Robotron Autopsy

Learning Hardware via Software

Jared Boone

Open Source Bridge - June 18, 2013

Saturday, June 22, 13

My name is Jared Boone, and I design open-source hardware for a living. And I want to introduce more people to hardware, so maybe, someday, all of you can hack on hardware, too.

Chronulator

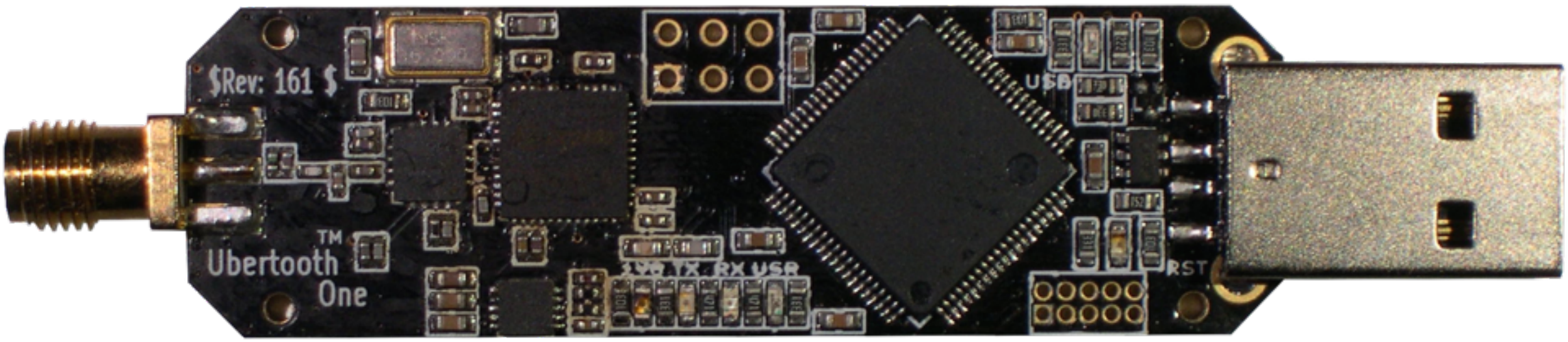
Kraig Sederquist

sharebrained.com/chronulator/

I've been involved to varying degrees in a few open-source hardware projects. There's a product I've sold for a few years now called the Chronulator. It's a simple clock kit that is great for learning how to solder.

I've also worked on other people's projects -- mostly related to wireless security.

Ubertooth



greatscottgadgets.com/ubertoothone/

I helped with the hardware design and coding on a Bluetooth testing tool called Ubertooth.

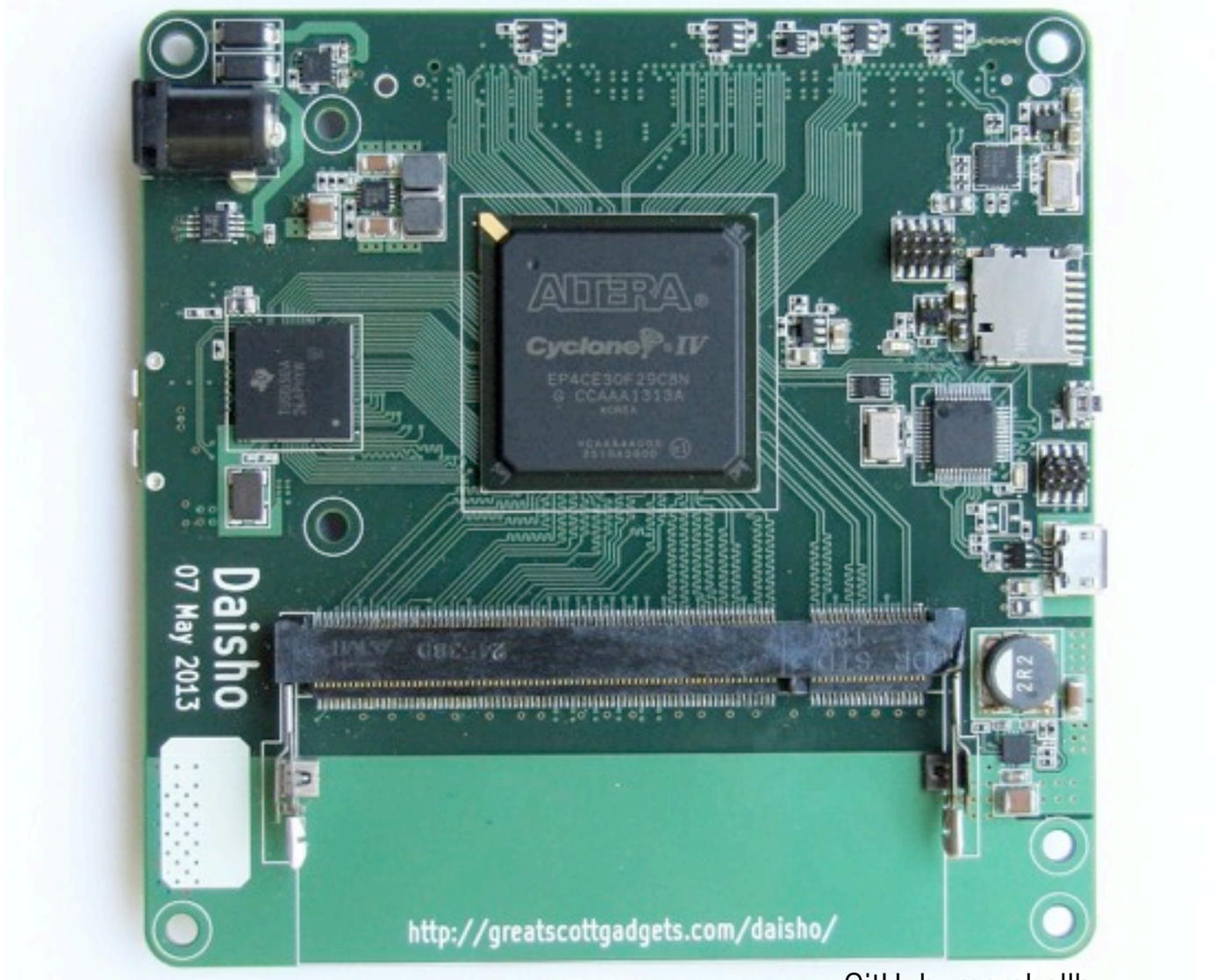
HackRF

greatscottgadgets.com/hackrf/

GitHub: fd0

I contributed to the hardware architecture and code for a software-defined radio called HackRF.

Daisho



GitHub: marshallh

greatscottgadgets.com/daisho/

Saturday, June 22, 13

And I'm doing a bunch of hardware design on a new project called Daisho, which is another security-focused tool for evaluating high speed communication protocols like USB Super Speed, HDMI, and Gigabit Ethernet.

Mike Ossmann

GreatScottGadgets.com

Those three security-related projects are the brainchildren of Mike Ossmann, who will be speaking at this conference tomorrow at 1:30 about the HackRF radio and how you too can play with radio technology, even if you have no hardware experience.

I, on the other hand, will give you a bit of hardware experience by building on your software experience.

arcadenostalgia.katorlegaz.com

And we'll do it by taking apart this thing. This is Robotron: 2084, an arcade game released in 1982. This is back when most video games were the kind you dropped quarters in to. The game is built into a giant wood cabinet with excellent graphics plastered all over the exterior. It has a big color cathode ray tube display, two joysticks, and an integrated speaker for sound effects.

Saturday, June 22, 13

The graphics are pretty clunky by modern standards. But Robotron is also proof that great gameplay can transcend hardware limitations.

Saturday, June 22, 13

The backstory on the game is that the Robotrons are robots that evolved to the point they see humans as inefficient, and naturally conclude they must be destroyed. Your job, as the mutant savior, is to destroy the robots and save the last human family from destruction.

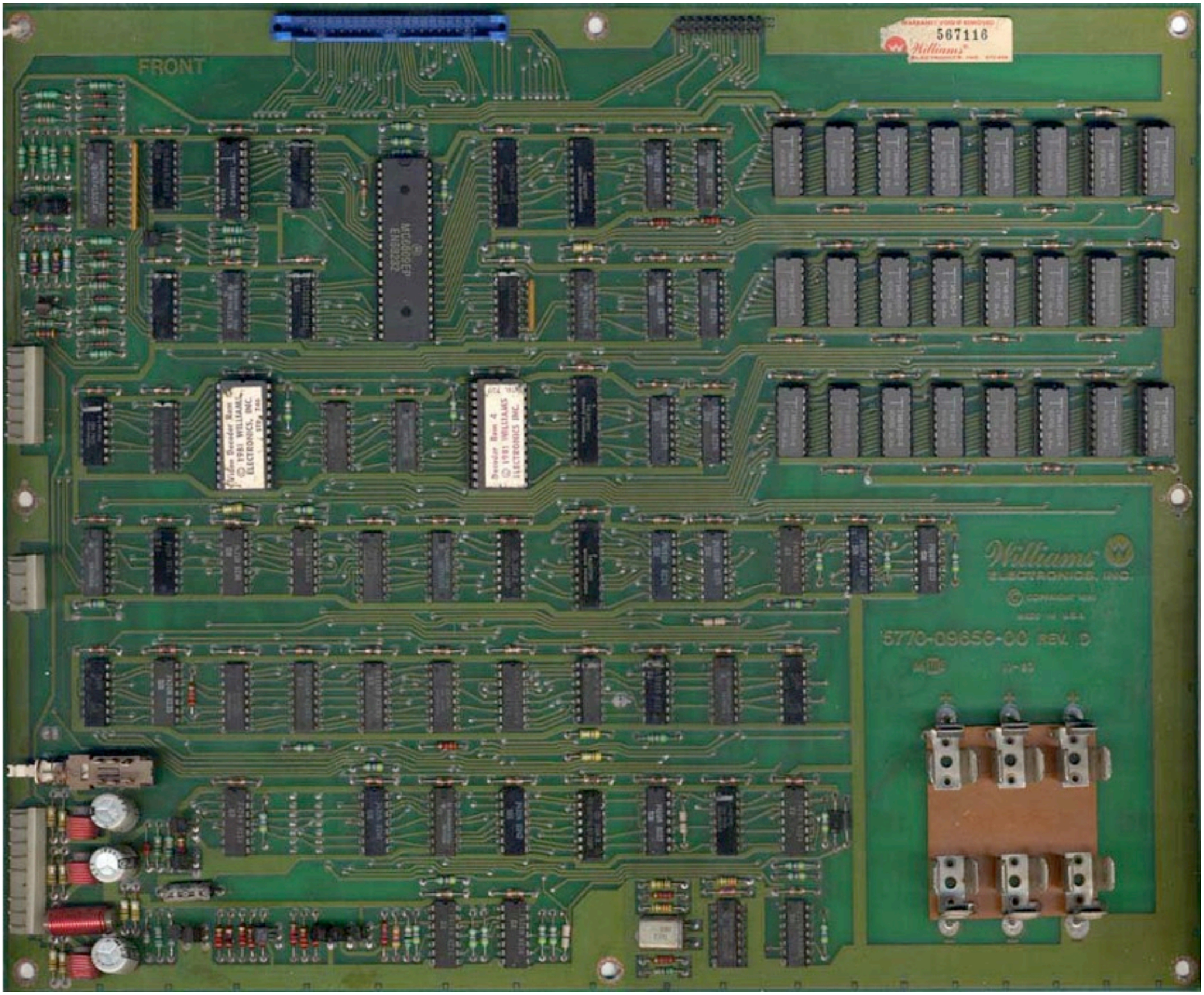
In this video, Jason demonstrates how that's done. This is the machine over at Ground Kontrol, on NW 5th and Couch, in downtown Portland ...if you should ever feel the need to kill some Robotrons yourself.

In this video, Jason demonstrates how that's done. This is the machine over at Ground Kontrol, on NW 5th and Couch, in downtown Portland ...if you should ever feel the need to kill some Robotrons yourself.

Eugene Jarvis

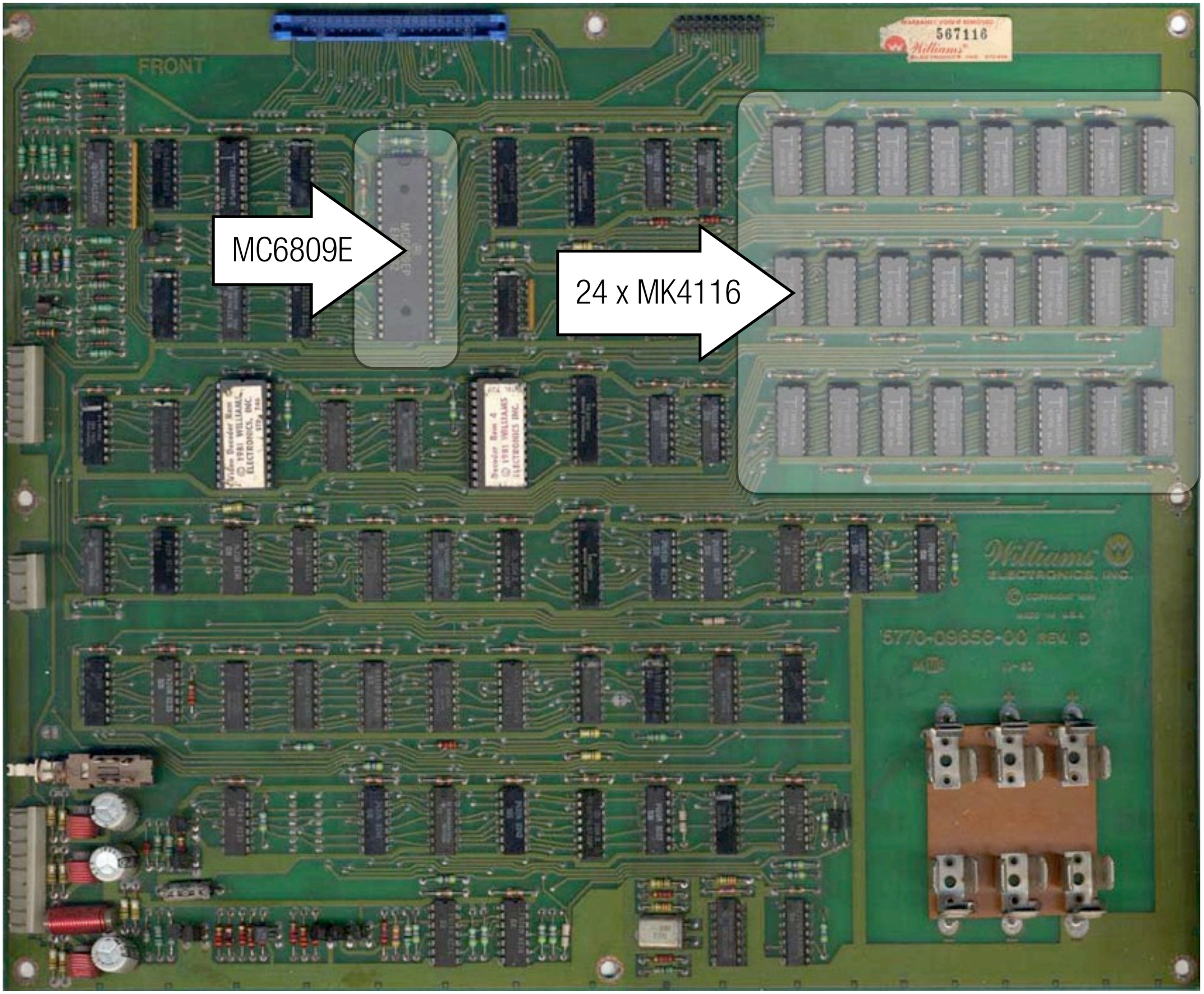
Wikipedia: AcidHelmNun

Robotron was designed for Williams Electronics by a game consulting company called Vid Kidz, consisting of Eugene Jarvis and Larry DeMar. Here is Eugene Jarvis in 2006.



robotron-2084.co.uk

Their handiwork consists of four giant circuit boards inside the game cabinet. Here is the largest one. It connects to the video monitor, and a couple of other boards. Let's look at the labeling on some of the chips and see if we can't Google datasheets to learn more about them.



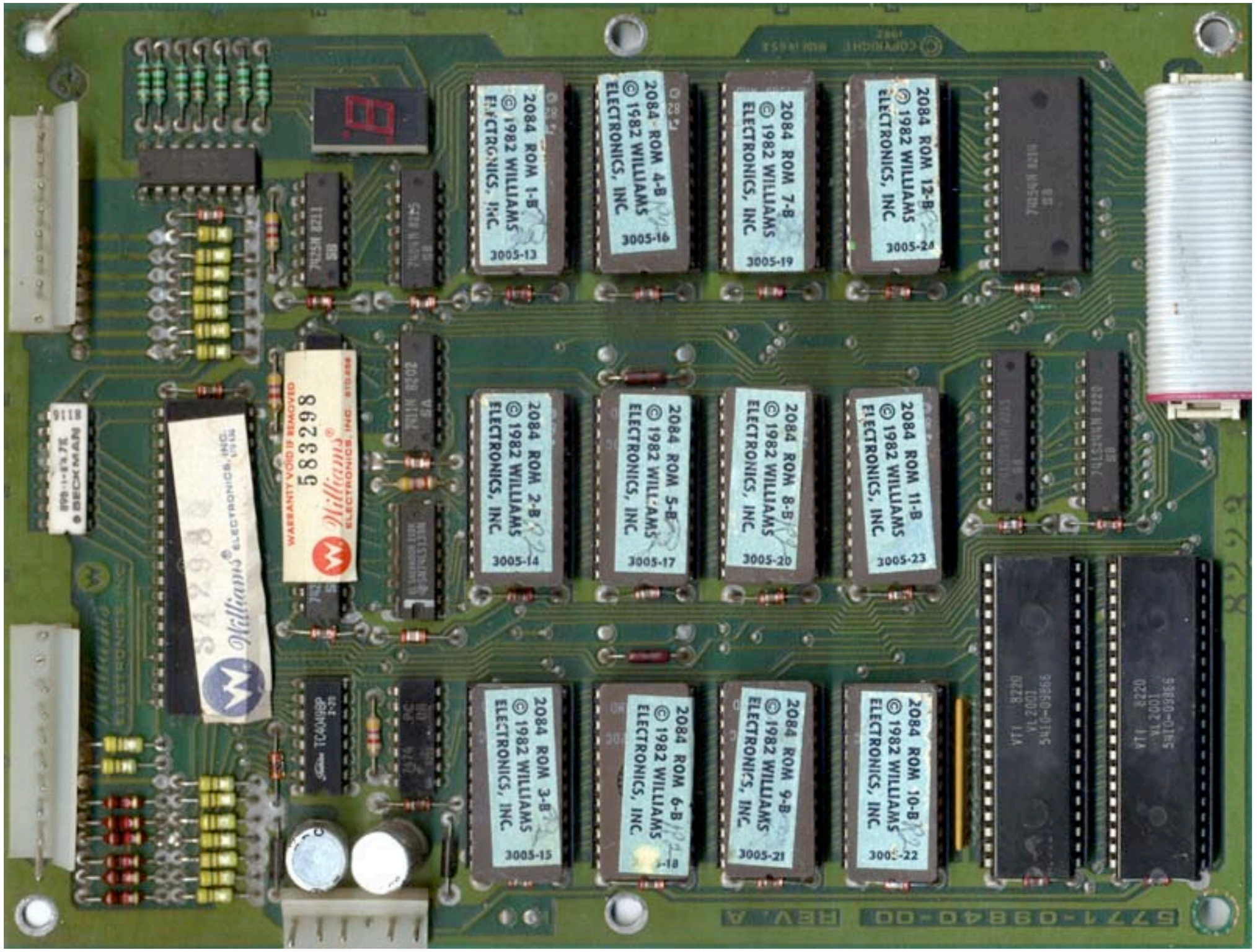
robotron-2084.co.uk

Saturday, June 22, 13

The big chip on the left is a Motorola MC6809E “8-bit Microprocessing Unit”. The 24 chips in the upper right are MK4116 “16 kilobit dynamic RAM” chips. So it sounds like this board contains an 8-bit computer with 48K bytes of memory.

Wikipedia: Bilby

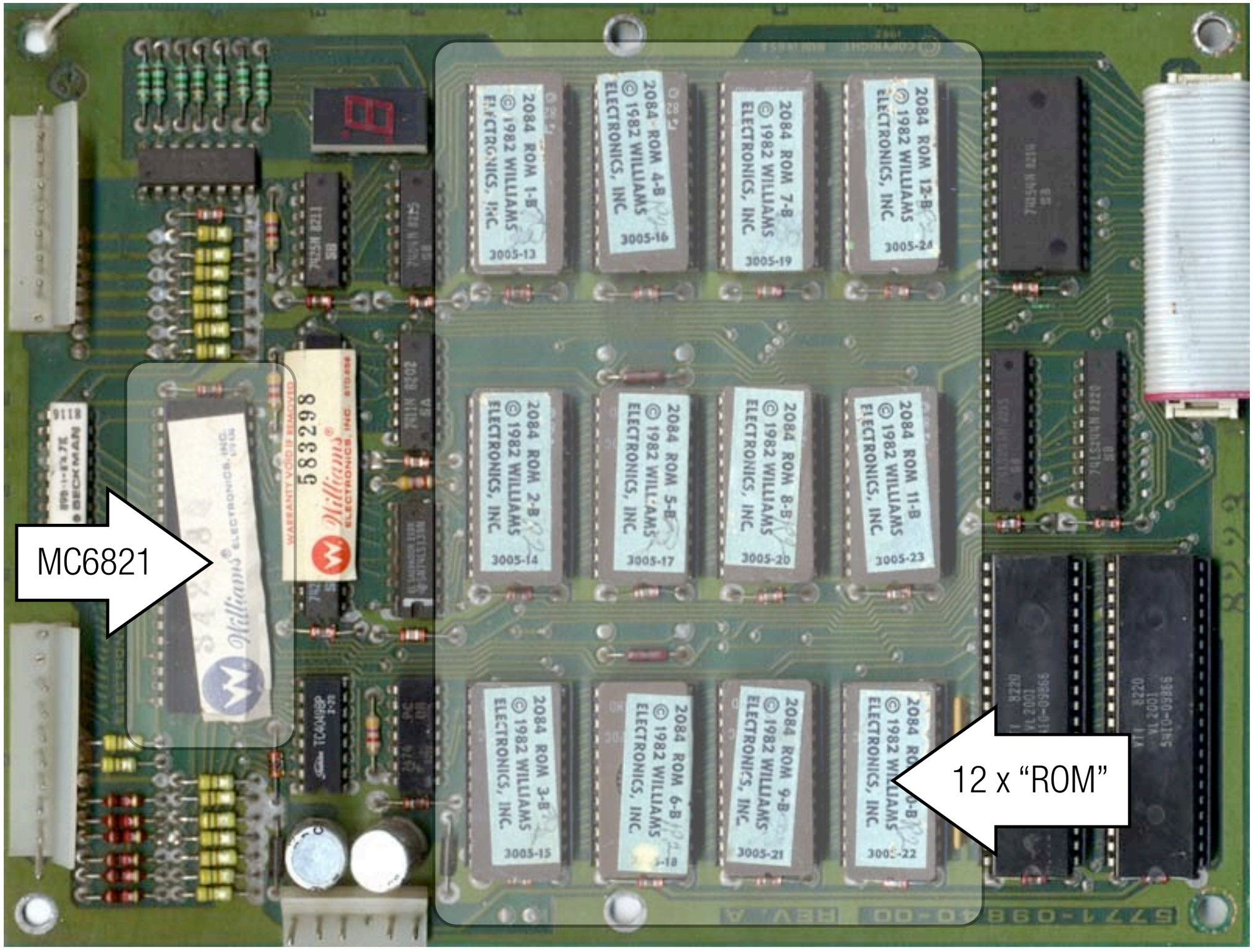
We could be describing this classic computer, the Radio Shack Color Computer. It's also built on the Motorola 6809 processor, and came with roughly the same amount of memory. Robotron is built very much like the computers of the day, but has some special hardware that makes it much better for implementing video games.



robotron-2084.co.uk

Saturday, June 22, 13

Here's another, smaller board that attaches to the large board via the gray ribbon cable in the upper right corner. It attaches to another small board via the top-left connector, and to the coin acceptors via the bottom-left connector.

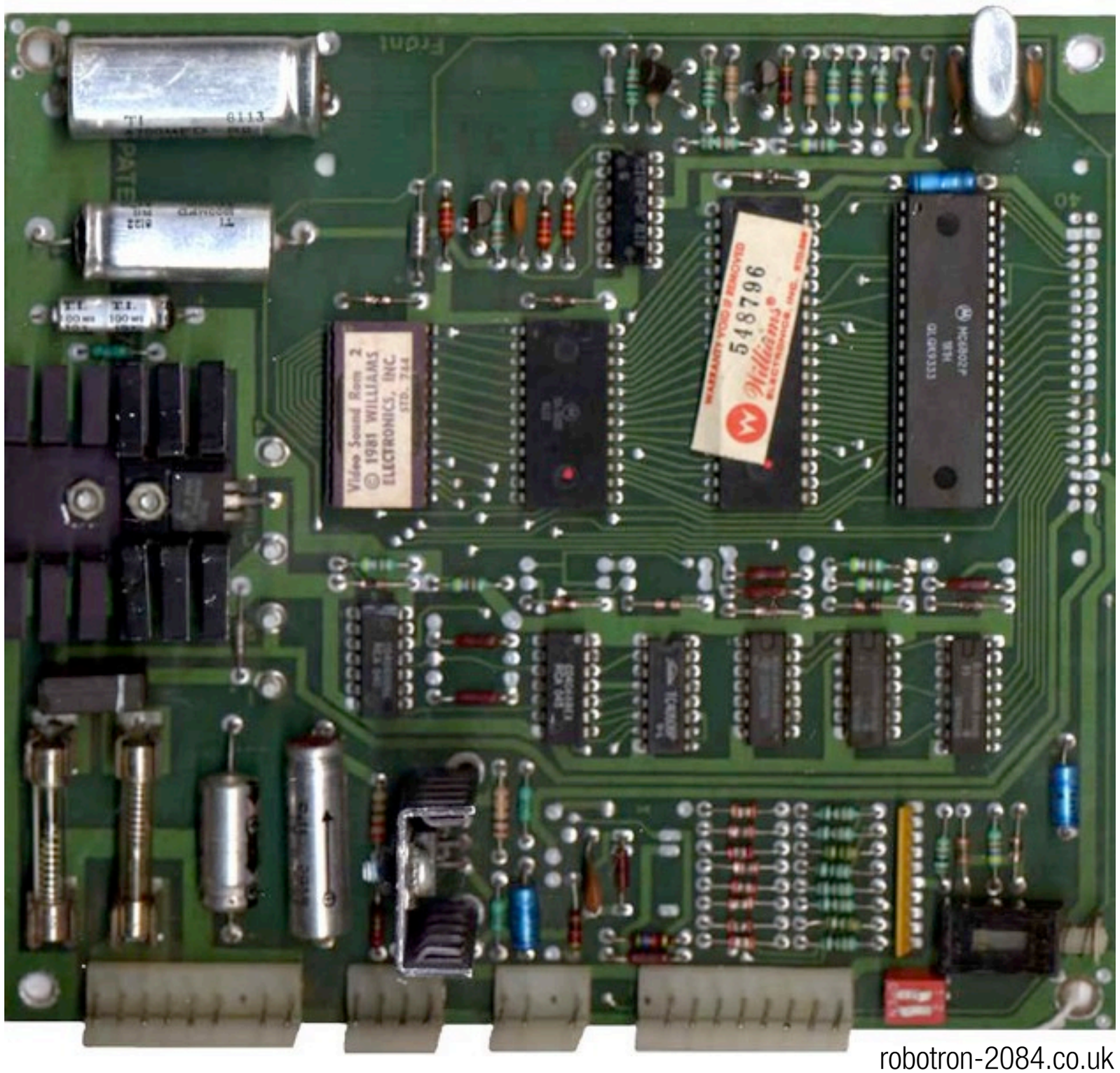


robotron-2084.co.uk

Saturday, June 22, 13

Twelve of these chips have labels on them that say “ROM”. They’re uniquely numbered from one to twelve. This must be the read-only memory where the program code for the game is stored.

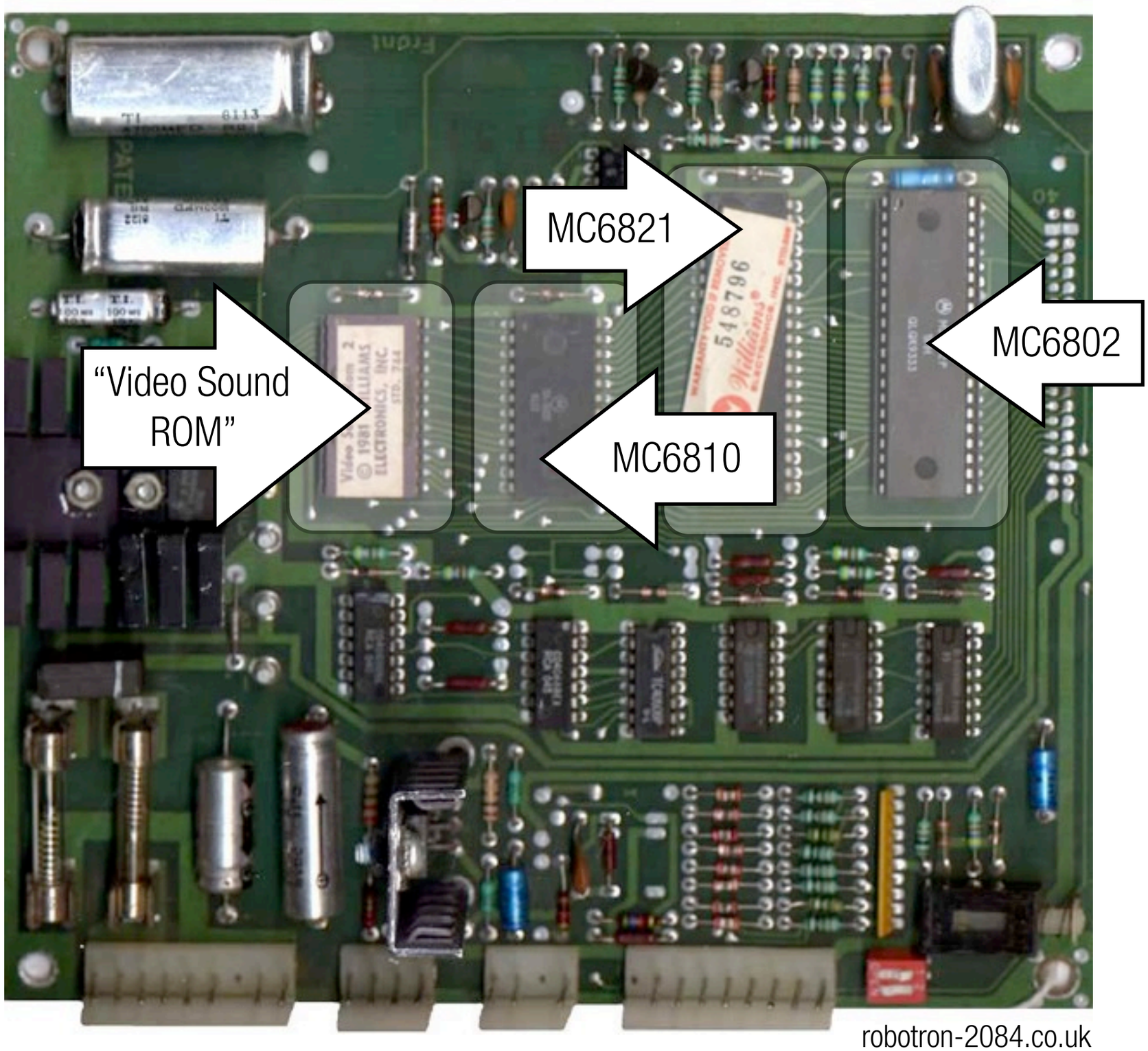
There’s an MC6821 Peripheral Interface Adapter next to the connectors on the left.



robotron-2084.co.uk

Saturday, June 22, 13

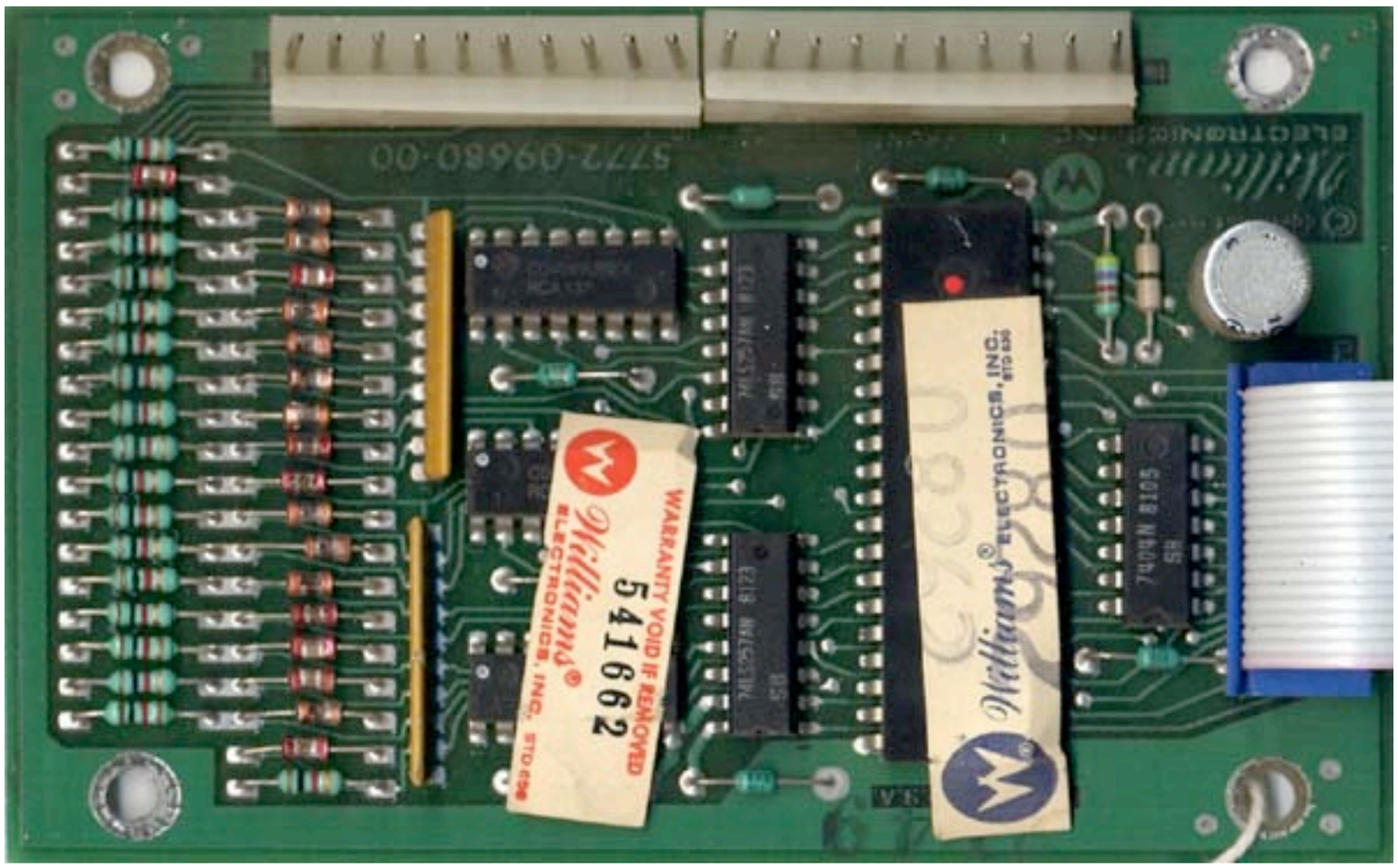
Here’s the third board, which connects to the prior board by a cable. It also attaches to the audio speaker.



robotron-2084.co.uk

Saturday, June 22, 13

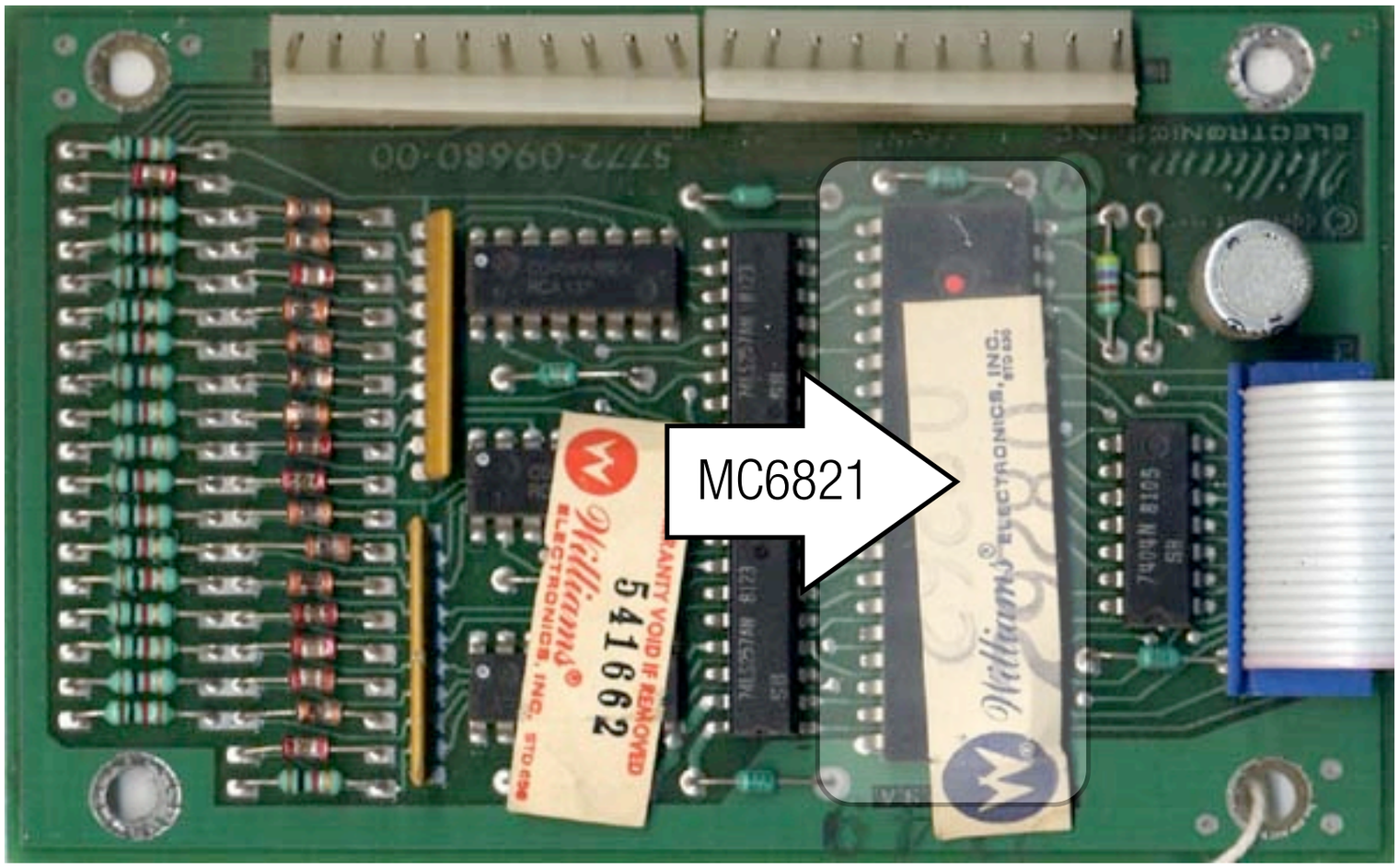
There are three interesting chips here. The MC6802 is another “8-bit Microprocessor”. The MC6810 is a 128 byte RAM chip. And there’s a ROM chip for code. There’s another MC6821 Peripheral Interface Adapter. So it seems like we have another 8-bit computer on this board. And it may be responsible for sound effects, because it has the speaker connection and has a “Video Sound ROM” chip on it.



robotron-2084.co.uk

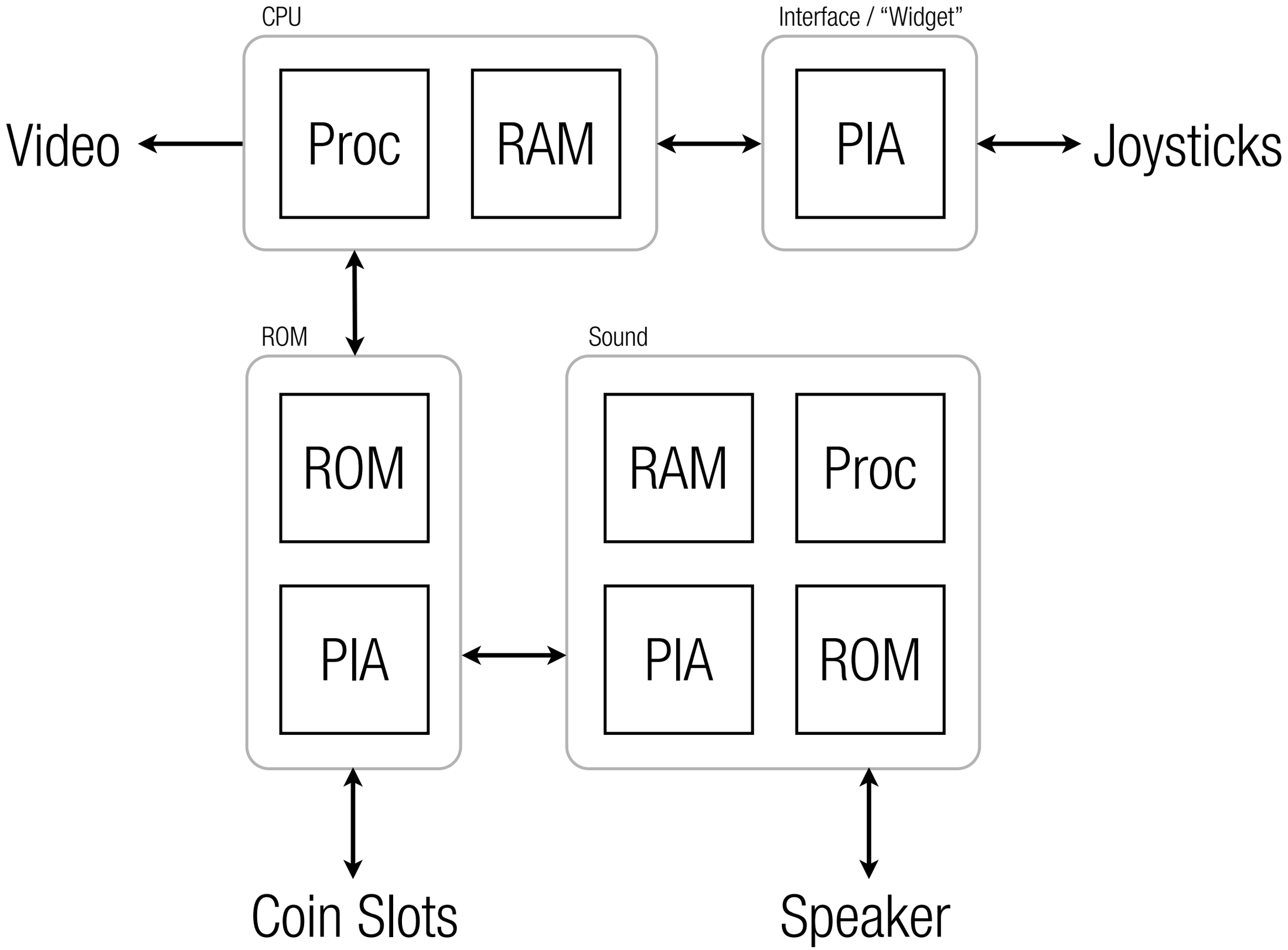
Saturday, June 22, 13

Here’s the last board, which connects to the big board via another gray ribbon cable. There are two connectors along the top edge that attach to the joysticks. If we look under the purple label...



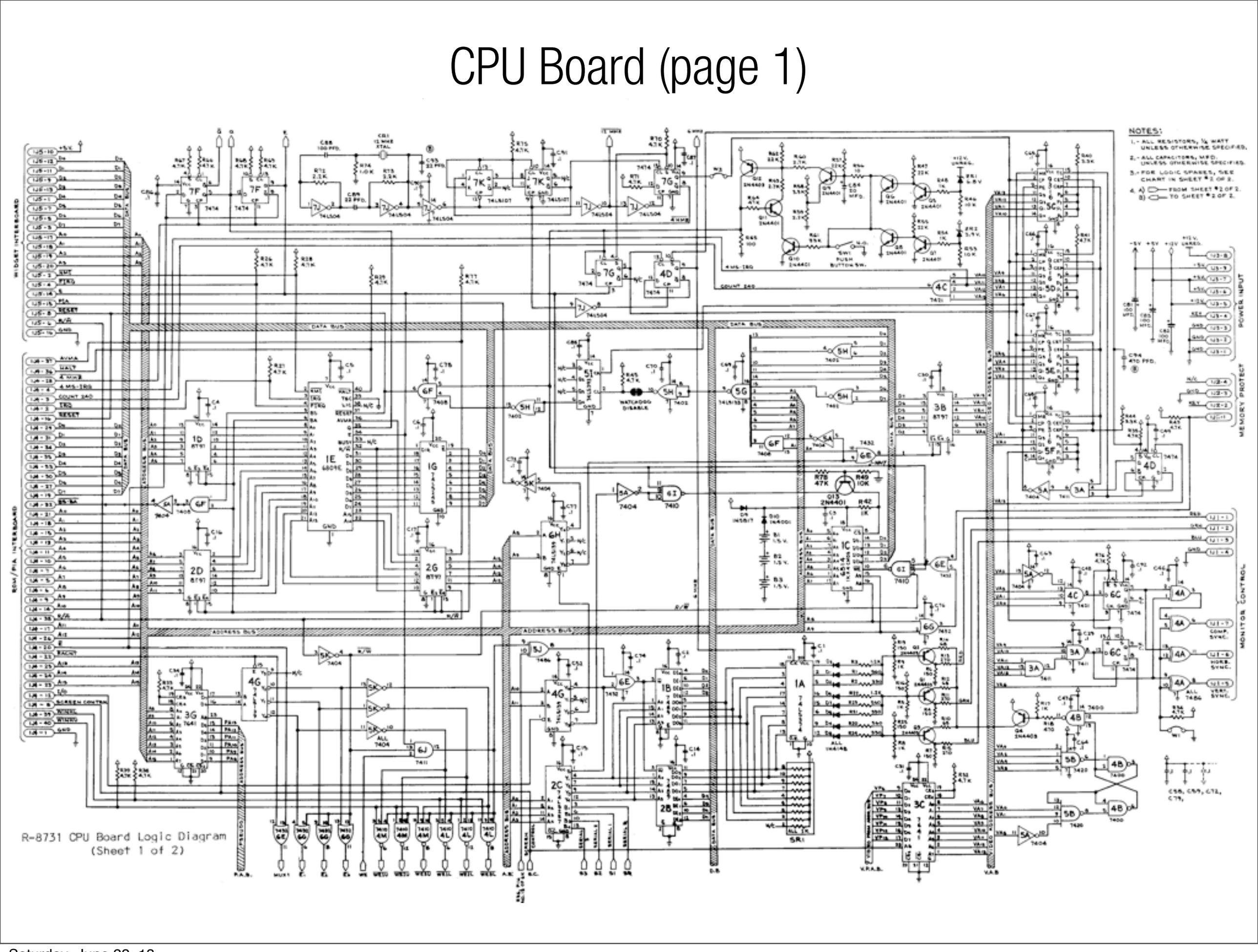
robotron-2084.co.uk

...we find yet another MC6821 Peripheral Interface Adapter. So this chip must interface the joysticks and buttons to the system.



Here’s a diagram of what we’ve learned so far about the four boards and how they connect. It seems like we have two distinct computers in this machine. One does the video stuff and checks the joysticks and coin slots. The other computer generates sound effects.

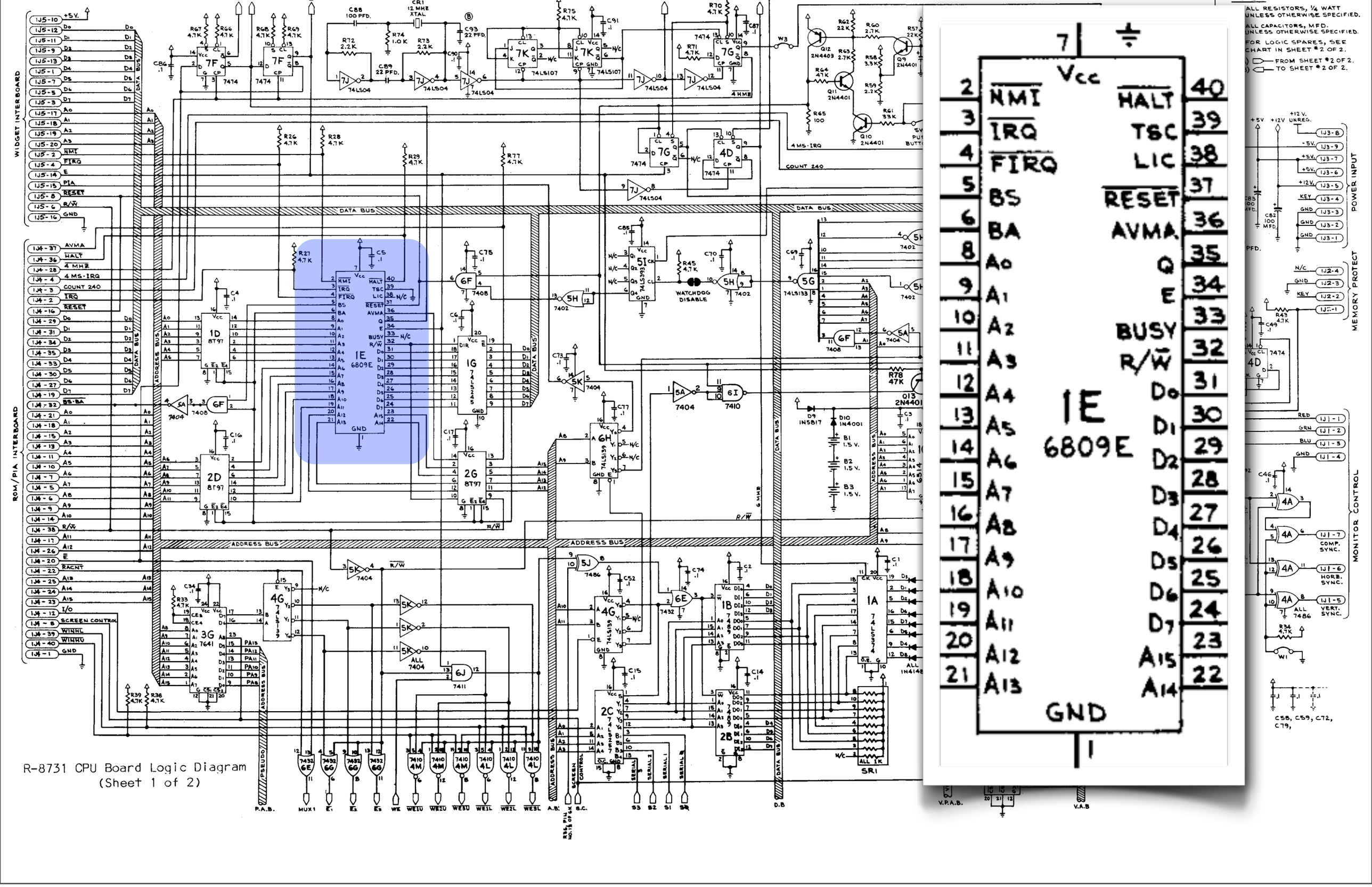
CPU Board (page 1)



Saturday, June 22, 13

This is one of five pages of schematics for the Robotron game. Robotron was made back when circuit boards and chips were large and easy to repair using common tools, and electronics weren't quite as reliable as they are today. So it made sense for the manufacturer to publish a repair manual that included complete schematics.

CPU Board (page 1)



Saturday, June 22, 13

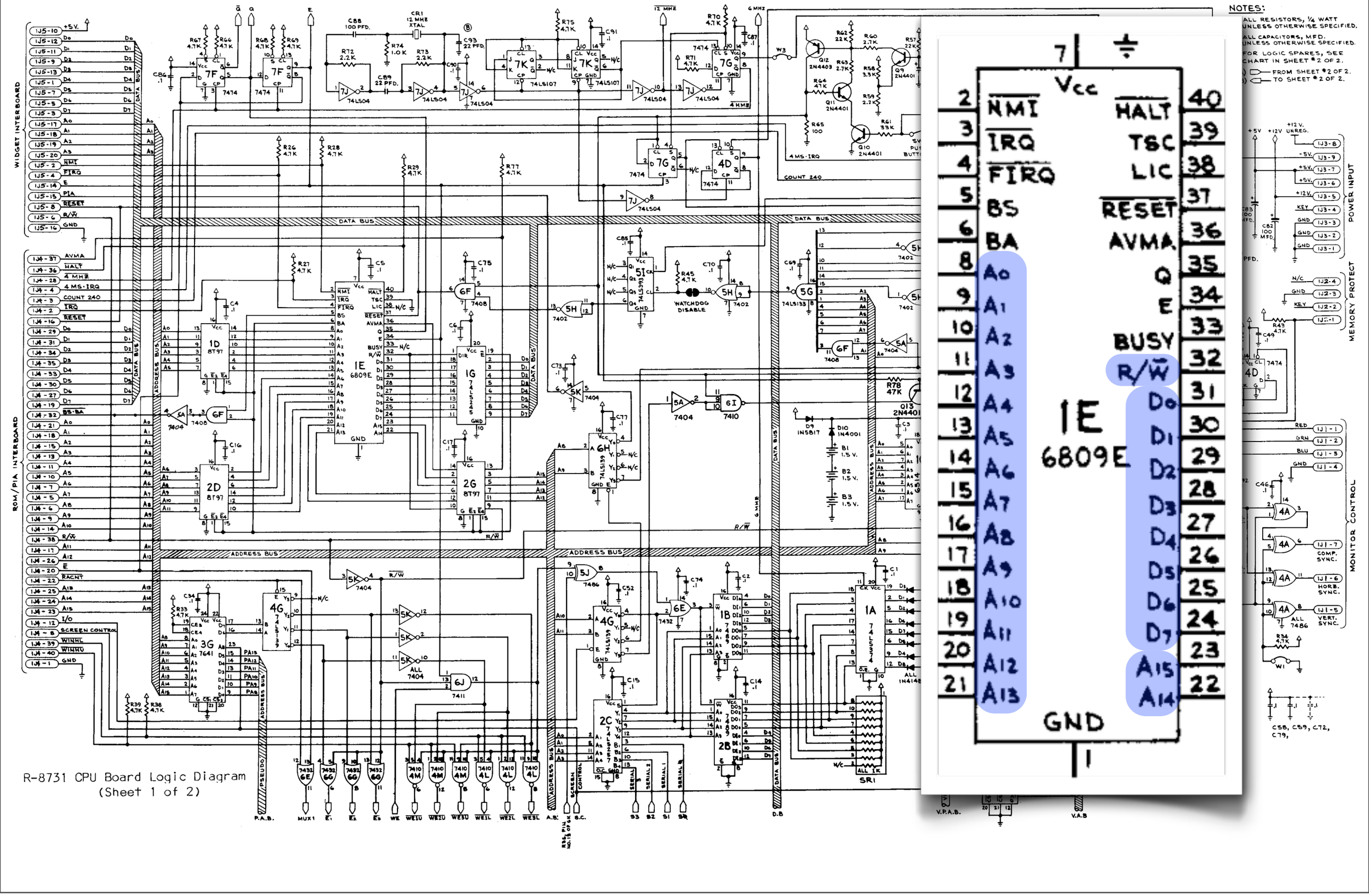
Here's the MC6809E microprocessor we found earlier by looking at the board photos.

