APPLICATION NOTES

Fairchild

F8

Microprocessor

TABLE OF CONTENTS

BIT MANIPULATION

Abstract

The following application note describes the testing of various conditions (or bit pattern) in a register. It is hopeful that this will clarify the usage of bit testing in F8.

Description

There are four condition codes in F8, namely-positive, zero, overflow and carry. The user can test for any of the above four or any combination of the above four. The logical implication of these four bits are as follows:

(a) Positive: set by the most significant bit of the result being zero.

(b) Zero: set by the result being zero.

(c) Carry: set by the carry coming out of the most significant bit.

(d) Overflow: set by exclusive R the carry from the 7th bit and 8th bit. This is an indication of a change of sign in signed arithmetic.

The following are for testing bits and values of registers.

(a) Test for 1 in 6th bit of register.

```
LR    A, R
NI    B'00100000'
BNZ
```

(b) Test for 1 in 6th or 4th bit.

```
LR    A,R
NI    B'00101000'
BNZ
```

(c) Test for 1 in 6th and 4th bit.

```
LR    A, R
NI    B'00101000'
XI    B'00101000'
BZ
```

(d) Test for 6th bit in register being 0.

```
LR    A,R
NI    B'00100000'
BZ
```

(e) Test for 1 in 6th or 4th bit.

```
LR    A,R
NI    B'00101000'
XI    B'00101000'
BNZ
```

(f) Test for 0 in 6th and 4th bit.

```
LR   A,R
NI   B'00101000'
BZ
```

(g) Test to see if register is smaller than 128.

```
CI   B'10000000'
BC
```

(h) Test to see if register is greater than 254.

```
LR   A,R
CI   B'11111110'
BNC
```

For 2s complement arithmetic, the testing involves both positive and overflow bit.

(as shown in following examples).

(i) Test to see if reg R is greater than B'01110000' (+112).

```
LR   A,R
CI   B'01110000'
BM   in this case, there is no need for
     testing overflow.
```

(j) Test to see if reg R greater than B'10000000' (-128)

```
LR   A,R
CI   B'10000000'
BM   in this case, there is no need to
     test for overflow.
```

(k) Test to see if reg R is less than B'01110000' (+112).

```
        LR   A,R
        CI   B'01110000'
        BNO  Test 2
        BM   Loc
Test 2  BP   Loc
```

(l) Test to see if reg R is less than B'10000000' (-128).

```
        LR   A,R
        CI   B'10000000'
        BNO  Test 2
        BP   Loc
Test 2  BM
```

The general relation in testing 2's complement values is such that

(1) Acc $\geq$ value if overflow and positive are the same.

(2) Acc $\leq$ value if overflow and P are different.

(3) Overflow will not be generated unless a 2 different signed number are compared.

The following condition code table summarizes various comparisons.

Acc	Value	Result	Condition Code (OZCP)
+ve	+ve	A > B	0000
+ve	+ve	A = B	0111
+ve	+ve	A < B	0011
-ve	-ve	A > B	0000
-ve	-ve	A = B	0111
-ve	-ve	A < B	0011
+ve	-ve	A > B A > B	0010 1011 } (depending on operand values)
-ve	+ve	A < B A < B	0001 1000 } (depending on operand values)

COMPARE INSTRUCTIONS

Abstract

The following application note describes the use of the compare instructions "CM" and "CI" in performing magnitude or Algebraic comparison between two signed or unsigned numbers.

Description

F8 Microprocessor instruction set has two instructions for comparing two numbers: 'CM' and 'CI'. In 'CM' numbers compared are Memory content addressed by DC register and accumulator, while in 'CI' numbers compared are Immediate Data (second byte of instruction) and accumulator. For this discussion, accumulator content is noted by 'A' while memory content or immediate operand is noted by 'B'.

When "compare" instruction is being performed, condition codes are set of the result of following arithmetic operation: B is added with its complement of A, and 1 is added to the result.

$$\text{Result} = B + \bar{A} + 1$$

Compare instructions do not affect accumulator contents.

Example $A = 45_{16}$
$B = 47_{16}$

	0100 0111	47_{16}
Result =	1011 1010	Com 45_{16}
	1	Add 1
C "1"	0000 0010	

Hence the flags set would be OZCP
0011

(overflow is exclusive OR of carries from sixth and seventh stages). "0" in this example because a carry was present from the sixth and seventh stages.

As the eight bits of data in A and B can be treated as signed number (2's complement) or unsigned number (only positive numbers) it is important to know various combinations of condition codes set by different values of A and B. Following table summarizes such various combinations.

Accumulator A	Number to be compared with B	Unsigned Magnitude	Signed Magnitude	Remarks	Condition Codes OZCP
Pos.	Pos.	$A > B$	$A > B$		0000
Pos.	Pos.	$A = B$	$A = B$		0111
Pos.	Pos.	$A < B$	$A < B$		0011
Neg.*	Neg.*	$A < B$	$A < B$		0011
Neg.	Neg.	$A = B$	$A = B$		0111
Neg.	Neg.	$A > B$	$A > B$		0000
Pos.	Neg.	$A < B$	$A > B$	without OVF	0010
		$A < B$	$A > B$	with OVF	1011
Neg.	Pos.	$A > B$	$A < B$	without OVF	0001
		$A > B$	$A < B$	with OVF	1000

*Negative numbers are with Bit 7 = 1

Following ovservations can be obtained from above table.

1. If two numbers are to be compared for equality only "Z" flag is to be checked irrespective of type of numbers.

2. If unsigned magnitudes to be compared between two numbers, carry bit indicates the difference in two numbers. If carry bit is set: $A \leq B$. If carry bit is not set: $A > B$.

3. If signed magnitudes to be compared between two numbers (any combination) both overflow and sign bits are to be checked.

   $A > B$ if both overflow and P are 1 or are zero ($0 \oplus P = 0$)

   $A \leq B$ if overflow and P are not equal ($0 \oplus P = 1$)

Programs for Comparisons

1. For comparing A = B (sign or unsigned)

A is in accumulator

```
            CI      B        compare with B
            BZ      Equal    if zero flags set go to equal
            ------------     unequal
```

2. For comparing A > B (unsigned)

```
            CI      B
            BNC     Y1       if A greater than B--go to Y1
            ---------        A is either less than or equal to B

      Y1
```

3. For comparing A ≥ B (unsigned)

```
            CI      B        compare with B
            BNC     Y1       If A greater than B go to Y1
            BZ      Y1       If A = B go to Y1
            =========        Now A is less than B


      Y1                     A is greater than or equal to Y1
```

4. Comparing two numbers algebraically

As per previous discussion to compare two signed numbers, exclusive or function of sign and overflow bit is required, one method of doing such function is shown below.

```
            CI      B        A is in accumulator
                             B is compared with A
            BF9     GRT      If both OVF and POS are
                             zero A > B hence branch to GRT

            BNO     Y1       To come  to this inst--either
                             overflow = 1 or Positive = 1.  If
                             overflow = Ø, positive is equal to 1
                             hence jump to Y1

            BN      GRT      Now overflow = 0, if positive =0,
                             A > B
   Y1                        At this point either OVF = 1, or POS = 1
                             hence A ≤ B
```

GRT Now OVF and Pos are both equal to zero or both equal to 1

Hence $A > B$.

Abstract

A Memory Interface Chip (3852 or 3853) may be used to expand the amount of read/write memory in an F8 system up to 64K bytes. Both MI chips minimize the number of additional external parts necessary to build a working RAM interface. This paper details two system designs; one using the 3852 MI with 4K dynamic RAM, and one using the 3853 with 1K static RAM.

Static RAM Interface

The 3853 Memory Interface Chip is designed for use in interfacing various types of static RAM with an F8 system. This chip provides the following signals:

16 Address Lines - allowing up to 64K bytes of RAM expansion

CPU READ - a signal for gating data from RAM onto the F8 Data Bus lines

$\overline{\text{RAM WRITE}}$ - a memory WRITE signal

REGDR - a pin allowing external selection of the address space driven by the MI chip.

Figure 1 shows a typical system using the 3853 to connect 8K of static 2102 RAM to an F8 CPU which has 2K of F8 ROM (3851).

Connections between the 3853 and the CPU are the same as those between the F8 ROMs and CPU. Namely, 5 ROMC control lines, 8 Data Bus lines and two timing lines (Ø and WRITE) not shown on the Figure. Thus, the MI chip looks just like another ROM chip to the CPU.

Because the Data Bus lines in an F8 system are bi-directional, any chip driving this bus must be 3-state. The CMOS devices labeled 340097 on Figure 1 perform this function. Data coming from 2102 RAM is buffered through the 340097's which have an external control line for forcing them into a high output impedance state. This control line is, in turn, driven by the MI chip signal CPU READ which is ANDed with a PAGE SELECT signal derived externally. Thus, when the Data Bus Line is to be driven by RAM (during, say, an instruction fetch), the 3853 raises the CPU READ line and if the PAGE SELECT line is true, information from RAM is gated onto the Data Bus. In all other cases the CPU READ line holds the buffers in high impedance output state. Note that for RAMs which have externally controllable 3-state capability, these buffers may not be required provided the total line maximum capacitance of 100 pf. is not exceeded on the Data Bus. In this case the CPU READ signal would be connected directly to the RAM 3-state control input.

The CMOS buffers (34050) shown driving the Data Bus out to the RAM array in Figure 1 may or may not be required depending on the total line capacitance. If the capacitance on each Data Bus line can be held to under 100 pf. total, including ROM, MI, RAM inputs and interconnect, then these buffers may be eliminated.

The address space of RAM driven by the MI chip is defined externally with decoding logic. The low order ten bits of address coming from the MI are connected directly to address line inputs of the RAM chips. Drivers on the Memory Interface are designed to drive a 500 pf. load on each address line. Higher order address lines (ADDR 1Ø - 13) are then used in this design to determine memory pages. Each page is 1K bytes of memory. To do this, these lines are decoded by a 9315 (1 of 10 decoder) which directly generates Chip Enable signals for the RAM. Note that the first two pages of memory are used by F8 ROM (i.e. mask options have defined their ROM contents as occupying page zero and page one) and hence connot be assigned to a page of RAM.

All of the RAM chip enable signals are then effectively ORed together by the 9N30 and a PAGE SELECT signal is formed. This signal is true whenever a page of RAM is being addressed by the F8 system and is used with the CPU READ signal to gate Memory data onto the Data Bus. PAGE SELECT is also fed back into the MI chip thru an open collector buffer (shown as 9N17) tied to the REGDR pin. This pin is wire-ANDed internally (an internal pullup is also provided) with other signals and used to determine when the MI chip should be driving the Data Bus.

The system shown in Figure 1 is designed to operate at a maximum Ø clock rate of 2 MHz. To meet this, the 2102 RAM used must have a maximum access time of 650 nS.

Further RAM expansion than that shown here may be accomplished in any one of several ways. As long as the current and capacitance load driving capabilities of the various chips used is not exceeded, one simple way of further expansion is to perform more extensive decoding of the high order address lines from the MI (say perhaps with a 1 of 16 decoder, or several). RAM pages are then just stacked deeper. Another way of expanding would be to use more than one MI chip in a system. The only requirement when doing this is that the address space assigned to each MI (by the external address decoding logic) be different. In this case, each MI chip would have its own set of Data Bus buffers as well as address decoding logic and RAM array.

It should be noted here that in a system which mixes F8 ROM and Memory Interface chips, a great deal of care must be taken when using the second DC of the MI because the ROM contains only a single DC. This involves the use of the SWAP instruction which exchanges the contents of the two DC's on the MI but is a no-op to the F8 ROM since it does not have a second DC. (See the F8 Manual for further details). As a result, it may be desirable in some systems to use standard ROM chips to replace some RAM on the outside of an MI chip instead of using F8 ROMs. This gains the full advantages of the MI's dual DC but sacrafices the I/O ports, Internal timers and External Interrupt facilities of the F8 ROMs.

Dynamic RAM Interface

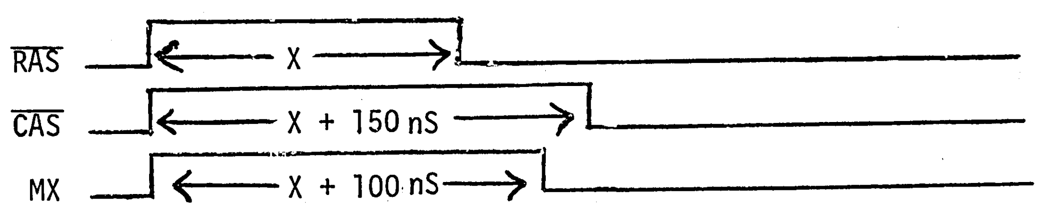
The 3852 Memory Interface Chip is designed for use in interfacing dynamic RAM with an F8 system. In addition to the signals generated by the 3853, the 3852 provides the following two signals:

CPU SLOT - used to divide a CPU instruction cycle in half. During the first half the CPU uses RAM; during the second half, RAM is refreshed.

CYCLE REQUEST - used to signal the beginning of a RAM access cycle. Each CPU cycle is divided into two sections by the CPU SLOT signal (except during execution of the ST instruction). The first section is always used for the CPU's access to RAM. The second section is then used every fourth or eighth time by the 3852 to refresh the dynamic RAM. The 3853, thus, internally multiplexes the low order Address lines between the correct RAM address (contents of the Program Counter) and an internal 6-bit refresh counter.

Figure 2 details a system design using the 3852 to connect 32K of Dynamic RAM (the 16 pin version - 4096) with an F8 CPU and 4K of F8 ROM. This design has a good deal in common with that shown in Figure 1 for static RAMs. Several other tasks must now be handled by the MI chip, however, because of the RAM refreshing requirements.

Since the 16 pin 4K RAMs are being used in this configuration it is necessary to multiplex the 12 address lines on 6 lines to be sent to the RAM chips. This may be simply accomplished using 3 9N51 (AOI) chips. The MI signal called CYCLE REQUEST is then used, thru a one-shot (9602) to generate the control gating signals for the multiplexing. CYCLE REQUEST is also used to generate the $\overline{RAS}$ and $\overline{CAS}$ strobes for the RAM chips. The timing for these signals is as follows:

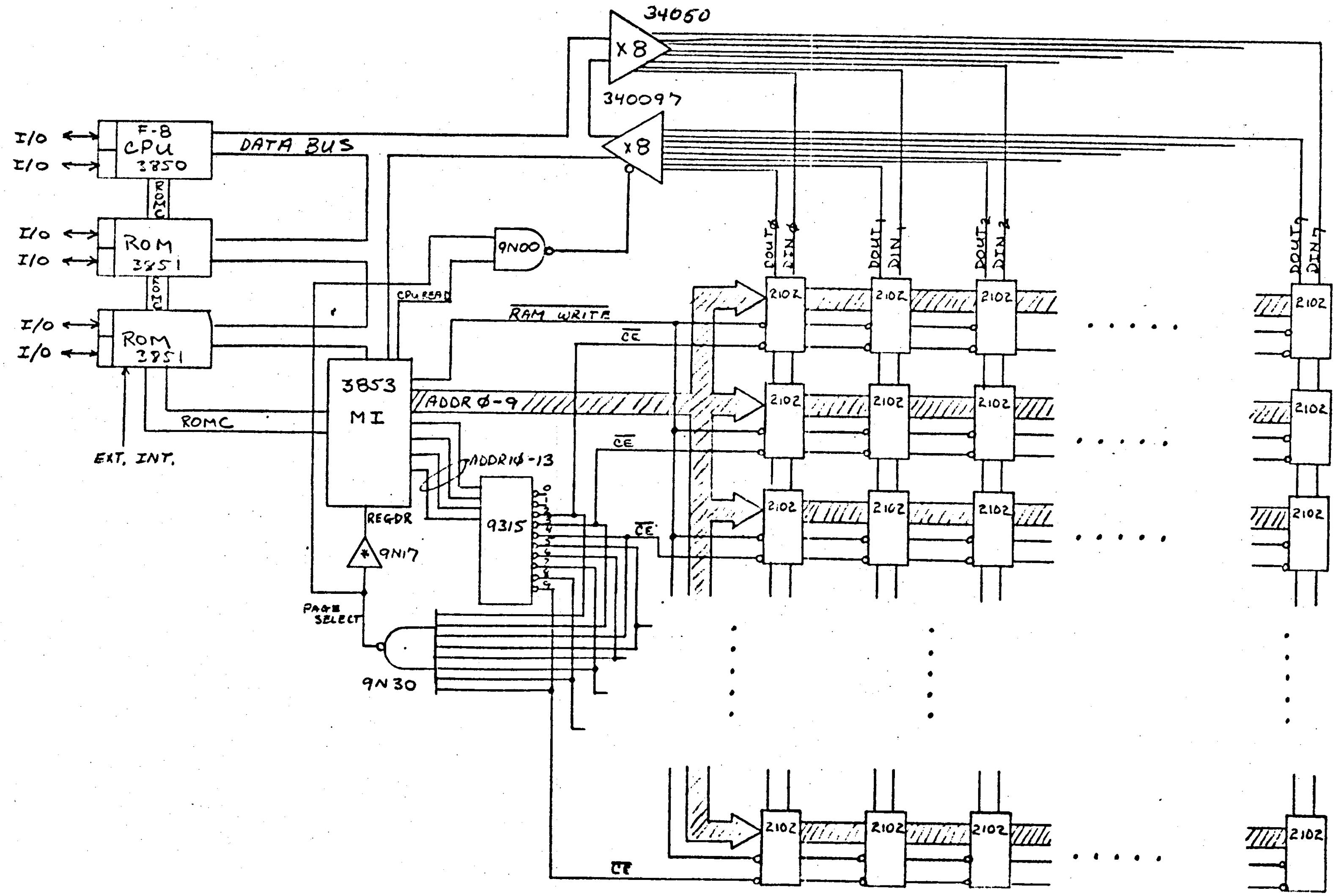


where a typical value for X is ≈ 300 nS and is the address set-up time on the MI chip.

During a normal CPU cycle when information is being fetched from RAM, it is necessary for the information to be held on the Data Bus lines for the entire length of the cycle. This means that in order to divide a machine cycle in half (and use the second part for refreshing RAM) it is necessary to store the information fetched during the first part of the cycle. In the design of Figure 2,

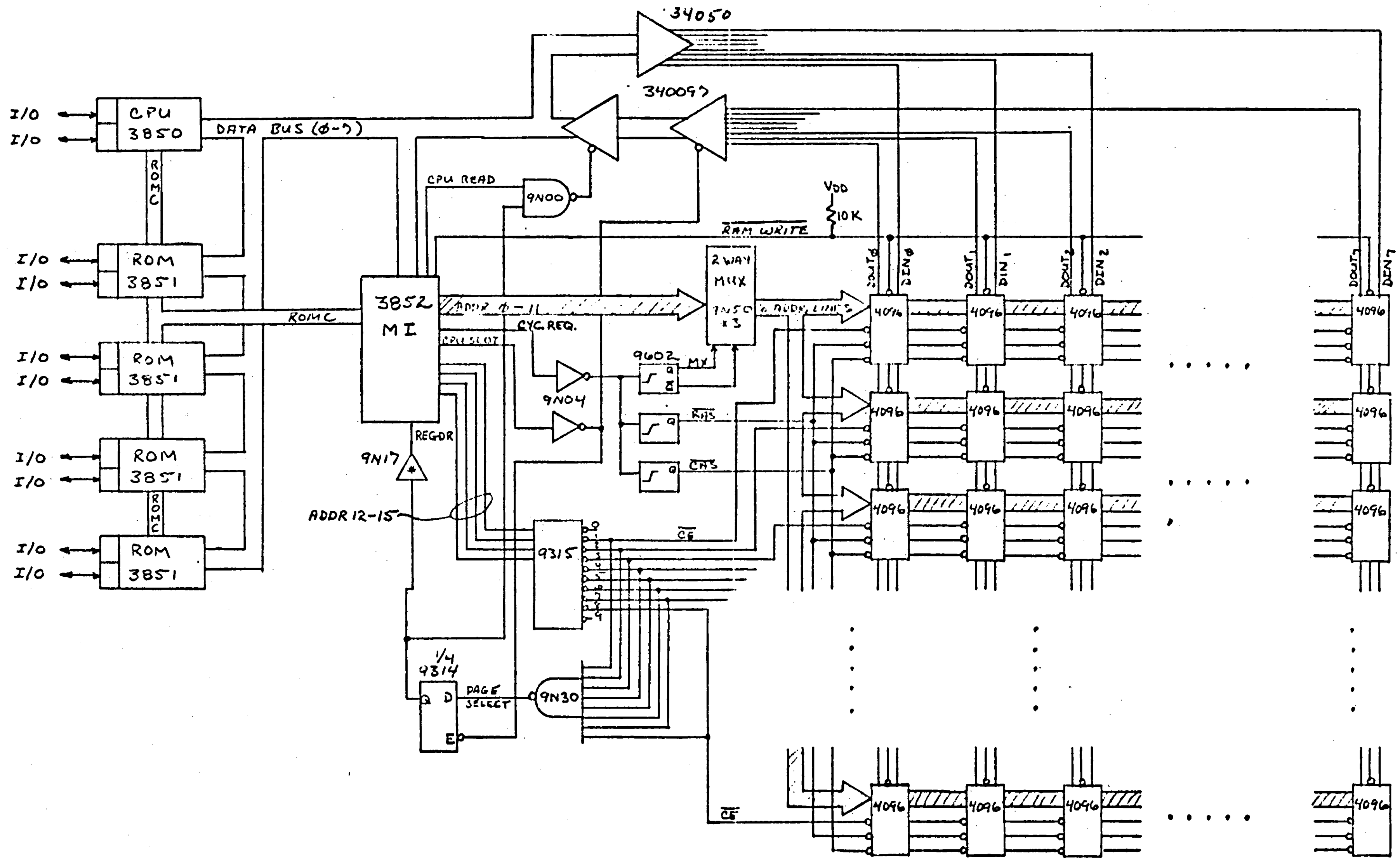
FIGURE 1

2 K ROM + 8 K RAM CONFIGURATION USING 2102's



GURE 2

4K ROM + 32K RAM·CONFIGURATION USING MK4096



this is accomplished for the Data Bus lines by using the CPU SLOT signal and two levels of CMOS buffering (340097's). During the first part of a cycle, CPU SLOT is true and information from RAM is passed onto the node between the two buffers. When the second part of the cycle begins (during which time RAM will be refreshed), CPU SLOT goes low temporarily latching the RAM information onto the node between the two buffers. This allows the data to remain constant on the Data Bus lines for the entire length of the cycle. An ordinary latch may also be used instead of the second level CMOS buffer described here.

