STUPID

Steve Toner
(solo)
Fall 1978
Course 6.112 term project

Abstract

STUPID (Simple Twelve-bit Unadorned Programmed Instruction Decoder) is a twelve bit microprogrammed single address, single accumulator computer which does not resemble the PDP-8.

Introduction & Overview

The STUPID instruction set closely resembles that of the "Little Man Computer" of 15.564. Instructions have a 4-bit op code and 8-bit address. STUPID is a binary machine, while the Little Man Computer is a decimal machine. The STUPID instruction set is (all numbers in hexadecimal):

ØØØ - HALT

1XX - ADD: Add the contents of memory location XX to the AC

2XX - SUB: Subtract the contents of location XX from the AC

3XX - AND: Logical bit-wise and (XX) with AC

4xx - OR: Logical bit-wise or (XX) with AC

5XX - CLEAR: Clear memory location XX

6XX - ISZ: Increment XX and skip the next instruction if the result = Ø

7XX - DSZ: Decrement XX and skip if result=Ø

8XX - LOAD: Load the AC with the contents of XX

9XX - STORE - Deposit the AC in location XX

AXX - HANG: Hang the processor

BXX - BRANCH: Branch to location XX

CXX - BZ: Branch if (AC)=Ø

DXX - BN: Branch if (AC)<Ø

EXX - BP: Branch if (AC)>Ø

FØX - NOP*

F1X - GCHAR: Read a character from the TTY

F2X - PCHAR: Type the character in AC on the TTY

F3X - RSWR: Read contents of switch register into AC

F4X-FFX - NOP*

*These are "reserved for future expansion"

The address space of STUPID is only 8 bits and there is no indirect addressing.

There are 5 registers in the STUPID machine: AC, PC, MAR, MBR, MWR. There is also a switch register (SWR) and Teletype out (TTY) "register". AC is the accumulator, where all arithmetic and logical operations leave their result and through which all I/O is performed. PC is the program counter, which functions in the obvious way. MAR is the memory address register, which holds the memory address for all reads from and writes to memory. MBR is the memory buffer register, which receives the data which is read out of memory (and so is a "read only" register). MWR is the memory write register, which holds the data which is to be written into the location specified by the MAR. It is a "write only" register. A block diagram of the system is shown in figure 18. The MBR appears on both the A and B busses, as when we do an "ADD" we will read the value to be added to the AC into the MBR, and then add this to the AC. Therefore the MBR must be on the B bus. But it must also be on the A bus, because we need to be able to add or subtract one from it when doing an ISZ or DSZ. The 74181 ALU gives A plus 1 and A minus 1 directly, but not B plus 1 or B minus 1.

The STUPID control box is microprogrammed. The microcode instruction format is shown in figure 15. There are three types of operations: ALU operations, which move data between registers; branch operations, which control the microprogram

sequencing; and "OP"s, which include everything not covered by the ALU and branch operations. These (OP) operations include reading from memory, writing to memory, dispatching on op code and lighting the "magic mode" light.

The STUPID microcode

The microcode for STUPID is included in Appendix A. Operation is fairly straightforward. When the RESET button is pressed, the microprogram sequencer is cleared and the micro instruction register is set to zero. This assures that the microcode starts at location Ø (a micro instruction of Ø = br Ø) At this point the magic mode switch is checked. If it is set, STUPID runs some simple diagnostics. If one of these diagnostics fails, the machine will hang and an address will be displayed in the micro pc lights. Appendix B lists all possible hangs and possible causes for each. If the diagnostics run through without error, STUPID examines the location specified by the SWR and displays it in the AC lights. If the data in this location is to be changed, the new value can be set into the SWR at this point and START pressed. STUPID will deposit the SWR data into the location and increment the PC (In magic mode the PC always shows the current location - that location which will be modified if START is pressed). The new location is displayed and can be modified as before. At this point, if the magic mode switch is turned off, or if it is not on when RESET is pressed, STUPID waits for START to be pressed. When START is pressed, the SWR is loaded into the PC and instruction execution begins. The instruction is fetched, and STUPID dispatches on the op code to the routine which does the particular instruction. That's all there is to the microcode. The only instructions which are not entirely straightforward are ISZ, DSZ and the instructions with

op code F. ISZ and DSZ must test to see if the result of the increment or decrement is zero. Since the MWR is a write only register, it cannot be tested for zero without a lot of special logic. Therefore, ISZ and DSZ do a simultaneous write and read when writing the result back. This puts the result in the MBR which can be tested. But the =Ø signal comes from a flip flop which is set only on ALU operations (see figure 2), so isn't it still valid after a write? Yes, but sometimes we don't see the obvious until it's too late. The =Ø condition started out as a general A=B condition, but only result=Ø was ever used, so it became =Ø (the original idea was to use the A=B output of the ALU, but as it turns out that signal is only valid if the right operation is specified, so that got the "deep six" as they say.) In the op F instructions, the second nibble indicates the particular instruction. It is necessary to dispatch on this value, so it must be put into the high order 4 bits of the MBR (where DISP gets its argument). This is accomplished by putting it in the AC and shifting left 4 bits. The original value of the AC is saved in location Ø and read back if the instruction does not put a new value in it (GCHAR, RSWR). Since the instruction bits must be in the MBR to do a dispatch, the shifted value is written into location FF and read back (simultaneously, to save time). Thus, locations ØØ and FF should not be used by a program. This awfulness could be alleviated somewhat by adding some temporary registers to the machine, but space limitations dictate that this not be done.

The A and B busses are implemented as multiplexors - A a 4 input mux and B a 2 input mux (see figs 11 & 12). Bits 1 and 2 of an ALU instruction specify which A input to select. Similarly, bits 3 and 4 specify the B input. Two bits here allow for easy expansion, and in fact B_3 is already defined to be the TTY in register. Space constraints limit B to be a 2-input multiplexor, however. Each ALU operation also specifies the carry and function in bits 5-1Ø. These are fed directly to the appropriate inputs on the ALU card (see figure 2 for ALU). This means that a carry in of 1 is specified by a Ø in bit 5, since the MSI 181 has inverted carry signals. The destination register is specified by bits 11-15. Each output register has a bit associated with it, though three bits could be used and these three bits decoded to one of 8. But a decoder is another card and ROM bits are free, provided we don't use more than 16 of them. If the machine were to be expanded to include an index register or other registers, the decoding scheme would have to be used. The destination bit is NANDed with $(BIT\emptyset \cdot \overline{\phi_2})$ to give the clock input to the MSI 161, which is used for all these registers (except TTY). (see figs 3, 4 & 1Ø) BITØ tells it that this is an ALU operation, so that bit which looks like a destination bit really is. The two phase clock (see figure 5) is used because the 74161 counter uses master-slave flip flops, and the load input cannot change when the clock line to these is low. Therefore, the micro sequence counter is incremented/loaded on the leading edge of ϕ_2 (see figure 14 for ϕ_1, ϕ_2 timing) and the ROM outputs (which are what cause the load line to change) are

clocked into a register on the leading edge of ϕ_1. This assures that the sequencer's load line changes only when its clock line is high. Since the data at the ALU outputs is valid sometime after ϕ_1 until the next rising edge of ϕ_1, $\overline{\phi_2}$ is as good as anything to clock the registers. I am not convinced that the 2-phase clock is necessary, but since the LSI 1702 outputs can drive only 1 load, there has to be something buffering them. The two-phase clock is safe (like it works, so don't knock it). The question is whether the output of the 1702 glitches (can BIT0 go 0->1->0, for example?) when a new address is selected. I don't know, so I assume the worst.

Branches are implemented by putting the inverted value of the condition on the load line of the micro sequencer and the branch address on the data inputs. On the next rising edge of ϕ_2, the counter will be loaded with the address if the condition was true. Simple. The condition bits (BIT2-4) address a multiplexor and if BIT0=BIT1=0, the condition (inverted) is let through (see figure 1). Otherwise (no branch), the load line is held high so the counter will incremented.

OPs are decoded by detecting BIT0=0, BIT1=1 (actually BIT0=0 and not BR). This signal is ANDed with the appropriate bit to decode the instructions. This causes a timing problem on a WRITE (you should now be looking at figure 6). Since OP must go through 2 NOR gates, it is delayed from the write signal, which comes directly from BIT15. Thus, if a READ is followed by an instruction which happens to have BIT15 set, BIT15 comes on, OP

is delayed by a couple of gate delays and w goes low, which makes the memory think it supposed to do a write. To fix this problem, gate delays are put on BIT15 (the *'d guys in figure 6) so that it cannot change until after OP has changed. Crude but effective.

A dispatch also causes a branch, so causes the load input of the sequencer to go low. However, this time the address comes only partly from the instruction. The low order 4 bits are specified by the high order 4 bits of the MBR. Only the high order 4 bits of the address are specified by the dispatch instruction.

The Memory

The basic memory box is the same one used in the memory dumper/loader (saves plugging wires if I just use what's already there - there's no point in reinventing the world when you've got something that works...). It is shown in figure 9. Each memory operation (read or write 4 bits) takes 7 clock cycles (memory timing shown in figure 13). If the mwrite line is high, the $\overline{r}/w$ line will go high during states write1-write4. Otherwise (on a read) $\overline{r}/w$ is held low. The memory is controlled by a shift register in which a single 1 is shifted right 1 bit on each clock pulse. 574 flip flops were used to construct this shift register, as they can be cleared or set asynchronously, so resetting the register to the idle state can be accomplished painlessly. The register is set to 100000 (leftmost bit corresponds to leftmost FF in figure 9) whenever the RESET button is pressed (as

will hopefully be done on power-up, when the states of the flip flops are unknown). Since the 574 outputs change from L->H faster than H->L, the $\overline{r}$/w line cannot glitch, which might otherwise cause problems when writing. To read a word (4 bits) from the memory, an address is put on the addr lines and mread is brought high. When resetr/w transitions from H->L, the data on the read data lines is valid. resetr/w is normally used to clock the read data into a register. A write is done by setting up the address and the data to be written and raising mwrite. mread and mwrite must stay high at least until the shift register leaves the idle state. The rising edge of resetr/w may be used to clear mread and mwrite.