Program Execution

Saturday, June 22, 13

The MC6809E is the 1982 equivalent of an ARM or Intel or AMD processor. It runs a program, step-by-step, from machine language instructions located in memory. The program instructs the processor to modify the program's state by reading and writing to variables stored in memory.

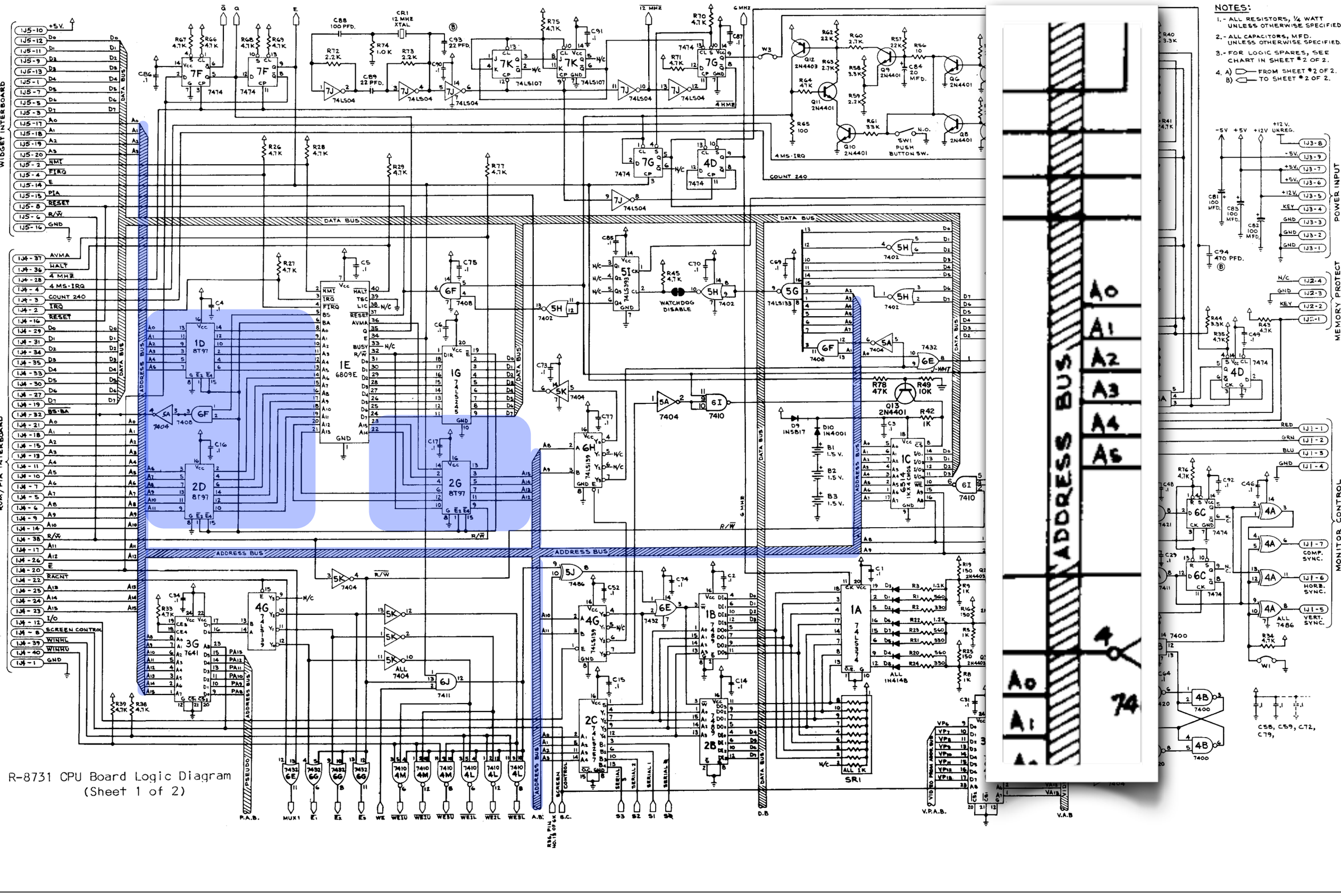
Tracing the Address Bus



Saturday, June 22, 13

If we look at the 6809 processor, we can see the chip has 16 “A” signals, eight “D” signals, and a “Read/Write” signal. These are the signals that interface to the memory and peripherals.

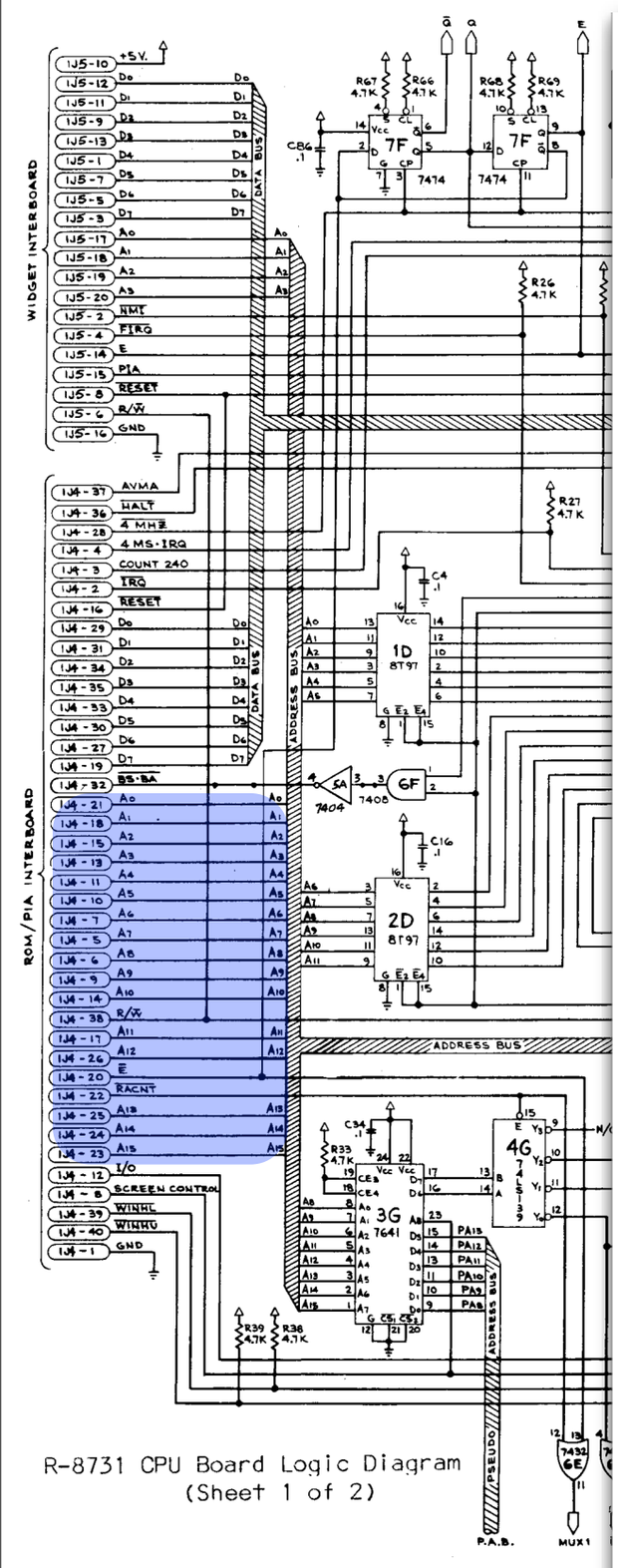
Tracing the Address Bus



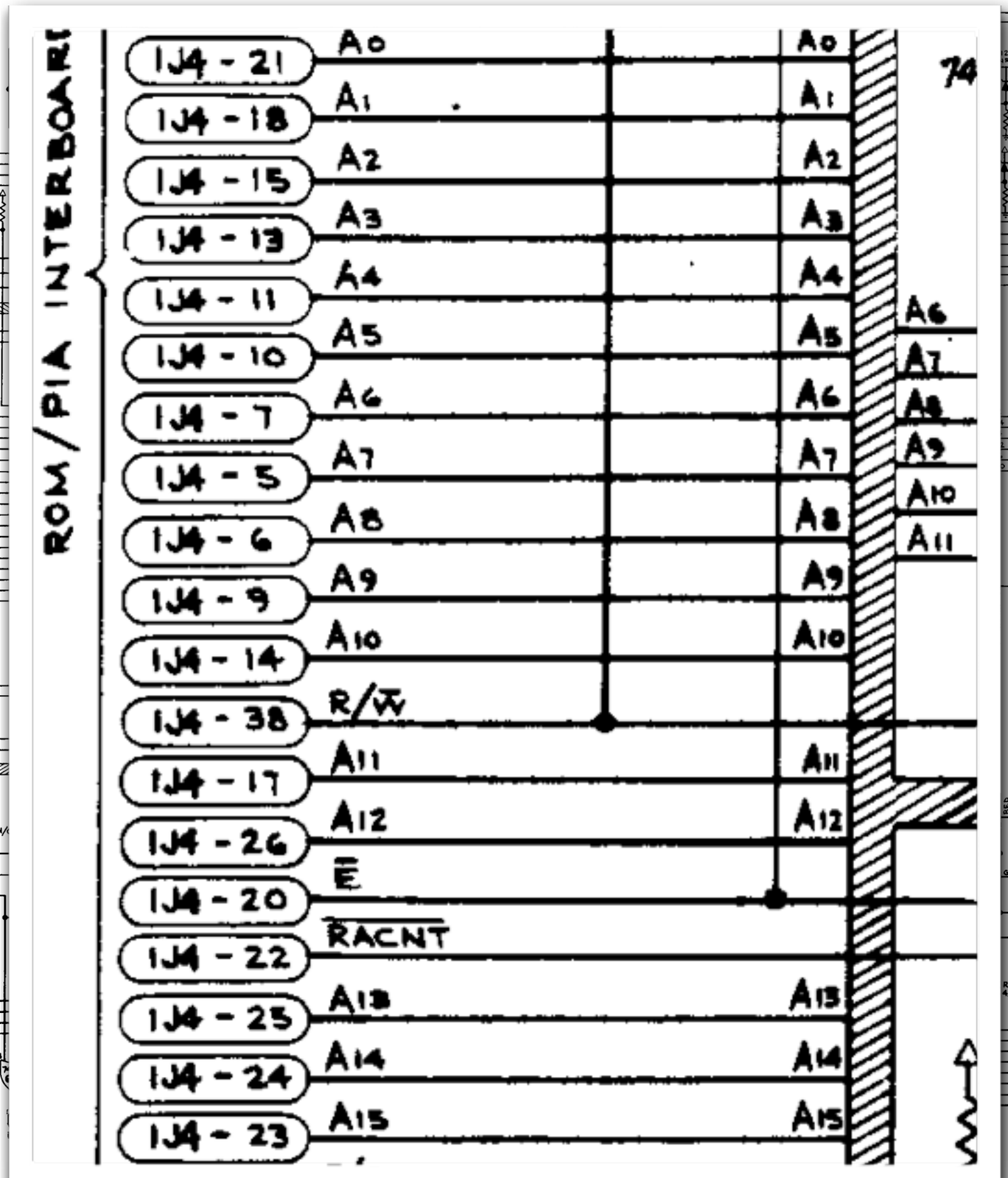
Saturday, June 22, 13

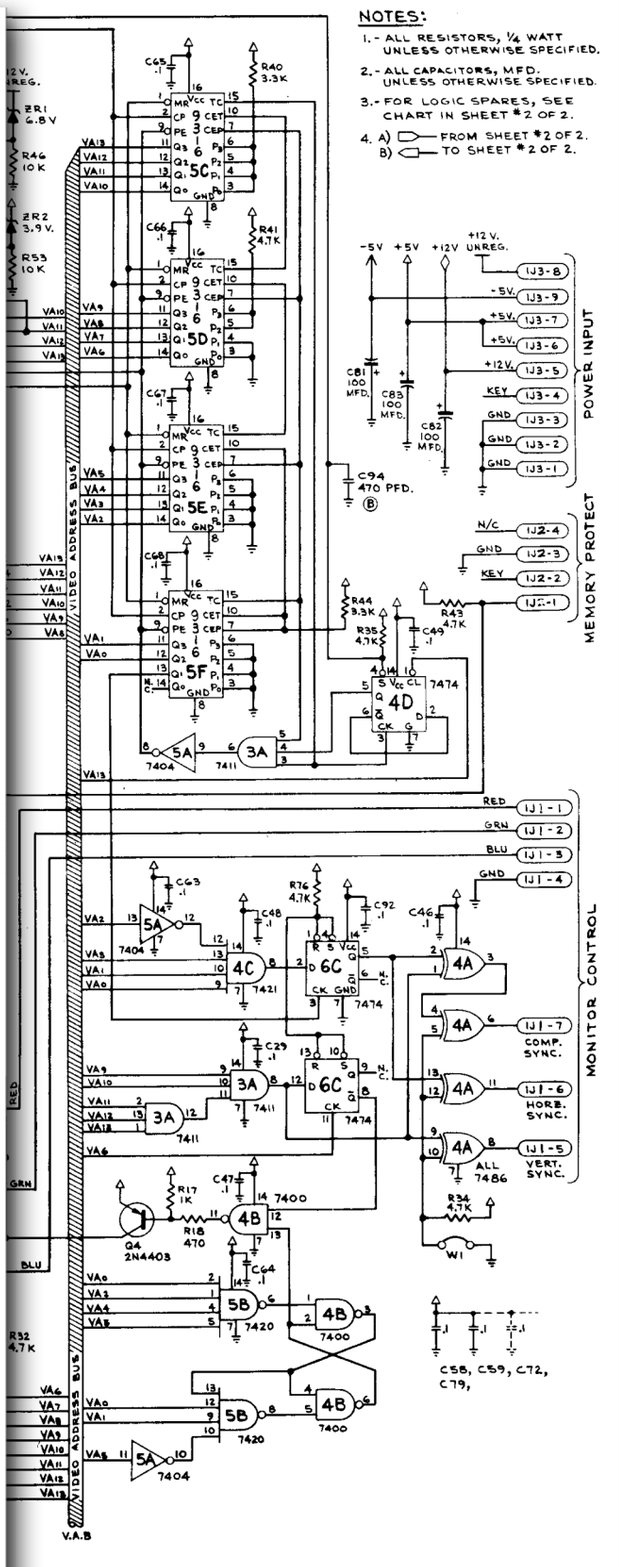
If we follow the address signals out and see where they connect, we'll find all of the memory and peripherals that are reachable from the microprocessor.

Tracing the Address Bus



R-8731 CPU Board Logic Diagram
(Sheet 1 of 2)

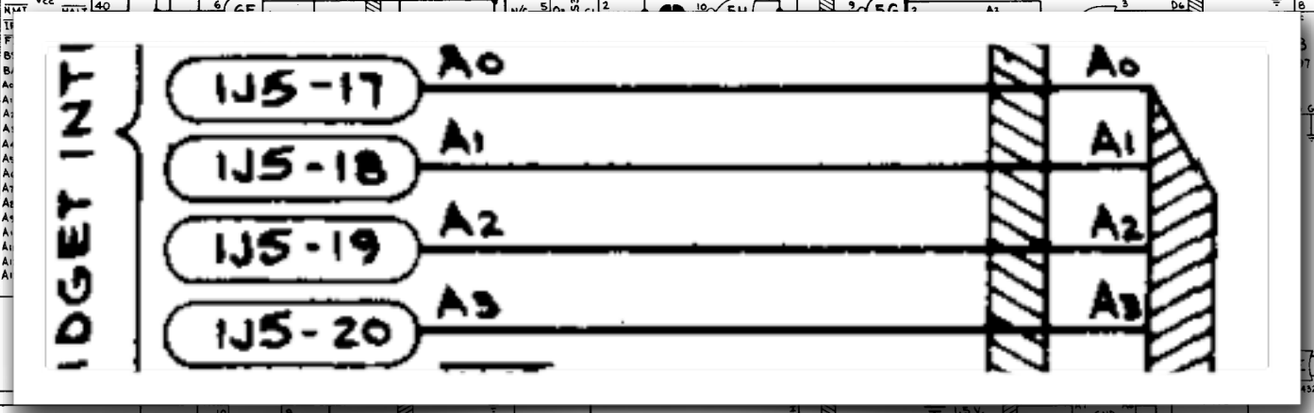




Saturday, June 22, 13

The address bus goes to the board with the ROM chips on it. It would make sense if the game's program code was in the ROM.

Tracing the Address Bus

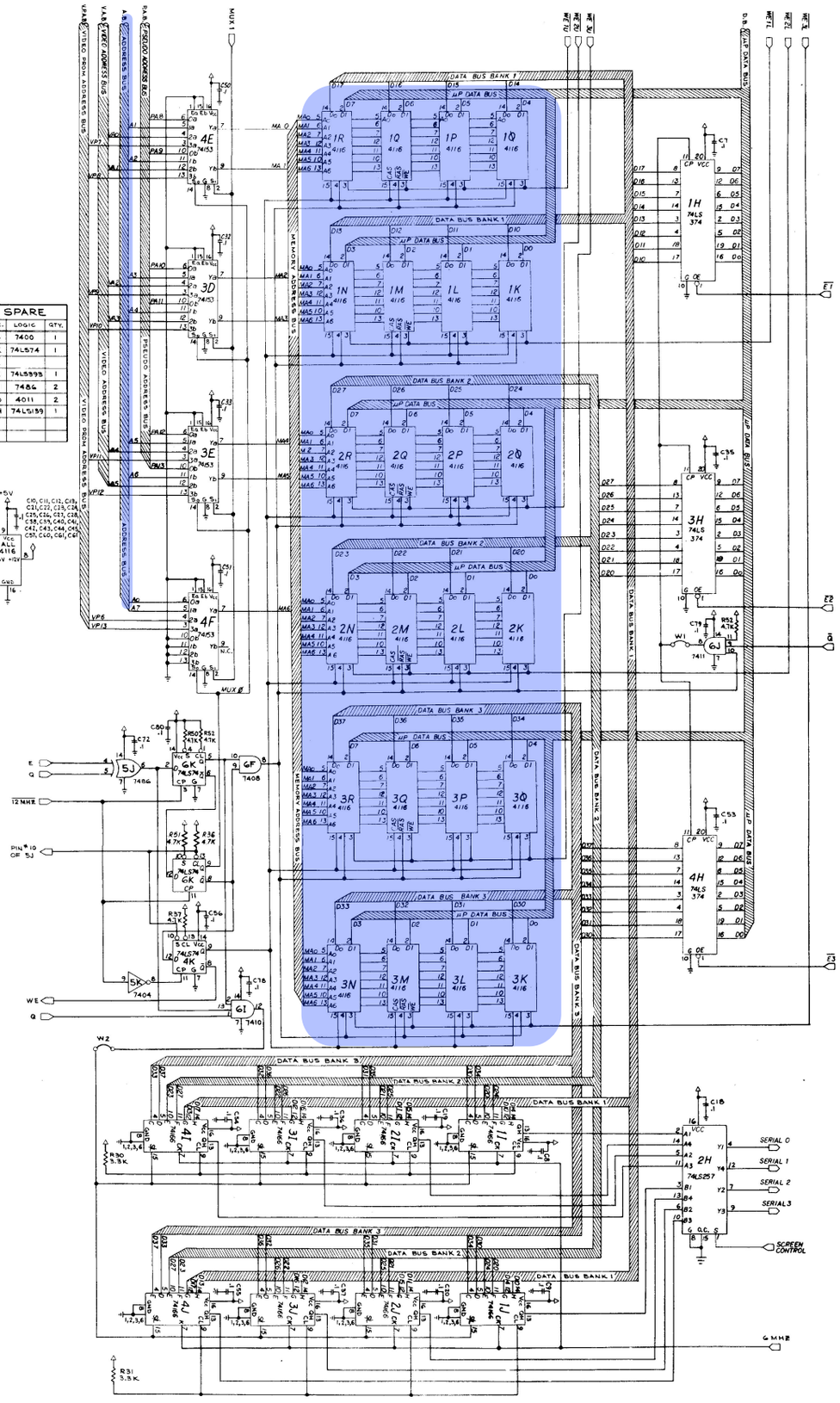


R-8731 CPU Board Logic Diagram
(Sheet 1 of 2)

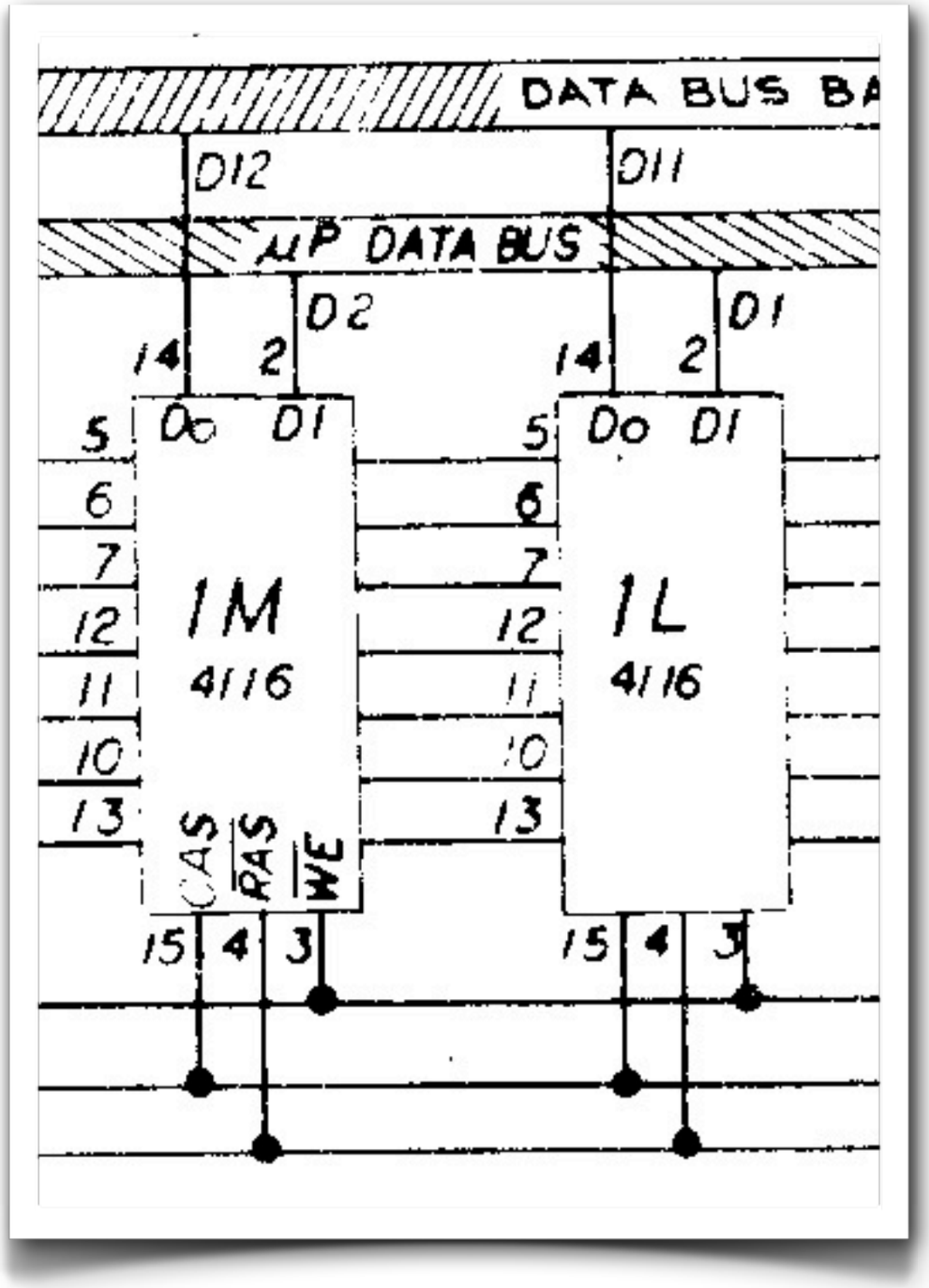
Saturday, June 22, 13

Some of the address bus bits also go to the joystick board.

CPU Board (page 2)

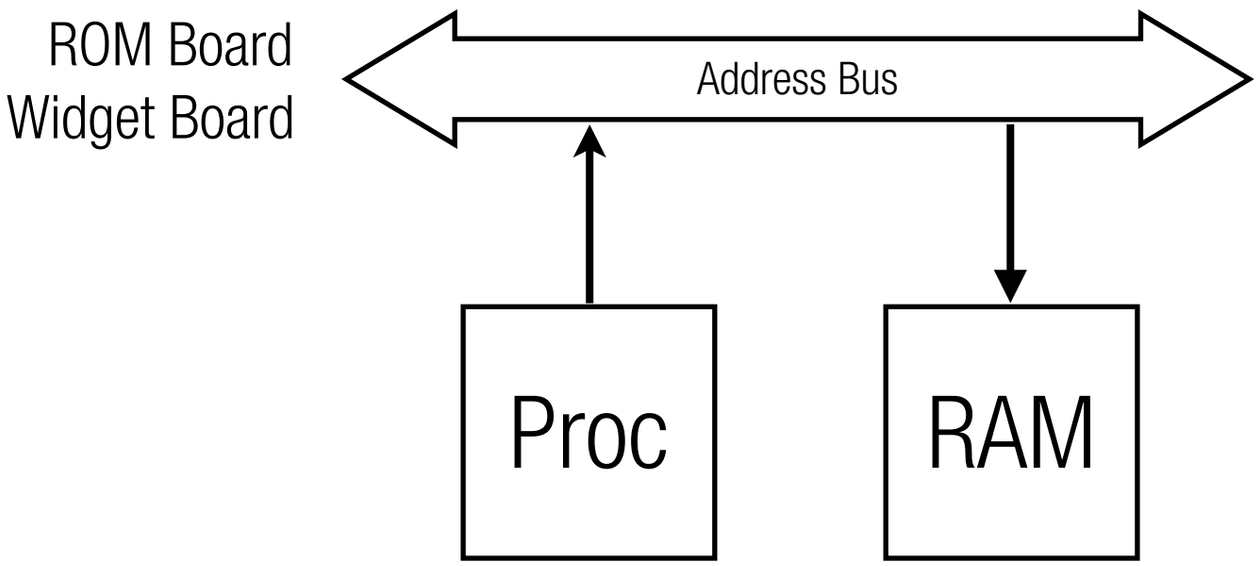


R-8731 CPU Board Logic Diagram (Sheet 2 of 2)



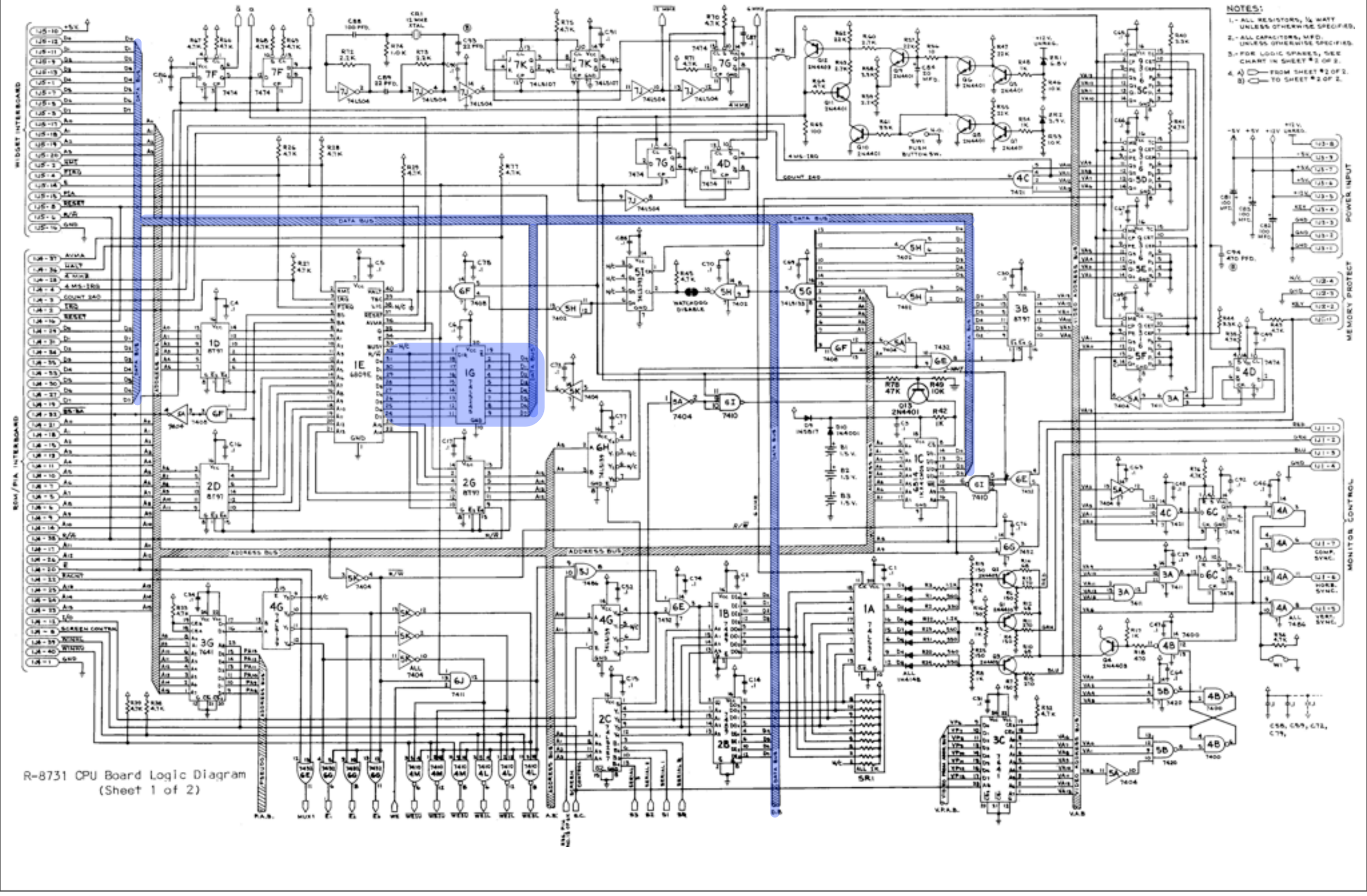
Saturday, June 22, 13

The address bus also goes to the CPU Board schematic, page 2, where it connects to the 48K byte of memory chips.



Here’s a diagram of what we know about the CPU board so far. The processor and RAM are both connected to the address bus, and the address bus goes to the ROM board.

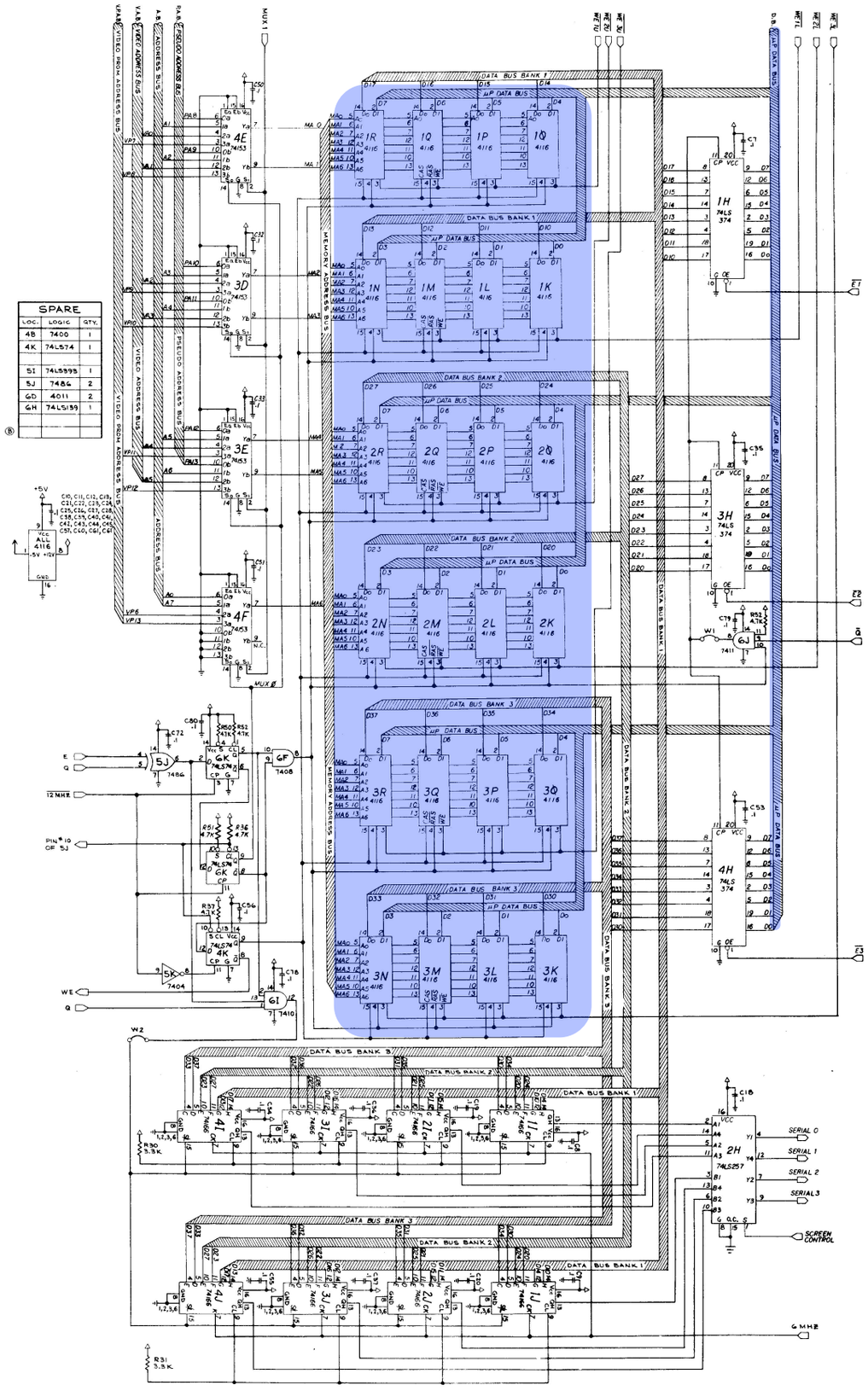
Tracing the Address Bus

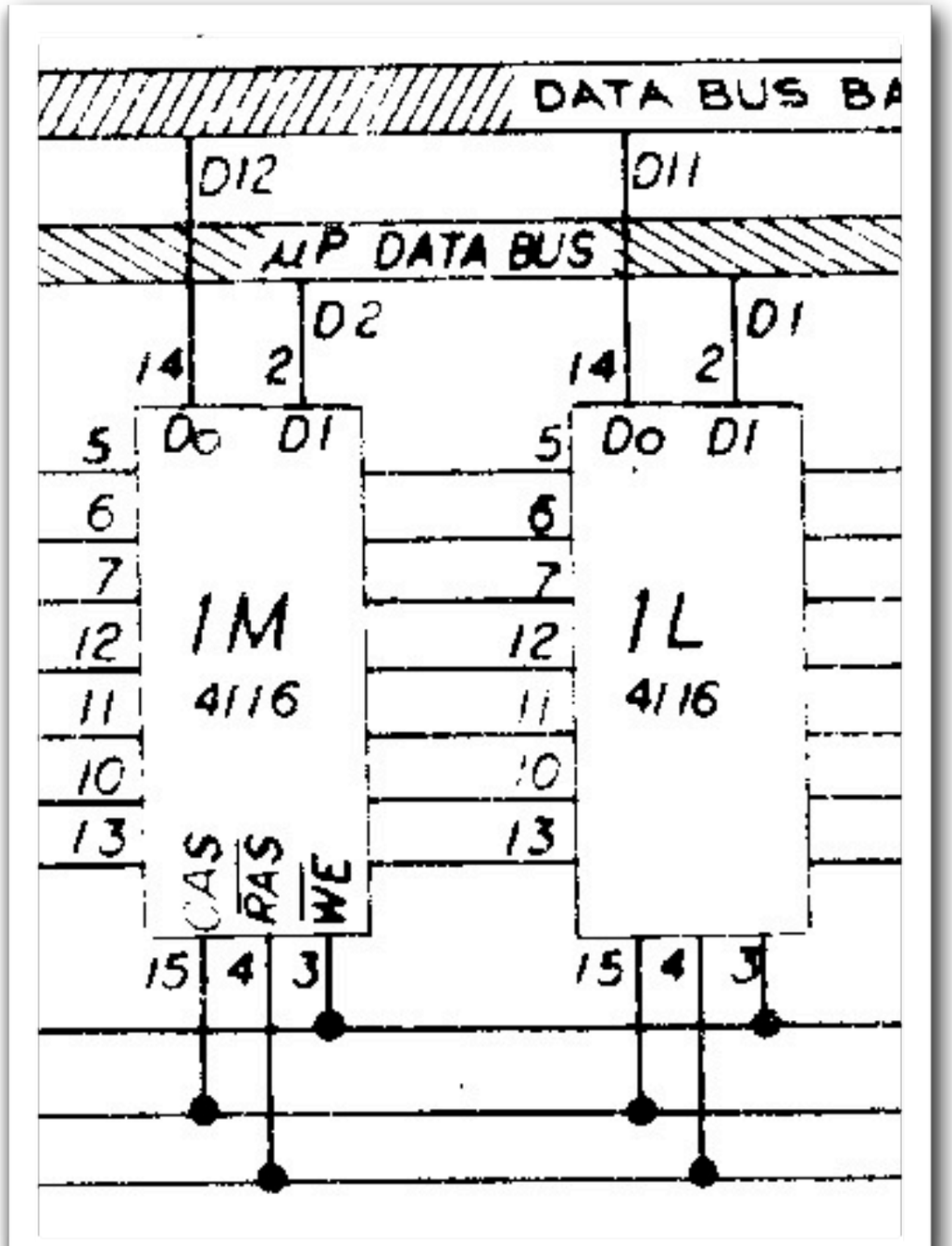


Saturday, June 22, 13

Back to the schematic, let's see where the processor data bus goes. The data bus also goes to the ROM board, the Widget board, and to page two of the CPU board schematic.

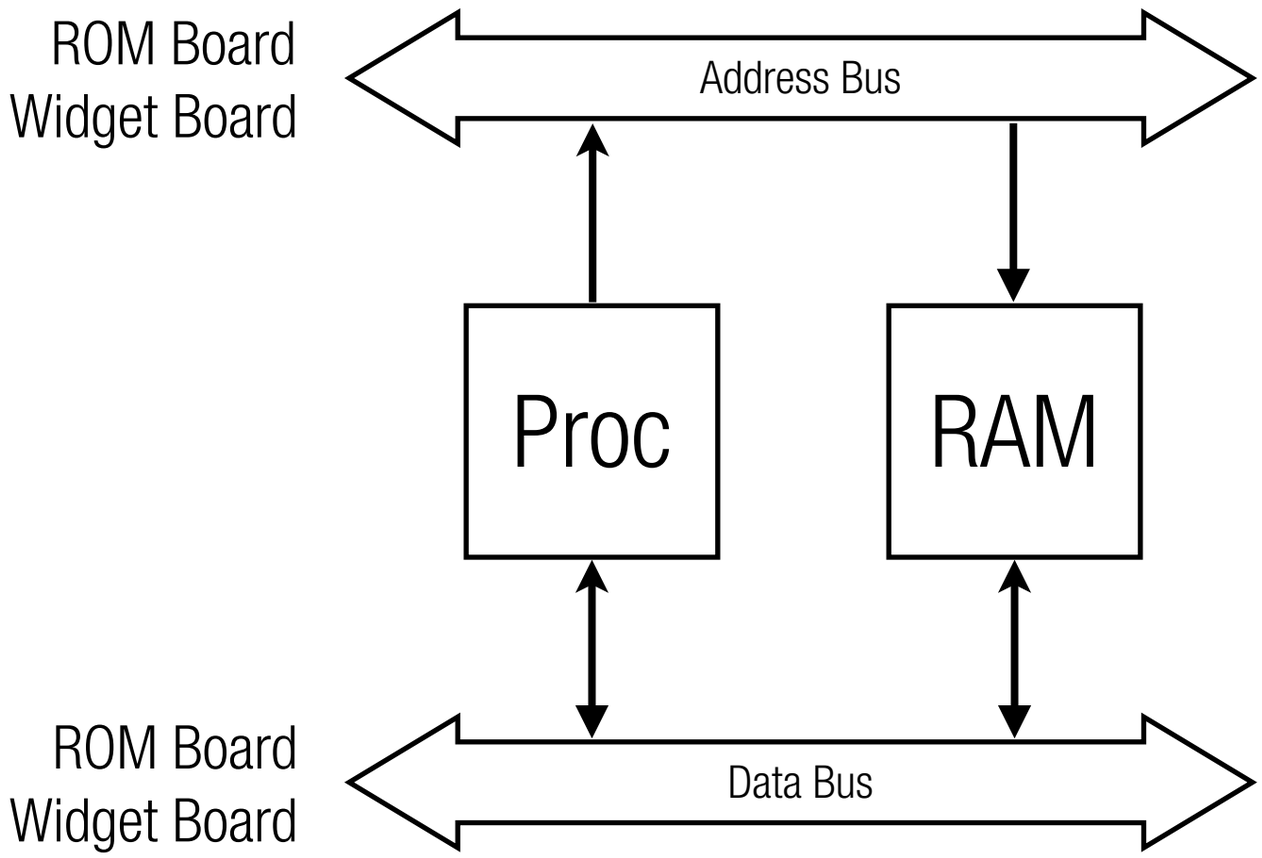
CPU Board (page 2)





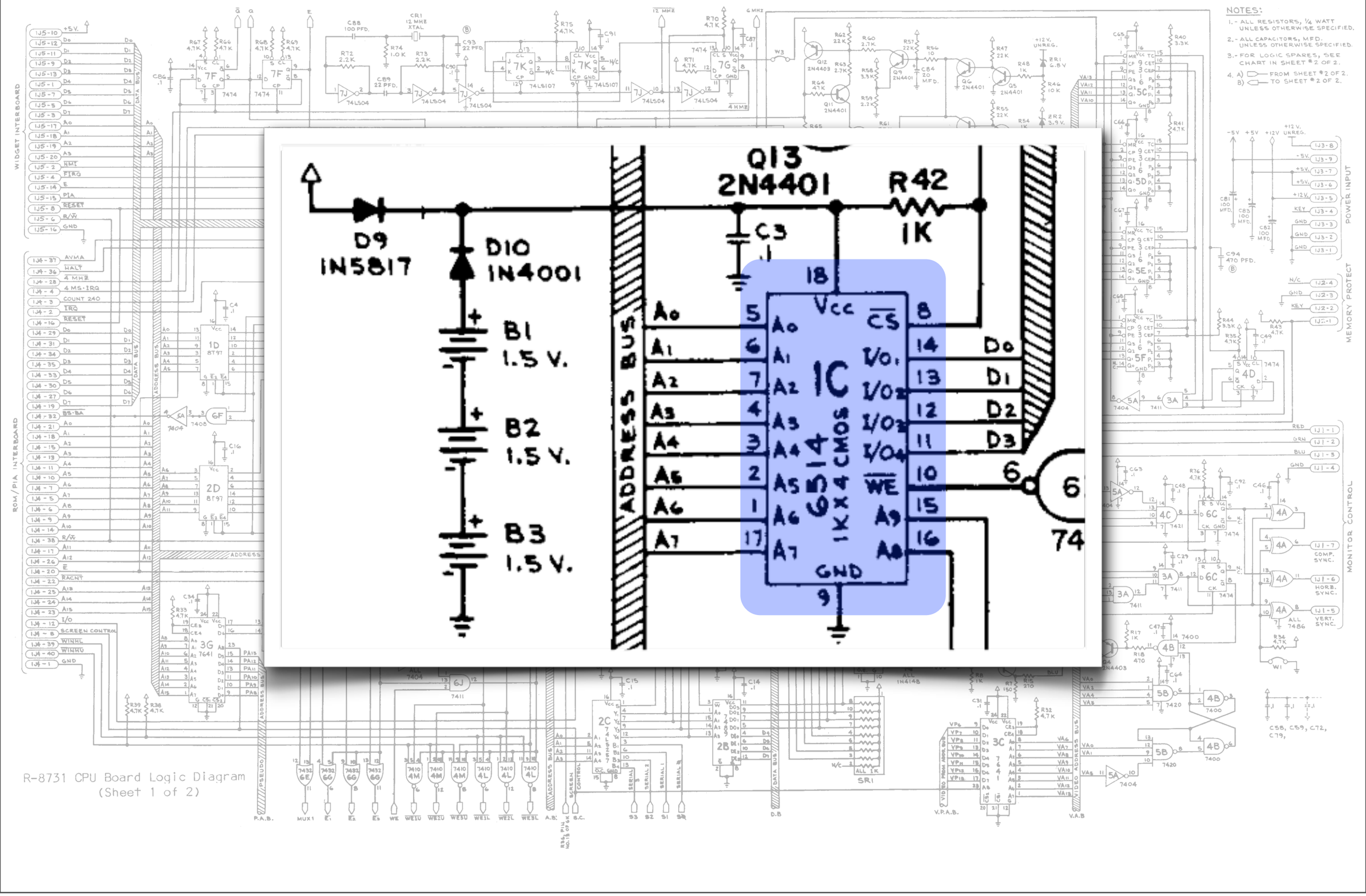
Saturday, June 22, 13

On page two of the CPU board schematic, the data bus goes to the 48K memory bank.



Here’s the data bus, added to the CPU board diagram.

Tracing the Address Bus



Saturday, June 22, 13

Let's look around the schematic for what else the address and data buses connect to. Here's a RAM chip that's not part of the 48K. It's a 1K x 4 RAM...

intersil™

HM-6514/883

March 1997

1024 x 4 CMOS RAM

Features

- **This Circuit is Processed in Accordance to MIL-STD-883 and is Fully Conformant Under the Provisions of Paragraph 1.2.1.**
- **Low Power Standby. . . . . 125μW Max**
- **Low Power Operation . . . . . .35mW/MHz Max**
- **Data Retention . . . . . at 2.0V Min**
- **TTL Compatible Input/Output**
- **Common Data Input/Output**
- **Three-State Output**
- **Standard JEDEC Pinout**
- **Fast Access Time . . . . . 120/200ns Max**
- **18 Pin Package for High Density**
- **Gated Inputs - No Pull Up or Pull Down Resistors Required**
- **On-Chip Address Register**

Description

The HM-6514/883 is a 1024 x 4 static CMOS RAM fabricated using self-aligned silicon gate technology. The device utilizes synchronous circuitry to achieve high performance and low power operation.

On chip latches are provided for addresses allowing efficient interfacing with microprocessor systems. The data output can be forced to a high impedance state for use in expanded memory arrays.

Gated inputs allow lower operating current and also eliminates the need for pull up or pull down resistors. The HM-6514/883 is fully static RAM and may be maintained in any state for an indefinite period of time.

Data retention supply voltage and supply current are guaranteed over temperature.

Ordering Information

120ns	200ns	300ns	TEMPERATURE RANGE	PACKAGE	PKG. NO.
HM1-6514S/883	HM1-6514B/883	HM1-6514/883	-55°C to 125°C	CERDIP	F18.3

Saturday, June 22, 13

Here's the first page of the chip's datasheet.

intersil™

HM-6514/883

March 1997

x 4 CMOS RAM

```
bit ram[1024][4]

process chip_hm6514:
  if write:
    ram[address] = data
  else:
    data = ram[address]

> write = 1

> address = 22, data = 13

> write = 0

> address = 22, print data[22]
13
```

Features

- This Circuit is Pro
  883 and is Fully C
  Paragraph 1.2.1.
- Low Power Standb
- Low Power Operat
- Data Retention . . .
- TTL Compatible In
- Common Data Inp
- Three-State Outpu
- Standard JEDEC P
- Fast Access Time
- 18 Pin Package fo
- Gated Inputs - No
  Required
- On-Chip Address

tic CMOS RAM fabri-
echnology. The device
eve high performance

sses allowing efficient
ems. The data output
te for use in expanded

ent and also eliminates
s. The HM-6514/883 is
ed in any state for an

oly current are guaran-

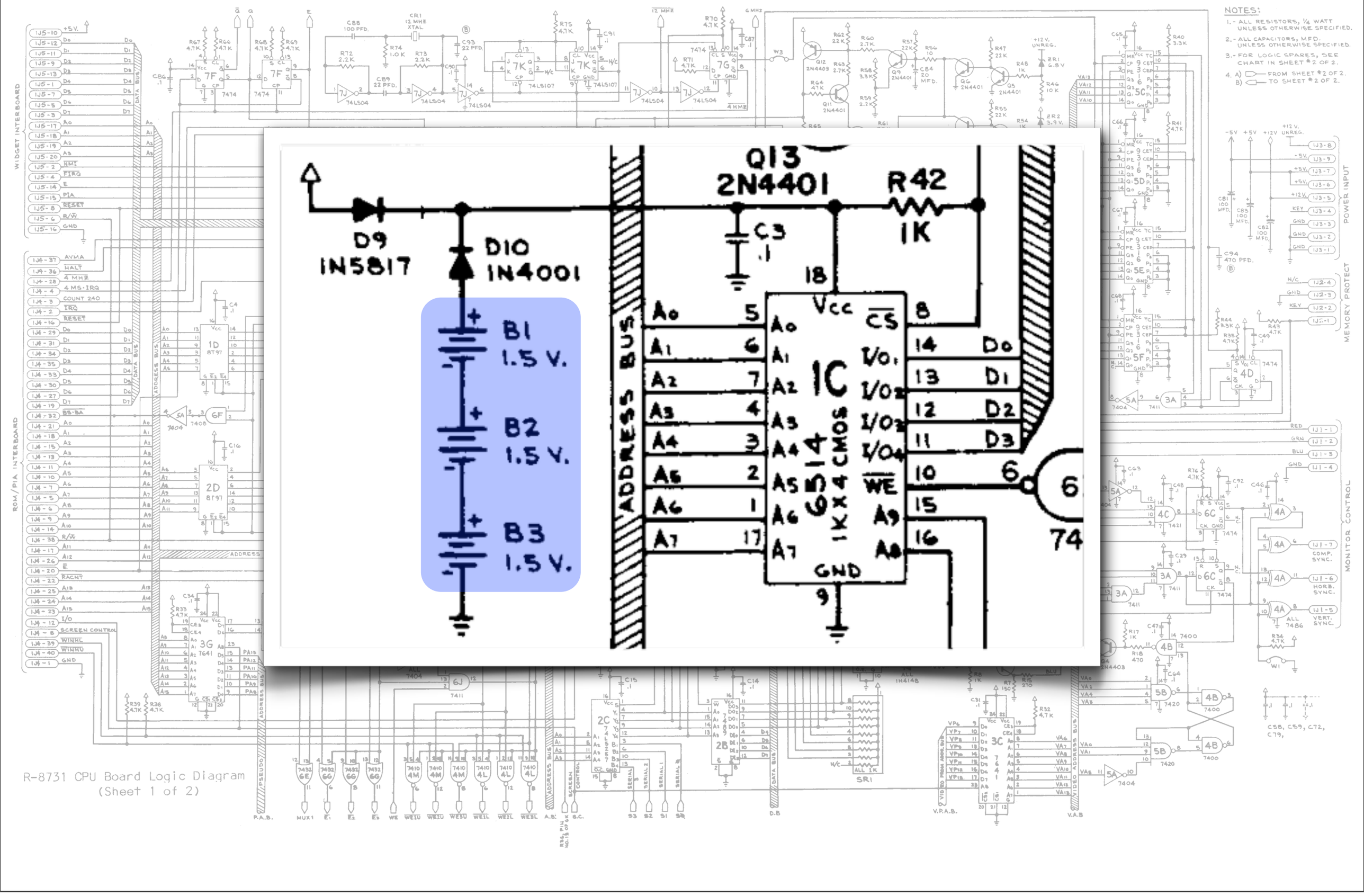
Ordering Information

120ns	200ns	300ns	TEMPERATURE RANGE	PACKAGE	PKG. NO.
HM1-6514S/883	HM1-6514B/883	HM1-6514/883	-55°C to 125°C	CERDIP	F18.3

Saturday, June 22, 13

And some pseudocode for how the chip behaves. An array is declared that represents the 1K x 4 bits of storage in the chip. A process in the chip outputs data found at an address in the array, unless the write signal is true, in which case, the chip writes data into its storage at the specified address.