Page select information must also be latched since the address lines will be changing during the second part of the cycle. One latch from a 9314 used in D-mode performs the desired function.

The system shown in Figure 2 is designed to operate with a Ø clock rate of 2 MHz. A maximum RAM access time of 350nS is required to meet this design goal.

As with the 3853 system, further expansion of RAM in a 3852 interfaced system may be accomplished in any one of several ways (further address, decoding and RAM array stacking or use of multiple MI chips). Also, the necessity of the buffers (340050) driving the Data Bus to the RAM array is contingent upon the total capacitance on those lines.

Conclusions

The 3852 and 3853 Memory Interface Chips provide an easy means of expanding the amount of Read/Write memory in an F8 system. The chips provide all timing and control signals necessary for dynamic RAM interfacing and static RAM interfacing, respectively, of most common memory chips available today. This minimizes the number of additional external circuits required and hence minimizes the overall system cost and complexity. For detailed information on the electrical and timing characteristics of the Memory Interface chips, refer to the F8 manual.

MULTIPLEXING F8 I/O PORTS

Abstract

Each I/O Port in an F8 system is 8 bits wide. Although the F8 CPU and F8 ROM each offer two I/O ports per circuit, some applications may require an expansion of I/O capability. This may be easily accomplished with standard multiplexing techniques. Three cases of I/O expansion are examined. First, consideration is given to expanding output latches of an F8 system. Next, expanding the number of inputs into an F8 is discussed. Finally, expansion of both outputs and inputs in the same system is described. All of these techniques employ MSI circuits connected to the F8 I/O port externally.

Expanding F8 Output Latches

A simple expansion of F8 I/O ports may be accomplished using two F8 I/O ports. One port will be used as the selector while the other is needed for data transfer. Using this scheme, a total of 128 output latches may be realized. The cost of this is twofold; extra circuits and additional programming.

For the purpose of analysis, this paper will expand the number of F8 output latches to eight; considerations for higher and lower expansion will also be discussed. Two types of MSI circuits are needed to accomplish this expansion. Each 8-bit I/O port will be a dual 4-bit latch. The 8-bit output latch is realized using the 9388 Dual Four-Bit Latch. A 1 of 10 decoder, the 9301, is also required.

The expansion of F8 I/O to 8 output ports is shown in Figure 1. Three leads of I/O Port 0 of the CPU are used to select one of ten ports. A fourth lead, bit 04, is used as an enable line by the latches. This bit will be used to control data entry into the latches. Port 1 of the CPU transfers data from the microprocessor to the latches. Using this method of I/O expansion, a new sequence of instructions are necessary to output an 8-bit byte from the accumulator of the CPU to the latches. Two types of F8 instructions are used. One of these is the Load Immediate Short (LIS) instruction.

LIS a

This instruction will load the low order four bits of the accumulator with the four bit operand "a". The upper four bits of the accumulator will be set to zero. The other instruction is the Output Short instruction (OUTS) which transfers the content of the accumulator to the port specified by the four bit operand "a".

OUTS a

Each eight bit transfer to an expanded I/O latch involves five instructions. First, the data must be transferred from the accumulator to I/O port 1 of the CPU. This may be accomplished with an OUTS 1 instruction. Next, the desired latch must be selected by loading CPU port 0 with the proper code. At this time, the enable line, bit 4, must be lowered. Performing this operation allows the data to be accepted by the desired latch. This operation requires two instructions.

LIS $1X_2X_1X_0$

OUTS 0

Note: $X_2X_1X_0$ is the three bit address of the output latch.

The LIS instruction will load the accumulator with the latch address and the enable bit; the OUTS instruction transfers the content of the accumulator to I/O port 0. Finally, the enable line of the latches must be brought high, so that the contents of the latch may be maintained. This may be done by clearing port 0 as shown below:

LIS 0

OUTS 0

The sequence of instructions for an output transfer becomes:

OUTS 1	Put data on Port One
LIS $1X_2X_1X_0$	Select a latch, lower the enable line
OUTS 0	Put the select bits on port 0
LIS 0	Clear the accumulator
OUTS 0	Raise the enable line port 0

The first instruction makes the data available for all of the latches. The third instruction selects the desired latch and drops the enable line, allowing the desired latch to accept the data. The last instruction raises the enable line so that the output data is held. All five instructions are one byte each. The OUTS instruction require two cycles to execute while the LIS instruction only requires one. Thus, each output to an expanded latch is a five byte operation executing in eight cycles.

Output Macro

A macro may be developed to handle a data transfer to these external latches. Suppose that a byte of data is already in the accumulator. A suitable output macro would be:

```
MACRO
OUTPUT &SEL     Macro Name
OUTS 1          Output Data From Accumulator
LIS &SEL        Put Selection Code in Accumulator
OUTS 0          Output Selection Code to Port 0
LIS 0           Clear Accumulator
OUTS 0          Raise Latch Enable Line
MEND
```

The argument, &SEL, is the selection code for the desired latch. When used within a program, this macro will transfer the accumulator into the latch designated by SEL. The content of the accumulator is cleared after this operation, as is the content of I/O Port 0. The data that was in the accumulator is stored in Port 1.

Example

Suppose an F8 system is configured with eight expanded output latches, as shown in Figure 1 and it is necessary to transfer the contents of scratchpad register one to latch six and scratchpad register two to latch three. The following sequence of instructions may be used.

```
LR A,1             Load R1 into the Accumulator
OUTPUT B'1110'
LR A,2             Load R2 into the Accumulator
OUTPUT B'1011'
```

Some cases may exist where the number of instructions per output may be reduced. For instance, in cases where all of the latches are to be loaded with identical data, only one output instruction to port number one is required. In another situation, where only one latch will experience data changes, that latch can be selected once, and the data in the latch may change by simply writing into I/O Port one. If all of the latches are to be cleared, the Master Reset (MR) line of these registers may be connected to a pin of port zero, the selector port. Thus, a pulse on this line will reset the latches.

Other Configurations

When a different number of latches are used in this type of output expanding technique, some system details may change. If, for instance, only seven latches or less are required, the need for a decoder is not necessary. The selector I/O port can use seven lines for latch selection and the eighth line for the enable strobe. If more than eight latches are employed, a larger decoder may be required. In addition, an extra instruction byte is required. LIS is a one byte instruction. However, if more than eight latches are used, the selection code will be at least four bits; add another bit for the enable bit and a five bit select word is require The LIS instruction only has a four bit operand. Thus, the Load Immediate instruction (LI), a two byte instruction, will be required.

Expanding F8 Inputs

An input into the F8 system may be sampled from any F8 Port. Applications exist where it may become necessary to increase the number of input sources into an F8 system. This may be done using multiplexing techniques. Once again, two F8 I/O Ports will be required. One port, CPU Port 0, will be used to select one of many input sources. The other port, CPU Port 1, will pass the data from the source to the accumulator of the CPU.

During this discussion, a system expansion to sixteen input sources will be examined. Only one type of TTL MSI circuit is required. This is a 16-input Multiplexer, the 93150.

The implementation of input expansion is rather straightforward. A diagram of the circuit is shown in Figure 2. The two CPU Ports are used. The output of port zero is connected through buffers to the input of the sixteen multiplexer decoder inputs. A buffer is required because the F8 I/O Port can only drive one standard TTL load. (Designs employing another technology, Low Power TTL or CMOS, may eliminate the need for a buffer, at the expense of other design tradeoffs). The output of the F8 I/O Ports are inverted. Thus, the external buffer will invert the output to its original state. Once the data source has been selected, an 8-bit byte is available on CPU I/O Port 1.

The sequence of F8 instructions for reading one of the sources of data from the configuration in Figure 2 is:

```
LIS   a   Select a Data Source
OUTS  0
INS   1   Read the Data
```

The four bit operand of the LIS instruction selects one of 16 data sources. Once this code has been entered into the accumulator, an output instruction transfers it to CPU Port 0. Thus, the select code will be issued to the multiplexers. The proper data is now available on Port 1 of the CPU. An input instruction, INS 1, will transfer this data from I/O Port 1 to the accumulator. In most cases, each input to the accumulator from one of the expanded input sources requires the above three instructions. These instructions are each one byte long. The OUTS and INS instructions execute in two cycles while the LIS instruction requires only one cycle. Thus, each input from configuration of Figure 2 is three bytes long and executes in five machine cycles.

Input Macro

A macro may be written to perform an input operation into the F8. The argument, SEL, selects the source of data.

```
MACRO
INPUT &SEL
LIS &SEL       Put Select Code into Accumulator
OUTS 0         Output Select Code to Port 0
INS 1          Read Data into Accumulator
MEND
```

This macro will read data from the desired source and leave it in the accumulator. The original content of the accumulator are lost.

Example

Suppose it is necessary to read the data from sources two and nine in Figure 2 and store the data in scratchpad registers one and two respectively. The program to perform this operation would be:

```
INPUT B'0010'
LR 1,A           Store Data in Reg 1
INPUT B'1001'
LR 2,A           Store Data in Reg 2
```

First, a select code for data source two must be put in the accumulator and then transferred to I/O Port 0. Then the data may be read into the accumulator and passed on to register one of the scratchpad. This sequence of operations is repeated for the second byte of data.

If data is continuously read from the same source, it will not be necessary to update the select code. This will save two instructions per input.

Other Configurations

As a larger number of data sources are required, the amount of multiplexing will increase proportionally. The multiplexers may be stacked to allow a total of 256 sources of data using two I/O Ports.

Expanding Inputs and Outputs Simultaneously

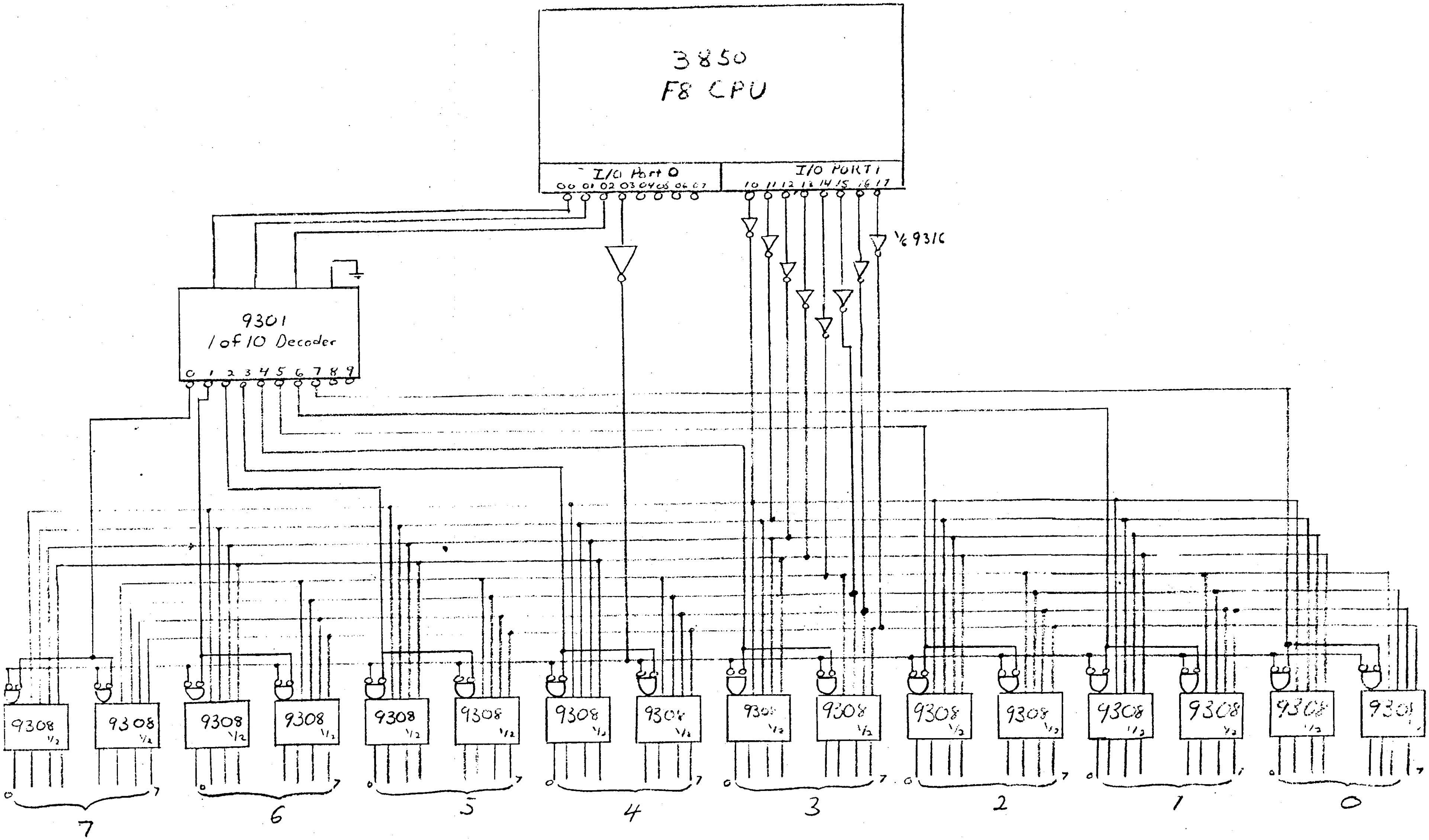
The concept of expanding the inputs and the outputs may be combined. Still using just two F8 I/O ports, a system may be configured so that the number of output latches and input sources is increased. One I/O Port will be used as a selector. In addition to choosing the source or destination of the data, the selector port must also specify the direction of data flow. The other F8 I/O Port will be used to transfer the data.

In order to combine both the input and output simultaneously, it is necessary that the input lines do not affect the logic levels of the I/O Port during the output mode. Therefore, buffering is necessary. A three-state CMOS Hex non-inverting buffer has been selected for this task.

The full expanded system is shown in Figure 3. Bit 7 of CPU I/O Port 0 controls the direction of data transfer. When it is HIGH (LOGIC ONE), data may be transferred to the output latches. When this bit is zero, the input data from the multiplexers may be sampled.

Conclusion

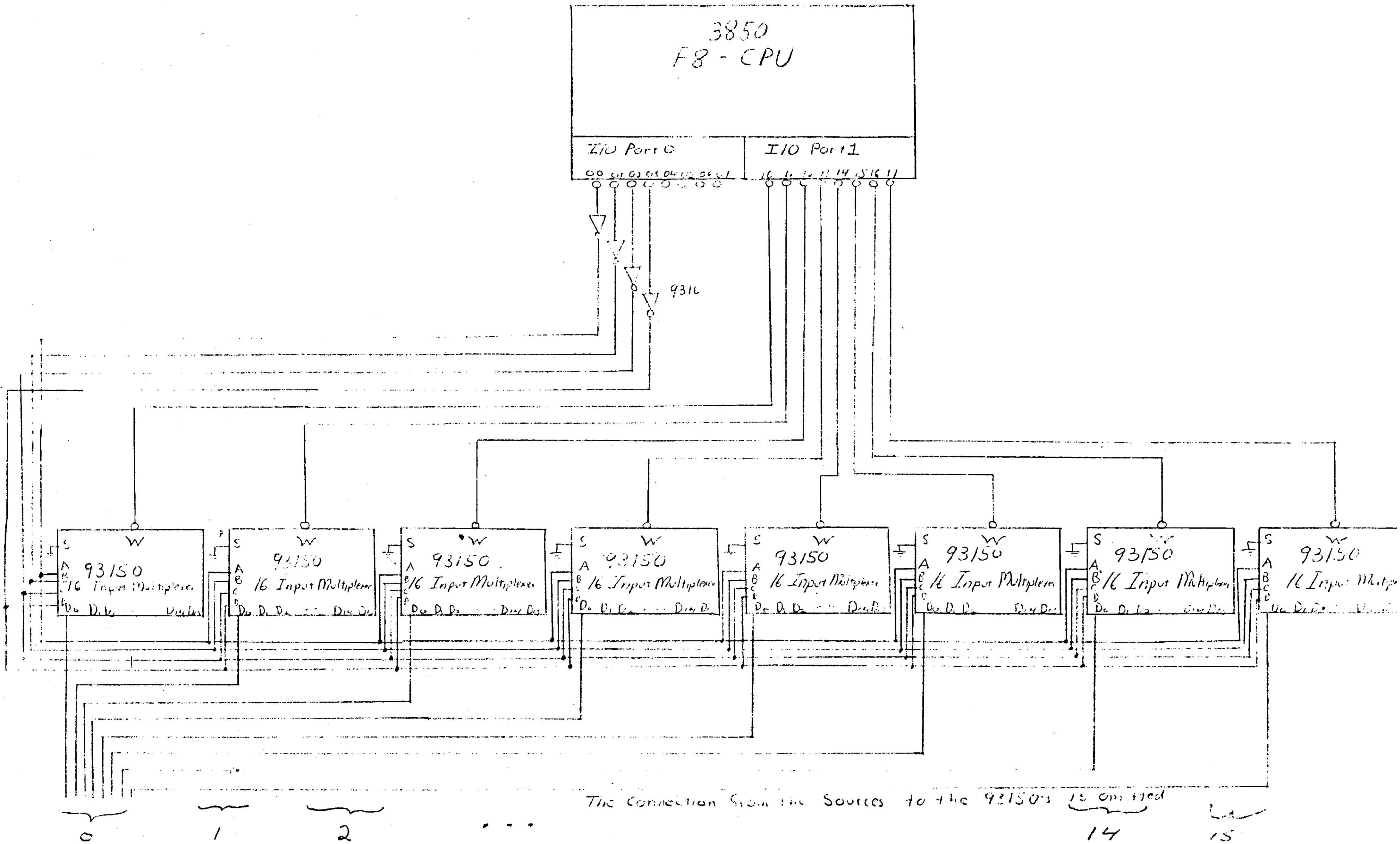
The expansion of F8 I/O ports with external MSI circuits may be accomplished using multiplexing techniques. Although this expansion requires additional programming and additional circuits, it is indeed simple. Timing considerations are kept to a minimum, in addition, the users do not have to worry about other aspects of the F8 system. The external circuits are completely independent to the microprocessor.



Expanded Output Of F8 Latches

Fig. 1

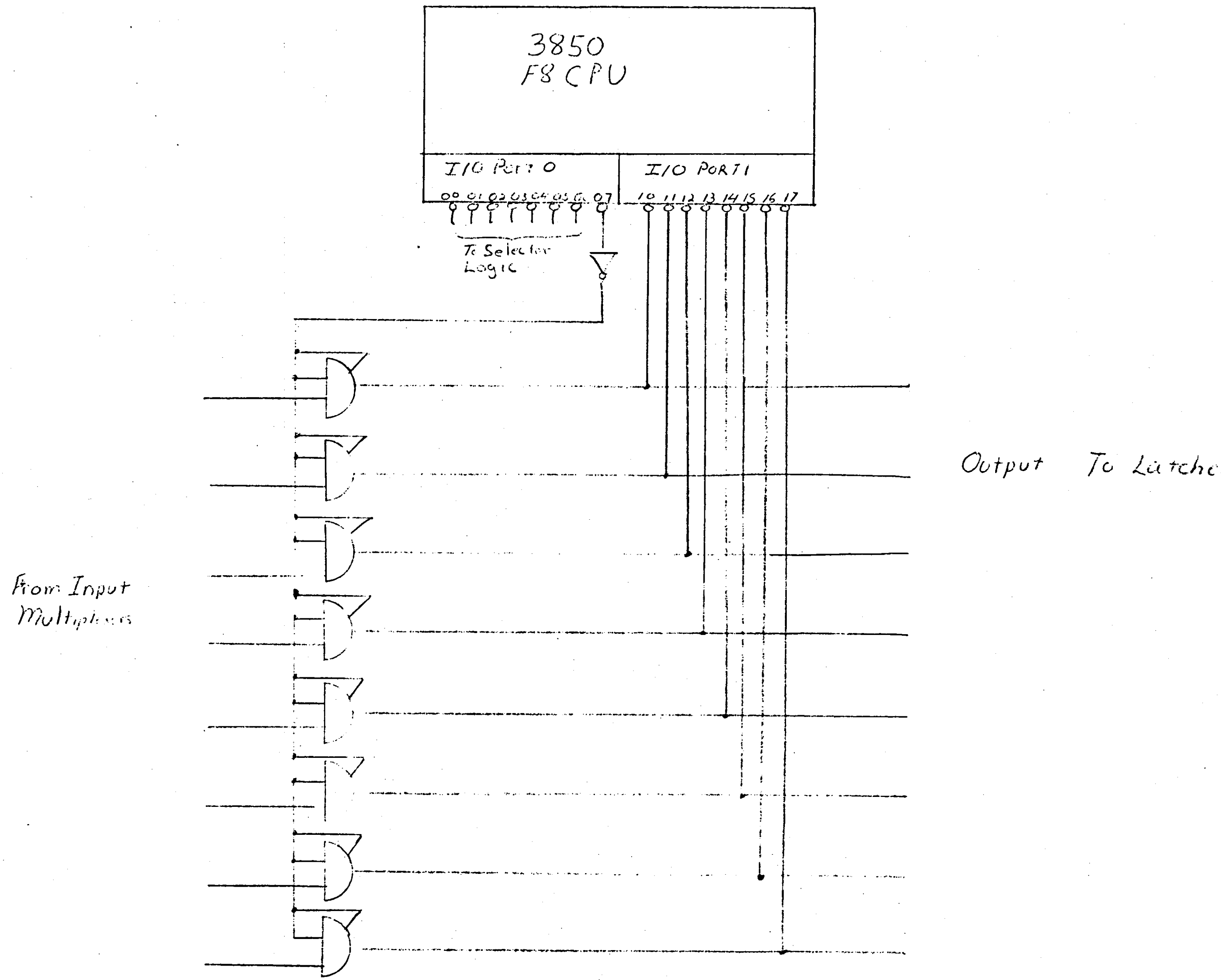
Part No.	Quantity
9308	8
9301	1
9316	2



Expanded Input To F8 I/O Ports

Fig 2

Part Number	Quan
93150	8
9316	1



Similutaneous I/O Expansion
Fig 3

EXTENDING I/O PORTS FROM THE DATA BUS

Hardware, external to the data bus, may be added to an F8 extending the I/O capability of the microprocessor system. Design of this external logic requires an understanding of the input and output instructions. The advantage of this technique is an economical increase of I/O capability without additional programming.