STUPID words are 12 bits long, which means that we must read or write 3 4-bit words for each STUPID word (this is necessary because of the restriction of 1 LSI 21Ø2 per person). The memory controller shown in figure 6 accomplishes this monumental task. When a read instruction is executed, $\overline{r}$ goes low. This clears the A and B flip flops and sets the mread FF. The r signal is NANDed with $\overline{\text{memdone}}$ to assure that the clock signal changes after $\overline{\text{memdone}}$ goes high - otherwise the clock transition would be lost. Setting mread causes the memory to read a single 4-bit word as described above. resetr/w is used to clock the A and B flip flops, which count in the sequence ØØ, 1Ø, 11, Ø1 (AB). On the ØØ->1Ø, 1Ø->11 and 11->Ø1 transitions, the read data is latched into the MBR (see figure 7). This means that the read data must stay valid until it can be latched. The LSI 17Ø2 data is valid

for about 100 nsec after the address change, which is plenty of time. In state 01, memdone is asserted and this clears mread. Q_A and Q_B are used as the low order 2 bits of the memory address. A write is similar to a read in the way it clears the A and B FFs and sets mwrite. This time, however, Q_A and Q_B are used to select which nibble of the MWR to apply to the write data lines (see figure 8).

ALU

The ALU is a standard 74181 and the microcode specifies the carry in, M and S_0-S_3, so any of the ALU functions may be used. Table 1 in figure 16 shows the possible ALU functions.

Rack Layout

The rack layout is shown in figure 99. It didn't fit in a single rack. Figure 99a shows the various lights and switches used...

Summary and conclusions

Well, it works. A demonstration program (which uses all the STUPID instructions except the NOPs, GCHAR and HANG) is included in Appendix C. Read the comments for documentation. At this point, many possible improvements and modifications cry out to be considered. STUPID has a teletype for output, but only a switch register for input. How about adding a TTY in? Originally it appeared that this would require extending the B bus by using 4 input multiplexors instead of 2. However, looking at it now (that it's too late), I see that the zero input on the B bus is not used. The ALU has functions A, A plus 1, etc, which do not require B to be zero, and the A=B instruction was changed to =Ø. So the TTy in register could replace the zero "register" on the B bus. Once this was done, the microcode could be changed to allow memory to be loaded from a paper tape in the TTY reader. that would make life much easier for the poor soul who has to load a program into memory. Since A=B was changed to =Ø, the microcode could also be changed so that it doesn't do unnecessary comparisons with zero. Or, since no register has to be specified to receive the result of an operation, the zero comparison could be done by complementing the register which is being tested, specifying no destination, and using the A=B outputs of the ALU instead of the NOR gates that are there now. One way saves time, the other hardware. Other possible changes include changing the instruction set to allow indirect addressing (no change in the hardware is necessary to allow single level indirection - just

some instructions would have to be thrown out) and, if somebody were to decide that he actually wanted to use this (STUPID) machine for something, he might want more memory and more than one I/O device. There are 4 bits which are unused in the GCHAR and PCHAR instructions - these could be turned into more general DATA IN and DATA OUT instructions with the low order 4 bits specifying the device. More condition branching might have to be added to the micro machine to allow this. Finally, what about the HANG instruction and all those NOPs? Some intelligent person ought to be able to come up with something to do with them. Right now they're not very useful.

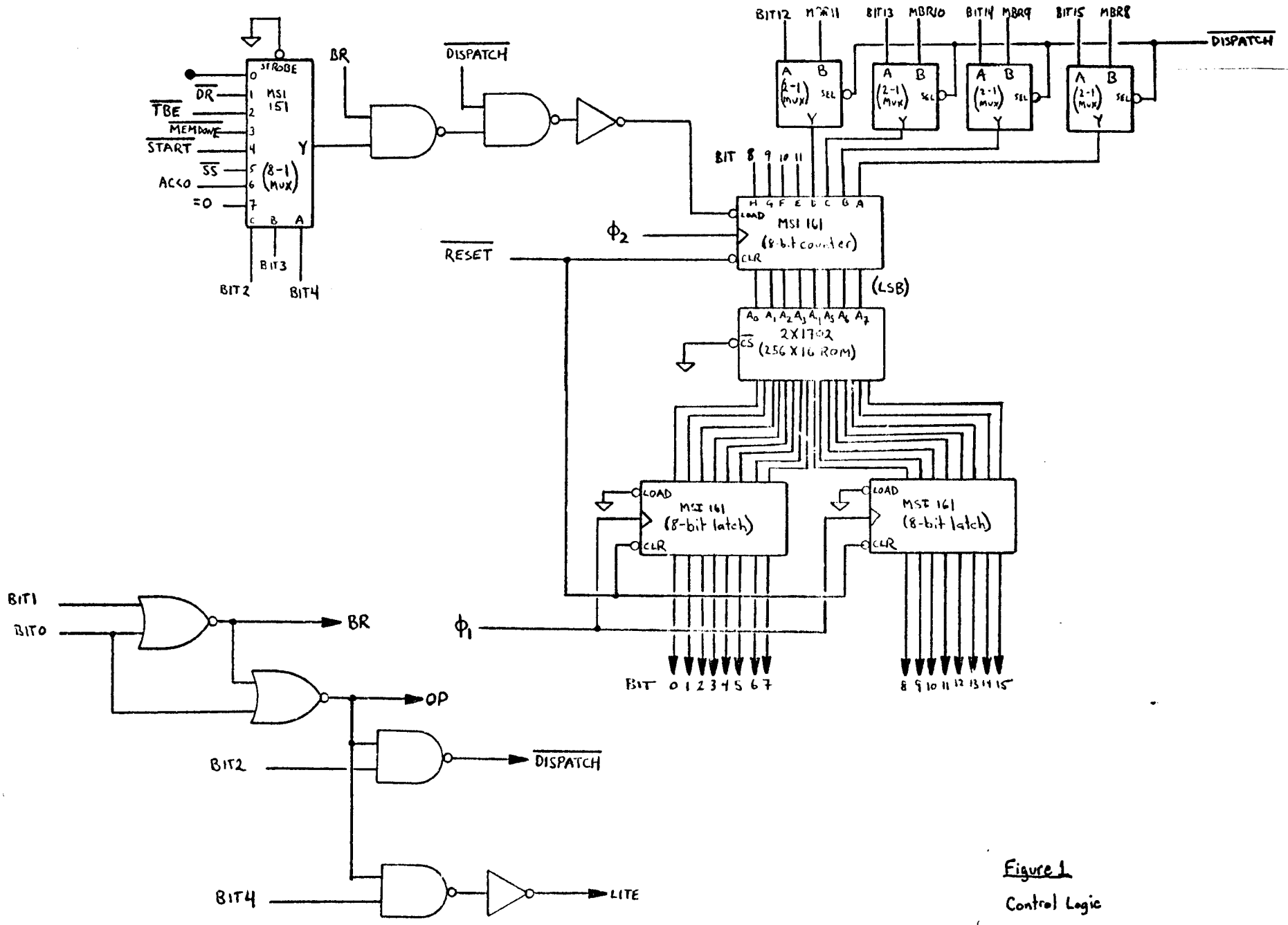


Figure 1

Control Logic

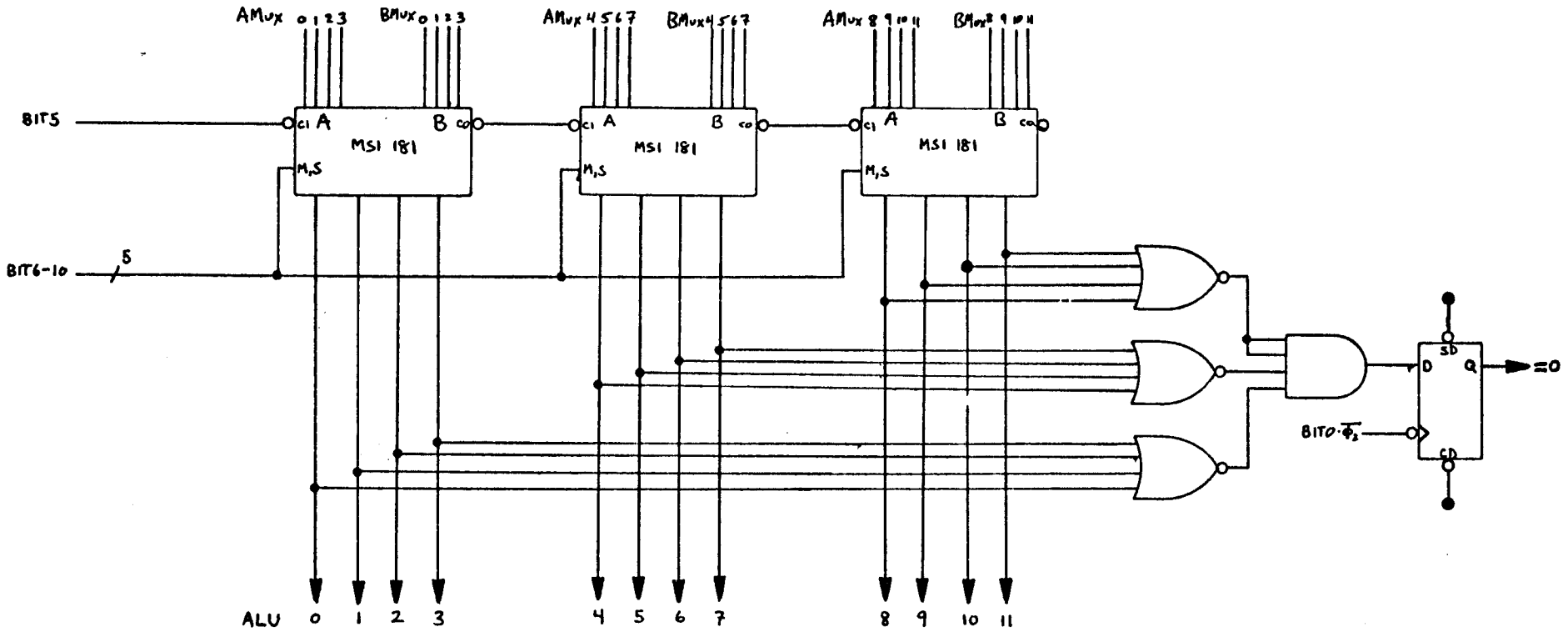

AMux 0 1 2 3
BMux 0 1 2 3
AMux 4 5 6 7
BMux 4 5 6 7
AMux 8 9 10 11
BMux 8 9 10 11
BIT5
BIT6-10
5
MSI 181
M,S
A
B
Cn
Cn+4
ALU 0 1 2 3 4 5 6 7 8 9 10 11
BIT0·Φ2
SD
D
Q
CD
=0