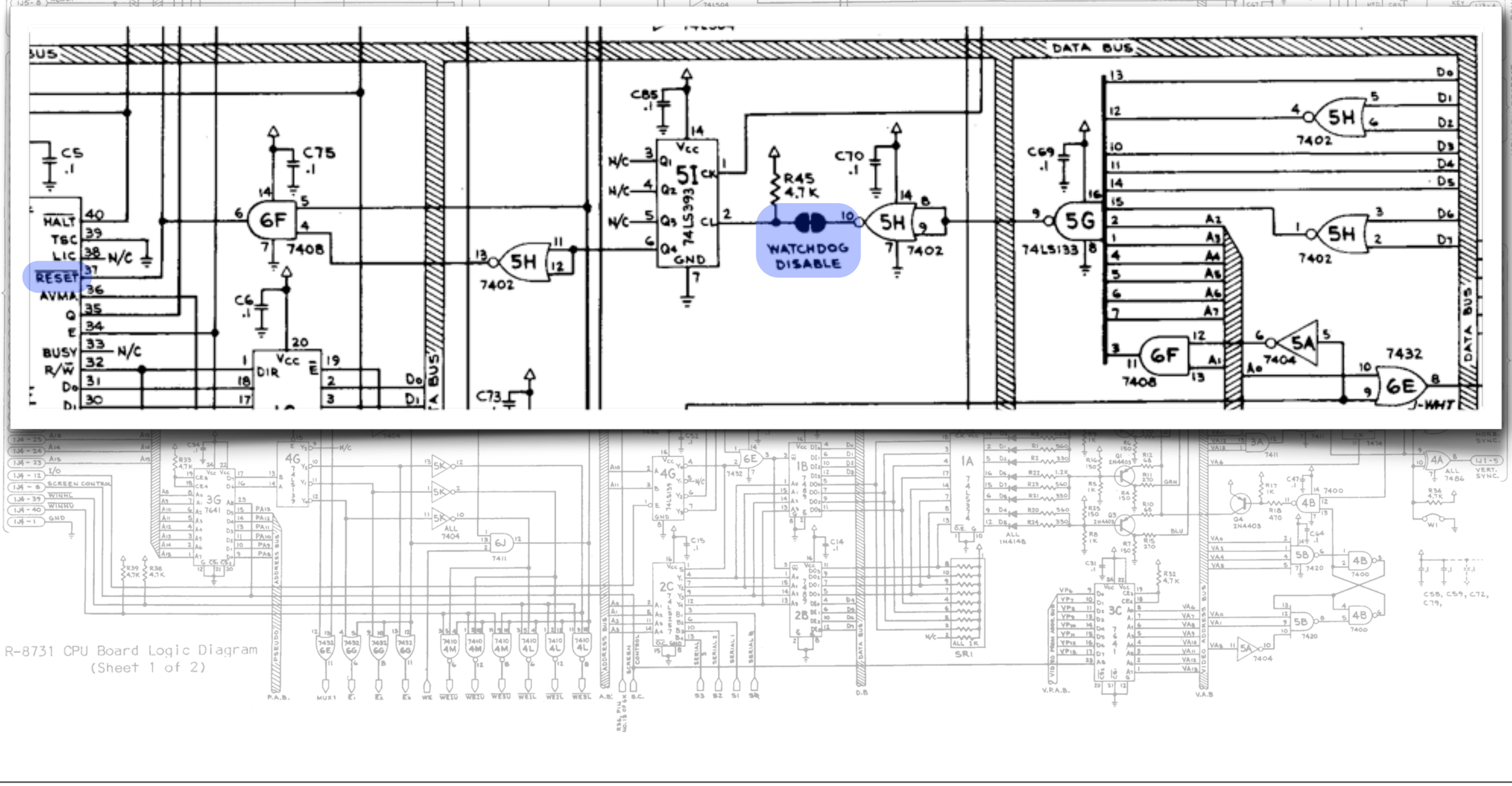
Tracing the Address Bus



Saturday, June 22, 13

There's a battery attached to this chip. If the machine loses power, the data in this memory chip will be preserved. This must be where the high scores and configuration are stored -- for when the arcade turns off the power to the machine every night. Your awesome high score will still be there in the morning!

Watchdog



Saturday, June 22, 13

Here's another chunk of circuitry connected to the processor address and data buses. It has a telling bit of text in the middle of it -- "Watchdog Disable". This must be the hardware watchdog circuit for the game. A hardware watchdog is a circuit that will reset the system if something goes wrong.

Watchdog

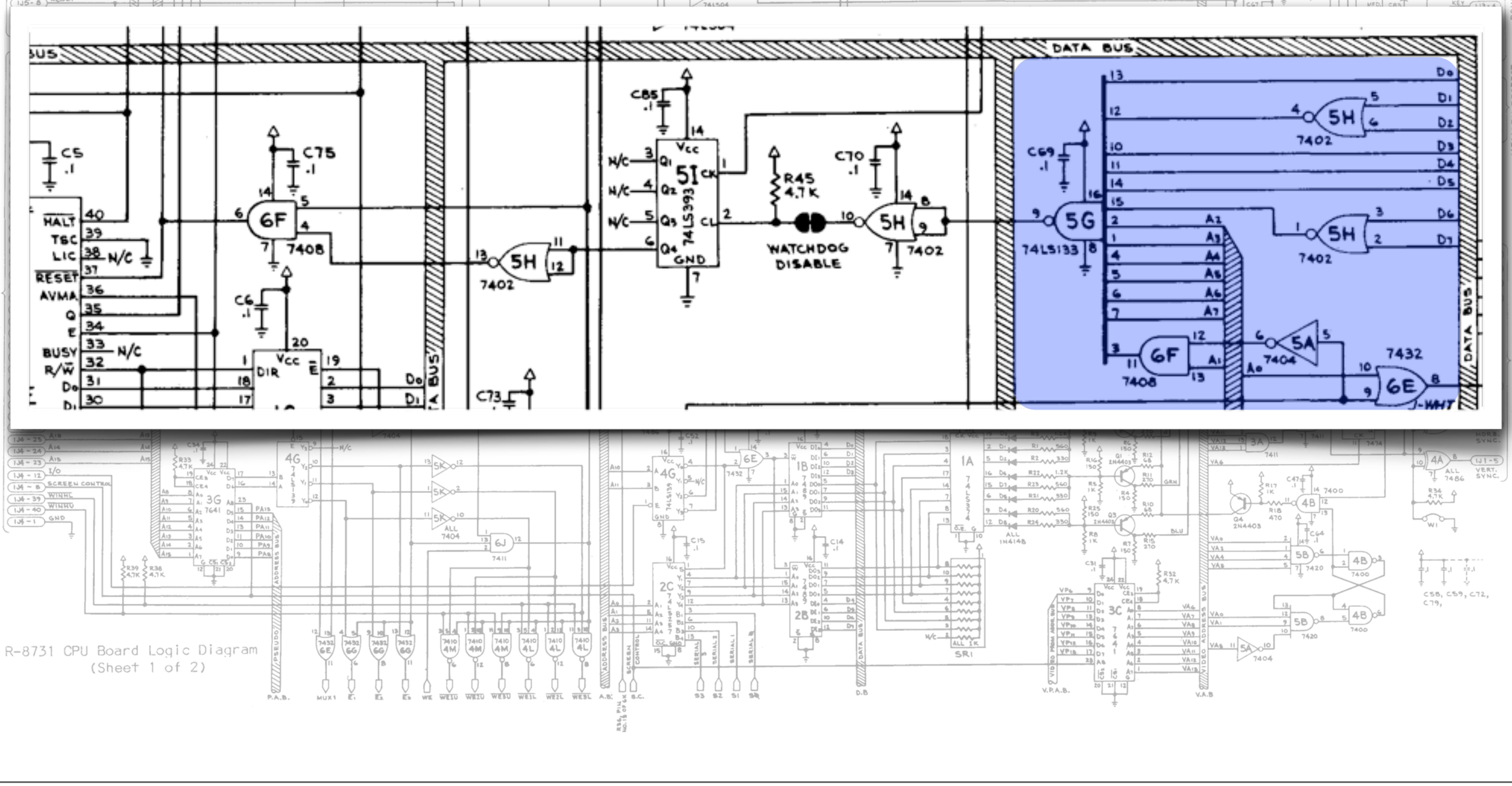
```
process watchdog_timer:
  wait for watchdog_timer_clock.event
  if watchdog_timer_count == timeout_count:
    processor.reset = 1
  else:
    processor.reset = 0
    watchdog_timer_count += 1

process watchdog_reset:
  if memory_address == watchdog_address:
    if memory_data == watchdog_reset_value:
      if write:
        watchdog_timer_count = 0

Application code:
  ... do fun stuff ...
  memory[watchdog_address] = watchdog_reset_value
  ... do more fun stuff ...
```


A common way to do this is by writing your application code so that it periodically resets a timer. The code causes the timer to reset by writing a specific piece of data to a specific memory address. If this isn't done by the time the timer reaches the timeout value, the watchdog circuit resets the processor on the assumption that the program has crashed somehow. (This is long before operating systems and exception handling and all that fancy stuff.)

Watchdog

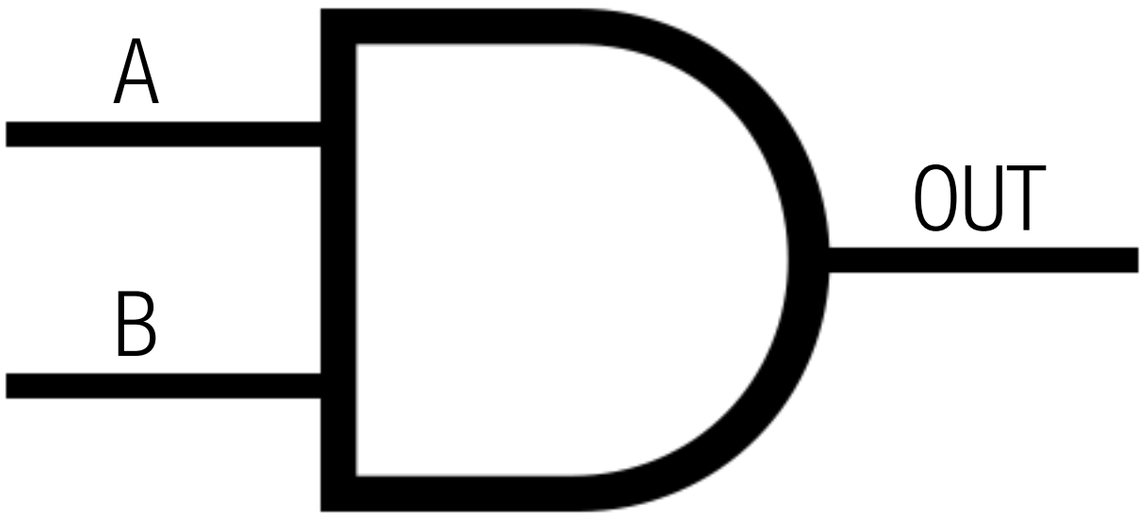


Saturday, June 22, 13

How is this achieved in the Robotron hardware? On the right side is some decoding logic that checks the address and data buses for the specific watchdog values. I'll work through this logic in a moment. But first, a quick detour into how boolean logic operations are depicted on a schematic...

Logic Symbols - AND

```
OUT = A and B
```



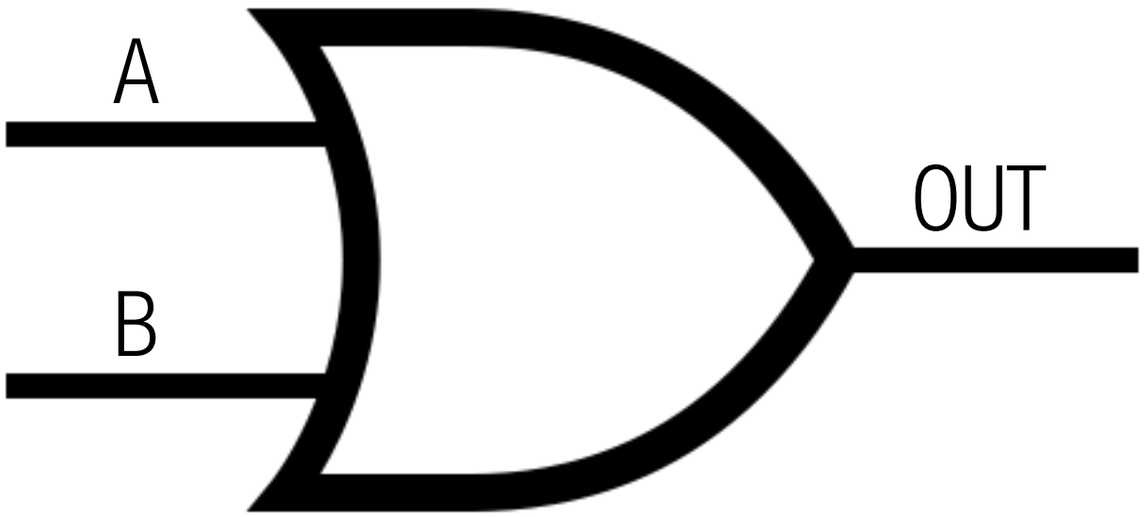
Wikipedia: jjbeard

AND		
A	B	OUT
0	0	0
0	1	0
1	0	0
1	1	1

Here's a two-input AND symbol. Flat on the input sides and rounded on the output side.

Logic Symbols - OR

```
OUT = A or B
```



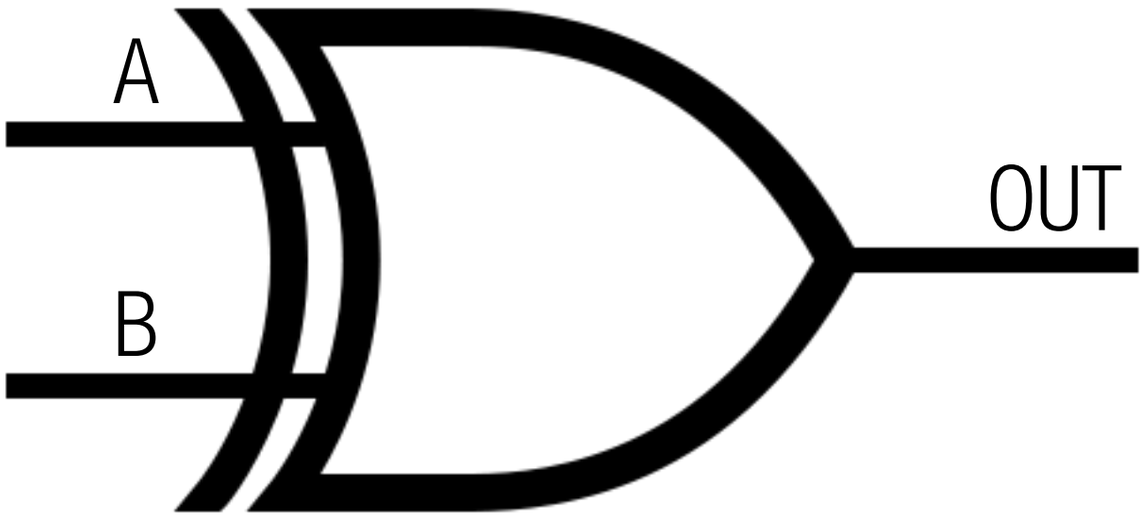
Wikipedia: jjbeard

OR		
A	B	OUT
0	0	0
0	1	1
1	0	1
1	1	1

...a two-input OR. Rounded and pointy on both ends.

Logic Symbols - XOR

```
OUT = (A != B)
```



Wikipedia: jjbeard

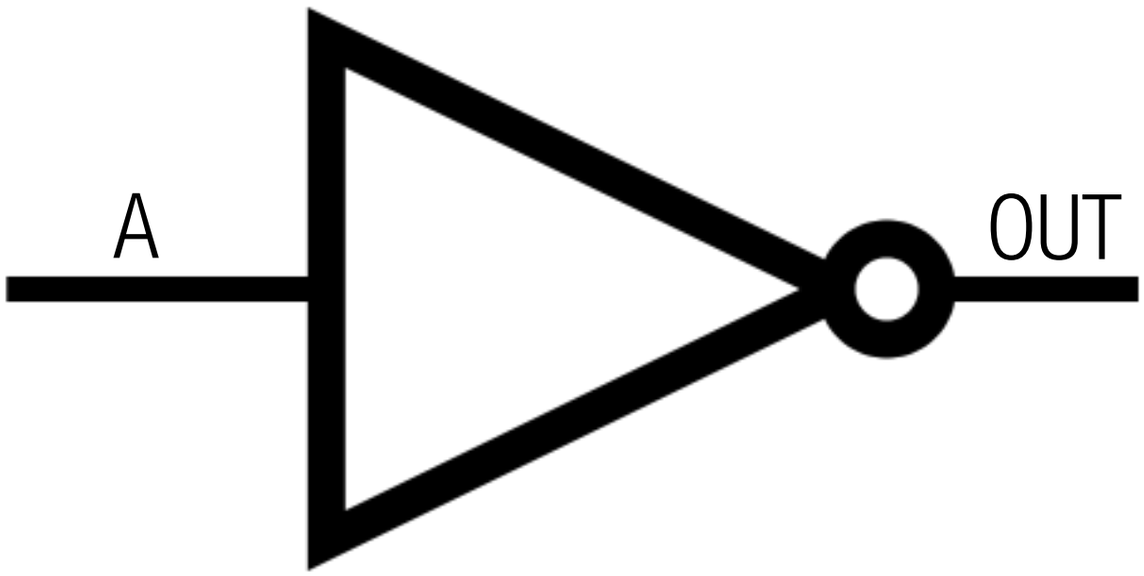
XOR		
A	B	OUT
0	0	0
0	1	1
1	0	1
1	1	0

...a two-input XOR, like an OR, but with an extra line across the input of the symbol.

An XOR is true only when the inputs are *not* the same.

Logic Symbols - NOT

```
OUT = not A
```



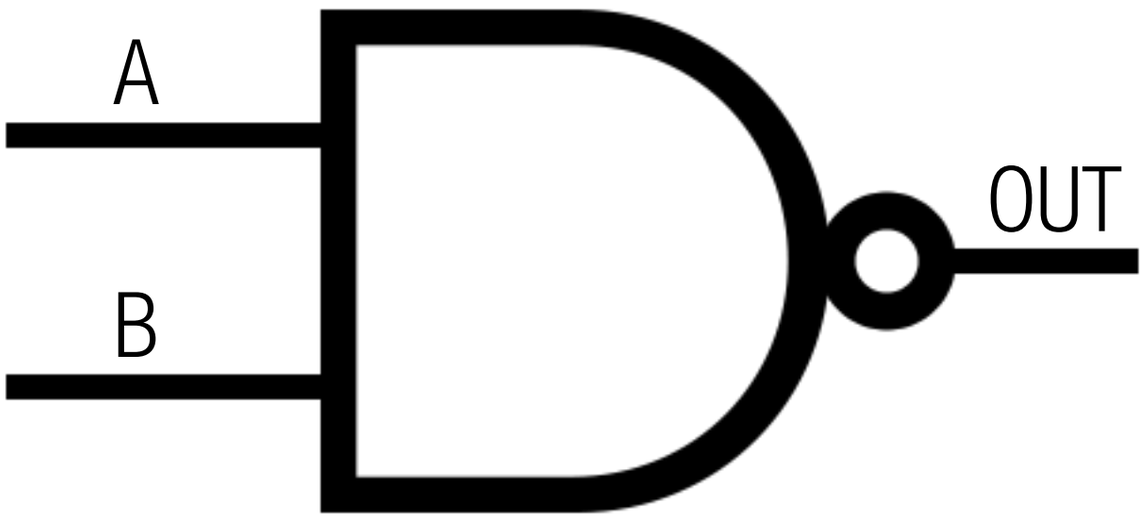
Wikipedia: jjbeard

NOT	
A	OUT
0	1
1	0

...this is a NOT symbol, also referred to as an inverter. Notice the circle on the output...

Logic Symbols - NAND

```
OUT = not (A and B)
```



Wikipedia: jjbeard

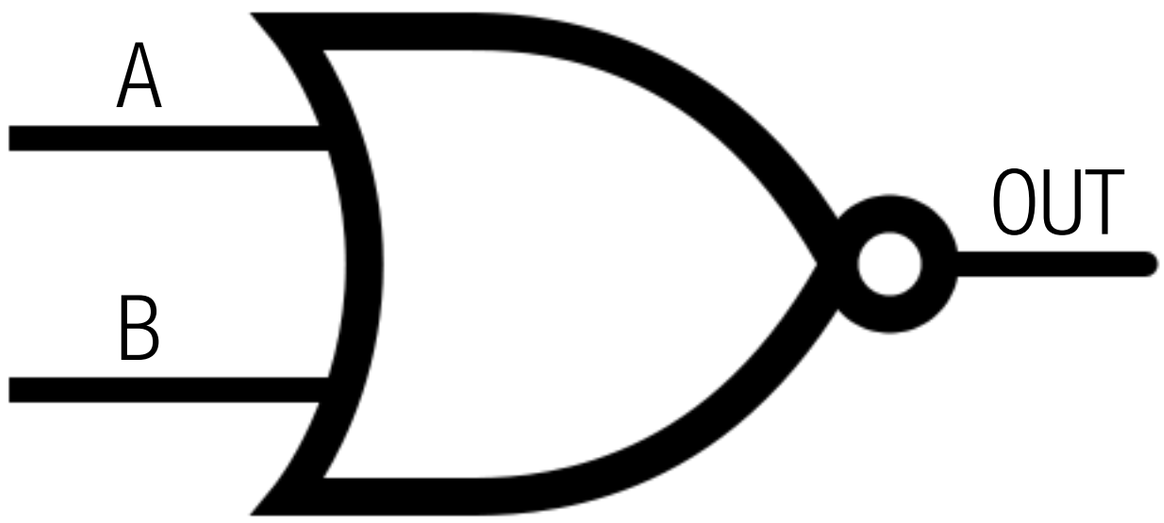
NAND		
A	B	OUT
0	0	1
0	1	1
1	0	1
1	1	0

Saturday, June 22, 13

...there's a circle on the output of this AND symbol, which *inverts* the AND output and makes it a "not-AND" or NAND function.

Logic Symbols - NOR

```
OUT = not (A or B)
```

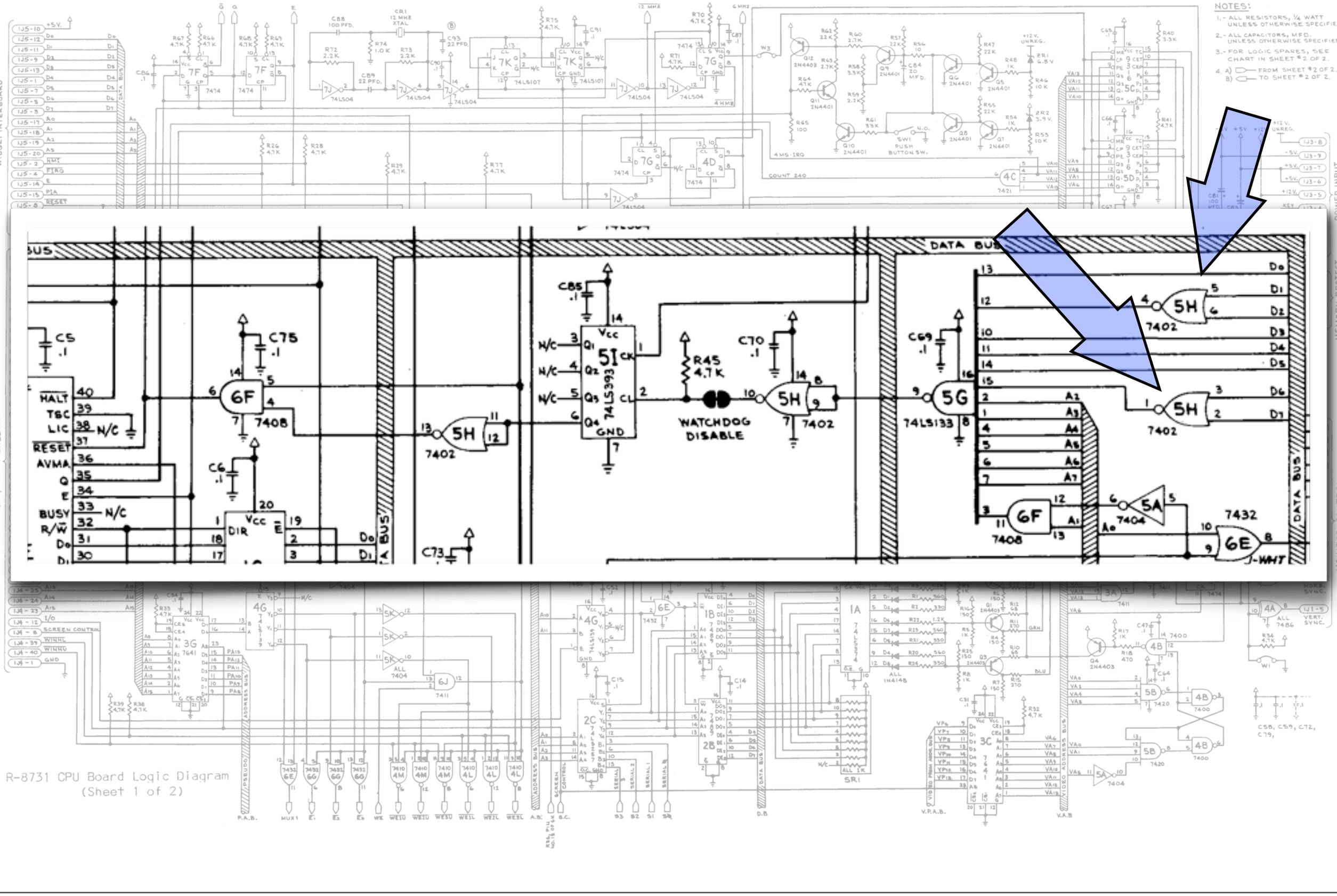


Wikipedia: jjbeard

NOR		
A	B	OUT
0	0	1
0	1	0
1	0	0
1	1	0

The same thing applies here... An OR symbol with a circle on the output becomes a “not-OR” or NOR function.

Watchdog



Saturday, June 22, 13

Back to the watchdog area of the schematic. Here are two NOR gates.

Watchdog

x or y

x or y

R-8731 CPU Board Logic Diagram
(Sheet 1 of 2)

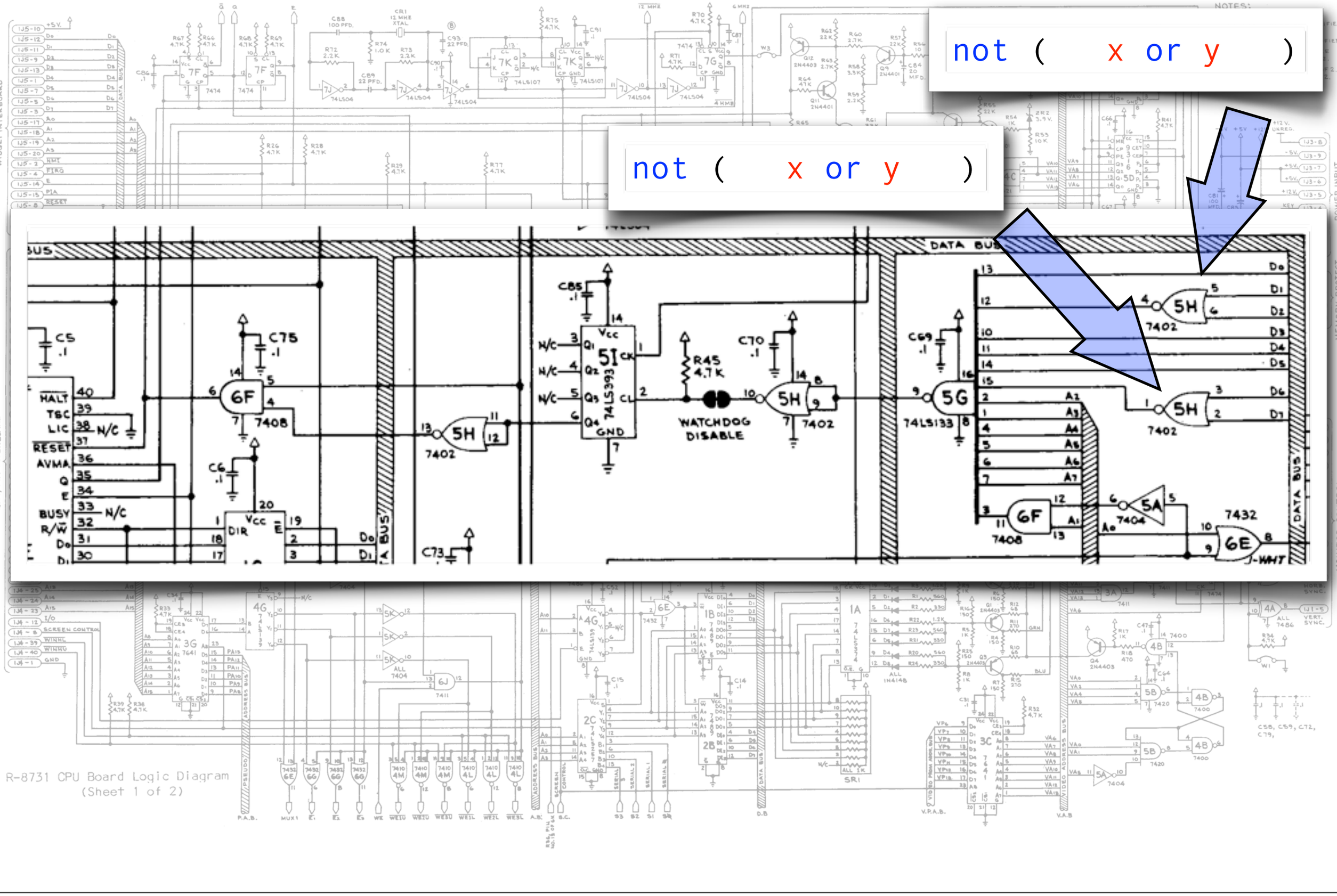
Saturday, June 22, 13

The shape of this symbol represents an OR function...

Watchdog

```
not (    x or y    )
```


```
not (    x or y    )
```

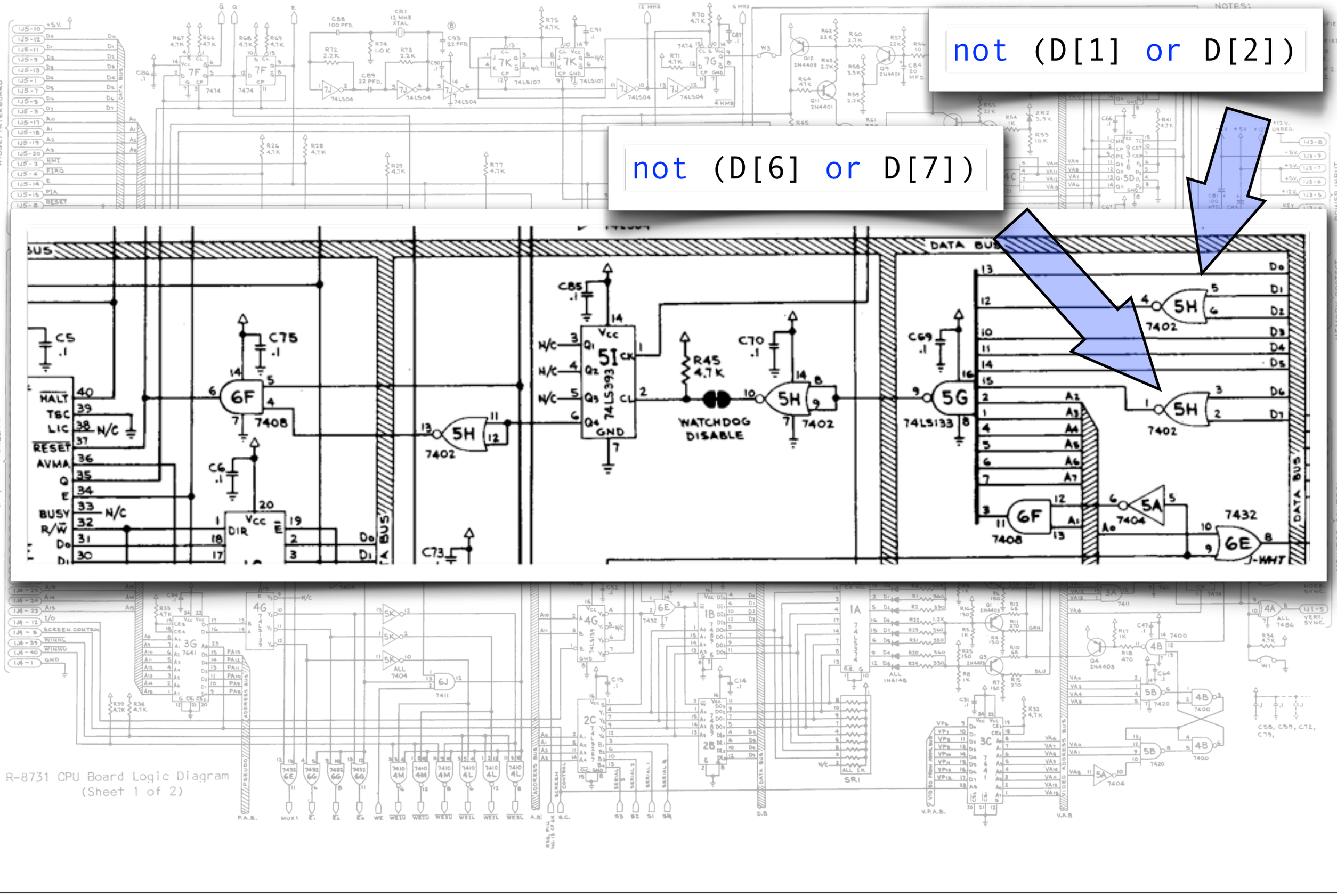
Saturday, June 22, 13

..but the circle on the output means that the output is inverted, making it a NOR function.

Watchdog

```
not (D[1] or D[2])
```

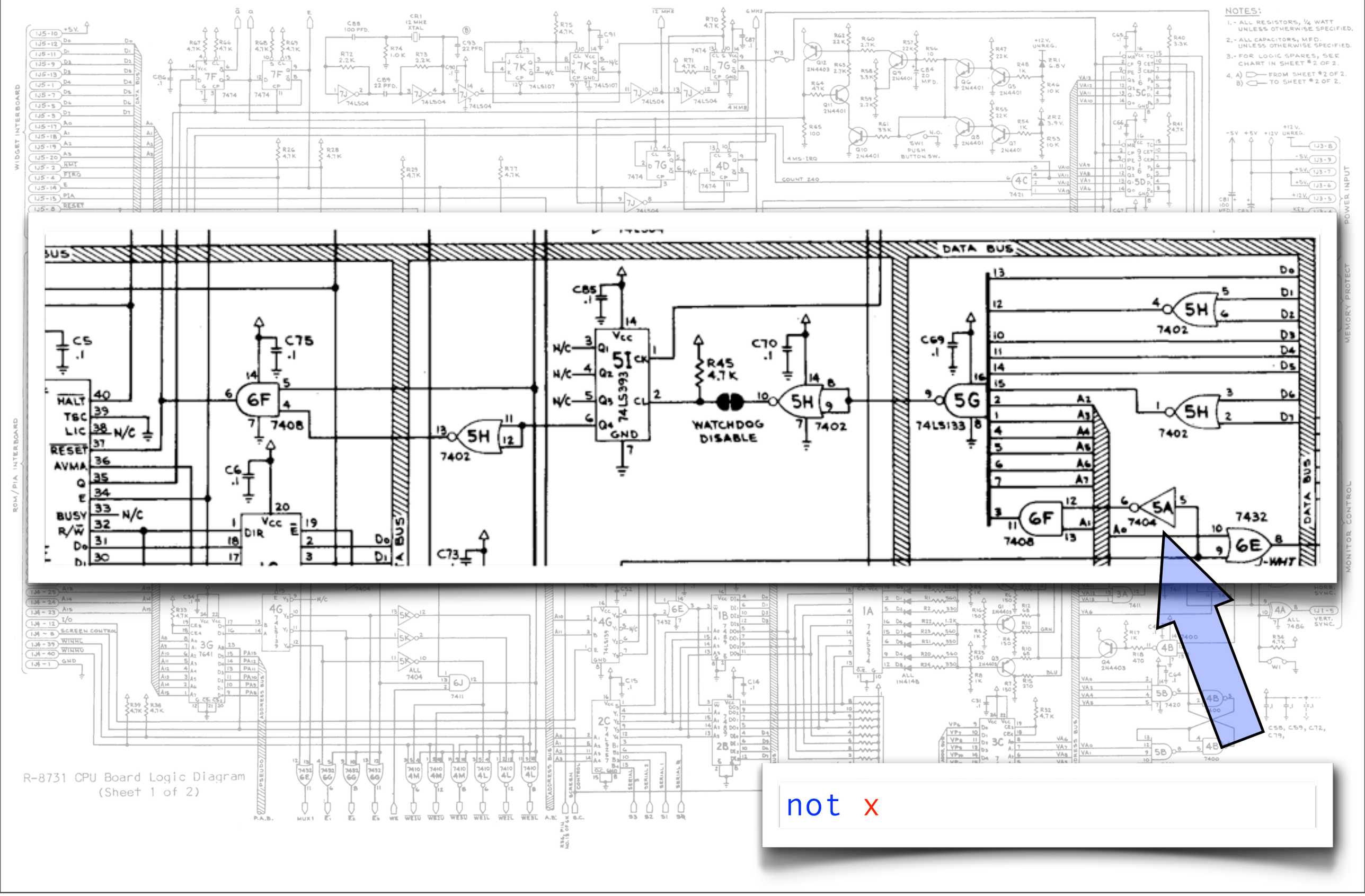
```
not (D[6] or D[7])
```



Saturday, June 22, 13

In this case, they're EACH taking two bits from the data bus as input.

Watchdog

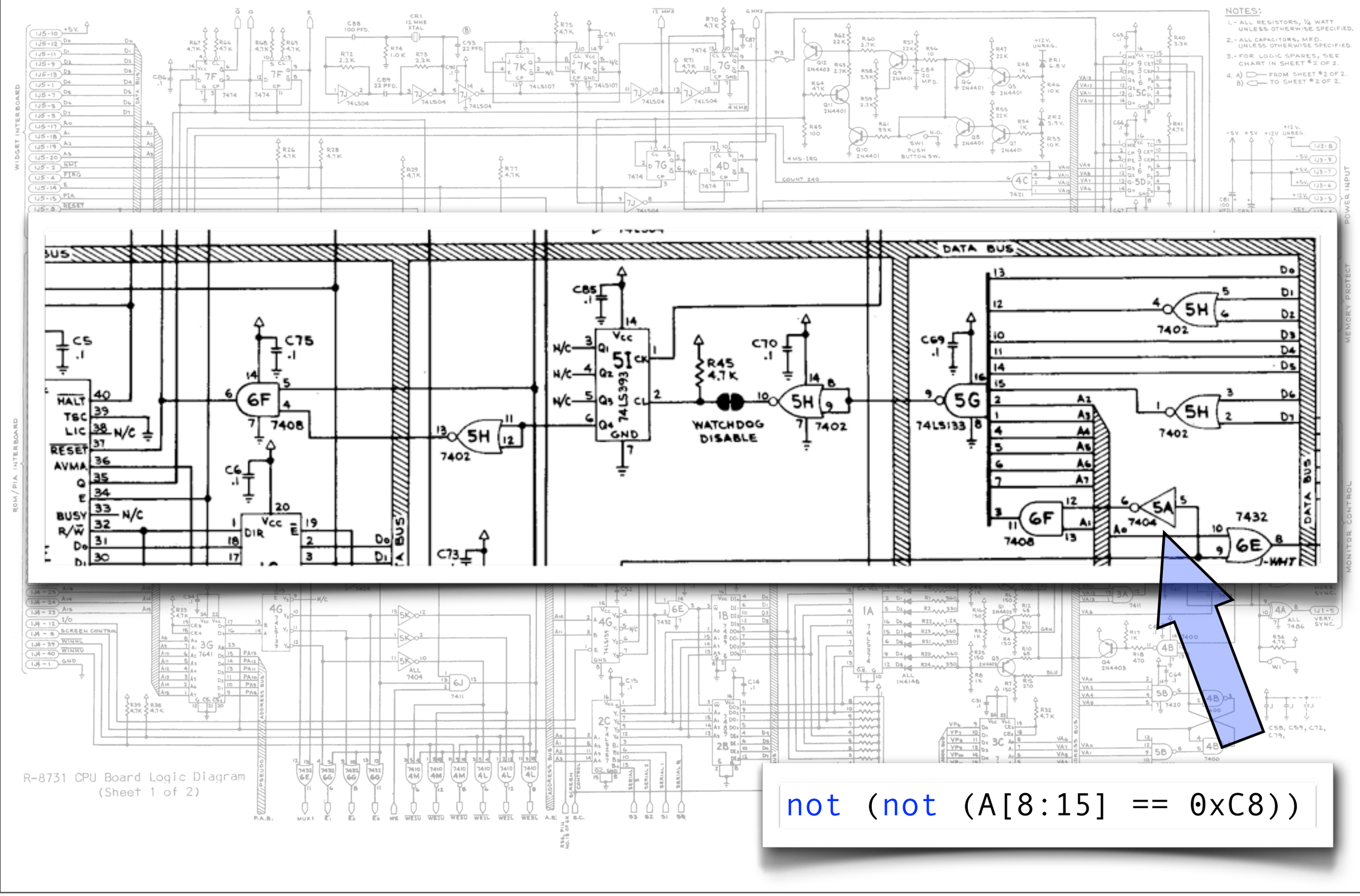



```
not x
```

Saturday, June 22, 13

And at the bottom, we have a NOT function, which just inverts the boolean value that goes in to it.

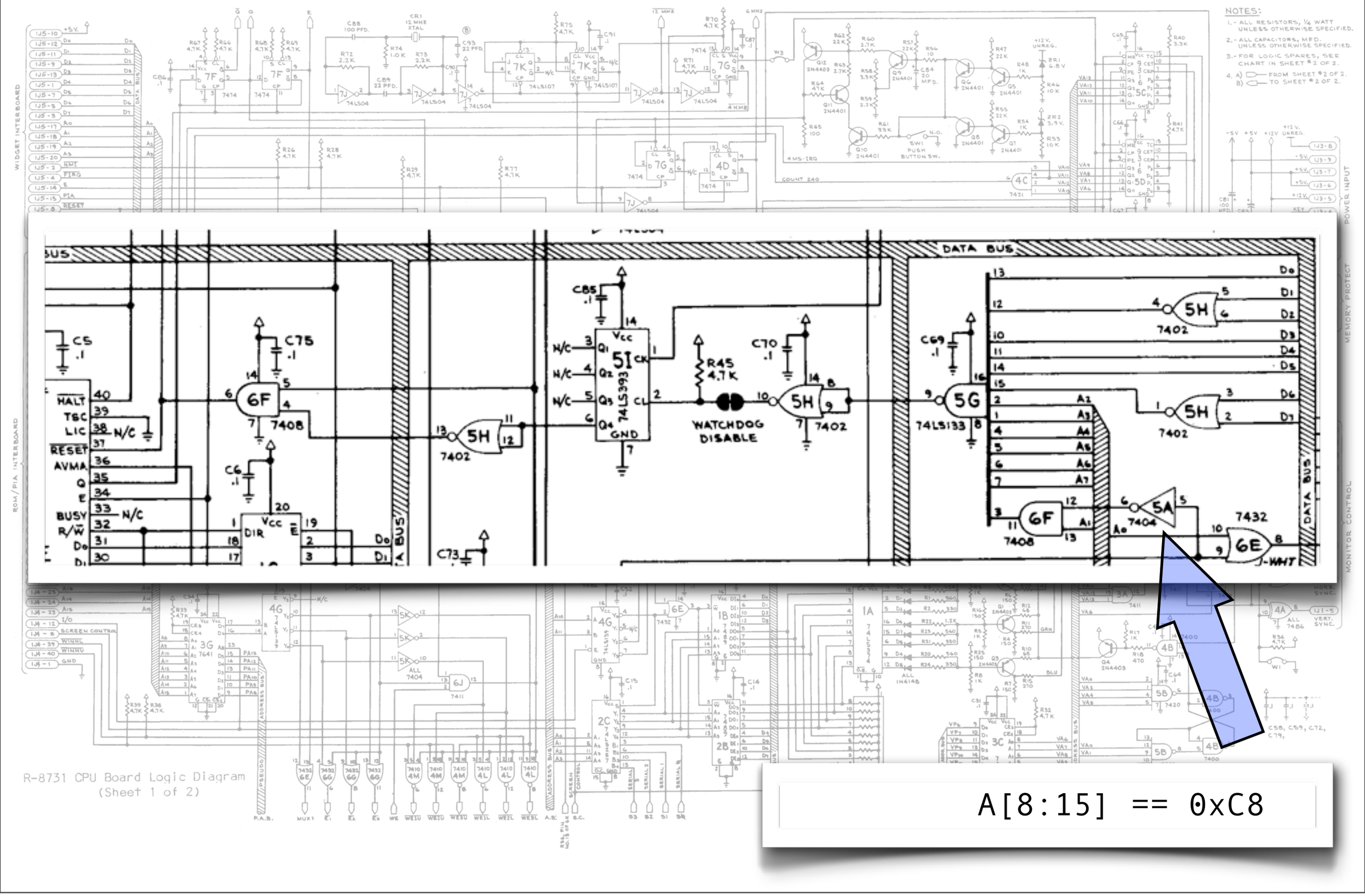
Watchdog



Saturday, June 22, 13

In this case, that value comes from somewhere else on the schematic, where address signals 8 through 15 are compared against C8 in hexadecimal.

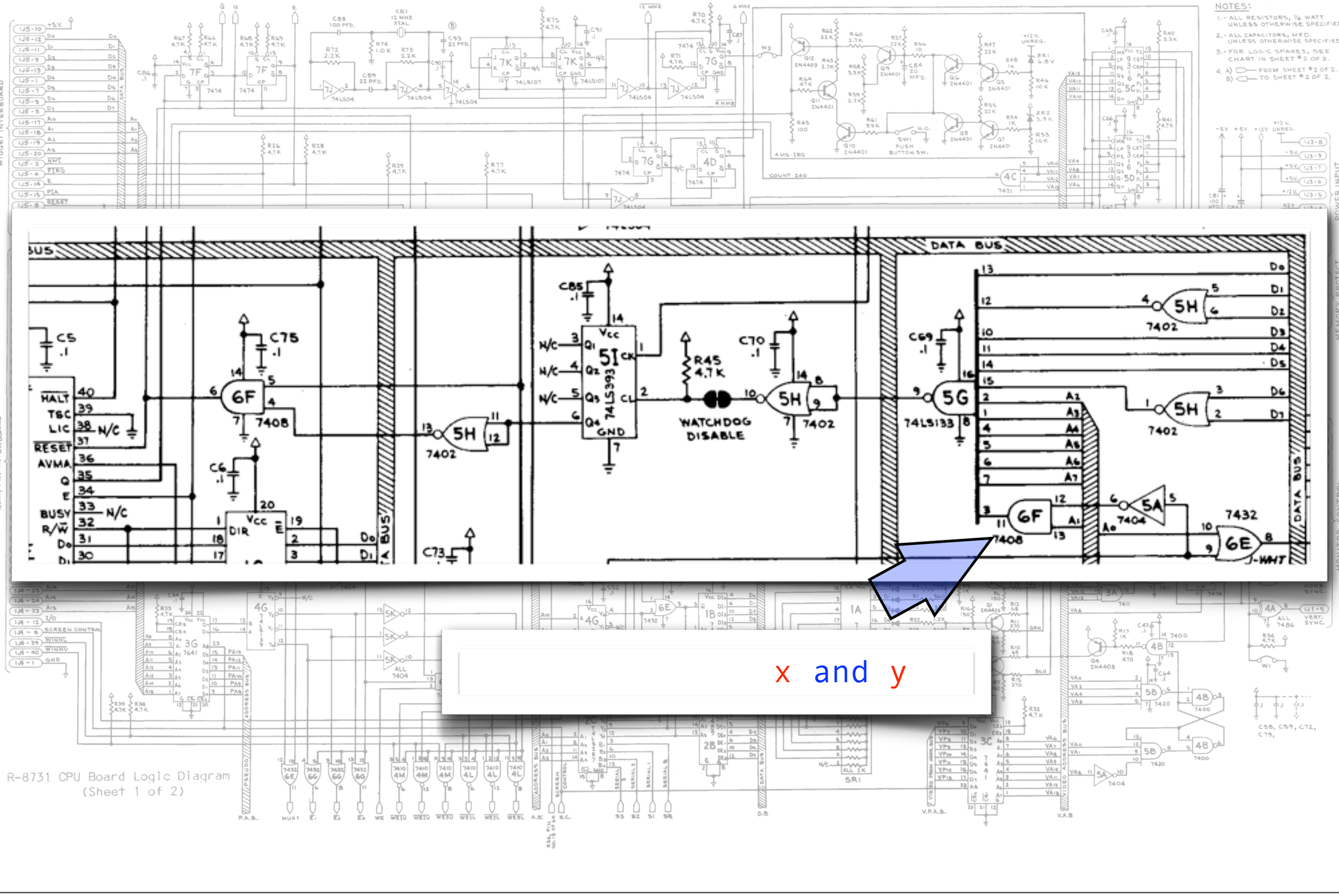
Watchdog



Saturday, June 22, 13

Two wrongs make a right (if you invert a value twice, you get the original value), so the two NOTs cancel out, and we're left with a simple comparison.

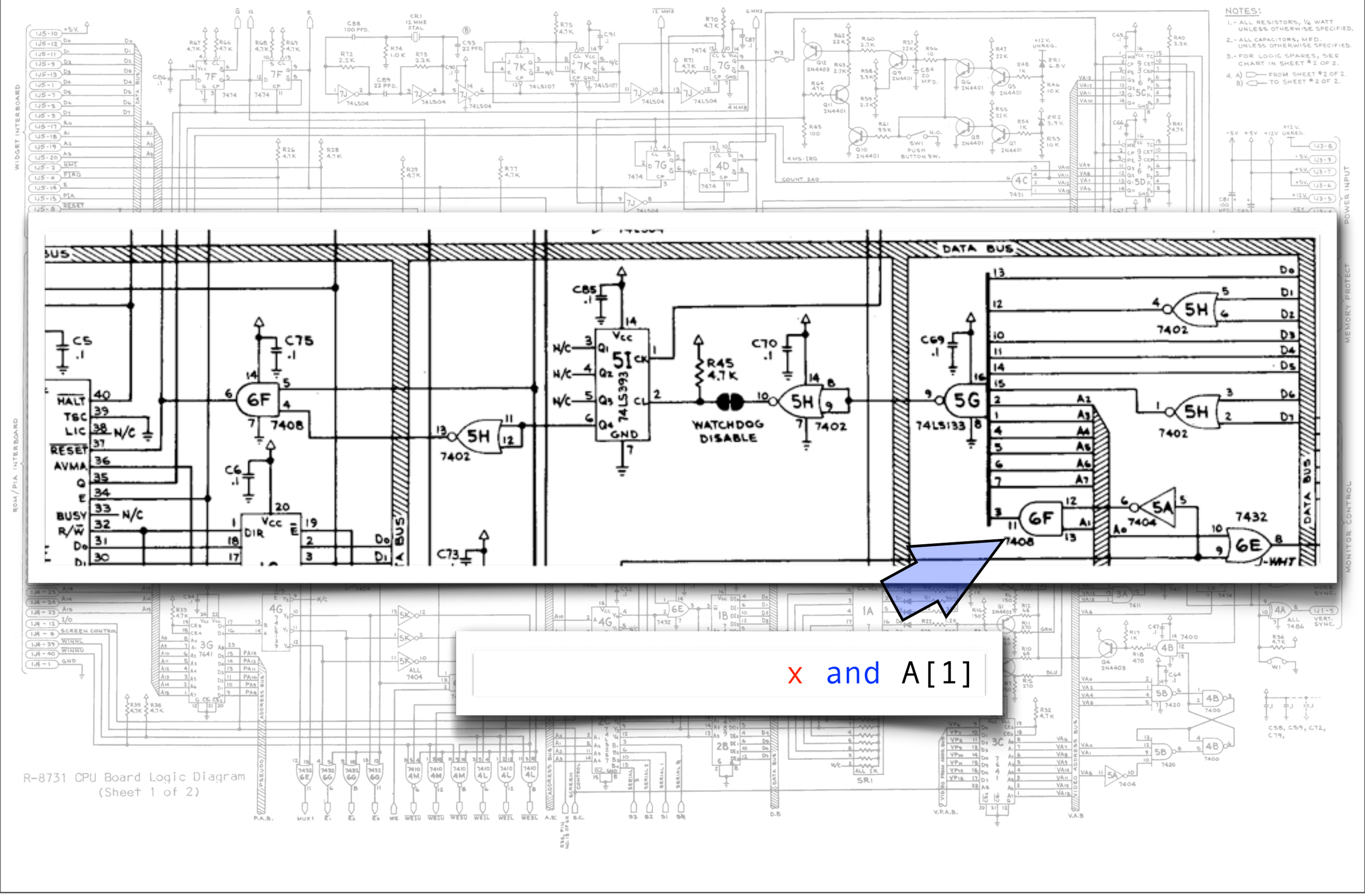
Watchdog



Saturday, June 22, 13

Here's an AND function. The output is a one only if both inputs are ones.