Input and Output Instructions

There are two F8 input instructions, IN and INS, and two F8 output instructions, OUT and OUTS. Each instruction requires three machine cycles to execute. The IN and OUT instructions are each two bytes long. The first byte is an 8-bit op code and the second byte, the operand, is an 8-bit port address. The INS and OUTS instructions are only one byte long. The first four bits are used for the op code and the remianing bits form the lower half byte of the port address. (The four high order bits of the port address for these instructions are set to be zero). Thus, the IN and OUT instructions may select one of 256 port address while the INS and OUTS only selects one of 16 ports.

All four I/O instructions require three machine cycles for execution. During the first cycle, the port address is available on the data bus. The second cycle is used to pass data to or from the accumulator while the third and final cycle is simply the next instruction fetch. The CPU dictates the state of the peripheral F8 circuits (F8 ROM, F8 MI, and F8 DMA) via five control lines. During an I/O transfer, the second cycle, these five lines will be at the following levels;

Control Line Levels

Control Line	OUTS or OUT	INS or IN
ROMC4	1	1
ROMC3	1	1
ROMC2	0	0
ROMC1	1	1
ROMC0	0	1

For an output instruction the data will be passed from the accumulator to the I/O port, while an input instruction transfers data to the accumulator from the port.

I/O Logic

A block diagram of the logic necessary for additional I/O ports is shown below. This circuit is connected to the F8 data bus.

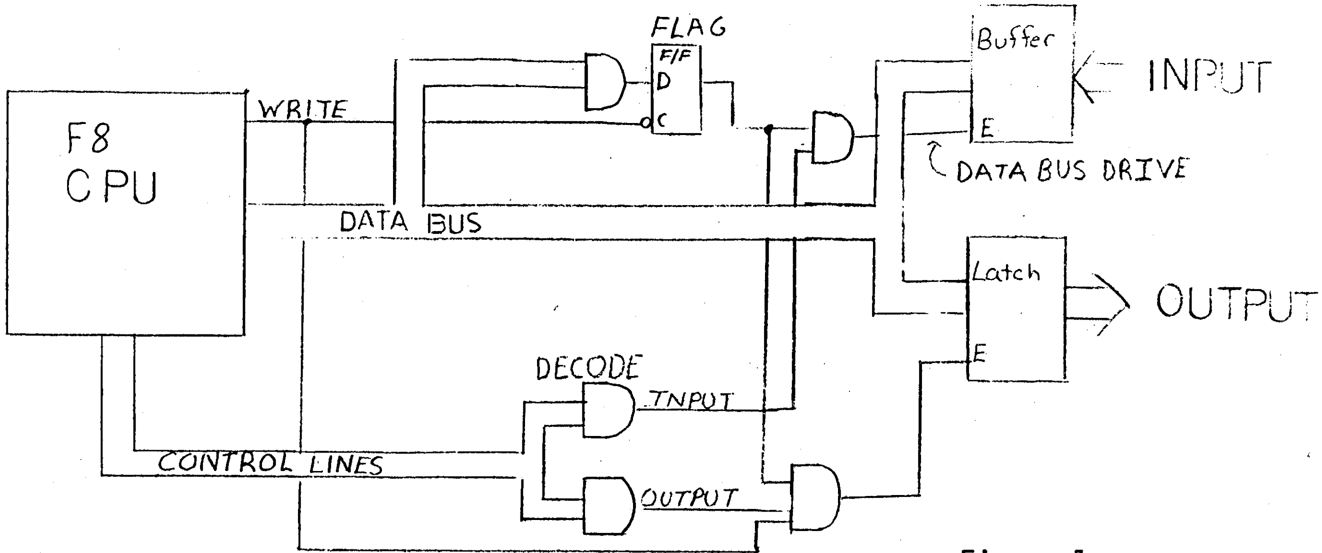
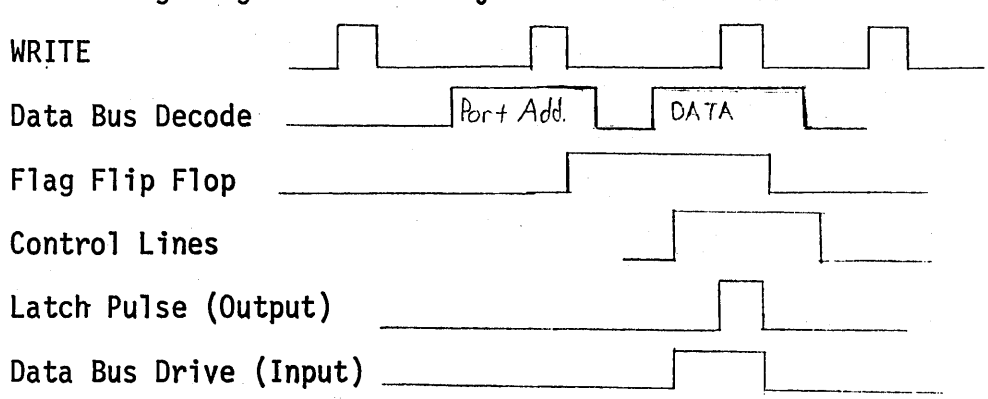


Figure 1

Each new port in the system must be assigned a port number for the instructions to reference. This circuit must decode the data bus for the specific port address. If a match is found, it will flag the match in a flip flop. Next, the CPU control lines are decoded to the Input and Output states. If an input instruction is found, the transfer buffer connected to the data bus is opened. If an output instruction is detected, the external latch is enabled to retain the content of the data bus. If neither instruction is detected,no action is performed. The WRITE pulse, generated by the CPU on the control lines, performs the system timing. This pulse identifies the end of machine cycles as well as time slots when valid data is available on the Data Bus and control lines.

The Timing diagram for this system is shown below.



The latch pulse is used to open the external latches while the data bus drive line allows the input source data to be put onto the data bus.

A very important consideration in this system must be given to the load that the external logic places on the CPU generated signals. Within an F8 system, the data bus lines, the CPU control lines, and the timing signals are primarily intended as intrasystem; i.e. they originate from an F8-MOS circuit and drive one or more F8-MOS circuits. As such, they have very little drive capability. For this reason, it is recommended that CMOS circuits be used to interface with the F8 signals and the external logic. In addition, a CMOS three state circuit is required for driving the data bus.

Example

As an illustration of this technique, Figure 2 is an example of an F8 system with an expansion to eight additional I/O ports. They are assigned port addresses H'F8' through H'FF'. Note the use of CMOS circuits buffering the CPU signals.

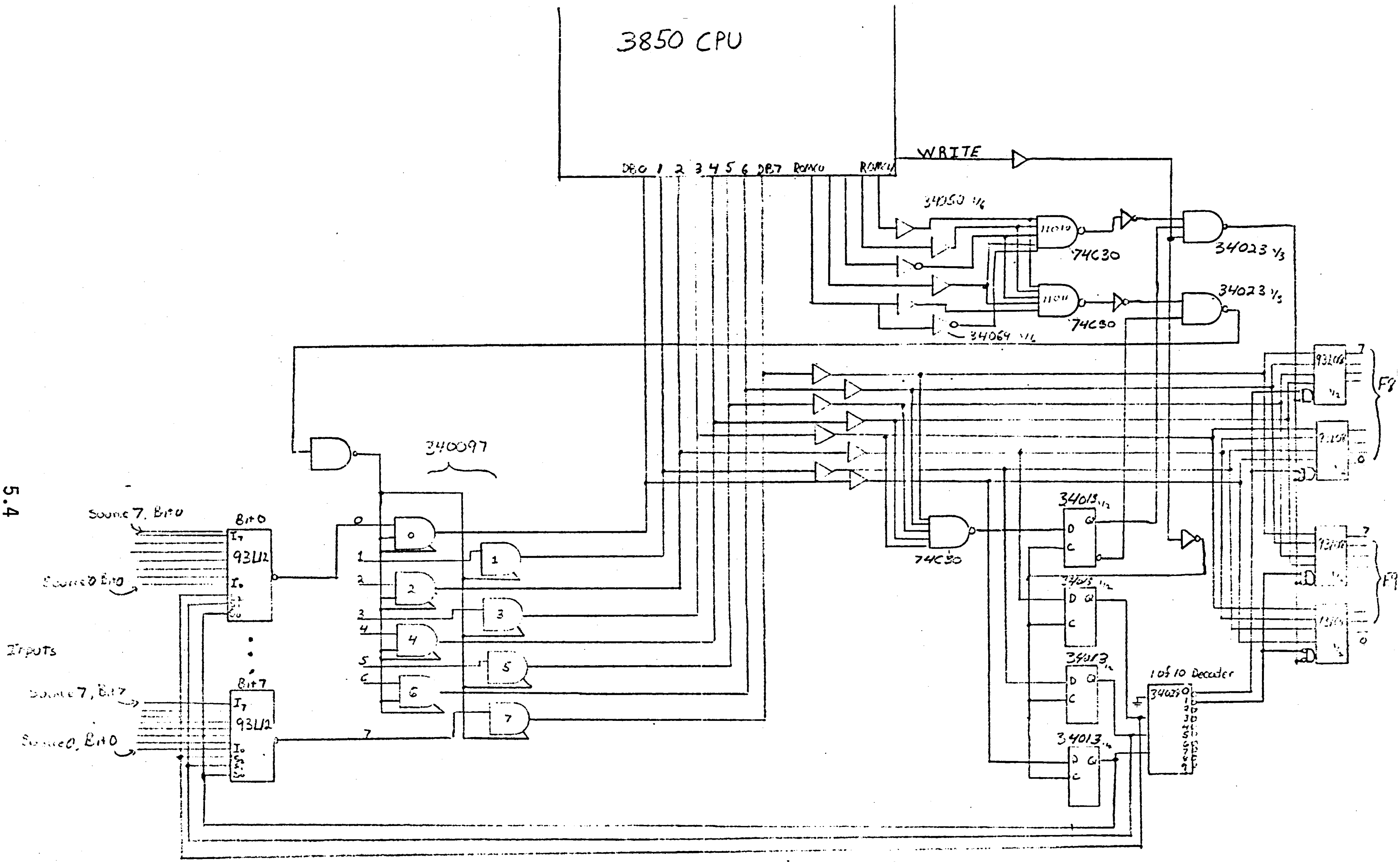
Since eight additional ports are required, it is necessary not only to flag an I/O port address match from the data bus, but to retain the selected port number. This is implemented using four flip flops; one retains an address match while the other three select the port.

The eight input sources to this system are fed into eight multiplexers. The select lines of these multiplexers are obtained from the three select flip flops. The outputs of these devices are connected to CMOS three state non-inverting gates. The drive control signal for these gates originates from the decoding logic.

The eight output latches receive their input from the data bus through CMOS buffers. The enable lines are controlled by the decoding logic and the WRITE pulse. To select one of eight latches, the outputs of the three select flip flops are fed into a decoder. Eight outputs of this decoder are directly connected to the enable line of the latch.

Conclusion

It is possible to add external circuits to an F8 system to increase the number of available I/O ports. The example given requires 29 additional circuits to increase the number of ports by eight. Twelve of these circuits perform a control function while nine are required for eight output latches and eight are necessary for input sources. The advantage of this approach is that it requires no additional programming steps.



Expanding I/O From F8 Data Bus

Figure 2

EXPANDING INPUT SOURCES THROUGH MEMORY

Abstract

Data may be entered into an F8 system using memory locations. A minimum number of external components are required to decode the address lines, select an input source, and drive the data bus. This approach increases the number of instructions that may be used to transfer data into the accumulator while, simultaneously, drastically reducing the programming required in certain applications.

Description

The accompanying figure is an F8 system with the number of input sources expanded to 16. The 16 8-bit inputs are fed into eight multiplexers. Address lines, originating from the Memory Interface circuit, are decoded to memory locations FFF0 through FFFF. (Of course, these lines may be used for other parts of the system not shown here). The lower four bits of the address are fed into the select lines of the multiplexers; thus, they may be used to select one of sixteen input sources. The decoding of the address lines enables the three state gates to drive the F8 data bus.

This method of input expansion offers a major advantage to the user. Instead of only having one instruction, IN or INS, to transfer data into the accumulator, a much larger set of instructions, the Memory Reference Instructions, may be used. Thus, the input data may be brought in and an arithmetic operation may be performed using only one instruction. For example, the XM (Exclusive OR with Memory) instruction may be used. The following operation will take place.

$$\text{ACC} \leftarrow (\text{ACC}) \oplus \text{INPUT DATA}$$

This operation would have required two instructions if, instead, an I/O port was used.

Example

Suppose a system has 128 bits to poll. Each bit may be the service flag bit of a peripheral in the system. A logic zero in the flag means that the unit does not need attention while a one is a request for action. A routine may be written to accomplish this.

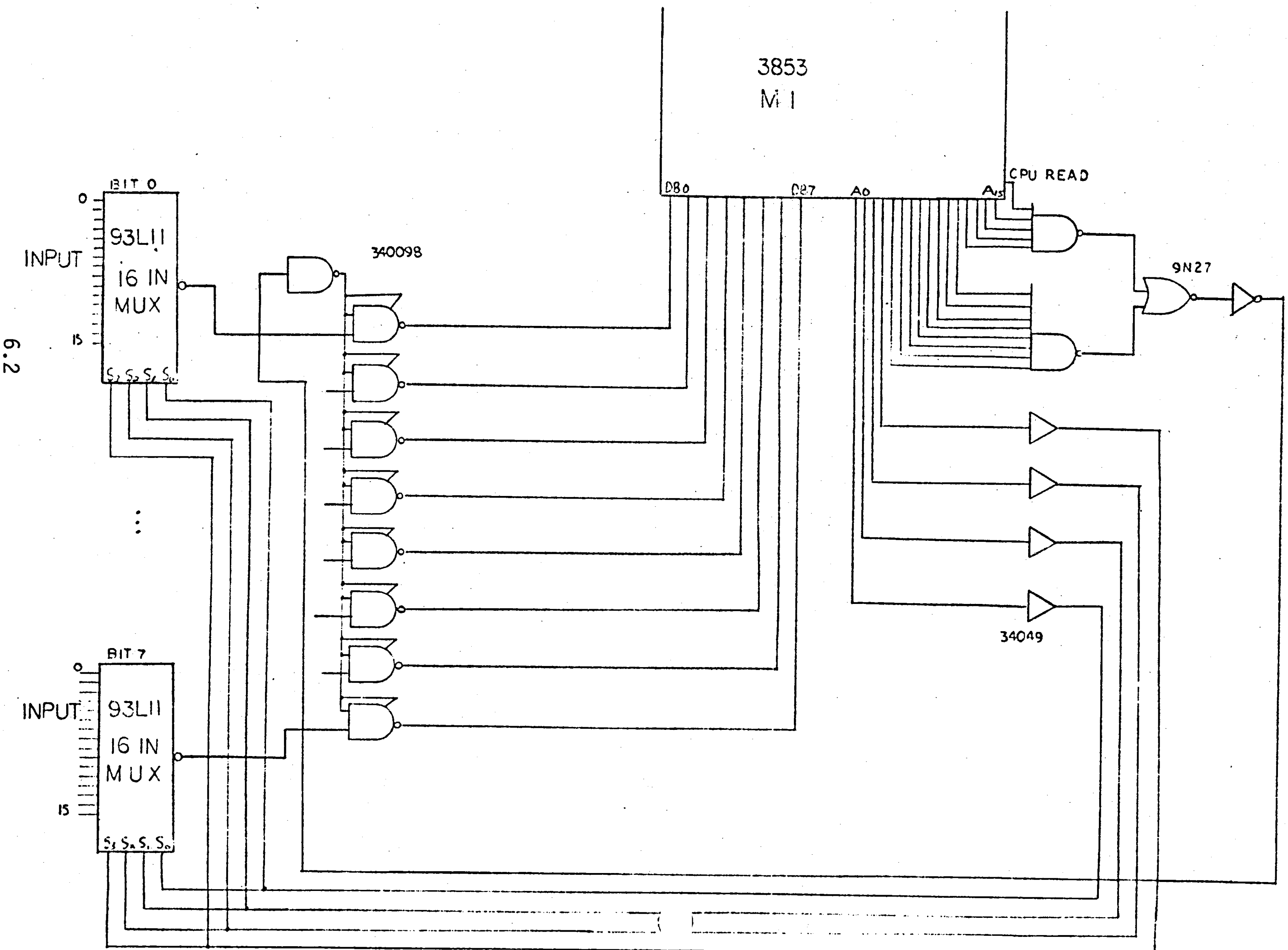
Label	Operation	Operand	Comment
	DCI	H'FFF0'	Load the data counter with the address of the first input source
	LIS	H'F'	Put 1111 in the accumulator
	LR	1, A	Store 1111 in a counter
	LIS	0	Clear the accumulator
TOP	CM		Compare Input Source to zero
	BNZ	EXIT	Branch if Input is not zero
	DS	1	Decrement the counter
	BNZ	TOP	Branch if counter is not zero

This routine will first load the data counter with the input address. This is a 16-bit address because the memory space is being used for the data. Scratchpad register one is the loop counter. Next, the program clears the accumulator and, using the CM instruction, compares the input against zero. If the result is non-zero, a jump to EXIT will occur. For a zero result the counter will be decremented. The counter serves two purposes. First, it signals the end of the loop, when all sources have been polled. The counter also identifies which input byte was non-zero.

Conclusion

Placing input sources in memory provides a simple scheme for performing certain input tasks. Polling many inputs is one of these. The memory reference instructions offer much flexibility during input operations. This concept may also be expanded to output latches. In a similar fashion, the address lines may be decoded to load a latch using a Memory Reference instruction.

6.2



PARTS	
93L11	8
340098	2
34049	1
9N27	1

EXPANDING INPUT SOURCES THRU MEMORY

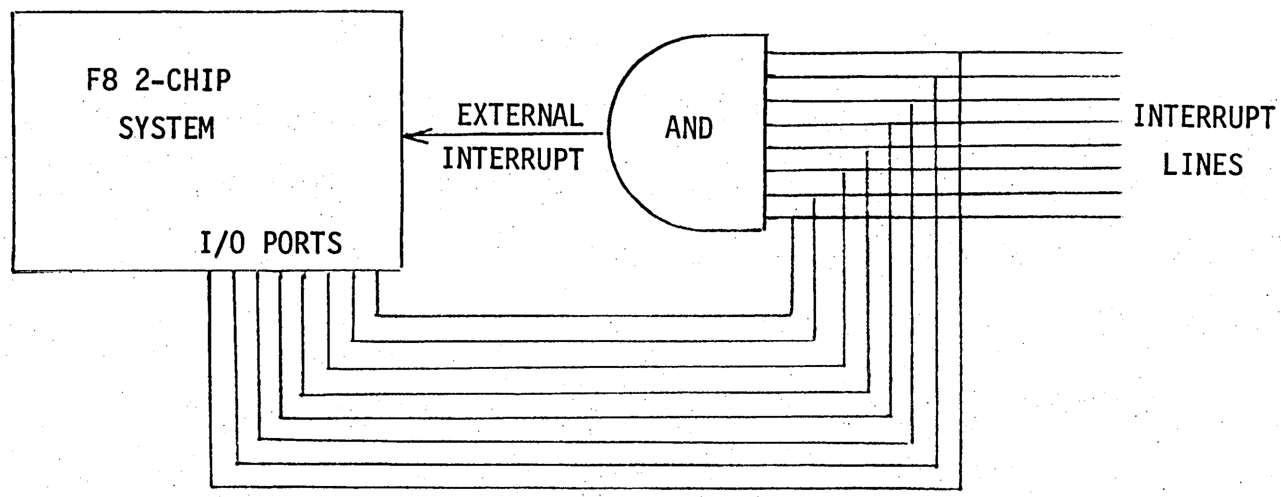
EXTENSION OF EXTERNAL PRIORITY INTERRUPT LEVELS IN A F8 2 CHIP CPU-ROM CONFIGURATION

Abstract

The F8 2 chip (CPU-ROM) system has only 1 level of external interrupt. However, it takes only 1 additional gate to extend the levels to as many as up to 32 levels. The following description, however, is confined to an extension of up to 8 levels.

Description

Fig. 1 describes such a configuration

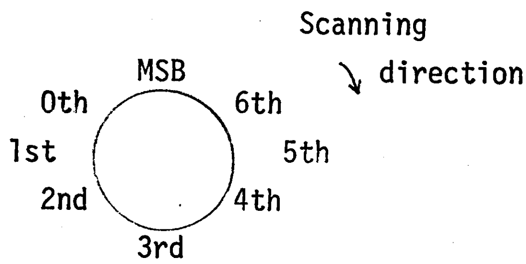


The Interrupt Line and the I/O Port Lines are all negative signals. The 'AND' gate is actually an 'OR' function of the negative signals.

Fig. 1

All the interrupt lines are 'AND'ed together to give a common interrupt signal to the ROM chip. However, each interrupt line is also tied to a line of an I/O port according to a priority basis. (e.g. the highest priority on the MSB line to the least priority on the LSB line).

