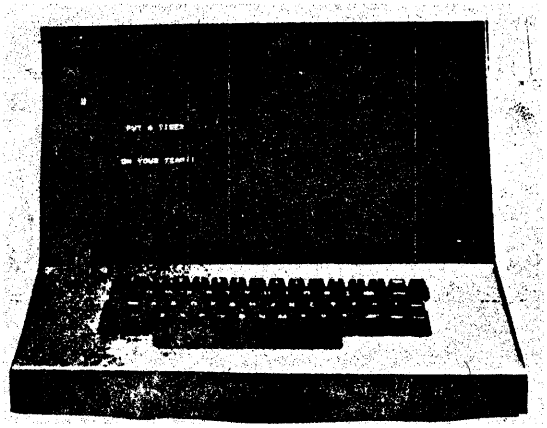


SPHERE

VOLUME V NUMBERS 4 and 5 May 1981

EDITORS: Jeffrey Brownstein
Roger J. Spott

PLEASE SEND MATERIAL FOR THE NEXT ISSUE TO: Dr. Jeff Brownstein
2 Tor Road
Wappingers, N.Y. 12590

PUT A 6809 IN YOUR SPHERE MICROCOMPUTER

Dear Sphere Users,

This article represents a very significant upgrade of the Sphere system. I have the conversion installed in mine and invite any of you to drop by when you are in the new york area. The MOD contains the following features:

One crystal controlled clock running both CPU chips.

Four to eight EPROM chips. Using 2708 or 2716 proms, four of them are active during 6800 operation while the others overlay the memory space and work with the 6809. Thus the reset and interrupt vectors which reside in the highest address prom are appropriate for each.

Only one monitor is required because the 6809 can access the I/O in the V#N#3 standard when needed. The software interrupt is re-vectored. My implementation leaves even the software interrupt alone and uses the wait for interrupt code. As described in a previous newsletter, the NMI is activated in my scheme upon execution of the wait for interrupt opcode.

The CPU of choice is selected statically with a dip switch or on the fly using one PIA B side line. This line also controls which proms are active and halts the unused processor.

For ease of removal, the adaptor board plugs into the empty 6800 socket. The CPU board will operate with the 6800 back in place but crystal clock is then not used. The adaptor board contains both CPU chips (socketed), the crystal as well as a few 7400LS chips.

The piggy back prom board plugs into one of the 1702A sockets just like the programma board. In fact it is an enhanced version to allow two banks of prom. The plans for the programma board are presented here; there is no reason that this enhancement alone cannot be implemented (but without paying out any money) on a simple perfboard. If you do not wish to use the 6809 then alternate memory banks (though still useful) are not absolutely necessary.

Besides the 6809 and some Eproms, the cost of this entire conversion is very small.

All of the features of the 6809 are supported including position independent code, multiple software interrupts, 16 bit instructions, hardware multiply, sync signal, etc. Note that the 6809E (which has an external clock and is reccommended for multiprocessing and multitasking) was not used. I could not find a supplier for the 6809E. The project could use the E version with some minor wiring changes.

Although not implemented yet, It should be possible to run the system at a fast clock rate while slowing down the CPU only during accesses to external memory. The 6809 has a memory ready line on it for this purpose. Certain mask numbers of the 6809 have different requirements for synchronising the memory ready with the system clock so I this has been left alone for now.

The first part of the conversion should be the construction of the prom piggyback board. Look over the Programma plans. An artwork sample for a PC board is included for those who would duplicate the Programma board exactly. With the plans in hand, I needed some time to figure out what was the best way to proceed. Make up your mind what type of proms you will use and where you would like them addressed. Look at the address table which is presented by Programma on the plans. Because of the peculiarities of I/O locations used by Sphere, you will see that although the 2716 will give far more memory available, the smaller 2708 will slightly better utilize the memory space around the I/O ports. Because I have other memory from D000-DFFF I chose the 2708 as did most of the people who bought the Programma boards.

The main feature of the decoding is the 74LS30 . I chose to place this in the spare socket of the cpu board. First I cut all traces going to all but pins seven and fourteen (the power and ground pins). If you are going to use the two memory banks you can use two 74LS30 chips mounted one on top of the other. All pins except one of the input lines are soldered together. The input pin of the top (piggyback) one is bent out. Now, the way these chips work, the proms (thru the 7442 of course) will only be selected when all of the inputs to the 74LS30 are positive. Now you can see how I implemented two prom banks. The control line for the processors and proms will either be hi or lo. It is tied to one of the 74LS30 chips input directly and to the other thru an inverter. The result is that one and only one bank of prom can be selected at any one time. It is possible to test the circuit so far by placing a scope or logic probe on the output of the 74LS30's one at a time. Verify that as you control their inputs and address memory in their range, the output goes lo (briefly). At this point the whole system is still intact with the 1702A chips in place and it still runs fine.

I now piggybacked a 7442LS on top of the existing one on the CPU board. One output from one 74LS30 goes to one 74LS42 and the other to the other 7442. You are left with a whole bunch of outputs on each 7442LS: these will go to the chip select pins of the proms according to which addresses and which kind of proms you are using. I know that this whole business with piggybacking sounds messy. Put separate sockets if you wish. The Programma board carried the 74LS30 on it. I preferred it on the CPU card to test the bank selects before removing the 1702A proms.

Construct from a piece of perfboard a piece about 2" wide by 3.75" long. If you want to use all 8 Eproms, make the board four inches wide instead. If you are going all the way to the 6809 conversion, also cut a piece of perfboard 3.75" X 3.75" and put it aside.

At your local Radio Shack you will find the necessary number of 24 pin sockets as well as a single ended 24 pin dip jumper (which will plug into the empty E35 socket of the CPU board). Most of the lines of the E35 socket will easily furnish signals appropriate for the 2708 board. Ground, -5, 12 A8 and A9 will have to be drawn from the CPU board individually. The chip selects of all the 2708 proms will also have to be brought individually to the edge of your perf board. All of the other pins of your 2708 chips are to be tied together and then to the 24 pin dip jumper according to the chart below.

Signal	1702A (and jumper) pin	2708 pin
A7	17	1
A6	18	2
A5	19	3
A4	20	4

Cont'd.

Signal	1702A (and jumper) pin		2708 pin
A3	21		5
A2	1		6
A1	2		7
A0	3		8
D1	4	Note: some	9
D2	5	data sheets list	10
D3	6	data lines as	11
D4	7	D0 to D7 while	13
D5	8	others use D1 to D8.	14
D6	9	In either case the	15
D7	10	correct pin connects	16
D8	11	are here.	17
VCC (+5 Volts)	13,15,22,23,12		24

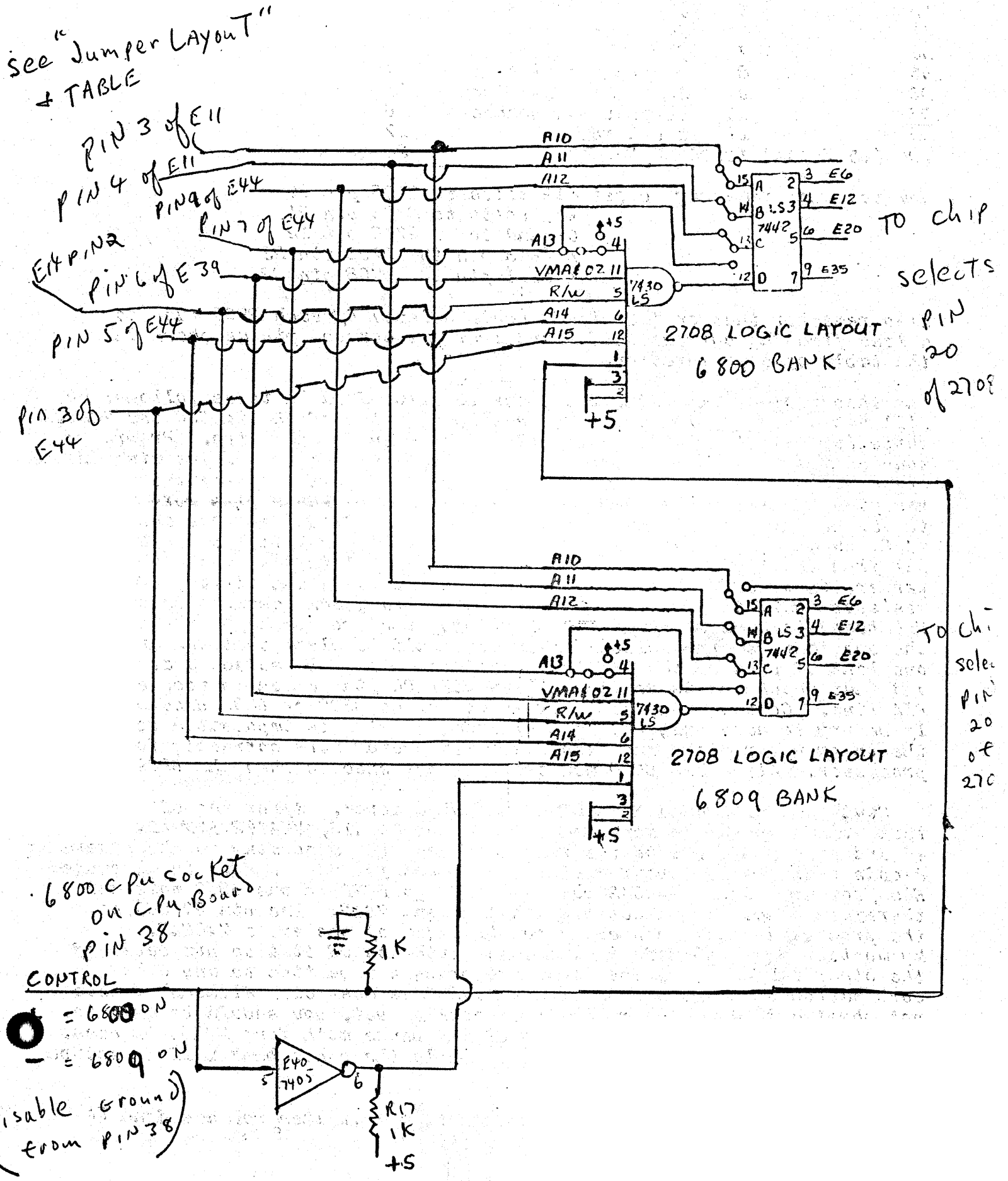
Now you must remember to get +12 volts to 2708 pin 19
-5 volts to 2708 pin 21
Ground to 2708 pin 12
A8(E13 pin 15) 2708 pin 23
A9(E13 pin 14) 2708 pin 22

Also remember that on the pin 20 (chip select) of each 2708 must go a lead from the correct 74LS42 and from the correct pin as shown in the table of configurations.

The theory of multiplexing the proms to allow 2 banks is as follows: We will use a PIA line to do the selecting. On top of the 74LS30 and 74LS42 chips (or next to them etc) we will connect duplicate chips. Programma mounted its 74LS30 on the conversion piggyback board but I put mine in the spare socket of the CPU board. According to Programma's circuit, the 74LS30 has pins 1,2 and 3 tied together. If any one of these pins were not tied to +5, no chip select could occur through that 74LS30 and its associated 7442. What I do is drive one of the 74LS30 's pin 1 with the CONTROL signal from the 6809 board. The other 74LS30 pin 1 is driven through an inverter. On my CPU board, E40 pin 5&6 was available and already had a 1K resistor (R17) tied to the output. This arrangement assures that if the CPU board is used without the 6809 piggy, the prom bank for the 6800 is the one that remains selected. Now it should be clear that one and only one bank of prom on the CPU board will be able to be selected at any time and that which one is dependent upon CONTROL line which is toggled by a PIA line. Control line also serves to select 6800 or 6809 processors. If we try to have only one bank of prom, it will be impossible to have the interrupt vectors and also the reset vector work correctly for each processor. Also, the 6800 monitor will not execute when the 6809 is up.

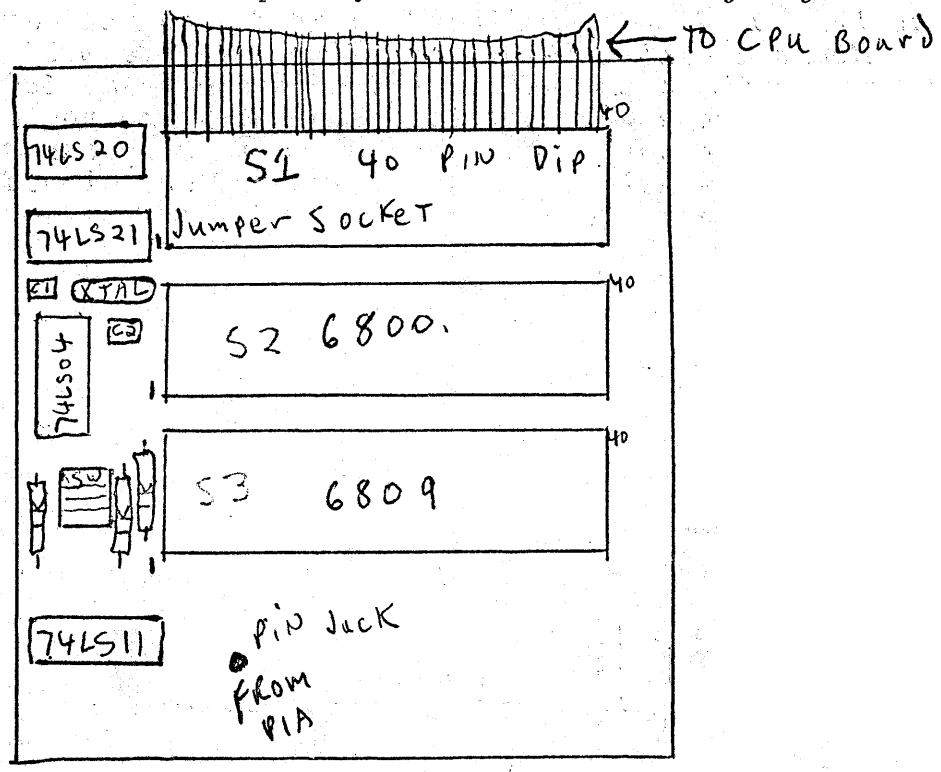
Study the Programma 2708 LOGIC LAYOUT diagram. After the CPU board back side jumpers are done, make certain that A13, VMA&Ø2,R?W,A14,A15, +5 and ground all get to the 74LS30 correctly. According to the programma decode chart, select your desired addresses for the proms. On my system E35 from the second 74LS42 goes for FC00 to FFFF in the 6809 mode and therefore hangs on pin seven of that second 7442. The pin 20(CS) of the prom is directly connected to pin seven of the extra 74LS42. My monitor for 6800 mode is connected with its CS tied to pin seven of the other 74LS42. Now the other two proms may be tied to any other combination of addresses on either 7442LS as desired. Although I have not physically added four more prom sockets yet, you should be able to see that there exists a potential of two banks with four 2708s in each. One bank will run when CONTROL is hi while the other bank will be active when CONTROL is lo.

The best thing about this arrangement is that when you are done it is easily tested. Do this before continuing to 6809s.