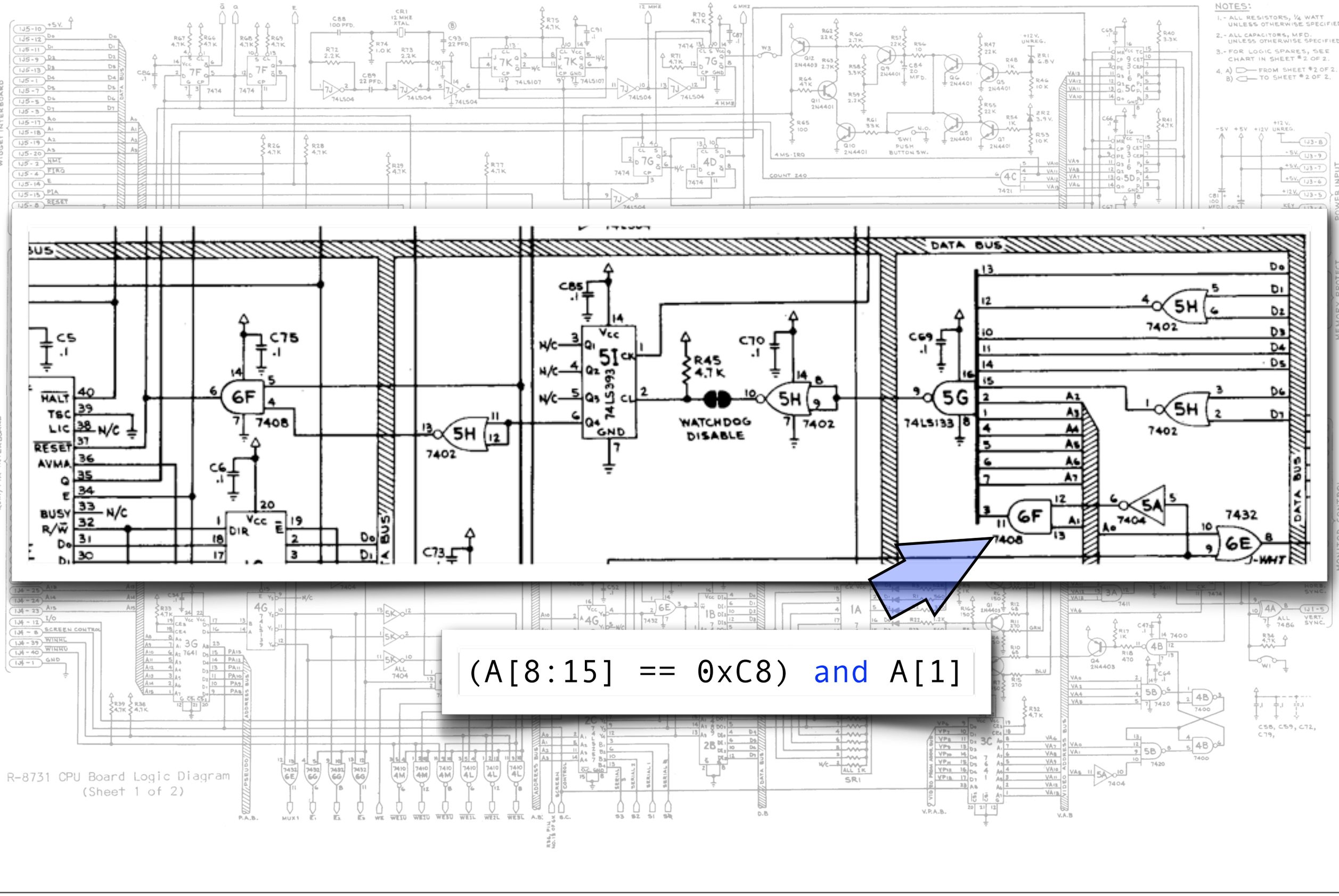
Watchdog



Saturday, June 22, 13

One of the inputs is address bit 1.

Watchdog

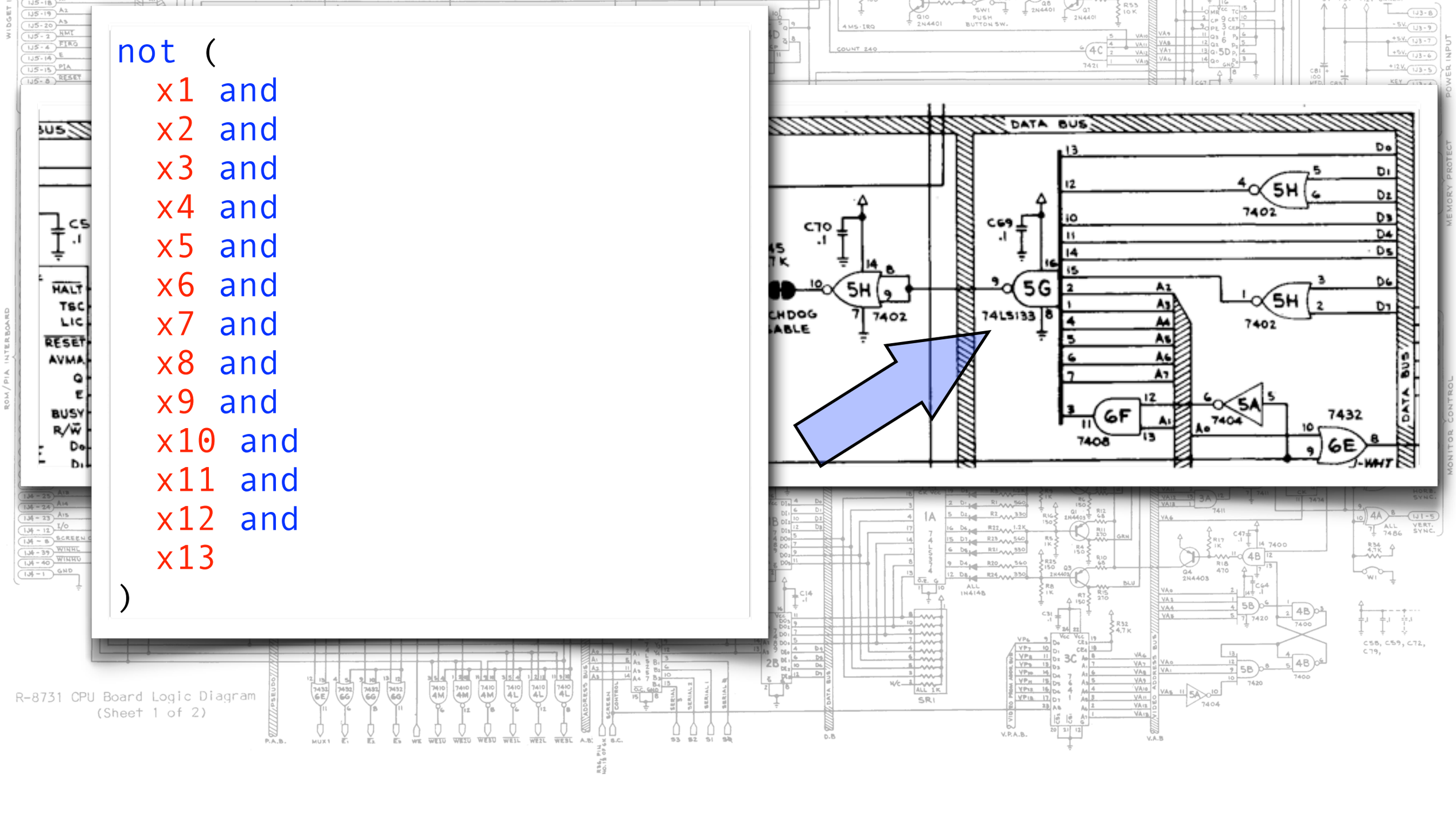


Saturday, June 22, 13

The other input comes from the output of the NOT function from a few slides ago.

Watchdog

```
not (
   x1 and
   x2 and
   x3 and
   x4 and
   x5 and
   x6 and
   x7 and
   x8 and
   x9 and
   x10 and
   x11 and
   x12 and
   x13
)
```



Saturday, June 22, 13

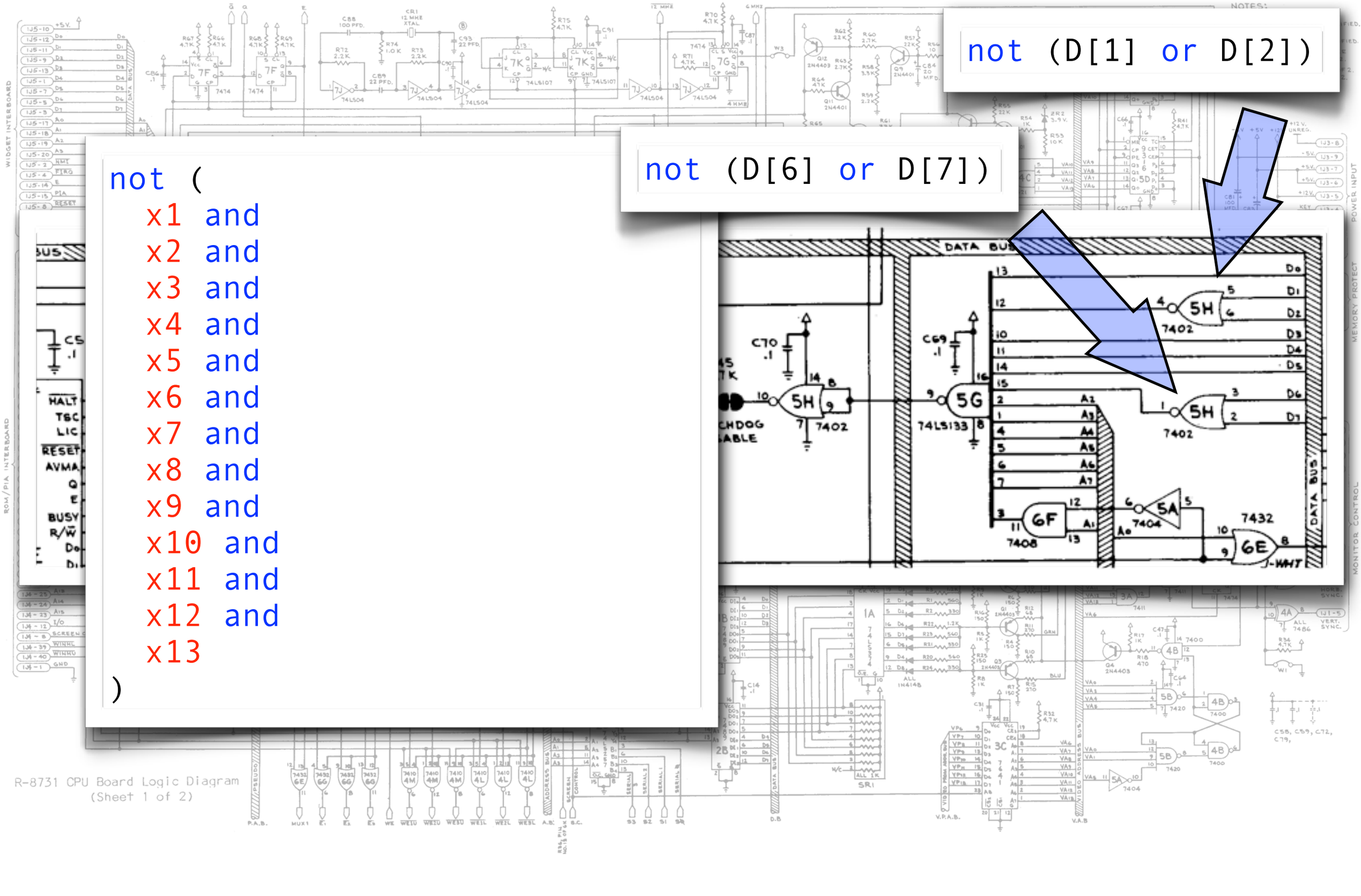
All of these functions flow into a giant NAND function -- an AND function with 13 inputs and an inverted output.

Watchdog

```
not (
   x1 and
   x2 and
   x3 and
   x4 and
   x5 and
   x6 and
   x7 and
   x8 and
   x9 and
   x10 and
   x11 and
   x12 and
   x13
)
```

```
not (D[6] or D[7])
```

```
not (D[1] or D[2])
```



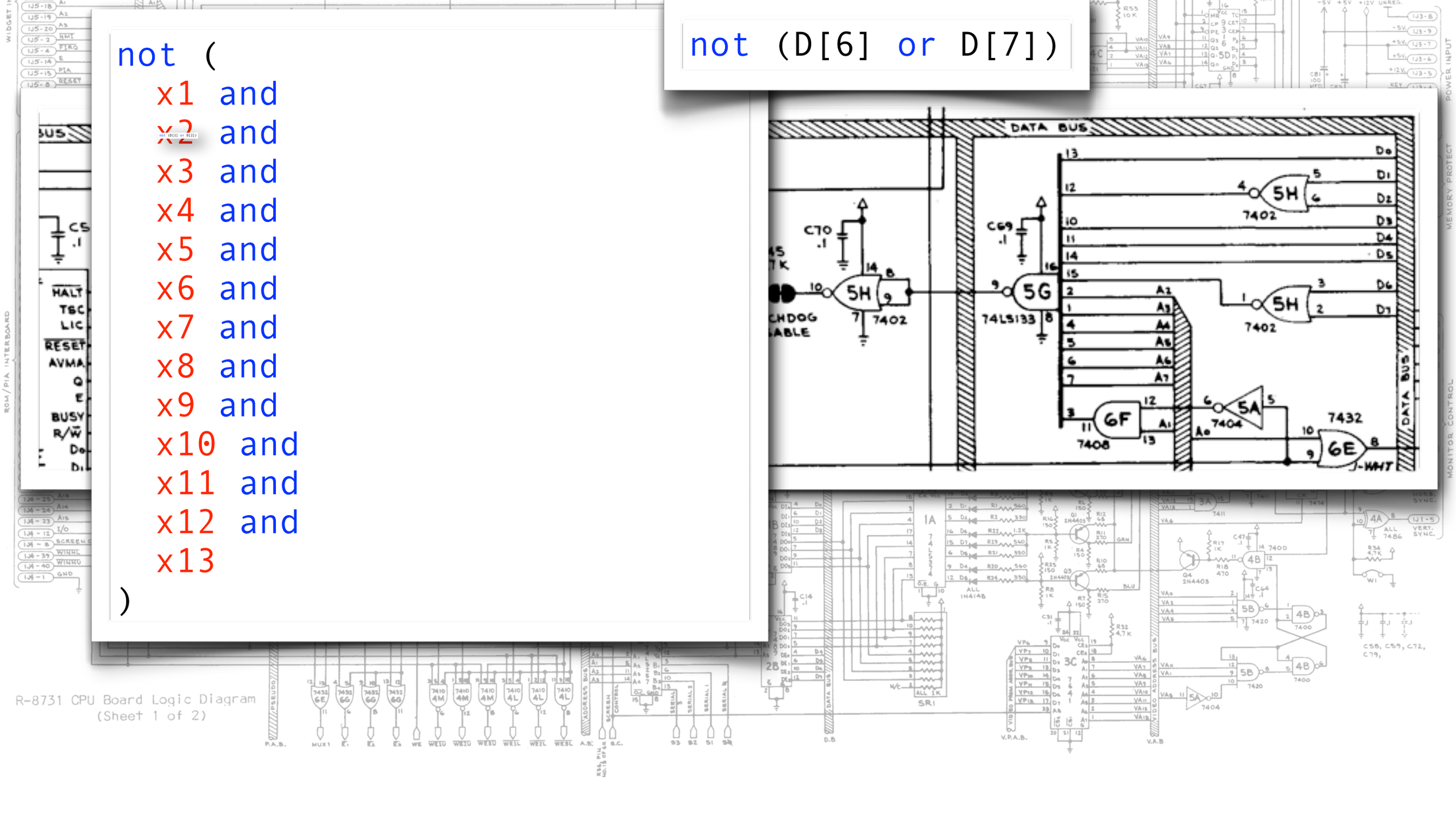
Saturday, June 22, 13

The two NOR functions go into the NAND... ***next***

Watchdog

```
not (
   x1 and
   x2 and
   x3 and
   x4 and
   x5 and
   x6 and
   x7 and
   x8 and
   x9 and
   x10 and
   x11 and
   x12 and
   x13
)
```

```
not (D[6] or D[7])
```



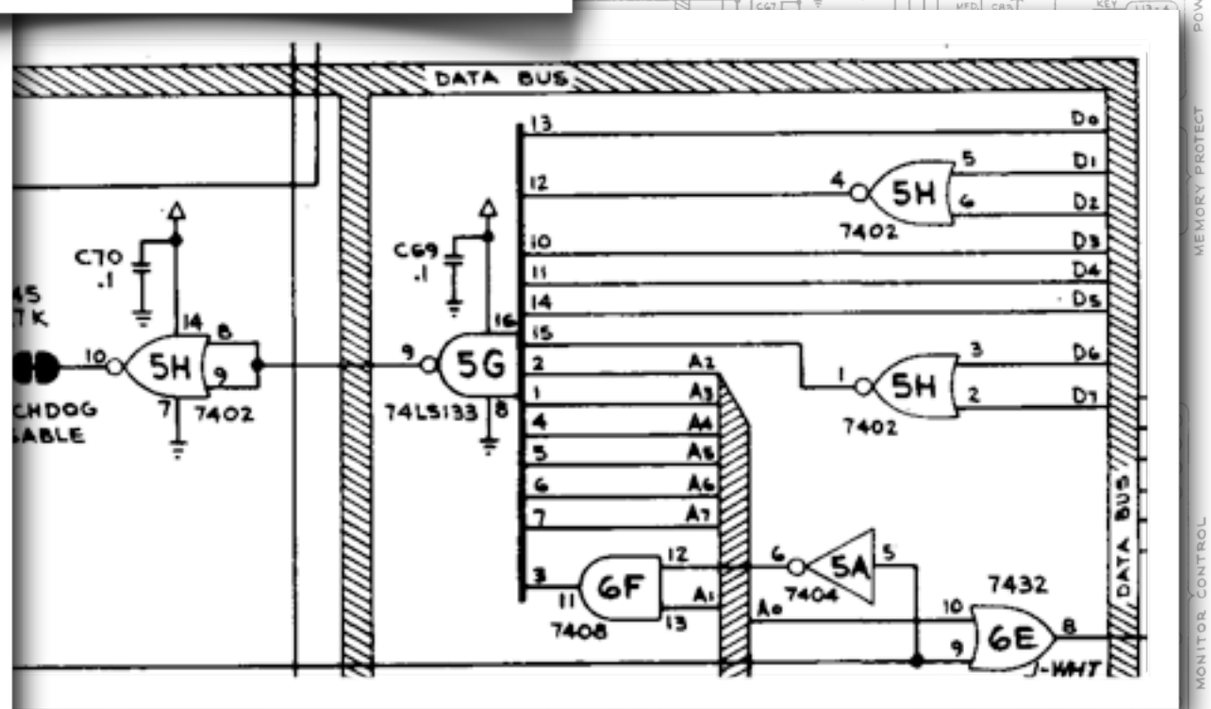
Saturday, June 22, 13

The two NOR functions go into the NAND... ***next***

Watchdog

```
not (
  x1 and
  (not (D[1] or D[2])) and
  x3 and
  x4 and
  x5 and
  x6 and
  x7 and
  x8 and
  x9 and
  x10 and
  x11 and
  x12 and
  x13
)
```

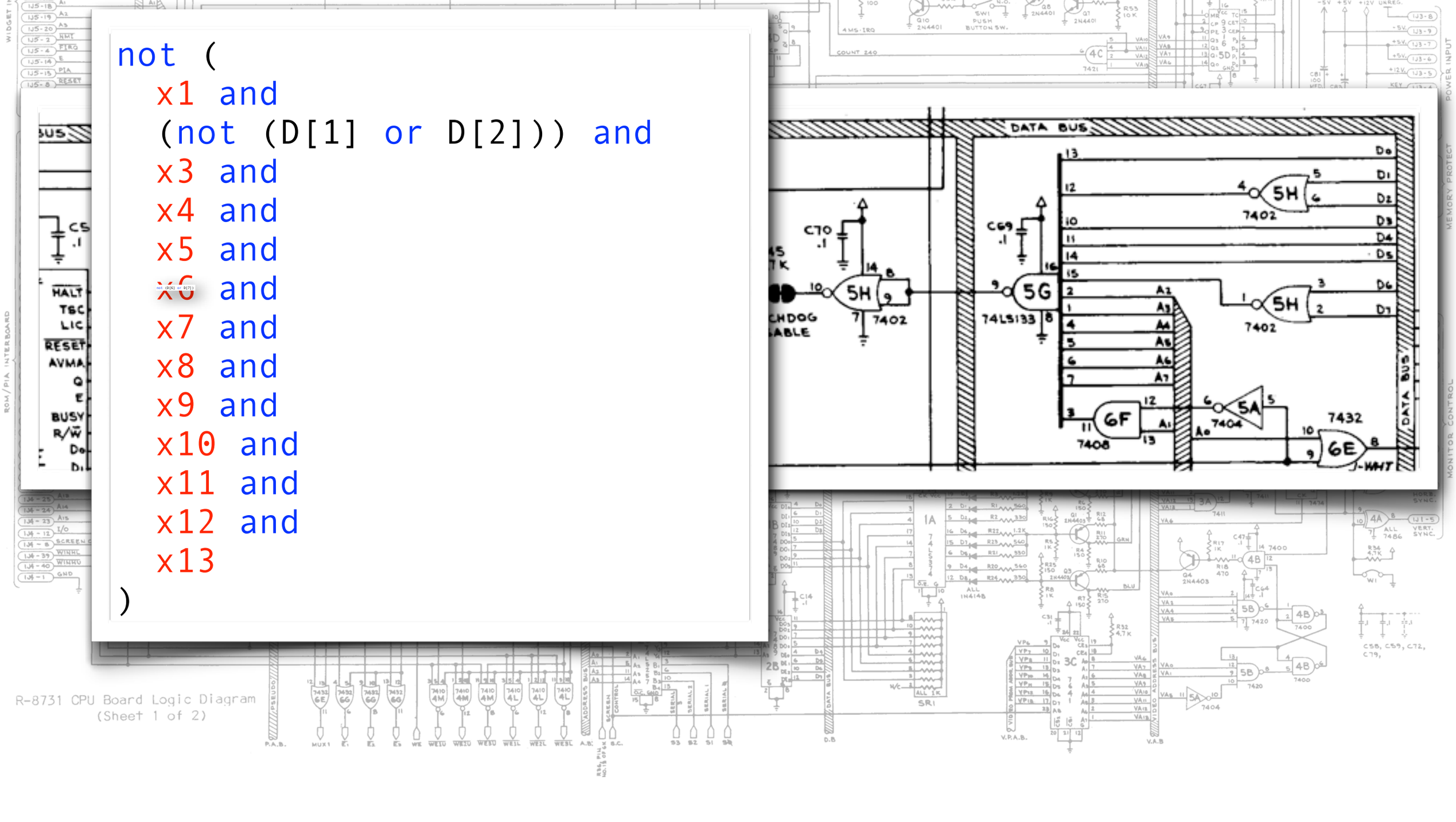
```
not (D[6] or D[7])
```



R-8731 CPU Board Logic Diagram
(Sheet 1 of 2)

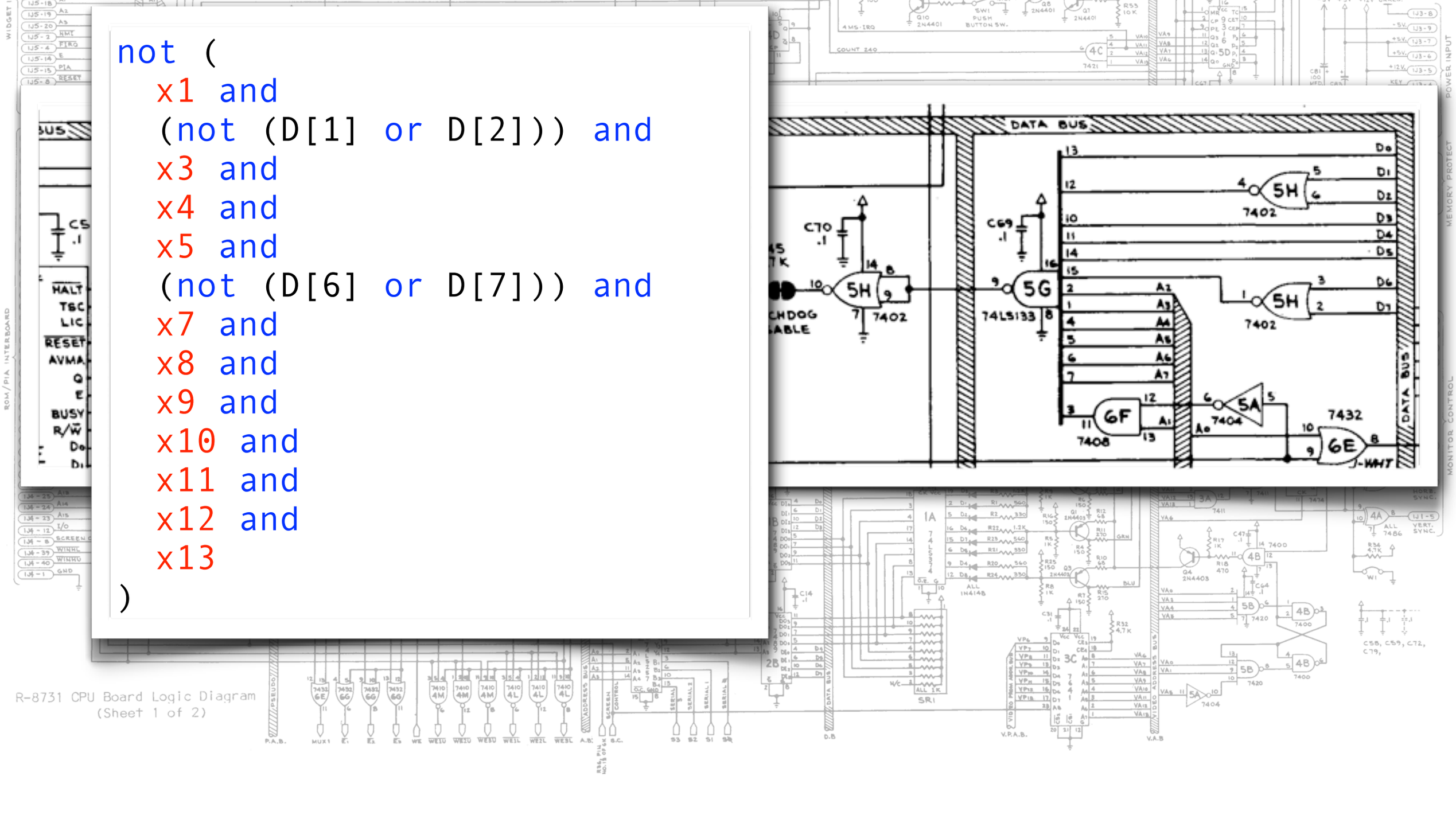
Watchdog

```
not (
   x1 and
   (not (D[1] or D[2])) and
   x3 and
   x4 and
   x5 and
   x6 and
   x7 and
   x8 and
   x9 and
   x10 and
   x11 and
   x12 and
   x13
)
```



Watchdog

```
not (
  x1 and
  (not (D[1] or D[2])) and
  x3 and
  x4 and
  x5 and
  (not (D[6] or D[7])) and
  x7 and
  x8 and
  x9 and
  x10 and
  x11 and
  x12 and
  x13
)
```



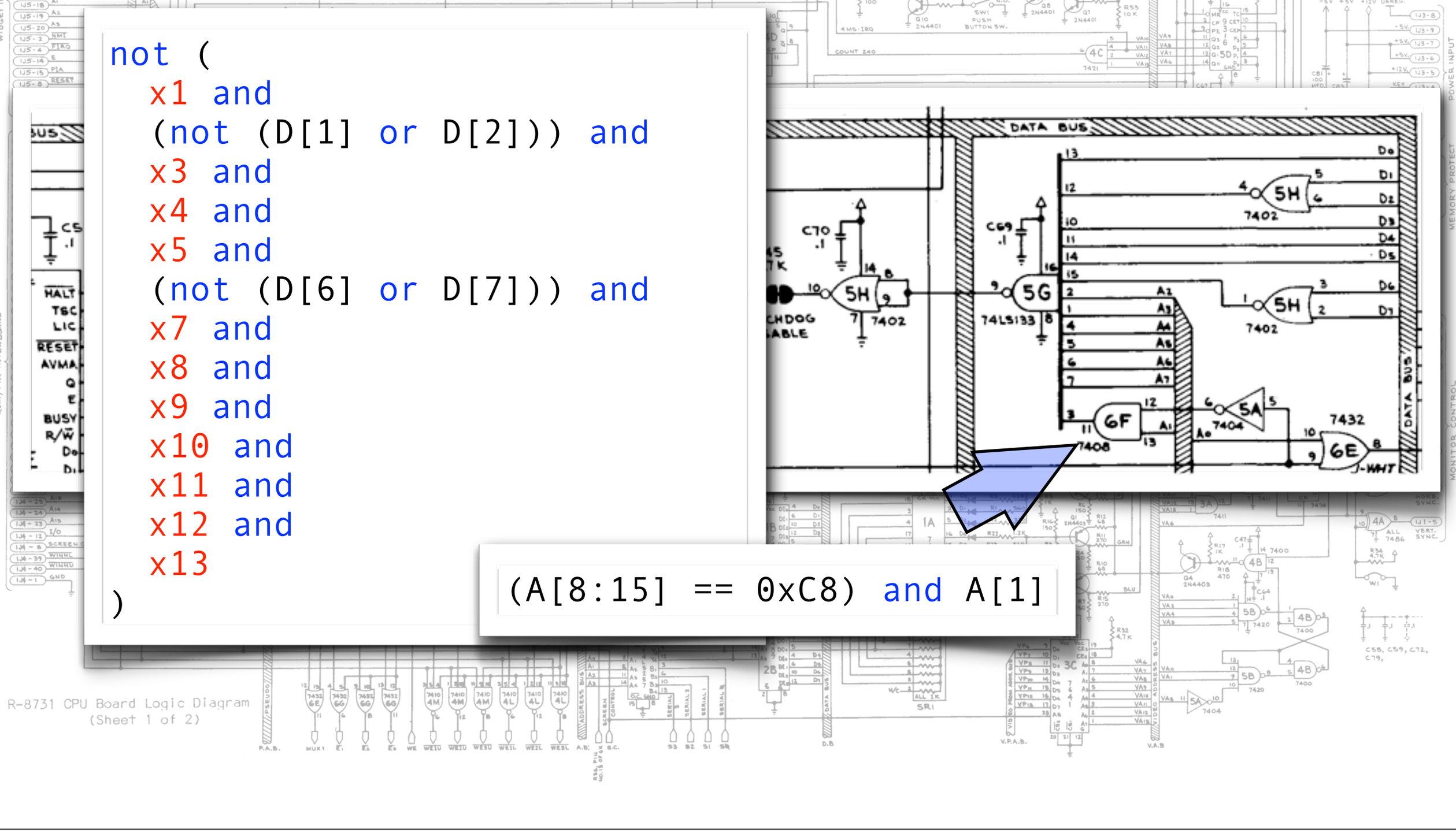
Saturday, June 22, 13

*** next ***

Watchdog

```
not (
   x1 and
   (not (D[1] or D[2])) and
   x3 and
   x4 and
   x5 and
   (not (D[6] or D[7])) and
   x7 and
   x8 and
   x9 and
   x10 and
   x11 and
   x12 and
   x13
)
```

```
(A[8:15] == 0xC8) and A[1]
```

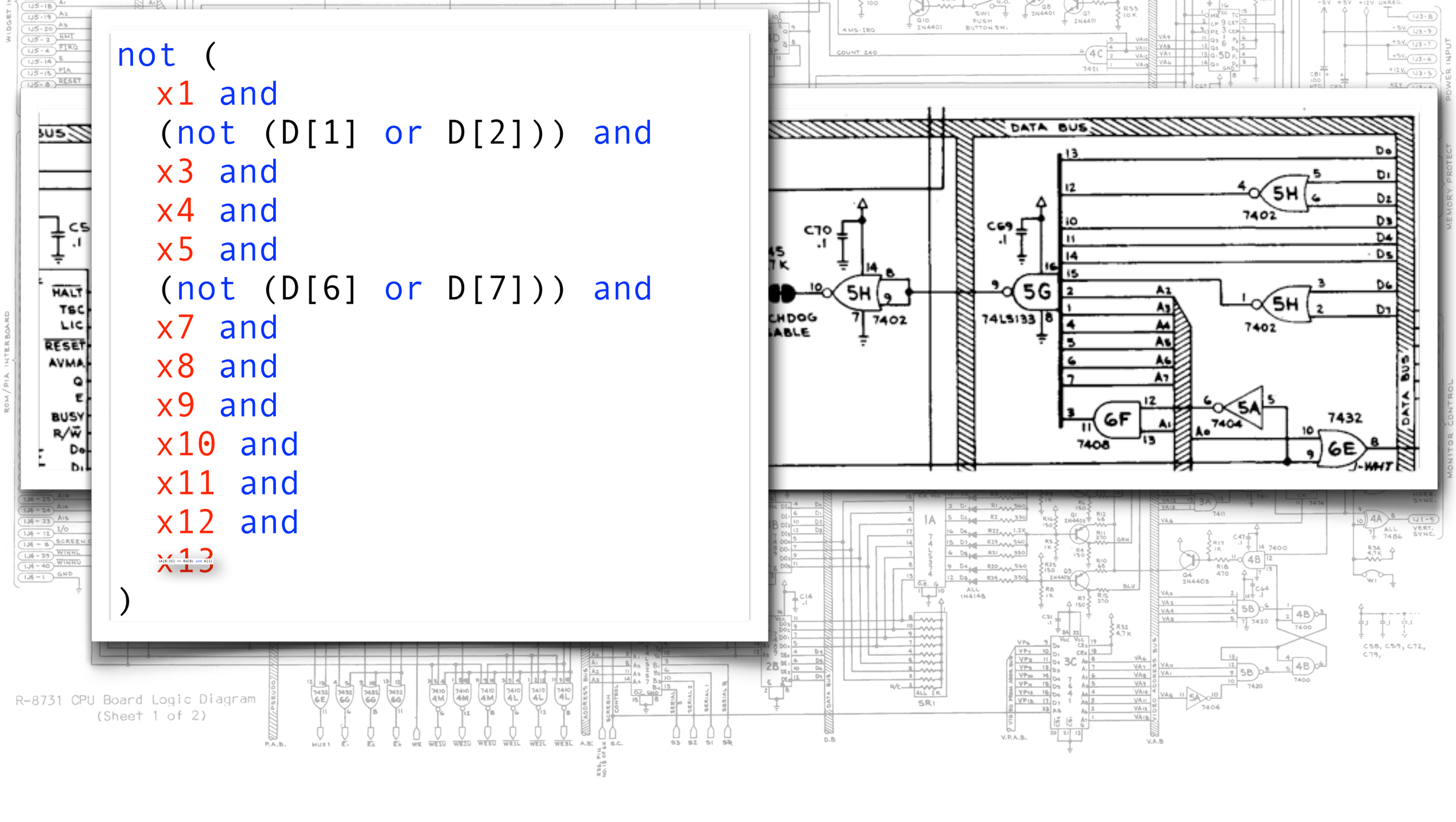


Saturday, June 22, 13

The AND function goes into the NAND... ***next***

Watchdog

```
not (
  x1 and
  (not (D[1] or D[2])) and
  x3 and
  x4 and
  x5 and
  (not (D[6] or D[7])) and
  x7 and
  x8 and
  x9 and
  x10 and
  x11 and
  x12 and
  x13
)
```

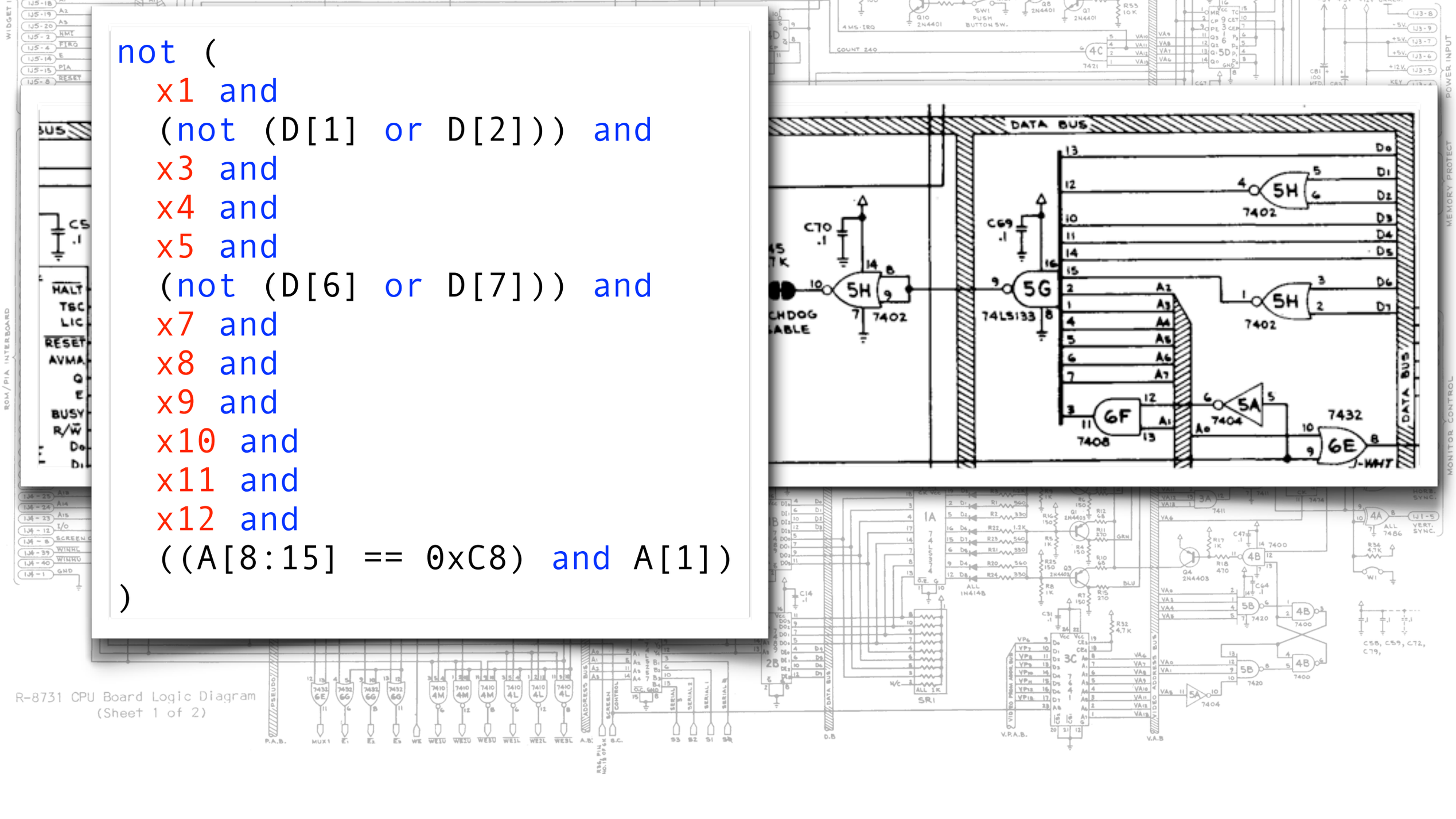


Saturday, June 22, 13

The AND function goes into the NAND... ***next***

Watchdog

```
not (
   x1 and
   (not (D[1] or D[2])) and
   x3 and
   x4 and
   x5 and
   (not (D[6] or D[7])) and
   x7 and
   x8 and
   x9 and
   x10 and
   x11 and
   x12 and
   ((A[8:15] == 0xC8) and A[1])
)
```

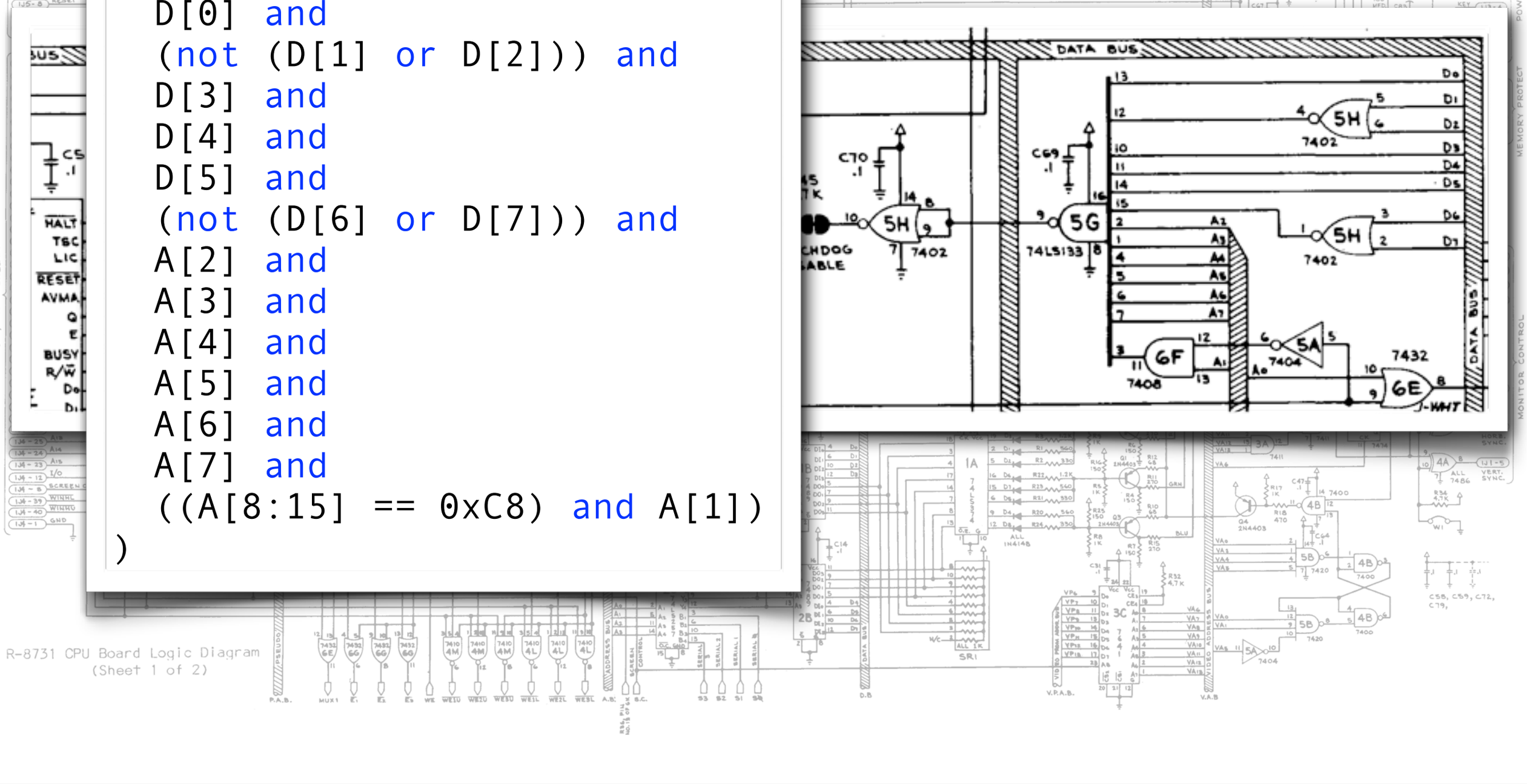


Saturday, June 22, 13

next

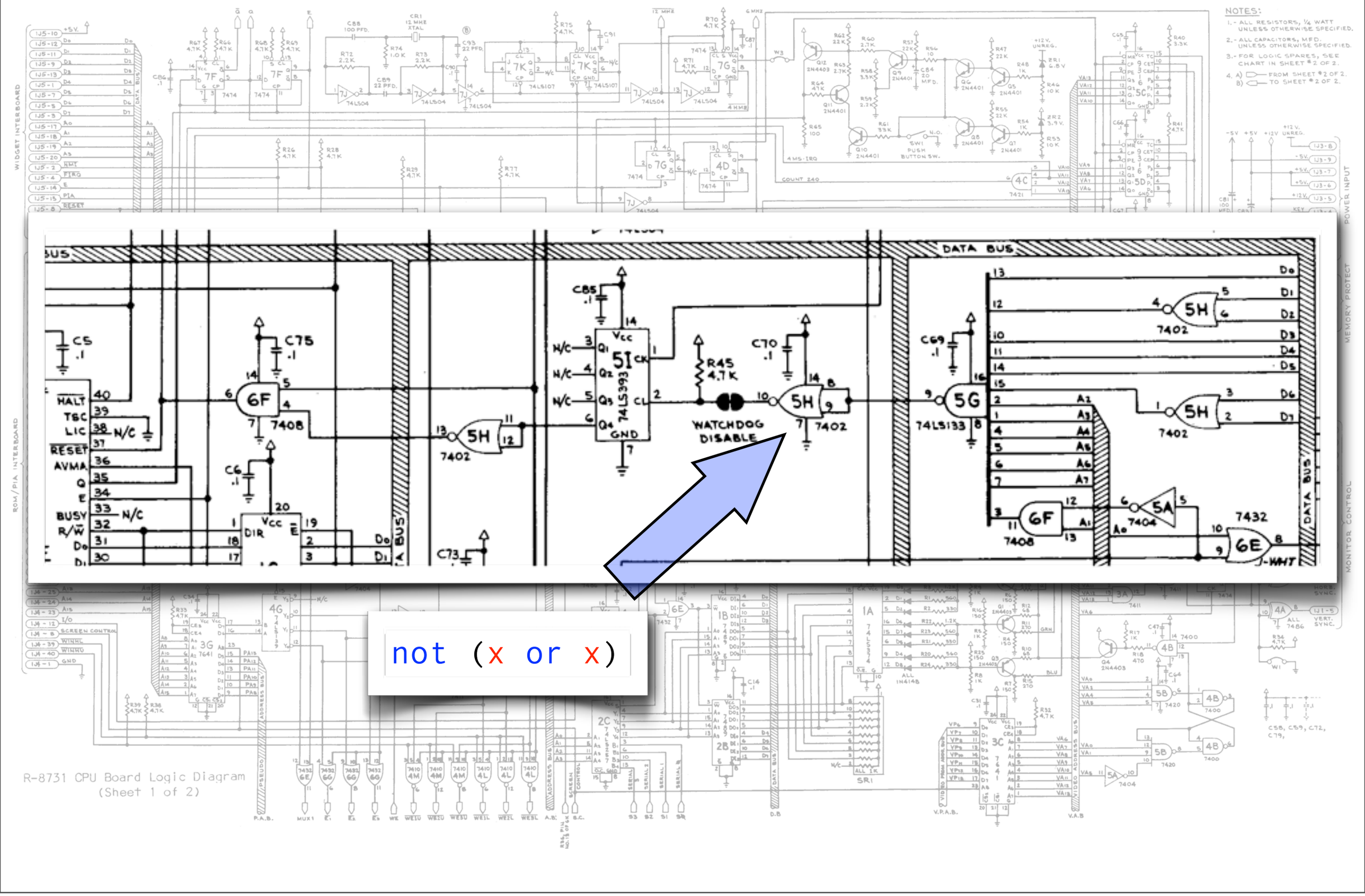
Watchdog

```
not (
  D[0] and
  (not (D[1] or D[2])) and
  D[3] and
  D[4] and
  D[5] and
  (not (D[6] or D[7])) and
  A[2] and
  A[3] and
  A[4] and
  A[5] and
  A[6] and
  A[7] and
  ((A[8:15] == 0xC8) and A[1])
)
```



...and lastly, a bunch of direct connections to various other data and address bits.

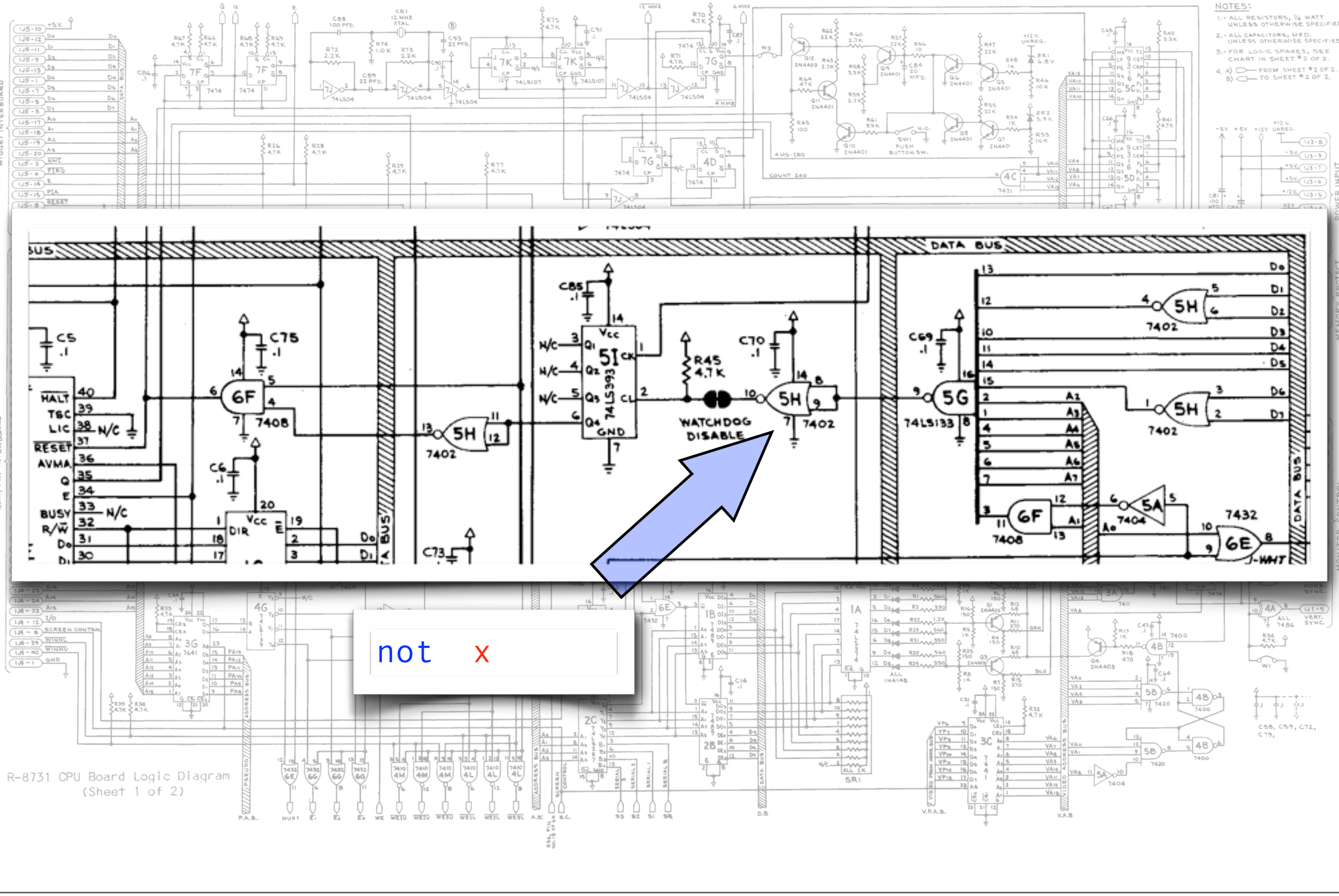
Watchdog



Saturday, June 22, 13

The output of that big ball of logic goes into a NOR, but with both inputs tied together -- having the same value. When the inputs are zero, the output is one. And when the inputs are one, the output is 0. That sounds like an inverter!

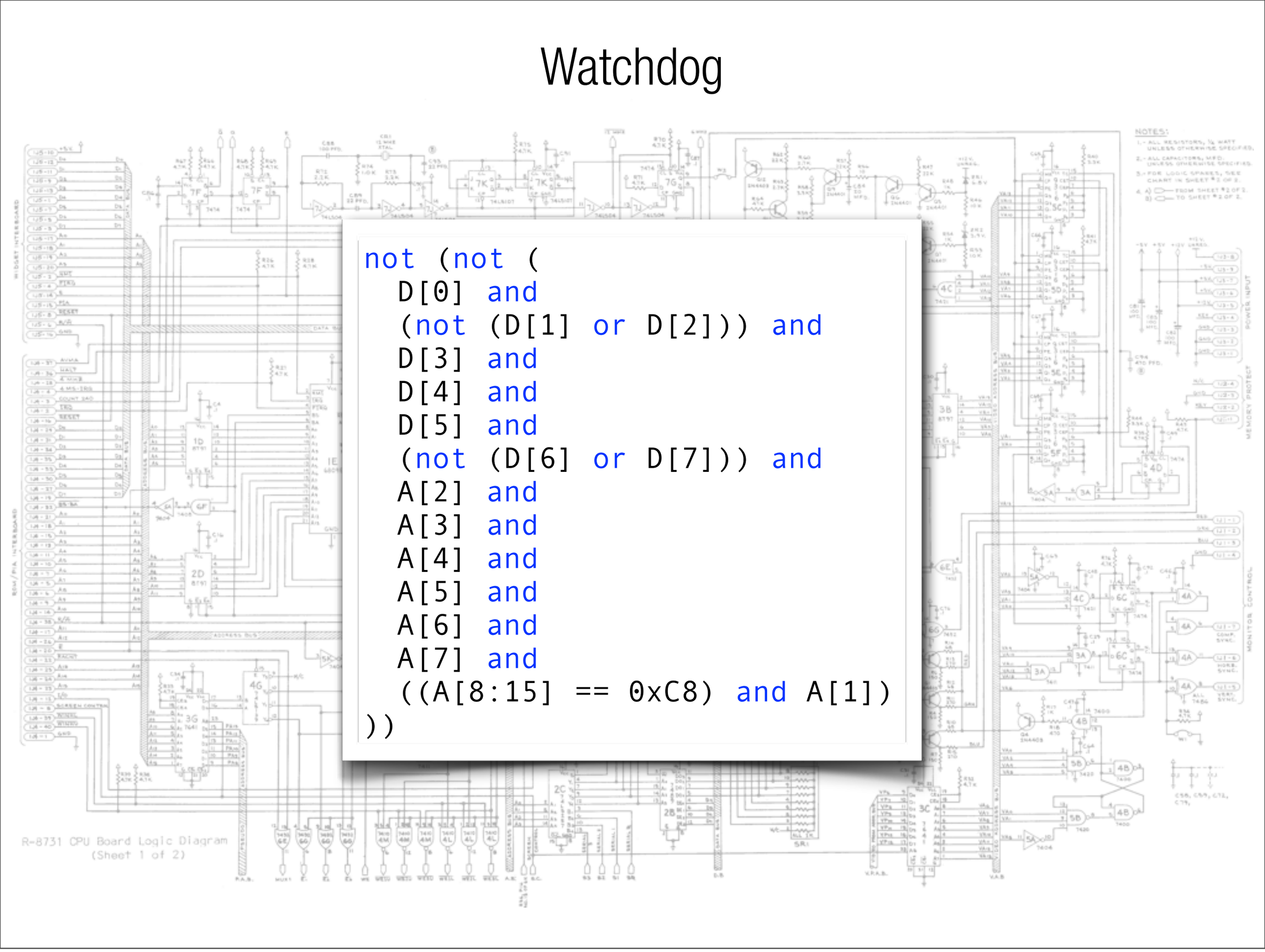
Watchdog



Saturday, June 22, 13

The hardware designer did this because logic functions usually come in multiples per chip. For a NOR function, you usually get four functions in a chip. The designer needed one more inverter, but only had an extra NOR. So they wired the extra NOR this way to make it act like an inverter. In doing so, the designer AVOIDED adding another inverter chip to the design.