Figure 2
ALU

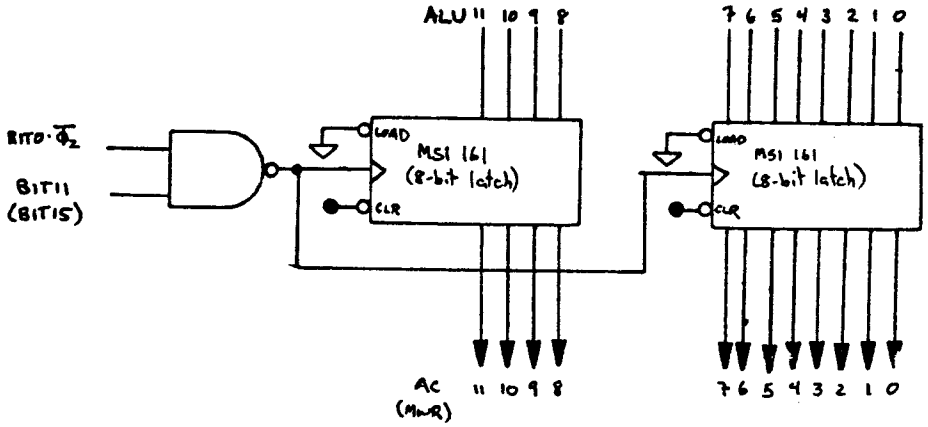


Figure 3
Accumulator, MWR

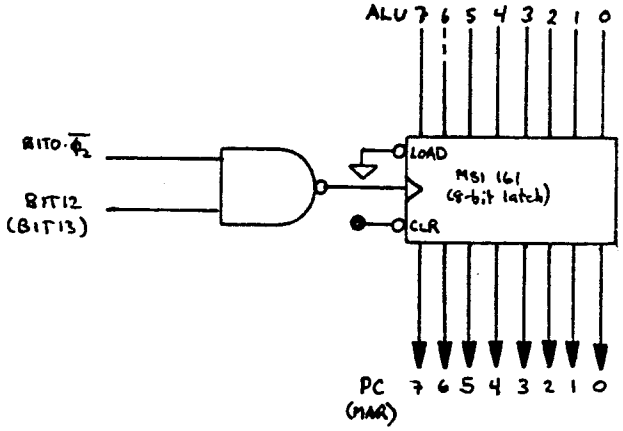


Figure 4
Program Counter, MAR

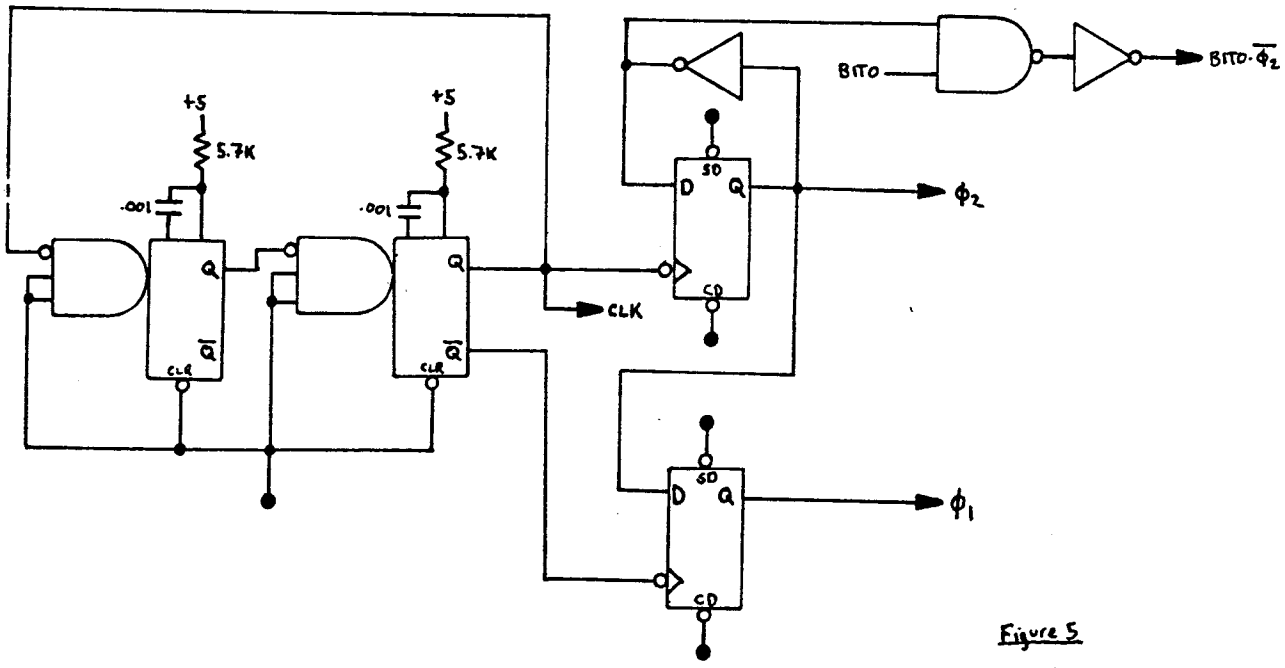
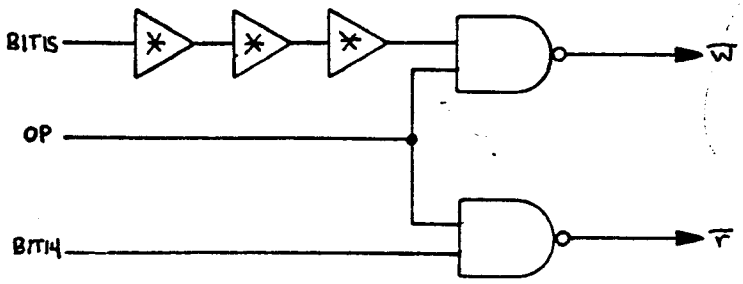
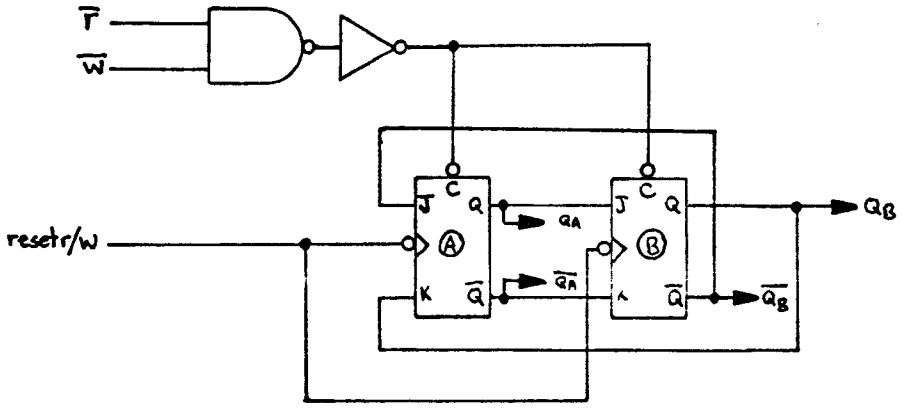