Once the system is interrupted by 1 of the eight lines, the system will go into an interrupt scanning routine which will scan for the active line on the I/O port from left to right, the result from that interrupt scan will be a code indicating which interrupt is active and ought to be serviced.



Scanning routine

```
Scan     LIS    8       Load a count of A
         LR     0,A     Store in Register 0
         INS    PORT    Get the interrupt information
         LR     1,A     Store in Register 1
Loop     AS     1       Add REG 1
         BC     OUT     MSB is a '1'
         LR     1,A     Put the left shifted result
         DS     0       back to R1
         BR     LOOP    Update the code
                        Do it again.

OUT . . . . . . . . . . . . . .
```

With the above scheme, a multi-level priority interrupt scheme is constructed. However, the user should also be aware of the timing of the interrupts responses since the scanning also takes time.

DOUBLE PRECISION BINARY ADD/SUBTRACT SUBROUTINE

Purpose

This subroutine will perform a 16-bit binary addition or subtraction and return to the user with the status of the 16-bit operation and the result of the operation.

Calling Sequence

Scratch Regs: 1,0 Contain AUGEND/SUBTRAHEND

3,2 Contain ADDEND/MINUEND

(bit 7 of SR1 and SR3 is the most significant bit and sign bit of the 16-bit numbers). SR7 = 0 for add, SR7 ≠ for subtract.

PI DBA (Calls subroutine to perform add or subtract

Upon Exit

The sum or difference is contained in SR3, SR2 with bit 7 of SR3 being the most significant bit and the sign bit.

The proper status is set in the W-Reg for the 16-bit operation.

Operation Performed

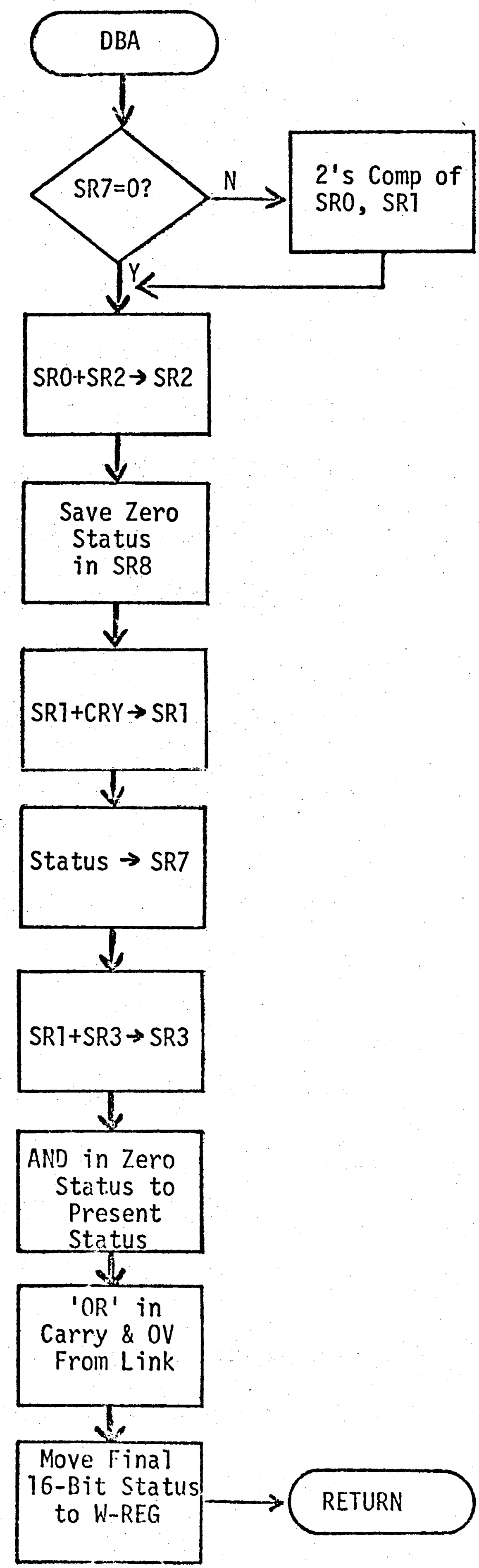
(SR3, SR2) ± (SR1, SR0) ⟶ (SR3, SR2)

Registers Used

ACC, W, J, SR0-SR3, SR7-SR8

Calls to other Subroutines

None


DBA
SR7=0?
N
2's Comp of SR0, SR1
Y
SR0+SR2→SR2
Save Zero Status in SR8
SR1+CRY→SR1
Status → SR7
SR1+SR3→SR3
AND in Zero Status to Present Status
'OR' in Carry & OV From Link
Move Final 16-Bit Status to W-REG
RETURN

```
                              * DOUBLE PRECISION BINARY ADD (16 BIT) OR
                              * SUBTRACT SUBROUTINE.
                              *   UPON.ENTRY:
                              *     IF SR7 = 0 ADD, OTHERWISE SUBTRACT.
                              *     AUGEND = SCRATCH REGS 0 AND 1(1 MSB).
                              *     ADDEND = SCRATCH REGS 2 AND 3(3 MSB).
                              *     (THESE DOUBLE REGISTERS ARE TREATED AS
                              *     16 BIT SIGNED NUMBERS WITH BIT 7 OF
                              *     REGS 1 AND 3 BEING THE SIGN BIT)
                              *     (IF SUBTRACT, THEN (R2,R3)-(R0,R1)).
                              *   UPON EXIT:
                              *     SUM OR DIFF IS CONTAINED IN R2,R3.
                              *     W-REG CONTAINS  THE STATUS OF THE 16
                              *     BIT ADD.
2007  70          126   DBA  LIS  0
2008  C7          127        AS   7
2009  84  B       128        BZ   ADD     IF SR7=0, THEN GO DO ADD.
200B  40          129        LR   A,0
200C  18          130        COM
200D  1F          131        INC
200E  50          132        LR   0,A     COMPLEMENT FOR SUBTRACT.
200F  1E          133        LR   J,W
20E0  41          134        LR   A,1
20E1  18          135        COM
20E2  1D          136        LR   W,J
20E3  19          137        LNK
20E4  51          138        LR   1,A
20E5  40          139   ADD  LR   A,0     GET LSB OF AUGEND.
20E6  C2          140        AS   2       ADD LSB OF ADDEND.
20E7  52          141        LR   2,A     SAVE PARTIAL SUM.
20E8  1E          142        LR   J,W
20E9  49          143        LR   A,9     GET STATUS AND OR IN ALL BITS
20EA  22  1B      144        OI   H'1B'   BUT ZERO.
20EC  58          145        LR   8,A     SAVE IN SR8 FOR CALC OF FINAL.
20ED  1D          146        LR   W,J     GET ORIG STATUS BACK IN W-REG.
20EE  41          147        LR   A,1     GET 2ND BYTE OF AUGEND.
```

8.2

```
LOC   OBJECT CODE   STMT   SOURCE STATEMENT
20EF  19            148         LNK          ADD 1 IF CARRY FROM PREV ADD.
20F0  1E            149         LR    J,W    SAVE STATUS, MAY BE OV OR CARRY.
20F1  C3            150         AS    3      ADD LSB OF ADDEND.
20F2  53            151         LR    3,A    SAVE IN REG 1.
20F3  49            152         LR    A,9    MOVE STATUS SAVED FROM LINK
20F4  57            153         LR    7,A    ADD TO SR 7.
20F5  1E            154         LR    J,W
20F6  49            155         LR    A,9    GET STATUS FROM FINAL ADD AND
20F7  F8            156         NS    8      'AND' IN ZERO STATUS FROM
20F8  58            157         LR    8,A    FIRST ADD.
20F9  47            158         LR    A,7
20FA  59            159         LR    9,A    MOVE STATUS SAVED FROM LINK
20FB  48            160         LR    A,8    BACK TO W-REG AND 'OR' IN
20FC  1D            161         LR    W,J    OV AND CARRY BITS TO FINAL
20FD  92   4        162         BNC   DBA1   STATUS.
20FF  22   2        163         OI    2
2101  1D            164         LR    W,J
2102  98   3        165    DBA1 BNO   DBAX
2104  22   8        166         OI    8
2106  59            167    DBAX LR    9,A    MOVE FINAL STATUS TO W-REG.
2107  1D            168         LR    W,J
2108  1C            169         POP          RETURN.
```

8.3

BINARY MULTIPLY SUBROUTINE

Purpose

This subroutine performs a binary multiplication of two 8-bit numbers resulting in a 16-bit product. The multiplication is accomplished by testing bit positions of the multiplier and adding the corresponding value to a partial product (when a bit is set) until all of the bits have been tested.

Calling Sequence

SR1 Contains the multiplier

SR2 Contains the multiplicand

PI BMPY (Calls subroutine to perform the multiplication)

Upon Exit

The product is contained in SR7, SR6, with bit 7 of SR7 being the most significant bit of the 16-bit product.

Operation Performed

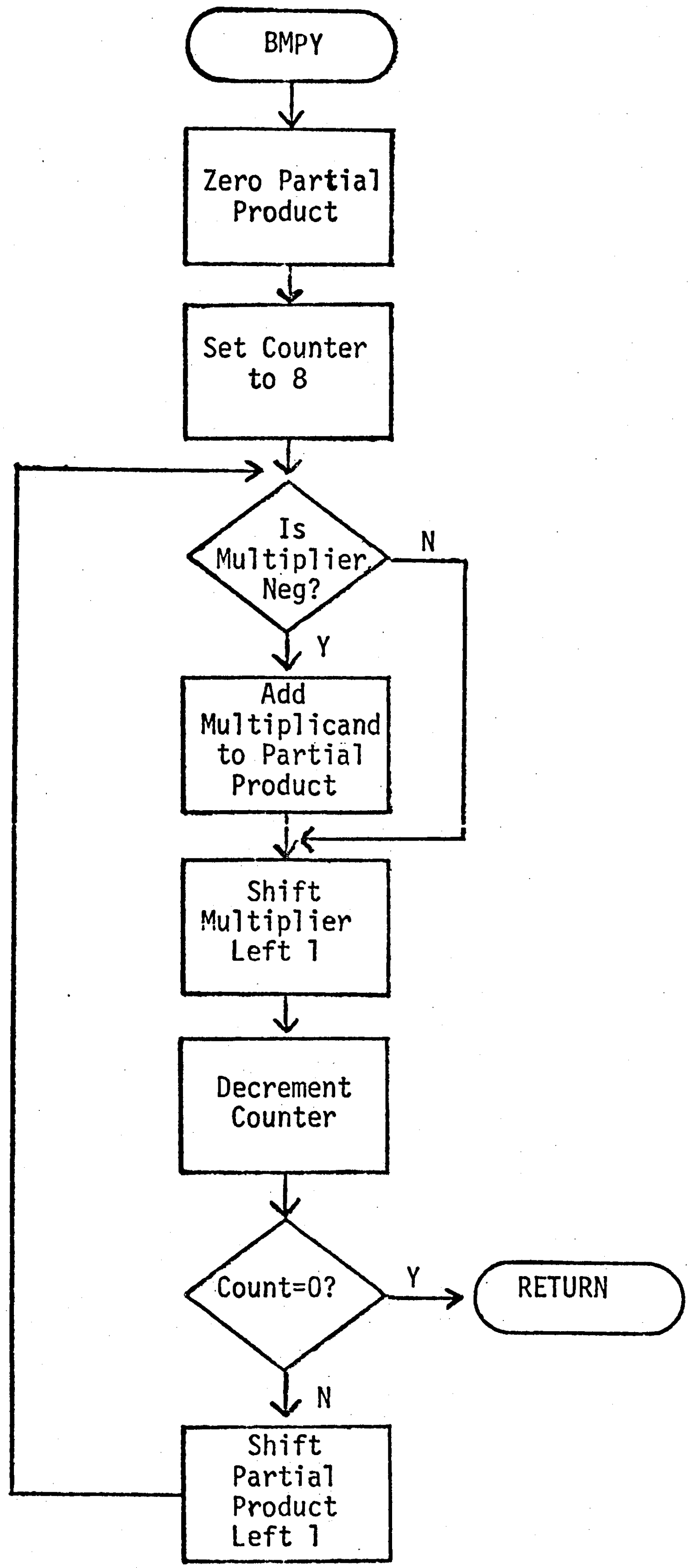
SR1 X SR2 $\longrightarrow$ (SR7, SR6)

Registers Used

ACC, J, W, SR1 - SR2, SR5 - SR7

Calls to Other Subroutines

None


BMPY
Zero Partial Product
Set Counter to 8
Is Multiplier Neg?
N
Y
Add Multiplicand to Partial Product
Shift Multiplier Left 1
Decrement Counter
Count=0?
Y
RETURN
N
Shift Partial Product Left 1

```
 LOC  OBJECT CODE   STMT   SOURCE STATEMENT                                          PAGE  7

                           * BINARY MULTIPLY SUBROUTINE
                           *  SCRATCH REG 1 CONTAINS MULTIPLIER.
                           *  SCRATCH REG 2 CONTAINS MULTIPLICAND.
                           *  SCRATCH REGS 6 AND 7 CONTAIN PRODUCT (SR7-MSB).
 2125  78           196    BMPY LIS  8        INITIALIZE COUNTER TO 8.
 2126  55           197         LR   5,A      INITIALIZE PARTIAL PRODUCT.
 2127  70           198         LIS  0
 2128  56           199         LR   6,A
 2129  57           200         LR   7,A
 212A  70           201    BMP1 LIS  0
 212B  C1           202         AS   1        IS SIGN BIT OF MULTIPLIER SET?
 212C  81  7        203         BP   BMP2     NO, GO SHIFT PARTIAL PRODUCT.
 212E  42           204         LR   A,2
 212F  C6           205         AS   6        YES, ADD MULTIPLICAND TO
 2130  56           206         LR   6,A      PARTIAL PRODUCT.
 2131  47           207         LR   A,7
 2132  19           208         LNK
 2133  57           209         LR   7,A
 2134  41           210    BMP2 LR   A,1
 2135  13           211         SL   1        SHIFT THE MULTIPLIER LEFT 1.
 2136  51           212         LR   1,A
 2137  35           213         DS   5        DECREMENT COUNTER
 2138  84  C        214         BZ   BMP3     EXIT IF DONE.
 213A  46           215         LR   A,6      SHIFT PARTIAL PRODUCT LEFT 1.
 213B  C6           216         AS   6        SHIFT LSB LEFT 1
 213C  56           217         LR   6,A
 213D  1E           218         LR   J,W      SAVE STATUS FOR CARRY.
 213E  47           219         LR   A,7
 213F  13           220         SL   1        SHIFT MSB LEFT 1 AND ADD IN
 2140  1D           221         LR   W,J      CARRY FROM PREV SHIFT (LSB)
 2141  19           222         LNK
 2142  57           223         LR   7,A
 2143  90  E6       224         BR   BMP1     GO BACK FOR NEXT.
 2145  1C           225    BMP3 POP           RETURN TO CALLER.
                    226         END
```

9.2

DOUBLE PRECISION BINARY DIVIDE SUBROUTINE

Purpose

This subroutine performs the binary division of a 16-bit unsigned value by another 16-bit unsigned value, and returns with a 16-bit unsigned quotient. The division is performed by repetitively subtracting the divisor from the dividend until a "Carry" is not detected.

Calling Sequence

SR1,-SR0 Contain the divisor (bit 7 of SR1 is most significant bit)

SR3, SR2 Contain the dividend (bit 7 of SR3 is most significant bit)

PI BVD (Calls subroutine to perform the division)

Upon Exit

The quotient is contained in SR6, SR5, with bit 7 of SR6 being the most significant bit of the 16-bit quotient.

Operation Performed

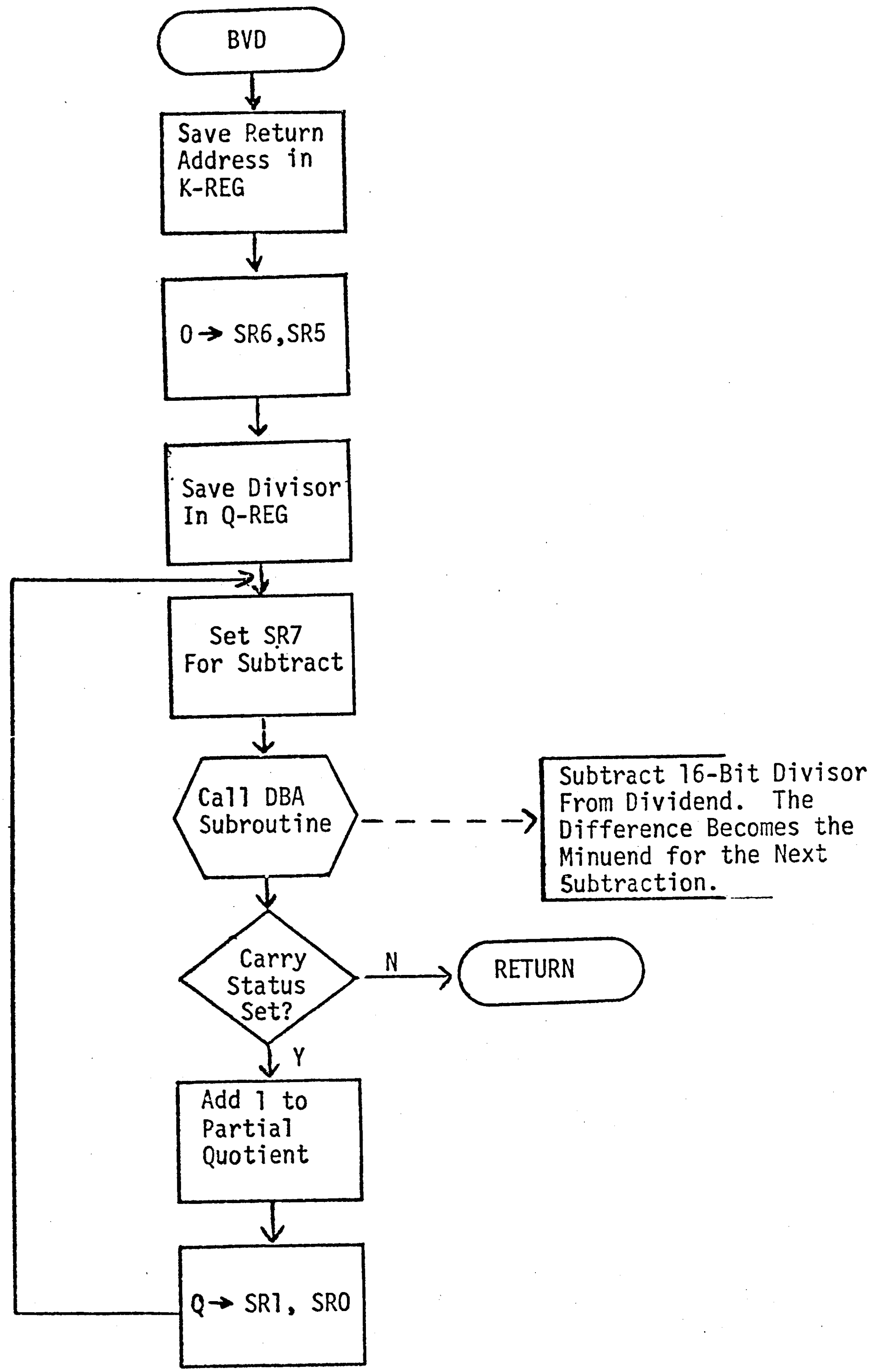
$(SR3, SR2) \div (SR1, SR0) \longrightarrow (SR6, SR5)$

Registers Used

ACC, K, W, SR0 - SR3, SR5 - SR7

Calls to Other Subroutines

DBA is called to perform the double precision subtraction.


BVD
Save Return Address in K-REG
0 → SR6,SR5
Save Divisor In Q-REG
Set SR7 For Subtract
Call DBA Subroutine
Subtract 16-Bit Divisor From Dividend. The Difference Becomes the Minuend for the Next Subtraction.
Carry Status Set?
N
RETURN
Y
Add 1 to Partial Quotient
Q → SR1, SR0

```
 LOC  OBJECT CODE      STMT   SOURCE STATEMENT

                              * DOUBLE PRECISION (16-BIT) BINARY DIVIDE
                              * SUBROUTINE.
                              *   UPON ENTRY:
                              *     SR2,SR3 CONTAIN DIVIDEND (SR3-MSB).
                              *     SR0,SR1 CONTAIN DIVISOR (SR1-MSB).
                              *   CALLS TO OTHER SUBROUTINES:
                              *     PI   DBA   (FOR DOUB. PREC. SUB).
                              *   UPON EXIT:
                              *     SR5,SR6 CONTAIN QUOTIENT (SR6-MSB).
 2109   8                171  BVD  LR    K,P      SAVE RETURN ADDR. IN K.
 210A   70               172       LIS   0
 210B   55               173       LR    5,A      ZERO QUOTIENT AREA.
 210C   56               174       LR    6,A
 210D   40               175       LR    A,0
 210E   6                176       LR    QU,A     SAVE DIVISOR IN Q-REG.
 210F   41               177       LR    A,1
 2110   7                178       LR    QL,A
 2111   71               179  BVD1 LIS   1        SUBTRACT DIVISOR FROM DIVIDEND.
 2112   57               180       LR    7,A      GO TO 'DBA' ROUTINE WITH SR7 NOT
 2113   28  20  D7       181       PI    DBA      EQU 0 FOR DOUB. PREC SUBTRACT.
 2116   92  D            182       BNC   BVD2
                              * QUOTIENT IS COMPLETE WHEN SUBTRACT RETURNS
                              * WITH 'NO CARRY' STATUS.
 2118   45               183       LR    A,5
 2119   1F               184       INC
 211A   55               185       LR    5,A      INCREMENT QUOTIENT.
 211B   46               186       LR    A,6
 211C   19               187       LNK
 211D   56               188       LR    6,A
 211E   2                189       LR    A,QU
 211F   50               190       LR    0,A      RESTORE DIVISOR.
 2120   3                191       LR    A,QL
 2121   51               192       LR    1,A
 2122   90  EE           193       BR    BVD1
 2124   C                194  BVD2 PK             RETURN TO CALLER.
```

10.2

DECIMAL ADD/SUBTRACT SUBROUTINE

Abstract

This subroutine will perform a Floating Decimal ADD or SUBTRACT on two 14 Digit Signed Numbers.