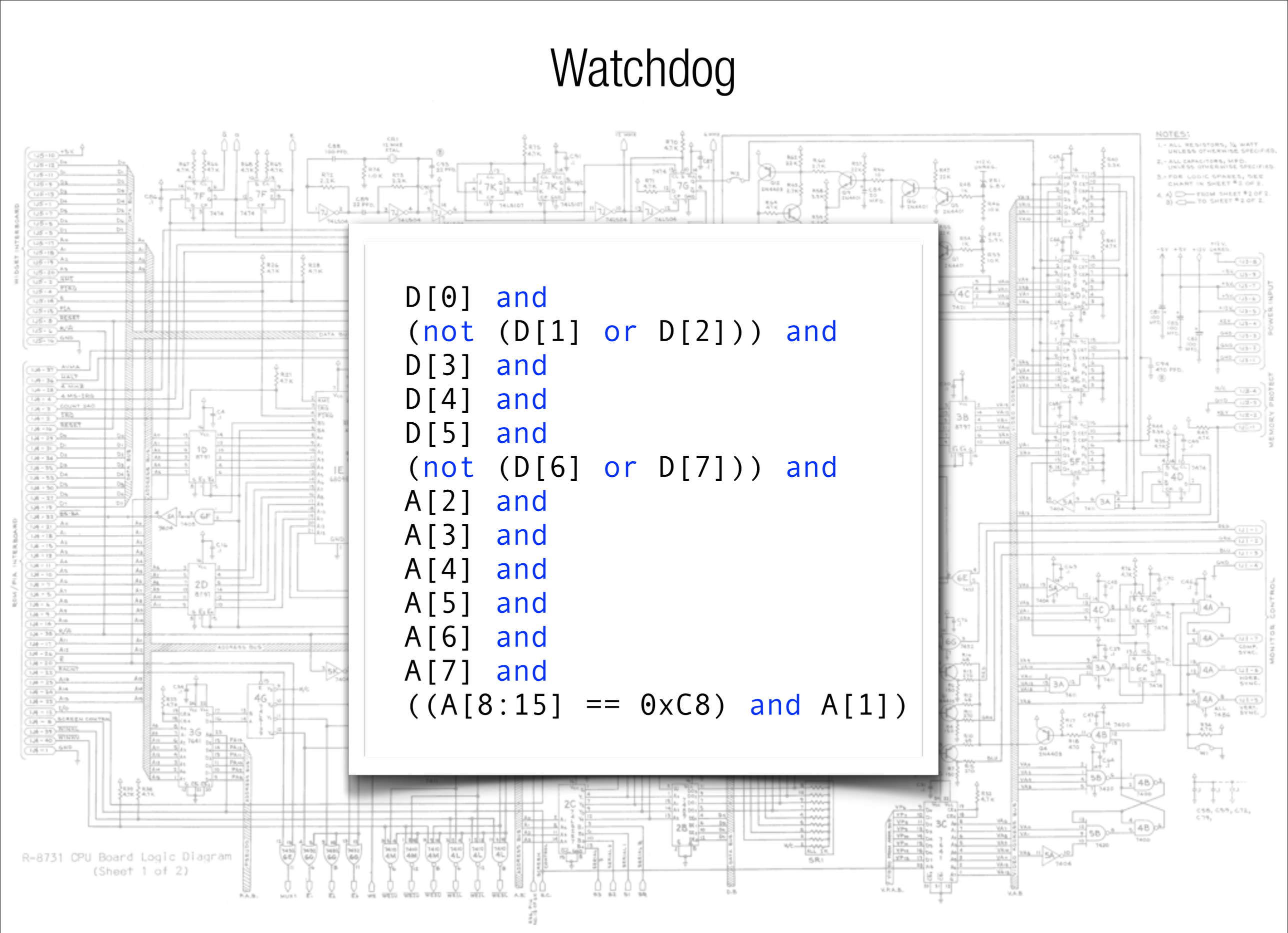
Watchdog



Saturday, June 22, 13

So with the additional NOT applied to the equation, we have this as our watchdog address and data decoder. Two NOTs cancel out, and we're left with this...

Watchdog



Saturday, June 22, 13

This is the equation which determines if the processor is trying to reset the watchdog timer. If this chunk of logic detects that the processor is trying to write a specific address with a specific data value, this equation will become true and will reset the watchdog timer, preventing the machine from being reset.

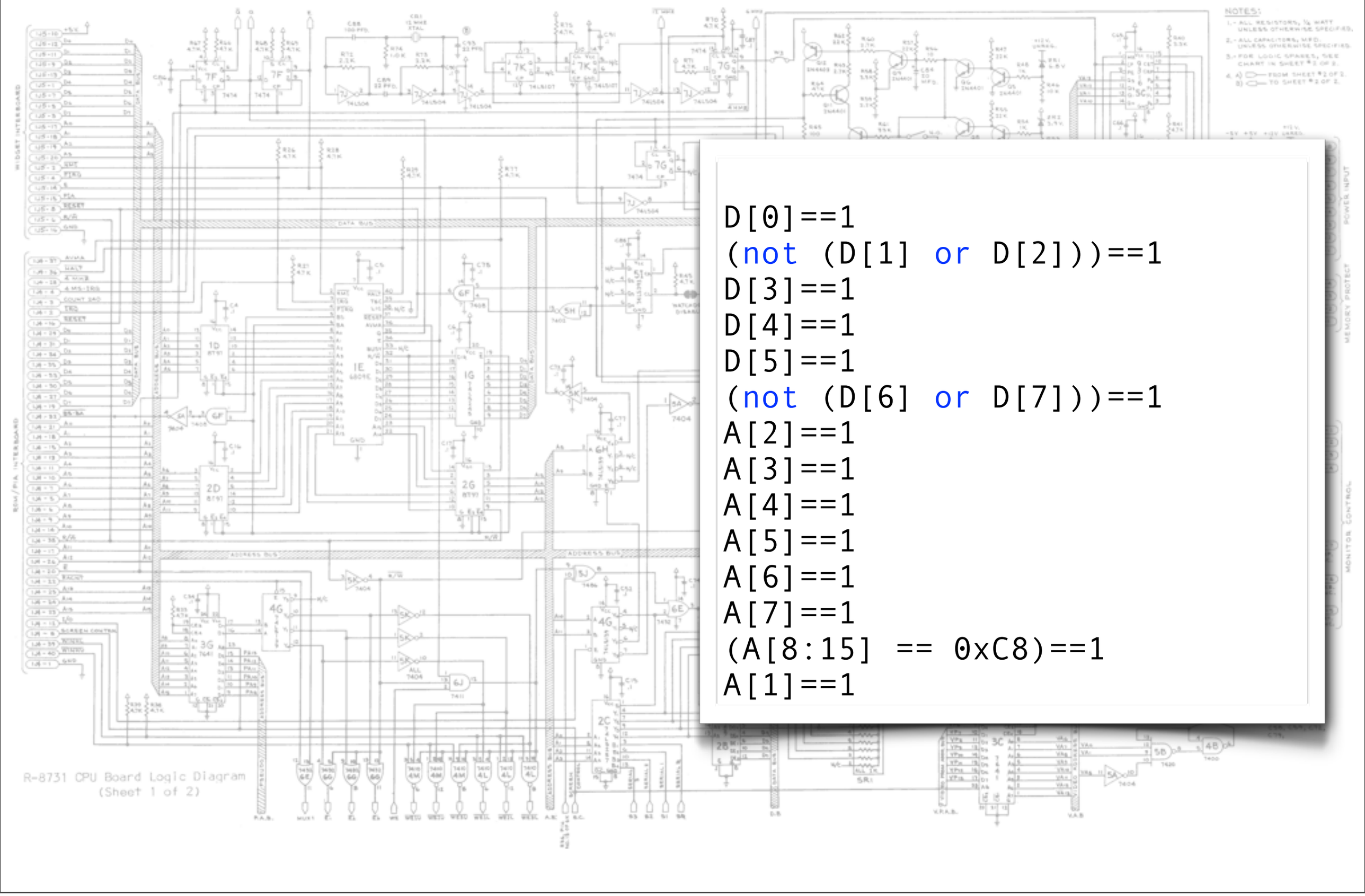
Watchdog

```
D[0] and
(not (D[1] or D[2])) and
D[3] and
D[4] and
D[5] and
(not (D[6] or D[7])) and
A[2] and
A[3] and
A[4] and
A[5] and
A[6] and
A[7] and
((A[8:15] == 0xC8) and A[1])
```

```
D[0]==1
(not (D[1] or D[2]))==1
D[3]==1
D[4]==1
D[5]==1
(not (D[6] or D[7]))==1
A[2]==1
A[3]==1
A[4]==1
A[5]==1
A[6]==1
A[7]==1
((A[8:15] == 0xC8) and
  A[1])==1
```


So let's substitute some values to figure out what will make this equation be 1 (true). At the outermost level, all the values are ANDed together. To make the equation equal 1 (true), D[0] needs to be 1 and D[3] needs to be 1, and D[4] needs to be 1, and so on...

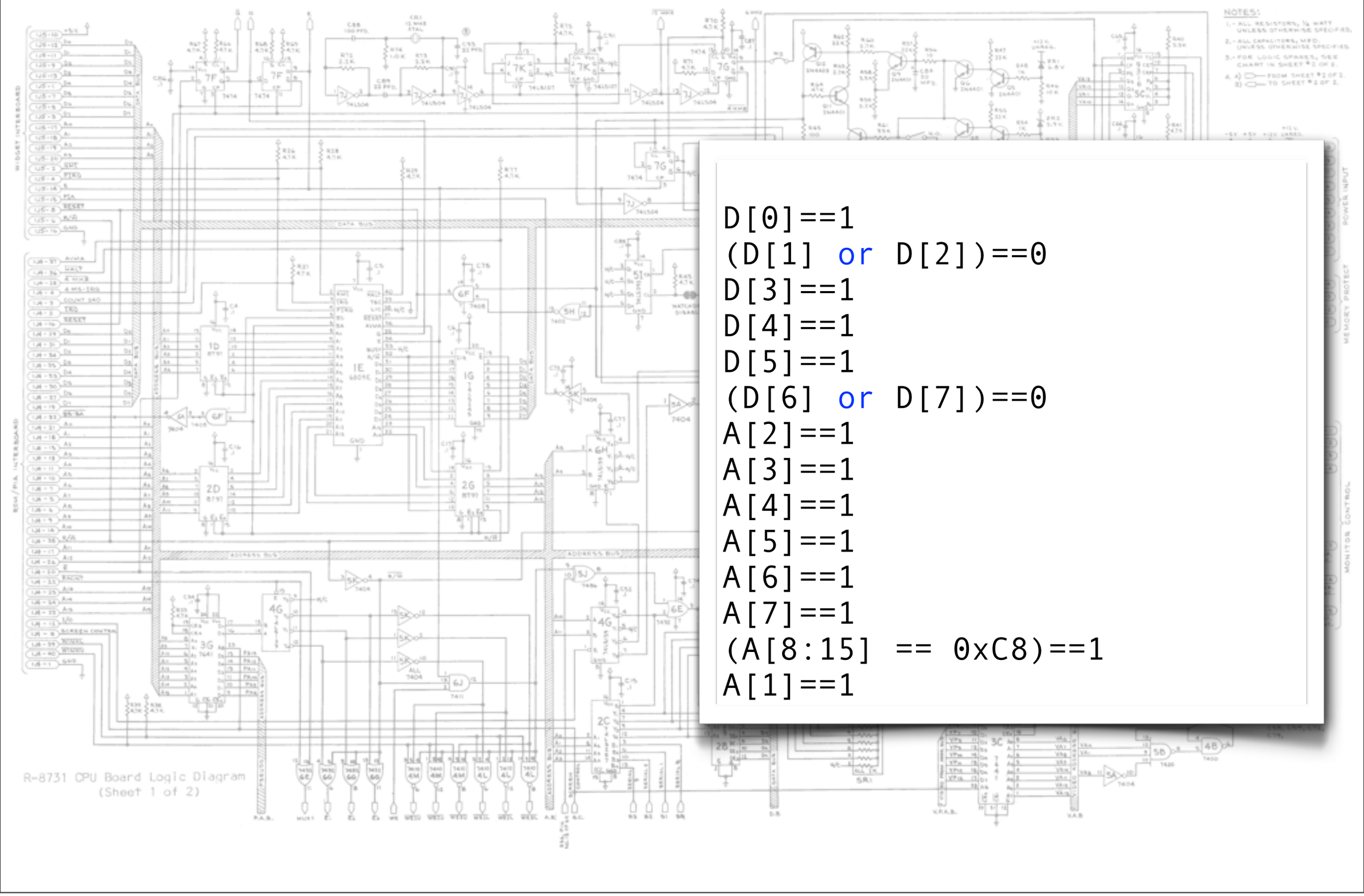
Watchdog



Saturday, June 22, 13

The test for A[8 through 15] equals C8 *and* A[1] equals 1 can be separated.

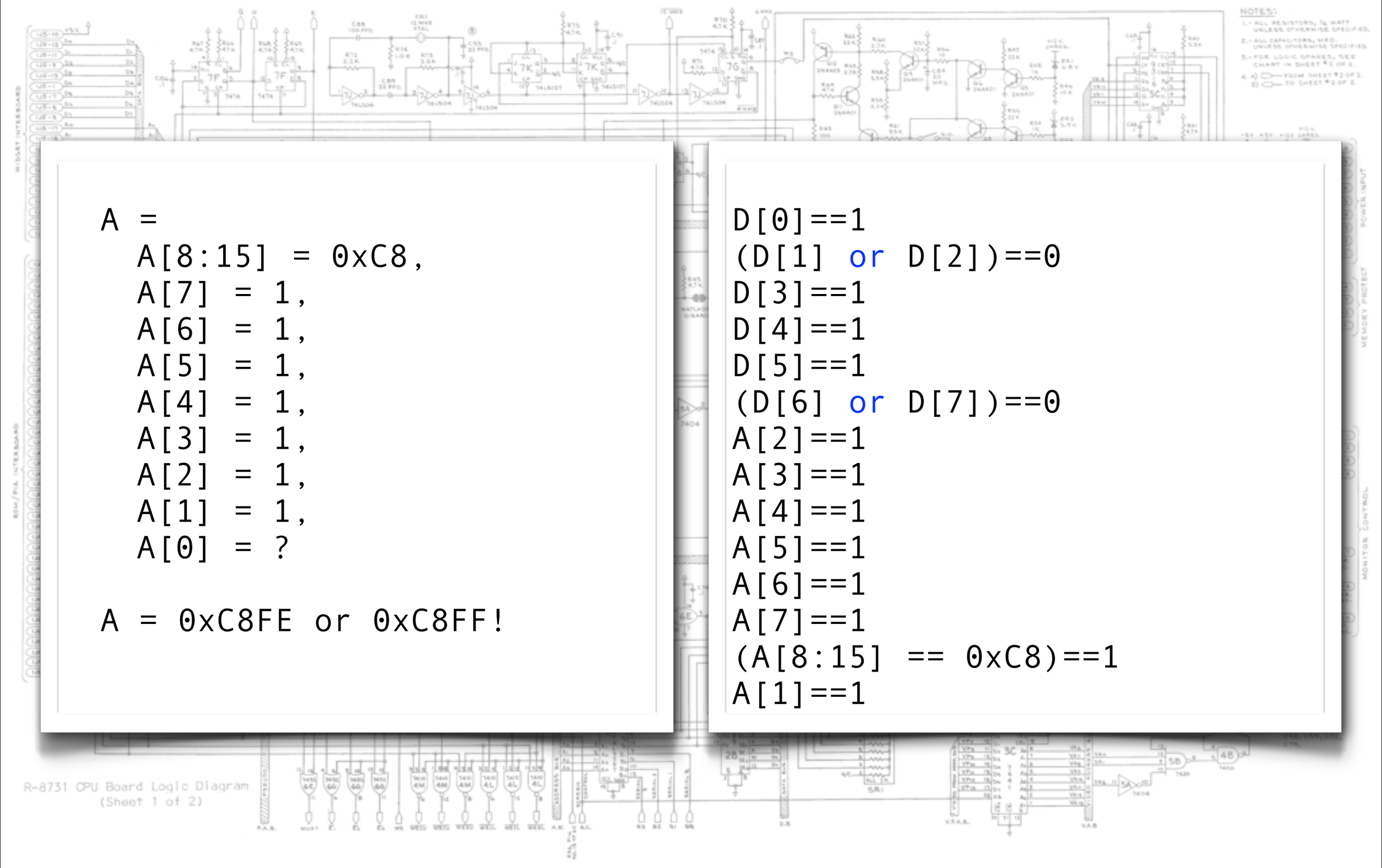
Watchdog



Saturday, June 22, 13

We can flip the truth on the two NOR equations... Removing the NOT and making the result equal 0 instead of 1.

Watchdog



```
A =
  A[8:15] = 0xC8,
  A[7] = 1,
  A[6] = 1,
  A[5] = 1,
  A[4] = 1,
  A[3] = 1,
  A[2] = 1,
  A[1] = 1,
  A[0] = ?

A = 0xC8FE or 0xC8FF!
```

```
D[0]==1
(D[1] or D[2])==0
D[3]==1
D[4]==1
D[5]==1
(D[6] or D[7])==0
A[2]==1
A[3]==1
A[4]==1
A[5]==1
A[6]==1
A[7]==1
(A[8:15] == 0xC8)==1
A[1]==1
```


From this, we can stick together the possible combinations and determine what the watchdog address is. Because A[0] isn't checked by the watchdog logic, it could be either a zero or a one, so the watchdog logic will respond to two different addresses, C8FE or C8FF.

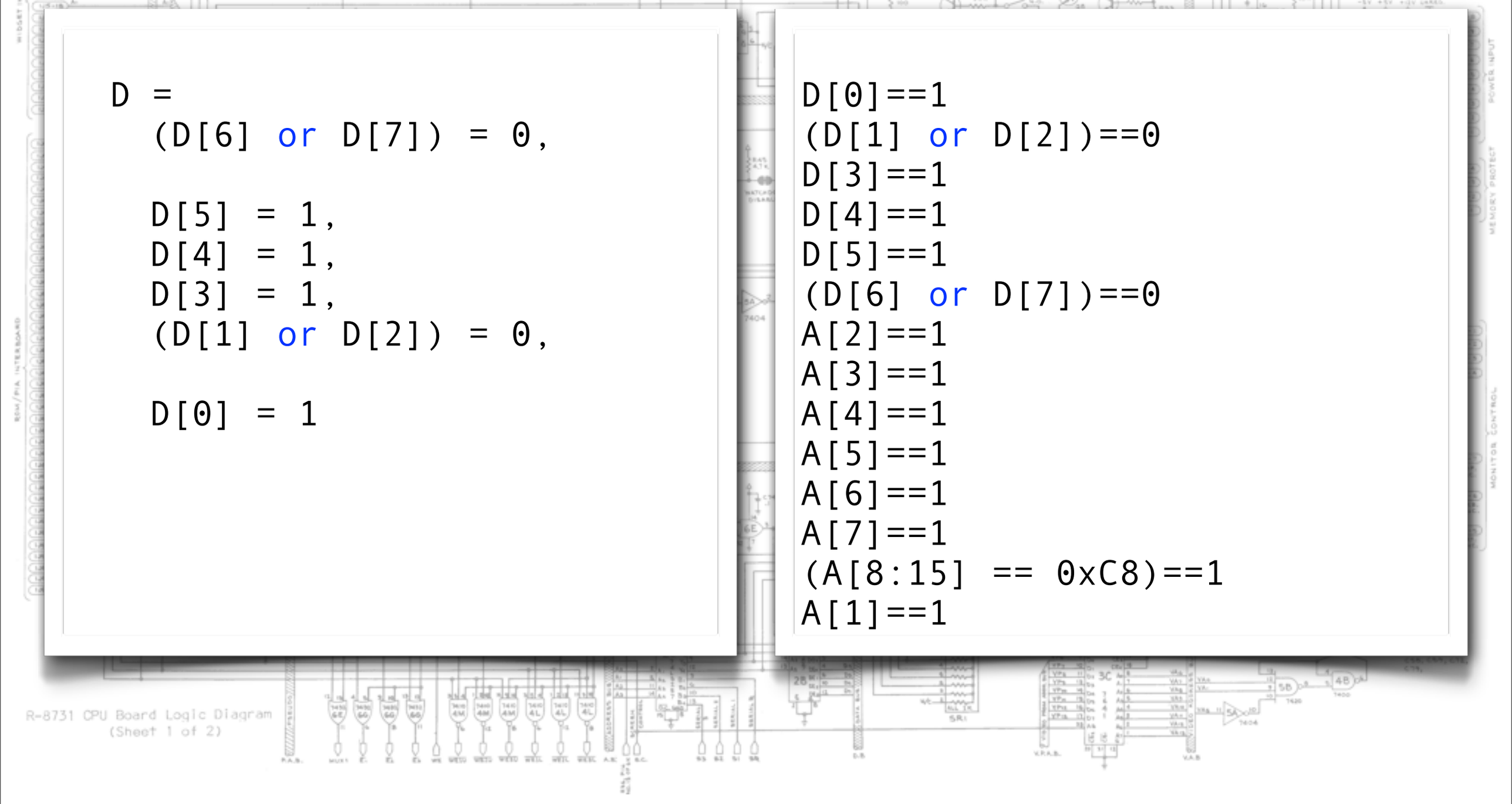
Watchdog

```
D =
  (D[6] or D[7]) = 0,

  D[5] = 1,
  D[4] = 1,
  D[3] = 1,
  (D[1] or D[2]) = 0,

  D[0] = 1
```

```
D[0]==1
(D[1] or D[2])==0
D[3]==1
D[4]==1
D[5]==1
(D[6] or D[7])==0
A[2]==1
A[3]==1
A[4]==1
A[5]==1
A[6]==1
A[7]==1
(A[8:15] == 0xC8)==1
A[1]==1
```



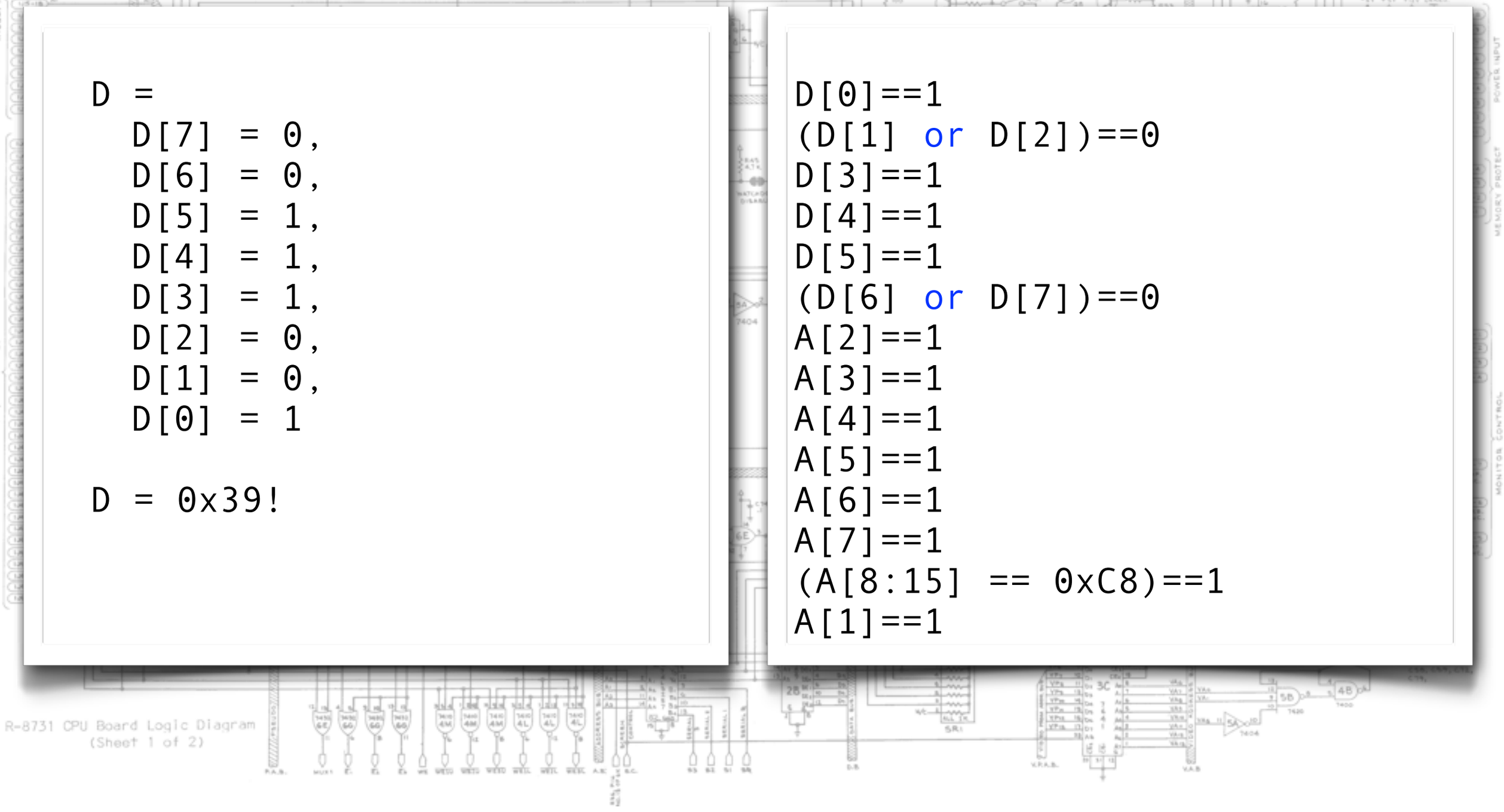
As for the data, we can do the same thing. For D[6] OR D[7] to be zero, both D[6] and D[7] must be zero. The same goes for D[1] and D[2].

Watchdog

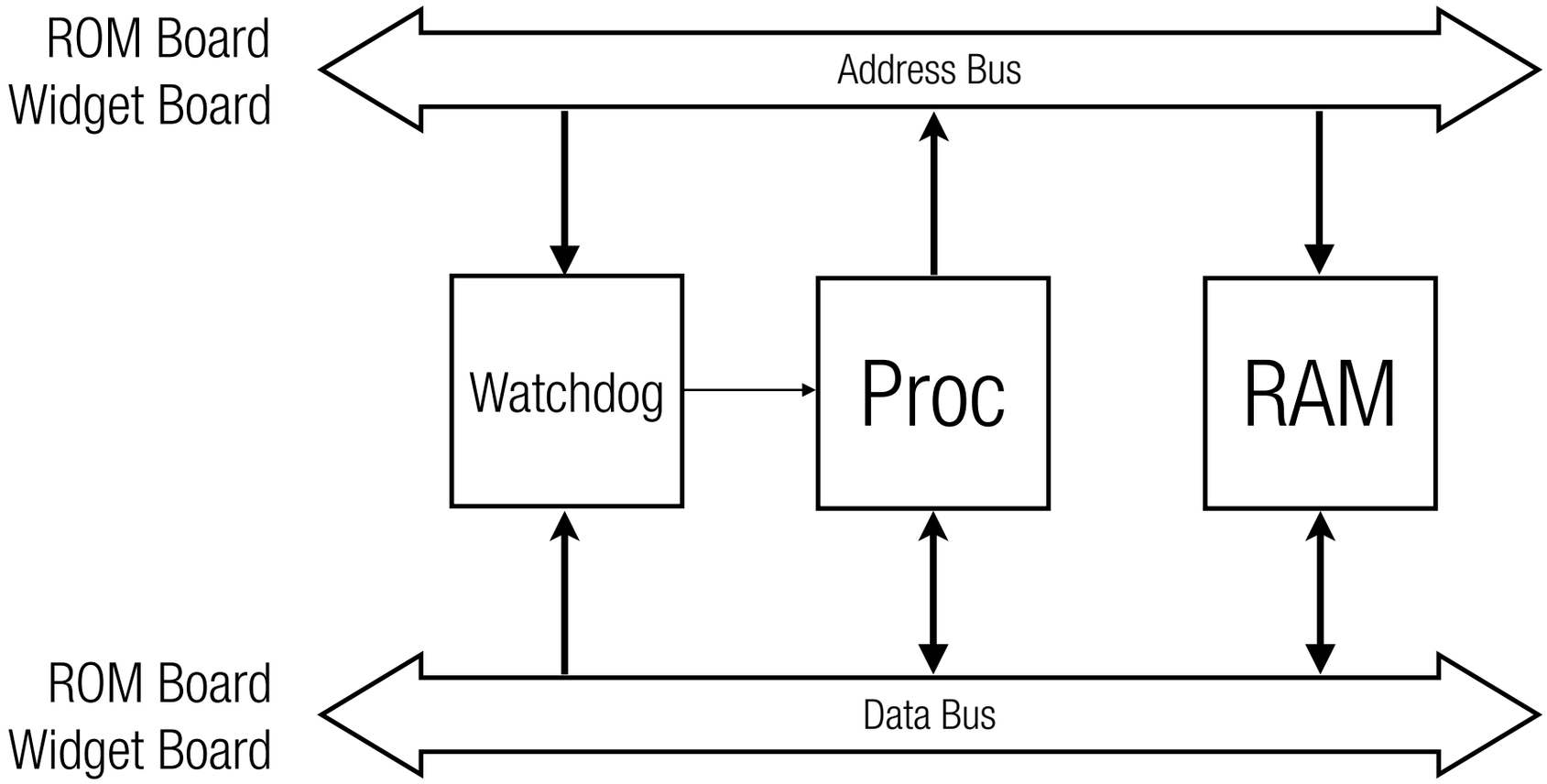
```
D =
  D[7] = 0,
  D[6] = 0,
  D[5] = 1,
  D[4] = 1,
  D[3] = 1,
  D[2] = 0,
  D[1] = 0,
  D[0] = 1

D = 0x39!
```

```
D[0]==1
(D[1] or D[2])==0
D[3]==1
D[4]==1
D[5]==1
(D[6] or D[7])==0
A[2]==1
A[3]==1
A[4]==1
A[5]==1
A[6]==1
A[7]==1
(A[8:15] == 0xC8)==1
A[1]==1
```

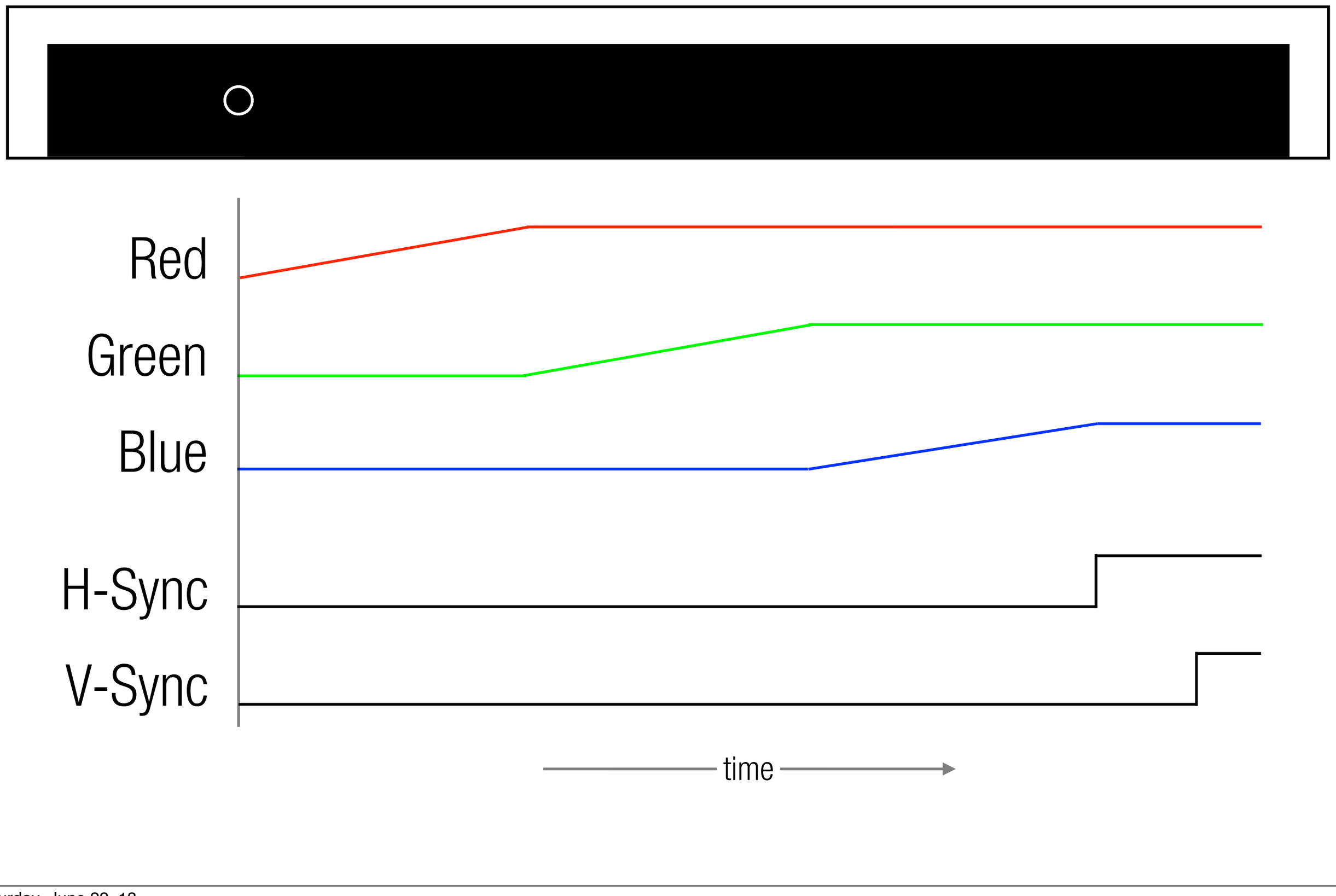


So the watchdog reset data value is 39 hex.



Here's the system architecture, updated with the watchdog timer. You can see that it listens to the address and data buses, and controls the RESET signal going to the processor.

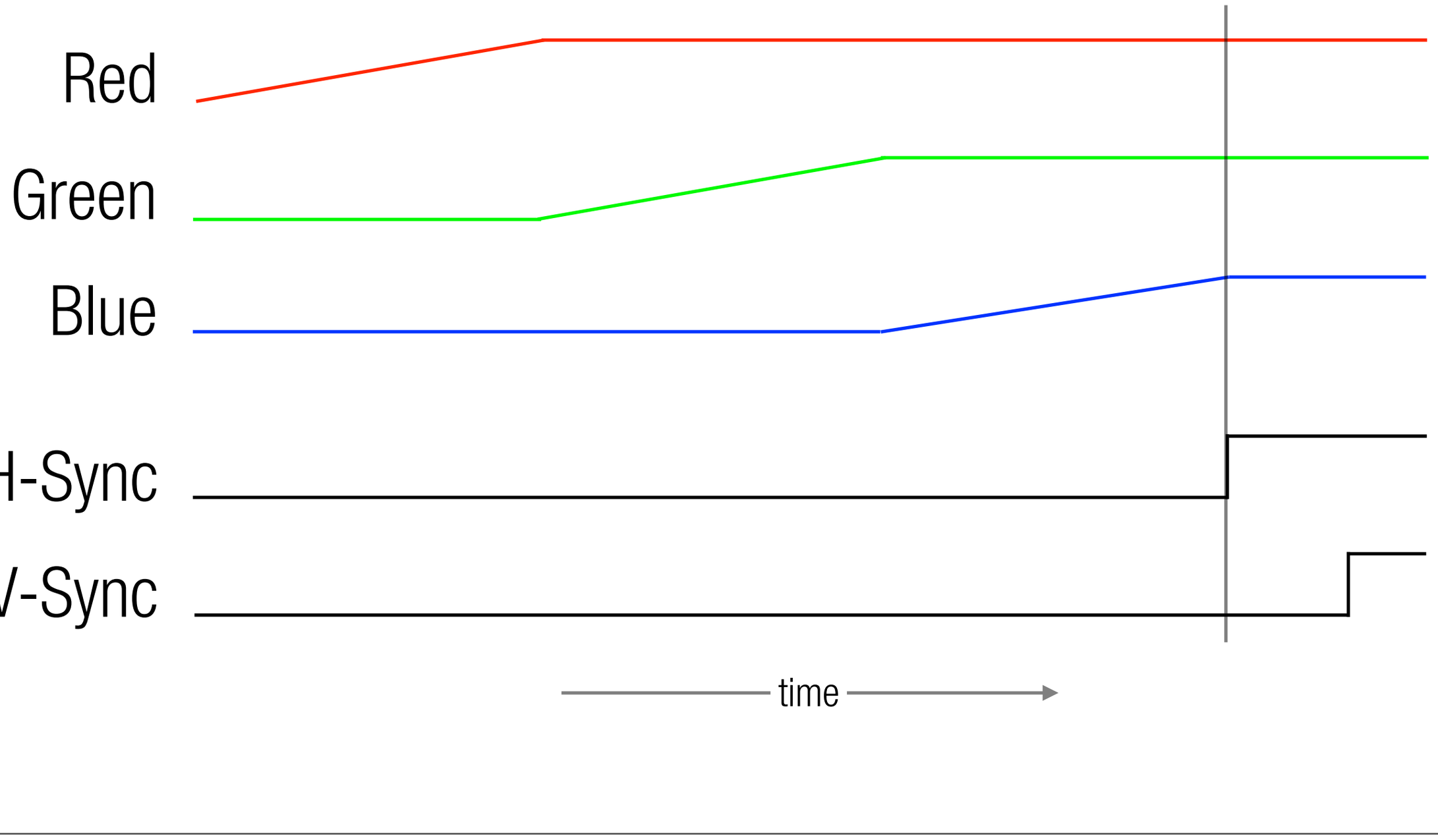
Video - Signals Overview



Saturday, June 22, 13

Next, the video system, and a quick refresher on how analog video signals work. ***next*** Pixels are scanned out to a video display one pixel at a time, over three color channels. The horizontal sync signal tells the video display when a row of pixels has ended, and to move back to the left side of the screen -- much like a carriage return and line feed in a text console. The vertical sync tells the video display when to start a new screen full of pixels. On most displays, this happens 60 times a second.

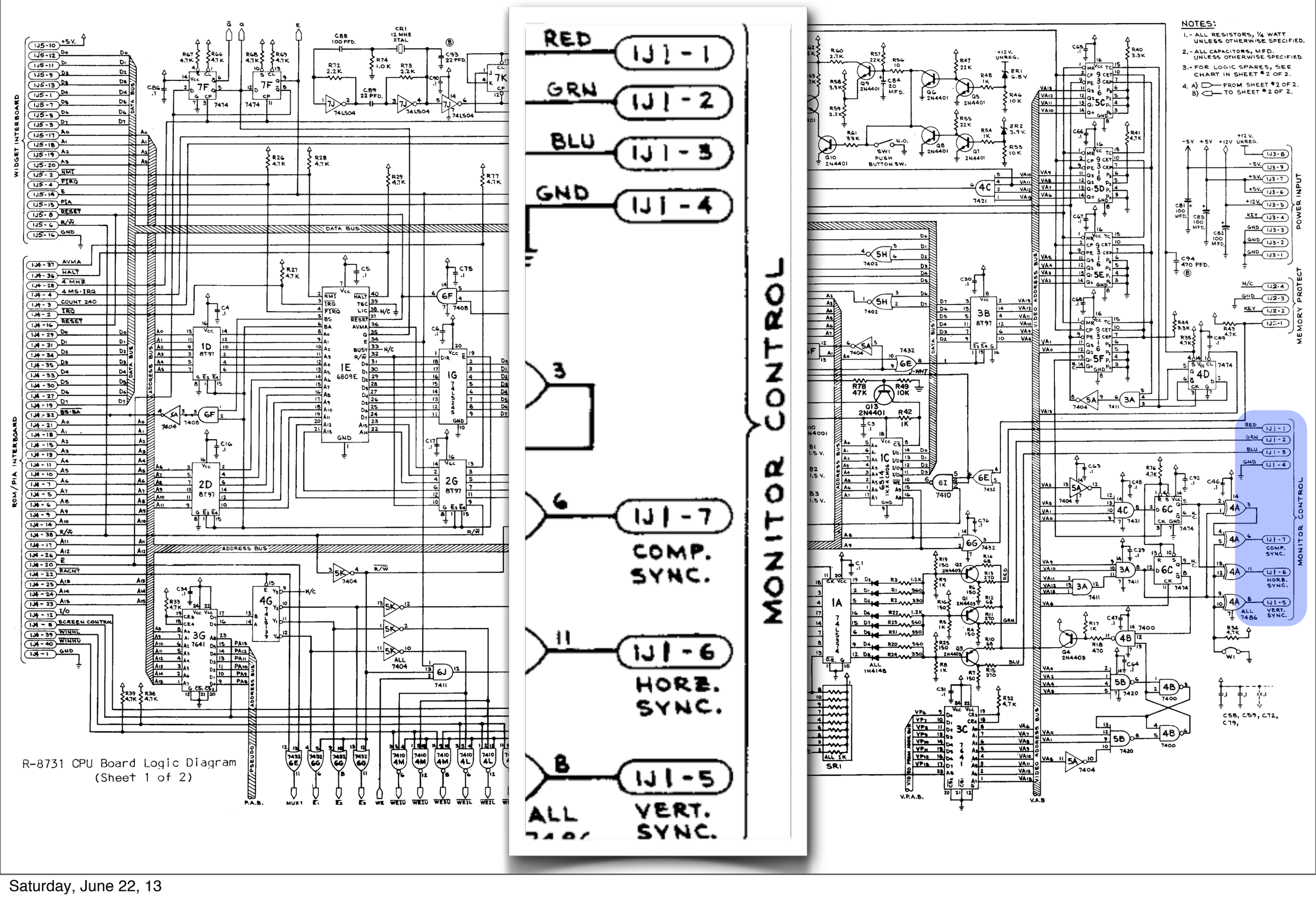
Video - Signals Overview



Saturday, June 22, 13

Next, the video system, and a quick refresher on how analog video signals work. ***next*** Pixels are scanned out to a video display one pixel at a time, over three color channels. The horizontal sync signal tells the video display when a row of pixels has ended, and to move back to the left side of the screen -- much like a carriage return and line feed in a text console. The vertical sync tells the video display when to start a new screen full of pixels. On most displays, this happens 60 times a second.

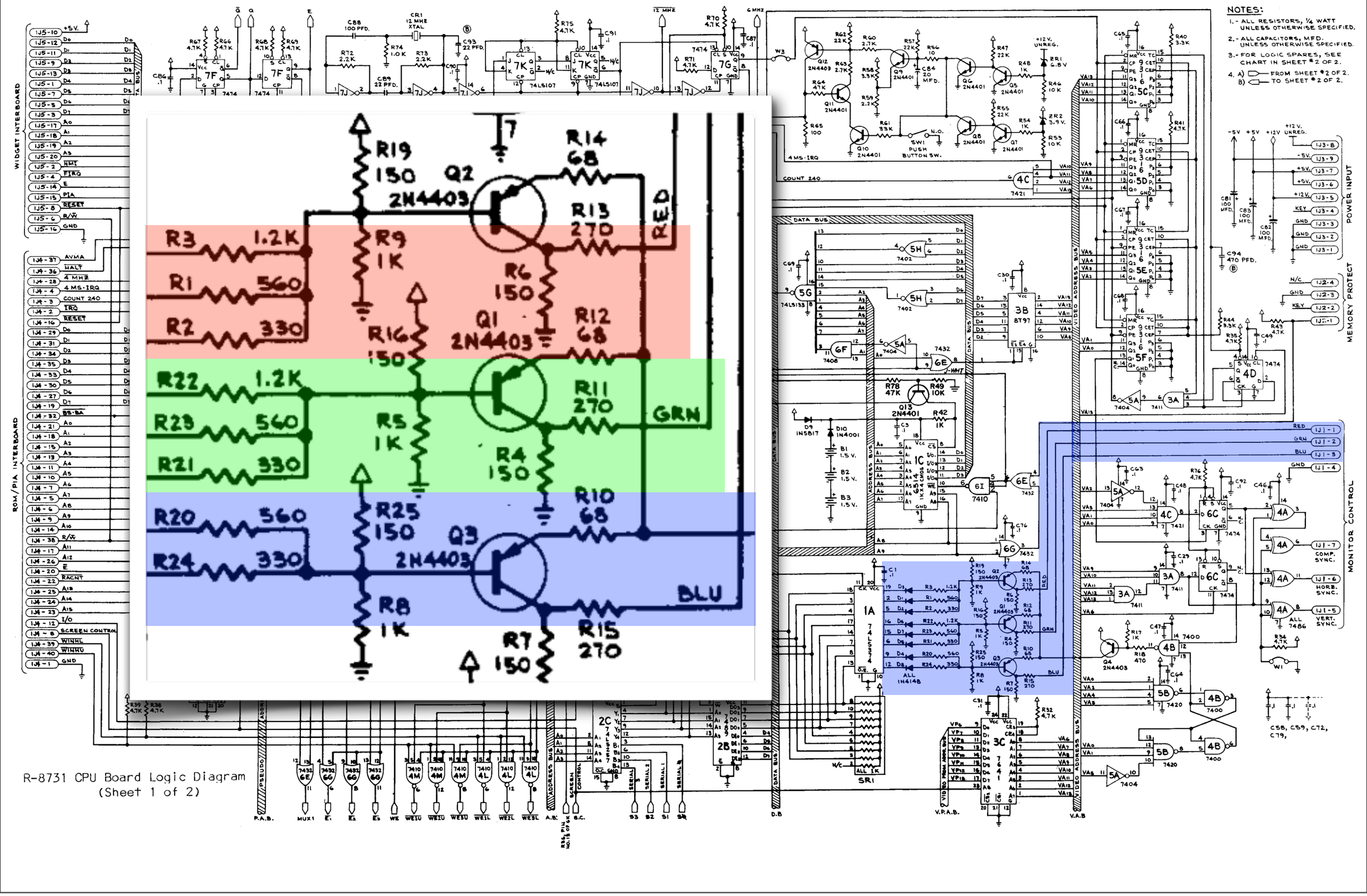
Video - Output



Saturday, June 22, 13

Here's the red, green, blue, horizontal sync, and vertical sync signals on the CPU board. This must be the video output. Let's trace back from these signals to see how the video hardware works.

Video - RGB Channels

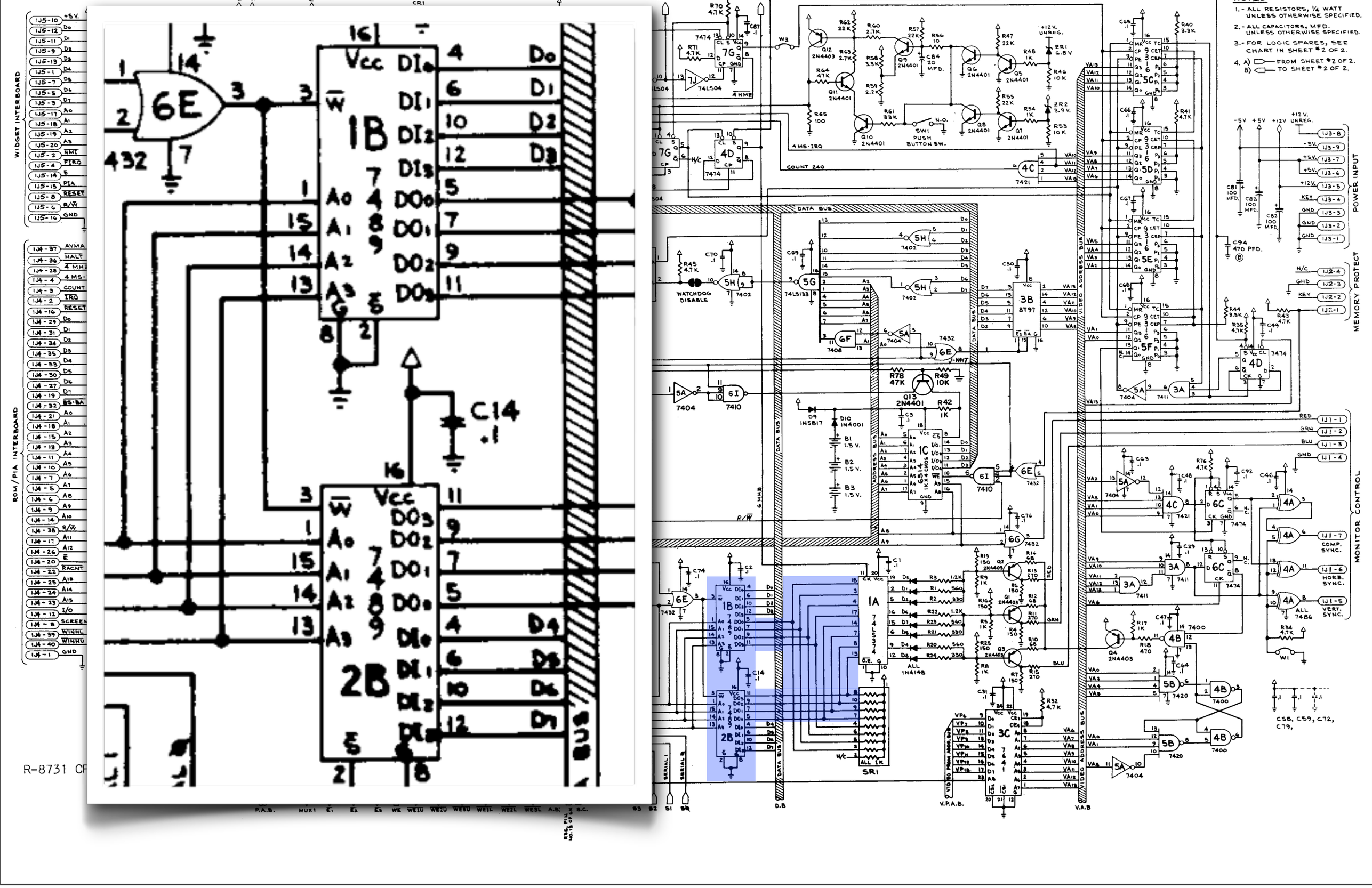


Saturday, June 22, 13

The red, green, and blue signals carry the pixel colors to the monitor as a video frame is scanned out for display, 60 times a second. The signals come through three transistors which buffer the output of three resistor ladders, one for each color channel.

Resistor ladders are a simple form of digital-to-analog converter. We can see that there are three bits (three resistors) allocated to red, three to green, but only two to blue. In total, we have eight bits of color depth. This is a big step back from the 24-bit color displays we enjoy on modern computers.

Video - Pixel Color Look-Up Table



Saturday, June 22, 13

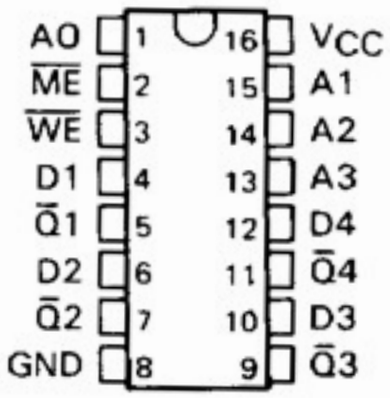
Continuing to work back from the pixel resistor ladders, we find two memory chips -- very small memory chips.

SN7489
64-BIT RANDOM-ACCESS READ/WRITE MEMORY

D1416, DECEMBER 1972—REVISED FEBRUARY 1984

- **For Application as a "Scratch Pad" Memory with Nondestructive Read-Out**
- **Fully Decoded Memory Organized as 16 Words of Four Bits Each**
- **Fast Access Time . . . 33 ns Typical**
- **Diode-Clamped, Buffered Inputs**
- **Open-Collector Outputs Provide Wire-AND Capability**
- **Typical Power Dissipation . . . 375 mW**
- **Compatible with Most TTL Circuits**

SN7489 . . . J OR N PACKAGE
(TOP VIEW)

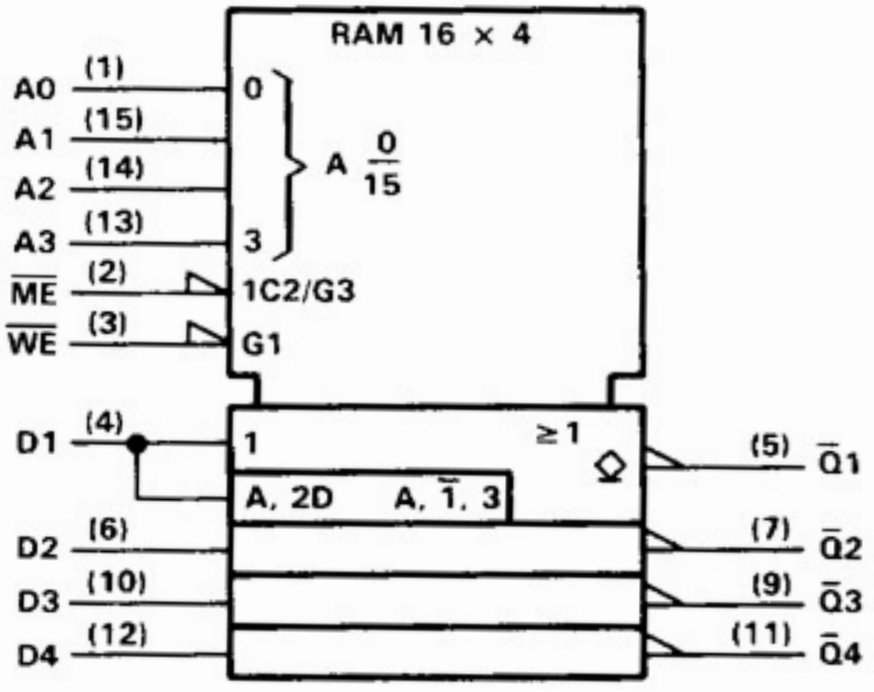


description

This 64-bit active-element memory is a monolithic, high-speed, transistor-transistor logic (TTL) array of 64 flip-flop memory cells organized in a matrix to provide 16 words of four bits each. Each of the 16 words is addressed in straight binary with full on-chip decoding.

The buffered memory inputs consist of four address lines, four data inputs, a write enable, and a memory enable for controlling the entry and access of data. The memory has open-collector outputs which may be wired-AND connected to permit expansion up to 4704 words of N-bit length without additional output buffering. Access time is typically 33 nanoseconds; power dissipation is typically 375 milliwatts.