Figure 5
Clock





COUNT SEQUENCE

A	B
0	0
1	0
1	1
0	1

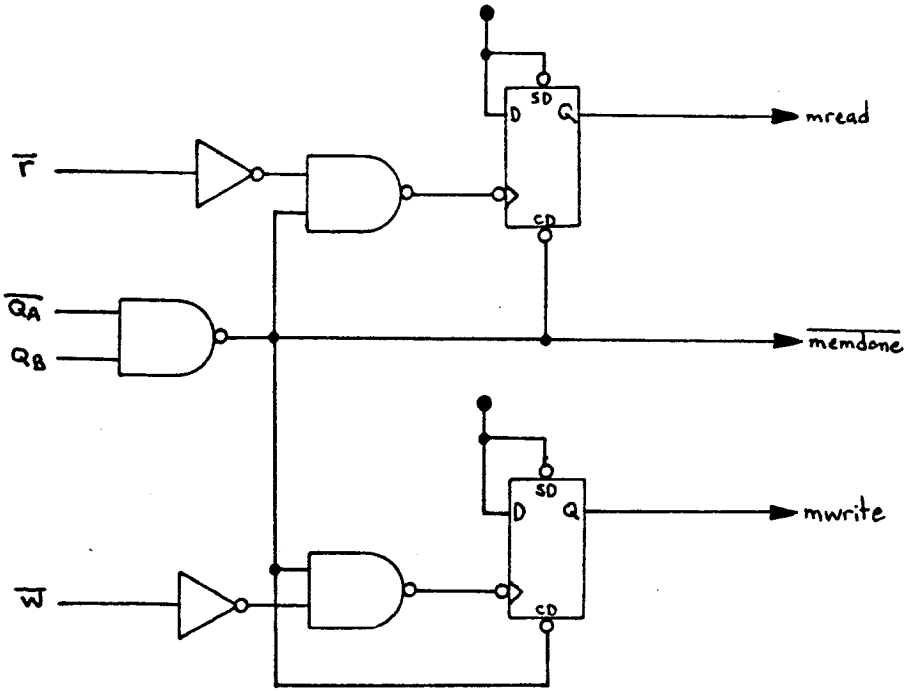


Figure 6
Memory Control

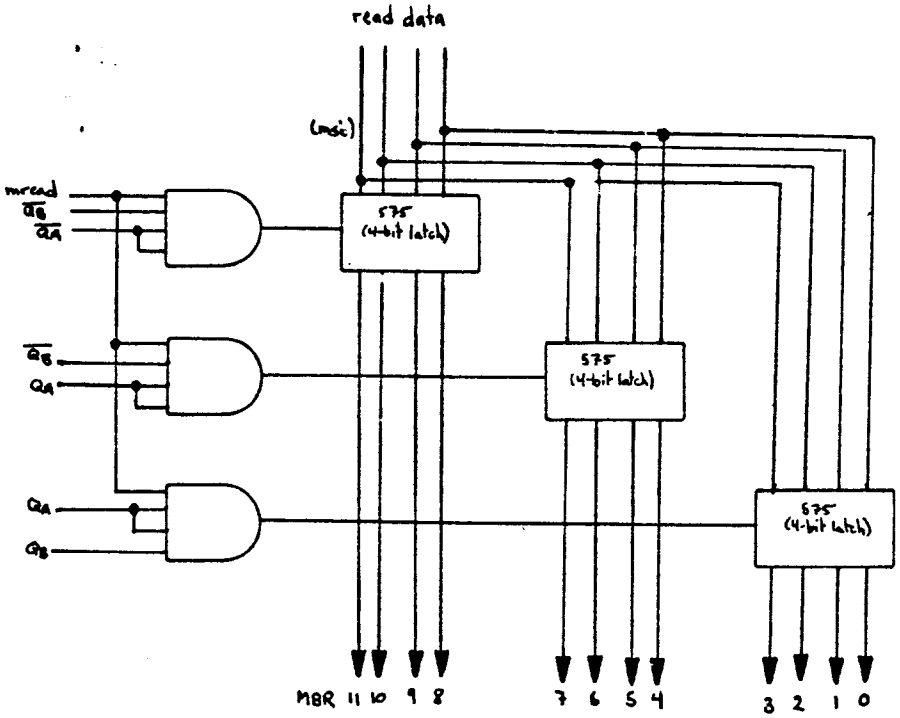


Figure 7

Memory Buffer Register

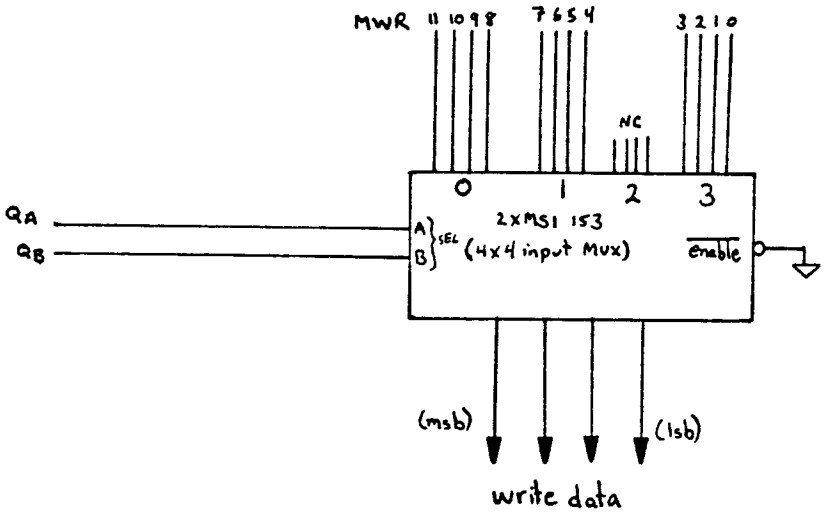


Figure 8

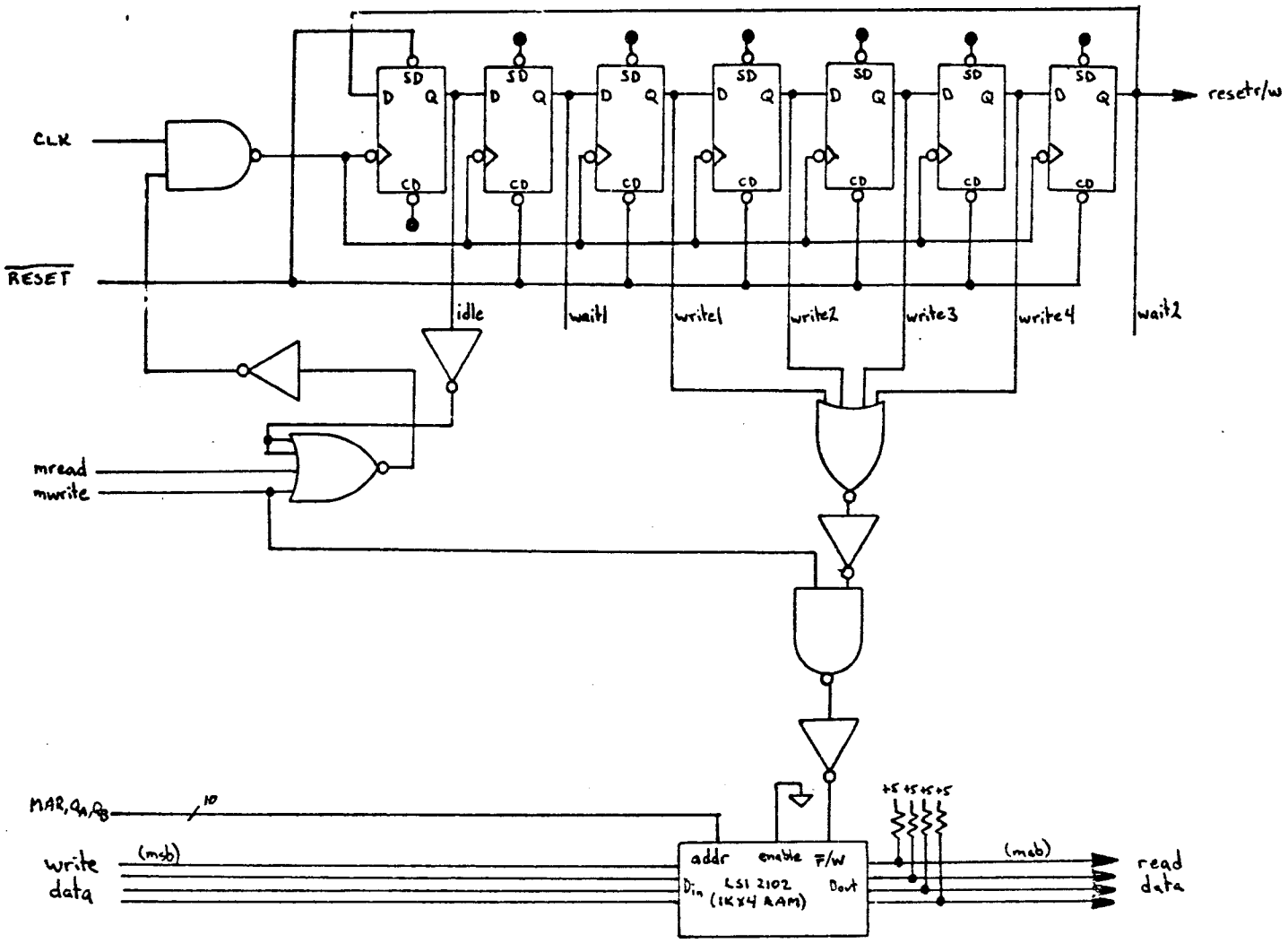


Figure 9

Memory

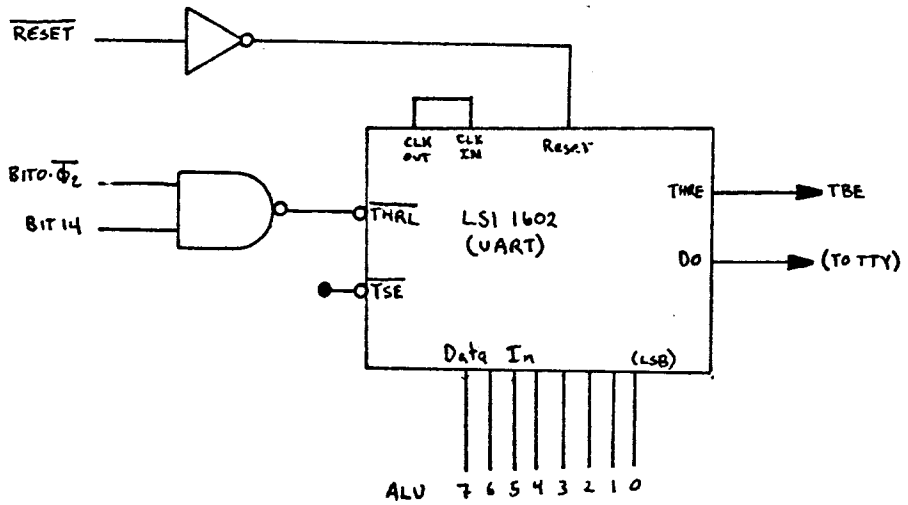


Figure 10

TTY

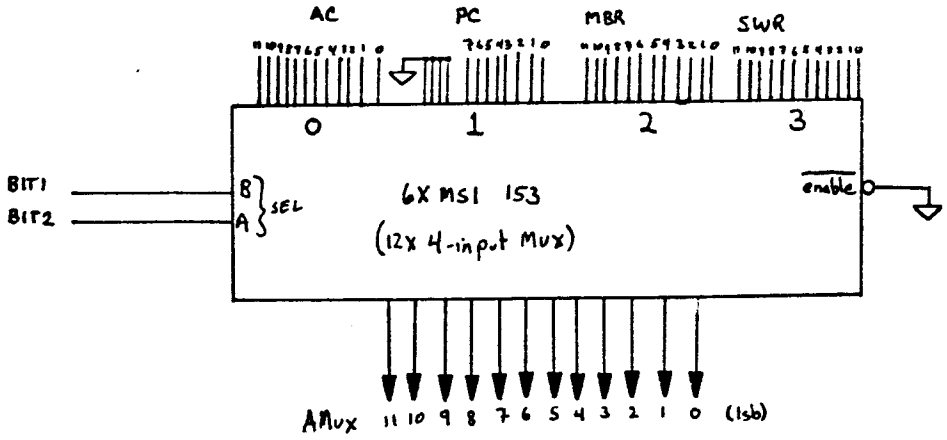


Figure 11

A Multiplexer

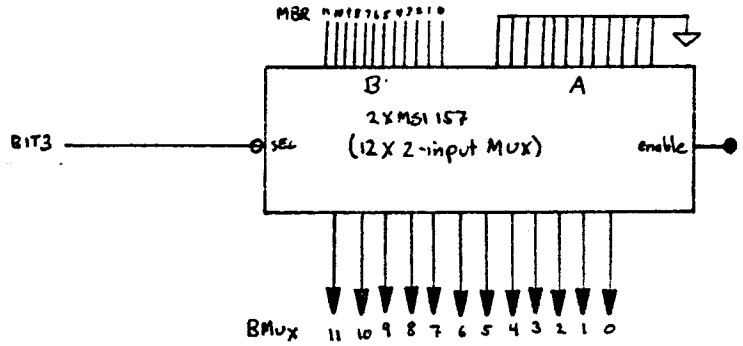
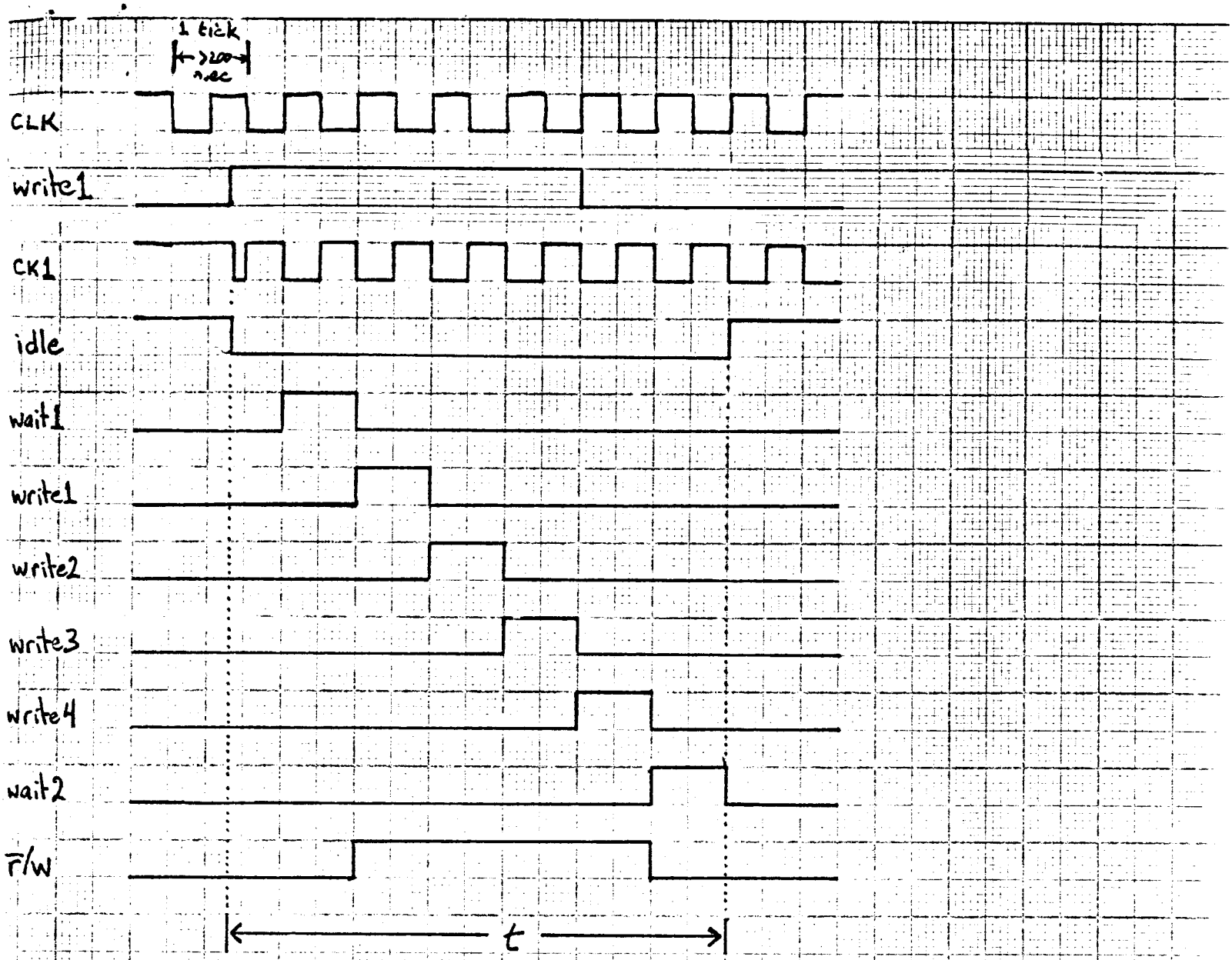


Figure 12

B Multiplexer



7 ticks < t < 8 ticks

Figure 13

Memory write cycle timing

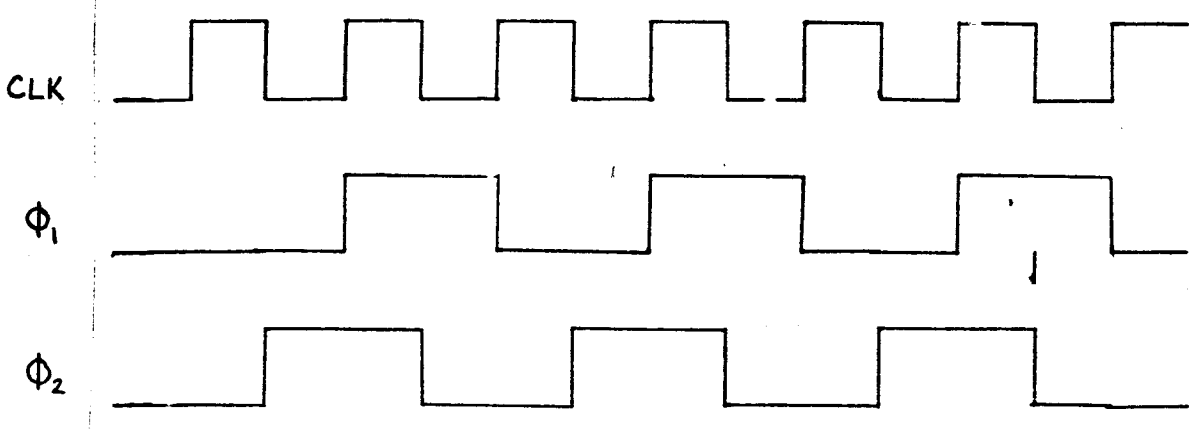
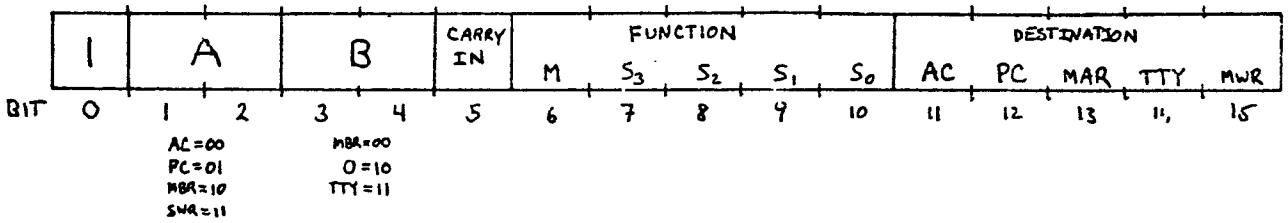
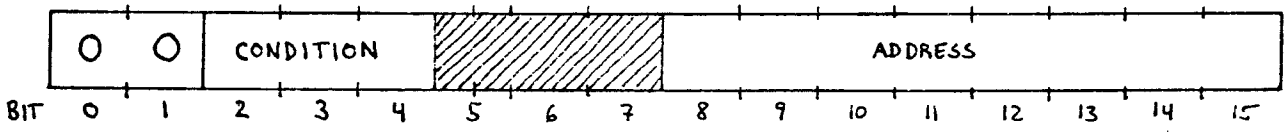


Figure 14
Φ_1, Φ_2 Timing

ALU instructions:

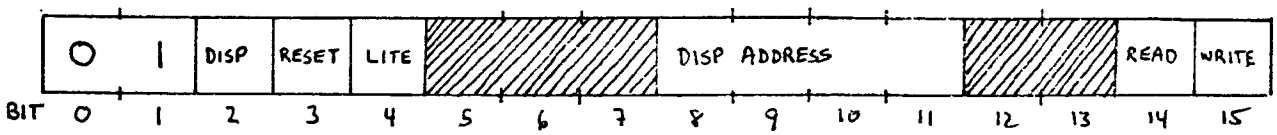


BRANCH instructions:



000 = TRUE - Always branch
001 = $\overline{DR}$ - Branch if no TTY character received