See "Jumper LAYOUT" + TABLE
PIN 3 of E11
PIN 4 of E11
PIN 4 of E44
PIN 7 of E44
E14 PIN 2
PIN 6 of E39
PIN 5 of E44
PIN 30 of E44
A10
A11
A12
A13
VMA & Q2
R/W
A14
A15
+5
7430 LS
74LS42
2708 LOGIC LAYOUT
6800 BANK
TO chip selects PIN 20 of 2708
2708 LOGIC LAYOUT
6809 BANK
TO chip selects PIN 20 of 2708
6800 CPU socket on CPU Board PIN 38
CONTROL
= 6800 ON
= 6809 ON
(Disable Ground from PIN 38)
1K
E40 7405
R17 1K
+5

To test your prom 2708 setup merely insert a 2708 in each socket. If you blast one containing the V3N monitor, you first can set that one up as the highest address of the 6800 prom bank. Other proms may be then inserted in the other available sockets and their contents read with the debugger. Now connect +5volts to CONTROL line and the monitor should run perfectly in the highest address socket of the other bank. When all of this runs fine you are ready to go on to the 6809 adaptor. I believe that Warren Weimer has access to a few Programma piggyback PC boards (although I built mine from scratch). Whether the Programma board is used or not, you will still have to add one 74LS30 and one 74LS42 at least. Actually, I changed the original 7442 to a 74LS42 for good luck.

It is necessary next to procure a 12" 40 pin double ended jumper. This part may be obtained by mail from DigiKey or Jameco. On the square perf-board lay out three 40 pin (female) sockets. My layout is as below:



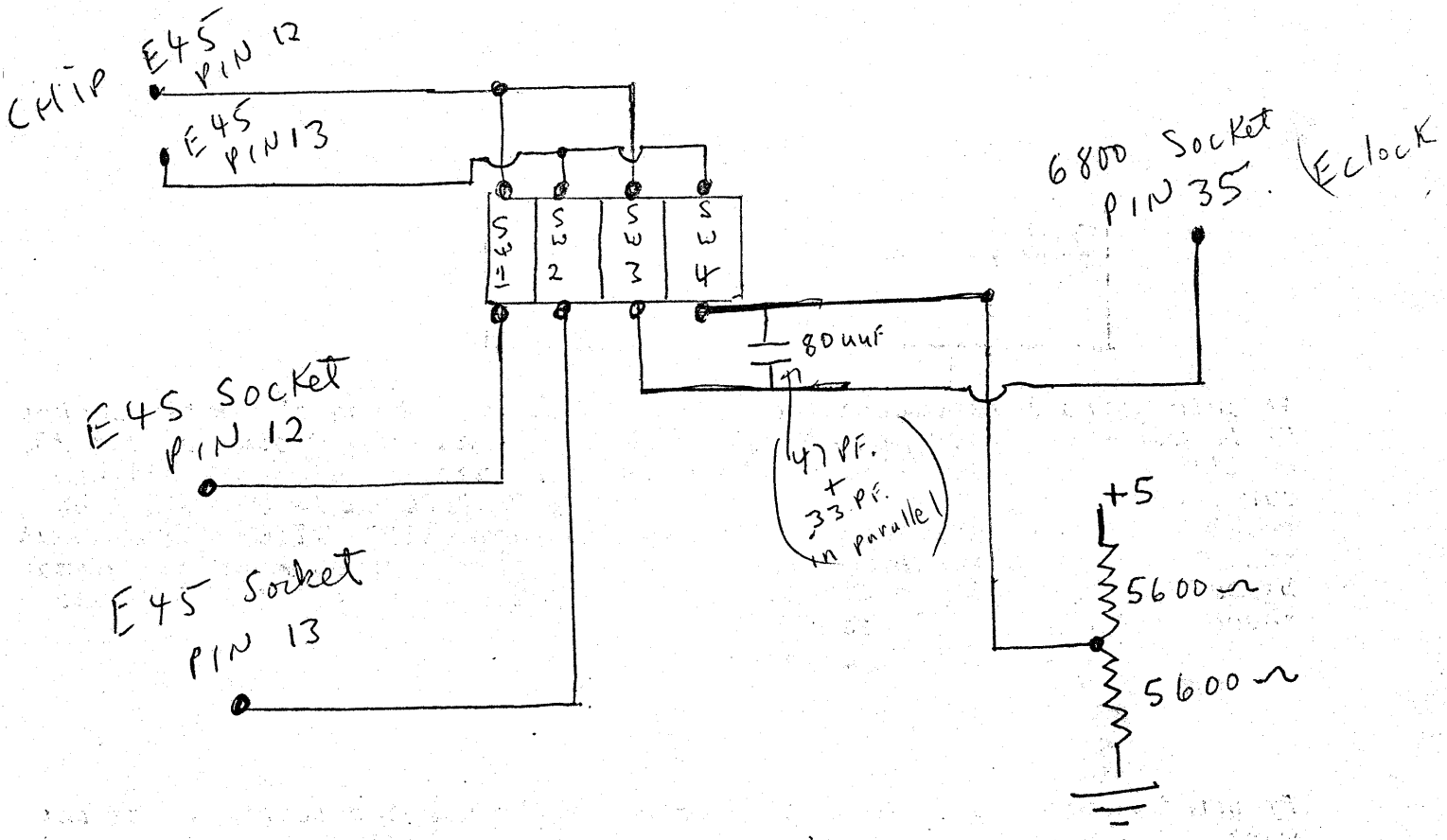
At this point I recommend that the 6800 socket S2 and the cable socket S1 be wired together pin for pin except for the unused pins 35 and 38. We will use them to allow two signals to transverse distance between our CPU and adaptor board. 38 will carry CONTROL while 35 will take the E clock from the adaptor board to the new CLOCK SWITCH (more about that later). There is no reason that the following connections cannot be made to the 6809 socket at this time (do not install the 6809 yet though): ALL 8 DATA lines
ALL 16 ADDRESS lines
R/W
RESET
VCC
VSS

If you install the 6800 in the socket of the adapter board it had better work. At least if it doesn't you have just a problem for solution with an ohmmeter. When all is well, continue on next page.

Now the next consideration is the clock circuitry. The 6800 chip gets clock pulses into it; two phases yet! The 6809, on the other hand, puts out clock pulses on the E and Q lines and crystal controlled. A quartz crystal is divided down in frequency by four. ARRRGH!!!!!!

Here is a way to reconcile all of this: E45 is the Sphere clock generating chip. It has two sections. (SEE SCHEMATIC) the right side generates Ø2 and then triggers the left side which generates Ø1 and then triggers the right side again. If we could break this cozy little loop and feed the right half with the E clock from the 6809, all would run as before including the left side still generating Ø1 for the 6800. Memory refresh would occur exactly as before and there would be a smooth transition from processor to processor since the same clock would run everything.

Now let's do it. Bend up pins 12 and 13 of E45 so they are not in the socket. Mount a 4unit dip switch below E51. We will make provision for running the old clock in the event that the 6809 board is ever removed. Also note that a little capacitor and voltage divider was necessary to force the right side of E45 to have the right input swing. The circuit is below. When SW1 and SW2 are closed the old clock works. When these are open and SW3 and SW4 are closed the 6809 clock will run the system. By the way, the divider resistors and caps have been tested & run ok for frequencies between 500KHz. and 1MHz. Different values may be needed if you pass these limits.



While you are near E45 you might as well chang R13 (above E45) to 11K or less to enhance memory write operations. (see previous newsletters).

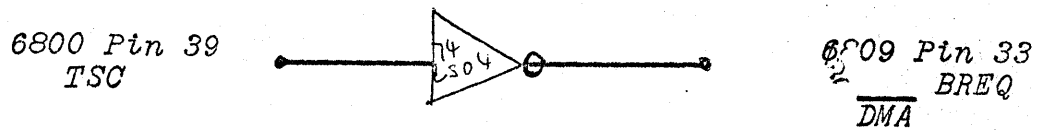
At this point the 6800 should still work while on the adapter board as long as SW1 and SW2 (see above) are closed. Now there are more

wiring changes to be made before the 6809 can be plugged into its socket. They are on the adapter board only.

1-Install crystal. The final project works well with a 4.000 MHz. quartz crystal. This allows both the 6800 and 6809 to run at their rated clock frequencies. I reccommend that during debugging of this project, a lower frequency crystal be substituted. Why add complications? You know the system runs ok at 6 or 7 KHz. so a 3MHz crystal would be ideal at the beginning. I actually used a 2.0 MHz. one because it was all I could find around. The crystal is connected between pins 38 and 39 of the 6809 socket. You should run a 27 Pf. capacitor from each crystal lead to ground. Try to keep this wiring short as possible.

2-The 6809 E Clock. This is the same as the 6800 Ø2 clock. Run a wire from 6809 socket (S3) pin 34 to pin 35 of the dip cable connector socket. Voila! Cable connector socket is S1.

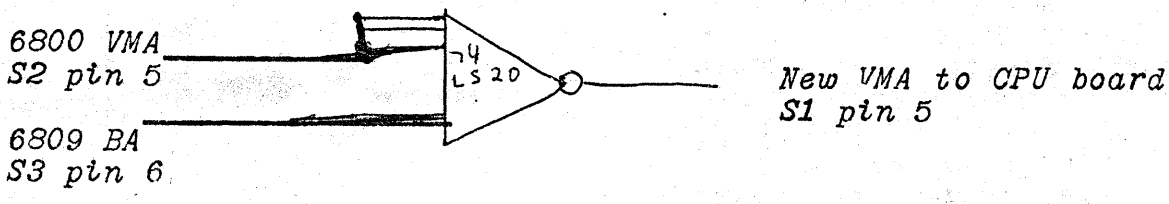
3-$\overline{DMA}$ BREQ of 6809. Run a line from 6800 S2 pin 39 (TSC) to the input of an inverter (74LS04). The output of the gate goes to $\overline{DMA}$ BREQ of the 6809 S3 pin 33



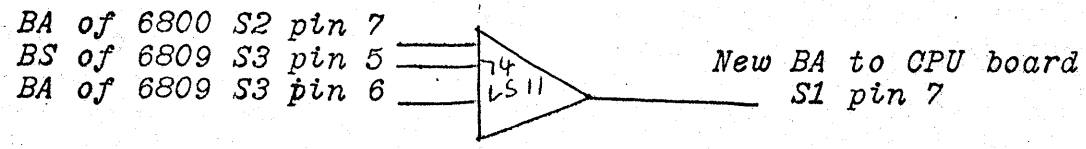
4- $\overline{FIRQ}$ of the 6809 Just connect pin 4 of the 6809 to +5 thru a 1K resistor.

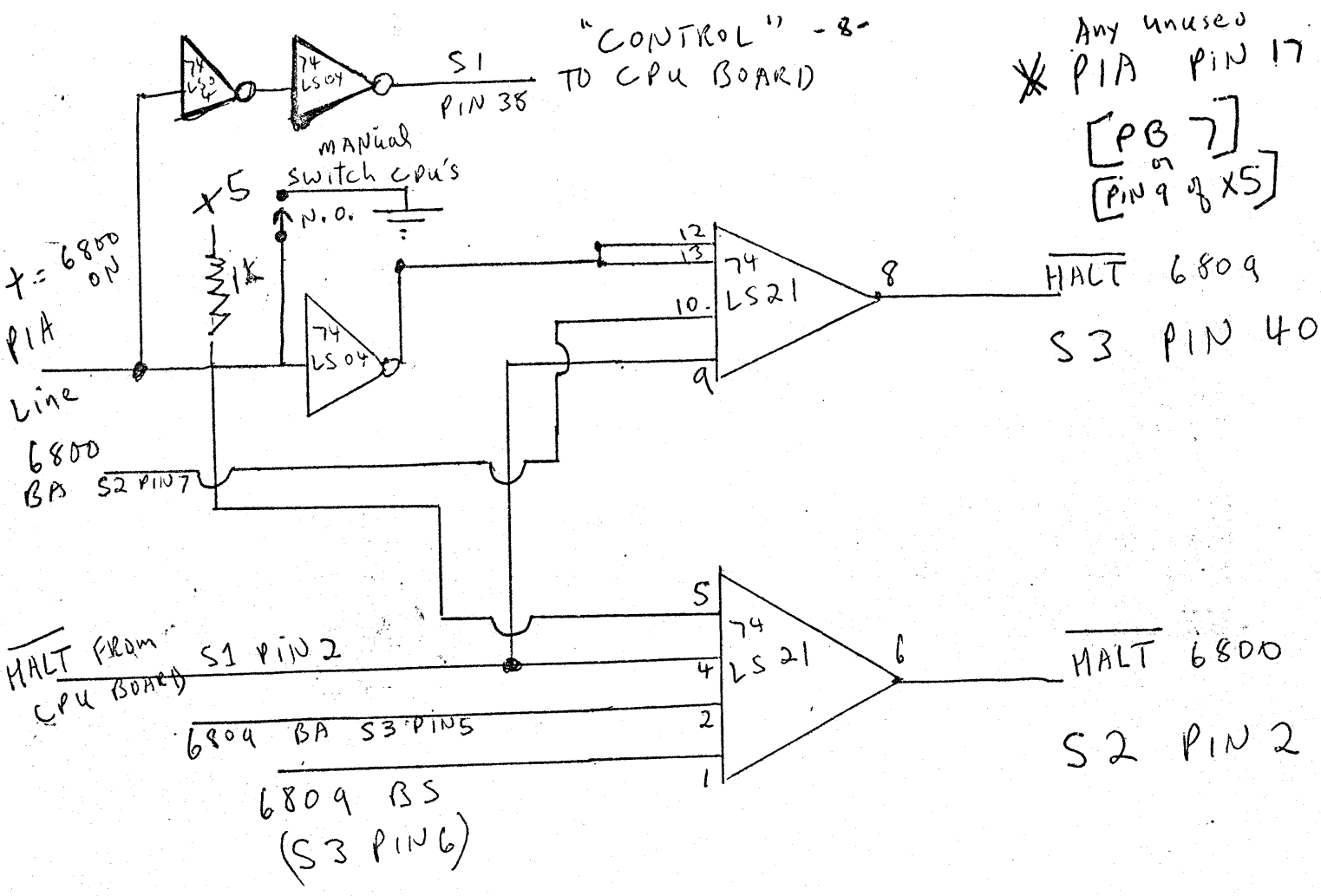
5- MREADY of the 6809 Just connect pin 36 of the 6809 to +5 thru a 1K resistor.

6_ VMA The old 6800 VMA is added and inverted with the 6809 BA to form the new system VMA. It works!!!



7-BA





Above is the circuit which, on the command from a PIA line, causes one, the other, or, during refresh, both of the processors to halt. A manual SPST switch is incorporated to allow easier troubleshooting and to provide a way for the machine to be locked into the 6809 mode if desired.

The software half of this project requires some RAM or ROM which is outside of the bank select areas. In other words, this program must always be on line. In my system it resides in some ROM at D000. Also, correct transfer vectors must be installed in the highest memory (FC00-FFFF) ROM for each bank. The correct tables of jumps will be given later. The only requirement for the old V3N monitor rom is that the software interrupt be vectored to handle the transfer from 6800 to 6809. The idea is that upon encountering a 3F the interrupt handler routine does a few housekeeping things and then switches the PIA line thus switching processors. The registers of the 6800 are transferred to a stack from which the 6809 retrieves them.

Personally, I never cared for dedicating the software interrupt to the above use. Some few newsletters ago I described a way of hooking the BA line of the 6800 to the NMI line. Now upon receiving a WAI or in object code 3E , the essentially same thing happens as with a 3F. Most of us have no other use at this time for the NMI line or the wait for interrupt instruction anyway. If you do it with NMI, all of your old software will have the breakpoint available as before. Another way to solve the problem might be to vector the software interrupt to RAM and have that next jump in RAM set one way for breakpointing and another when using the 6809 processor.