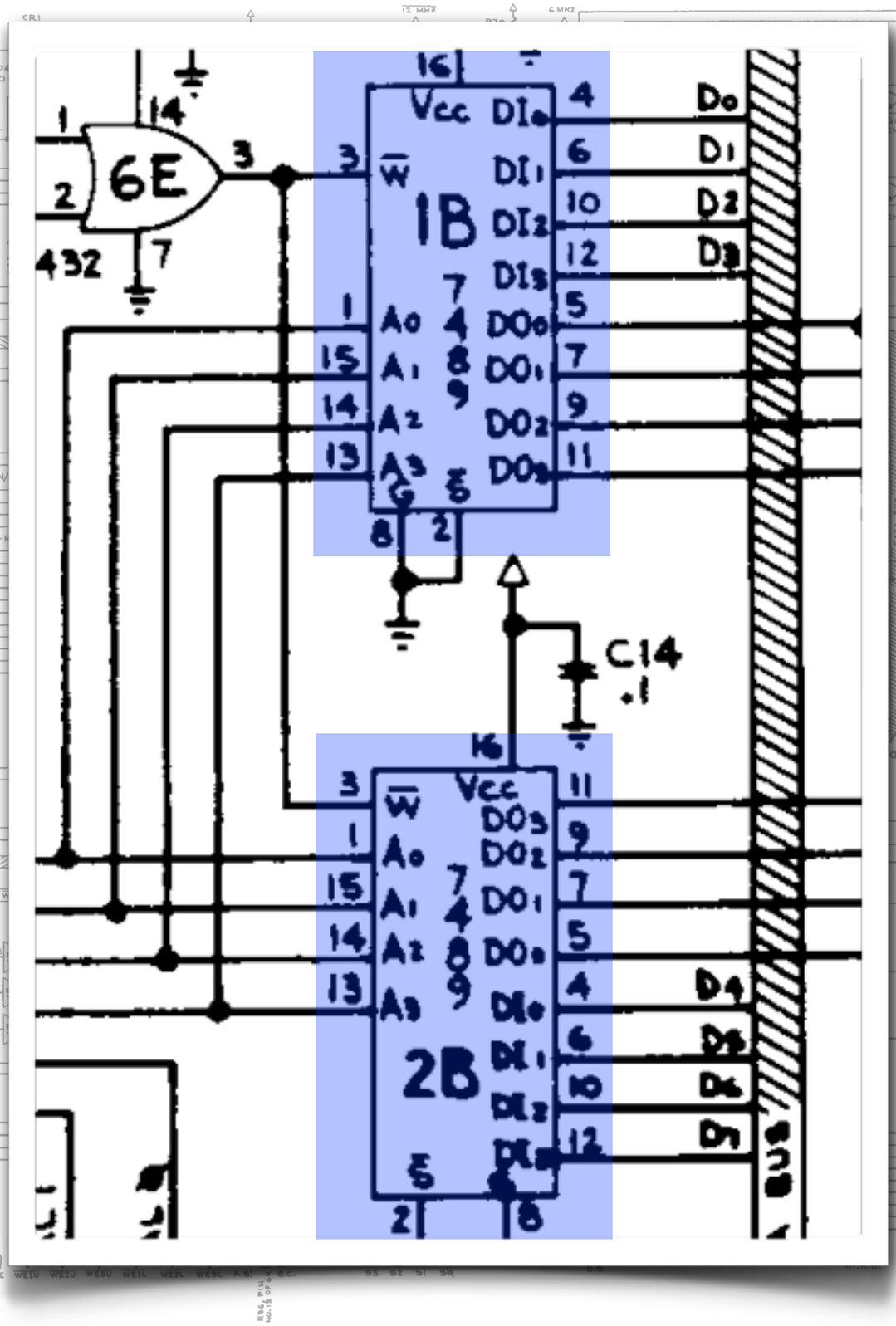
logic symbol



Saturday, June 22, 13

...a measly 64 bits, in fact, organized as an array of 16 x 4 bits.

Video - Pixel Color Look-Up Table

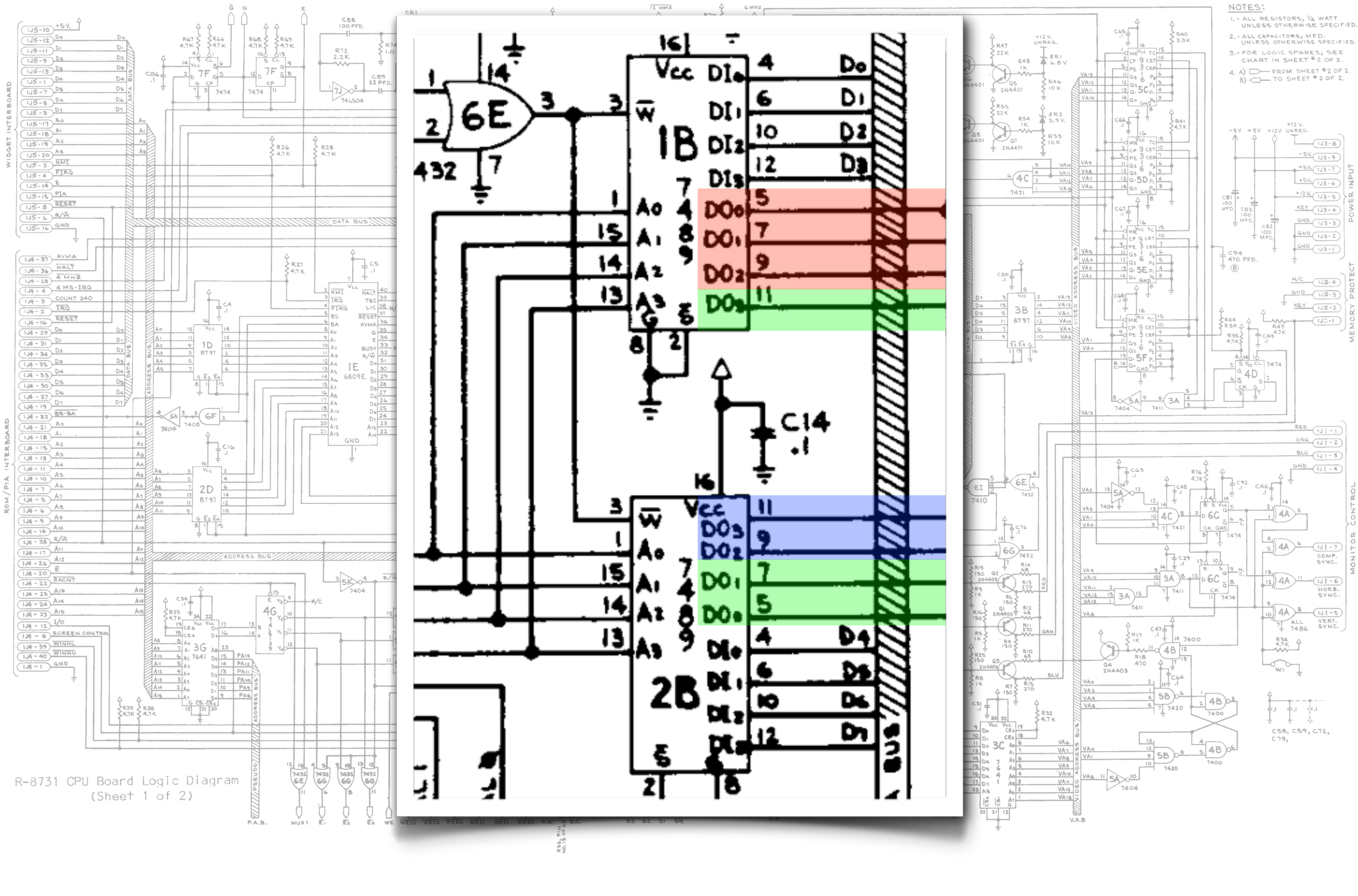


R-8731 CPU Board Logic Diagram
(Sheet 1 of 2)

Saturday, June 22, 13

Two of these RAM chips are wired up into a 16 x 4 x 2 arrangement, for an effective array size of 16 x 8 bits.

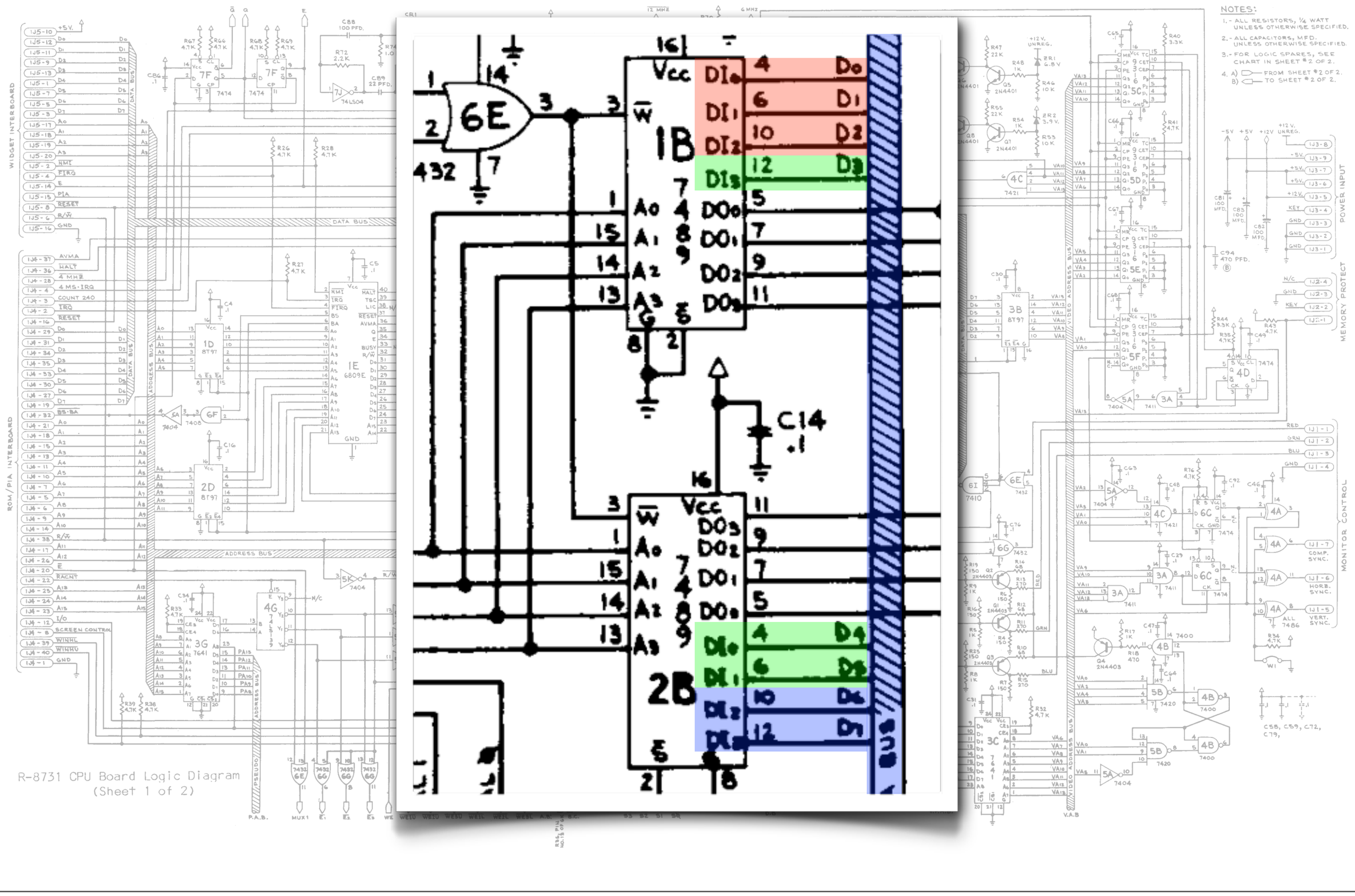
Video - Pixel Color Look-Up Table



Saturday, June 22, 13

The eight data outputs of these memory chips are wired to the RGB color channels.

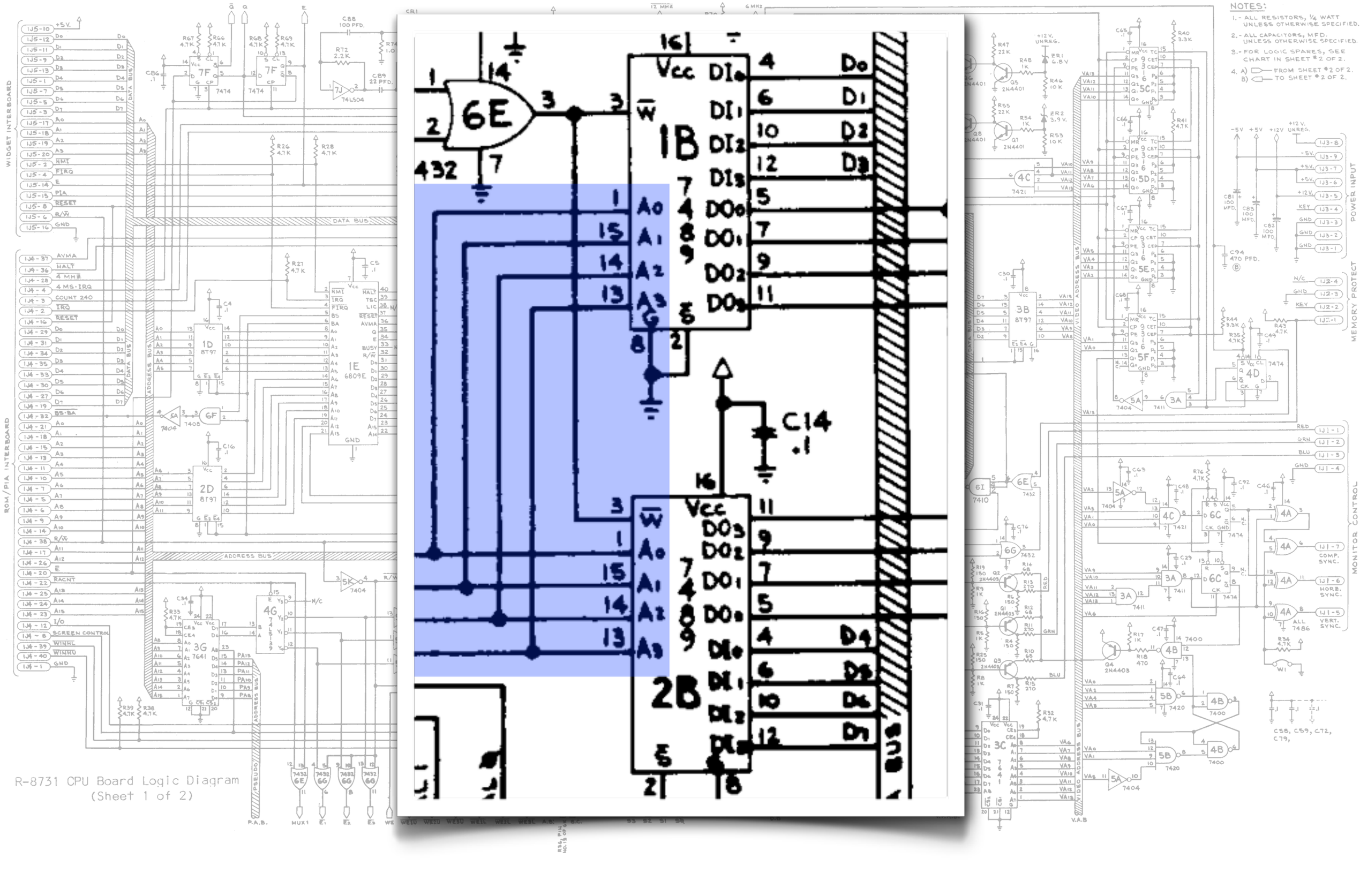
Video - Pixel Color Look-Up Table



Saturday, June 22, 13

And the eight data inputs are connected to the processor's data bus.

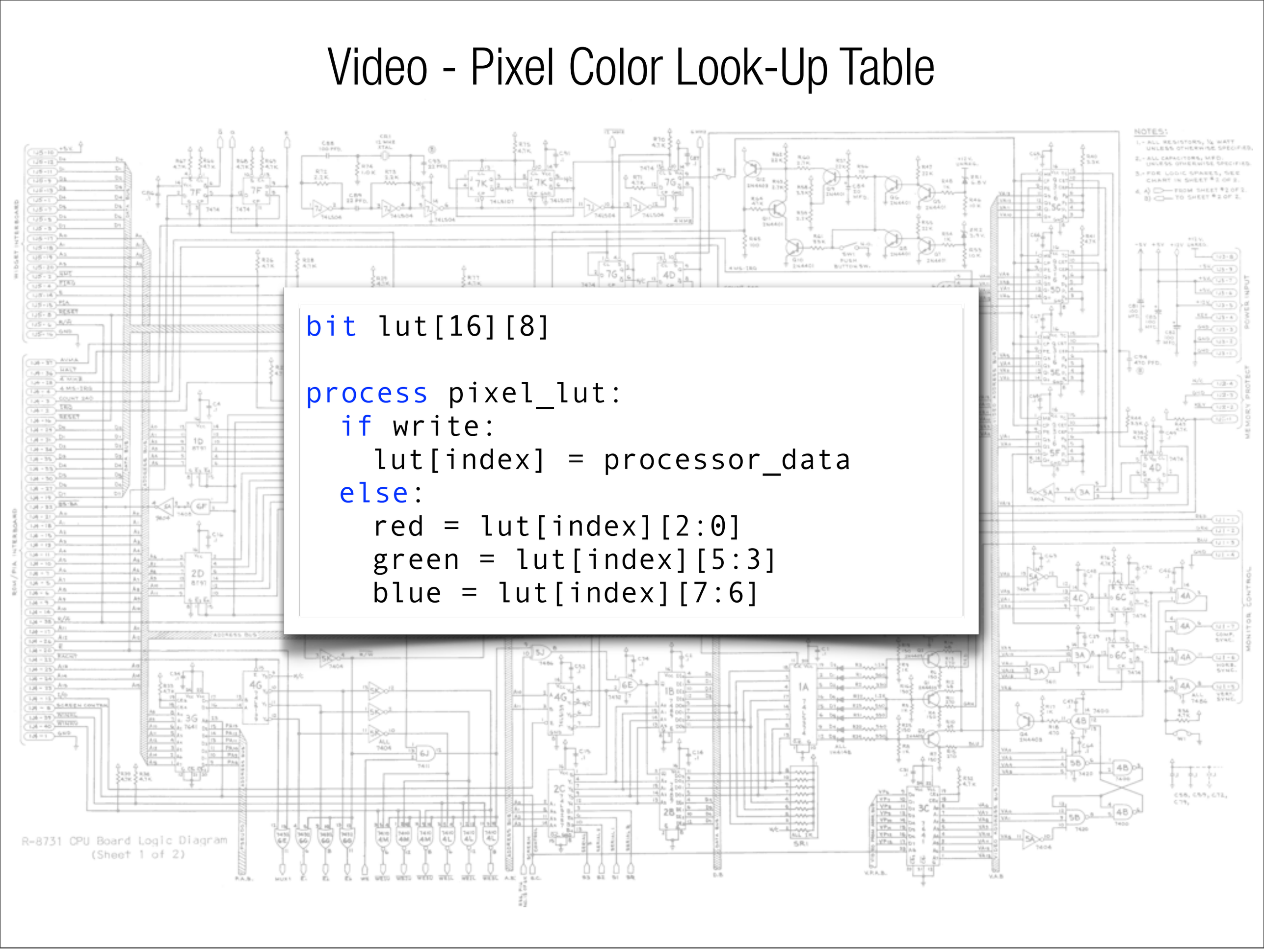
Video - Pixel Color Look-Up Table



Saturday, June 22, 13

The address input (the index into the memory array) comes from the memory bank on the other page of the CPU board schematic.

Video - Pixel Color Look-Up Table



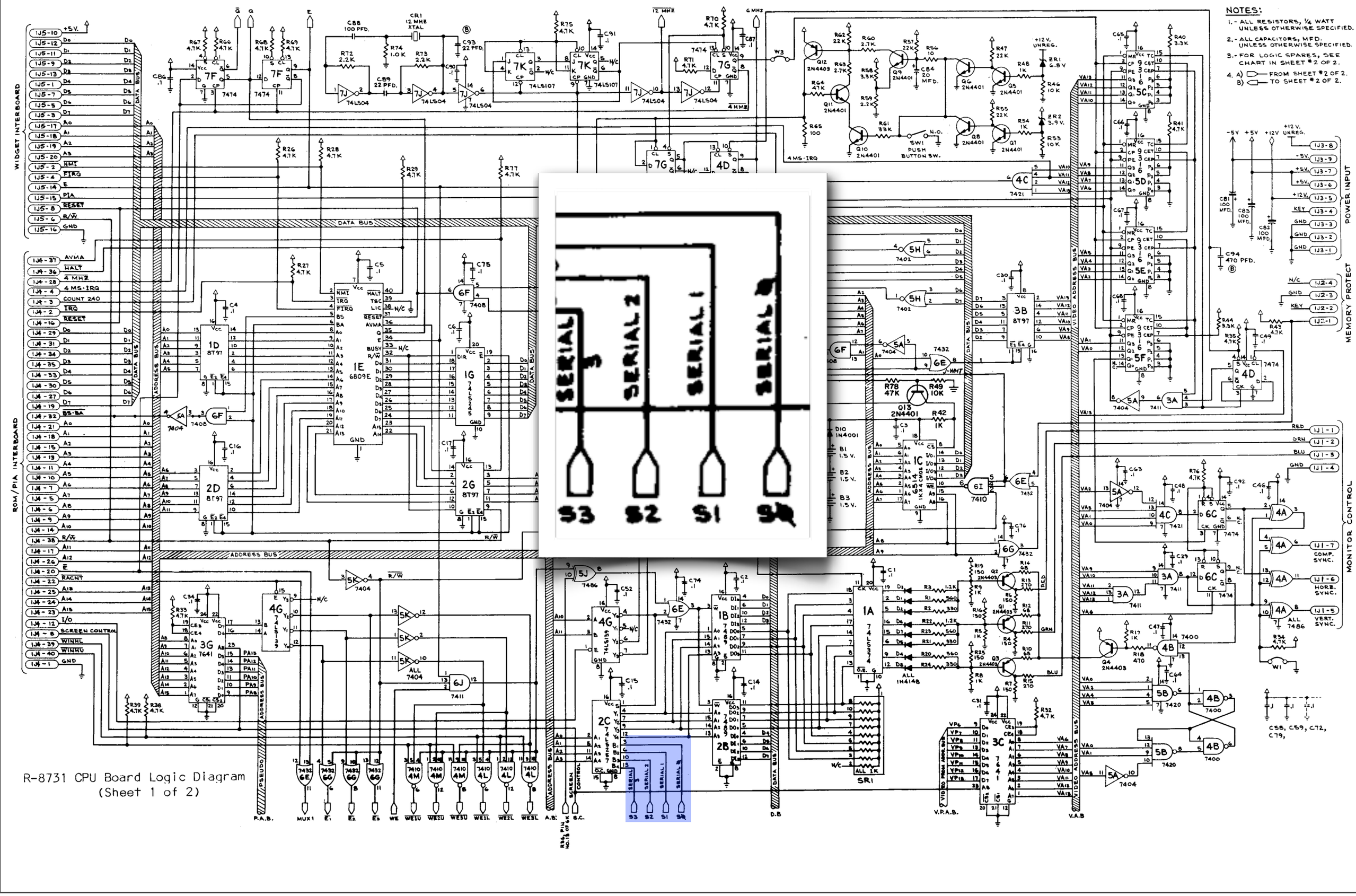
Saturday, June 22, 13

At this point, it's safe to say that the display image lives in a buffer in the 48K memory bank, and that these two little memory chips are providing a look-up table, turning four-bit pixel values into eight-bit RGB values. In other words, this is a color table or color palette memory. The processor can modify the color palette to change the RGB color that a pixel index corresponds to. This is what gives Robotron its crazy palette animation effects.

Modern computers, without palette lookup tables, would have to redraw virtually the entire screen to achieve these epilepsy-inducing color effects. But with the Robotron's palette-based graphics system, the processor just rewrites only 16 palette RAM values for each video frame, and animation is achieved!

Modern computers, without palette lookup tables, would have to redraw virtually the entire screen to achieve these epilepsy-inducing color effects. But with the Robotron's palette-based graphics system, the processor just rewrites only 16 palette RAM values for each video frame, and animation is achieved!

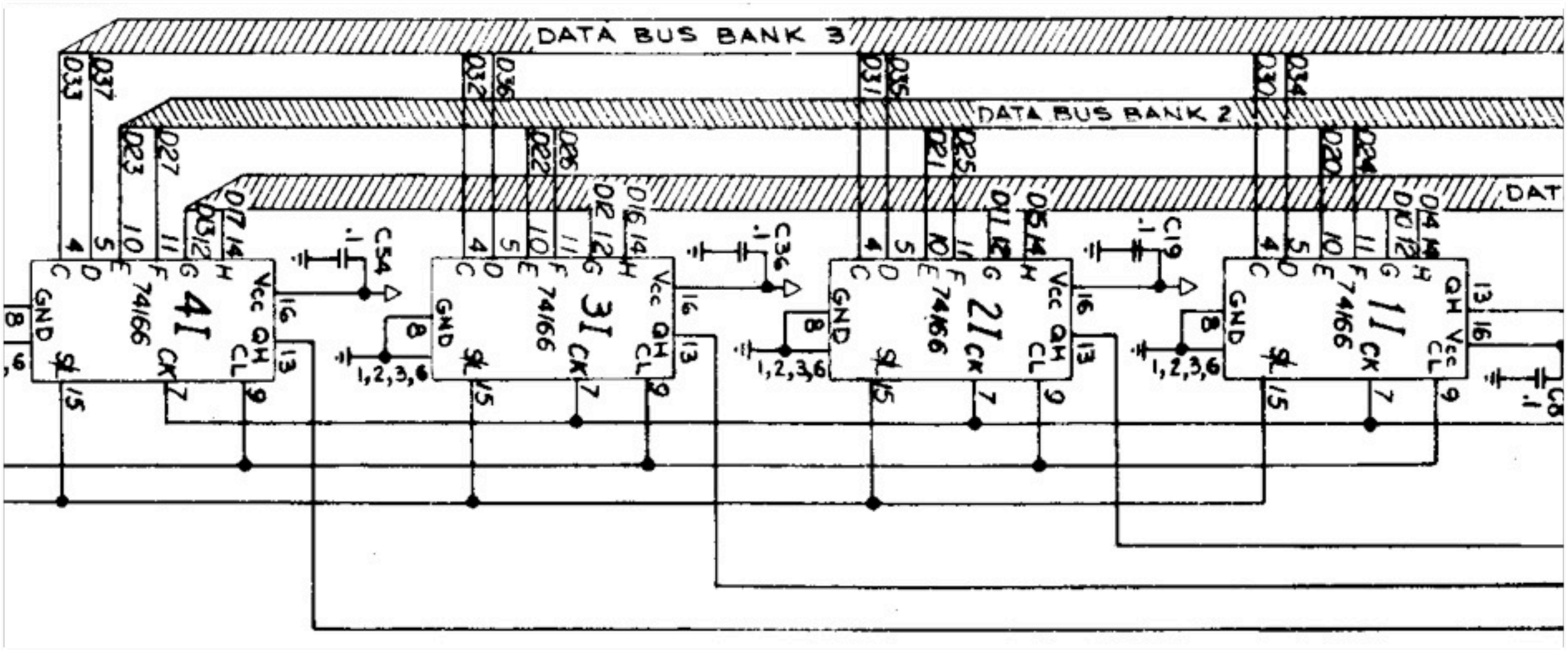
Video - Pixel Color Look-Up Table

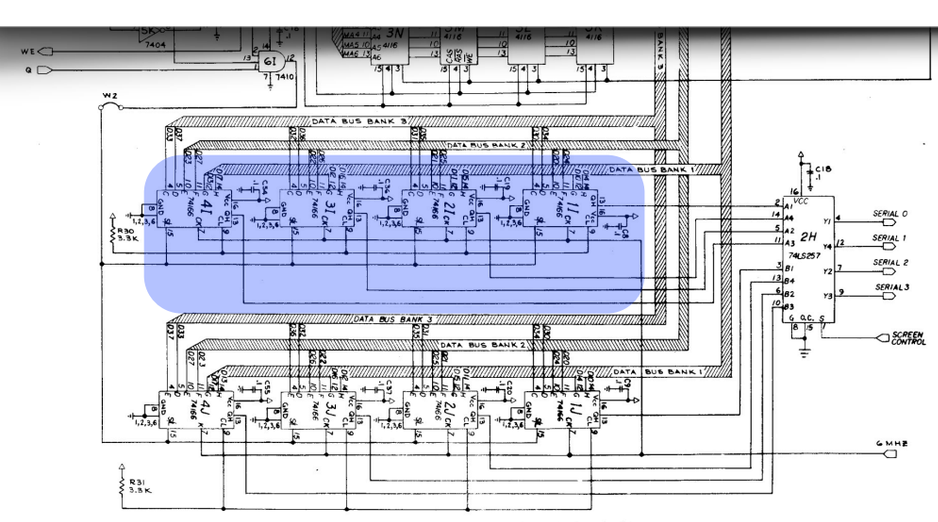


Saturday, June 22, 13

How does pixel data get from the 48K memory bank to the color palette memory? It comes in from the other schematic sheet, four bits at a time, as signals “serial0” through “serial3”.

Video - Memory:Pixel Serializer





R-8731 CPU Board Logic Diagram (Sheet 2 of 2)

Saturday, June 22, 13

Those four bits come from this circuit, which is made up of four 74166 chips. Each chip is an eight-bit shift register. These shift registers take bytes and turn them into a stream of bits.

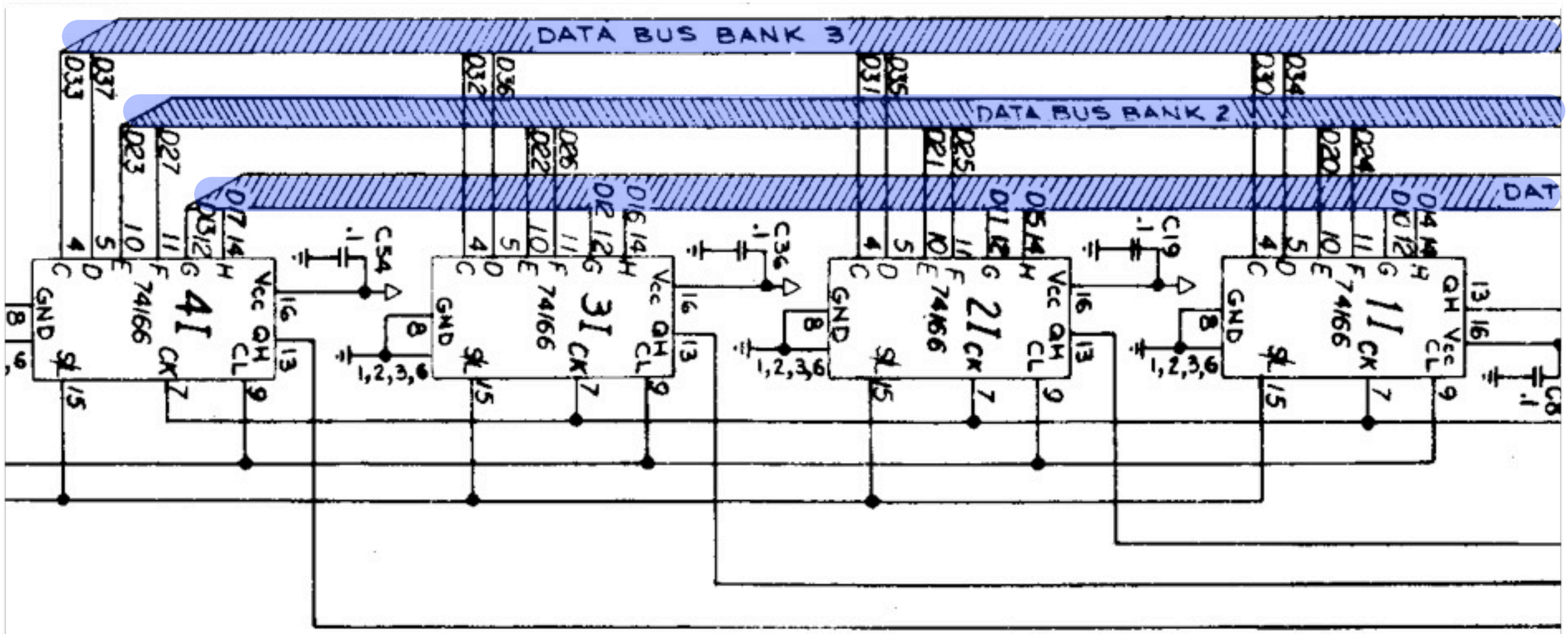
Video - Memory:Pixel Serializer

```
bit value[8]

process shift_instance:
  wait for clock.event
  if load:
    value = input
  else:
    value = value >> 1
  output = value & 1
```

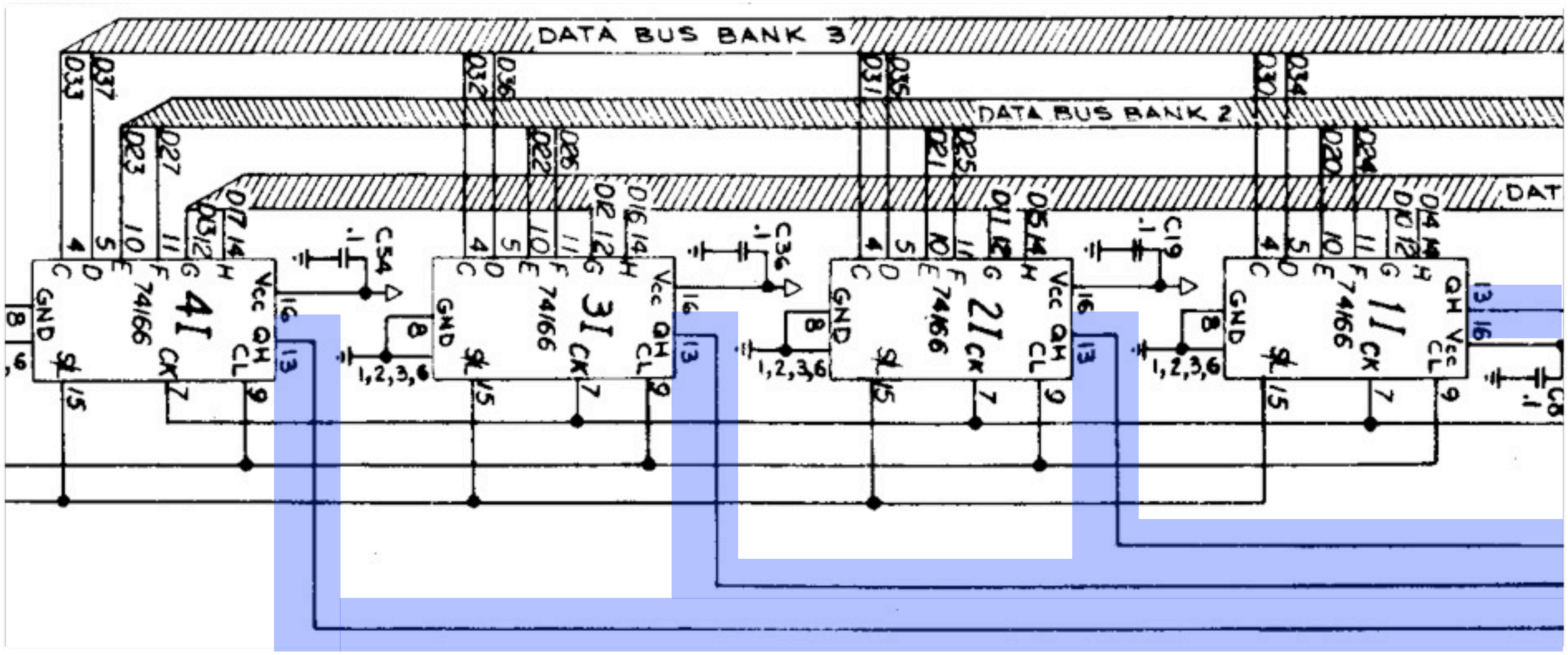

In pseudo-code, this is how an eight-bit shift register would look. The shift register only does work when a clock event occurs. If the load variable is true, the shift register value loaded from the input byte. If the load variable is false, the shift register shifts the register one bit. The output is always the value of the shift register's least-significant bit.

Video - Memory:Pixel Serializer



How are these shift registers connected? The input to the shift registers comes from the memory bank, 24 bits at a time.

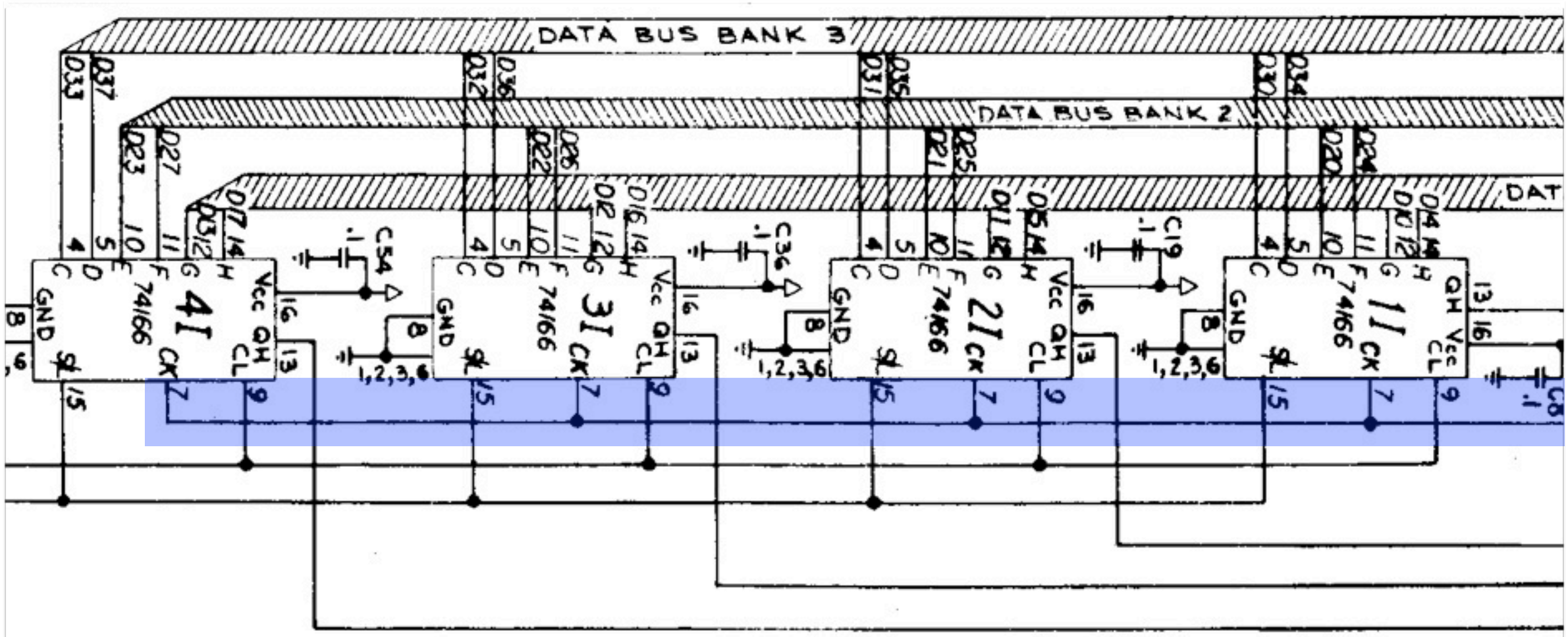
Video - Memory:Pixel Serializer



Saturday, June 22, 13

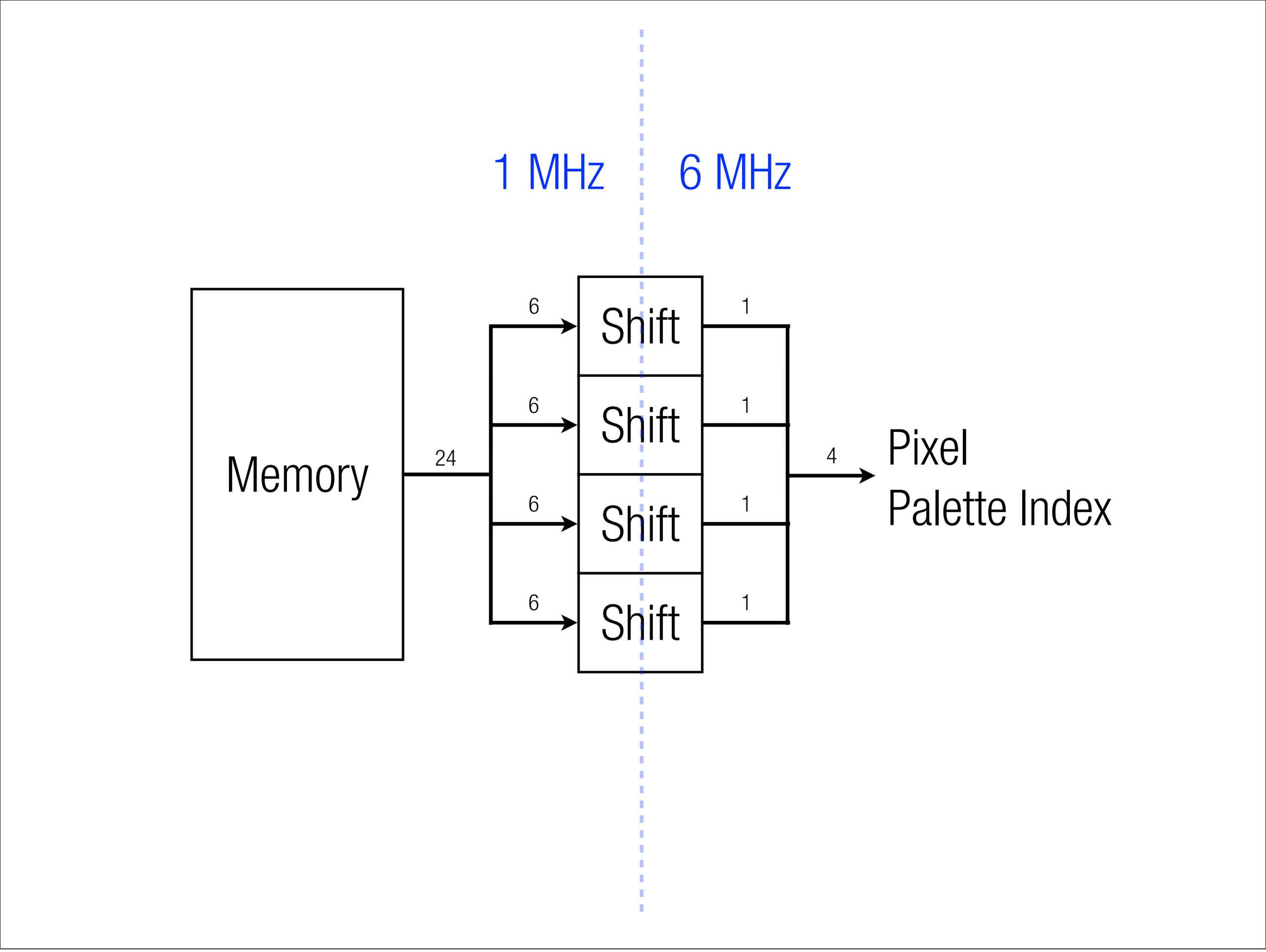
Those 24 bits are then doled out from the shift registers, four bits at a time, into the color palette memory.

Video - Memory:Pixel Serializer



Saturday, June 22, 13

The clock signal going to each shift register controls how fast the bits are shifted out of the shift registers. Elsewhere in the schematic, the clock is shown to be 6 MHz.



So to sum up this bit of the circuit: The pixel data is read from the memory 24 bits at a time, at a rate of 1 MHz, and loaded into the shift registers. The shift registers serialize this data to produce four bits at a time, or one pixel at a time, at a rate of 6 MHz. But why this complicated arrangement with shift registers? Why couldn't the hardware read two pixels (one byte) directly from memory at a rate of 3 MHz and do away with the shift registers?

MOSTEK 16,384 × 1 Bit Dynamic Ram | MK 4116P-2/3

FEATURES

- ☐ Recognized industry standard 16-pin configuration from MOSTEK
- ☐ 150ns access time, 375ns cycle (MK 4116-2)
  200ns access time, 375ns cycle (MK 4116-3)
- ☐ ± 10% tolerance on all power supplies (+12V, ±5V)
- ☐ Low power: 462mW active, 20mW standby (max)
- ☐ Output data controlled by $\overline{CAS}$ and unlatched at end of cycle to allow two dimensional chip selection and extended page boundary
- ☐ Common I/O capability using "early write" operation
- ☐ Read-Modify-Write, $\overline{RAS}$-only refresh, and Page-mode capability
- ☐ All inputs TTL compatible,low capacitance, and protected against static charge
- ☐ 128 refresh cycles

DESCRIPTION

The MK 4116 is a new generation MOS dynamic random access memory circuit organized as 16,384 words by 1 bit. As a state-of-the-art MOS memory device, the MK 4116 (16K RAM) incorporates advanced circuit techniques designed to provide wide operating margins, both internally and to the system user, while achieving performance levels in speed and power previously seen only in MOSTEK's high performance MK 4027 (4K RAM).

The technology used to fabricate the MK 4116 is MOSTEK's double-poly, N-channel silicon gate, POLY II™ process. This process, coupled with the use of a single transistor dynamic storage cell, provides the maximum possible circuit density and reliability, while maintaining high performance capability. The use of dynamic circuitry throughout, including sense amplifiers, assures that power dissipation is minimized without any sacrifice in speed or operating margin. These factors combine to make the MK 4116 a truly superior RAM product.

Multiplexed address inputs (a feature pioneered by MOSTEK for its 4K RAMS) permits the MK 4116 to be packaged in a standard 16-pin DIP. This recognized industry standard package configuration, while compatible with widely available automated testing and insertion equipment, provides highest possible system bit densities and simplifies system upgrade from 4K to 16K RAMs for new generation applications. Non-critical clock timing requirements allow use of the multiplexing technique while maintaining high performance.

FUNCTIONAL DIAGRAM

PIN CONNECTIONS

Saturday, June 22, 13

To answer that, have a look at the datasheet for the RAM chips that make up the memory bank where the video frame buffer lives.

MOSTEK 16,384 × 1 Bit Dynamic Ram | MK 4116P-2/3

FEATURES

- ☐ Recognized industry standard 16-pin configuration from MOSTEK
- ☐ 150ns access time, 375ns cycle (MK 4116-2)
  200ns access time, 375ns cycle (MK 4116-3)
- ☐ ± 10% tolerance on all power supplies (+12V, ±5V)
- ☐ Low power: 462mW active, 20mW standby (max)
- ☐ Output data controlled by $\overline{CAS}$ and unlatched at end of cycle to allow two dimensional chip selection and extended page boundary
- ☐ Common I/O capability using "early write" operation
- ☐ Read-Modify-Write, $\overline{RAS}$-only refresh, and Page-mode capability
- ☐ All inputs TTL compatible,low capacitance, and protected against static charge
- ☐ 128 refresh cycles

DESCRIPTION

The MK 4116 is a new generation MOS dynamic random access memory circuit organized as 16,384 words by 1 bit. As a state-of-the-art MOS memory device, the MK 4116 (16K RAM) incorporates advanced circuit techniques designed to provide wide operating margins, both internally and to the system user, while achieving performance levels in speed and power previously seen only in MOSTEK's high performance MK 4027 (4K RAM).

The technology used to fabricate the MK 4116 is MOSTEK's double-poly, N-channel silicon gate, POLY II™ process. This process, coupled with the use of a single transistor dynamic storage cell, provides the maximum possible circuit density and reliability, while maintaining high performance capability. The use of dynamic circuitry throughout, including sense amplifiers, assures that power dissipation is minimized without any sacrifice in speed or operating margin. These factors combine to make the MK 4116 a truly superior RAM product.

Multiplexed address inputs (a feature pioneered by MOSTEK for its 4K RAMS) permits the MK 4116 to be packaged in a standard 16-pin DIP. This recognized industry standard package configuration, while compatible with widely available automated testing and insertion equipment, provides highest possible system bit densities and simplifies system upgrade from 4K to 16K RAMs for new generation applications. Non-critical clock timing requirements allow use of the multiplexing technique while maintaining high performance.

FUNCTIONAL DIAGRAM

PIN CONNECTIONS

Saturday, June 22, 13

You'll see that this chip takes between 150 ns and 200 ns to access data, depending on whether you use the faster or slower version of the chip. Just like DDR3 memory speeds in modern computers, fast memory chips in 1982 commanded a premium price. In fact, the designers specified a much older model of this chip that took 450 ns to access data!

MOSTEK 16,384 × 1 Bit Dynamic Ram | MK 4116P-2/3

FEATURES

- ☐ Recognized industry standard 16-pin configuration from MOSTEK
- ☐ 150ns access time, 375ns cycle (MK 4116-2)
  200ns access time, 375ns cycle (MK 4116-3)
- ☐ ± 10% tolerance on all power supplies (+12V, +5V)
- ☐ Low power: 462m
- ☐ Output data contro
  end of cycle to allo
  tion and extended
- ☐ Common I/O capability using "early write" operation
- ☐ Read-Modify-Write, $\overline{RAS}$-only refresh, and Page-mode capability
- ☐ All inputs TTL compatible, low capacitance, and

```
> time = 450e-9      # 450 ns

> frequency = 1.0 / time

> print(frequency)
2222222.2222222
```

DESCRIPTION

The MK 4116 is a
random access memo
words by 1 bit. As a state-of-the-art MOS memory device, the MK 4116 (16K RAM) incorporates advanced circuit techniques designed to provide wide operating margins, both internally and to the system user, while achieving performance levels in speed and power previously seen only in MOSTEK's high performance MK 4027 (4K RAM).

The technology used to fabricate the MK 4116 is MOSTEK's double-poly, N-channel silicon gate, POLY II™ process. This process, coupled with the use of a single transistor dynamic storage cell, provides the maximum possible circuit density and reliability, while maintaining high performance

c circuitry through-
assures that power
ut any sacrifice in
ese factors combine
to make the MK 4116 a truly superior RAM product.

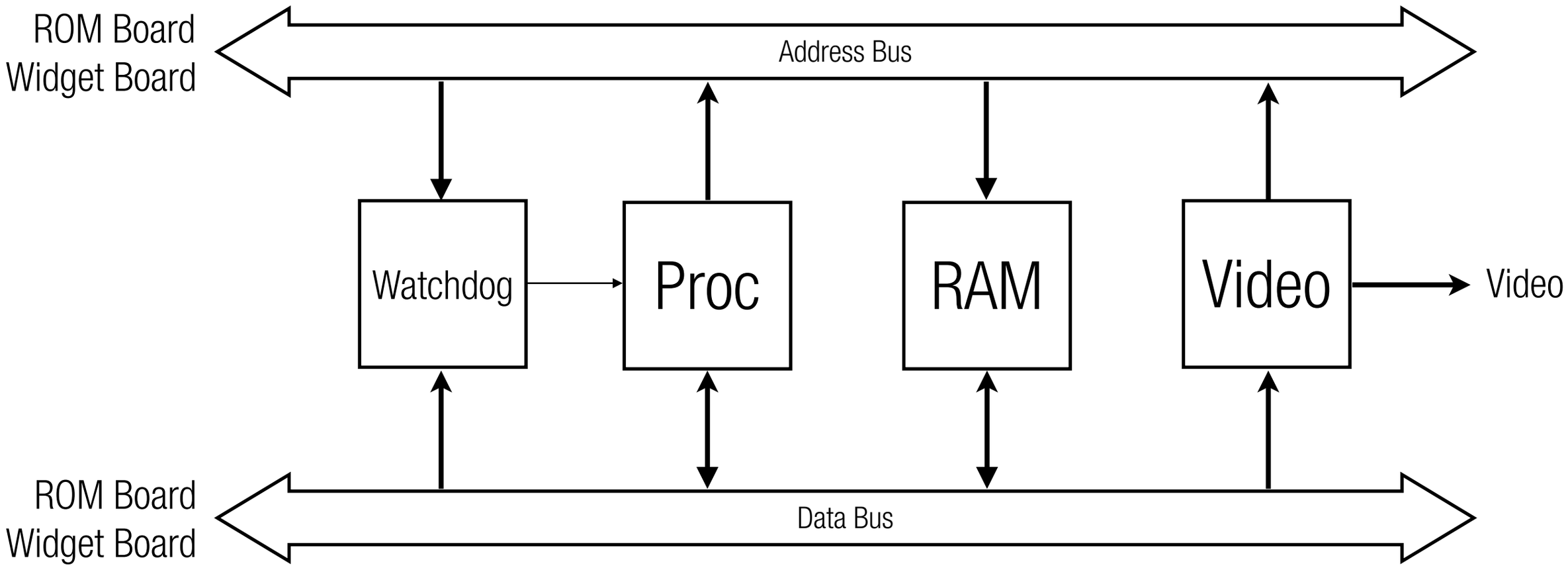
Multiplexed address inputs (a feature pioneered by MOSTEK for its 4K RAMS) permits the MK 4116 to be packaged in a standard 16-pin DIP. This recognized industry standard package configuration, while compatible with widely available automated testing and insertion equipment, provides highest possible system bit densities and simplifies system upgrade from 4K to 16K RAMs for new generation applications. Non-critical clock timing requirements allow use of the multiplexing technique while maintaining high performance.