010 = $\overline{TBE}$ - Branch if TTY output buffer not empty
011 = $\overline{MEMDONE}$ - Branch if memory read/write not finished
100 = $\overline{START}$ - Branch if START button not pressed
101 = $\overline{SS}$ - Branch if Magic Mode switch not set
110 = AC<0 - Branch if (AC)<0 (AC11 =1)
111 = =0 - Branch if last ALU operation produced a zero result

OP instructions:



= unused, "reserved for future use"

Figure 15
Microcode Format

description (continued)

ALU Signal Designations

The '181, 'LS181, and 'S181 can be used with the signal designations of either Figure 1 or Figure 2.

The logic functions and arithmetic operations obtained with signal designations as in Figure 1 are given in Table 1; those obtained with the signal designations of Figure 2 are given in Table 2.

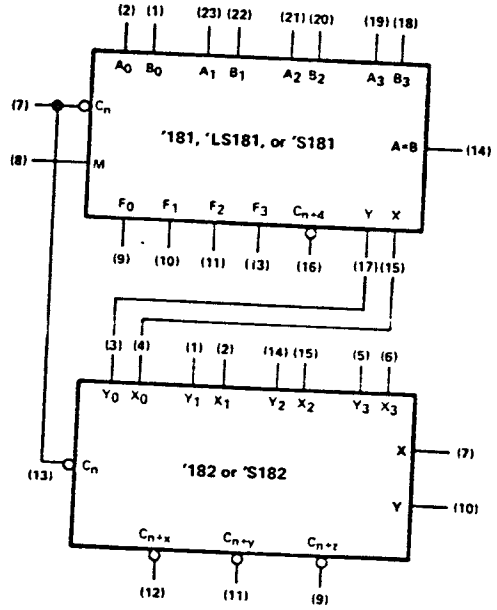


FIGURE 1
(FOR TABLE 1)

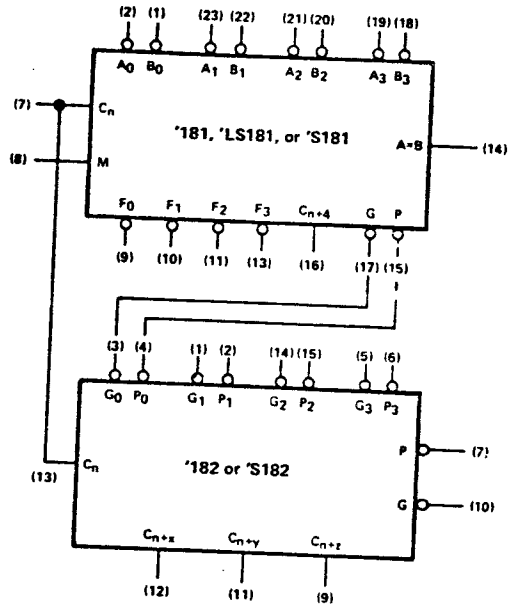


FIGURE 2
(FOR TABLE 2)