Description

The two numbers are stored using one scratchpad location for every 2 BCD digits The sign uses one half byte, scratchpad location. The exponent of the number uses one scratchpad location. The following scratchpad locations are used in the program example:

K REGISTER

Digit No.	Sign	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Scratch Pad location (Octal)	R27		R26		R25		R24		R23		R22		R21		R20	

R27 Upper, Sign of Number "0" Positive
"F" Negative

R27 Lower, overflow digit

R20 to R26 14 BCD digits, R26 upper - Most significant, R20 lower-Least Significant

The exponent of the number is stored in binary scratchpad location R60 (octal)

Binary "0" represent 10^{-1}
Binary "1" represent 10^{0}
Binary "2" represent 10^{1}
Binary "3" represent 10^{2} etc...

A REGISTER

Digit No	Sign	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Scratch Pad location (Octal)	R37		R36		R35		R34		R33		R32		R31		R30	

R37 Upper, Sign of Number

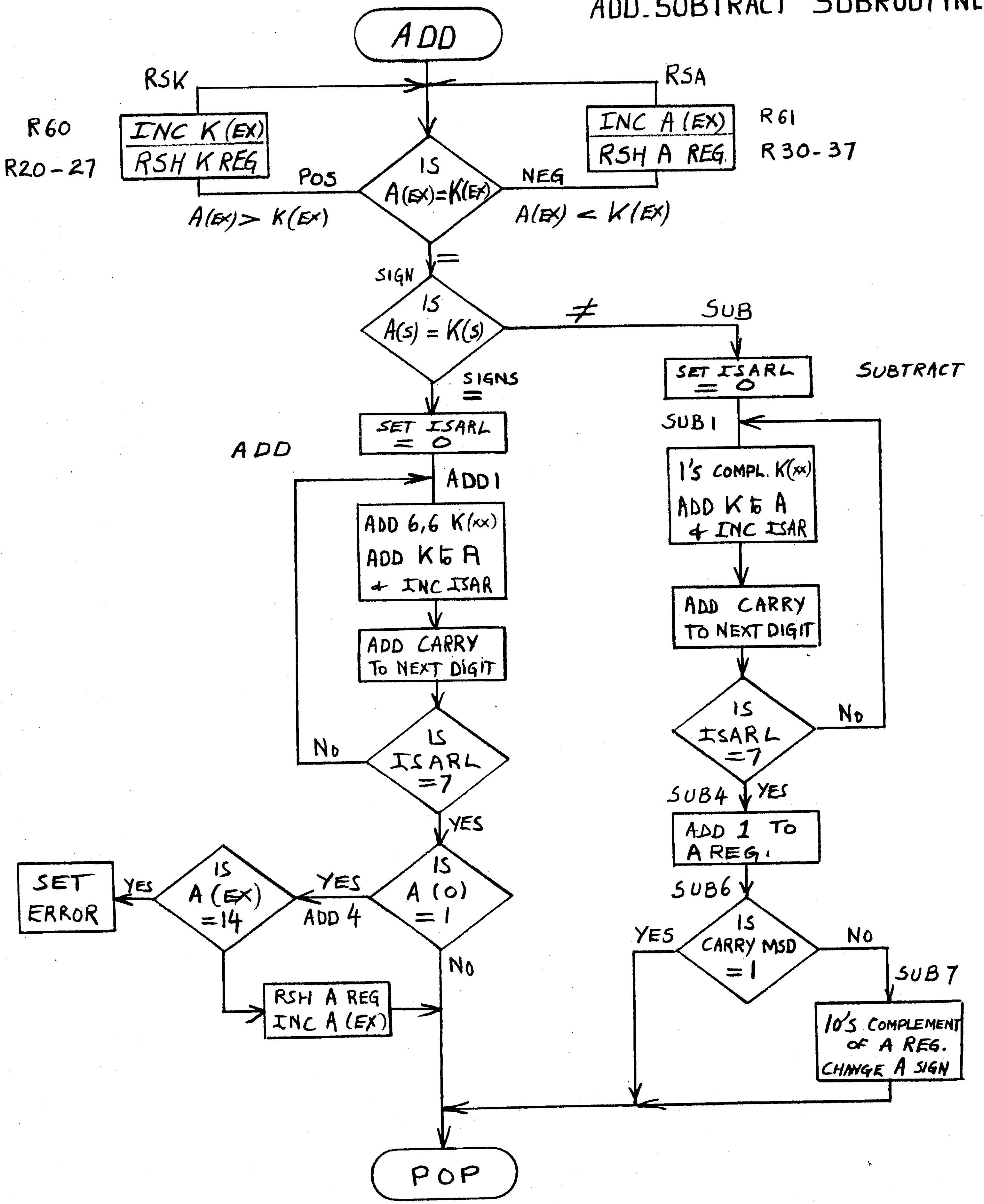
R37 Lower, overflow digit

R30 to R36 14 BCD Digits

The exponent of the number is stored in binary in scratch pad location R61

The number in the K Register is Added to or Subtracted from the number in the A Register. The result is retained in the A Register.

ADD.SUBTRACT SUBROUTINE



```
OBJECT CODE          SOURCE STATEMENT

                     ******* ADD , SUBTACT SUBROUTINE *******
66                   ADD     LISU    6        SET ISAR
68                           LISL    0            = 60
4D                           LR      A,I      LOAD K EXP.(R60),INC ISAR
18                           COM              GET 2'S
1F                           INC                 COMPLEMENT
CC                           AS      S        SUB A(EX)-K(EX)
84  3B                       BZ      SIGN     EQUAL BRANCH
91  1D                       BM      RSAS     NEGATIVE BRANCH
8                            LR      K,P      SAVE PC1
28   2  53                   PI      RSK      GO TO RIGHT SHIFT K(REG)
9                            LR      P,K      RETURN PC1
66                           LISU    6        SET ISAR
68                           LISL    0           =60
4C                           LR      A,S      LOAD K(EX)
1F                           INC              INC.K(EX)
5C                           LR      S,A      STORE K(EX)IN R60
90  EB                       BR      ADD
62                   RSK     LISU    2        SET ISAR
68                           LISL    0           =20
4C                   RSK1    LR      A,S      LOAD FROM R(XX)
14                           SR      4        SHIFT DIGIT RIGHT
5D                           LR      I,A      STORE IN R(XX),INC ISAR
4E                           LR      A,D      LOAD FROM R(XX)+1,DEC ISAR
15                           SL      4        SHIFT DIGIT LEFT
EC                           XS      S        COMBINE TWO DIGITS
5D                           LR      I,A      STORE IN R(XX),INC ISAR
8F  F8                       BR7     RSK1
20  F0                       LI      H'F0'    CLEAR
FC                           NS      S           K(0)
5C                           LR      S,A            R27 LOWER
1C                           POP
8                    RSAS    LR      K,P      SAVE PC1
28   2  6F                   PI      RSA      GO TO RIGHT SHIFT A(REG)
9                            LR      P,K      RETURN PC1
66                           LISU    6        SET ISAR
69                           LISL    1           =61
4C                           LR      A,S      LOAD AND
1F                           INC                 INC A(EX)
5C                           LR      S,A      STORE IN R61
90  CF                       BR      ADD
63                   RSA     LISU    3        SET ISAR
68                           LISL    0           =30
4C                   RSA1    LR      A,S      RIGHT
14                           SR      4           SHIFT
5D                           LR      I,A          A REG.
4E                           LR      A,D            R30,36
15                           SL      4                 ONE
EC                           XS      S                  DIGIT
5D                           LR      I,A
8F  F8                       BR7     RSA1
20  F0                       LI      H'F0'    CLEAR
FC                           NS      S           A(0)
5C                           LR      S,A            R37 LOWER
1C                           POP
63                   SIGN    LISU    3        SET ISAR
6F                           LISL    7           =37
20  F0                       LI      H'F0'    ISOLATE UPPER..
FC                           NS      S        ..DIGIT,A(SIGN) R37
62                           LISU    2              AND
```

```
OBJECT CODE                 SOURCE STATEMENT

 EC                                XS      S         COMPARE WITH K(SIGN) R27
 94  23                            BNZ     SUB       SIGNS DIFFERENT BRANCH
 68                                LISL    0         SIGNS EQUAL ADD
 62                         ADD1   LISU    2
 20  66                            LI      H'66'     BEGIN DECIMAL ADD
 CC                                AS      S         ADD 6,6 TO K DIGIT,R(XX)
 63                                LISU    3
 DC                                ASD     S         ADD DECIMAL K TO A
 5D                                LR      I,A       STORE SUM IN A,INC ISARL
 4C                                LR      A,S       ADD CARRY
 19                                LNK                     IF PRESENT
 5C                                LR      S,A             TO NEXT DIGIT
 8F  F5                            BR7     ADD1
 7F                                LIS     H'F'      ISOLATE LOWER DIGIT
 FC                                NS      S         OF R37 , A(0)
 94   2                            BNZ     ADD4      A(0) =1 BRANCH
 1C                                POP               RETURN
 66                         ADD4   LISU    6         SET ISAR
 69                                LISL    1             = 61
 20   E                            LI      D'14'     LOAD 14
 EC                                XS      S         TEST A(EX), R61
 84  49                            BZ      OVER      A(EX)=14, OVERFLOW
 4C                                LR      A,S       LOAD AND
 1F                                INC                   INCREMENT A(EX)
 5C                                LR      S,A       STORE IN R61
  8                                LR      K,P       SAVE PC1
 28   2  6F                        PI      RSA       GO TO RIGHT SHIFT A(REG)
  9                                LR      P,K       RETURN PC1
 1C                         ADD5   POP               RETURN
 68                         SUB    LISL    0
 62                         SUB1   LISU    2          ISAR =20 TO 27
 4C                                LR      A,S         LOAD K DIGIT R(XX)
 18                                COM                1'S COMPLEMENT OF K DIGIT
 63                                LISU    3
 DC                                ASD     S          SUBTRACT DECIMAL K FROM A
 5D                                LR      I,A        STORE SUM IN A , INC. ISARL
 1E                                LR      J,W        SAVE STATUS OF CARRY
 4C                                LR      A,S        ADD CARRY
 19                                LNK                     IF PRESENT
 5C                                LR      S,A             TO NEXT DIGIT
 8F  F5                            BR7     SUB1
 68                                LISL    0          ISAR = 30
 20  67                            LI      H'67'      ADD 1 TO A 14
 DC                         SUB4   ASD     S          ADD DECIMAL CORRECTION
 5D                                LR      I,A
 4C                                LR      A,S        ADD CARRY
 19                                LNK                   IF PRESENT
 5C                                LR      S,A             TO NEXT DIGIT
 8F   3                            BR7     SUB5       NOT 7 CONTINUE
 90   5                            BR      SUB6       JUMP
```

```
OBJECT CODE              SOURCE STATEMENT

 20   66          SUB5     LI      H'66'
 90   F4                   BR      SUB4
 1D               SUB6     LR      W,J       LOAD STATUS
 82   1C                   BC      SUB10     CARRY =1 RETURN
 68                        LISL    0         ISAR = 30
 4C               SUB7     LR      A,S       LOAD A DIGIT FROM R(XX)
 18                        COM               1'S COMPLEMENT OF A DIGIT
 5C                        LR      S,A       STORE BACK DIGIT
 70                        LIS     0         CLEAR ACCUMULATOR
 DC                        ASD     S         SUBTRACT A DIGIT FROM ZERO
 5D                        LR      I,A       STORE A DIGIT ,INC ISARL
 8F   F9                   BR7     SUB7
 68                        LISL    0         ISAR = 30
 20   67                   LI      H'67'     ADD 1 TO A14
 DC               SUB8     ASD     S         ADD DEC. CORECTION
 5D                        LR      I,A       STORE & INC. ISAR
 4C                        LR      A,S       ADD CARRY
 19                        LNK                  IF PRESENT
 5C                        LR      S,A           TO NEXT DIGIT
 8F    3                   BR7     SUB9      CONTINUE
 90    5                   BR      SUB9A     ISAR=7 JUMP
 20   66          SUB9     LI      H'66'
 90   F4                   BR      SUB8
 4C               SUB9A    LR      A,S       SIGN OF A REG (R37)
 18                        COM               CHANGE SIGN OF A
 21   F0          SUB10    NI      H'F0'     CLEAR A(0) R37 LOWER
 5C                        LR      S,A       STORE BACK DIGIT
 1C                        POP
 29    1   11     OVER     JMP     ERROR     GO TO ERROR
```

DECIMAL MULTIPLY SUBROUTINE

Abstract

This subroutine will perform a Floating Decimal MULTIPLY on two 14 Digit Signed Numbers. The product is also a 14 Digit Signed Number.

Description

The two numbers (multiplicand and multiplier) as well as the product are stored using one scratch pad location for every 2 BCD digits. The sign uses one-half byte or one-half scratch pad location. The exponent of the numbers and product use one scratch pad location each. The following scratch pad locations are used in the program example:

First Number, Multiplicand, F Register

Digit No	Sign	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Scratchpad location (Octal)	R47		R46		R45		R44		R43		R42		R41		R40	

R47 Upper, Sign of Number "0" positive "F" negative

R47 Lower, Overflow digit

R40 to R46, 14 BCD digits, R46 upper - Most Significant, R40 lower - Least Significant

The exponent of the number is stored in binary, in scratchpad location R62 (Octal).

Second Number, Multiplier, W Register

Digit No	Sign	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Scratchpad Location (Octal)	R57		R56		R55		R54		R53		R52		R51		R50	

R57 Upper, Sign of Number

R57 Lower, Overflow Digit

R50 to R56, 14 BCD Digits

The exponent of the number is stored in binary, in scratchpad location R63 (Octal)

Product, K Register

Digit No	Sign	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Scratchpad location (Octal)	R27		R26		R25		R24		R23		R22		R21		R20	

R27 Upper, Sign of Number

R27 Lower, Overflow digit

R20 to R26, 14 BCD digits

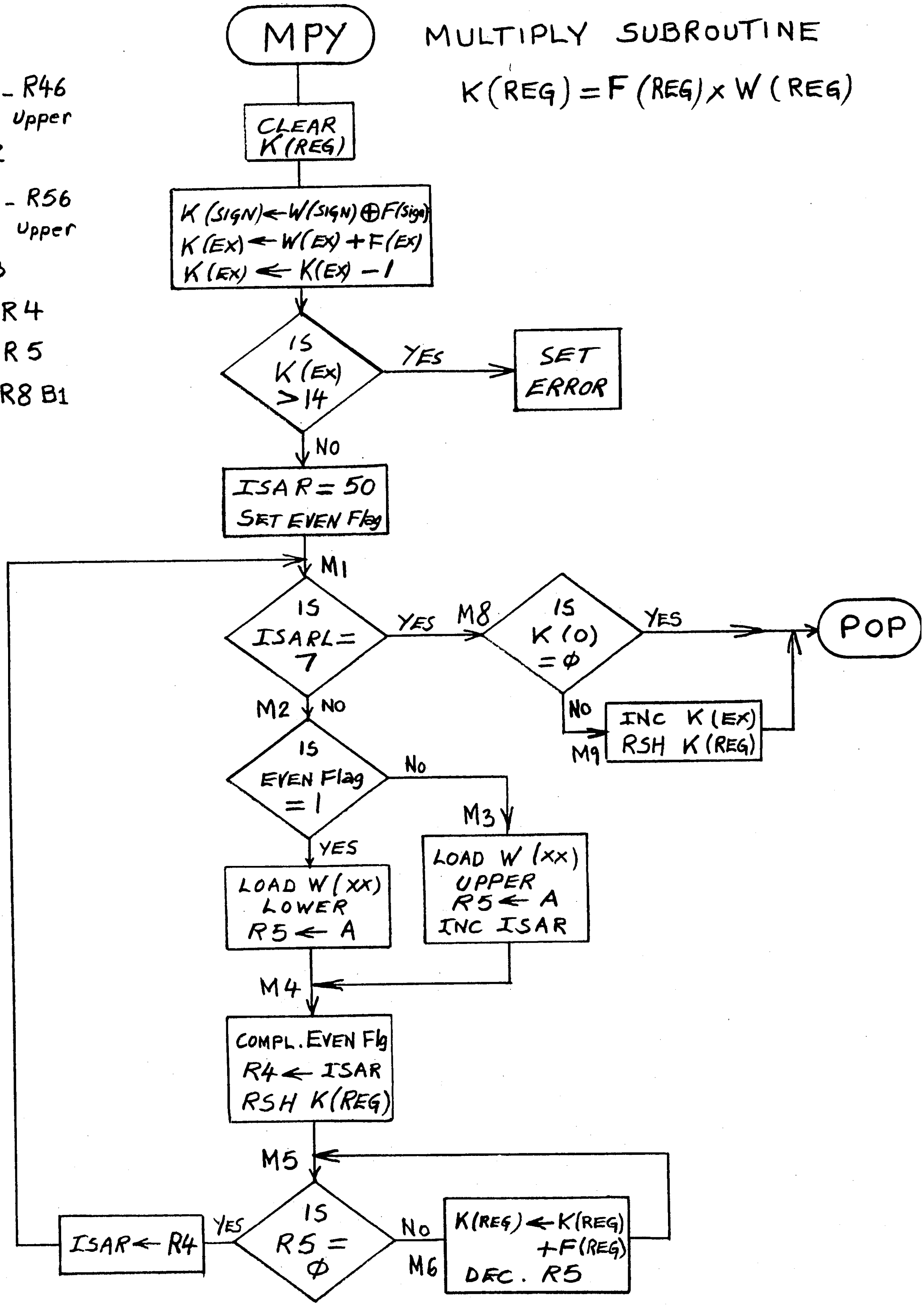
The exponent of the number is stored in binary, in scratchpad location R60.

The number in the F register is multiplied by the number in the W register, the result or Product is stored in the K register with the correct sign and exponent.

MULTIPLY SUBROUTINE

K(REG) = F(REG) x W(REG)

F(REG) = R40 - R46
F(SIGN) = R47 Upper
F(EX) = R62
W(REG) = R50 - R56
W(SIGN) = R57 Upper
W(EX) = R63
TEMP. ISAR = R4
TEMP. DIGIT = R5
EVEN FLAG = R8 B1



```
OBJECT CODE          SOURCE STATEMENT

                     ******* MULTIPLY SUBROUTINE *******
 65                  MPY     LISU    5
 6F                          LISL    7        ISAR=57
 20  F0                      LI      H'F0'    ISOLATE SIGN
 FC                          NS      S          OF W(REG)
 64                          LISU    4        ISAR=47
 EC                          XS      S        XOR SIGNS OF F & W (REG)
 62                          LISU    2        ISAR=27
 5E                          LR      D,A      LOAD SIGN OF K(REG)
 70                          LIS     0        CLEAR K(REG)
 5E                          LR      D,A         R26 TO R20
 8F  FE                      BR7     *-1
 66                          LISU    6        ISAR
 6B                          LISL    3           =63
 4E                          LR      A,D      LOAD W(EX)
 CC                          AS      S        ADD F(EX)
 68                          LISL    0        ISAR=60
 5C                          LR      S,A      STORE W(EX)+F(EX) IN K(EX)
 3C                          DS      S        DEC.K(EX)
 24  F1                      AI      H'F1'    ADD MINUS 15
 85  E6                      BT      5,OVER   K(EX) GREATER THAN 14 OVERFLOW
 65                          LISU    5        ISAR =50
 48                          LR      A,8      SET EVEN FLAG
 22   2                      OI      2          R8B1 =1
 58                          LR      8,A            AND STORE
 8F   3              M1      BR7     M2       IS ISAR =7 ?
 90  31                      BR      M8       YES JUMP
 72                  M2      LIS     2        IS EVEN FLAG
 F8                          NS      8           = 1 ?
 84   7                      BZ      M3
 4C                          LR      A,S      LOAD W (XX)
 21   F                      NI      H'0F'    ISOLATE LOWER DIGIT
 55                          LR      5,A      STORE IN R5
 90   4                      BR      M4
 4D                  M3      LR      A,I      LOAD W(XX)&INC ISAR
 14                          SR      4        SHIFT UPPER DIGIT
 55                          LR      5,A      LOAD MULTIPLIER DIGIT IN R5
 72                  M4      LIS     2        COMPLEMENT
 E8                          XS      8           EVEN FLAG
 58                          LR      8,A         R8B1
 A                           LR      A,IS     SAVE ISAR
 54                          LR      4,A         IN R4
 8                           LR      K,P      SAVE PC1
 28   2  53                  PI      RSK      GO TO RIGHT SHIFT K(REG)
```

```
OBJECT CODE          SOURCE STATEMENT

  9                          LR       P,K      RETURN PC1
 20  FF             M5       LI       H'FF'    TEST IF NUMBER
 F5                          NS       5           IN R5 IS ZERO ?
 94   5                      BNZ      M6
 44                          LR       A,4      LOAD R4
  B                          LR       IS,A       IN ISAR
 90  DD                      BR M1
 68                 M6       LISL     0        ADD F(REG) TO K(REG)
 64                 M7       LISU     4        ISAR = 40
 20  66                      LI       H'66'    BEGIN DECIMAL ADD
 CC                          AS       S        ADD 6,6 TO F(XX)
 62                          LISU     2
 DC                          ASD      S        ADD DECIMAL F(XX) TO K(XX)
 5D                          LR       I,A      STORE SUM IN K, INC ISAR
 4C                          LR       A,S      ADD CARRY
 19                          LNK                  IF PRESENT
 5C                          LR       S,A           TO NEXT DIGIT
 8F  F5                      BR7      M7
 35                          DS       5        DECREMENT NUMBER IN R5
 90  E8                      BR       M5
 62                 M8       LISU     2        ISAR =27
 7F                          LIS      H'F'     ISOLATE LOWER DIGIT
 FC                          NS       S           OF R27,K(0)
 94   2                      BNZ      M9       K(0) NOT ZERO , BRANCH
 1C                          POP               K(0) =ZERO , RETURN
 66                 M9       LISU     6        SET ISAR
 68                          LISL     0           =60
 4C                          LR       A,S      LOAD AND
 1F                          INC                  INCREMENT K(EX)
 5C                          LR       S,A      STORE IN R 60
  8                          LR       K,P      SAVE PC1
 28   2   53                 PI       KSK      GO TO RIGHT SHIFT K(REG)
  9                          LR       P,K      RETURN PC1
 1C                          POP               END OF MPY
```

DECIMAL DIVIDE SUBROUTINE

Abstract

This subroutine will perform a Floating Decimal DIVIDE on two 14 Digit Signed Numbers. The quotient is also a 14 Digit Signed Number.