* see page 29

THE SOFTWARE 6800/6809 SWAPPER (a Bside wire will go from your PIA pin 17 to the new hardware)

```
PIADATA   EQU   F18B
PIASTAT   EQU   F188      A side is only used as register to see if cold
PIADIR    EQU   F18A
SP6800    EQU   C382      put it where you wish in RAM
SP6809    EQU   C05C       "      "      "
STACK09   EQU   C028
```

Vectors for 6800 Hi PROM FC00-FFFF (contains v3N monitor).

```
FFF8  0104  IRQ     same as before
      0100  SWI     If you decide to shift processors with 3F then E932
      E932  NMI     WAI (3E) causes NMI which jumps to SWAPPER routine
      FC00  RESET   same as before
```

Vectors for 6809 Hi PROM FC00-FFFF (rest of prom available for user)

```
      FFF0  RESERVED by MOTOROLA 0000
      FFF2  E098 SWI 3   use for something else later
      FFF4  E998 SWI 2   use for something else later
      FFF6  E992 FIRQ    set like reset for now
      FFF8  E992 IRQ      "      "
      FFFA  E998 SWI 1   for now any of the SWI's will switch you to 6800
      FFFC  E992 NMI     use for something later
      FFFE  E992 RESET 09
```

ACTUAL SWAP ROUTINE (must not reside in banked proms)

```
            ORG E932
7D F188     TST PIASTAT    Is it cold start? SWI COLD START
26 08       BNE GO         do not reset stack unless cold start
CE C01D     LDXI           STACK 09-11
FF C05C     STX            SP 6809
6F 03       CLRX,3
GO BF C382  STS            SP 6800          SWI WARM START!
FE C05C     LDX            SP 6809
32          PULA
A700        STAX,0         PULL DATA FROM
32          PULA           6800 STACK AND
A702        STAAX,2        STORE IT ON 6809
32          PULA           STACK.
A701        STAAX,1
32          PULA
A704        STAAX,4
32          PULA
A705        STAAX,5
32          PULA
A70A        STAAX,10
32          PULA
A70B        STAAX,11
7F F18A     CLR PIADIR     SWAP PROCESSORS
7F F18B     CLR PIADATA
86 FF       LDA#FF
B7 F18A     STAA PIADIR
B7 F18B     STAA PIADATA
4F          CLRA
B7 F18A     STAA PIADIR
01          NOP            WAIT HERE  (do not try to omit this NOP)
FE C05C     LDX            SP 6809
A60B        LDAAX,11       TRANSFER FORM 6809 STACK
36          PSHA           TO 6800 STACK
A60A        LDAAX,10
36          PSHA
A605        LDAAX,5
36          PSHA
A604        LDAAX,4
```

Swapper software continued.

```
36          PSHA
A601        LDAAX,1
36          PSHA
A602        LDAAX,2
36          PSHA
A600        LDAAX,0
36          PSHA
3B          RTI             AND RETURN

RESET 09  10FE          INITIALIZE THE 6809 STACK and PULL EVERYTHING
          C05C           (this is 6809 code)
          35FF

SWI 1     10FF          SAVE STACK POINTER AND SWAP PROCESSORS
          C05C            BACK TO  6800
          86 FF
          B7 F188          (this is 6809 code)
          7F F18B
          7F F18A
          12 3B         Do not try to eliminate the NOP
```

Actual address of RESET 09 was E992
Actual address of SWI 1 was E998

Notes: Besides getting the correct vectors into each prom, the software is relatively short but do not be deceived by its apparent simplicity. Be careful if you try to rewrite it. It is possible to cut two thirds of it away by coding the largest part in 6809 instead of 6800 but we have not debugged this yet. Except for the prom vectors, the thing may be debugged and played with in RAM. The prom vectors may also be vectored to ram where you can modify them if you want to play around.

There is a need for the swapper program to reside outside of the banked proms. I had a 2716 from E800-EFFF so I put it in that. Here is another possibility: Once you have 2708's in the Programma or home brew piggyback, the prom on your SIM will be unplugged. The routines will (should) be in the prom which now is at E20. What to do with that empty socket?? It might be possible, after re-addressing the socket slightly, to try to find one of your 1702A proms which will run ok at 1MHz. This is easy to try (and nevermind what is in the prom). Unplug your 2708 at E20 and stick each prospective 1702A into the prom socket on the sim board. You might hit it lucky. I could burn in your 1702A with the swapper code. If none of the 1702a's will work, you could put a 2708 in and rewire the socket as well as the addressing circuitry. Of course, you could keep the swapper stuff all in RAM if desired or you could experiment by trying to divide it up and put it in the banked proms (good luck).

Gentlemen, this entire thing works very well. I could build another prototype very quickly though it took literally hundreds of hours to get the first one up. Nothing ever works right the first time for me. Please forgive my writing style as I really have not got time to rewrite this article over. I will help anyone who decides to go

6809.

More Notes on the 6809 conversion.

On the previous page I suggested that the unused 1702A socket on your SIM board might be an easy place to put the SWAPPER routines. Well, I just tried it out. Sure enough, the first junky old surplus 1702A worked fine. It executed perfectly in my system with the 1MHz. crystal clock. That is not to say that every 1702A will work but I bet that most will. According to where you have unused space in the memory map, a few jumpers could re-address that sim board prom socket!

I never gave you a test program to run 6800/6809 so here's one.

```
3E      Wait for interrupt.  If you utilized software interrupt then
        this byte will be 3F instead.
6809 routine

3F  go back to 6800
```

```
0200 BD FC4A   get A character from keyboard
0203 3E  use WAI to goto 6809
0204 4C  Acc. A is incremented by the 6809
0205 3F  goback to 6800
0206 bd FCBC output the character
0209 7E 0200  LOOP
```

This little test routine will print out the ASCII character which is one greater than the one typed in by the keyboard.

P.S. The absolute cheapest way to go 6809 is to buy a PERCOM adapter board for 39 dollars. Use the clock circuit to feed the E clock of the 6809 into your 8602 circuit just as I did for this conversion (capacitor and voltage divider and all). Now select 4 1702a proms which will work a 1MHz. and put a monitor into them (ours but written in 6809 code or PSYMON) Psymon comes as a listing with your Percom kit. I bought the Percom kit because it was the only place that I could find a 6809! Board is left over! Of course the thing is 6809 only unless you want to pull the cpu out of its socket and switch back and forth every time. No thanks!

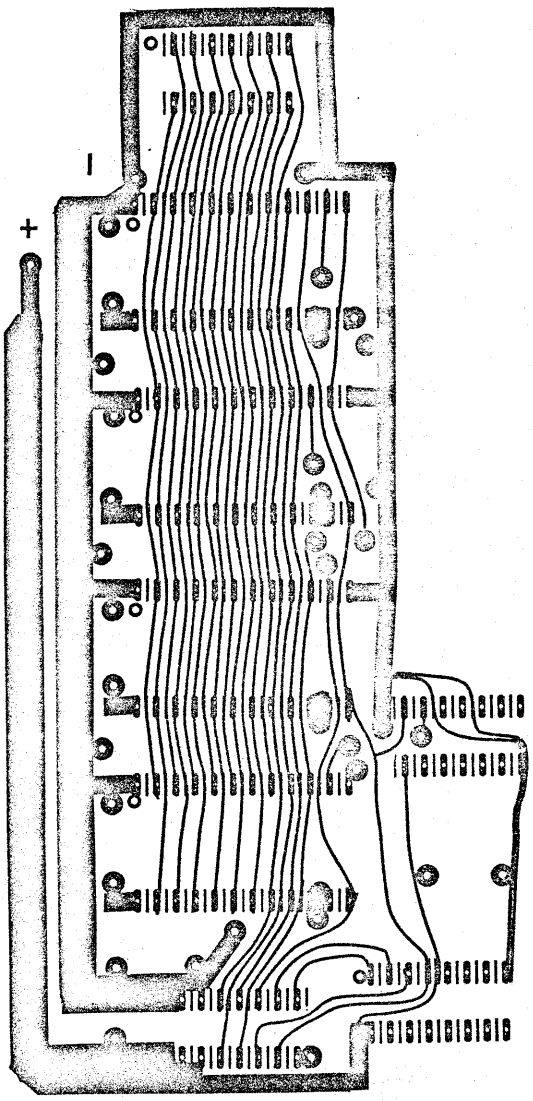
HELP!!!!!!!!!

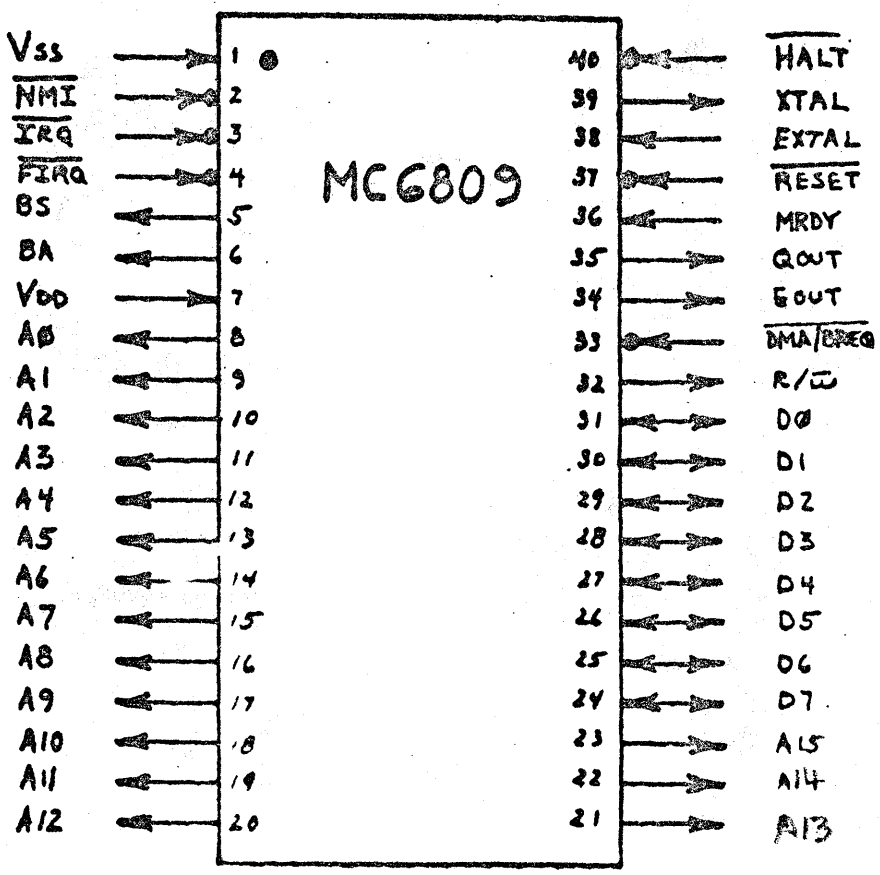
Hey! we need an assembler for the 6809 badly. Probably we could modify a 6800 assembler but we need a good algorithm for the "post byte" which the 6809 uses.
Motorola makes an editor and assembler on cassette for their MEK6809D4B kit (sort of like the old D2 kit.).

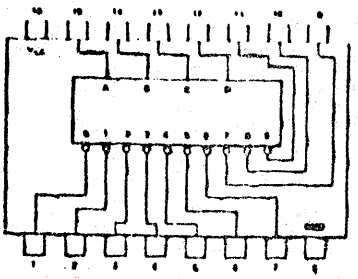
The part number of the cassette is: MEK6809AC

When the local Motorola office heard that I am not with any big company, they would not help me.

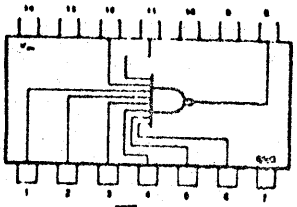
PROGRAMMA PIGGYBACK BOARD ARTWORK



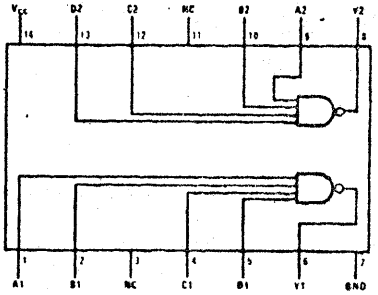




7442 LS

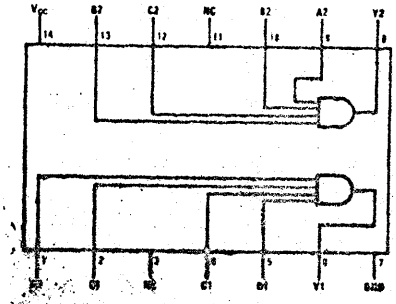


74LS30

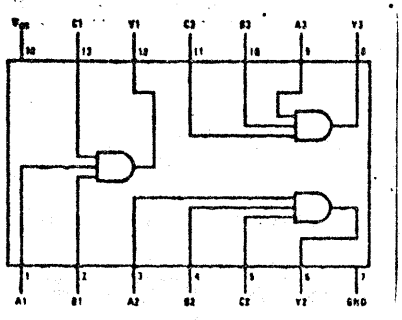


LS
7420

5420/7420(J),(N); 54H20/74H20(J),(N); 54L20/74L20(J),(N); 54LS20/74LS20(J),(N),(W); 74S20(N)

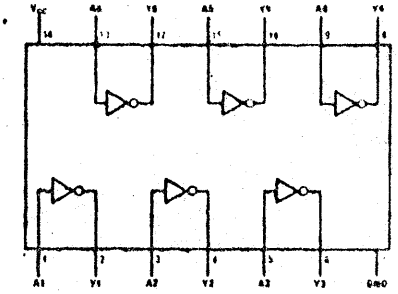


7421 LS



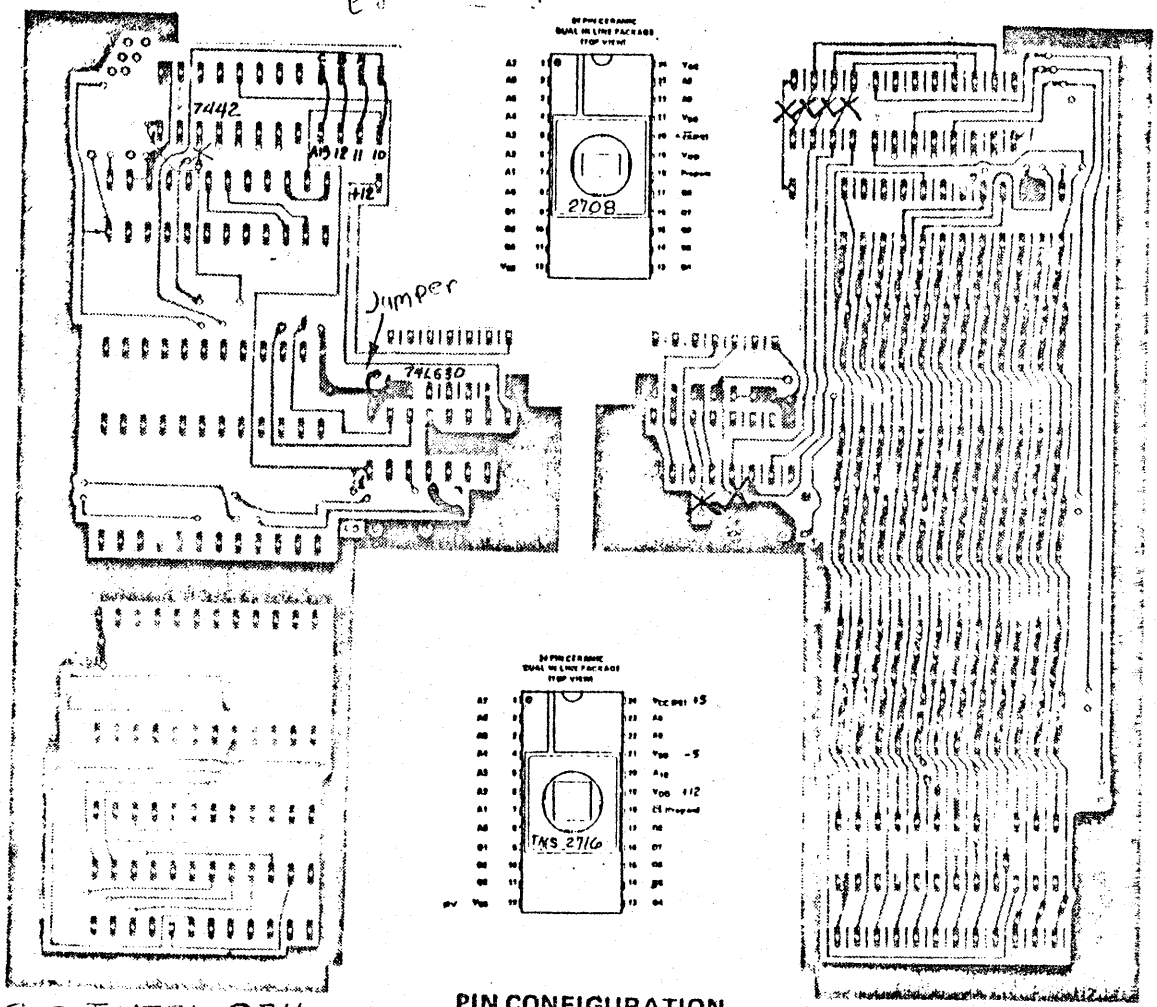
5411/7411(J), (N); 54H11/74H11(J), (N); 54L11/74L11(J), (N), (W); 54LS11/74LS11(J), (N), (W); 74S11(N)