TABLE 1

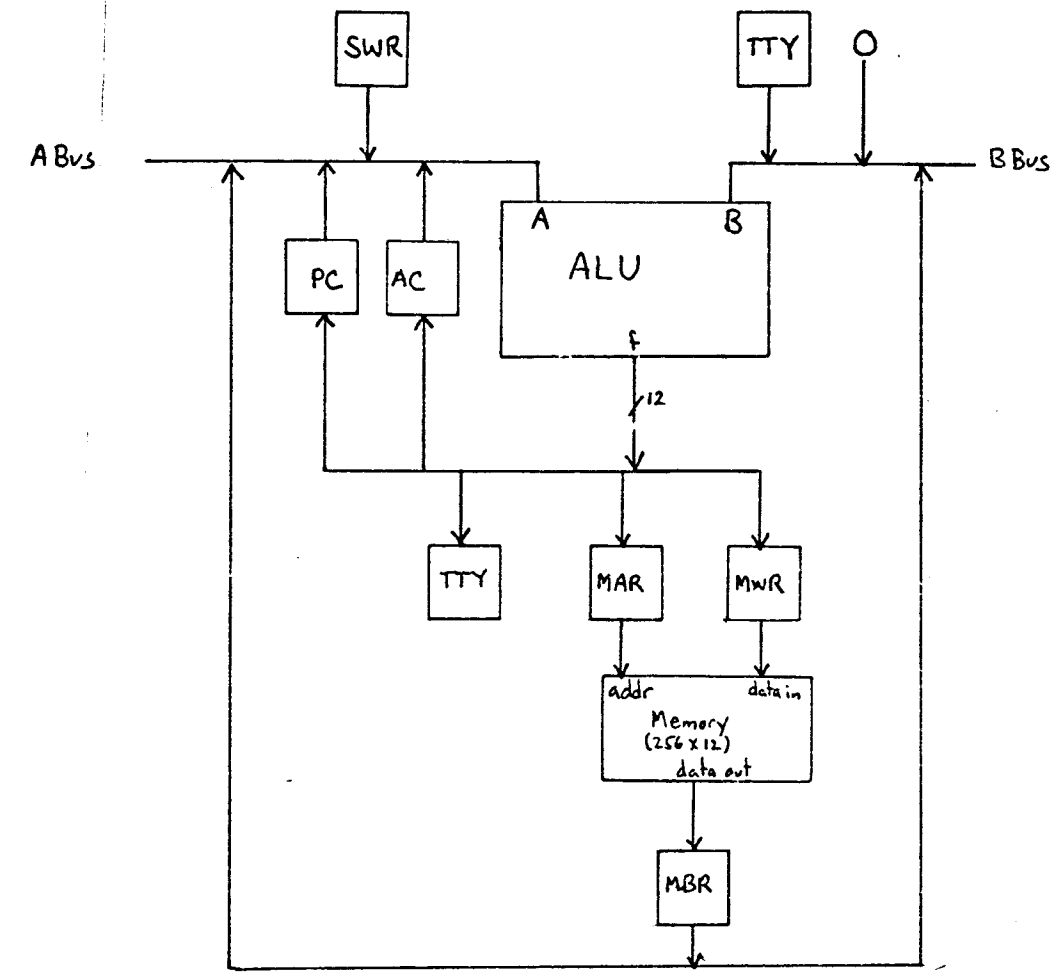
SELECTION	M = H	ACTIVE HIGH DATA	
		M = L: ARITHMETIC OPERATIONS	
S3 S2 S1 S0	LOGIC FUNCTIONS	C_n = H (no carry)	C_n = L (with carry)
L L L L	F = $\overline{A}$	F = A	F = A PLUS 1 ●
L L L H	F = $\overline{A+B}$	F = A + B	F = (A + B) PLUS 1
L L H L	F = $\overline{A}$B	F = A + $\overline{B}$	F = (A + $\overline{B}$) PLUS 1
L L H H	F = 0	F = MINUS 1 (2's COMPL)	F = ZERO ●
L H L L	F = $\overline{AB}$	F = A PLUS A$\overline{B}$	F = A PLUS A$\overline{B}$ PLUS 1
L H L H	F = $\overline{B}$	F = (A + B) PLUS A$\overline{B}$	F = (A + B) PLUS A$\overline{B}$ PLUS 1
L H H L	F = A $\overline{\odot}$ B	F = A MINUS B MINUS 1	F = A MINUS B ●
L H H H	F = A$\overline{B}$	F = A$\overline{B}$ MINUS 1	F = A$\overline{B}$
H L L L	F = $\overline{A}$ + B	F = A PLUS AB	F = A PLUS AB PLUS 1
H L L H	F = A $\overline{\odot}$ B	F = A PLUS B ●	F = A PLUS B PLUS 1
H L H L	F = B	F = (A + $\overline{B}$) PLUS AB	F = (A + $\overline{B}$) PLUS AB PLUS 1
H L H H	F = AB ●	F = AB MINUS 1	F = AB
H H L L	F = 1	F = A PLUS A ●	F = A PLUS A PLUS 1
H H L H	F = A + $\overline{B}$	F = (A + B) PLUS A	F = (A + B) PLUS A PLUS 1
H H H L	F = A + B ●	F = (A + $\overline{B}$) PLUS A	F = (A + $\overline{B}$) PLUS A PLUS 1
H H H H	F = A ●	F = A MINUS 1 ●	F = A

*Each bit is shifted to the next more significant position.

TABLE 2

SELECTION	M = H	ACTIVE LOW DATA	
		M = L: ARITHMETIC OPERATIONS	
S3 S2 S1 S0	LOGIC FUNCTIONS	C_n = L (no carry)	C_n = H (with carry)
L L L L	F = $\overline{A}$	F = A MINUS 1	F = A
L L L H	F = $\overline{AB}$	F = AB MINUS 1	F = AB
L L H L	F = $\overline{A}$ + B	F = A$\overline{B}$ MINUS 1	F = A$\overline{B}$
L L H H	F = 1	F = MINUS 1 (2's COMP)	F = ZERO
L H L L	F = $\overline{A+B}$	F = A PLUS (A + $\overline{B}$)	F = A PLUS (A + $\overline{B}$) PLUS 1
L H L H	F = $\overline{B}$	F = AB PLUS (A + $\overline{B}$)	F = AB PLUS (A + $\overline{B}$) PLUS 1
L H H L	F = A $\overline{\odot}$ B	F = A MINUS B MINUS 1	F = A MINUS B
L H H H	F = A + $\overline{B}$	F = A + $\overline{B}$	F = (A + $\overline{B}$) PLUS 1
H L L L	F = $\overline{A}$B	F = A PLUS (A + B)	F = A PLUS (A + B) PLUS 1
H L L H	F = A $\overline{\odot}$ B	F = A PLUS B	F = A PLUS B PLUS 1
H L H L	F = B	F = A$\overline{B}$ PLUS (A + B)	F = A$\overline{B}$ PLUS (A + B) PLUS 1
H L H H	F = A + B	F = A + B	F = (A + B) PLUS 1
H H L L	F = 0	F = A PLUS A*	F = A PLUS A PLUS 1
H H L H	F = A$\overline{B}$	F = AB PLUS A	F = AB PLUS A PLUS 1
H H H L	F = AB	F = A$\overline{B}$ PLUS A	F = A$\overline{B}$ PLUS A PLUS 1
H H H H	F = A	F = A	F = A PLUS 1