Description

The two numbers (dividend and divisor) as well as the quotient are stored using one scratchpad location for every 2 BCD digits. The sign uses one-half byte or one-half scratchpad location. The exponent of the numbers and quotient, one scratchpad location each. The following scratchpad locations are used in the program example:

First Number, Divisor, F Register

Digit No	Sign	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Scratchpad location (Octal)	R47		R46		R45		R44		R43		R42		R41		R40	

R47 Upper, Sign of Number "0" positive "F" negative

R47 Lower, Overflow digit

R40 to R46, 14 BCD digits, R46 upper - Most Significant, R40 lower - Least Significant

The exponent of the number is stored in binary, in scratchpad location R62 (Octal).

Second Number, Divident, W Register

Digit No	Sign	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Scratchpad location (Octal)	R57		R56		R55		R54		R53		R52		R51		R50	

R57 Upper, Sign of Number

R57 Lower, Overflow Digit

R50 to R56, 14 BCD Digits

The exponent of the number is stored in binary, in scratchpad location R63 (Octal).

Quotient, K Register

Digit No	Sign	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Scratchpad location (Octal)	R27		R26		R25		R24		R23		R22		R21		R20	

R27 Upper, Sign of Number

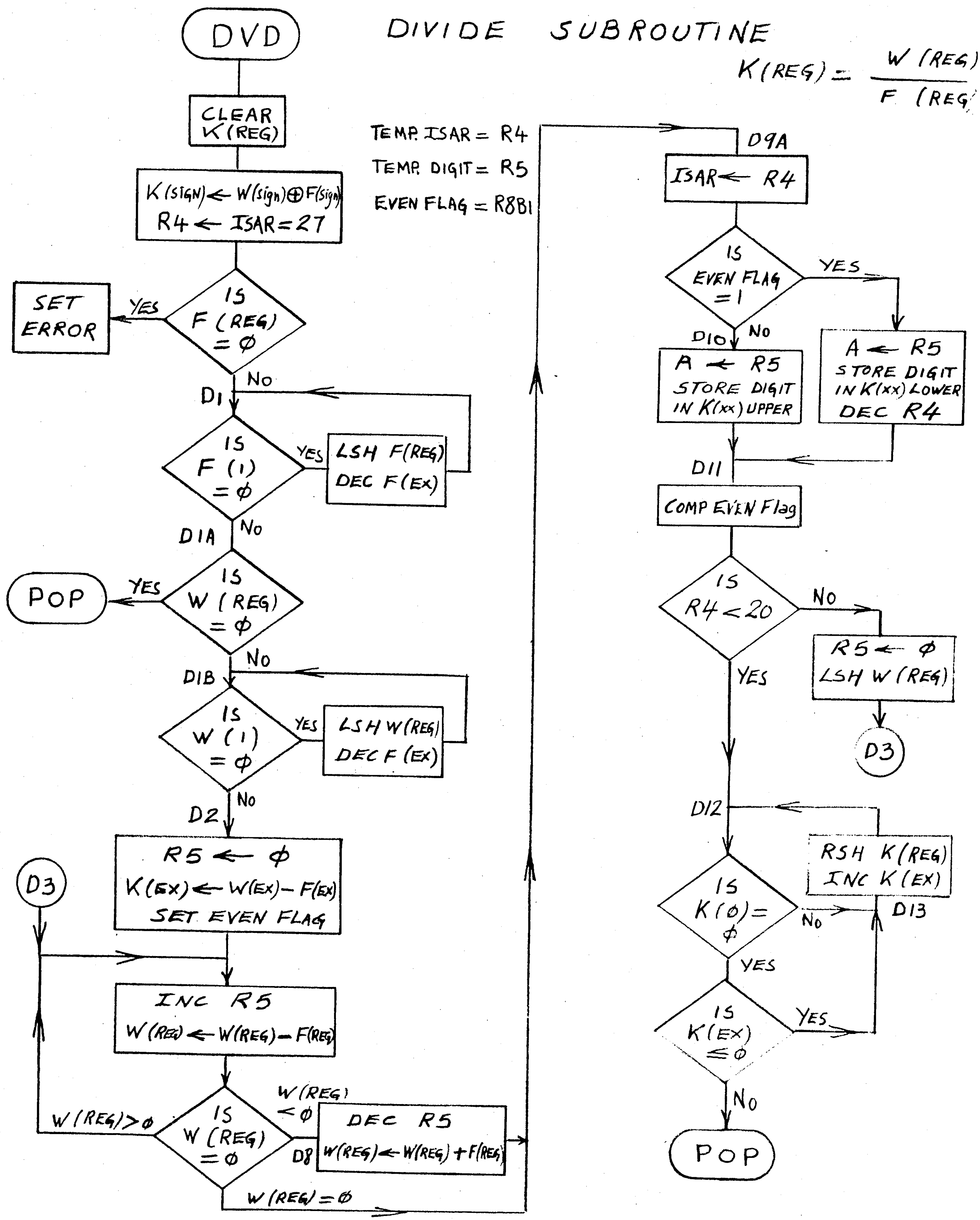
R27 Lower, Overflow Digit

R20 to R26, 14 BCD digits

The exponent of the number is stored in binary, in scratchpad location R60.

The number in the W register is divided by the number in the F register, the result or quotient is stored in the K register with the correct sign and exponent.

DIVIDE SUBROUTINE



```
                              ******* DIVIDE SUBROUTINE *******
65                            DVD     LISU    5
6F                                    LISL    7         ISAR=57
20   F0                               LI      H'F0'     ISOLATE SIGN
FC                                    NS      S           OF W (REG)
64                                    LISU    4         ISAR=47
EC                                    XS      S         XOR SIGNS OF W & F (REG)
62                                    LISU    2         ISAR=27
5E                                    LR      D,A       LOAD SIGN OF K (REG)
70                                    LIS     0         CLEAR K(REG)
5E                                    LR      D,A           R26 TO R20
8F   FE                               BR7     *-1
 A                                    LR      A,IS      STORE ISAR
54                                    LR      4,A           IN R4
64                                    LISU    4         SET ISAR
6E                                    LISL    6            =46
70                                    LIS     0         LOAD ZERO
CE                                    AS      D         TEST F(REG)
94    5                               BNZ     D1        NOT ZERO BRANCH
8F   FC                               BR7     *-3
90   83                               BR      OVER      F(REG)=0 ERROR
64                            D1      LISU    4
6E                                    LISL    6         ISAR=46
20   F0                               LI      H'F0'     TEST F(1)
FC                                    NS      S          . IN R46 UPPER
94    F                               BNZ     D1A       IF NOT ZERO JUMP
4D                                    LR      A,I       LEFT
14                                    SR      4           SHIFT
EC                                    XS      S             F(REG)
5E                                    LR      D,A             BY
4C                                    LR      A,S               ONE
15                                    SL      4                   DIGIT
5E                                    LR      D,A
8F   F8                               BR7     *-7
66                                    LISU    6
6A                                    LISL    2         ISAR=62
3C                                    DS      S         DECREMENT F(EX)
90   EC                               BR      D1
65                            D1A     LISU    5         SET ISAR
6F                                    LISL    7            =57
70                                    LIS     0         CLEAR W(0 & SIGN)
```

```
OBJECT CODE          SOURCE STATEMENT

 5E                         LR      D,A         R57
 CE                         AS      D        TEST W(REG)
 94   4                     BNZ     D1B      NOT ZERO BRANCH
 8F  FC                     BR7     *-3
 1C                         POP              W(REG)=0 RETURN
 65                  D1B    LISU    5
 6E                         LISL    6        ISAR=56
 20  F0                     LI      H'F0'    TEST W(1)
 FC                         NS      S          IN R56 UPPER
 94   F                     BNZ     D2       NOT ZERO JUMP
 4D                         LR      A,I      LEFT
 14                         SR      4          SHIFT
 EC                         XS      S            W(REG)
 5E                         LR      D,A            BY
 4C                         LR      A,S              ONE
 15                         SL      4                  DIGIT
 5E                         LR      D,A
 8F  F8                     BR7     *-7
 66                         LISU    6
 6B                         LISL    3        ISAR=63
 3C                         DS      S        DECREMENT W(EX)
 90  EC                     BR      D1B
 70                  D2     LIS     0
 55                         LR      5,A      CLEAR TEMP. QUOTIENT R5
 66                         LISU    6        SET ISAR
 6A                         LISL    2            =62
 4D                         LR      A,I      LOAD F(EX)
 18                         COM
 1F                         INC               2'S COMPLEMENT
 CC                         AS      S        W(EX) - F(EX)
 68                         LISL    0        ISAR= 60
 5C                         LR      S,A      STORE NEW K(EX)
 48                         LR      A,8      SET EVEN FLAG
 22   2                     OI      2          R8B1 =1
 58                         LR      8,A       AND STORE
 45                  D3     LR      A,5
 1F                         INC              INCREMENT
 55                         LR      5,A        TEMP. QUOTIENT IN R5
 68                         LISL    0
 64                  D4     LISU    4        ISAR =40
 4C                         LR      A,S      LOAD F(XX)
 18                         COM              1'S COMPLEMENT
 65                         LISU    5
 DC                         ASD     S        SUBTACT DEC.F(XX) FROM W(XX)
 5D                         LR      I,A      STORE RESULT IN W(XX),INC ISAR
 4C                         LR      A,S      ADD CARRY
 19                         LNK                IF PRESENT
 5C                         LR      S,A          STORE IN W(XX)+1
 8F  F6                     BR7     D4
 20  FF                     LI      H'FF'    SUBTACT '00' FROM R57
```

```
OBJECT CODE        SOURCE STATEMENT

DC                        ASD    S
5C                        LR     S,A        AND STORE
1E                        LR     J,W       SAVE STATUS OF CARRY
68                        LISL   0
20  67                    LI     H'67'     ADD 1 TO W(14)
DC               D5       ASD    S         ADD DECIMAL CURECTION
5D                        LR     I,A          AND STORE
4C                        LR     A,S
19                        LNK              ADD CARRY IF PRESENT
5C                        LR     S,A          AND STORE
8F   3                    BR7    D6
90   5                    BR     D7
20  66           D6       LI     H'66'
90  F4                    BR     D5
20  66           D7       LI     H'66'     ADD '00' TO
DC                        ASD    S            R57
5C                        LR     S,A          AND STORE
82  18                    BC     D9A       W(REG) =0 BRANCH
1D                        LR     W,J       LOAD STATUS
92   3                    BNC    D8        RESULT NEGATIVE ADD F(REG) TO W(RE
90  D2                    BR     D3
35               D8       DS     5         DEC. TEMP. QUOTIENT
68                        LISL   0
64               D9       LISU   4
20  66                    LI     H'66'
CC                        AS     S         ADD 6,6 TO F DIGITS F(XX)
65                        LISU   5
DC                        ASD    S         ADD DECIMAL F(XX) TO W(XX)
5D                        LR     I,A         AND STORE
4C                        LR     A,S
19                        LNK              ADD CARRY IF PRESENT
5C                        LR     S,A          AND STORE
8F  F5                    BR7    D9
20  66                    LI     H'66'     ADD '00'
DC                        ASD    S            TO R57
5C                        LR     S,A          AND STORE
44               D9A      LR     A,4       LOAD R4
 B                        LR     IS,A         IN ISAR
72                        LIS    2          IS EVEN FLAG
F8                        NS     8             =1 ?
84   7                    BZ     D10
45                        LR     A,5       LOAD QUOTIENT
EC                        XS     S            IN
5C                        LR     S,A          K(XX) EVEN
34                        DS     4         DECREMENT DIGIT COUNTER
90   4                    BR     D11
45               D10      LR     A,5       LOAD QUOTIENT
15                        SL     4            IN
5C                        LR     S,A         K(XX) ODD
72               D11      LIS    2         COMPLEMENT
```

```
OBJECT CODE              SOURCE STATEMENT

 E8                               XS      8          EVEN FLAG
 58                               LR      8,A         (R8B1)
 7F                               LIS     H'F'     LOAD 15 (OCTAL 17)
 E4                               XS      4
 84   10                          BZ      D12      R4=17 OCTAL , BRANCH
 70                               LIS     0
 55                               LR      5,A      CLEAR TEMP. QUOTIENT
 65                     LSW       LISU    5        SET ISAR
 6E                               LISL    6           =56
 4D                     LSW1      LR      A,I      LEFT
 14                               SR      4          SHIFT
 EC                               XS      S            W REG
 5E                               LR      D,A
 4C                               LR      A,S            BY
 15                               SL      4                ONE
 5E                               LR      D,A                DIGIT
 8F   F8                          BR7     LSW1
 90   9B                          BR      D3
 62                     D12       LISU    2        SET ISAR
 6F                               LISL    7           =27
 7F                               LIS     H'F'
 FC                               NS      S        ISOLATE K(0) , R27 LOWER
 94    A                          BNZ     D13
 66                               LISU    6        SET ISAR
 68                               LISL    0           =60
 70                               LIS     0        TEST
 CC                               AS      S           K(EX) R60
 91    4                          BM      D13      K(EX) NEG. BRANCH
 84    2                          BZ      D13      K(EX) ZERO BRANCH
 1C                               POP
  8                     D13       LR      K,P      SAVE PC1
 28    2   53                     PI      RSK      GO TO RIGHT SHIFT K (REG)
  9                               LR      P,K      RETURN PC1
 66                               LISU    6        SET ISAR
 68                               LISL    0           =60
 4C                               LR      A,S      LOAD K(EX)
 1F                               INC
 5C                               LR      S,A      INC. K(EX) IN R60
 90   E6                          BR      D12
```

BOOTSTRAP PROGRAM FOR READING PAPERTAPES

Abstract

This application note describes an F8 program that is used to load information from a paper tape into Read/Write memory. The tape may be read from either a serial teletype or a parallel device such as a tape reader; input routines for both type of devices are included. Program size is 250 bytes. Flow chart and operating instructions included.

This application note describes a program that reads a formatted paper tape into an F8 system. The system is assumed to include the F8 MI (Memory Interface) circuit and RAM memory. This program is designed for loading user's programs into an F8 Micromodule system - the program is supplied in two PROMs (256 X 4 each) and is able to load into RAM that occupies addresses H'0800' and H'0BFF' during loading.

The paper tape is typically produced as output from the F8 cross-assembler which resides in a time-sharing service. All information on the tape is in ASCII representation (i.e. the character 'A' is represented by one line of holes 11000y001 where y is the spocket hole). Each block on the tape begins with an X; the X is then followed by 16 characters from the hex set 0 - 9 and A - F. The 16 characters give 8 data bytes to be loaded into RAM memory. The 16 data characters are followed by a one character check sum; the checksum is used to check the validity of the input. The entire block is typically followed by a carriage return and line feed; other comments can also be inserted if "X"s and "S"s are not included. Since the tape is all ASCII, it can be understood when read and printed. Figure 1 is an example.

The checksum is included in each block of data. It consists of a single character. The checksum is calculated by performing a summation of all 16 half bytes of the block, using hex arithematic. The resulting sum is truncated to its least 4 binary digits; the checksum character is the ASCII representation of the resulting hex digit. When the program reads a paper tape it calculates an internally held

```
******************      LEADER
******************
******************
******************
S0000*************      Set Load Address for H'0000'
X71B0206050A015819      Data Lines
X0E2092512800DD207
XB2077006900E209CF      "X" at start
X077006B4B5A51381F
X0390FF2A08007054E
X28009B255384322 55     16 characters represent 8 bytes
X2A843B235894F2572
X78555428007C1735F
X94FA28009B213F24E      Checksum character
XD082032439181FC78
X210F84D5A01581070
X2093512800DD90FFF
X28007C24085A28004
X7C5B1090BCF484B97
XA01581072093512 8E
X00DD90FF087256287
X009B213F24D082039
X2439155114C75736B
X8405415390EA41149
XC3091C0DA591FE7FE
X181F94FEA41813121
X5171B5A581FE70B5A
X411C7952A091FE40E
X18900123 1F94FBA01
X91F32180C13291115
X840212514018BBBB0
XBB2B1F94FAA090EB5
XA04181EF1C7B52702
XB04018BBBBBB2B1F7
X94FA32840C412101C
XB0411224805190EA8
X1C00000000000000D
******************      Marks end of tape
******************
******************
******************
```

Figure 1

checksum using these same rules. If the internal checksum does not match the checksum read from the tape, the tape is immediatly stopped--which will be at the end of the bad block.

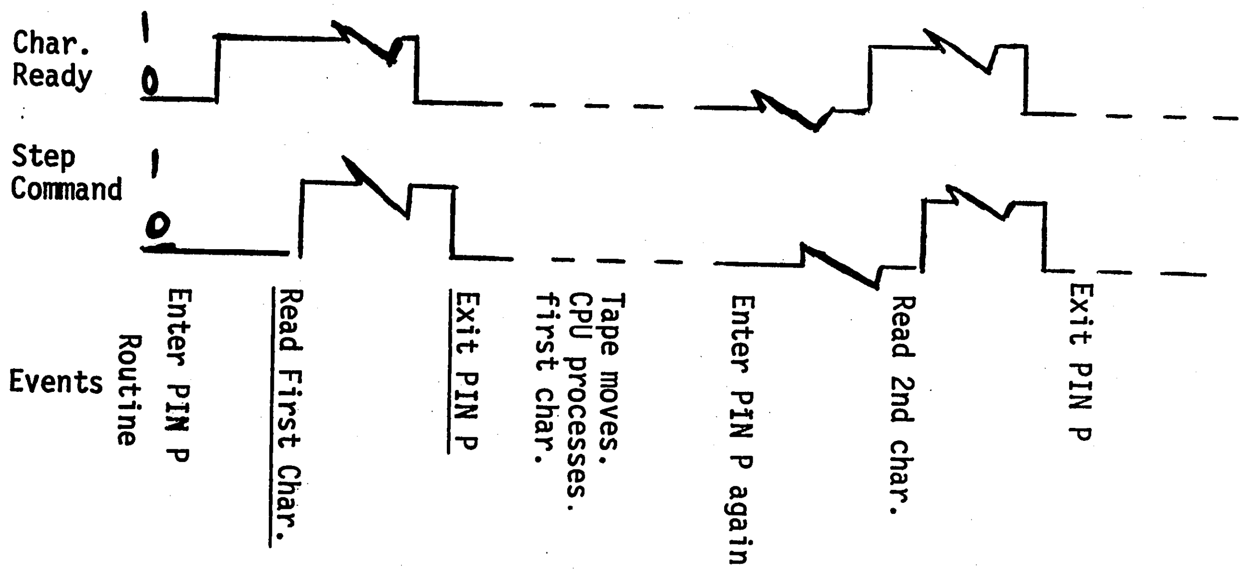
The paper tape format also includes provisions for setting an address that is to be used for loading data blocks that follow. The format is "SXXXX" where XXXX represents the 4 digit hex load address. There may be more than one load address on a tape--additional load addresses may be inserted between data blocks.

The paper tape may be read from either a serial or parallel device. I/O Port Ø is used for the serial device. If bit 3 of Port Ø is strapped to ground, the program will assume that a serial device is present and that input will be from it. The program sends READER ON, READER OFF control characters to control tape start and stop.

The serial routines use the 11 bit asynchronous format that is standard on teletypes; the 11 bits are 1 start, 8 data,and 2 stop bits. Parity is not checked on input. The teletype routines use software delay loops to set the character rate. The delay loops are set for characters/sec--30 characters/second on a 2usec system; however, the character rate is easily adjusted by changing the number that is loaded into the register "BAUD" at the start of the program.

