 LT    X'8F'
0D66 E720                 AND   7,T
0D67 BD00                 ZOF   13          S13 ZERO TEST FILE
0D68 1040                 SPF
0D69 085F                 JE    L,R         SEPERATE (L/R)
                   *********
                   * BOX£B *
                   *********
0D6A 6DC7                 CP    13,X'C7'
0D6B 1577                 JP    CD          A COMMAND
                   *********
                   * BOX£C *
                   *********
0D6C 6DC1                 CP    13,X'C1'
0D6D 1571                 JP    CE          INDEX£COMMAND
                   *********
                   * BOX£F *   X£COMMAND
                   *********
0D6E 1080                 SSF
0D6F C101                 MOV   1,(T)
0D70 1575                 JP    CG
                   *********
                   * BOX£E *
                   *********
0D71 06C4          CE     JE    PAGE        GO PAGE IN INDEX£REGISTER
0D72 1080                 SSF
0D73 C801                 MOV   8,(T)
0D74 BD00                 ZOF   13
                   *********
                   * BOX£G *
                   *********
0D75 1040          CG     SPF
0D76 1578                 JP    CH
                   *********
                   * BOX£D *
                   *********
```

```
0D77 C101          CD      MOV     1,(T)
                   *********
                   * BOX£H *
                   *********
0D78 4C07          CH      TZ      12,X'07'   SIGN REQUIRED??
0D79 1585                  JP      CI         NO SIGN NEEDED
                   *********
                   * BOX£J *                  SIGN IS REQUIRED
                   *********
0D7A 5B07                  TN      11,X'07'   IS THIS THE (0,0) CASE??
0D7B 15DC                  JP      CKK        THIS IS THE (0,0) CASE, SET =
                   *********
                   * BOX£K *
                   *********
0D7C 1080                  SSF
0D7D BC20                  CPY     12,T       SAVE SIGN OF REGISTER
0D7E 1040                  SPF
                   *********
                   * BOX£L *
                   *********
0D7F CA03                  MOV     10,(N)
0D80 C902                  MOV     9,(M)
0D81 ACC0                  RMF     12,I       P12<=P12+1, READ SIGN BYTE
                   *********
                   * BOX£M *
                   *********
0D82 1080                  SSF
0D83 BB20                  CPY     11,T       SAVE SIGN OF M
0D84 1588                  JP      CS
                   *********
                   * BOX£I *
                   *********
0D85 1080          CI      SSF
0D86 2B00                  LF      11,X'00'
0D87 2C00                  LF      12,X'00'
                   *********
                   * BOX£S *
                   *********
0D88 1040          CS      SPF
0D89 CB01                  MOV     11,(T)
0D8A 8A23                  ADD     10,T,(N)
0D8B 8982                  ADD     9,L,(M)
0D8C CC01                  MOV     12,(T)
0D8D 9829                  SBT*    11,T,(T)
0D8E B760                  CPY     7,T,I
                   *********
                   * BOX£U *
                   *********
```

```
0D8F 8B49                 INC*  11,(T)
0D90 B826                 CPY   8,T,(U)
                   *********
                   * BOX£T *    SEPERATE INDEXING COMMANDS
                   *********
0D91 6DC7                 CP    13,X'C7'
~~0D92 1595                 JP    CU1~~         A COMMAND
0D93 6DC1                 CP    13,X'C1'
0D94 15A9                 JP    CBB         INDEX
                   *********
                   * BOX£U1*
                   *********
0D95 A000          CU1    RMF   0
0D96 4D07                 TZ£   13,X'07'
0D97 1080                 SSF
0D98 903F                 SBT*  0,T,C,(S)
0D99 159E                 JP    CY
                   *********
                   * BOX£V *
                   *********
0D9A A000          CV     RMF   0
                   *********
                   * BOX£W *
                   *********
0D9B 4D07                 TZ    13,X'07'
                   *********
                   * BOX£X£*
                   *********
0D9C 1080                 SSF
                   *********
                   * BOX£Y *
                   *********
0D9D 90BF                 SBT*  0,T,L,C,(S)
0D9E 1080          CY     SSF
0D9F CD20                 LOR   13,T
0DA0 1040                 SPF
0DA1 37FF                 AF    7,X'FF'   DECREMENT£P7
                   *********
                   * BOX£Z *
                   *********
0DA2 57FF                 TN    7,X'FF'
0DA3 15D3                 JP    CJJ         P7 IS ZERO
                   *********
                   * BOX£AA*  P7 IS NON=ZERO
                   *********
0DA4 9A43                 DEC   10,(N)
0DA5 9982                 SBT   9,L,(M)
0DA6 38FF                 AF    8,X'FF'   DECREMENT£U & P8
```