Figure 16



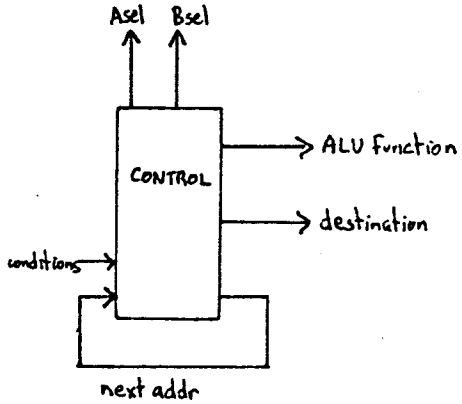


Figure 18
STUPID block diagram

DIGITAL SYSTEMS LAB

BACKPLANE LAYOUT

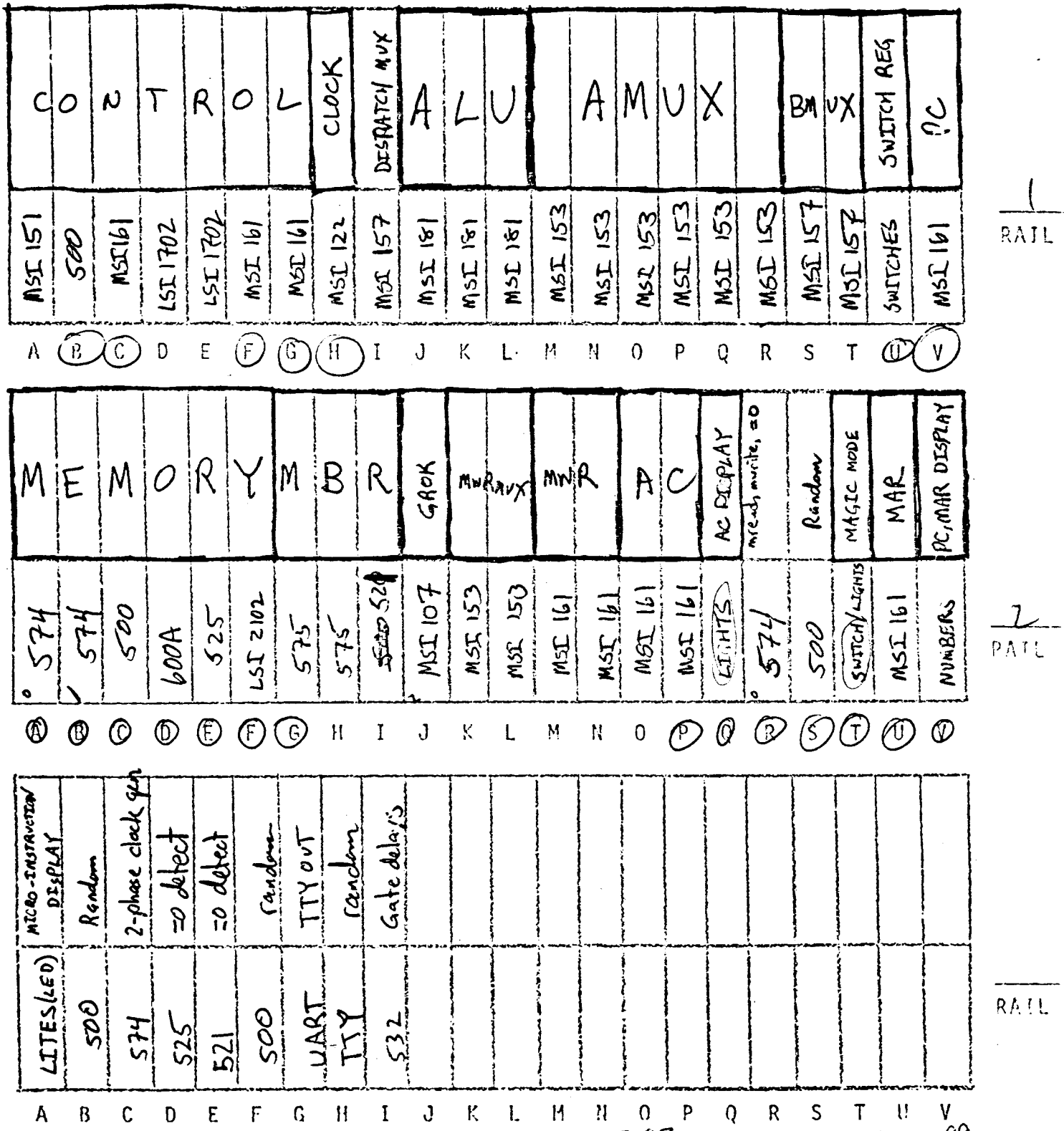


Figure 99

7397 7179

7397
647
6750
629

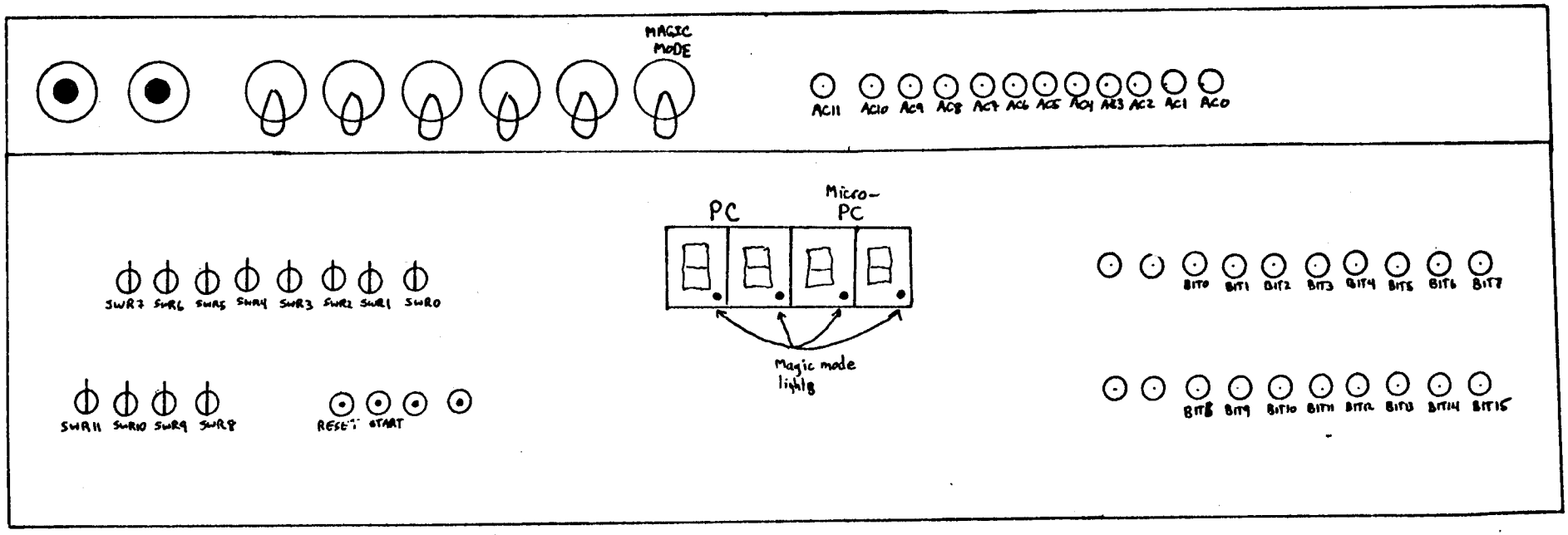


Figure 99a
STUPID display

```
; STUPID microcode

00 2802  gnorf:  bnss .+2          ; Magic mode?
01 0090          br test           ;   Yes! Go load memory
02 2002  wait:   bnstart .         ;   No. Wait for START
03 F7E8          pc<-swr           ; Start at addr in switches
04 0009          br inst

         ; Get next instruction and execute it:
05 2809  next:   bnss inst         ; Single stepping?
06 2008          bnstart .+2       ;   Yes - make sure START is off
07 0006          br .-1            ; It wasn't
08 2008          bnstart .         ; Now wait for him to press it
09 B7C4  inst:   mar<-pc           ; Get the instruction
0A 4002          read
0B B008          pc<-pc+1          ; Get ready for next
0C 180C          bmnr    .         ; Now wait for the bloody thing
0D 6010          disp 10           ; Dispatch on op code
0E 000E          br .
0F 000F          br .              ; Bad news

10 00E0          br halt           ; OP 0 = HALT
11 0027          br add            ; OP 1 = ADD mem to AC
12 002C          br sub            ; OP 2 = SUBtract mem from AC
13 0031          br and            ; OP 3 = AND mem with AC
14 0036          br or             ; OP 4 = OR mem with AC
15 003B          br clr            ; OP 5 = CLeaR mem
16 0040          br isz            ; OP 6 = Inc mem, Skip if result=0
17 004B          br dsz            ; OP 7 = Dec mem, skip if result=0
18 0056          br load           ; OP 8 = LOAD AC from mem
19 005B          br store          ; OP 9 = STORE AC in mem
1A 001A          br .              ; OP A = HANG
1B 0060          br branch         ; OP B = Branch to mem
1C 0062          br bz             ; OP C = Branch if (AC)=0
1D 0065          br bn             ; OP D = Branch if (AC)<0
1E 0067          br bp             ; OP E = Branch if (AC)>0
1F 97E1          mwr<-ac           ; Op F = Special functions (like I/O)
20 9664          mar<-0
21 4001          write             ; Save AC in location 0
22 D590          ac<-shift(mbr)    ; Shift left 4 bits
23 9590          ac<-shift(ac)     ;  to get the operation
24 9590          ac<-shift(ac)
25 9590          ac<-shift(ac)
26 006D          br patch          ; OOPS!

27 C7E4  add:    mar<-mbr
28 4002          read
29 1829          bmnr .
2A 8530          ac<-ac+mbr
2B 0005          br next

2C C7E4  sub:    mar<-mbr
2D 4002          read
```

```
2E 182E          bmnr .
2F 80D0          ac<-ac-mbr
30 0005          br next

31 C7E4  and:    mar<-mbr
32 4002          read
33 1833          bmnr .
34 8770          ac<-ac&mbr
35 0005          br next

36 C7E4  or:     mar<-mbr
37 4002          read
38 1838          bmnr .
39 87D0          ac<-ac!mbr
3A 0005          br next

3B C7E4  clr:    mar<-mbr
3C 8661          mwr<-0
3D 4001          write
3E 183E          bmnr .
3F 0005          br next

40 C7E4  isz:    mar<-mbr
41 4002          read
42 1842          bmnr .
43 D001          mwr<-mbr+1
44 4003          write,read
45 1845          bmnr .
46 D7C1          mwr<-mbr!0      ; Compare
47 3849          b=0 .+2         ; Result=0?
48 0005          br next         ;   Nope. skip the skip
49 B008          pc<-pc+1        ;   Yes - skip next instr
4A 0005          br next

4B C7E4  dsz:    mar<-mbr
4C 4002          read
4D 184D          bmnr .
4E D5E1          mwr<-mbr-1
4F 4003          write,read
50 1850          bmnr .
51 D7C1          mwr<-mbr!0
52 3854          b=0 .+2
53 0005          br next
54 B008          pc<-pc+1
55 0005          br next

56 C7E4  load:   mar<-mbr
57 4002          read
58 1858          bmnr .
59 D7F0          ac<-mbr
5A 0005          br next
```

```
5B C7E4  store:  mar<-mbr
5C 97E1          mwr<-ac
5D 4001          write
5E 185E          bmnr .
5F 0005          br next

60 D7E8  branch: pc<-mbr
61 0005          br next          ; Simplicity personified

62 97D0  bz:     ac<-ac!0
63 3860          b=0 branch
64 0005          br next

65 3060  bn:     bac<0 branch
66 0005          br next

67 3005  bp:     bac<0 next       ; If <0 it can't be positive
68 97D0          ac<-ac!0         ; If = 0 it's not either
69 3805          b=0 next
6A D7E8          pc<-mbr          ; It's positive!!
6B 0005          br next
6C 006C          br .
6D 186D  patch:  bmnr .
6E 00CA          br p1            ; Go figger out what instr is
6F 006F          br .

         ; Dispatch table for OP F instructions
70 0080          br getbak        ; F0X = NOP
71 0085          br gchar         ; F1X = Get char from TTY
72 0088          br pchar         ; F2X = Type (AC) on TTY
73 00C7          br rswr          ; F3X = Read Switch register
74 0080          br getbak        ; F4X = NOP
75 0080          br getbak        ; F5X = NOP
76 0080          br getbak        ; F6X = NOP
77 0080          br getbak        ; F7X = NOP
78 0080          br getbak        ; F8X = NOP
79 0080          br getbak        ; F9X = NOP
7A 0080          br getbak        ; FAX = NOP
7B 0080          br getbak        ; FBX = NOP
7C 0080          br getbak        ; FCX = NOP
7D 0080          br getbak        ; FDX = NOP
7E 0080          br getbak        ; FEX = NOP
7F 0080          br getbak        ; FFX = NOP

80 9664  getbak: mar<-0           ; Get AC back from loc 0
81 4002          read
82 1882          bmnr .
83 D7F0          ac<-mbr
84 0005          br next

85 0885  gchar:  bnchr .          ; Char in buffer?
86 9F50          ac<-tty          ;   Yes. Read it in
87 0005          br next
```

```
88 9664  pchar:  mar<-Ø          ; Get the char back
89 4ØØ2          read
8A 188A          bmnr .
8B D7FØ          ac<-mbr
8C 1ØBC          bnobe .         ; Output buffer empty?
8D 97E2          tty<-ac         ;   Yes - send the char
8E ØØØ5          br next         ; Yipe!!!!!
8F ØØ8F          br .

9Ø 967Ø  test:   ac<-Ø
91 97C8          pc<-ac!Ø        ; A little test to see if the machine
92 3894          b=Ø .+2         ; maybe possibly might work
93 ØØ93          br .            ;   It don't.
94 B7C4          mar<-pc!Ø
95 3897          b=Ø .+2
96 ØØ96          br .            ; pc is broken
97 9661          mwr<-Ø
98 4ØØ1          write           ; Location Ø = Ø
99 1899          bmnr .
9A 4ØØ2          read
9B 189B          bmnr .          ; Hang here and memory is kaput
9C 87DØ          ac<-ac!mbr
9D 38C5          b=Ø gotcha      ; Did we read back a Ø?
9E ØØ9E          br .            ;   NO! memory is no good
9F ØØ9F          br .

         ; Dispatch table for simple dispatch test
AØ ØØBØ          br ok           ; Looks like disp might work
A1 ØØA1          br .
A2 ØØA2          br .
A3 ØØA3          br .
A4 ØØA4          br .
A5 ØØA5          br .
A6 ØØA6          br .
A7 ØØA7          br .
A8 ØØA8          br .
A9 ØØA9          br .
AA ØØAA          br .
AB ØØAB          br .
AC ØØAC          br .
AD ØØAD          br .
AE ØØAE          br .
AF ØØAF          br .

BØ 9461  ok:     mwr<- -1        ; Maybe the memory is stuck on zero
B1 4ØØ1          write
B2 18B2          bmnr .
B3 4ØØ2          read
B4 18B4          bmnr .
B5 D75Ø          ac<-mbr
B6 3ØB8          bac<Ø .+2       ; Is it negative?
B7 ØØB7          br .            ;   No. Something's broken
B8 9Ø1Ø          ac<-ac+1
```

```
B9 97D0          ac<-ac!0        ; Was it in fact -1?
BA 38BC          b=0 .+2         ;  well...?
BB 00BB          br .            ;    arrrrgh!
BC 9010          ac<-ac+1        ; Now try something with not all
BD 97E1          mwr<-ac         ;  the nibbles the same
BE 4001          write
BF 18BF          bmnr .
C0 4002          read
C1 18C1          bmnr .
C2 D5F0          ac<-mbr-1
C3 38F0          b=0 exam        ; Looks maybe like it works
C4 00C4          br .

C5 60A0  gotcha: disp 'A0
C6 00C6          br .

C7 F7F0  rswr:   ac<-swr         ; Read switches
C8 0005          br next
C9 00C9          br .
CA 97E1  p1:     mwr<-ac
CB 9464          mar<- -1        ; Need a location to write thru
CC 4003          write,read      ;  so we can get cpcode back into MBR
CD 18CD          bmnr .
CE 6070          disp 70
CF 00CF          br .

D0 2802  memlod: bnss halt       ; Magic mode off?
D1 4800          lite            ;   No - light the light
D2 20D0          bnstart .-2     ;    and wait for 1 or the other
D3 B7C4          mar<-pc         ; Deposit
D4 F7E1          mwr<-swr
D5 4001          write
D6 B008          pc<-pc+1
D7 18D7          bmnr .
D8 20F1          bnstart displ   ; /must unpress START
D9 00D8          br .-1
DA 00DA          br .
DB 00DB          br .
DC 00DC          br .
DD 00DD          br .
DE 00DE          br .
DF 00DF          br .

E0 2002  halt:   bnstart wait
E1 00E0          br .-1          ; On HALT, must unpress START
E2 00E2          br .
E3 00E3          br .
E4 00E4          br .
E5 00E5          br .
E6 00E6          br .
E7 00E7          br .
E8 00E8          br .
E9 00E9          br .
```

```
EA 00EA          br .
EB 00EB          br .
EC 00EC          br .
ED 00ED          br .
EE 00EE          br .
EF 00EF          br .

F0 F7E8  exam:   pc<-swr          ; Magic Mode: Examine loc in swr
F1 B7C4  displ:  mar<-pc
F2 4002          read
F3 18F3          bmnr .
F4 D750          ac<-mbr          ; Display it
F5 00D0          br memlod        ;  and let him change it
F6 00F6          br .             ;  if he wants to
F7 00F7          br .
F8 00F8          br .
F9 00F9          br .
FA 00FA          br .
FB 00FB          br .
FC 00FC          br .
FD 00FD          br .
FE 00FE          br .
FF 00FF          br .
```