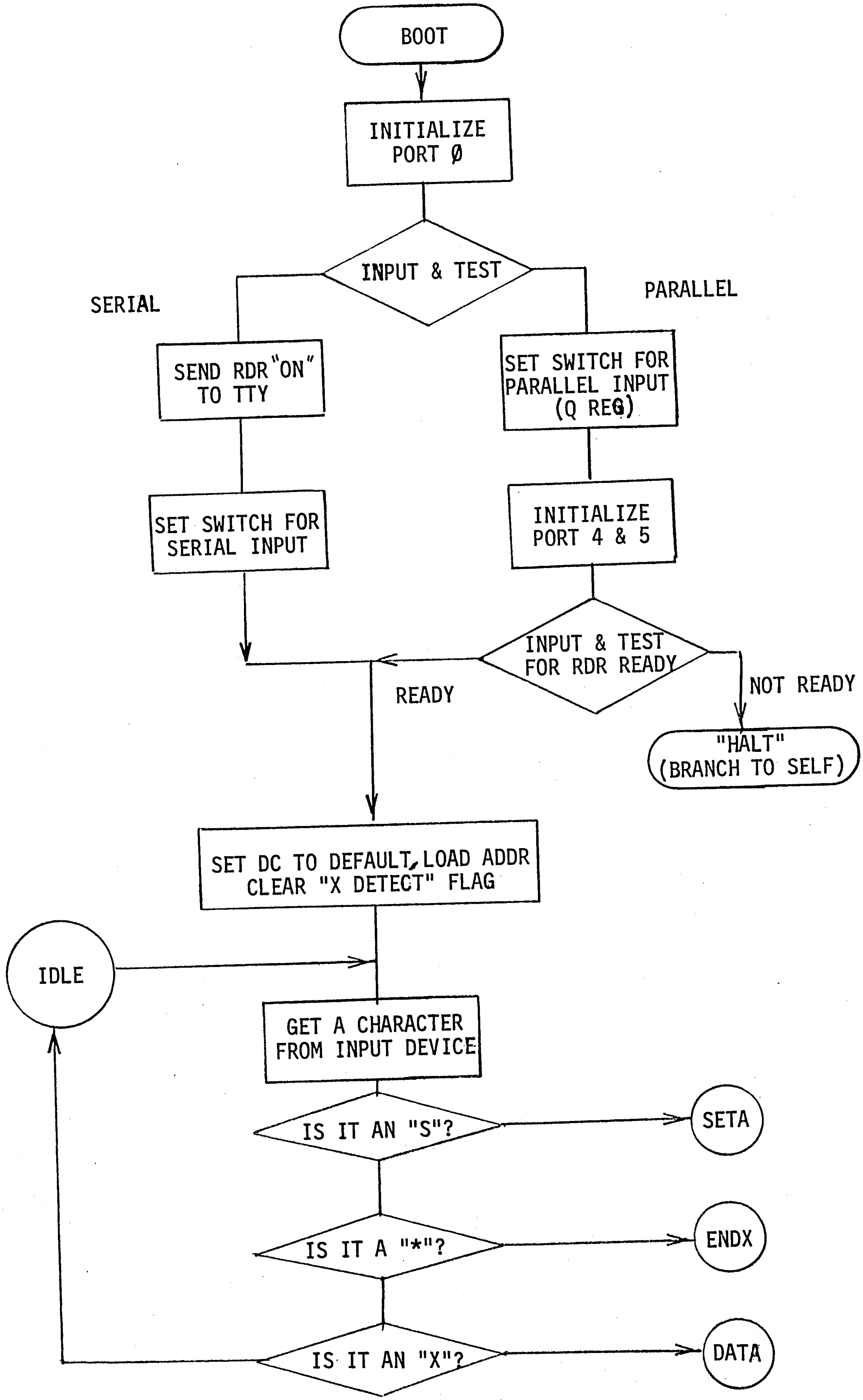
The parallel input routine uses a generalized "handshaking" protocol that applies to many devices.

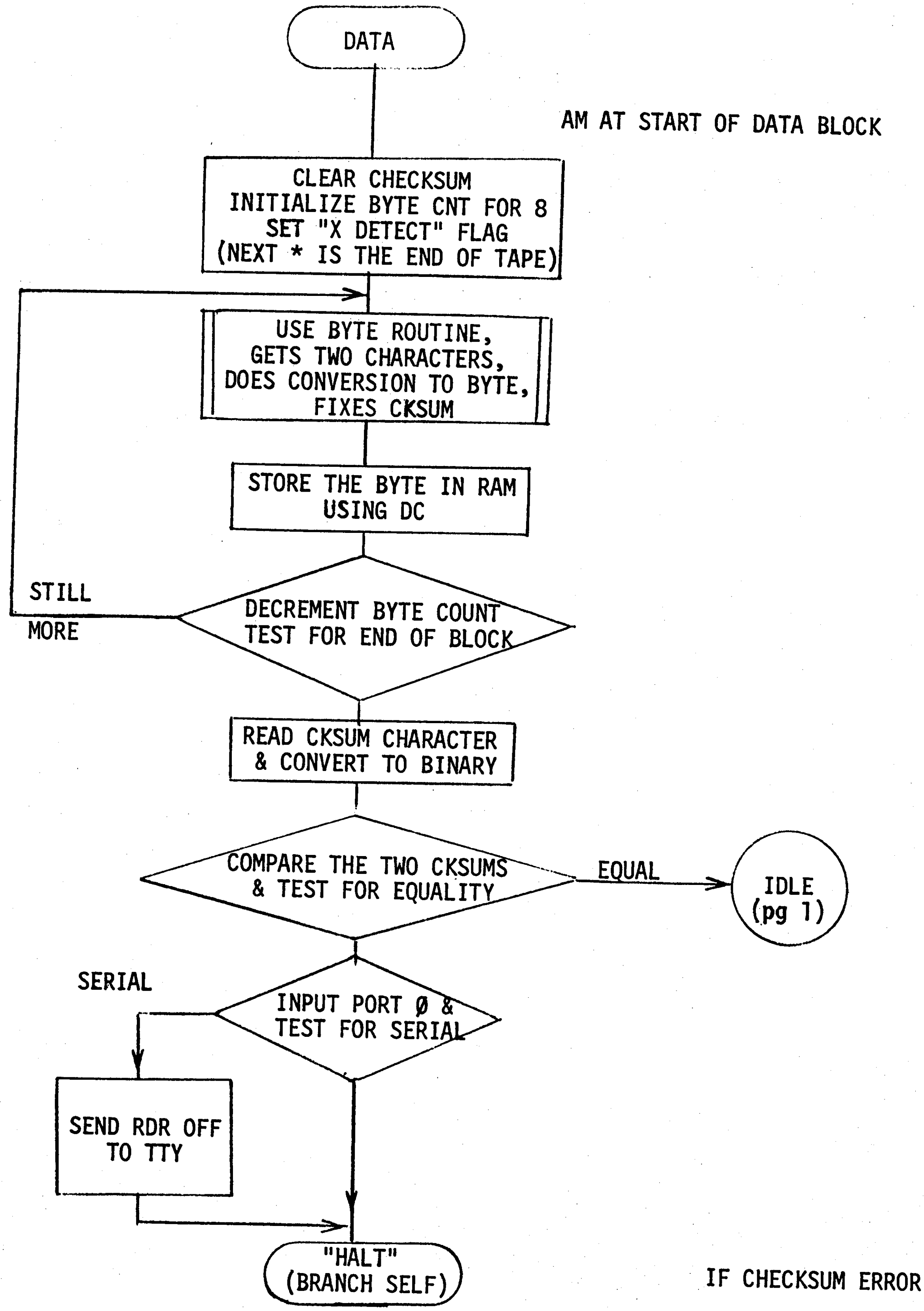
The program waits for the device to indicate ready before reading a character; the device waits for a command from the program before starting to move the tape to the character. Thus slow and fast readers can be used with equal ease. The parallel input could even accommodate a FIFO; the only requirement is that the device be able to operate in the step mode and that the character ready stay low after the step command is received for at least the minimum width of that command: 24usec with a 2µsec F8 system. The figure below shows the timing relationship of the step command and character ready status line.

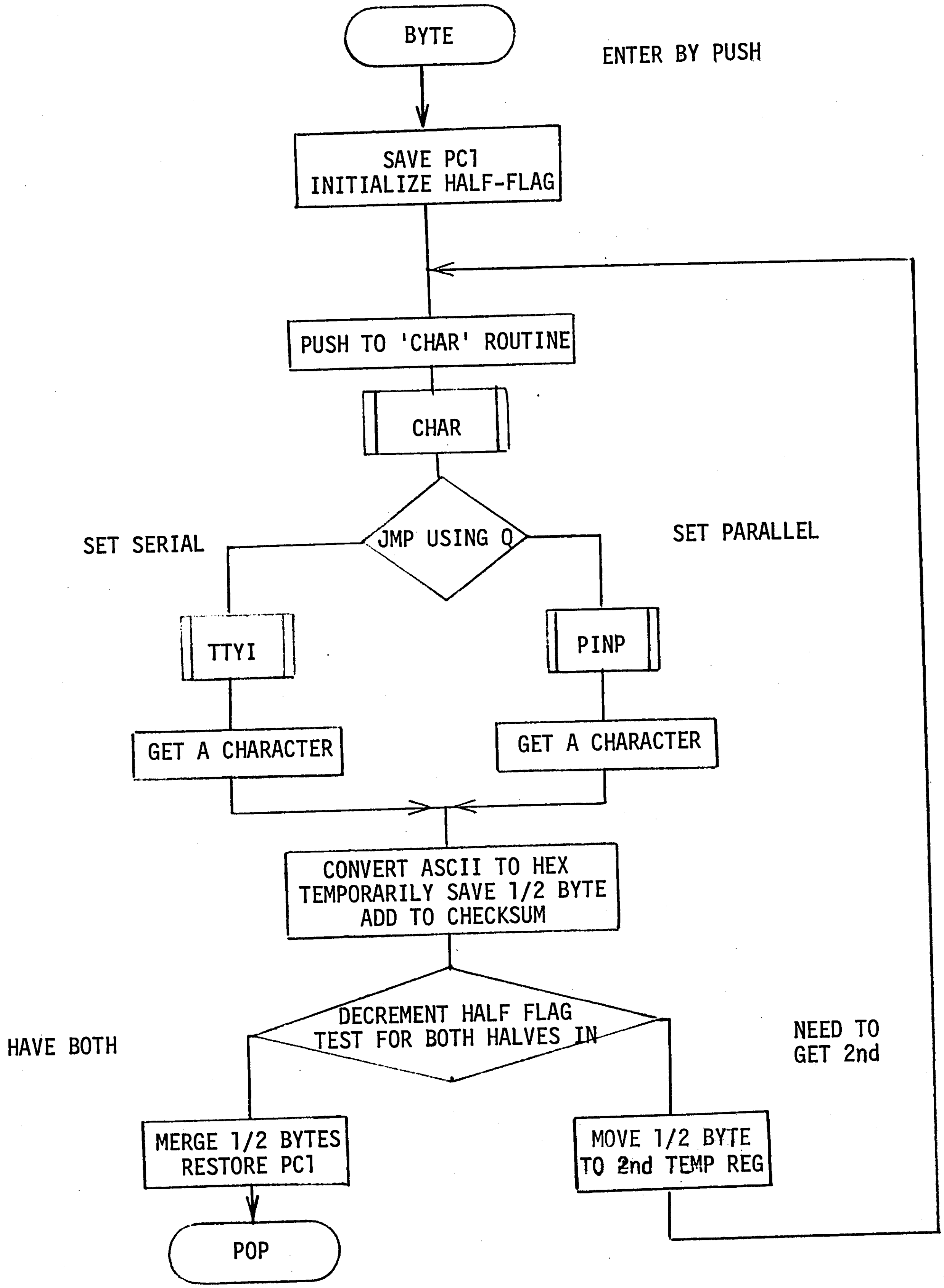


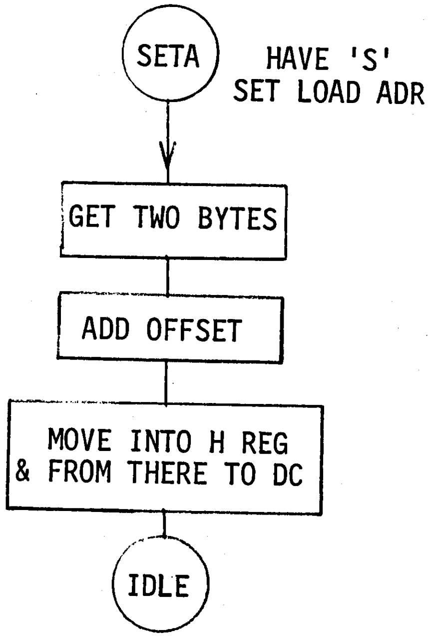
OPERATING INSTRUCTIONS:

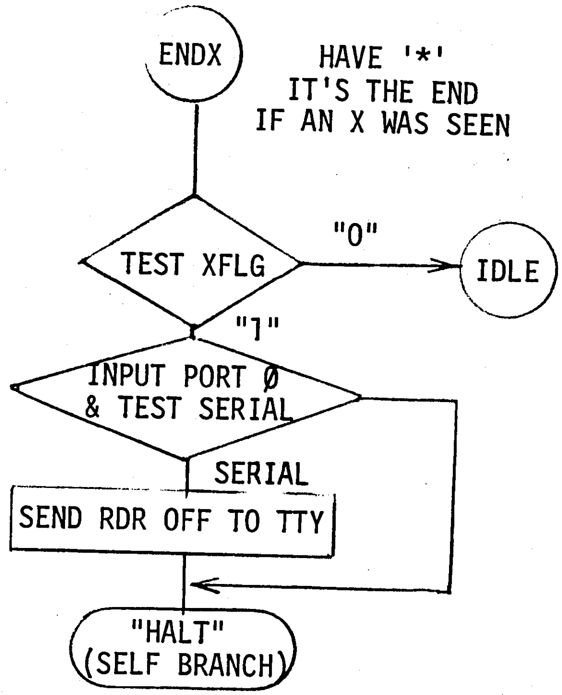
1. Disable the reader: for instance open the read head or set reader control to STOP or FREE.
2. Reset the system and start it running to initialize the I/O ports.
3. Load the tape into the reader anywhere before the start of the punching.
4. Again reset the system: the tape will begin moving as it is read into the system.
5. The system will "halt" at H'007C' when finished. A "halt" at H'0060' indicates a tape error; restart. A "halt" at H'0021' while using a parallel reader occurs if the reader is not ready. (The "halt" is a branch instruction branching to itself).

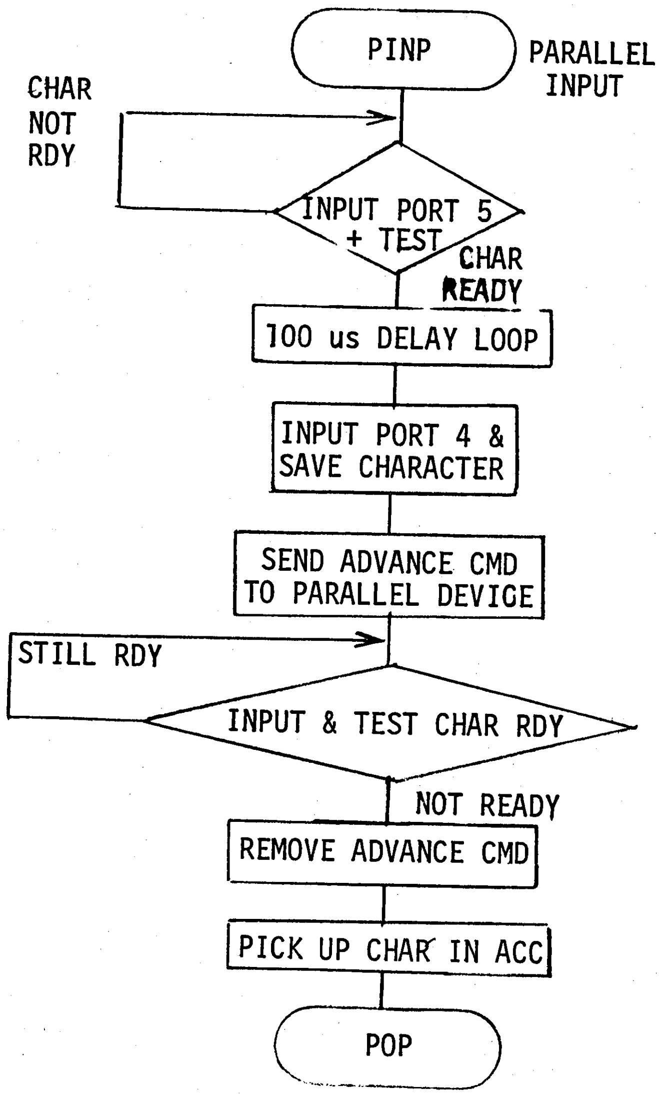

BOOT
INITIALIZE PORT Ø
INPUT & TEST
SERIAL
PARALLEL
SEND RDR "ON" TO TTY
SET SWITCH FOR PARALLEL INPUT (Q REG)
SET SWITCH FOR SERIAL INPUT
INITIALIZE PORT 4 & 5
INPUT & TEST FOR RDR READY
READY
NOT READY
"HALT" (BRANCH TO SELF)
SET DC TO DEFAULT, LOAD ADDR CLEAR "X DETECT" FLAG
IDLE
GET A CHARACTER FROM INPUT DEVICE
IS IT AN "S"?
SETA
IS IT A "*"?
ENDX
IS IT AN "X"?
DATA


DATA
AM AT START OF DATA BLOCK
CLEAR CHECKSUM
INITIALIZE BYTE CNT FOR 8
SET "X DETECT" FLAG
(NEXT * IS THE END OF TAPE)
USE BYTE ROUTINE,
GETS TWO CHARACTERS,
DOES CONVERSION TO BYTE,
FIXES CKSUM
STORE THE BYTE IN RAM
USING DC
STILL
MORE
DECREMENT BYTE COUNT
TEST FOR END OF BLOCK
READ CKSUM CHARACTER
& CONVERT TO BINARY
COMPARE THE TWO CKSUMS
& TEST FOR EQUALITY
EQUAL
IDLE
(pg 1)
SERIAL
INPUT PORT Ø &
TEST FOR SERIAL
SEND RDR OFF
TO TTY
"HALT"
(BRANCH SELF)
IF CHECKSUM ERROR


BYTE
ENTER BY PUSH
SAVE PC1
INITIALIZE HALF-FLAG
PUSH TO 'CHAR' ROUTINE
CHAR
JMP USING Q
SET SERIAL
SET PARALLEL
TTYI
PINP
GET A CHARACTER
GET A CHARACTER
CONVERT ASCII TO HEX
TEMPORARILY SAVE 1/2 BYTE
ADD TO CHECKSUM
DECREMENT HALF FLAG
TEST FOR BOTH HALVES IN
HAVE BOTH
NEED TO
GET 2nd
MERGE 1/2 BYTES
RESTORE PC1
MOVE 1/2 BYTE
TO 2nd TEMP REG
POP


SETA
HAVE 'S'
SET LOAD ADR
GET TWO BYTES
ADD OFFSET
MOVE INTO H REG
& FROM THERE TO DC
IDLE


ENDX
HAVE '*'
IT'S THE END
IF AN X WAS SEEN
TEST XFLG
"0"
IDLE
"1"
INPUT PORT Ø
& TEST SERIAL
SERIAL
SEND RDR OFF TO TTY
"HALT"
(SELF BRANCH)

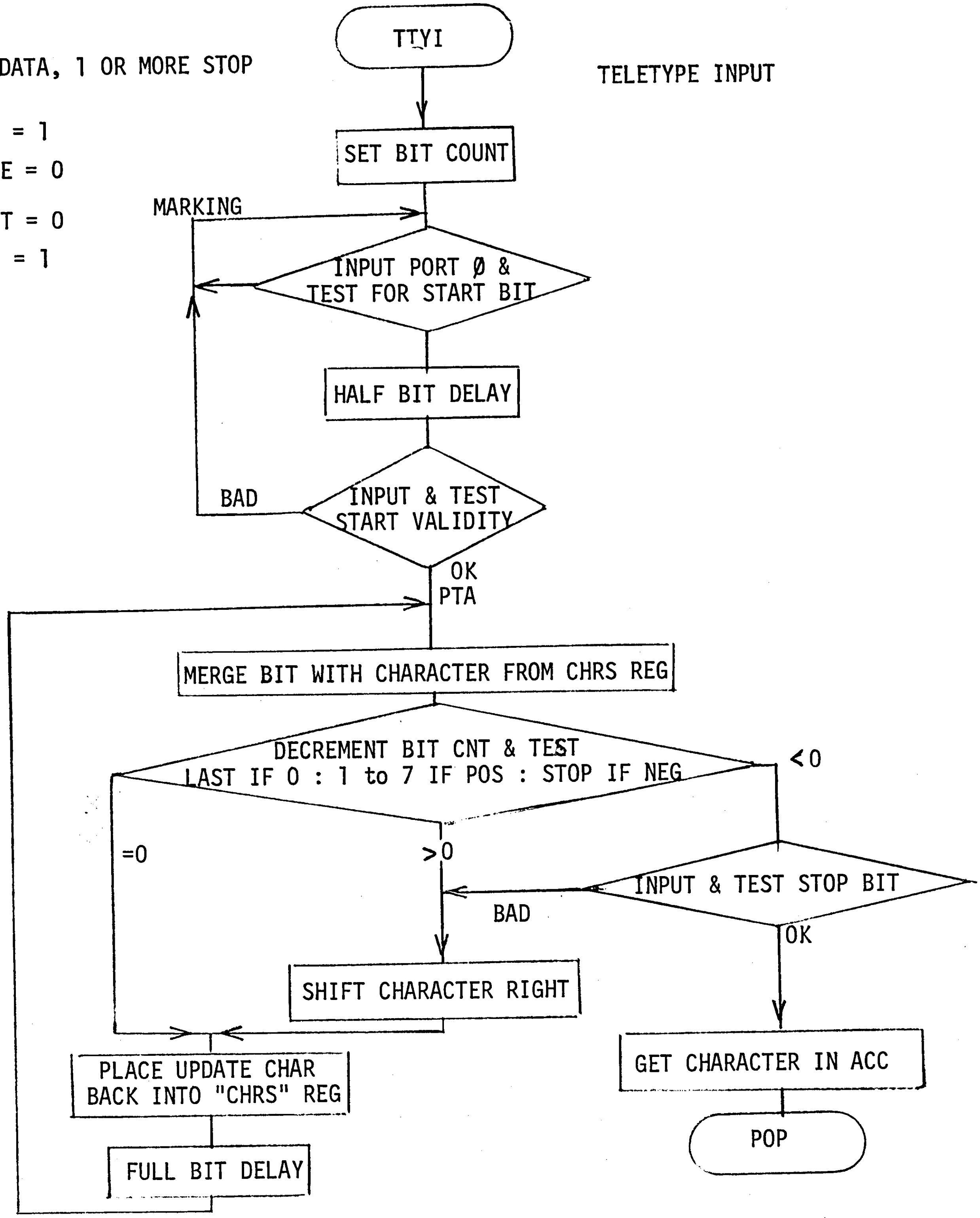

PINP
PARALLEL
INPUT
CHAR
NOT
RDY
INPUT PORT 5
+ TEST
CHAR
READY
100 us DELAY LOOP
INPUT PORT 4 &
SAVE CHARACTER
SEND ADVANCE CMD
TO PARALLEL DEVICE
STILL RDY
INPUT & TEST CHAR RDY
NOT READY
REMOVE ADVANCE CMD
PICK UP CHAR IN ACC
POP