FUNCTIONAL DIAGRAM

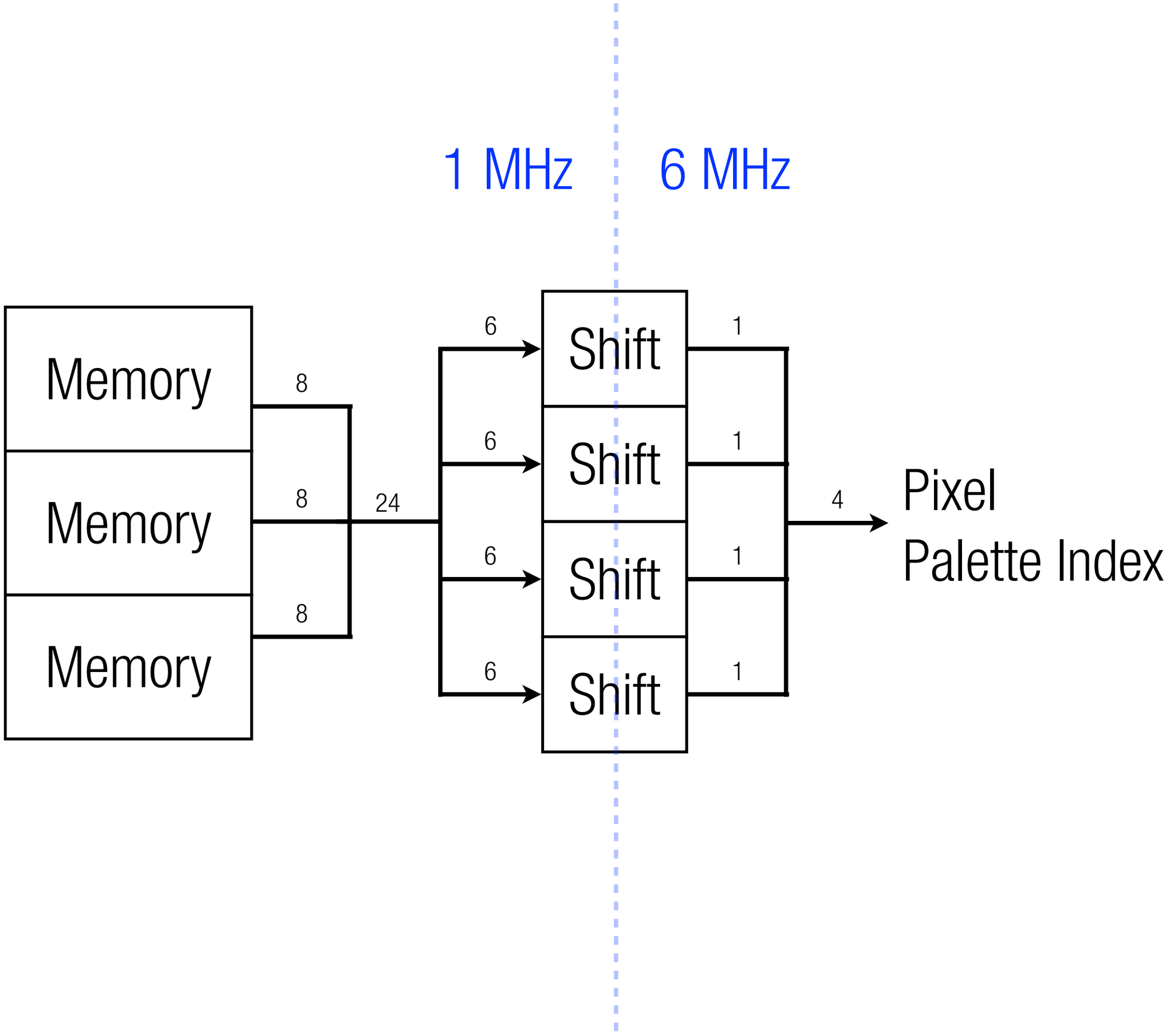
PIN CONNECTIONS

Saturday, June 22, 13

So how many accesses per second can a 450 ns memory chip perform? 2.2 million. Remember how we needed to read 3 million bytes a second from memory to have a video pixel output rate of 6 million pixels a second? 450 ns memory chips clearly aren't fast enough.



Add to that the problem that the memory is being shared with the processor. The processor has a 1 MHz clock cycle, and needs to access the memory up to 1 million times a second. So we need a combined memory bandwidth of 4 million bytes a second for both the video and processor.



This is why the hardware designers made the memory bank three bytes wide instead of one byte wide, and added the shift registers for the pixel data. If memory accesses take 450 ns, but you're reading three bytes each time instead of just one byte, your aggregate bandwidth is 6.7 million bytes per second -- enough to satisfy the combined 4 million bytes per second for the video hardware and the processor.

Video - Time for a Cocktail

Stephan Suys: http://www.arcade.chezsuys.com/RoboCocktail.html

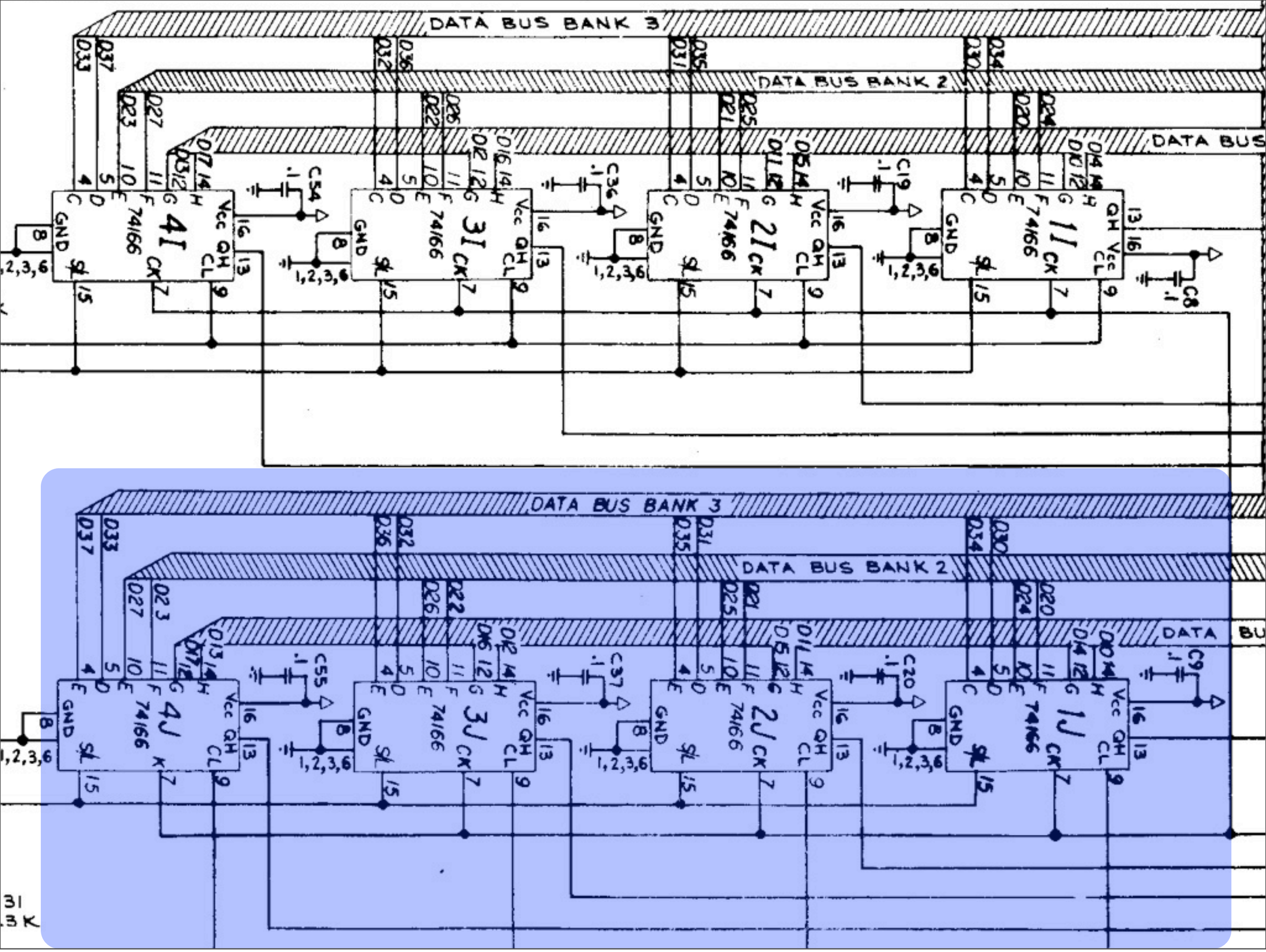
While we're on the subject of video hardware, I should tell you about the cocktail cabinet versions of Robotron. These are cabinets built like tables. Two players sit on opposite ends of the machine, with the screen in between (or a baby, in this case), and play two-player games against each other.

Video - Time for a Cocktail

I suppose it goes without saying that the image on the screen needs to flip upside-down for player 2 to be able to play the game.

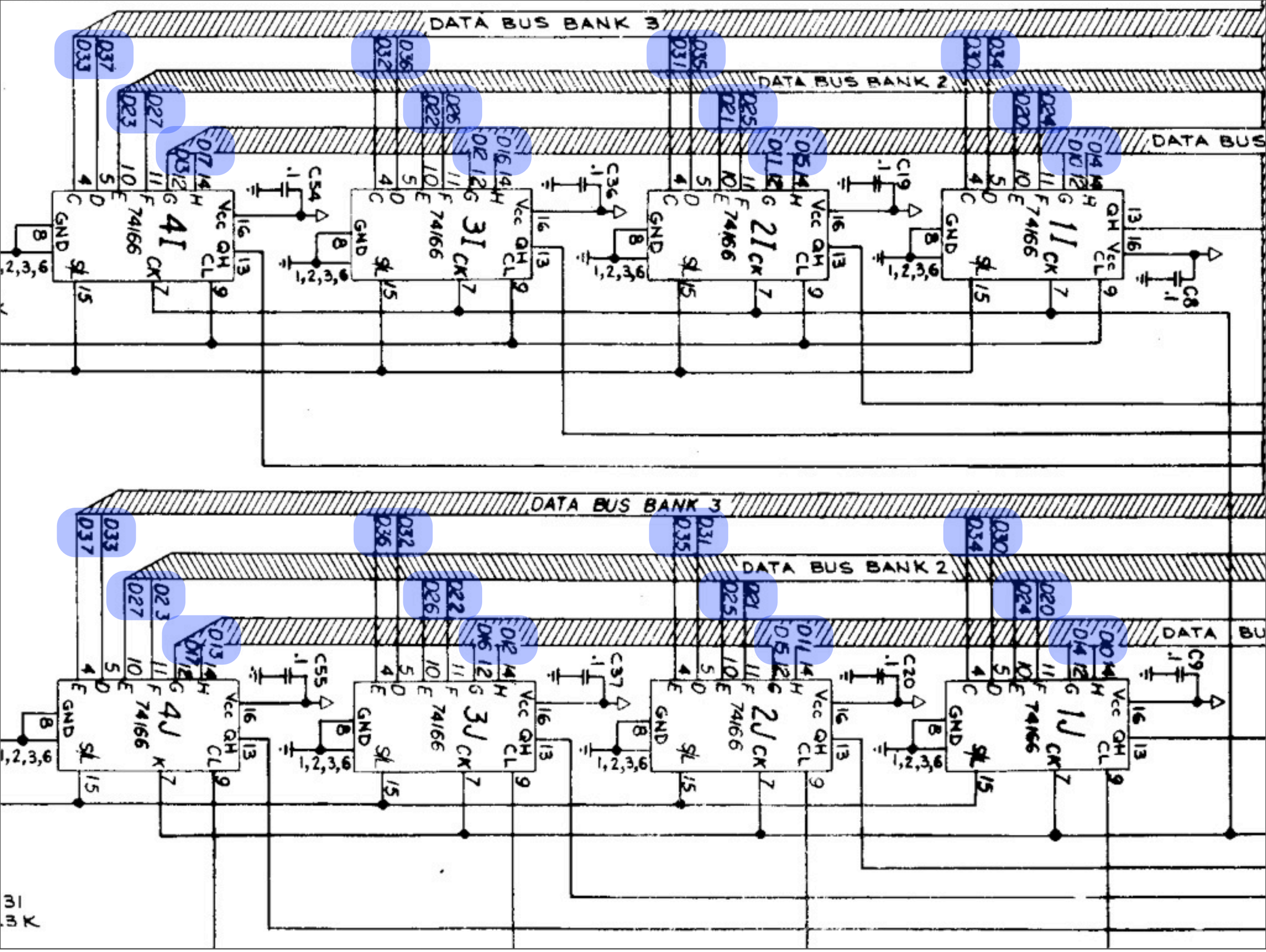
Video - Time for a Cocktail

I suppose it goes without saying that the image on the screen needs to flip upside-down for player 2 to be able to play the game.



Saturday, June 22, 13

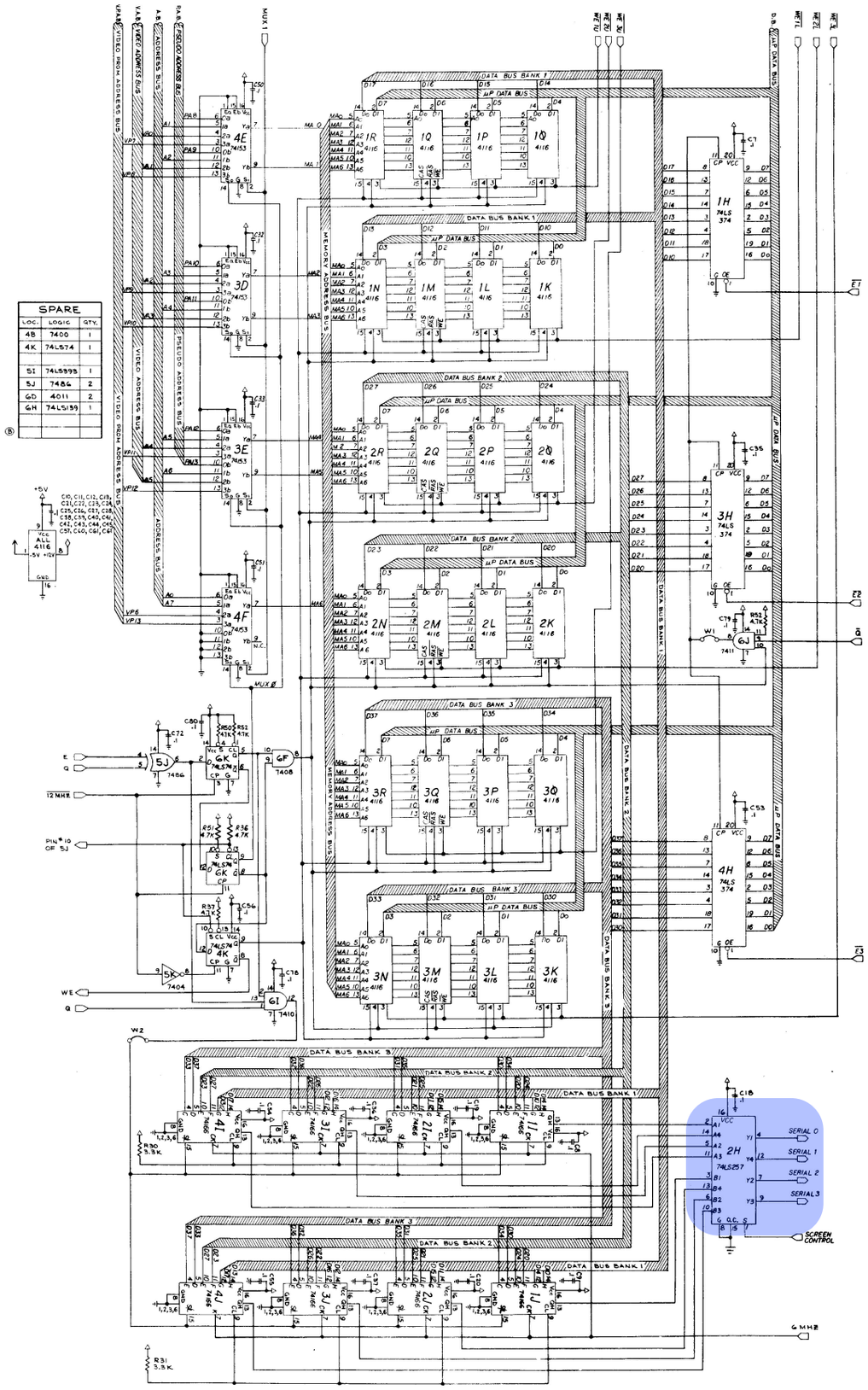
I conveniently ignored the fact that there's a second set of pixel shift registers next to the ones we just discussed. There's a subtle difference, though...

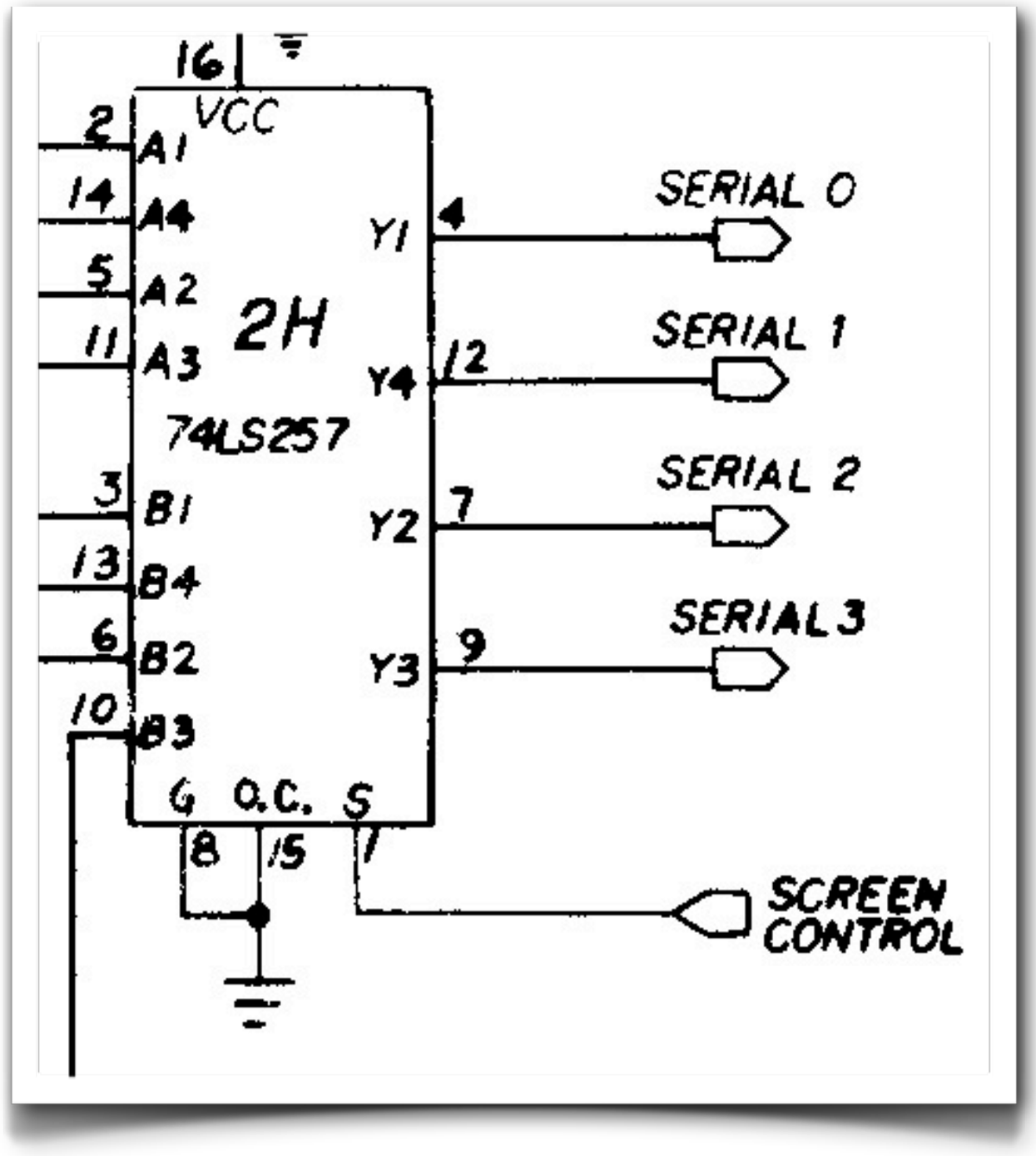


Saturday, June 22, 13

The bits from memory are connected to the bottom shift registers in a different order from the top, reversing the order of pairs of pixels. Instead of the pixel order being 1-2, 3-4, 5-6, it's 2-1, 4-3, 6-5. Very interesting...

Video - Screen Control

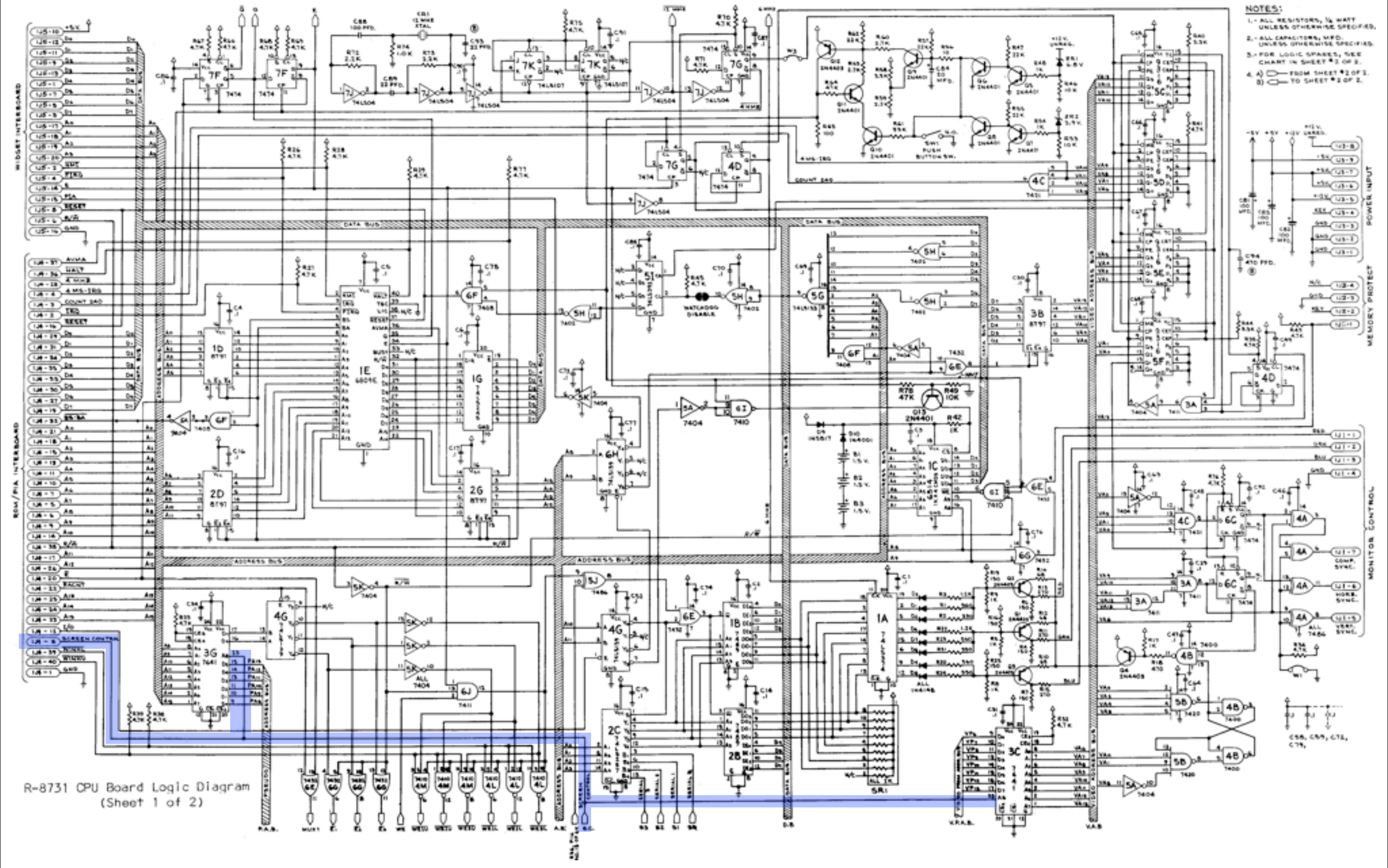




Saturday, June 22, 13

The two sets of pixel shift registers flow into a 2:1 selector, which switches between the output of the top shift registers and the bottom shift registers, based on a mysterious signal called “SCREEN CONTROL”. Let’s look around for other places on the schematics where this SCREEN CONTROL signal appears.

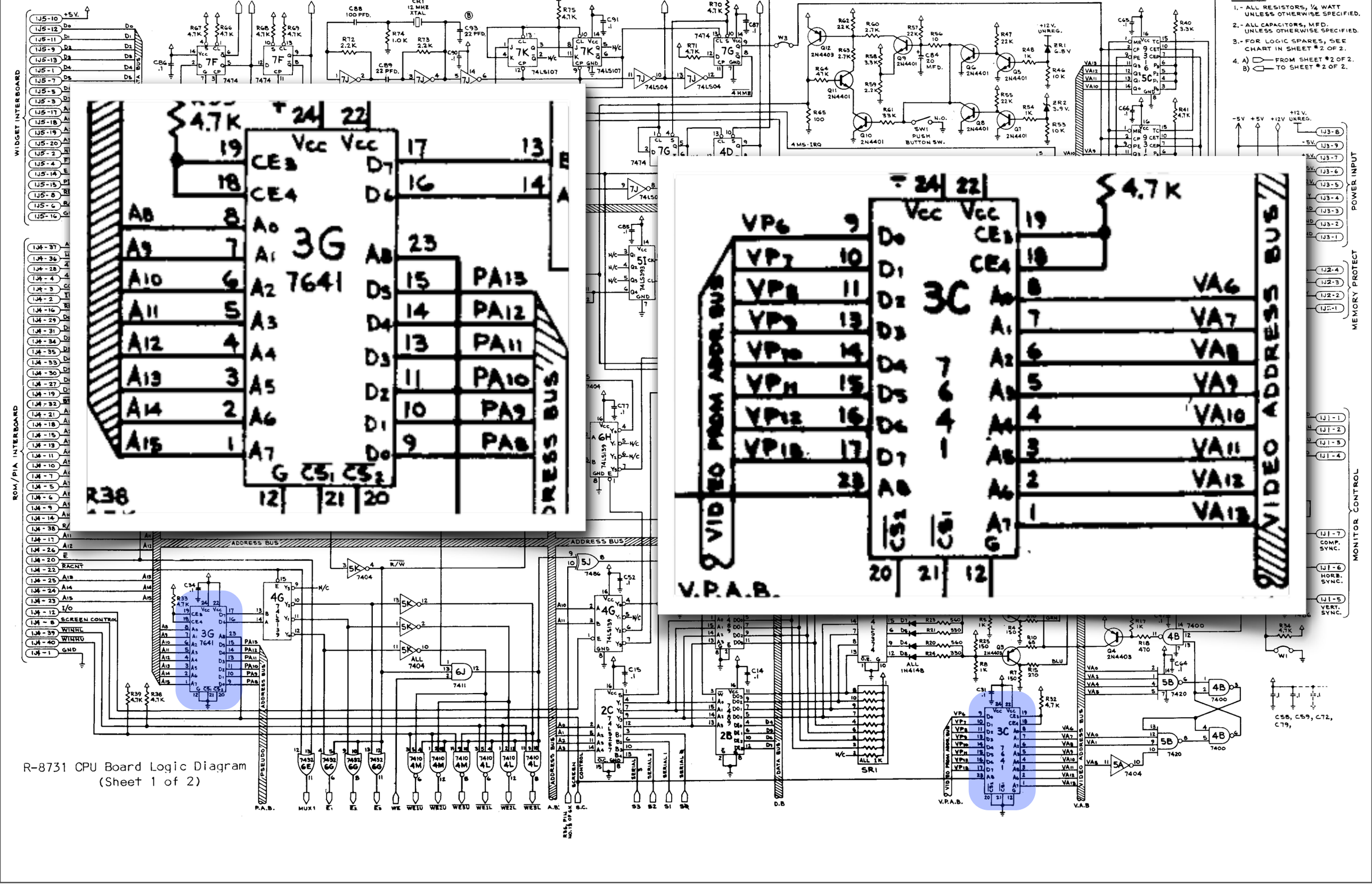
Video - Screen Control



Saturday, June 22, 13

SCREEN CONTROL shows up on the first page of the CPU board schematic, where it goes off to the ROM board. But it also goes into two chips.

Video - Screen Control



Saturday, June 22, 13

Both are the same kind of a chip, the 7641.

HARRIS
SEMICONDUCTOR
PRODUCTS DIVISION
A DIVISION OF HARRIS CORPORATION

HM-7640/41

512 x 8 PROM

HM-7640 — Open Collector Outputs
HM-7641 — "Three State" Outputs

Features

- 70ns MAXIMUM ADDRESS ACCESS TIME
- "THREE STATE" OR OPEN COLLECTOR OUTPUTS AND FOUR CHIP ENABLE INPUTS.
- SIMPLE HIGH SPEED PROGRAMMING PROCEDURE – ONE PULSE/BIT. ASSURES FAST PROGRAMMING AND SUPERIOR RELIABILITY.
- FAST ACCESS TIME – GUARANTEED FOR WORST CASE N^2 SEQUENCING OVER COMMERCIAL AND MILITARY TEMPERATURE AND VOLTAGE RANGES.
- INDUSTRY'S HIGHEST PROGRAMMING YIELD

Description

The HM-7640/41 are fully decoded high speed Schottky TTL 4096-Bit Field Programmable ROMs in a 512 word by 8 bit/word format and are available in a 24 pin DIP (ceramic or epoxy) and a 24 pin flatpack.

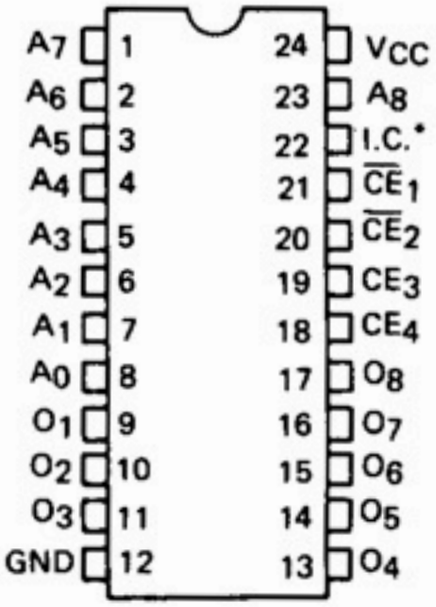
All bits are manufactured storing a logical "1" (positive logic) and can be selectively programmed for a logical "0" in any bit position.

Nickel-chromium fuse technology is used on this and all other Harris Bipolar PROMs.

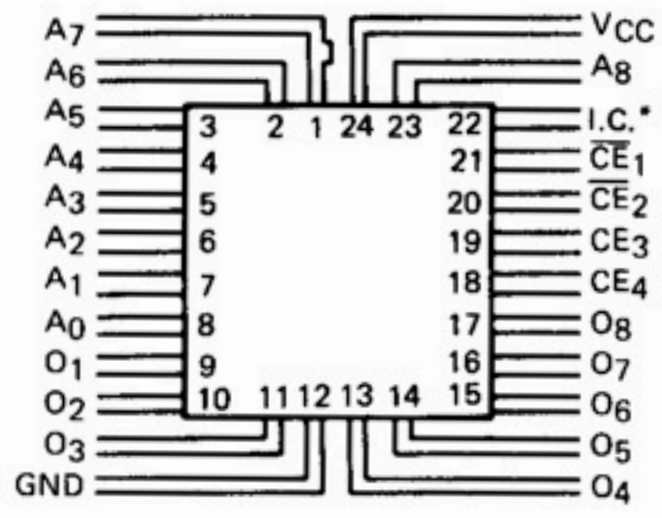
The HM-7640/41 contain test rows and columns which are in addition to the storage array to assure high programmability and guarantee parametric and A.C. performance. The fuses in these test rows and columns are blown prior to shipment.

Pinouts

TOP VIEW – DIP



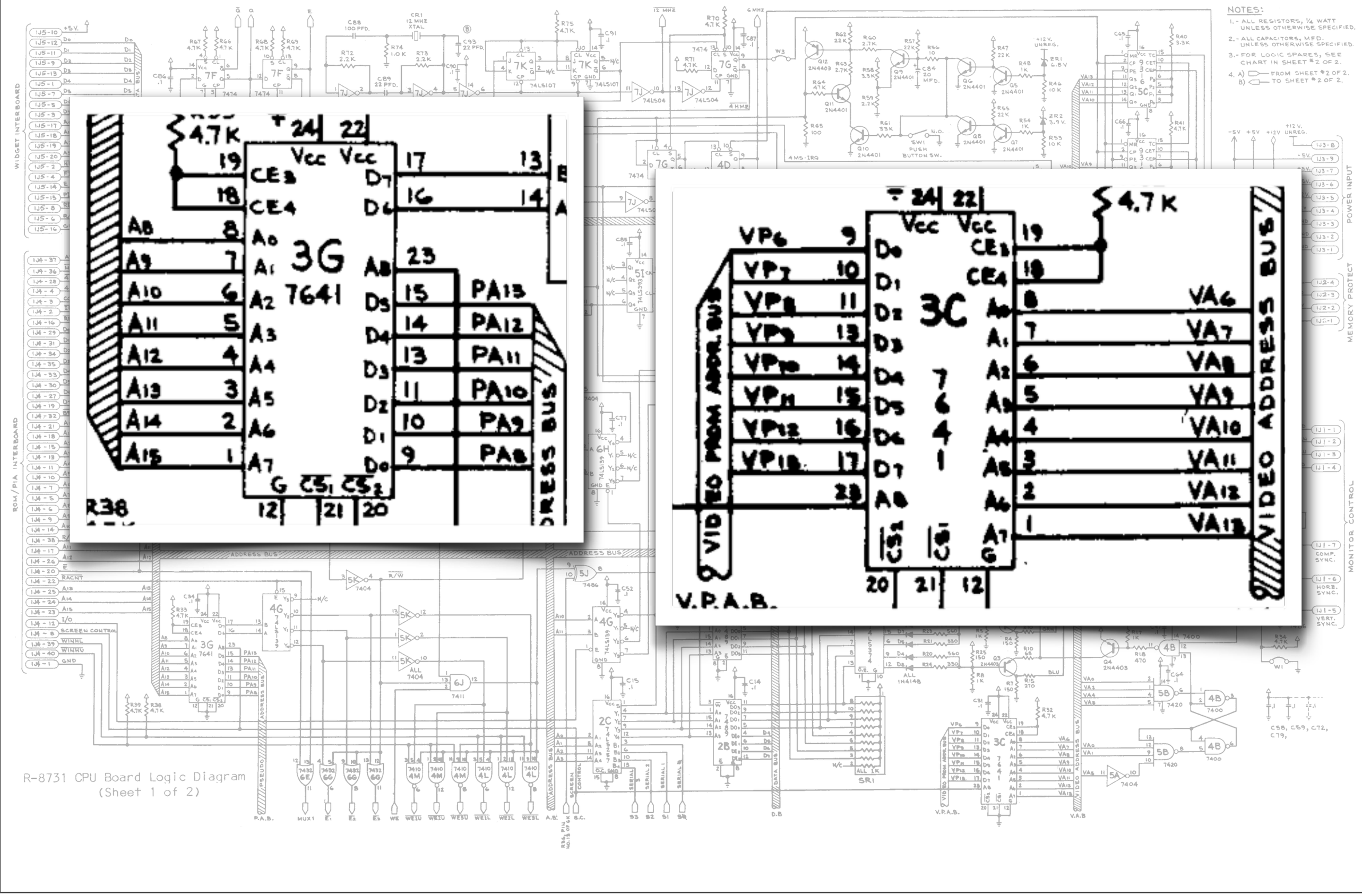
TOP VIEW – FLATPACK



Saturday, June 22, 13

The 7641 is a 512 byte programmable read-only memory (or PROM).

Video - Screen Control

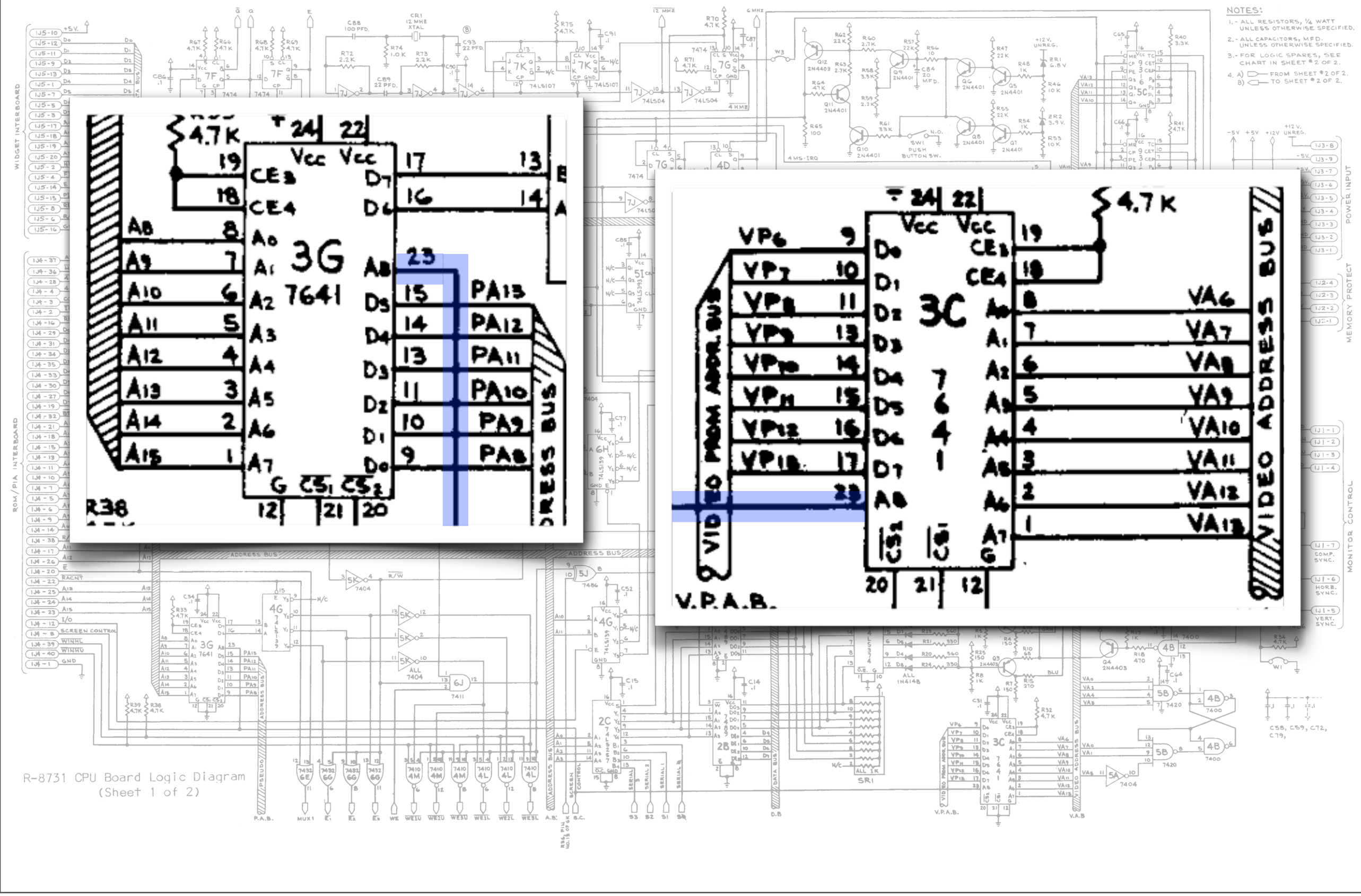


Saturday, June 22, 13

The PROM chip on the left is connected between the processor address bus and a “pseudo” address bus. The pseudo address bus goes to the 48K memory bank, where the video frame buffer and game runtime state lives.

The PROM chip on the right is connected between the 48K memory bank and the video pixel counter, which keeps track of which video pixel is being sent to the display at any instant in time.

Video - Screen Control

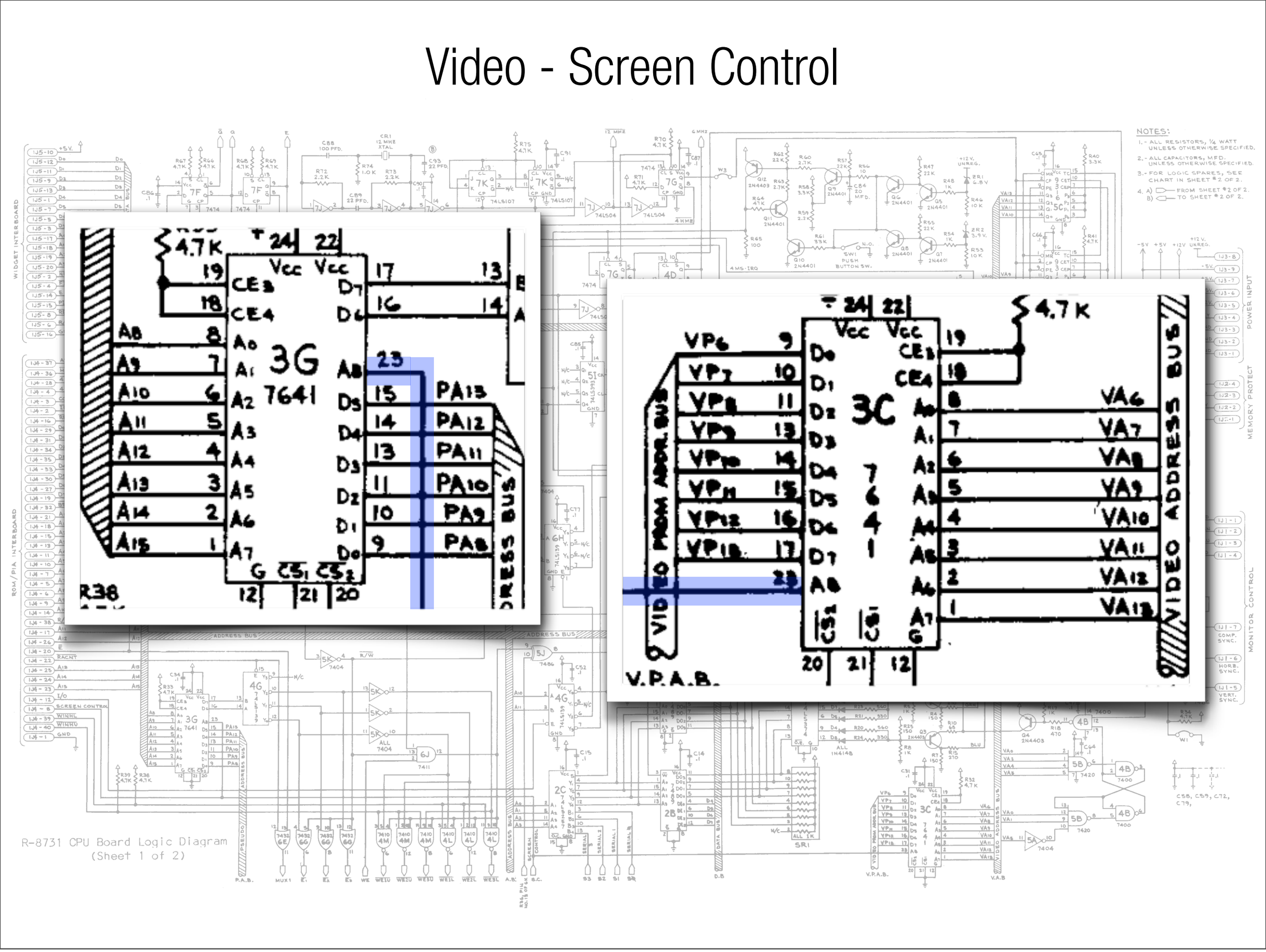


Saturday, June 22, 13

The SCREEN CONTROL signal comes in to one of the address lines of each PROM chip. So what is it doing?

The two chips work in concert to change the order pixels are written to memory, and the order they're read for memory when being scanned onto the display. By changing the SCREEN CONTROL signal, the image on the screen will reverse without any additional effort from the processor.

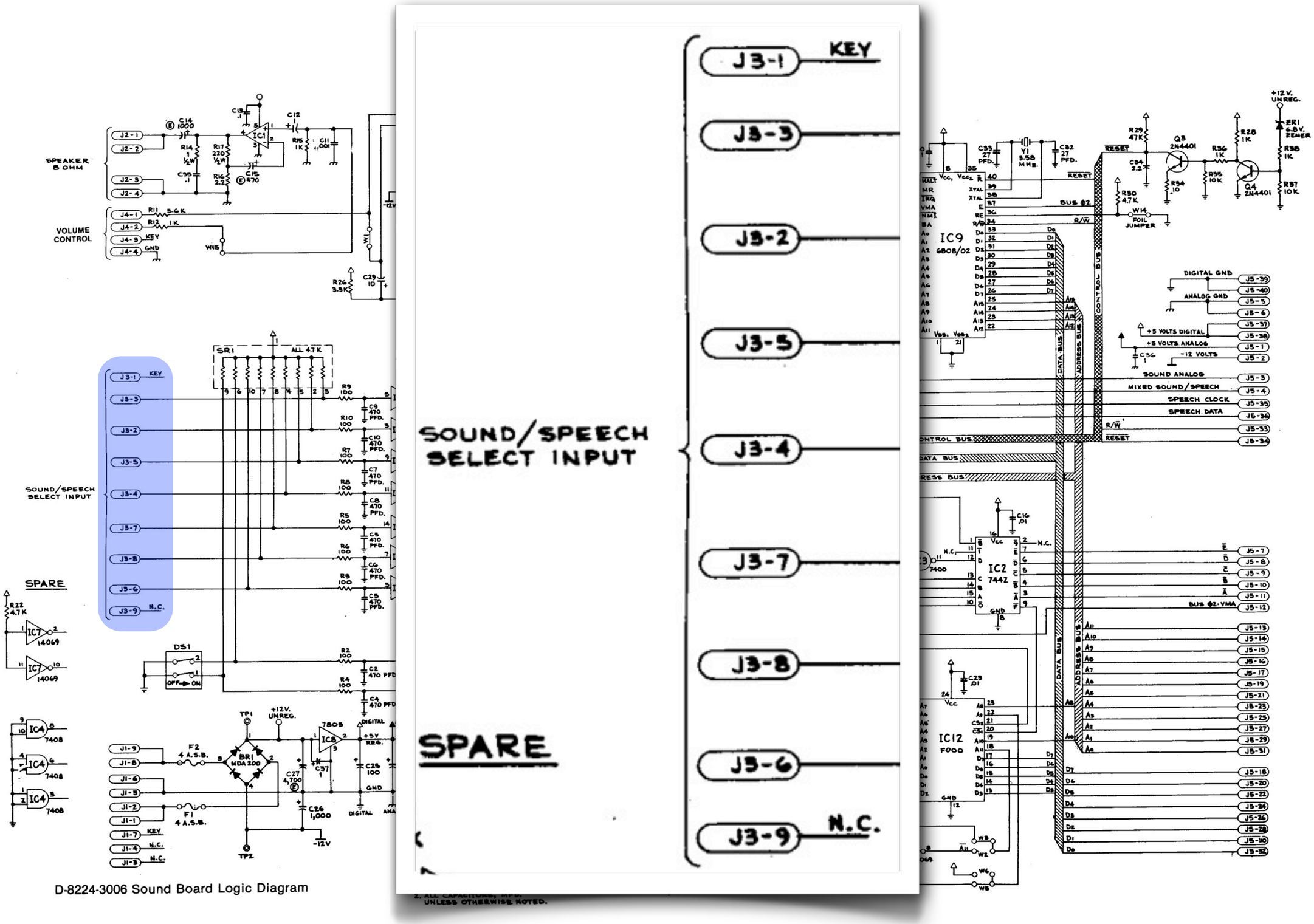
Video - Screen Control



Saturday, June 22, 13

Why do this? Remember, this processor runs at 1MHz, and executes maybe 300,000 instructions per second (modern processors perform billions of instructions per second). The processor doesn't have time to do the extra math to reverse coordinates when it's drawing. And there's no fancy, modern GPU chip to do it, either. The designers solved the problem with a simple bit of hardware that alters the way memory is addressed, and it's transparent to the processor.

Sound Board



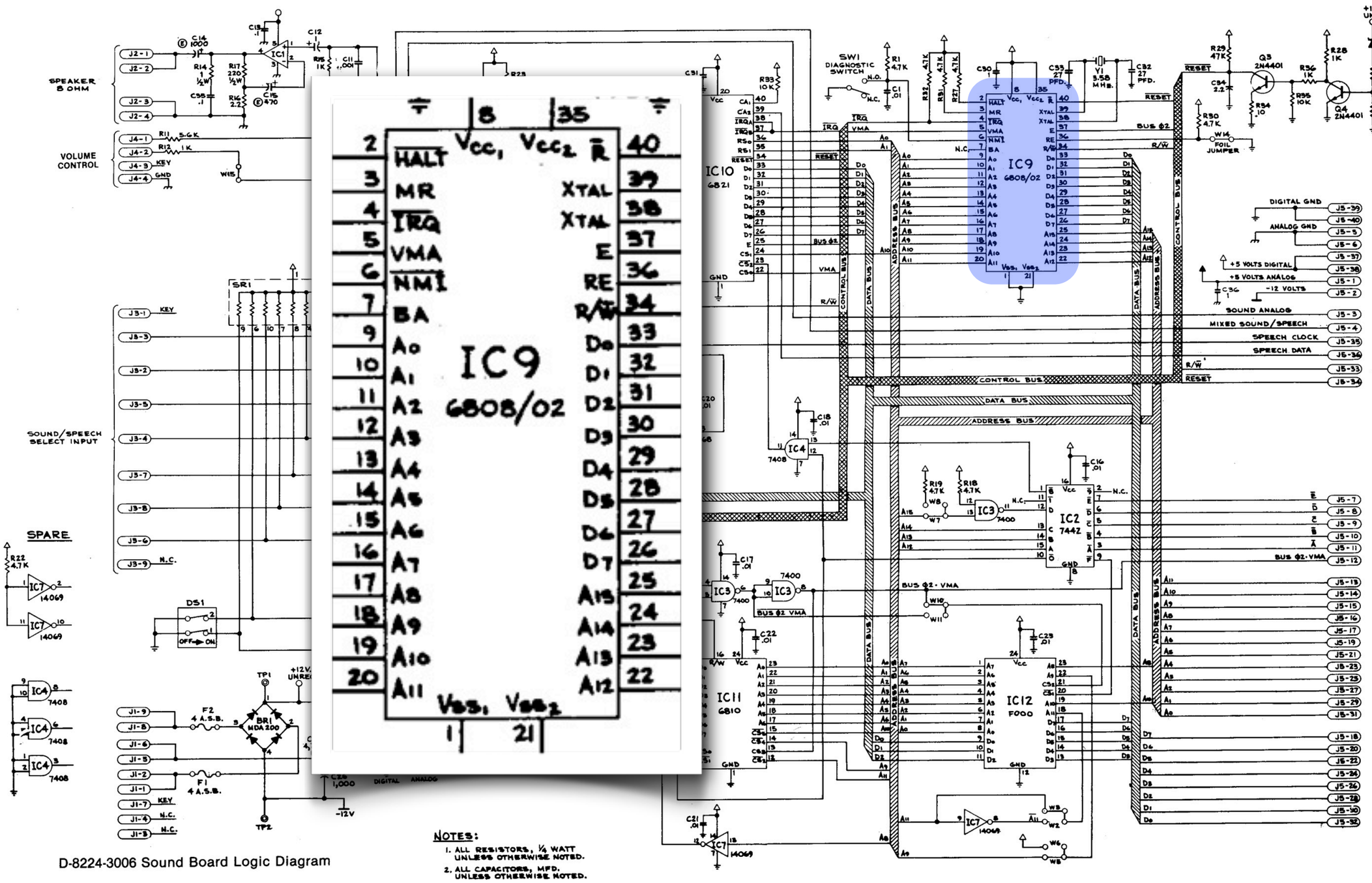
D-8224-3006 Sound Board Logic Diagram

Saturday, June 22, 13

I've touched on most of the interesting stuff on the ROM board, so let's have a quick look at the sound board. Recall that we found all the chips required to make a full computer -- an 8-bit processor, a RAM chip, and some ROM for program code.

Here is the connection to the ROM board. The processor on the CPU board controls the sound board through this interface.

Sound Board

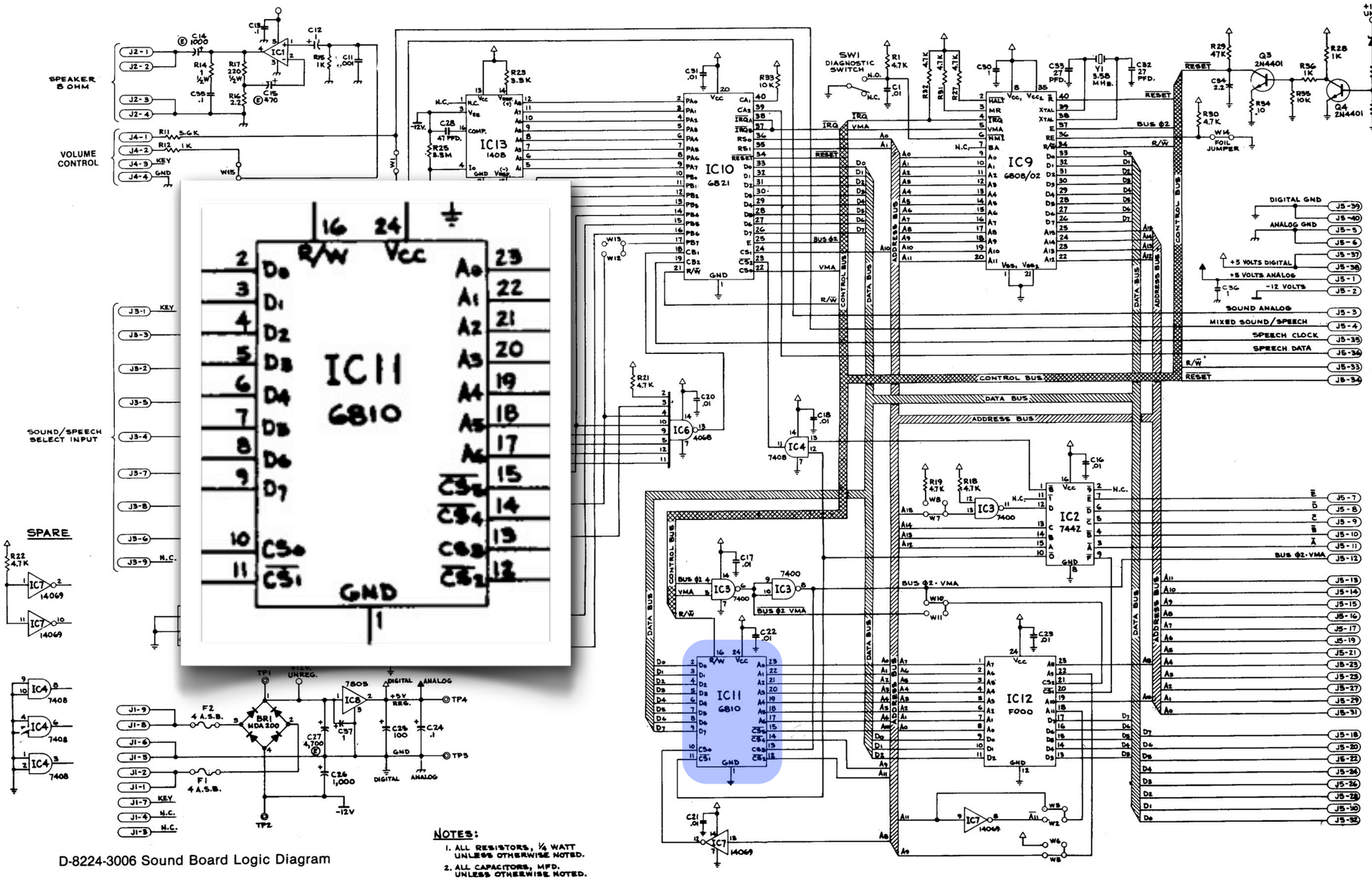


D-8224-3006 Sound Board Logic Diagram

Saturday, June 22, 13

Here's the MC6802 microprocessor.