addr	Cause?
Ø2	START broken
ØC	Mem hang on read
OE	DISPATCH doesn't do branch
ØF	????????
1A	HANG instruction executed
29	Mem hang on read
2E	Mem hang on read
33	Mem hang on read
38	Mem hang on read
3E	Mem hang on write
42	Mem hang on read
45	Mem hang on simul write,read
4D	Mem hang on read
5Ø	Mem hang on simul write,read
58	Mem hang on read
5E	Mem hang on write
6C	????????
6D	Mem hang on write
6F	????????
82	Mem hang on read
85	TTY in not raising DA
8A	Mem hang on read
8C	TTY out not raising TBE/
8F	????????
93	AC not loading or result=Ø FF kaput
96	PC not loading properly
99	Mem hang on write
9B	Mem hang on read
9E	Val read from mem is not same as val written
9F	????????
A1-AF	DISPATCH or memory kaput
B2	Mem hang on write
B4	Mem hang on read
B7	Either read does not get written data or AC<Ø is broken
BB	read data not same as written data (Mem stuck on zero?)
BF	Mem hang on write
C1	Mem hang on read
C4	Mem writes same thing to all 3 nibbles?
C6	DISPATCH broken
C9	????????

CF	????????
D7	Mem hang on write
DA-DF	????????
E2-EF	?????????
F3	mem hang on read
F6-FF	????????

```
        ; STUPID demonstration program
        ; Accepts two BCD numbers (4 digits each)
        ; from the switches, typing them out
        ; on the TTY as they are read in. Then
        ; goes into operation mode, where the two
        ; numbers can be added, subtracted, anded or ored
        ; and the result printed in octal on the TTY.

        ; Start at location 9E. Antisocial people start
        ; at location 05

                        .loc    5

05  587   gnorf:    clear   resl            ; Will hold 1st operand
06  588             clear   res2            ; 2nd operand
07  87E             load    four            ; 4 BCD digits per word
08  98B             store   count
09  98C             store   tnuoc

0A  F30   grok:     reads                   ; read the ol' switches
0B  37B             and     hibit
0C  C0E             bz      igor            ; Make sure Hi order bit is off
0D  B0A             branch  grok            ; Otherwise must wait

0E  F30   igor:     reads                   ; Get the value
0F  989             store   s
10  37B             and     hibit           ; High bit=1 => take it
11  C0E             bz      igor
12  889             load    s               ; Get num back
13  382             and     f               ; Mask to digit
14  989             store   s               ; s has BCD digit
15  181             add     zero            ; Type it out
16  F20             pchar
17  887             load    resl            ; Old result
18  187             add     resl            ; old*2
19  98A             store   temp
1A  18A             add     temp            ; old*4
1B  987             store   resl
1C  187             add     resl            ; old*8
1D  18A             add     temp            ; old*10
1E  189             add     s               ; old*10+new
1F  987             store   resl
20  78B             dsz     count           ; Got 4 digits?
21  B0A             branch  grok            ;   Not yet

22  87F             load    cr
23  F20             pchar                   ; Ker
24  880             load    lf
25  F20             pchar                   ;    chink
26  F30   korg:     reads
27  37B             and     hibit
28  C2A             bz      rogi            ; Wait for it to go low again
29  B26             branch  korg
```

```
2A  F30   rogi:   reads
2B  989           store   s                 ; Do it all agin
2C  37B           and     hibit
2D  C2A           bz      rogi
2E  889           load    s
2F  382           and     f
30  989           store   s
31  181           add     zero
32  F20           pchar
33  888           load    res2
34  188           add     res2
35  98A           store   temp
36  18A           add     temp
37  988           store   res2
38  188           add     res2
39  18A           add     temp
3A  189           add     s
3B  988           store   res2
3C  78C           dsz     tnouc
3D  B26           branch  korg
3E  87F   crlf:   load    cr
3F  F20           pchar
40  887           load    lf
41  F20           pchar

42  F30   loop:   reads
43  37B           and     hibit             ; Hey, I don't need this instr!
44  D42           bn      loop              ; Because it's negative anyway

45  F30   pool:   reads
46  989           store   s
47  37B           and     hibit
48  C45           bz      pool

49  889           load    s                 ; Look for operation now
4A  383           and     sum               ;  He wanna add?
4B  E56           bp      addem             ;   Yup.
4C  889           load    s
4D  384           and     diff              ; Subtract?
4E  E59           bp      subem
4F  889           load    s
50  385           and     .and              ; and?
51  E5C           bp      andem
52  889           load    s
53  386           and     .or               ; or?
54  E5F           bp      orem
55  000           halt                      ; None of the above. Quit.

56  887   addem:  load    res1
57  188           add     res2
58  B62           branch  type              ; Go type out the result
```

```
 9  887   subem:   load     resl
 A  288            sub      res2
 B  B62            branch   type

 C  887   andem:   load     resl
 D  388            and      res2
 E  B62            branch   type

 F  887   orem:    load     resl
 0  488            or       res2
 1  B62            branch   type

 2  98A   type:    store    temp        ; Save result
 3  87E            load     four        ; Four octal digits per word
 4  98C            store    tnuoc

 5  87D   there:   load     three       ; 3 bits per octal digit
 6  98B            store    count
 7  58D            clear    digit
 8  88A   here:    load     temp        ; Get word back
 9  37B            and      hibit
 A  C6C            bz       skip        ; 0 or 1?
 B  87C            load     one         ; 1 - make it 1 instead of 1B11
 C  98E   skip:    store    xxx         ; Save the bit
 D  88A            load     temp
 E  18A            add      temp
 F  98A            store    temp        ; Shift result word
 0  88D            load     digit       ; Running sum...
 1  18D            add      digit       ; *2
 2  18E            add      xxx         ;   + bit
 3  98D            store    digit       ; And that's the new one...
 4  78B            dsz      count       ; Got a whole octal digit yet?
 5  B68            branch   here        ;   Nope. Keep going

 6  181            add      zero        ; ASCII-ize it
 7  F20            pchar                ; Type the digit
 8  78C            dsz      tnuoc       ; Finished?
 9  B65            branch   there       ;   Not yet
 A  B3E            branch   crlf        ;   Yes. Go get another op

 B  800   hibit:   800                  ; Most signif bit
 C  001   one:     1                    ; Yup.
 D  003   three:   3
 E  004   four:    4
 F  00D   cr:      D                    ; Carriage return
 0  00A   lf:      B                    ; Line feed
 1  030   zero:    30                   ; ASCII "0"
 2  00F   f:       F                    ; Low order 4 bit mask
 3  001   sum:     1                    ; Bit which means "add"
 4  002   diff:    2                    ; "subtract"
 5  004   .and:    4                    ; "and"
 6  008   .or:     8                    ; "or"
 7  000   resl:    0
```

```
38  000   res2:     0
39  000   s:        0
3A  000   temp:     0
3B  000   count:    0
3C  000   tnuoc:    0
3D  000   digit:    0
3E  000   xxx:      0

          .loc      9E

9E  8A8   start:    load      lenth
9F  98B             store     count
A0  8A9             load      kludge
A1  9A2             store     inst
A2  8AA   inst:     load      msg
A3  F20             pchar                       ; Type out something
A4  6A2             isz       inst              ; Next char...
A5  78B             dsz       count             ; Done?
A6  BA2             br        inst              ;   Notchet
A7  B05             br        gnorf

A8  00B   lenth:    B                           ; 11 chars in mess
A9  8AA   kludge:   load      msg               ; typer modifies inst...
AA  048   msg:      "H
AB  049             "I
AC  020             "
AD  054             "T
AE  048             "H
AF  045             "E
B0  052             "R
B1  045             "E
B2  00D             D
B3  00B             B
B4  00B             B

          .end
```