1 START, 8 DATA, 1 OR MORE STOP

MARK = 1
SPACE = 0

START = 0
STOP = 1

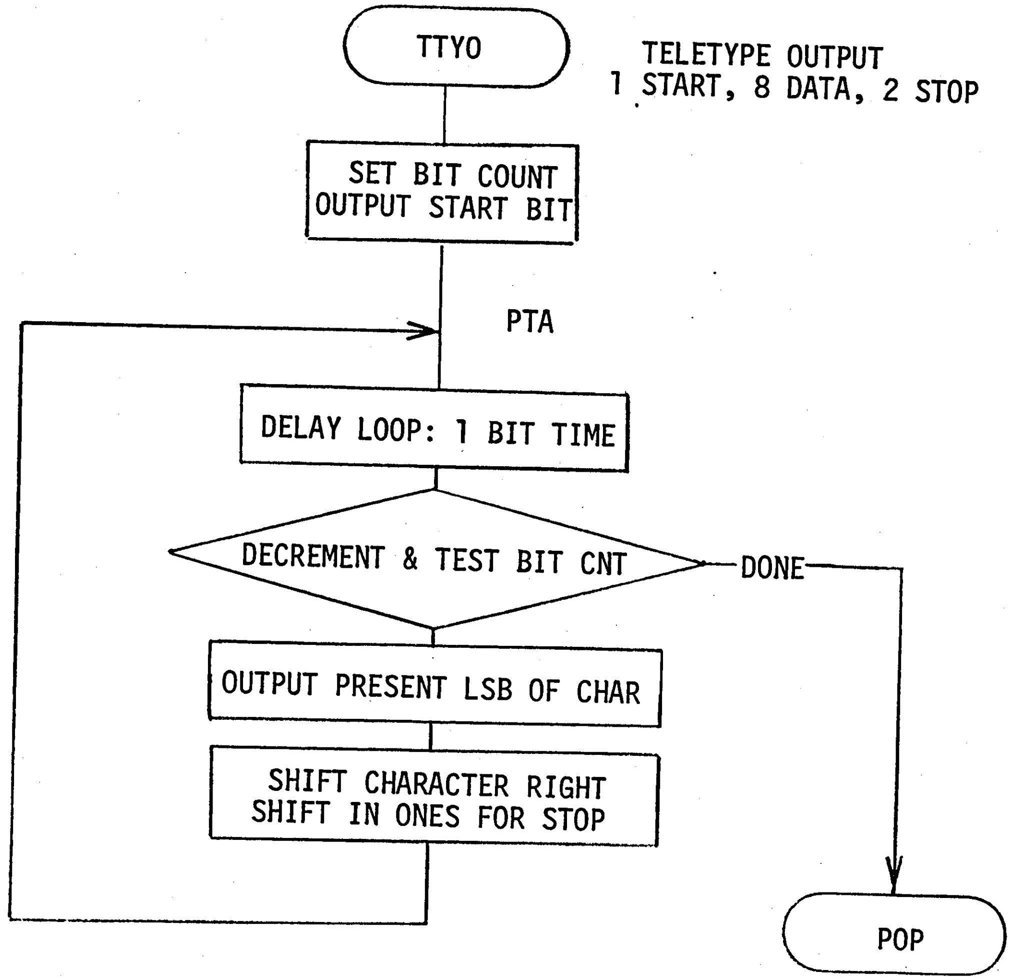
TELETYPE INPUT



SNAPSHOTS AT PT A

INPUT = "E"
0100 0101

BIT	BCNT	CONTENTS OF REG 'CHRS'
START	9	XXXX XXXX
D0—	8	0XXX XXXX
D1—	7	10XX XXXX
D2—	6	010X XXXX
D3—	5	1010 XXXX
D4—	4	0101 0XXX
D5—	3	0010 10XX
D6—	2	0001 010X
D7—	1	100[illegible] 1[illegible]
STOP	0	01[illegible]



CHARACTER & BIT COUNT AT PT A
IF INPUT IS "E" = 0100 0101

	CHAR	BIT CNT	OUTPUT
START	0100 0101	11	0
0	1010 0010	10	1
1	1101 0001	9	0
2	1110 1000	8	1
3	1111 0100	7	0
4	1111 1010	6	0
5	1111 1101	5	0
6	1111 1110	4	1
7	1111 1111	3	0
STOP 1	1111 1111	2	1
STOP 2	1111 1111	1	1

```
  LOC   OBJECT CODE     STMT    SOURCE STATEMENT

PASS 1 COMPLETE      6.25 SEC
                        ********  BOOT PROGRAM FOR MMOD AND TTY OR READER AND FBA FORMAT****
                        *****        FOR EITHER SERIAL TTY OR PARALLEL TAPE READER ***
                        *****            SERIAL TTY WILL BE USED IF BIT 3 OF PORT 0 IS TIED LOW
                        *** FILE ALLOCATION          0: TEMP STORE FOR CHAR
                        *                            1: CHK SUM CHARACTER
                        *                            2: CHARACTER COUNT -- 16 CHARACTERS PER LIN
                        *                            3:  HALF BYTE FLAG
                        *                            4: TEMP STORE FOR 1ST HALF BYTE (UPPER)
                        *                            5:  X DETECT FLAG
                        *
                        *
                        ***** PORT ALLOCATION
                        *                            0: SERIAL TTY--  BIT 0: PNTR OUTPUT MARK=0V
                        *                                 BIT 3: TIE LOW TO CHOOSE TTY
                        *                                 BIT 7: KEYBRD INPUT MARK=0V
                        *                            4: PARALLEL BYTE INPUT
                        *                            5:  BIT 0-- STEP READER  0V=DRIVE LEFT
                        *                              BIT 6-- READER READY 5V=READY
                        *                                BIT 7-- SPROCKET  5V=CHAR READY
                        *
                        *
                        *
                   1           BASE      HEX
                        *
                        *
                   2           ORG       H'0000'
                   3    BAUD   EQU       0
                   4    CHRS   EQU       1
                   5    BCNT   EQU       2
                   6    TEMP   EQU       3
```

```
LOC  OBJECT CODE   STMT   SOURCE STATEMENT
                     7   XFLG   EQU    4
                     8   CCNT   EQU    5
                     9   HFLG   EQU    6
                    10   CKSM   EQU    7
  0   71            11          LIS    01
  1   B0            12          OUTS   0        INITIALIZE TTY PORT TO MARK STATE
  2   20  6         13          LI     H'06'
  4   50            14          LR     BAUD,A   SET DELAY TIME FOR TTY ROUTINES--
                         *                      3.3MS=30CPS, 9.09MS=10CPS
                         *
  5   A0            15          INS    0        INPUT SERIAL PORT
  6   15            16          SL     4
  7   81  E         17          BP     PB00                AND CHK FOR SERIAL TTY
                         *
  9   20  91        18          LI     H'91'    READER ON CHARACTER
  B   51            19          LR     CHRS,A   PASS IT
  C   28  0  DD     20          PI     TTYO              AND TYPE IT
  F   20  B2        21          LI     TTYI
 11   7             22          LR     QL,A     SET Q REG FOR SERIAL INPUT CALLS
 12   70            23          LIS    0
 13   6             24          LR     QU,A
 14   90  E         25          BR     BOTI     SKIP OVER PARALLEL CODE
                         *
 16   20  9E        26   PB00   LI     PINP
 18   7             27          LR     QL,A     SET Q FOR PARALLEL INPUT CALLLS
 19   70            28          LIS    0
 1A   6             29          LR     QU,A
 1B   B8            30          OUTS   8        INITIALIZE INPUT PORT
 1C   B9            31          OUTS   9        INITIALIZE CONTROL PORT FOR READER OFF
 1D   A9            32          INS    9
 1E   13            33          SL     1        GET TO READER READY BIT
 1F   81  3         34          BP     *+4
 21   90  FF        35   SLFI   BR     SLFI     LOCK UP IN LOOP IF READER NOT READY
                         *
                         *
 23   2A  8  0      36   BOTI   DCI    H'0800'  SET DC FOR LOADING ADDRESS
 26   70            37          LIS    0
 27   54            38          LR     XFLG,A   CLEAR FIRST X DETECT FLAG
                         *
                         *
 28   28  0  9D     39   IDLE   PI     CHAR     GET HEADER CHARACTER
 2B   13            40          SL     1        CLEAR PARITY BIT
```

```
LOC  OBJECT CODE    STMT   SOURCE STATEMENT

2C   12             41              SR    1
2D   25  53         42              CI    C'S'      IS IT AN LOAD ADDRESS?
2F   84  32         43              BZ    SETA
31   25  2A         44              CI    C'*'      IS IT THE END OF THE TAPE?
33   84  3B         45              BZ    ENDX
35   23  58         46              XI    C'X'      WELL, IF IT ISN'T AN X, THEN LET'S GO
37   94  F0         47              BNZ   IDLE
                           *
                           *
                           ******   HAVE THE START OF A DATA LINE   *******
                           *
39   57             48              LR    CKSM,A    INTIALIZE CHK SUM  (ACC=0)
3A   78             49              LIS   H'08'
3B   55             50              LR    CCNT,A    INITIALIZE BYTE COUNT TO 8
3C   54             51              LR    XFLG,A    SHOW THAT X HAS BEEN DETECTED
                           *
3D   28  0  7E      52     CONT     PI    BYTE
40   17             53              ST              STORE THE BYTE
41   35             54              DS    CCNT                AND DECREMENT BYTE COUNT
42   94  FA         55              BNZ   CONT
                           *
                           ***                NOW TIME FOR MAKING CKH SUM CHK
44   28  0  9D      56              PI    CHAR      GET CHK CHAR FROM TAPE
47   21  3F         57              NI    H'3F'     MASK TO SIX BITS
49   24  D0         58              AI    H'D0'     ASCII CONVERT-- FIRST 0-9
4B   82  3          59              BC    *+4       CARRY IF IT WAS TWEEN A AND F
4D   24  39         60              AI    H'39'     FINISH CONVERSION OF A TO F
4F   18             61              COM
50   1F             62              INC
51   C7             63              AS    CKSM      MAKE THE COMPARISON
52   21  F          64              NI    H'0F'     MASK TO LSB 4 BITS
54   84  D3         65              BZ    IDLE      IF OK , LET'S GET SOME MORE
                           *
                           **********    CHK SUM ERROR HALT   ****************
56   A0             66              INS   0
57   15             67              SL    4         CHK FOR SER/PARALLEL
58   81  7          68              BP    SLF2      BIT IS NOT SET IF PARALLEL
5A   20  93         69              LI    H'93'     TURN READER OFF
5C   51             70              LR    CHRS,A
5D   28  0  DD      71              PI    TTYO                AND TYPE IT
60   90  FF         72     SLF2     BR    SLF2
```

```
LOC   OBJECT CODE       STMT    SOURCE STATEMENT

                                *
                                *
 62   28   0  7E          73    SETA   PI     BYTE      GET NEW LOAD ADDRESS FROM TAPE
 65   24   8              74           AI     H'08'     ADD IN OFFSET TO GET TO RAM
 67   5A                  75           LR     10,A
 68   28   0  7E          76           PI     BYTE
 6B   5B                  77           LR     11,A
 6C   10                  78           LR     DC,H      SET THE ADDRESS INTO DC
 6D   90  BA              79           BR     IDLE
                                *
                                *
 6F   F4                  80    ENDX   NS     XFLG      HAVE AN *, BUT DOES IT FOLLOW ANY DATA ?
 70   84  67              81           BZ     IDLE      MUST BE ONLY AT THE HEAD OF THE TAPE
                                *                       WAS 2A+0  OR 2A+8
                                *********   HALT LOOP FOR WHEN FINISHED  ******
 72   A0                  82           INS    0
 73   15                  83           SL     4
 74   81   7              84           BP     SLF3      MAKE TEST FOR SERIAL OR PARALLEL
 76   20  93              85           LI     H'93'     SKIP OVER IF PARALLEL
 78   51                  86           LR     CHRS,A    READER OFF CHARACTER
 79   28   0  DD          87           PI     TTYO          PASS IT IN CHRS
 7C   90  FF              88    SLF3   BR     SLF3
                                *
                                *
                                *
                                *
                                *****   GET A BYTE   *********
                                **          GETS THE BYTE, CONVERTS, AND ADDS TO CHK SUM
                                *
 7E   8                   89    BYTE   LR     K,P       SAVE PC1
 7F   72                  90           LIS    2
 80   56                  91           LR     HFLG,A    SET THE HALF FLAG
 81   28   0  9D          92    AGAN   PI     CHAR
 84   21  3F              93           NI     H'3F'     MASK TO 6 BITS
 86   24  D0              94           AI     H'D0'     ASCII CONVERT-- FIRST 0-9
 88   82   3              95           BC     *+4
 8A   24  39              96           AI     H'39'     NEXT CLEAN UP A-F
```

```
LOC  OBJECT CODE     STMT   SOURCE STATEMENT
 8C   15               97          SL     4
 8D   51               98          LR     CHRS,A     TEMP STORE IN REG CHRS
 8E   14               99          SR     4
 8F   C7              100          AS     CKSM       ADD NEW CHAR TO CHK SUM
 90   57              101          LR     CKSM,A
 91   36              102          DS     HFLG       DECREMENT HALF COUNT
 92   84   5          103          BZ     TWOD       HAVE BOTH HALVES
                          *
 94   41              104          LR     A,CHRS     ONLY HAVE UPPER HALF-- MUST GO BACK
 95   53              105          LR     TEMP,A        BUT FIRST SAVE THIS HALF AWAY
 96   90  EA          106          BR     AGAN
                          *
 98   41              107   TWOD   LR     A,CHRS     GET THE LAST CHAR READ
 99   14              108          SR     4             AND SHIFT IT TO LSB END
 9A   C3              109          AS     TEMP       ADD IN MSB HALF BYTE
 9B    9              110          LR     P,K        RESTORE PC1
 9C   1C              111          POP
                          *
```

```
                                      *
                                      *
                                      ********     PARALLEL AND SERIAL INPUT ROUTINES     ********
                                      **
                                      **      COMMON CALL IS A PUSH TO CHAR
                                      **      CHAR USES Q TO JUMP TO APPROPRIATE ROUTINE
                                      **
                                      **
   9D     D                    113    CHAR      LR        P0,Q       JUMP TO INPUT ROUTINE INDIRECTLY THRU Q RE*
                                      *
                                      *
                                      *
                                      **   PINP:  GET A CHARACTER FROM PORT 4
                                      **
                                      **    TYPICALLY USED WITH TAPE READER, BUT HAS A HANDSHAKING
                                      **    DISCIPLINE THAT IS APPLICABLE TO OTHER DEVICES SUCH AS FIFOS
                                      **  , LOOKS FOR A CHARACTER READY INPUT, AND THEN GETS THE
                                      **    CHARACTER. NEXT CPU GIVES AN ADVANCE PULSE THAT IS REMOVED
                                      **    AFTER THE DEVICE READY INPUT GOES NOT READY.
                                      **
                                      **
   9E     A9                   114    PINP      INS       9          GET A CHAR FROM 300CPS READER
   9F     91  FE               115              BM        PINP       LOOK FOR SPOCKET= HIGH
   A1     7F                   116              LIS       H'OF'      100 US DELAY AFTER SEEING SPOCKET
   A2     18                   117              COM
   A3     1F                   118    POLY      INC
   A4     94  FE               119              BNZ       POLY
                                      *
   A6     A8                   120              INS       8                    AND NOW GET THE DATA BYTE
```

```
   LOC  OBJECT CODE      STMT   SOURCE STATEMENT
    A7   18              121            COM
    A8   51              122            LR      CHRS,A    TEMP STORE CNEW CHAR
                                *
    A9   71              123            LIS     H'01'     LET'S ADVANCE THE READER TO NEXT CHAR
    AA   B9              124            OUTS    9
    AB   A9              125    NOSP    INS     9         GET READER STATUS
    AC   81  FE          126            BP      NOSP          AND LOOK FOR MOVING OFF SPOCKET
    AE   70              127            LIS     0
    AF   B9              128            OUTS    9         REMOVE DRIVE PULSE NOW THAT IT IS MOVING
                                *
    B0   41              129            LR      A,CHRS    PICK BACK UP THE NEW CHARACTER
    B1   1C              130            POP
                                *
                                *
                                *
                                *** TTYI:  SERIAL INPUT      ...CHARACTER RETURNED IN ACC AND REGO
                                *
                                **  REG BCNT HOLDS BIT COUNT . REG CHRS HOLDS CHARACTER
    B2   79              131    TTYI    LIS     9
    B3   52              132            LR      BCNT,A    SET BIT COUNT FOR 8 DATA BITS
    B4   A0              133    START   INS     0
    B5   91  FE          134            BM      START     LOOK FOR START BIT
    B7   40              135            LR      A,BAUD    GET DELAY COUNT
    B8   2B              136            NOP
    B9   90   1          137    DLY3    BR      *+2       SILLY BRANCH FOR DELAY
    BB   24   1          138            AI      H'01'
    BD   94  FB          139            BNZ     DLY3      THIS LOOP IS HALF AS MUCH DELAY
                                *
    BF   A0              140            INS     0         CHECK START BIT VALIDITY
    C0   91  F3          141            BM      START
    C2   21  80          142    LOOP    NI      H'80'     MASK TO GET INPUT BIT ONLY
    C4   C1              143            AS      CHRS      (LD'G START BIT WILL CLR GARBAGE)
    C5   32              144            DS      BCNT      DROP BIT CNT: 0 IF LAST DATA,NEG IF STOP
    C6   91  11          145            BM      STOP      NEG IF LOOKING FOR STOP BIT
    C8   84   2          146            BZ      LOP2      IF LAST BIT, DO NOT SHIFT
    CA   12              147    LOP3    SR      1         SHIFT ASSEMBLED CHARACTER TO MAKE ROOM
    CB   51              148    LOP2    LR      CHRS,A    STORE ASSEMBLED CHARACTER
                                *
    CC   40              149            LR      A,BAUD    START OF FULL BIT TIME DELAY
    CD   2B              150            NOP
```

```
LOC  OBJECT CODE     STMT   SOURCE STATEMENT

 CE   BB              151   DLY4   OUTS   H'0B'    NOP FOR DELAY
 CF   BB              152          OUTS   H'0B'
 D0   BB              153          OUTS   H'0B'
 D1   24  1           154          AI     H'01'    INCR WITH A 5US INST
 D3   94  FA          155          BNZ    DLY4
                            *
 D5   A0              156          INS    0        GET NEW BIT
 D6   90  EB          157          BR     LOOP
                            *
 D8   A0              158   STOP   INS    0        GET STOP BIT
 D9   41              159          LR     A,CHRS   GET CHAR IN ACC
 DA   81  EF          160          BP     LOP3     BAD STOP BIT IF BIT7 OF INPUT IS 0
                            *
 DC   1C              161          POP
                            *
```

14.17

```
                                   **
                                   **
                                   ******* SERIAL OUTPUT ROUTINE  ***********
                                   **
                                   **  HAS 1 START, 8 DATA, 2 STOP. USES PORT 0, BITS 0 AND 7
                                   **  BAUD RATE IS SET BY A DELAY COUNT IN REG BAUD
                                   **   CALL BY PUTTING CHAR IN REG CHRS, AND SETTING DELAY IN BAUD
                                   **   ROUTINE RETURNS WITH ALL 1'S IN REG CHRS, REG BAUD INTACT
                                   **
                                   **
   DD    7B                 163   TTYO     LIS      H'0B'
   DE    52                 164            LR       BCNT,A    SET BIT COUNT FOR 11 BITS
   DF    70                 165            LIS      0
   E0    B0                 166            OUTS     0         OUTPUT START BIT
                                   *
                                   ***   DELAY ROUTINE-- 3.3MS FOR 300 BAUD, 9 MS FOR 110 BAUD
                                   *
   E1    40                 167   DLY1     LR       A,BAUD    GET DELAY COUNT
   E2    2B                 168            NOP
   E3    BB                 169   DLY2     OUTS     H'0B'     NOP FOR DELAY
   E4    BB                 170            OUTS     H'0B'
   E5    BB                 171            OUTS     H'0B'
   E6    24   1             172            AI       H'01'     INCR WITH A 5US INST
   E8    94   FA            173            BNZ      DLY2
                                   *
   EA    32                 174            DS       BCNT      DECREMENT BIT COUNT
   EB    84   C             175            BZ       DONE
   ED    41                 176            LR       A,CHRS    GET CHARACTER
   EE    21   1             177            NI       H'01'     MASK OFF ALL BUT BIT 0
   F0    B0                 178            OUTS     0         OUTPUT THE NEW DATA BIT
                                   *
   F1    41                 179            LR       A,CHRS    NOW SHIFT THE CHARACTER FOR NEXT BIT
   F2    12                 180            SR       1
   F3    24   80            181            AI       H'80'     FILL WITH 1'S FOR STOP BITS
   F5    51                 182            LR       CHRS,A
```

14.18

```
03/05/75    13:46:06
                                                                    F8A -- VERSION  4D --  2/18/75
   LOC   OBJECT CODE        STMT    SOURCE STATEMENT                                      PAGE  10
    F6    90  EA             183                BR        DLY1       GO WAIT OUT THIS BIT
                                         *
    F8    1C                 184        DONE     PUP                  ALL FINISHED
                                         *
                                         *
                                         *
                             185                 END
PASS 2 COMPLETE      9.17 SEC
NO ERRORS IN ABOVE ASSEMBLY
```

14.19