```
0DA7 C806                MOV    8,(U)
0DA8 159A                JP     CV
                *********
                * BOX£BB*
                *********
0DA9 3804       CBB      AF     8,X'04'
0DAA 6BFC                CP     11,X'FC'
0DAB 15C3                JP     CG1-1
0DAC 3BFD                AF     11,X'FD'  SUBTRACT£3
0DAD C806                MOV    8,(U)
0DAE BC10                ZOF    12,C
                *********
                * BOX£C1*
                *********
0DAF A000                RMF    0
0DB0 1080                SSF
0DB1 903F                SBT*   0,T,C,(S)
0DB2 15B8                JP     CHH
                *********
                *********
                * BOX£CC*
                *********
0DB3 5BFF       CCC      TN     11,X'FF'
0DB4 15C8                JP     CGG
                *********
                * BOX£FF*
                *********
0DB5 A000                RMF    0
0DB6 1080                SSF
                *********
                * BOX£HH*
                *********
0DB7 90BF                SBT*   0,T,L,C,(S)
0DB8 CD20       CHH      LOR    13,T
0DB9 1040                SPF
0DBA 37FF                AF     7,X'FF'   DECREMENT£P7 BY 1
0DBB 3BFF                AF     11,X'FF'  DECREMENT£P11 BY 1
0DBC 9A43                DEC    10,(N)
0DBD 9982                SBT    9,L,(M)
                *********
                * BOX£DD*
                *********
0DBE 38FF                AF     8,X'FF'
0DBF C806                MOV    8,(U)
                *********
                * BOX£N *
                *********
0DC0 57FF                TN     7,X'FF'
```

```
0DC1 1503                    JP     CJJ
0DC2 1583                    JP     CCC
                   * BOX£G1*
                   *********
0DC3 BC10                    ZOF    12,C
0DC4 A000          CG1       RMF    0
0DC5 1000                    NOP                 WAIT FOR£T REGISTER
0DC6 9C38                    SBT*   12,T,C
0DC7 15CB                    JP     CII
                   *********
                   * BOX£GG*
                   *********
0DC8 A000          CGG       RMF    0
0DC9 1000                    NOP
                   *********
                   * BOX£II*
                   *********
0DCA 9CB8                    SBT*   12,T,L,C
0DCB 1080          CII       SSF
0DCC CD20                    LOR    13,T
0DCD 1040                    SPF
0DCE 37FF                    AF     7,X'FF'      DECREMENT£P7 BY 1
0DCF 9A43                    DEC    10,(N)
0DD0 9982                    SBT    9,L,(M)
                   *********
                   * BOX£EE*
                   *********
0DD1 47FF                    TZ     7,X'FF'
0DD2 15C8                    JP     CGG
                   *********
                   * BOX£JJ*
                   *********
0DD3 5004          CJJ       TN     0,X'04'
0DD4 15DF                    JP     CLL          NON=ZERO RESULT
                   *********
                   * BOX£UU*      ZERO RESULT
                   *********
0DD5 1080                    SSF
0DD6 5DFF                    TN     13,X'FF'
0DD7 15DC                    JP     CKK          BOTH OPERANDS = 0; SET =
0DD8 CB01                    MOV    11,(T)
0DD9 DC38                    XOR*   12,T,C
                   *********
                   * BOX£VV*
                   *********
0DDA 5004                    TN     0,X'04'
0DDB 15E2                    JP     CMM
                   *********
```