LS 7411



5404/7404(J), (N); 54H04/74H04(J), (N); 54L04/74L04(J), (N), 54LS04/74LS04(J),(N),(W); 74S04(N)

LS 7404 (± 7405)

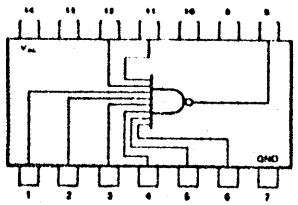


STRAPING FOR INTEL 2716

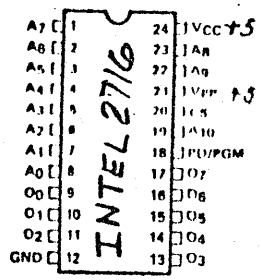
(1) CUT Etch where X marks the spot (6 X's)

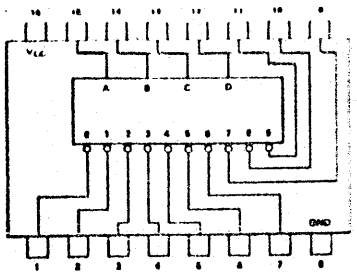
(2) put in jumpers

(Black Lines) (6 jumpers)



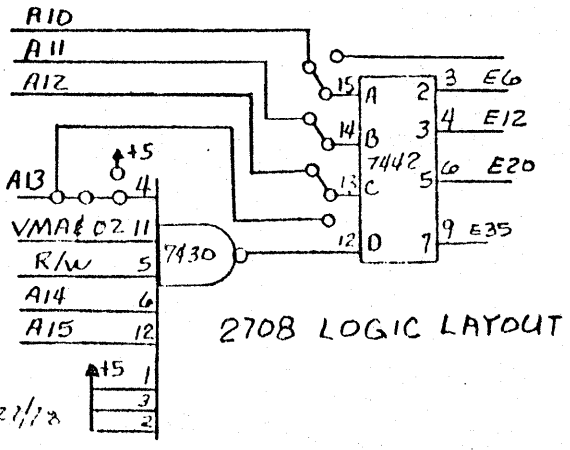
PIN CONFIGURATION





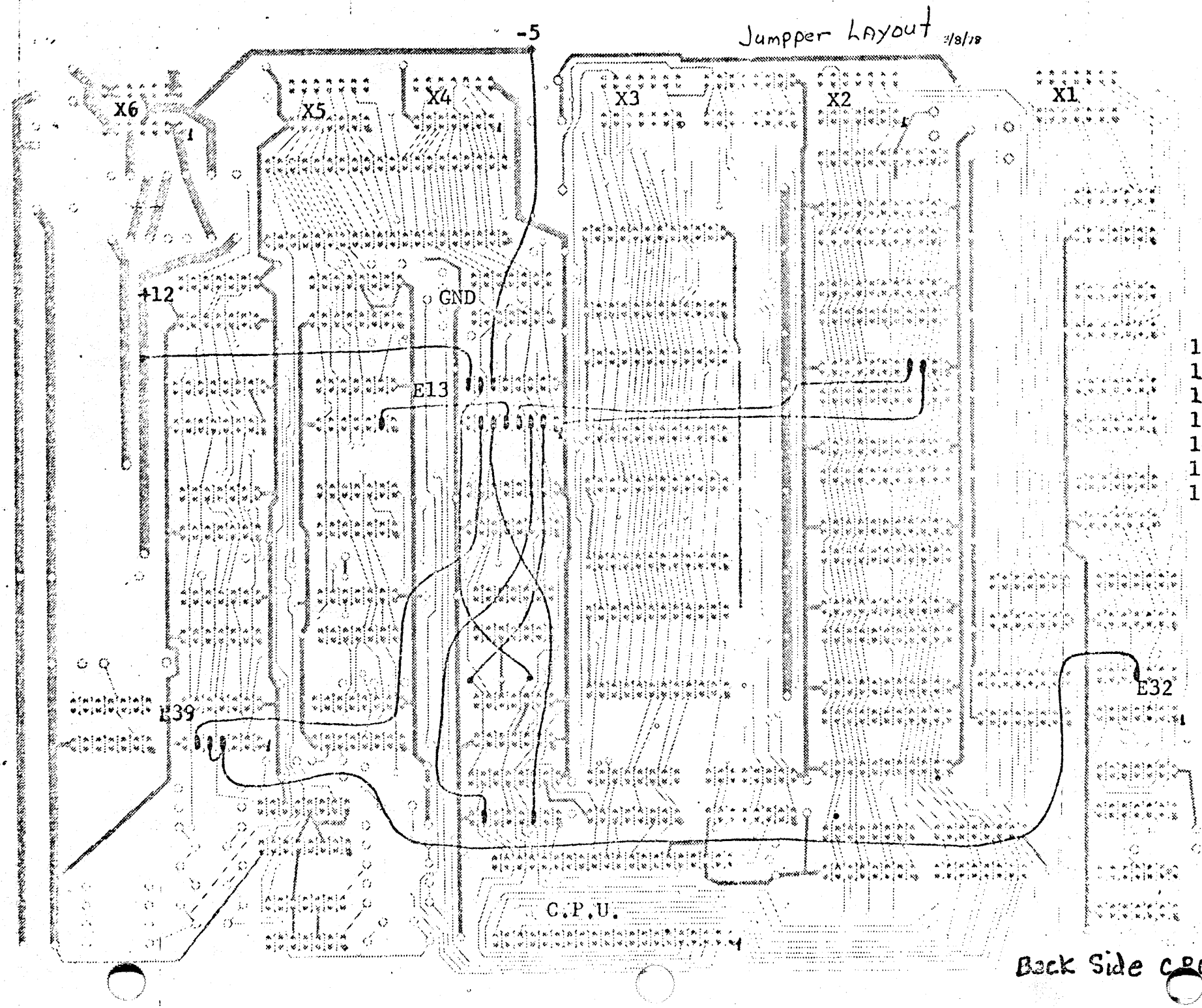
2716 Address Range	PIN number	Binary Code A	B	C	D	Decode number	2708 Address Range	Present Hook up
C000 - C7FF	1	0	0	0	0	0	E000 - E3FF	
C800 - CFFF	2	1	0	0	0	1	E400 - E7FF	
D000 - D7FF	3	0	1	0	0	2	E800 - EBFF	E6
D800 - DFFF	4	1	1	0	0	3	EC00 - EFFF	E12
E000 - E7FF	5	0	0	1	0	4	F000 - F3FF	
E800 - EFFF	6	1	0	1	0	5	F400 - F7FF	E20
F000 - F7FF	7	0	1	1	0	6	F800 - FBFF	
F800 - FFFF	9	1	1	1	0	7	FC00 - FFFF	E35
N/A	10	0	0	0	1		N/A	

F000



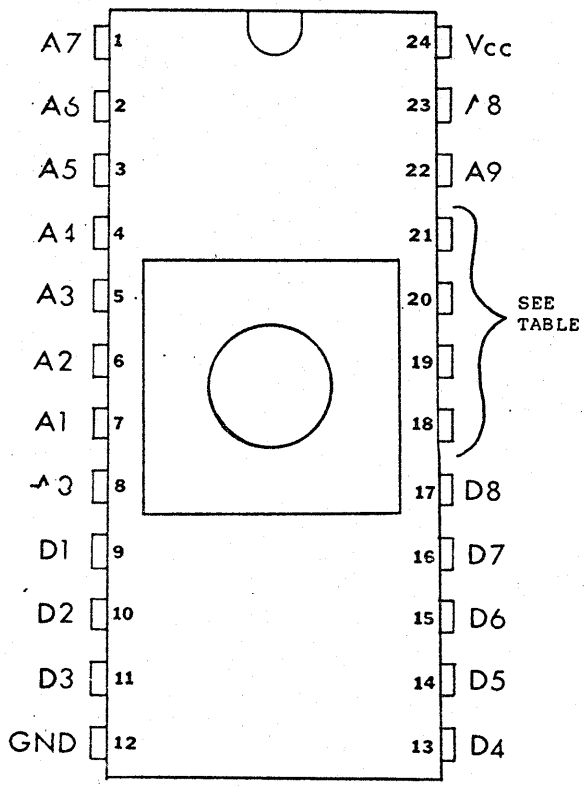
2708 LOGIC LAYOUT

UPDATED 3/22/78


-5
Jumpper Layout 3/8/18
X6
X5
X4
X3
X2
X1
+12
GND
E13
E39
E32
C.P.U.
Pin # B13 fun.
1 = A10
2 = A12
3 = A13
4 = A11
5 = R/W
6 = A15
7 = VMA&02
8 = GND
9 = +12
10 = A14
11 = -5
12 = n/a
13 = GND
14 = A9
15 = A8
16 = +5
Back Side C.P.U.#1

The Hardware Connection

by Hugh Turner



24 PIN MEMORY EPROMS, PROMS & RAMS

DEVICE	TYPE	MANUF	SIZE	PIN 18	PIN 19	PIN 20	PIN 21
2708	EPROM	TI	1K x 8	PROG	+12V	CS*(PE)	-5V
2508	EPROM	TI	1K x 8	PD/PROG	NC	CS*	Vpp
2758	EPROM	TI	1K x 8	PD/PROG	AR	CS*	Vpp
3628	PROM	INTEL	1K x 8	CS4	CS3	CS2*	CS1*/PROG
4801	RAM	MOSTEK	1K x 8	CS*	NC	OE*	WE*
4118	RAM	MOSTEK	1K x 8	CS*	L*	OE*	WE*
4008	RAM	TI	1K x 8	CS*	AR	OE*	WE*
2716	EPROM	INTEL	2K x 8	CS*/PROG	+12V	A10	-5V
2516/2716	EPROM	TI	2K x 8	CS*/PROG	A10	OE*	Vpp
3636	PROM	INTEL	2K x 8	CS3	CS2	CS1*/PROG	A10
4802/4016	RAM	MOS/TI	2K x 8	CS*	A10	OE*	WE*
2732	EPROM	INTEL	4K x 8	CS*	A10	OE*/PROG	A11
2532	EPROM	TI	4K x 8	A11	A10	PD/PROG	Vpp
4732	PROM	TI	4K x 8	A11	A10	CS	CS*/PROG
4764	PROM	TI	8K x 8	A11	A10	S*/PROG	A12

This is the first in a series of articles which I hope you will find useful if you do any building or designing. These data sheets can be copied or cut out and kept in a reference notebook.

Memory devices using 24 pins only have 4 pins which are used to define the device, the rest of these pins are all the same.

Please feel free to contact me if you have any hardware data yo would like to see in this collection. I can be reached at 276-1638.

CS*/S* = Chip Select (Low)
OE* = Output Enable (Low)
WE* = Write Enable (Low)
PD = Power Down
PROG/(PE) = Program Enable
Vpp = +25V (Program Voltage)
L* = LATCH (LOW)
AR = Array: If True Output VI0
If False Output VI1

EDN Software Note #55

6800 routine extracts square roots

Mike McBeath
Microface, Torrance, CA

The algorithm shown in the figure determines a 16-bit square root for any number up to 64k in essentially the time required for a 6800 μP to divide once. In the worst case, that time equals 514 CPU clock cycles.

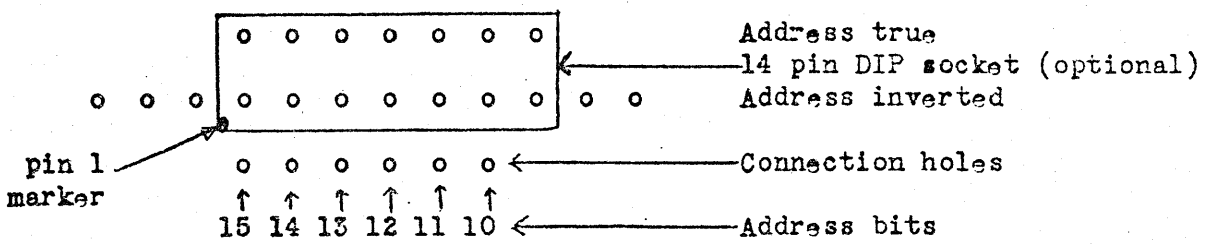
The routine stores the original number in HIBYTE and LOBYTE, then successively interpolates the square root two bits at a time. It begins with the number's two most significant digits and works its way down to the two least significant bits on its eighth and final loop. Accumulator A contains the running estimate, which after the last iteration represents the number's exact square root. EDN

```
                                NAM   SQRT
0000                HIBYTE      RMB   1
0001                LOBYTE      RMB   1
0002                DEVISR      RMB   1
0003                COUNT       RMB   1
1000                            ORG   $1000
1000 86 08          SQRT        LDA A #8
1002 97 03                      STA A COUNT
1004 4F                         CLR A
1005 5F                         CLR B
1006 79 00 01       SQRT1       ROL   LOBYTE
1009 79 00 00                   ROL   HIBYTE
100C 59                         ROL B
100D 79 00 01                   ROL   LOBYTE
1010 79 00 00                   ROL   HIBYTE
1013 59                         ROL B
1014 48                         ASL A
1015 97 02                      STA A DEVISR
1017 78 00 02                   ASL   DEVISR
101A D1 02                      CMP B DEVISR
101C 23 06                      BLS   SQRT2
101E 4C                         INC A
101F 7C 00 02                   INC   DEVISR
1022 D0 02                      SUB B DEVISR
1024 7A 00 03       SQRT2       DEC   COUNT
1027 2E DD                      BGT   SQRT1
1029 3F                         SWI
                                END
        NO ERROR(S) DETECTED
    SYMBOL TABLE:
COUNT   0003        DEVISR  0002
SQRT    1000        SQRT1   1006
HIBYTE  0000        LOBYTE  0001
SQRT2   1024
```

Interpolating two bits at a time, *this routine calculates a 16-bit number's square root in no more than 514 CPU clock cycles.*

SPHERE READ ONLY MEMORY BOARD

The SPHERE ROM/1 board is designed to provide up to 4K of read only memory program space using the commonly available 1702 PROM. The addressing on the board is fully selectable on 1K boundaries by the user. All four banks must be either address strapped or grounded - no banks' address select logic may be left open. If the addressing on your board will change frequently a 14 pin DIP socket may be installed at J1-J4 and 2 inch wire-wrap type wires in the adjoining connecting holes. These wires may then be pushed into the correct hole of J1-J4



To select an address, jumper each connection hole to the appropriate address selection hole. To make a 'don't care' bit (either on or off will do) leave the jumper wire from the connection hole open (no connection). It is for this reason that unused banks must be strapped down.

To select 1000 as the starting address of bank 1 (U13 1000, U12 1100, U11 1200, U10 1300) connect
- 15 to address inverted, pin 1 of J1
- 14 to address inverted, pin 2 of J1
- 13 to address inverted, pin 3 of J1
- 12 to address true, pin 11 of J1
- 11 to address inverted, pin 5 of J1
- 10 to address inverted, pin 6 of J1

To select DC00 as the starting address of bank 3 (U25 DC00, U24 DD00, U23 DE00, U22 DF00) connect
- 15 to address true, pin 14 of J3
- 14 to address true, pin 13 of J3
- 13 to address inverted, pin 3 of J3
- 12 to address true, pin 11 of J3
- 11 to address true, pin 10 of J3
- 10 to address true, pin 9 of J3

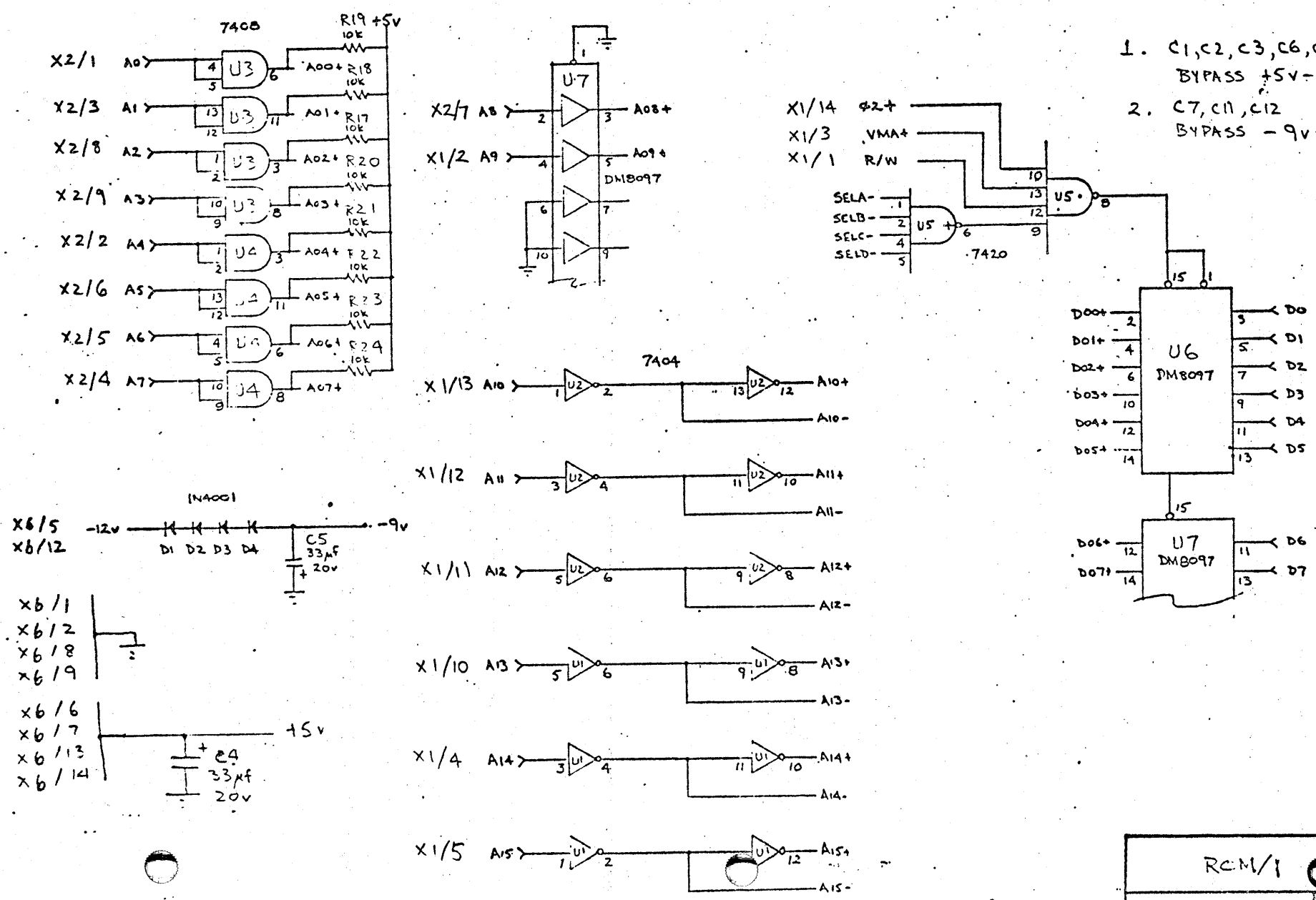
To strap a bank of ROM off, connect any (at least one) connection hole to a ground, available feed-throughs near pin 7 of U9, U15, and U21.

In selecting addresses, remember that below 1000 is dedicated to RAM and above E000 are I/O devices and system ROM's.

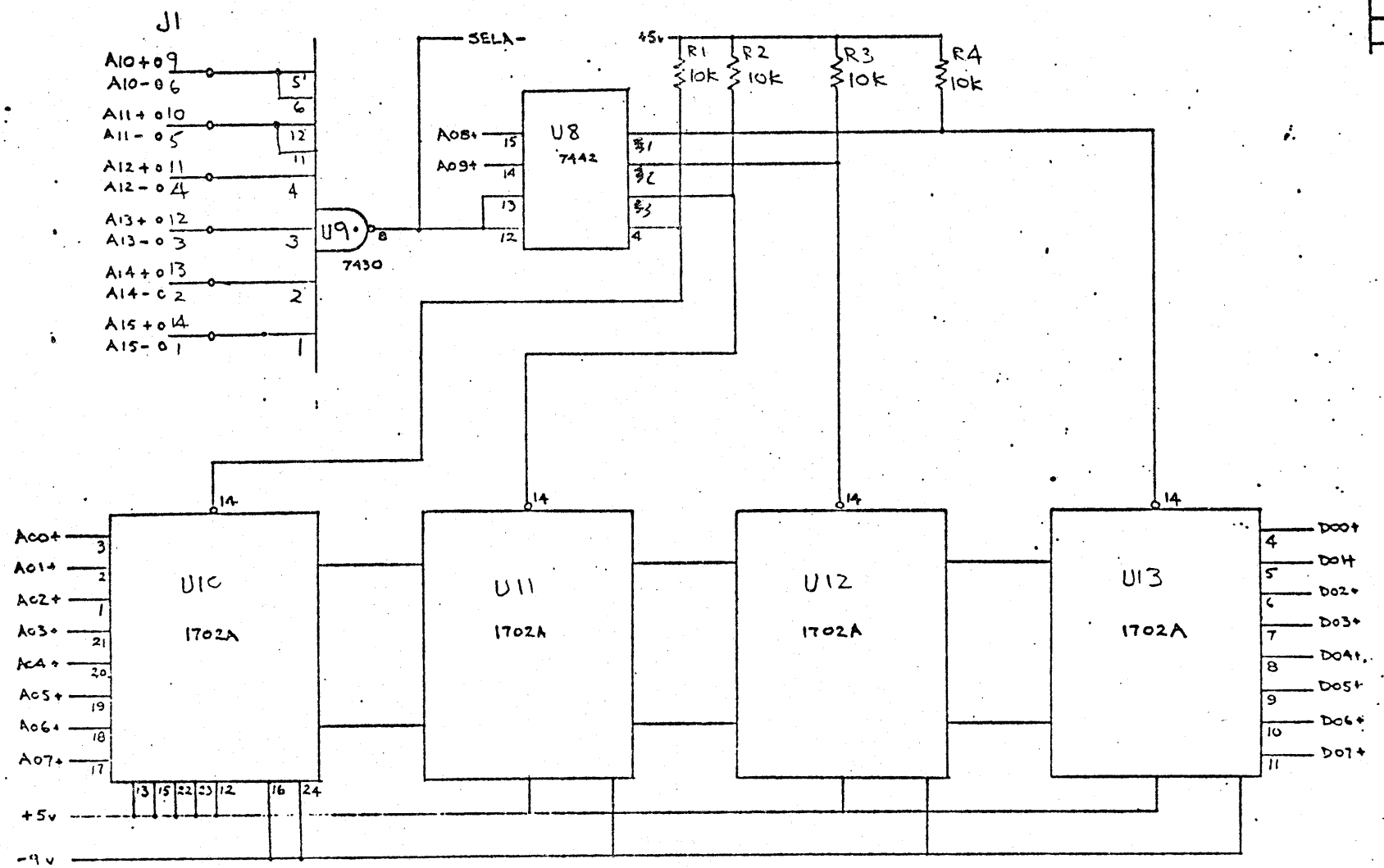
Good programming to you all.

Ernest Dixon

Ernie Dixon
SPHERE Corporation

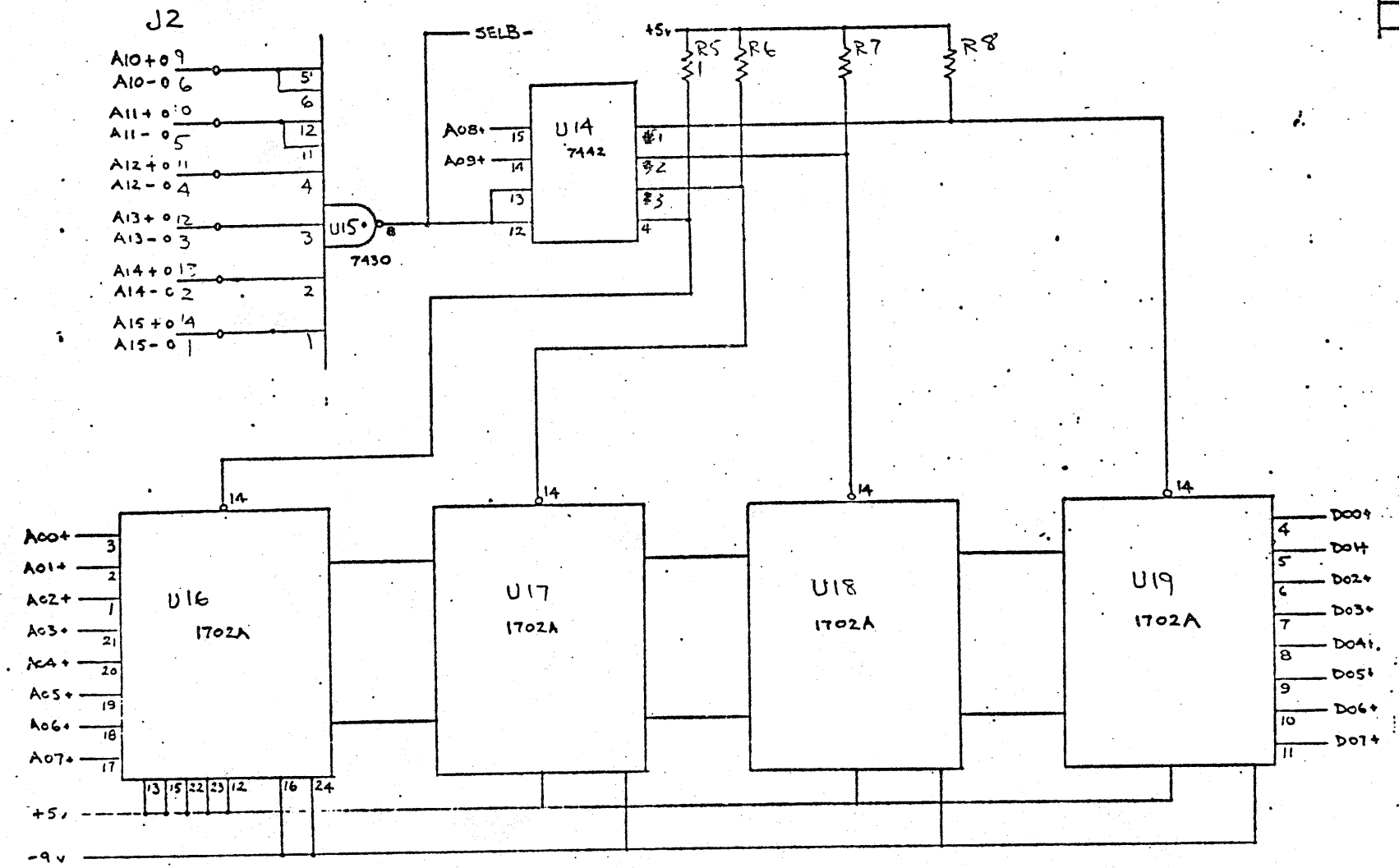

CHANGE NOTES
1. C1,C2,C3,C6,C8,C9,C10
BYPASS +5V-GND
2. C7,C11,C12
BYPASS -9V-GND
7408
R19 +5V
10K
X2/1 A0
X2/3 A1
X2/8 A2
X2/9 A3
X2/2 A4
X2/6 A5
X2/5 A6
X2/4 A7
U3
U4
A00+
A01+
A02+
A03+
A04+
A05+
A06+
A07+
R18
R17
R20
R21
R22
R23
R24
U7
X2/7 A8
X1/2 A9
A08+
A09+
DM8097
X1/14 Φ2+
X1/3 VMA+
X1/1 R/W
SELA-
SELB-
SELC-
SELD-
U5
7420
D00+
D01+
D02+
D03+
D04+
D05+
D06+
D07+
U6
DM8097
U7
DM8097
D0
D1
D2
D3
D4
D5
D6
D7
7404
X1/13 A10
X1/12 A11
X1/11 A12
X1/10 A13
X1/4 A14
X1/5 A15
A10+
A10-
A11+
A11-
A12+
A12-
A13+
A13-
A14+
A14-
A15+
A15-
1N4001
X6/5
X6/12
-12v
-9v
D1 D2 D3 D4
C5
33μf
20v
X6/1
X6/2
X6/8
X6/9
X6/6
X6/7
X6/13
X6/14
+5v
C4
33μf
20v
RCM/1
SHEET 1 OF 5