```
                    * BOX£KK*       ZERO RESULT ==> [M[ = [R[
                    *********
0DDC 1120           CKK     LT      X'20'
0DDD 1080                   SSF
0DDE 15E7                   JP      CQQ
                    *********
                    * BOX£LL*
                    *********
0DDF FD80           CLL     SFL     13,L        SHIFT LINK INTO P13
0DE0 4D01                   TZ£     13,X'01'    TEST LINK BIT
0DE1 15EA                   JP      CNN         POSITIVE RESULT
                    *********
                    * BOX£MM*        NEGATIVE RESULT ==> [M[ > [R[
                    *********
0DE2 1140           CMM     LT      X'40'
0DE3 1080                   SSF
                    *********
                    * BOX£OO*
                    *********
0DE4 2D50                   LF      13,X'50'
0DE5 4B80                   TZ£     11,X'80'
                    *********
                    * BOX£SS*
                    *********
0DE6 DD29                   XOR*    13,T,(T)    FLIP SETTING
                    *********
                    * BOX£QQ*       SET LEG FLAGS AND RETURN
                    *********
0DE7 C720           CQQ     LOR     7,T
0DE8 1600                   LU      X'00'
                    *********
                    * BOX£RR*
                    *********
0DE9 07E1                   JE      FETCH       RETURN
                    *********
                    * BOX£NN*       POSITIVE RESULT ==> [R[ > [M[
                    *********
0DEA 1110           CNN     LT      X'10'
0DEB 1080                   SSF
                    *********
                    * BOX£PP*
                    *********
0DEC 2D50                   LF      13,X'50'
0DED 4C80                   TZ£     12,X'80'
0DEE DD29                   XOR*    13,T,(T)    FLIP SETTING
0DEF 15E7                   JP      CQQ
```

```
0DF0 0000                        END    0
```

VII. CONCLUSIONS

As with most projects of any size, several conclusions can be drawn by looking back at the effort as a whole. The conclusions drawn here deal not only with the development of the project but offer some evaluation of the Microdata 1600/30 and the MIX 1009 computers.

Regarding the Microdata 1600/30 as a tool for emulation, the following points can be made concerning the relation between the 1600/30 and the target machine:

1. Unless the target machine has an 8 bit byte, emulation will not be efficient. This results from problems with byte allignment as well as difficiulities implementing arithmetic operations on the Microdata's 8 bit ALU.
2. Unless the word size of the target machine equals 2^N Microdata bytes, $N = 1,2,3,\ldots,$ word boundary control will not be efficient.
3. Unless

   $$N * M + 2 * P < 30,$$

   where N = number of Microdata bytes per
   target machine word,
   M = number of full word registers in
   the target machine,
   P = number of address registers, (i.e.
   index registers),

emulation must necessarily involve register paging.

Regarding the MIX 1009 computer as a target machine which is to be emulated, four conclusions can be drawn.

1. The five byte word implies a degree of firmware inefficiency when the host machine is a binary computer.
2. The requirement that a byte assume 64 to 100 states is restrictive in view of many present architectures. This restriction, if followed, forces the use of a 6 bit byte on all implementations using a binary host machine.
3. The character code adheres to no standard.
4. Sign plus magnitude is a somewhat obsolete architecture but results in no major firmware problems.

Regarding the development ot the project as a whole the following points are presented:

1. Initially, Knuth's architecture was considered inviolable and many of the early firmware coding problems were due to strict adherence to Knuth's design.
2. With the passing of time and with increasing experience in the cost of implementing all parts of Knuth's design, his architecture

was considered less and less inviolable.

3. The resulting MIX computer, with 8 bit bytes and ASCII code is not only easier to emulate but represents an instructional computer whose architecture is more compatable with commercially available machines.

REFERENCES

(1) MIX, publication number AL 1/73 03808, Addison-Wesley Series in Computer Science and Information Processing, 1970.

(2) Microdata, Computer Reference Manual, Microdata 1600/30, publication number RM 20001630-1, Microdata Corporation, 1973.

(3) Microdata, Micro 1600 Computer Reference Manual, publication number 71-1-1600-001, Microdata Corporation, 1971.

(4) Microprogramming Handbook, Second Edition, Microdata Corporation, 1972.

(5) Knuth, D.E., Fundamental Algorithms Vol. 1, The art of Computer Programming, (Varga, R.S. and Harrison, M.A., eds.), pp. 120-153, Reading, Mass., 1969.

(6) Richards, R.K., Digital Design, (Richard, R.K. ed.), pp. 341-368, New York, N.Y., 1971.

(7) King, W. and Dennis, T.D., "A Paging System for the Control Memory in a Minicomputer System", COMPCON 75, Tenth IEEE Computer Society International Conference, San Francisco, California, Feb. 25-27, 1975.