CHANGE NOTES



SHEET 2 OF

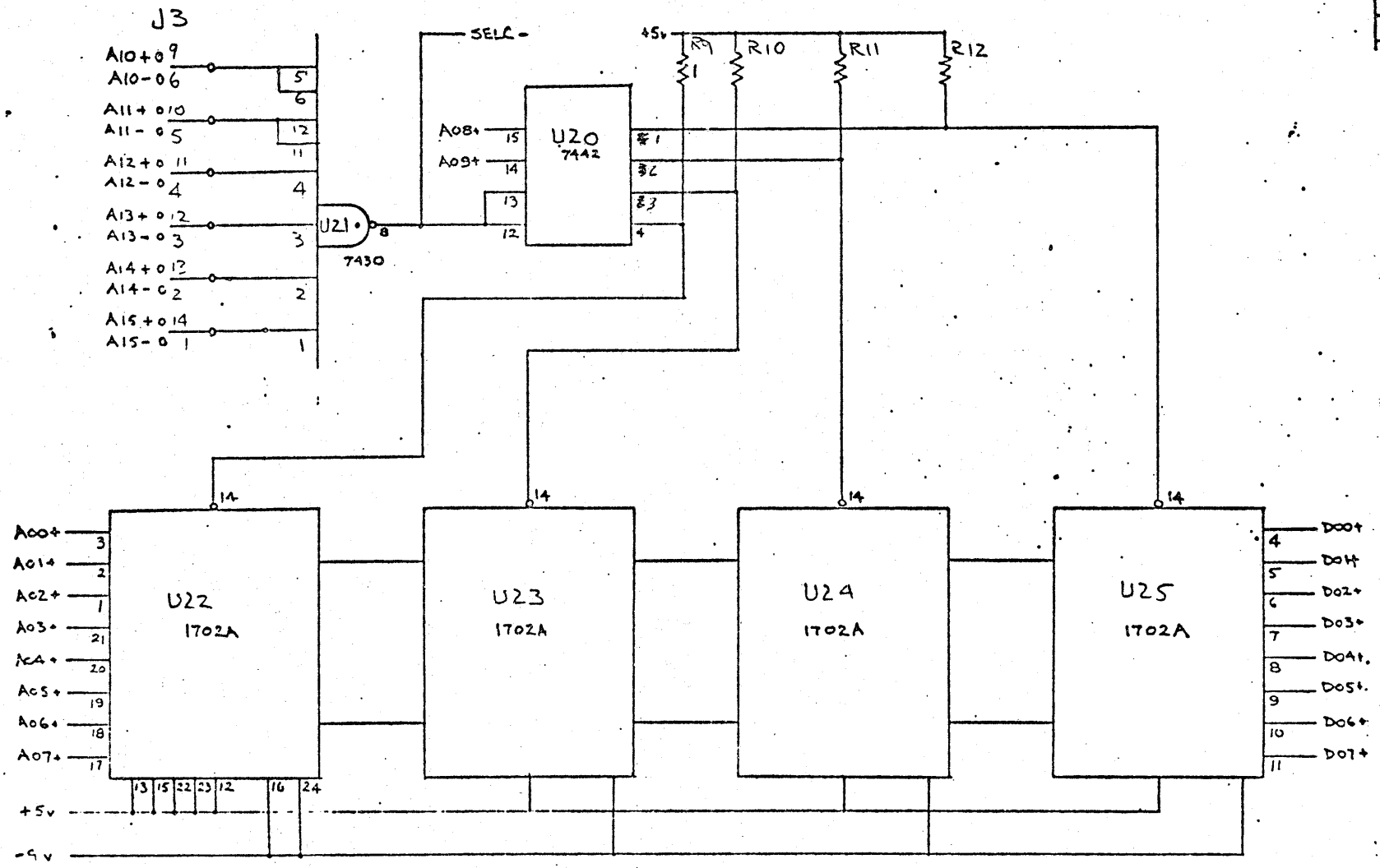
CHANGE NOTES



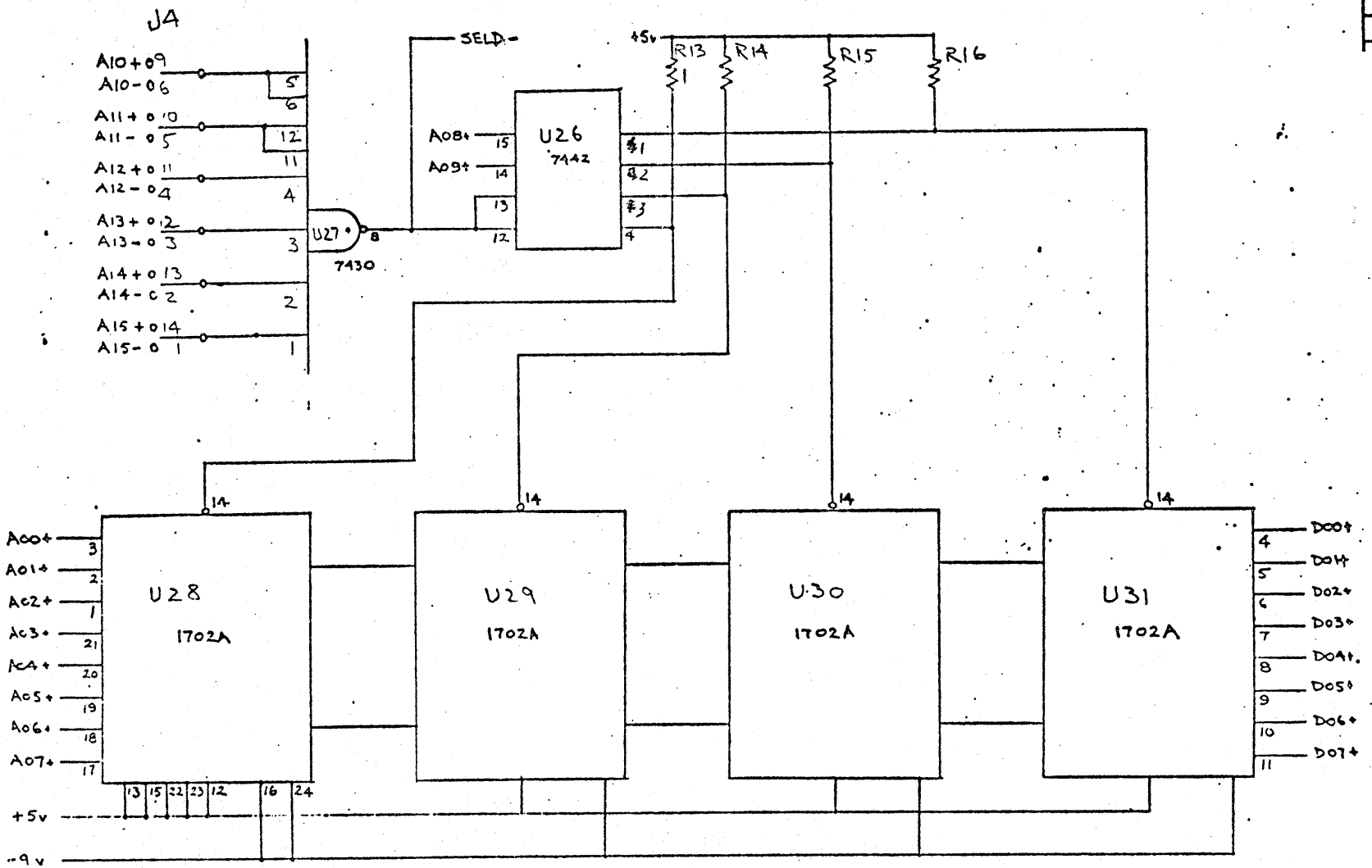
-21-

SHEET 2 OF 5

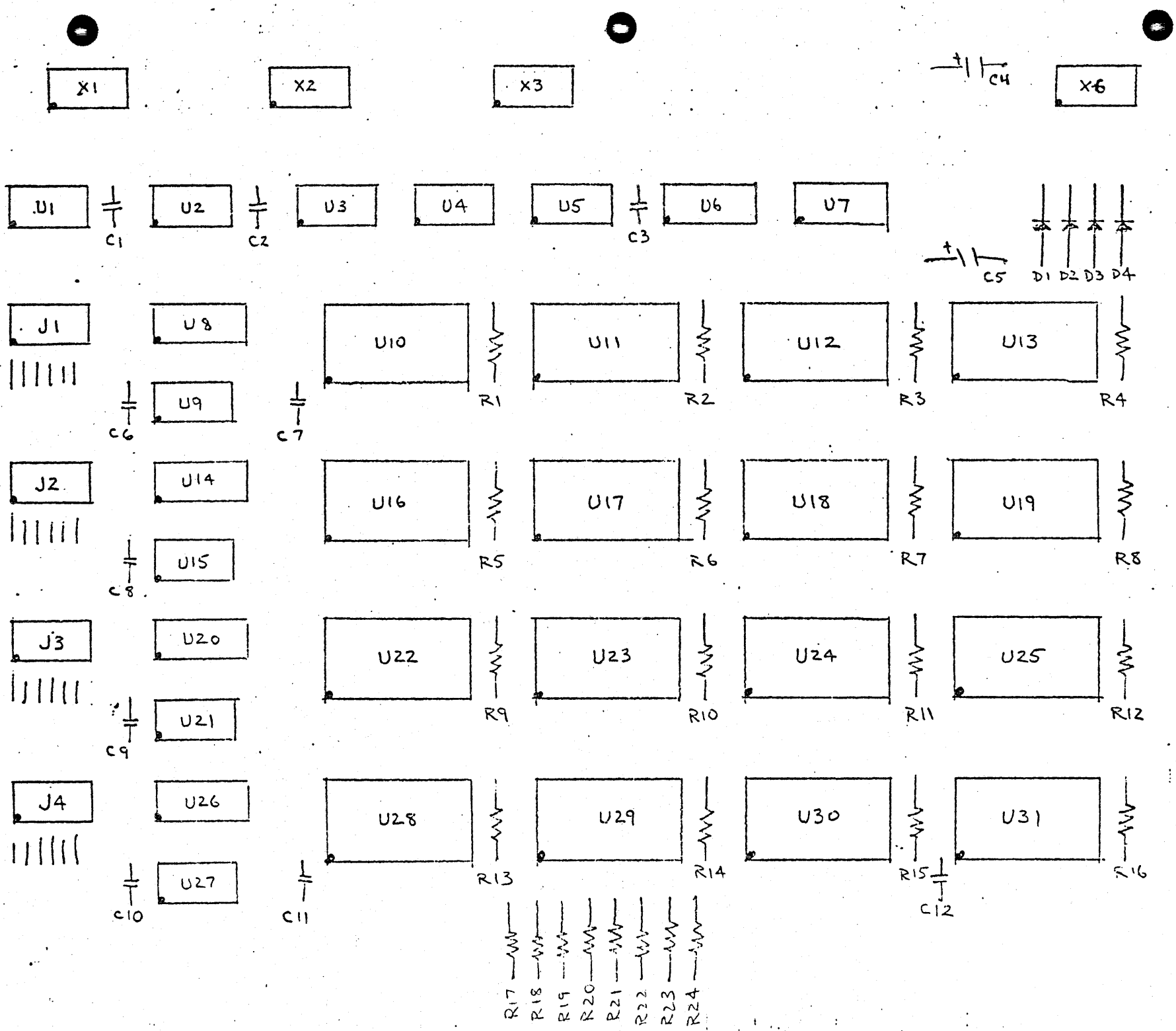
CHANGE NOTES



SHEET 1 OF



CHANGE NOTES


X1
X2
X3
C4
X6
U1
C1
U2
C2
U3
U4
U5
C3
U6
U7
C5
D1
D2
D3
D4
J1
U8
U9
C6
C7
U10
R1
U11
R2
U12
R3
U13
R4
J2
U14
U15
C8
U16
R5
U17
R6
U18
R7
U19
R8
J3
U20
U21
C9
U22
R9
U23
R10
U24
R11
U25
R12
J4
U26
U27
C10
C11
U28
R13
U29
R14
U30
R15
C12
U31
R16
R17
R18
R19
R20
R21
R22
R23
R24

PARTS FOR RCM/1 EHD (PCB ASSY)

#	QTY	TYPE	DESCRIPTION	LOC
1	2	7404	IC HEX INVERTER	U1, U2
2	2	7408	IC 2-INP QUAD AND	U3, U4
3	1	7420	IC 4-INP DUAL NAND	U5
4	4	7430	IC 8-INP NAND	U9, U15, U21, U27
5	4	7442	IC 4-10 DECODER	U8, U14, U20, U26
6	2	DM8097	IC HEX BUS DRIVER	U6, U7
7	16	1702A	IC 256X8 PROM	U(10-13), U(16-19), U(22-25), U(28-31)
8	4	IN4001	50v RECTIFIER DIODE	D1, D2, D3, D4
9	2	33µf/20v	CAPACITOR, ELECTROLYTIC	C4, C5
10	10	0.1µf/50v	CAPACITOR, CERAMIC	C1, C2, C3, C(6-12)
11	24	4.7K, ¼W, 10%	RESISTOR, CARBON COMP	R(1-24)
12	4, 4	14-pin sockets (DIP)		X1, X2, X3, X6, (J1, J2, J3, J4)
13	1	RCM/1 EHD	PCB	
14	16	24-pin DIP sockets		U(10-13), U(16-19), U(22-25), U(28-31)

UNDERLINED ITEMS NOT SUPPLIED

The Sort Routine

The Basic interpreter ~~is~~ can be equipped with an alphabetic sort command which should prove useful in business applications. One can sort 250 names in ten seconds. Full 16 bit arithmetic is used so you could theoretically sort 64,000 one bit strings.

The command name is SORT

You must dimension your array on the first line of the program.
ie: 10 DIM A$(100) To sort 100 items

The default string length is 32 bytes. Do not declare string= if you plan to use less than 32 bytes. You must use STRING= if you plan to use longer strings (max 128 bytes)

Jeff's Sort Enhancement

```
DE 22      Save STRAR
DF FE
DE 2C      GEI BASIC POINTER
BD 0609    READ VARIABLE OFF LINE
DE 42      POINT TO LOCATION WHICH
09         HOLDS LOCATION IN
09         TABLE
EE 00      GET LOCATION in X
DF 22      PUT IN 22
BD ROGER'S SORT
DE FE      RESTORE ORIGINAL 22
DF 22
39         EXIT
```

Comments about the sort: I tried Roger's sort out and found that it worked fast and well. Basic limited the number of strings that could be sorted at once to 256. To make the sorting better and to permit other arrays to be sorted in the same program, I wrote a mod of the thing. With my addition, you call your array as follows: SORT A$(1). You can have many arrays sorted; call them by their first member! The DIM does not any longer have to be on the first line of the program.

Jeff

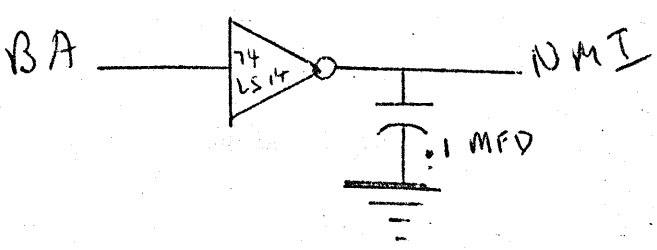
```
                                NAM   SMSORT-ROGER J. SPOTT APRIL 1981
                                OPT   PNT,XRF
                          *
                          * SHELL-METZNER SORT FOR CSS BASIC
                          *
0000                      MDATA EQU   X'B0'
0000                      KDATA EQU   X'B2'
0000                      JDATA EQU   X'B4'
0000                      IDATA EQU   X'B6'
0000                      LDATA EQU   X'B8'
0000                      SADDI EQU   X'BA'
0000                      SADDL EQU   X'BC'
0000                      STPTR EQU   X'BE'
0000                      STRST EQU   X'C0'
0000                      CNTR  EQU   X'C2'
0000                      ATEMP EQU   X'C4'
0000                      STRLN EQU   X'46'
0000                      STRAR EQU   X'22'
0000                      ARA   EQU   X'06'
0000                      DIVIDE EQU  X'FFAF'
0000                      MULTY EQU   X'FF93'
                          *
                          * SHELL-METZNER SORT FOR CSS BASIC
                          *
  4R 0000 DE 06           START LDX   ARA
   9 0002 DF C4                 STX   ATEMP
  11 0004 5F                    CLRB
  13 0005 0C                    CLC
  16 0006 96 46                 LDAA  STRLN       GET STRING LENGTH
  19 0008 9B 23                 ADDA  STRAR+1     ADD IT TO START LOC.
  23 000A 97 C1                 STAA  STRST+1     STORE LSB
  26 000C D9 22                 ADCB  STRAR       ADD WITH CARRY
  30 000E D7 C0                 STAB  STRST       STORE MSB
  34 0010 DE 22                 LDX   STRAR       GET NUMBER OF ENTRIES (N)
  39 0012 A6 02                 LDAA  02,X
  44 0014 E6 03                 LDAB  03,X
  48 0016 97 B1                 STAA  MDATA+1
  52 0018 D7 B0                 STAB  MDATA
  54 001A 86 02           POS1  LDAA  #2          DIVIDE M BY 2
  60 001C 7F 0006               CLR   ARA
  64 001F 97 07                 STAA  ARA+1
  67 0021 96 B1                 LDAA  MDATA+1
  70 0023 D6 B0                 LDAB  MDATA
  79 * 0025 BD FFAF             JSR   DIVIDE
  83 0028 97 B1                 STAA  MDATA+1
  87 002A D7 B0                 STAB  MDATA
  93 002C 7D 00B0               TST   MDATA       DOES M=0?
  97 002F 26 0A                 BNE   GETK        NO,GO ON
   6R 0031 7D 00B1              TST   MDATA+1
  10 0034 26 05                 BNE   GETK
   4R 0036 DE C4                LDX   ATEMP
   9 0038 DF 06                 STX   ARA
  14 003A 39                    RTS
  18 003B DE 22           GETK  LDX   STRAR       FOR K=N-M GET K
  23 003D A6 02                 LDAA  02,X
  28 003F E6 03                 LDAB  03,X
  30 0041 0C                    CLC
  33 0042 90 B1                 SUBA  MDATA+1     SUBTRACT M
  36 0044 D2 B0                 SBCB  MDATA
  40 0046 97 B3                 STAA  KDATA+1
  44 0048 D7 B2                 STAB  KDATA
  50 004A 7F 00B4               CLR   JDATA
  56 004D 7F 00B5               CLR   JDATA+1
```

```
 65  66     0053 DE B4          POS2    LDX   JDATA
 66  71     0055 DF B6                  STX   IDATA      LET I=J
 67  75     0057 20 02                  BRA   POS3
 68  79     0059 20 BF (001A)   POSI1   BRA   POS1
 69   3R    005B 96 B1          POS3    LDAA  MDATA+1
 70   6     005D D6 B0                  LDAB  MDATA      GET M
 71   8     005F 0C                     CLC
 72  11     0060 9B B7                  ADDA  IDATA+1    ADD IT TO I
 73  14     0062 D9 B6                  ADCB  IDATA
 74  18     0064 97 B9                  STAA  LDATA+1    STORE IT IN L
 75  22     0066 D7 B8                  STAB  LDATA
 76  25     0068 96 46                  LDAA  STRLN      COMPUT ADDR.FOR L STR.
 77  29     006A 97 07                  STAA  ARA+1
 78  32     006C 96 B9                  LDAA  LDATA+1
 79  35     006E D6 B8                  LDAB  LDATA
 80  44 *   0070 BD FF93                JSR   MULTY
 81  47     0073 90 46                  SUBA  STRLN
 82  49     0075 C2 00                  SBCB  #00
 83  51     0077 0C                     CLC
 84  54     0078 9B C1                  ADDA  STRST+1
 85  58     007A 97 BD                  STAA  SADDL+1
 86  61     007C D9 C0                  ADCB  STRST
 87  65     007E D7 BC                  STAB  SADDL
 88  68     0080 96 46                  LDAA  STRLN      COMPUT ADDR.FOR I STRING
 89  74     0082 7F 0006                CLR   ARA
 90  78     0085 97 07                  STAA  ARA+1
 91  81     0087 96 B7                  LDAA  IDATA+1
 92  84     0089 D6 B6                  LDAB  IDATA
 93  93 *   008B BD FF93                JSR   MULTY
 94  96     008E 90 46                  SUBA  STRLN
 95  98     0090 C2 00                  SBCB  #00
 96 100     0092 0C                     CLC
 97 103     0093 9B C1                  ADDA  STRST+1
 98 107     0095 97 BB                  STAA  SADDI+1
 99 110     0097 D9 C0                  ADCB  STRST
100 114     0099 D7 BA                  STAB  SADDI
101 118     009B 20 17                  BRA   COMP
102   2R    009D 5F             POS4    CLRB
103   4     009E 86 01                  LDAA  #01
104   7     00A0 9B B5                  ADDA  JDATA+1
105  10     00A2 D9 B4                  ADCB  JDATA
106  14     00A4 97 B5                  STAA  JDATA+1
107  18     00A6 D7 B4                  STAB  JDATA
108  21     00A8 D1 B2                  CMPB  KDATA
109  25     00AA 22 AD (0059)          BHI   POSI1
110   3R    00AC 91 B3                  CMPA  KDATA+1
111   7     00AE 22 A9 (0059)          BHI   POSI1
112  11     00B0 20 A1 (0053)          BRA   POS2
113  15     00B2 20 A7 (005B)  POSI3    BRA   POS3
114   3R    00B4 D6 46          COMP    LDAB  STRLN      COUNTER STRING LENGTH
115   7     00B6 DE BC                  LDX   SADDL      ADDR. OF L INTO INDEX RE
116  12     00B8 9F BE                  STS   STPTR      SAVE STACK
117  16     00BA 9E BA                  LDS   SADDI      ADDRESS OF I INTO STACK
118  20     00BC 34                     DES              POSITION STACK
119  24     00BD 32             NEXT    PULA             GET FIRST CHAR. OF STRI
120  29     00BE A1 00                  CMPA  00,X       COMPARE TO 1ST IN IND.RE
121  33     00C0 26 06                  BNE   TEST       FIND OUT WHICH IS BIGGE
122   2R    00C2 5A                     DECB             TEST NEXT CHARACTER
123   6     00C3 27 05                  BEQ   RESTR      END OF STRING
124   4R    00C5 08                     INX
125   8     00C6 20 F5 (00BD)          BRA   NEXT
126  12     00C8 22 04          TEST    BHI   EXCH       I>L SWITCH PLACES
127   4R    00CA 9E BE          RESTR   LDS   STPTR      NO SWITCH
```

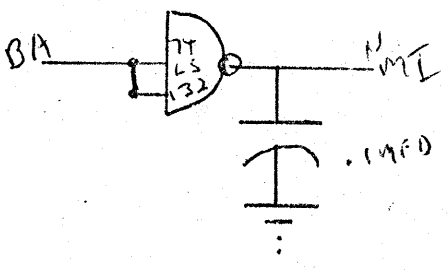
SORT Cont'd

```
129   3R  00CE 96 46          EXCH    LDAA  STRLN     GET STRING LENGTH
130    7  00D0 97 C2                  STAA  CNTR      PUT INTO COUNTER
131   11  00D2 DE BA                  LDX   SADDI     1ST STRING ADDRESS (I)
132   15  00D4 9E BC                  LDS   SADDL     2ND STRING ADDRESS(L)
133   19  00D6 34                     DES
134   23  00D7 32                PULL PULA            GET 1ST CHARACTER INTO A
135   28  00D8 E6 00                  LDAB  00,X      2ND CHARACTER INTO B
136   34  00DA A7 00                  STAA  00,X      PUT A WHERE B CAME FROM
137   38  00DC 08                     INX             MOVE INDEX POINTER
138   42  00DD 37                     PSHB            PUT B WHERE A CAME FROM
139   46  00DE 31                     INS             MOVE STACK POINTER
140   52  00DF 7A 00C2                DEC   CNTR
141   56  00E2 26 F3 (00D7)           BNE   PULL      NOT DONE YET
142   4R  00E4 9E BE                  LDS   STPTR     RESET STACK
143    7  00E6 96 B7                  LDAA  IDATA+1
144   10  00E8 D6 B6                  LDAB  IDATA
145   13  00EA D1 B0                  CMPB  MDATA
146   17  00EC 22 06                  BHI   SUB
147   21  00EE 27 02                  BEQ   LSB
148   25  00F0 20 AB (009D)           BRA   POS4
149   3R  00F2 91 B1             LSB  CMPA  MDATA+1
150    7  00F4 22 04                  BHI   SUB
151   11  00F6 27 BA (00B2)           BEQ   POSI3
152   15  00F8 20 A3 (009D)           BRA   POS4
153   2R  00FA 0C                SUB  CLC
154    5  00FB 90 B1                  SUBA  MDATA+1
155    8  00FD D2 B0                  SBCB  MDATA
156   12  00FF D7 B6                  STAB  IDATA
157   16  0101 97 B7                  STAA  IDATA+1
158   20  0103 20 AD (00B2)           BRA   POSI3
159       0105                        END
```

BA TRIGGERS NMI (Ignores Refresh)



OR



use Schmidt trigger 74LS14 or 74LS13.

THIS PROGRAM WILL PROVIDE A PITCH AT LINE Ø FROM THE PIA WHICH CORRESPONDS TO THE KEY PUSHED, ANY KEY OF THE MIDDLE ROW. IF I REMEMBER CORRECTLY, "A" (the key) GIVES A FAIRLY LOW MUSICAL "C", S GIVES C#, D GIVES THE NOTE D, F GIVES D#, ETC. THEY ARE ALL USED UP AND INCLUDING THE VERTICAL LINE, BUT NOT DELETE.

```
 2000  B6 LDA A $F043
R20B2G
>  2003  84 AND A #$FB          ..
>  2005  B7 STA A $F043        ..C
>  2008  86 LDA A #$01          ..
>  200A  B7 STA A $F042        ..B
>  200D  B6 LDA A $F043        ..C
>  2010  8A ORA A #$04          ..
>  2012  B7 STA A $F043        ..C To here, initialize PIA
>  2015  F6 LDA B $F041        ..A key test ( should say, key down test)
>  2018  C5 BIT B #$40          .@
>  201A  27 BEQ   $F9     2015 '.
>  201C  F6 LDA B $F040        ..@ Load from PIA which key was pressed
>  201F  D7 STA B $C0           ..
>  2021  20 BRA   $02     2025 ..
>  2023  20 BRA   $F0     2015 .. (island)
>  2025  C1 CMP B #$7C          .. (Is it the rightmost key- vertical line)
>  2027  26 BNE   $05     202E &.
>  2029  CE LDX   #$0046       ..F Yes? LDX 46
>  202C  20 BRA   $6E     209C .. And go to tone generator
>  202E  C1 CMP B #$7E          .. Is it tilda?
>  2030  26 BNE   $05     2037 &.
>  2032  CE LDX   #$004A       ..J if so, LDX w/ 4A
>  2035  20 BRA   $65     209C ..    and go to tone generator
>  2037  C1 CMP B #$3A          .:   Colon
>  2039  26 BNE   $05     2040 &.
>  203B  CE LDX   #$004F       ..O   LDX w/ 4F ( the higher the #, the longer it takes
>  203E  20 BRA   $5C     209C .\        to count down, so the lower the pitch)
>  2040  C1 CMP B #$3B          .;
>  2042  26 BNE   $05     2049 &.
>  2044  CE LDX   #$0054       ..T
>  2047  20 BRA   $53     209C .S
>  2049  C1 CMP B #$4C          .L
>  204B  26 BNE   $05     2052 &.
>  204D  CE LDX   #$0059       ..Y  (G#)
>  2050  20 BRA   $4A     209C .J
>  2052  C1 CMP B #$4B          .K
>  2054  26 BNE   $05     205B &.
>  2056  CE LDX   #$005F       ..-  (G)
>  2059  20 BRA   $41     209C .A
>  205B  C1 CMP B #$4A          .J
>  205D  26  BNE   $05     2064 &.
>  205F  CE LDX   #$0064       ...
>  2062  20 BRA   $38     209C .8
>  2064  C1 CMP B #$48          .H
>  2066  26 BNE   $05     206D &.
>  2068  CE LDX   #$006A       ...
>  206B  20 BRA   $2F     209C ./
>  206D  C1 CMP B #$47          .G
>  206F  26 BNE   $05     2076 &.
>  2071  CE LDX   #$0071       ...
>  2074  20 BRA   $26     209C .&
>  2076  C1 CMP B #$46          .F
>  2078  26 BNE   $05     207F &.
>  207A  CE LDX   #$0077       ...
```

Musical Tone Generator Cont'd

```
>  207D  20 BRA    $1D      209C ..
>  207F  C1 CMP B #$44            .D
>  2081  26 BNE    $05      2088 &.
>  2083  CE LDX    #$0080         ...
>  2086  20 BRA    $14      209C ..
>  2088  C1 CMP B #$41            .A
>  208A  26 BNE    $05      2091 &.
>  208C  CE LDX    #$0090         ...
>  208F  20 BRA    $0B      209C ..
>  2091  C1 CMP B #$53            .S
>  2093  26 BNE    $05      209A &.
>  2095  CE LDX    #$0089         ...
>  2098  20 BRA    $02      209C ..
>  209A  20 BRA    $87      2023 ..
>  209C  DF STX    $C1            ..
>  209E  DE LDX    $C1            ..
>  20A0  B7 STA A  $F042          ..B
>  20A3  09 DEX                   .
>  20A4  26 BNE    $FD      20A3 &.
>  20A6  88 EOR A  #$01           ..
>  20A8  F6 LDA B  $F040          ..@
>  20AB  D1 CMP B  $C0            ..
>  20AD  27 BEQ    $EF      209E '.
>  20AF  7E JMP    $201C          ...
>
```

Begin the tone generator - store the starting no.

Store A in the PIA - output
DEC X

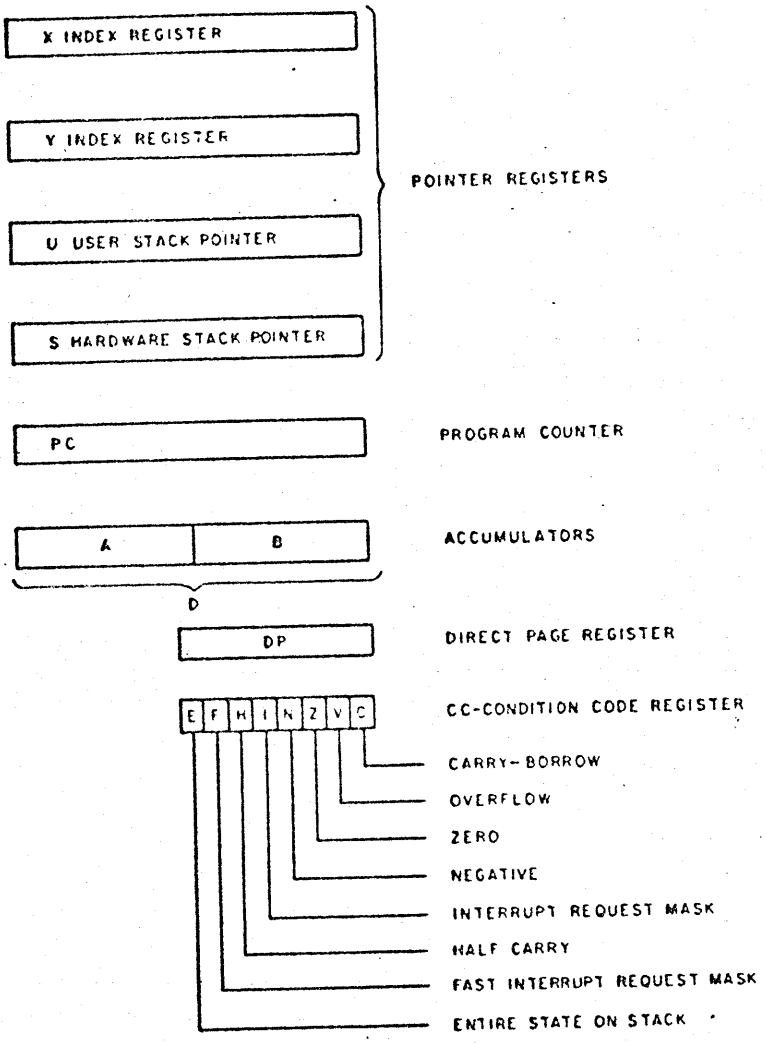
If A were high, go low, if low, go high; switches to form square wave output.

```
B6 F0 43 84 FE B7 F0 43 86 01 B7 F0 42 B6 F0 43
8A 04 B7 F0 43 F6 F0 41 C5 40 27 F9 F6 F0 40 D7
C0 20 02 20 F0 C1 7C 26 05 CE 00 46 20 6E C1 7E
26 05 CE 00 4A 20 65 C1 3A 26 05 CE 00 4F 20 5C
C1 3B 26 05 CE 00 54 20 53 C1 4C 26 05 CE 00 59
20 4A C1 6B 26 05 CE 00 5F 20 41 C1 4A 26 05 CE
00 64 20 38 C1 48 26 05 CE 00 6A 20 2F C1 47 26
05 CE 00 71 20 26 C1 46 26 05 CE 00 77 20 1D C1
44 26 05 CE 00 80 20 14 C1 41 26 05 CE 00 90 20
0B C1 53 26 05 CE 00 89 20 02 20 87 DF C1 DE C1
B7 F0 42 09 26 FD 83 01 F6 F0 40 D1 C0 27 EF 7E
20 1C 00
```

3.0 SOFTWARE ARCHITECTURE

3.1 6809 PROGRAMMING MODEL

The 6809 contains four 8-bit registers and five 16-bit registers which are visible to the programmer:



The Double-Accumulator D consits of the two 8-bit accumulators concatenated A:B. The A-register is the MS byte of the pair while the B-register is the LS byte.

3.1.1 Accumulators (A, B & D)

The A and B registers are general purpose accumulators used for arithmetic calculations and data manipulation. With the exception of ABX, DAA and 16-bit operations, the two accumulators are completely interchangeable. In the catenated form the A-register is the MS byte of the pair thru forming the 16-bit Double Accumulator, or D-register.

3.1.2 Direct Page Register (DP)

The Direct Page register defines the MS byte to be used in the direct mode of addressing; the DP is catenated with the byte following the direct-mode op code to form a 16-bit effective address. The DP will be initialized to $00 by $\overline{\text{RESET}}$ for 6800 compatibility.

3.1.3 Condition Code Register (CC)

The Condition Code register defines the state of the processor flags at any given time. The bits in the CC are:

B7	B6	B5	B4	B3	B2	B1	B0
E	F	H	I	N	Z	V	C

Bit 5 and bits 3-0 are set as the result of instructions that manipulate data; for details, see condition code section for each instruction.

3.1.4 Index Register (X, Y)

The index registers are used in indexed mode addressing. They provide a 16-bit address to be added to an optional offset (of up to 16-bits) for indexed instructions; the result of the addition is the effective address of the instruction. For more details see the section on addressing modes. The X and Y registers are essentially equivalent in usage and support the same instructions. Because automatic pre-increment and post-decrement options are available on indexed-mode operations, these registers may be used to easily implement software stacks, queues, and buffers.

3.1.5 Stack Pointers (U , S)

The stack pointer registers contain addresses that point to the top of a push-down/pop-up stack. Data and machine state can be pushed onto the stack (stored at the next memory address to that "pointed" to by the U or S) or pulled from the stack in a last-in first-out manner. Pushes decrement the stack pointer before the data is stored while pulls increment the stack pointer after the data is recovered; the stack pointers point at the last byte placed on the stack. The S is used by the hardware to automatically store subset or entire machine states during subroutines and interrupts. The User Stack (U) is controlled exclusively by the programmer and can be used to pass arguments to and from subroutines. Both the U and S have the same indexed-mode addressing capabilities as

3.1.5 (Continued)

the X and Y index registers; the stack pointers are enhanced index registers (although the operation as LEA is slightly different on the stack registers). This allows the 6809 to be used efficiently as a stack processor, greatly enhancing its ability to support higher level languages.

3.1.6 Program Counter (PC)

The PC is used by the hardware to point to the next instruction to be executed by the processor. Limited indexed-mode addressing is available on the PC (i.e., auto-increment/decrement is not available). For notational convenience the description of each instruction assumes that the program counter points one location past the last byte of the op code, as it would after decoding the instruction. As additional bytes are used by the instruction the PC always points to the next unused byte.

EXAMPLE: The branch instructions are available in either short or long forms; in general the short form takes a one-byte opcode, while the long form takes two bytes. After decoding the opcode, the PC points at either a one- (short branch) or two-byte (long) immediate value, which is taken into the machine for addition to the PC. If the branch is not taken, the addition never happens and the PC remains pointing to the next instruction. Indexed-mode instructions also have variable length fields.

6809 INDEXED ADDRESSING

TYPE	FORMS	NON - INDIRECT				INDIRECT			
		SOURCE	POST- BYTE	$\overset{+}{\sim}$	$\overset{+}{\#}$	SOURCE	POST- BYTE	$\overset{+}{\sim}$	$\overset{+}{\#}$
CONSTANT OFFSET FROM R	NO OFFSET	,R	1RR00100	0	0	[,R]	1RR10100	3	0
	5-BIT OFFSET	n,R	ORRnnnnn	1	0	defaults to 8-bit			
	8-BIT OFFSET	n,R	1RR01000	1	1	[n,R]	1RR11000	4	1
	16-BIT OFFSET	n,R	1RR01001	4	2	[n,R]	1RR11001	7	2
ACCUMULATOR OFFSET FROM R	A-REGISTER OFFSET	A,R	1RR00110	1	0	[A,R]	1RR10110	4	0
	B-REGISTER OFFSET	B,R	1RR00101	1	0	[B,R]	1RR10101	4	0
	D-REGISTER OFFSET	D,R	1RR01011	4	0	[D,R]	1RR11011	7	0
AUTO-INCREMENT/ -DECREMENT R	INCREMENT BY 1	,R+	1RR00000	2	0	not allowed			
	INCREMENT BY 2	,R++	1RR00001	3	0	[,R++]	1RR10001	6	0
	DECREMENT BY 1	,-R	1RR00010	2	0	not allowed			
	DECREMENT BY 2	,--R	1RR00011	3	0	[,--R]	1RR10011	6	0
CONSTANT OFFSET FROM PC	8-BIT OFFSET	n,PCR	1XX01100	1	1	[n,PCR]	1XX11100	4	1
	16-BIT OFFSET	n,PCR	1XX01101	5	2	[n,PCR]	1XX11101	8	2
EXTENDED		use non-indexed				[n]	10011111	5	2

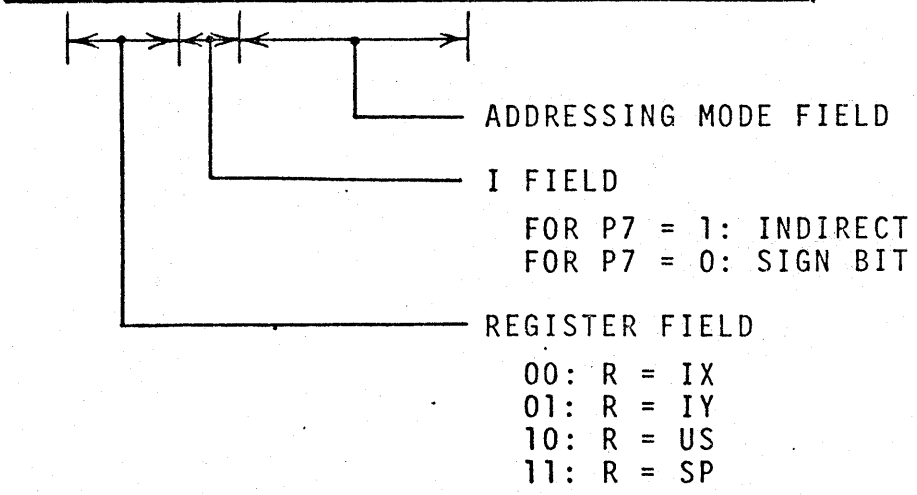
Figure 4: Indexed Addressing Modes. All instructions with indexed addressing have a base size and number of cycles. The $\overset{+}{\sim}$ and $\overset{+}{\#}$ columns indicate the number of additional cycles and bytes for the particular variation. The post byte opcode is the byte that immediately follows the normal opcode.

3.12 INDEXED-MODE POST-BYTE

POST BYTE REGISTER

BIT ASSIGNMENTS

POST-BYTE REGISTER BIT								INDEXED ADDRESSING MODE
7	6	5	4	3	2	1	0	
1	X	X	X	0	0	0	1	,R++
1	X	X	0	0	0	0	0	,R+
1	X	X	0	0	0	1	0	,-R
1	X	X	X	0	0	1	1	,--R
1	X	X	X	0	1	0	0	EA=(R ± 0 OFFSET)
1	X	X	X	0	1	0	1	EA=(R ± ACCB OFFSET)
1	X	X	X	1	0	0	0	EA=(R±7BIT OFFSET)
1	X	X	X	1	0	0	1	EA=(R±15 BIT OFFSET)
1	X	X	X	1	1	0	0	EA=(PC ±7 BIT OFFSET)
1	X	X	X	1	1	0	1	EA=(PC ±15 BIT OFFSET)
0	X	X	X	X	X	X	X	EA=(R±4 BIT OFFSET)
1	X	X	X	0	1	1	0	EA=(R±ACCA OFFSET)
1	X	X	X	1	0	1	1	EA=(R±D OFFSET)
1	X	X	1	1	1	1	1	EA=(ADDRESS)



3.14 BRANCH GROUPS

Simple Conditional Branches

Condition	Complement
BEQ {Z=1}	BNE
BMI {N=1}	BPL
BCS {C=1}	BCC
BVS {V=1}	BVC

Signed Conditional Branches

Condition	Complement
BGT $\{(\overline{N \oplus V}) \wedge \overline{Z}=1\}$	BLE
BGE $\{(\overline{N \oplus V})=1\}$	BLT
BEQ {Z=1}	BNE
BLE $\{(N \oplus V) \vee Z=1\}$	BGT
BLT $\{(N \oplus V)=1\}$	BGE

Unsigned Conditional Branches*

Condition	Complement
BHI $\{(\overline{C} \wedge \overline{Z})=1\}$	BLS
BHS $\{\overline{C}=1\}$	BLO
BEQ {Z=1}	BNE
BLS $\{C \vee Z=1\}$	BHI
BLO {C=1}	BHS

* Not useful, in general, after INC/DEC, LD/ST, TST/CLR/COM.

ABX	Add B-register to X-register unsigned
ADCA,ADCB	Add memory to accumulator with carry
ADDA,ADDB	Add memory to accumulator
ANDA,ANDB	And memory with accumulator
ANDCC	And immediate with condition code register
ASLA,ASLB,ASL	Arithmetic shift left accumulator or memory
ASRA,ASRB,ASR	Arithmetic shift right accumulator or memory
BITA,BITB	Bit test memory with accumulator
CLRA,CLRB,CLR	Clear accumulator or memory
CMPA,CMPB	Compare memory with accumulator
COMA,COMB,COM	Complement accumulator or memory
DAA	Decimal Adjust A-accumulator
DECA,DECB,DEC	Decrement accumulator or memory
EORA,EORB	Exclusive or memory with accumulator
EXG R1,R2	Exchange R1 with R2
INCA,INCB,INC	Increment accumulator or memory
LDA,LDB	Load accumulator from memory
LSLA,LSLB,LSL	Logical shift left accumulator or memory
LSRA,LSRB,LSR	Logical shift right accumulator or memory
MUL	Unsigned multiply (8 bit x 8 bit = 16 bit)
NEGA,NEGB,NEG	Negate accumulator or memory
ORA,ORB	Or memory with accumulator
ORCC	Or immediate with condition code register
PSHS {register}$^{8}_{0}$	Push register(s) on hardware stack
PSHU {register}$^{8}_{0}$	Push register(s) on user stack
PULS {register}$^{8}_{0}$	Pull register(s) from hardware stack
PULU {register}$^{8}_{0}$	Pull register(s) from user stack
ROLA,ROLB,ROL	Rotate accumulator or memory left
RORA,RORB,ROR	Rotate accumulator or memory right
SBCA,SBCB	Subtract memory from accumulator with borrow
STA,STB	Store accumulator to memory
SUBA,SUBB	Subtract memory from accumulator
TSTA,TSTB,TST	Test accumulator or memory
TFR R1,R2	Transfer register R1 to register R2

FIGURE 1 8-BIT OPERATIONS

ADDD	Add to D accumulator
SUBD	Subtract from D accumulator
LDD	Load D accumulator
STD	Store D accumulator
CMPD	Compare D accumulator
LDX,LDY,LDS,LDU	Load pointer register
STX,STY,STS,STU	Store pointer register
CMPX,CMPY,CMPU,CMPS	Compare pointer register
LEAX,LEAY,LEAS,LEAU	Load effective address into pointer register
SEX	Sign Extend
TFR register,register	Transfer register to register
EXG register,register	Exchange register to register
PSHS (register)$^{8}_{0}$	Push register(s) onto hardware stack
PSHU (register)$^{8}_{0}$	Push register(s) onto user stack
PULS (register)$^{8}_{0}$	Pull register(s) from hardware stack
PULU (register)$^{8}_{0}$	Pull register(s) from user stack

FIGURE 2 16-BIT OPERATIONS

0,R	indexed with zero offset
[0,R]	indexed with zero offset indirect
,R+	auto increment by 1
,R++	auto increment by 2
[,R++]	auto increment by 2 indirect
,-R	auto decrement by 1
,--R	auto decrement by 2
[,--R]	auto decrement by 2 indirect
n,P	indexed with signed n as offset (n=5,8, or 16-bits)
[n,P]	indexed with signed n as offset indirect
A,R	indexed with accumulator A as offset
[A,R]	indexed with accumulator A as offset indirect
B,R	indexed with accumulator B as offset
[B,R]	indexed with accumulator B as offset indirect
D,R	indexed with accumulator D as offset
[D,R]	indexed with accumulator D as offset indirect

R = X, Y, U or S

P = PC, X, Y, U or S

FIGURE 3 INDEXED ADDRESSING MODES

BCC,LBCC	Branch if carry clear
BCS,LBCS	Branch if carry set
BEQ,LBEQ	Branch if equal
BGE,LBGE	Branch if greater than or equal (signed)
BGT,LBGT	Branch if greater (signed)
BHI,LBHI	Branch if higher (unsigned)
BHS,LBHS	Branch if higher or same (unsigned)
BLE,LBLE	Branch if less than or equal (signed)
BLO,LBLO	Branch if lower (unsigned)
BLS,LBLS	Branch if lower or same (unsigned)
BLT,LBLT	Branch if less than (signed)
BMI,LBMI	Branch if minus
BNE,LBNE	Branch is not equal
BPL,LBPL	Branch if plus
BRA,LBRA	Branch always
BRN,LBRN	Branch never
BSR,LBSR	Branch to subroutine
BVC,LBVC	Branch if overflow clear
BVS,LBVS	Branch if overflow set

FIGURE 4 RELATIVE SHORT AND LONG BRANCHES

CWAI	Clear condition code register bits and wait for interrupt
NOP	No-operation
JMP	Jump
JSR	Jump to subroutine
RTI	Return from interrupt
RTS	Return from subroutine
SEX	Sign extend B-register into A-register
SWI,SWI2,SWI3	Software interrupts
SYNC	Synchronize with interrupt line

FIGURE 5 MISCELLANEOUS INSTRUCTIONS