Sound Board

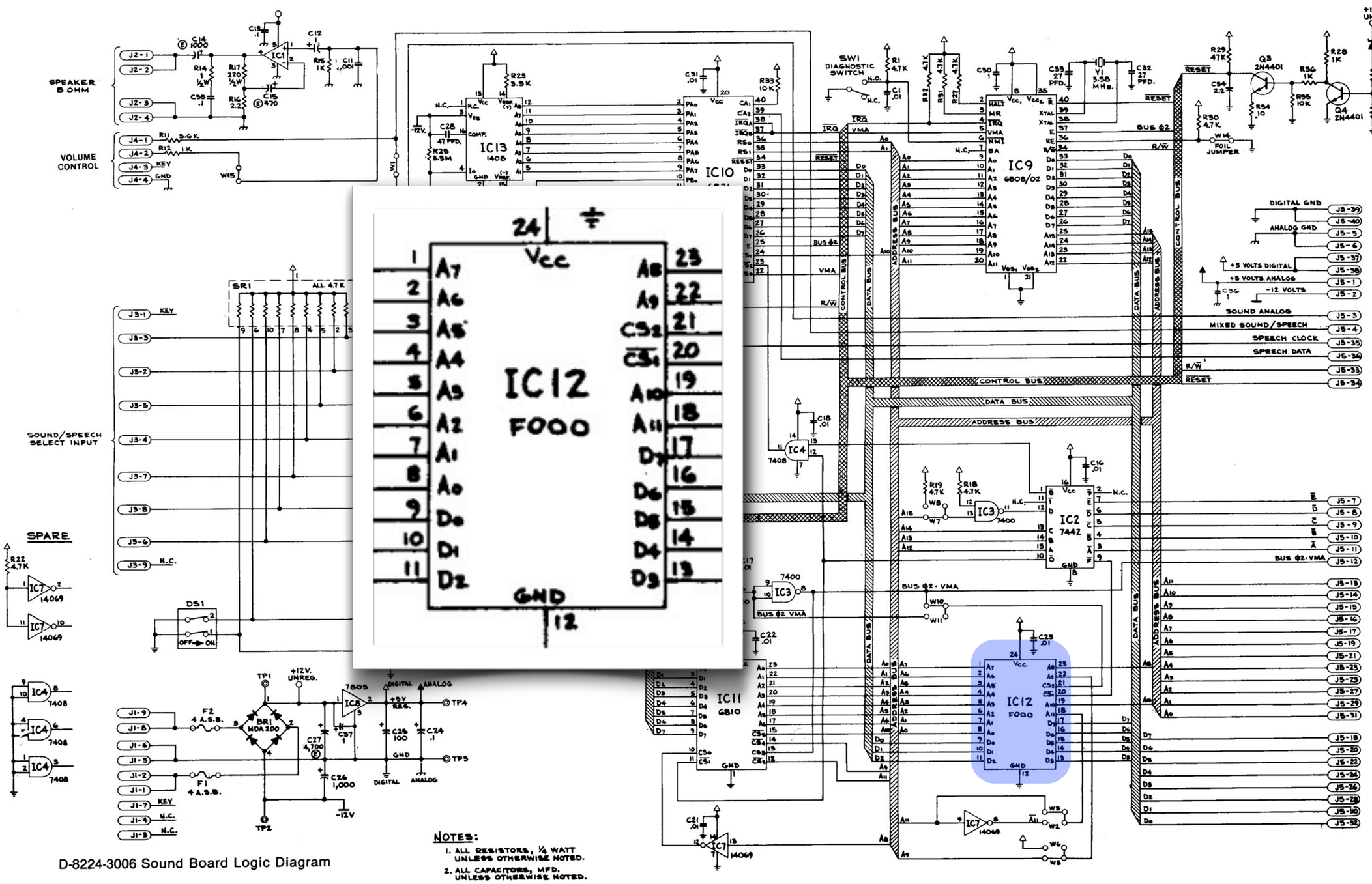


D-8224-3006 Sound Board Logic Diagram

Saturday, June 22, 13

The 128 byte memory.

Sound Board

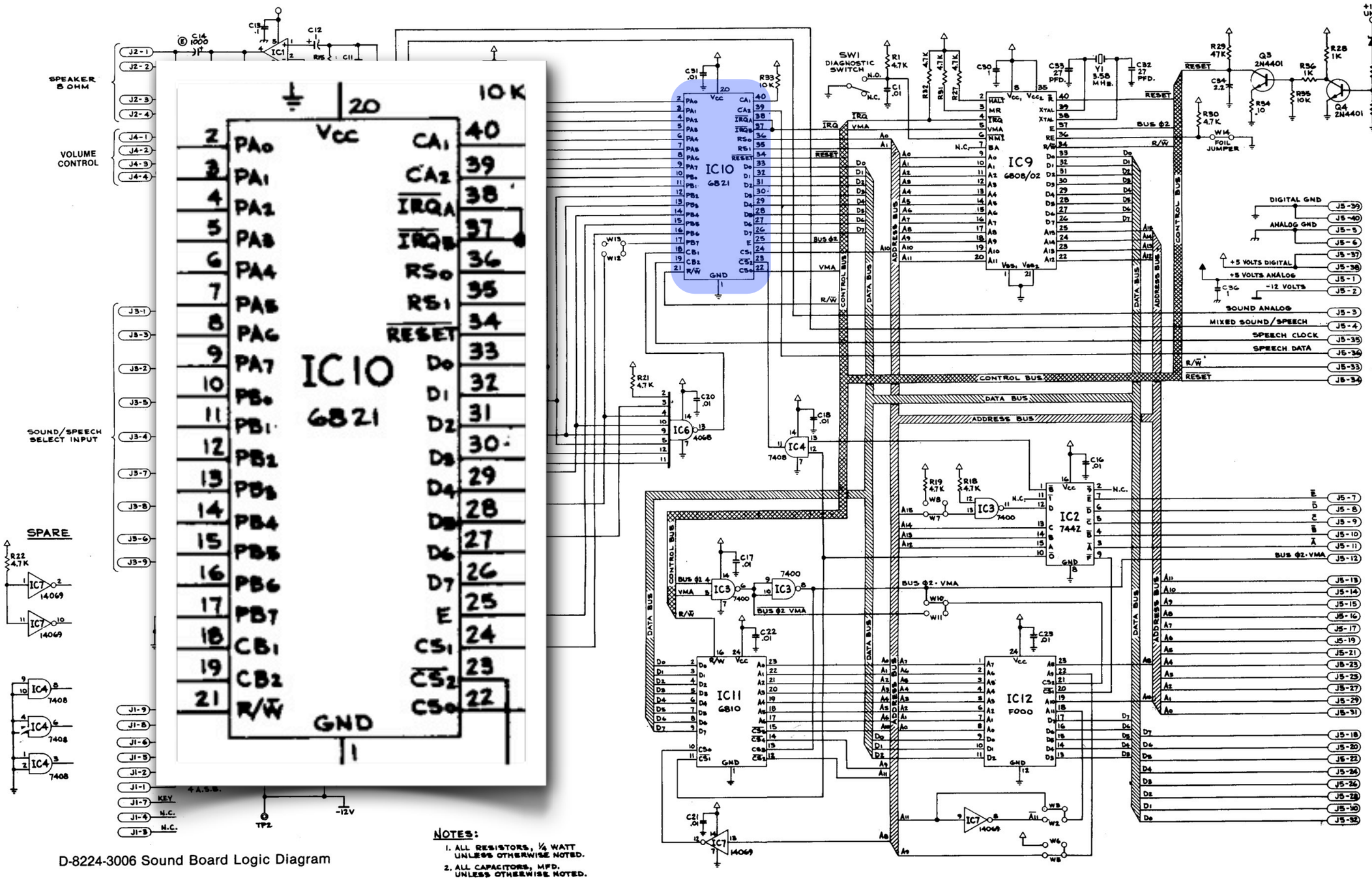


D-8224-3006 Sound Board Logic Diagram

Saturday, June 22, 13

The ROM chip.

Sound Board



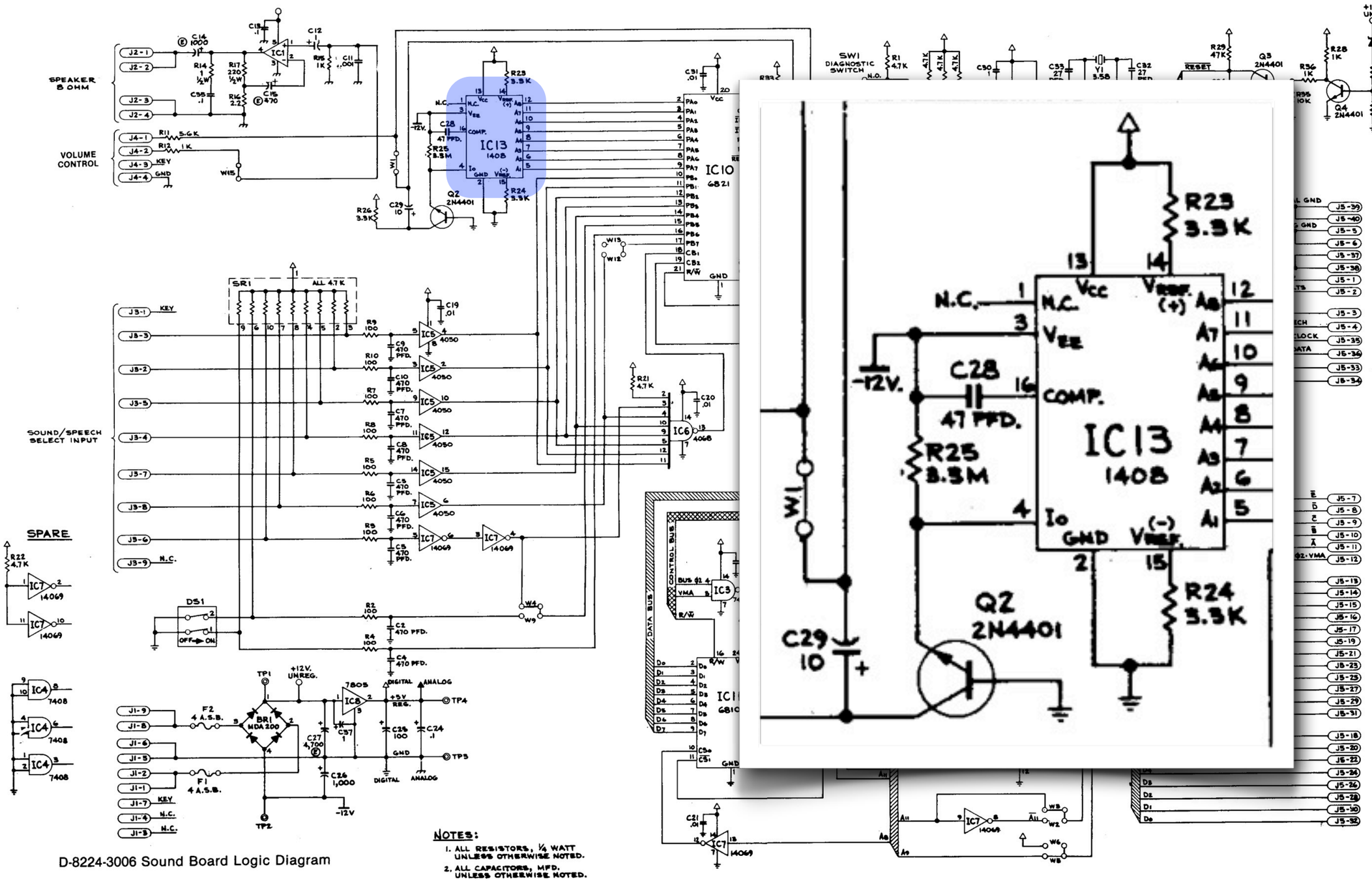
D-8224-3006 Sound Board Logic Diagram

NOTES:
1. ALL RESISTORS, ¼ WATT UNLESS OTHERWISE NOTED.
2. ALL CAPACITORS, MFD. UNLESS OTHERWISE NOTED.

Saturday, June 22, 13

...and yet another MC6821 Peripheral Interface Adapter.

Sound Board



D-8224-3006 Sound Board Logic Diagram

NOTES:
1. ALL RESISTORS, ¼ WATT UNLESS OTHERWISE NOTED.
2. ALL CAPACITORS, MFD. UNLESS OTHERWISE NOTED.

Saturday, June 22, 13

Connected to the Peripheral Interface Adapter is an MC1408 chip, which is then connected to a speaker and volume control.

MOTOROLA
SEMICONDUCTOR
TECHNICAL DATA

MC1408
MC1508

EIGHT-BIT MULTIPLYING DIGITAL-TO-ANALOG CONVERTER

. . . designed for use where the output current is a linear product of an eight-bit digital word and an analog input voltage.

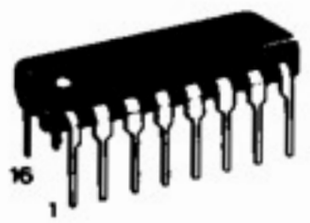
- Eight-Bit Accuracy Available in Both Temperature Ranges
  Relative Accuracy: ±0.19% Error maximum
  (MC1408L8, MC1408P8, MC1508L8)
- Seven and Six-Bit Accuracy Available with MC1408 Designated by 7 or 6 Suffix after Package Suffix
- Fast Settling Time — 300 ns typical
- Noninverting Digital Inputs are MTTL and CMOS Compatible
- Output Voltage Swing — +0.4 V to -5.0 V
- High-Speed Multiplying Input
  Slew Rate 4.0 mA/μs
- Standard Supply Voltages: +5.0 V and -5.0 V to -15 V

EIGHT-BIT MULTIPLYING DIGITAL-TO-ANALOG CONVERTER

SILICON MONOLITHIC INTEGRATED CIRCUIT

L SUFFIX
CERAMIC PACKAGE
CASE 620

P SUFFIX
PLASTIC PACKAGE
CASE 648



ORDERING INFORMATION

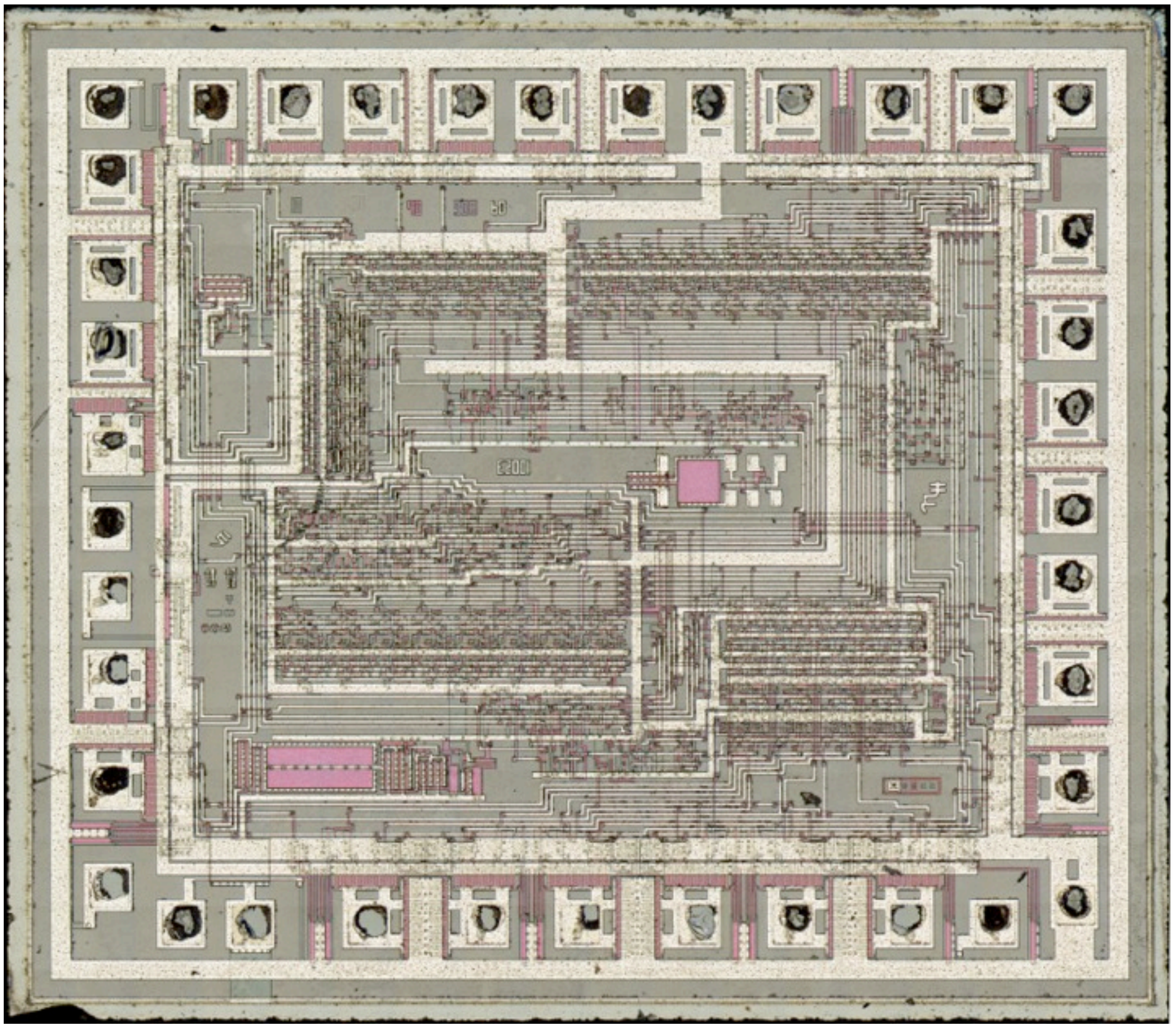
Device	Temperature Range	Package
MC1408PB	0 to +75°C	Plastic
MC1408LB		Ceramic
MC1508LB	–55 to +1255°C	Ceramic

6

Saturday, June 22, 13

The MC1408 is an eight-bit digital-to-analog converter. This must be the audio output, and the computer on the sound board must be an eight-bit digital audio synthesizer!

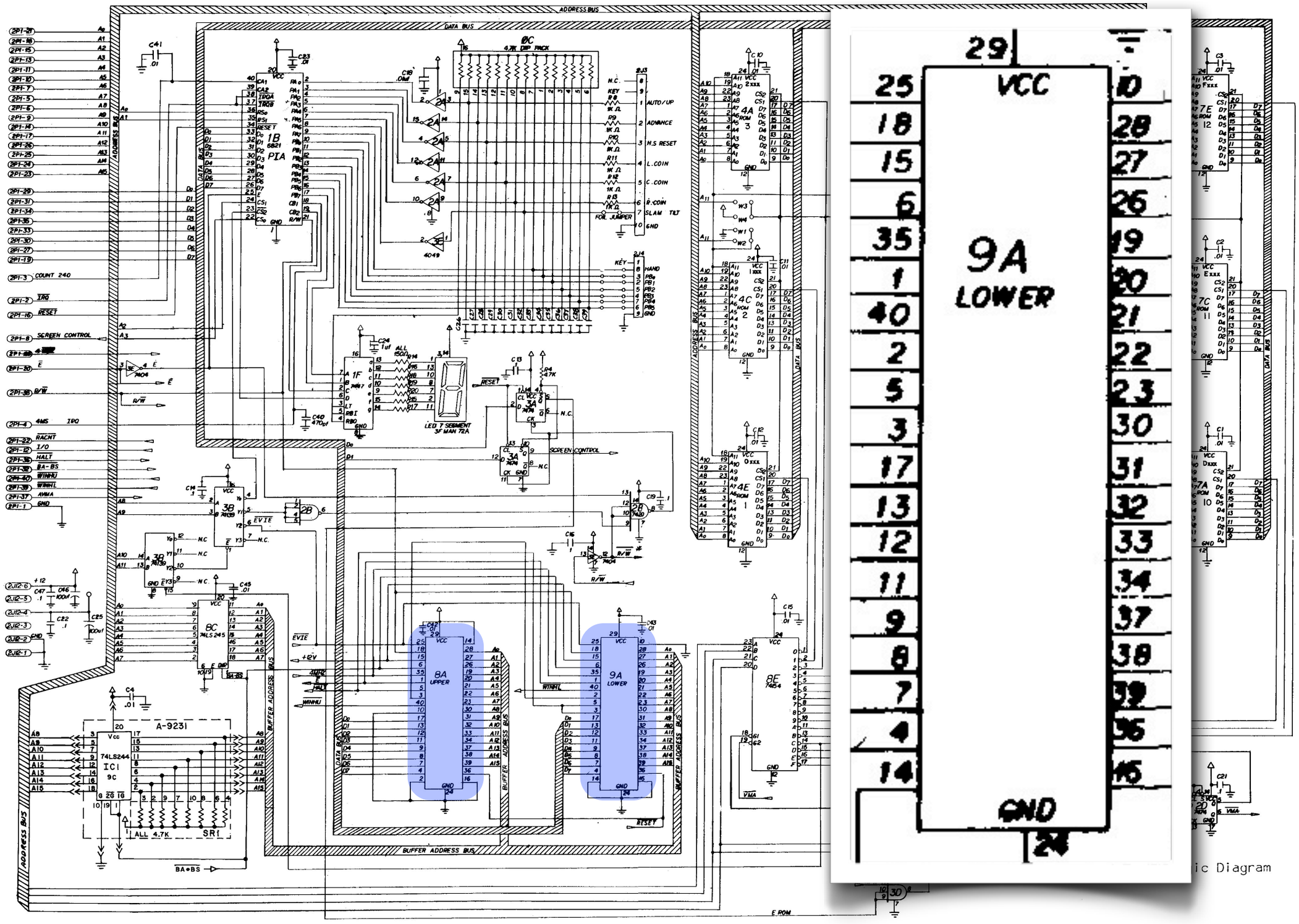
Graphics - Special Chip 1



Saturday, June 22, 13

This is the Williams Special Chip 1 -- worthy of a talk all by itself, so I'm going to skip it this time. This chip is the 1982 equivalent of a GPU. It was designed to move pixels around in memory without the involvement of the main processor. Since the 6809E processor can only execute a few hundred thousand instructions per second (it runs at only 1 MHz), it needs help moving pixels fast enough to implement the game.

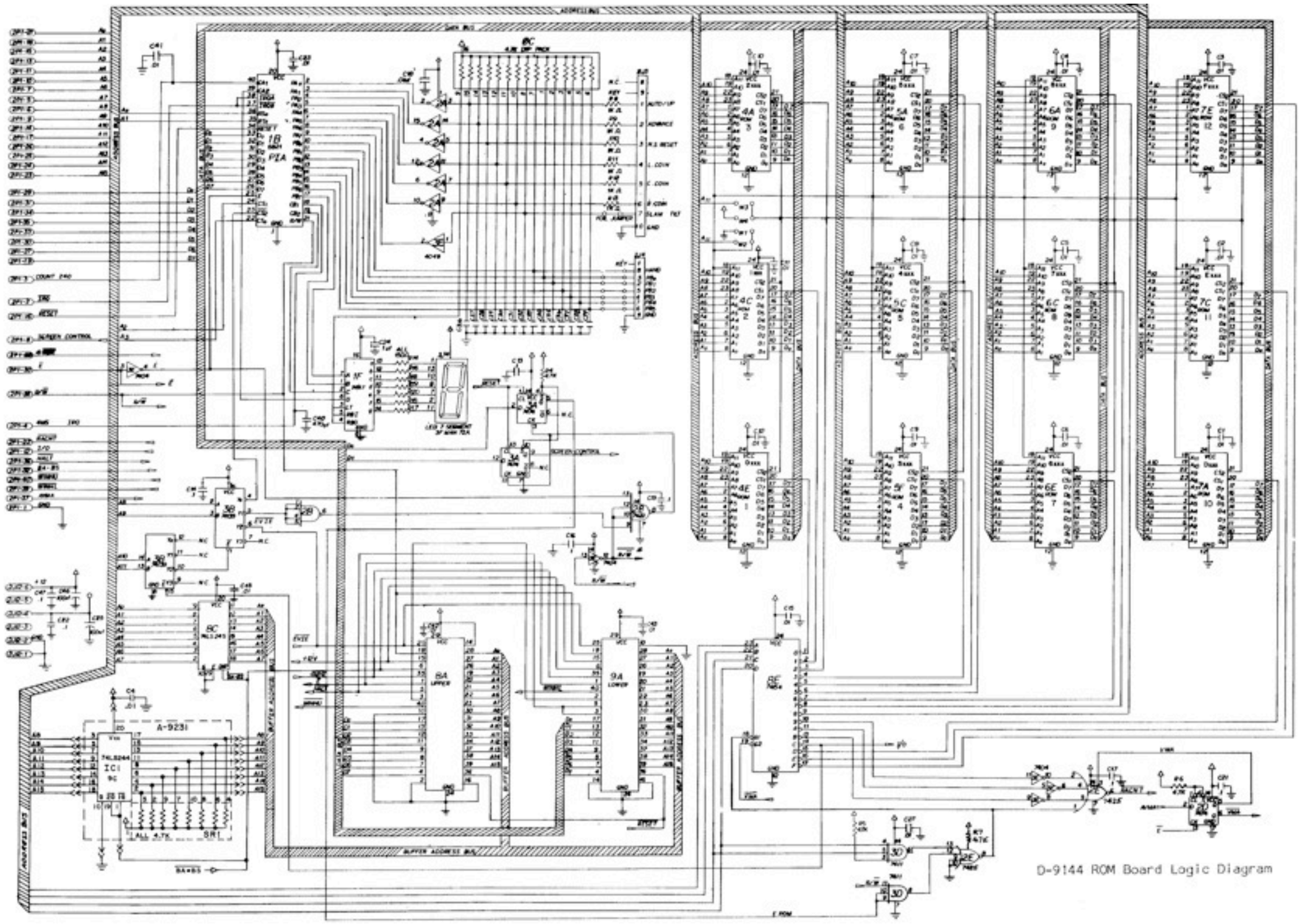
Graphics - Special Chip 1



Saturday, June 22, 13

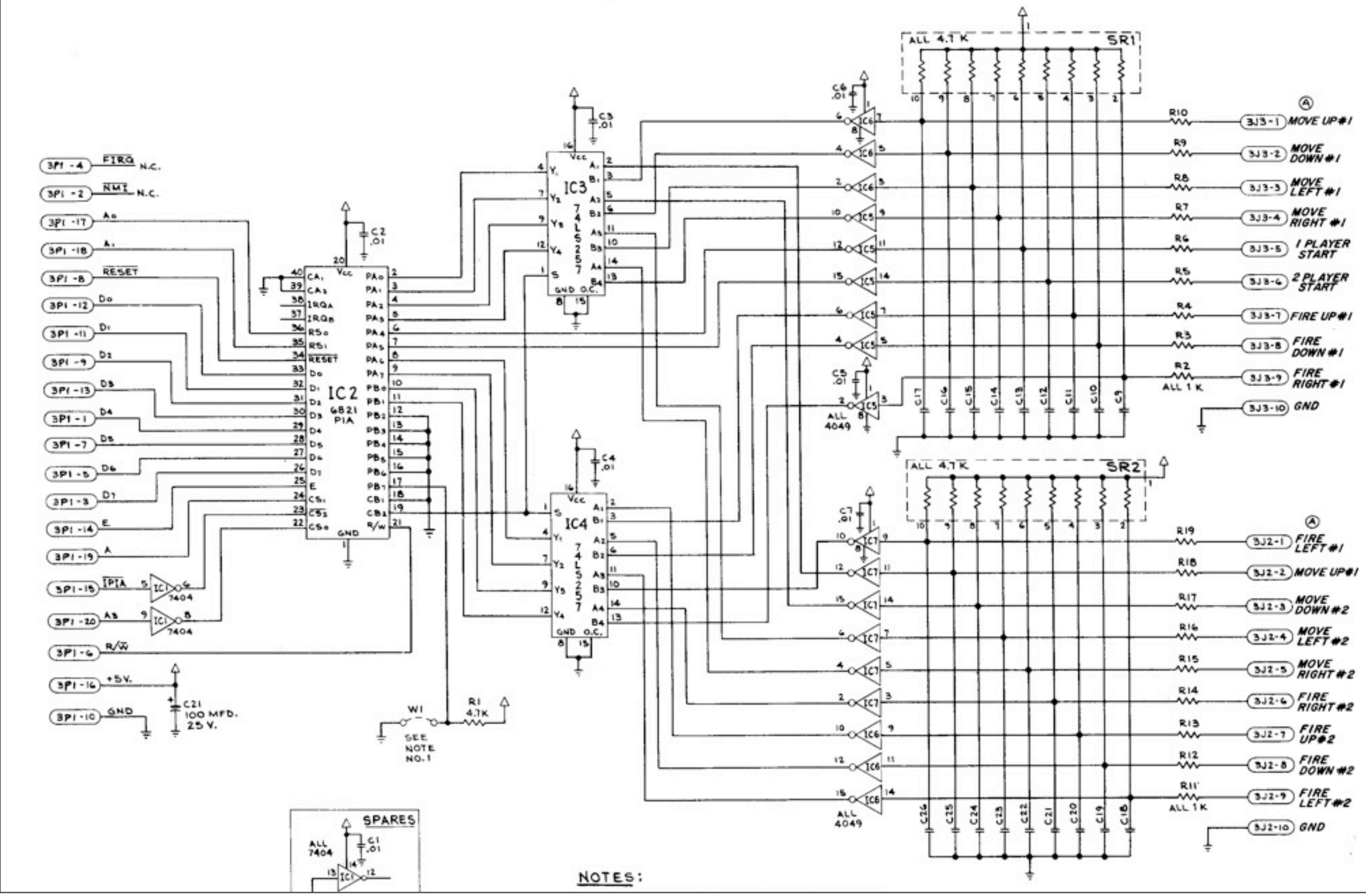
A pair of these chips are on the ROM board of each Robotron machine, and are commanded by the processor to do the bulk of the graphics and animation, including copying bitmap font images from ROM to video memory.

ROM Board



I'm going to skip the rest of the ROM board, since there's not too much more interesting on it, just a ROM bank, and the interface to the coin acceptors and sound board.

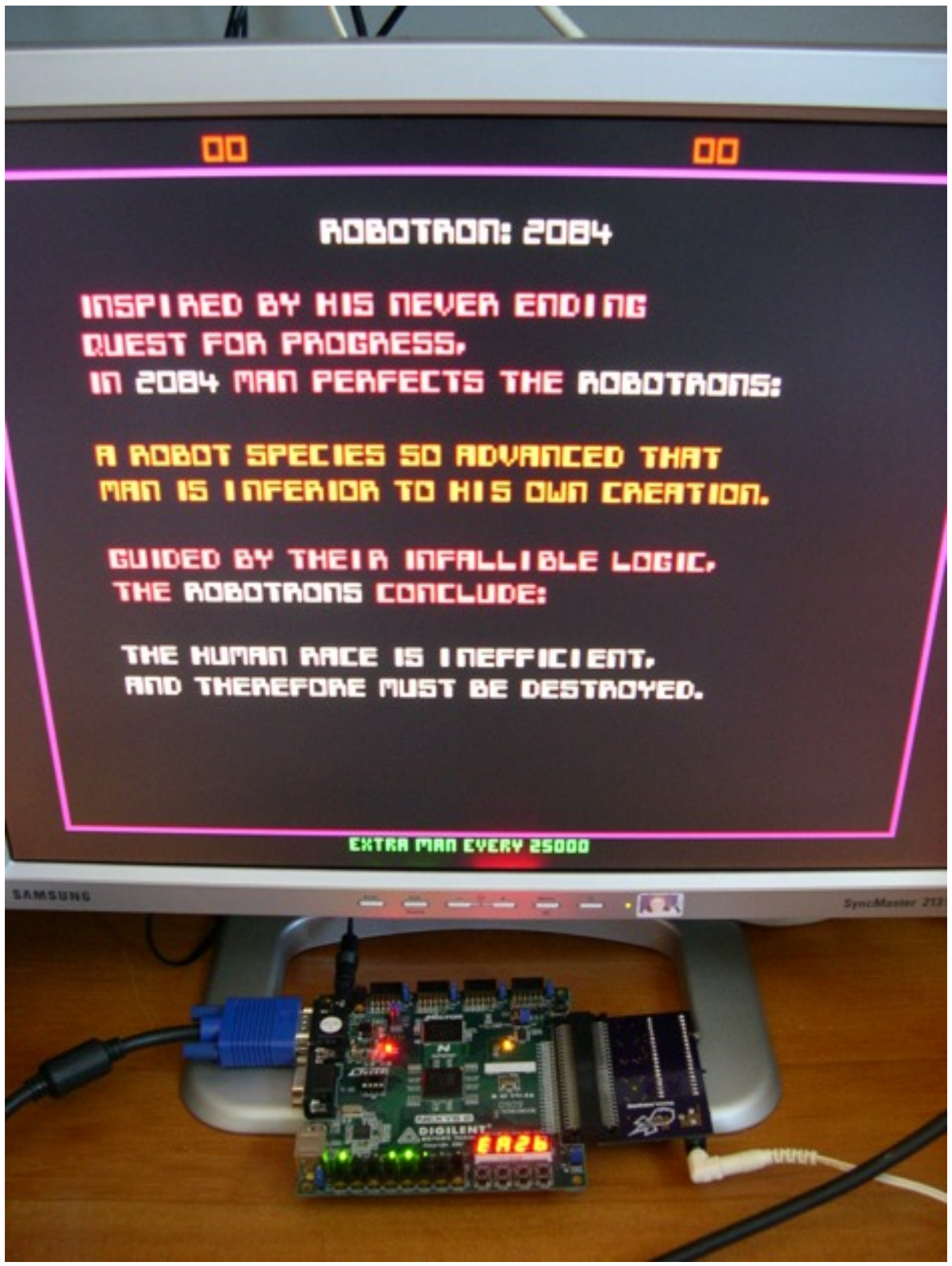
Interface (Widget) Board



Saturday, June 22, 13

I'm also going to skip the interface board. AGain, not much of note, just an interface to the joystick and player buttons.

What's the Point?

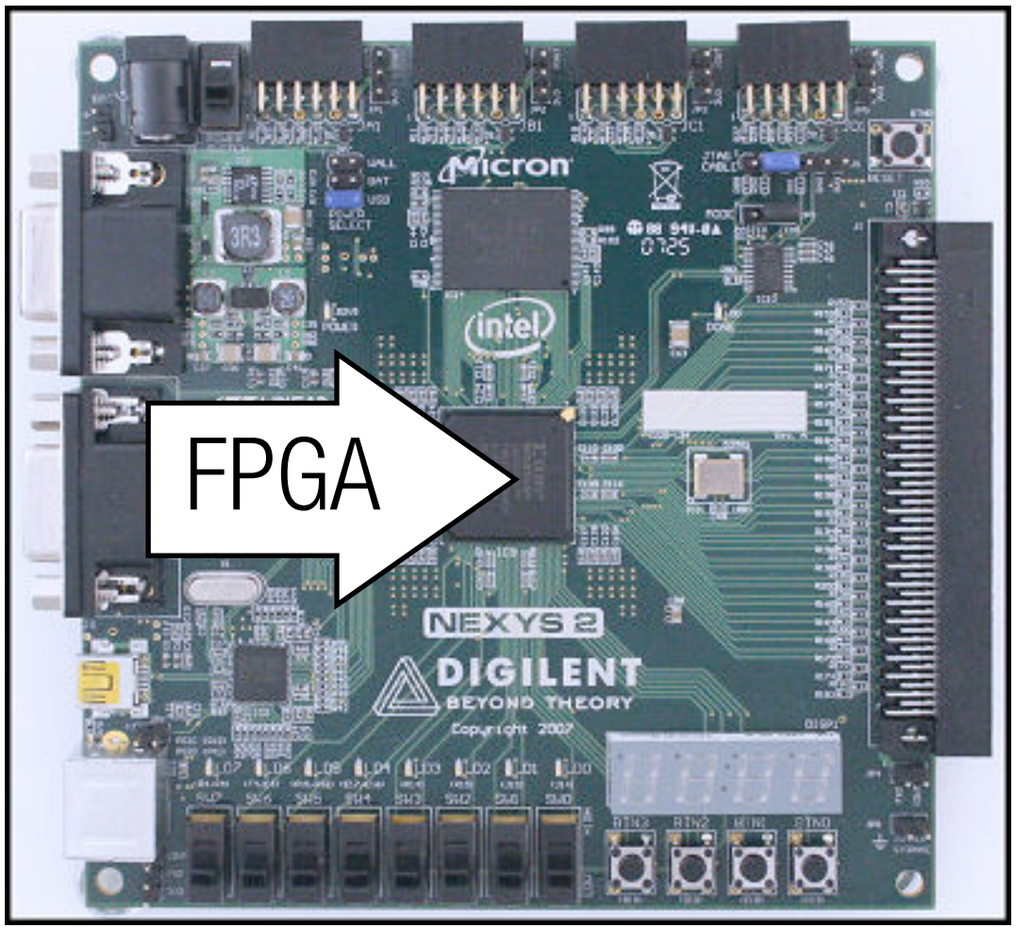


So besides having fodder for this talk, why did I reverse-engineer Robotron? I wanted to recreate the game. But not as software...

Software Isn't Quite Like Hardware

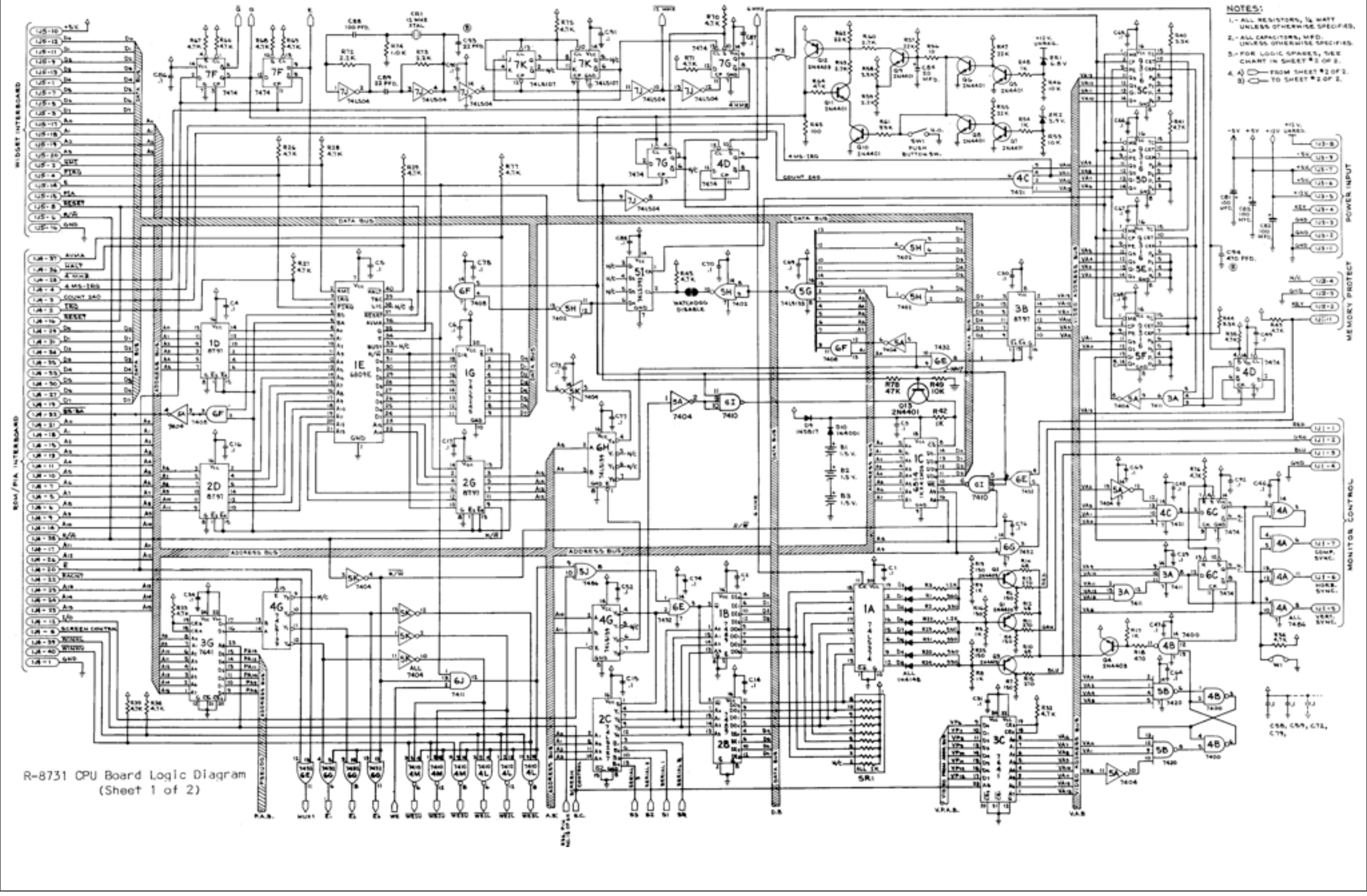
...that's already been done by software like MAME, the multi-arcade machine emulator. But it's not a very literal simulation. It doesn't try too hard to act like the actual hardware.

FPGAs - Stem Cells for Silicon



I wanted to recreate the game using the schematics, and a real 6809E processor. To do this, I used a “field programmable gate array” or FPGA chip, which I like to describe as “stem cells for silicon”. An FPGA is like any other chip, except it’s not wired to do anything in particular. Instead, you load your own wiring into a matrix on the chip that connects the chip’s logic the way you want it. An FPGA could act like a microprocessor, a video card, a BitCoin miner, a music synthesizer, a high-speed data decryptor, a RADAR data processing system... or simulate video game hardware...

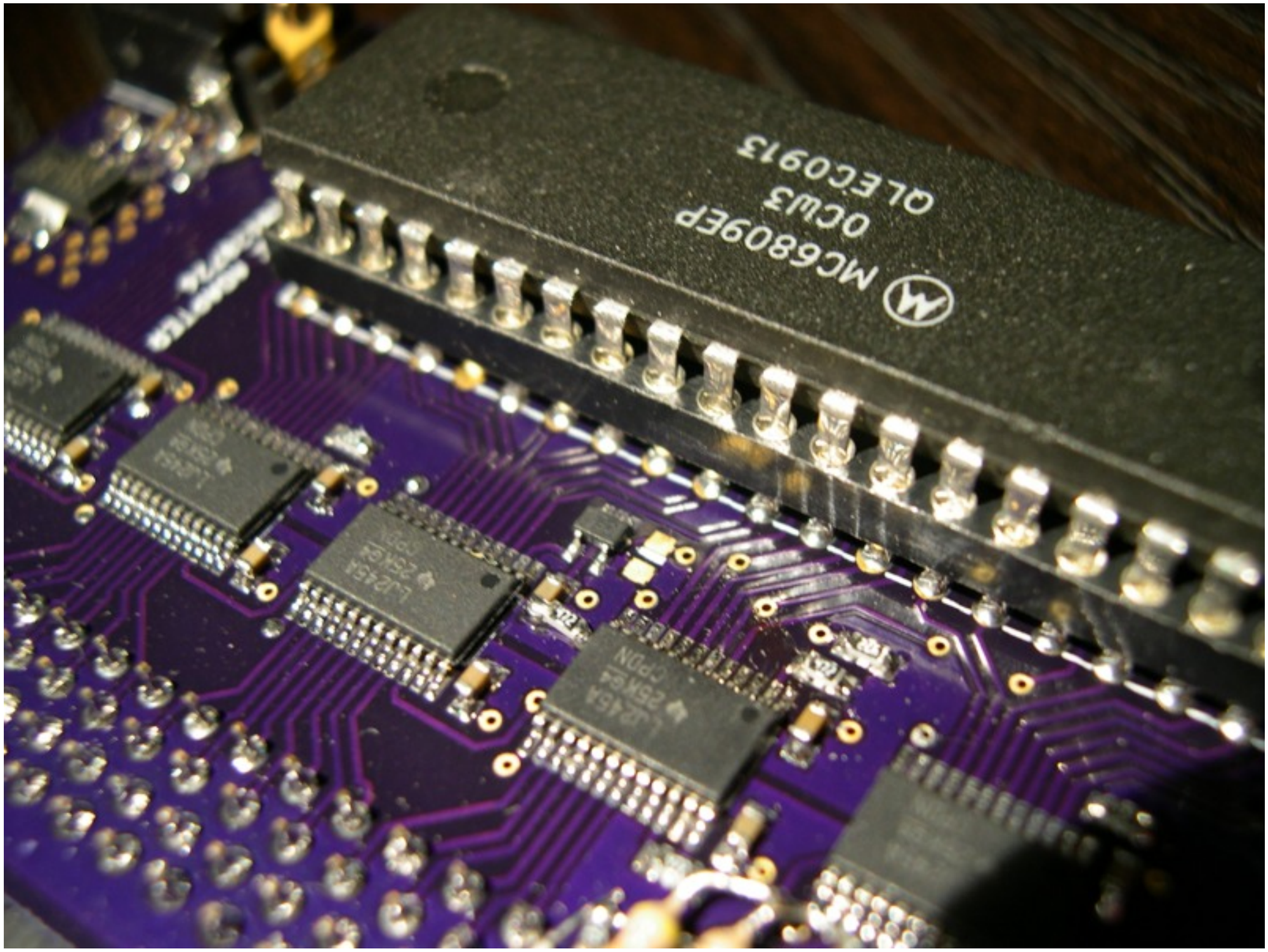
Logic == Logic



Saturday, June 22, 13

An FPGA isn't executing code, it's acting exactly like the chips on the schematic. And therefore, it has the potential to be a more accurate recreation of the game.

Cop-Out, or Retro Reverence?



Saturday, June 22, 13

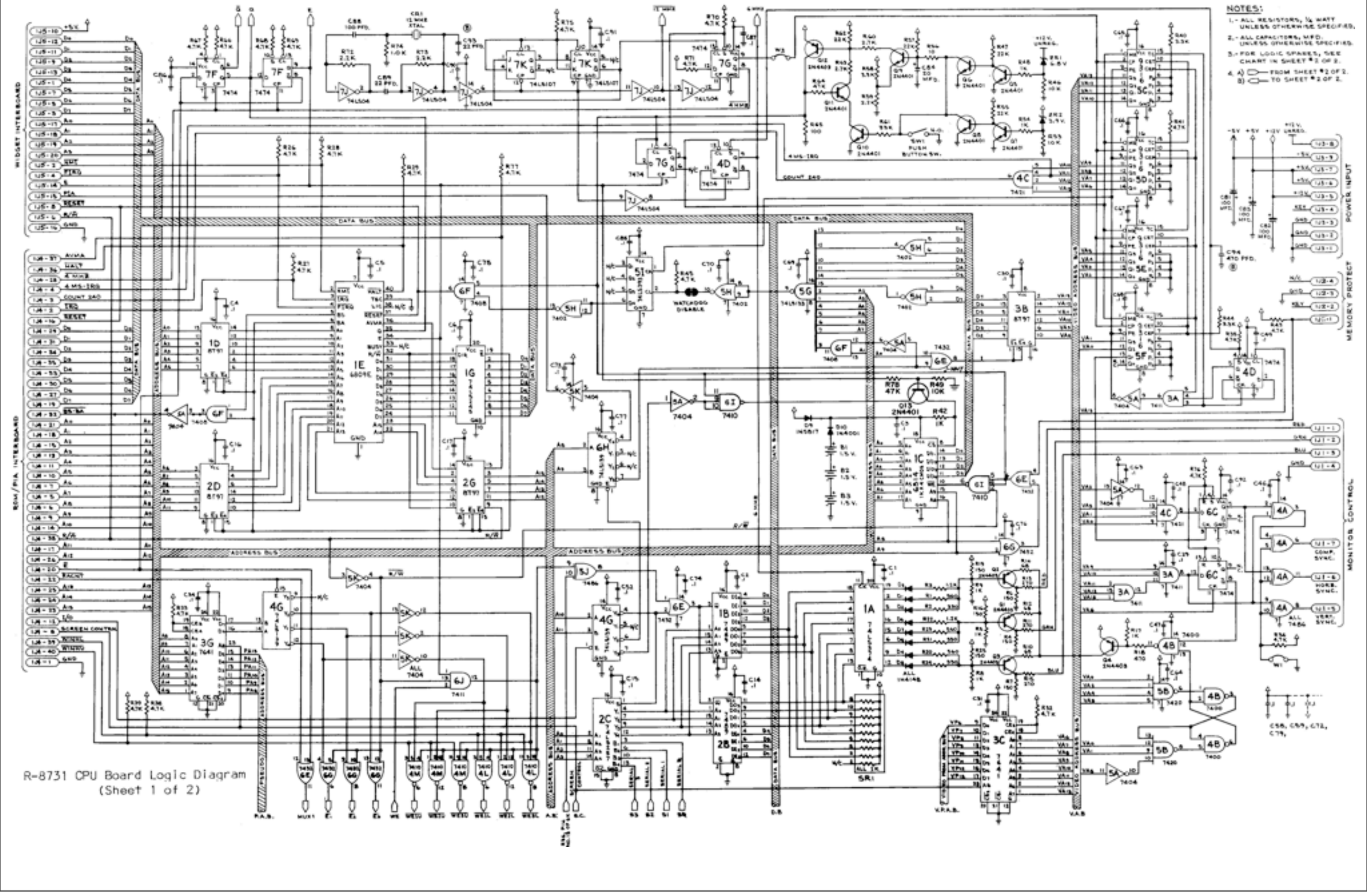
I did cop out on one thing, though. I didn't want to try and recreate the MC6809E microprocessor in the FPGA. It was too much of a challenge for my FPGA skills. So I created an interface board so I could plug a real 6809 into my FPGA board and avoid that work. In retrospect, I kinda like that there's still an original chip in my project. It makes it even *more* real...

Hardware Description Language

```
process(clock)
begin
    if rising_edge(clock) then
        if reset_request = '1' then
            reset_counter <= (others => '0');
            reset <= '1';
        else
            if reset_counter < 100 then
                reset_counter <= reset_counter + 1;
            else
                reset <= '0';
            end if;
        end if;
    end if;
end process;
```


So how do you describe to an FPGA how you want it to act? With a hardware description language or HDL. There are two major languages -- VHDL and Verilog. It looks a lot like software, but it's not. In HDL, the code you write turns into actual bits of hardware -- circuitry that performs exactly the tasks you describe. And all these circuits operate simultaneously, which means FPGAs are great for doing massively parallel processing tasks.

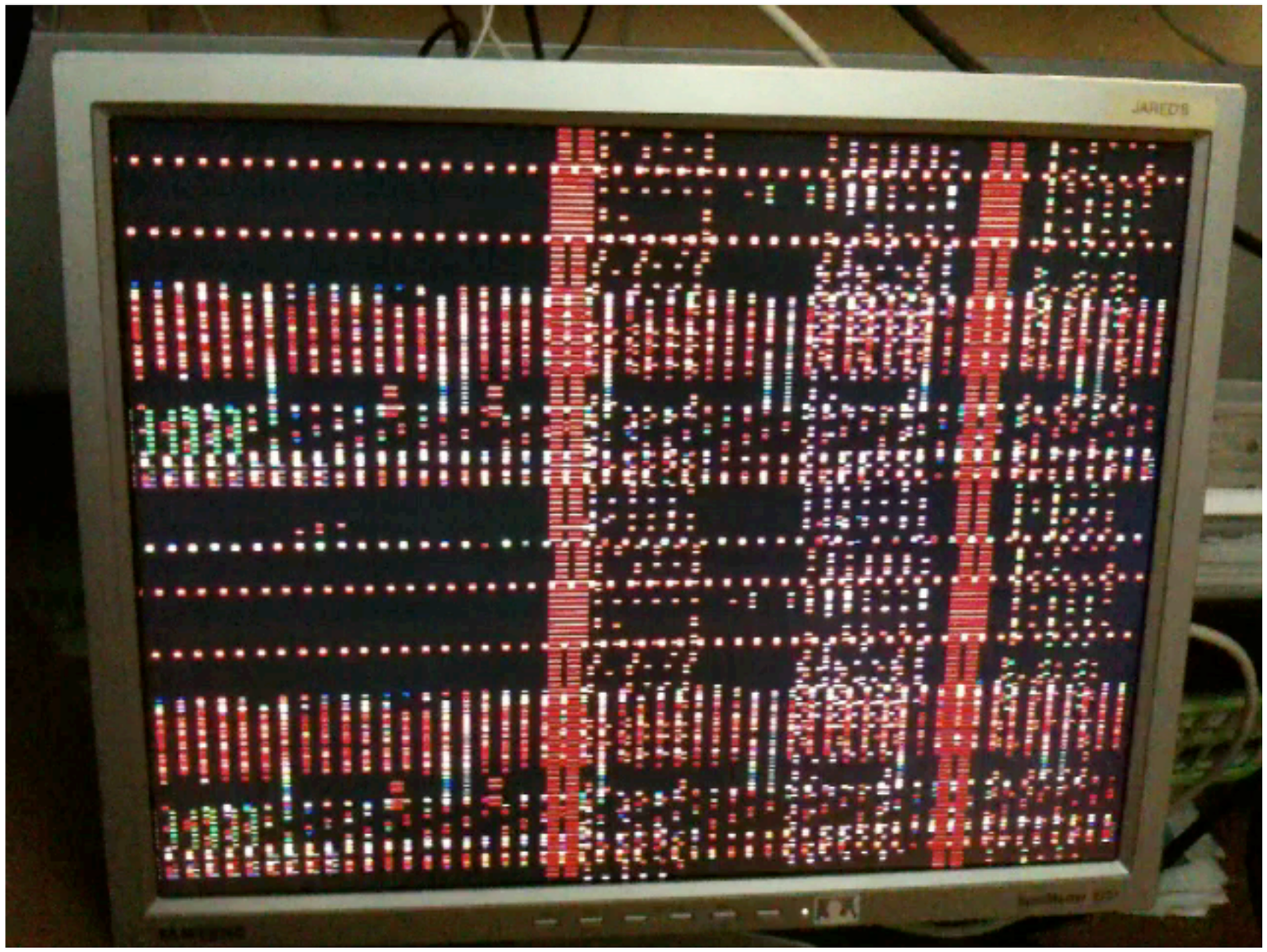
Implementation from Schematic



Saturday, June 22, 13

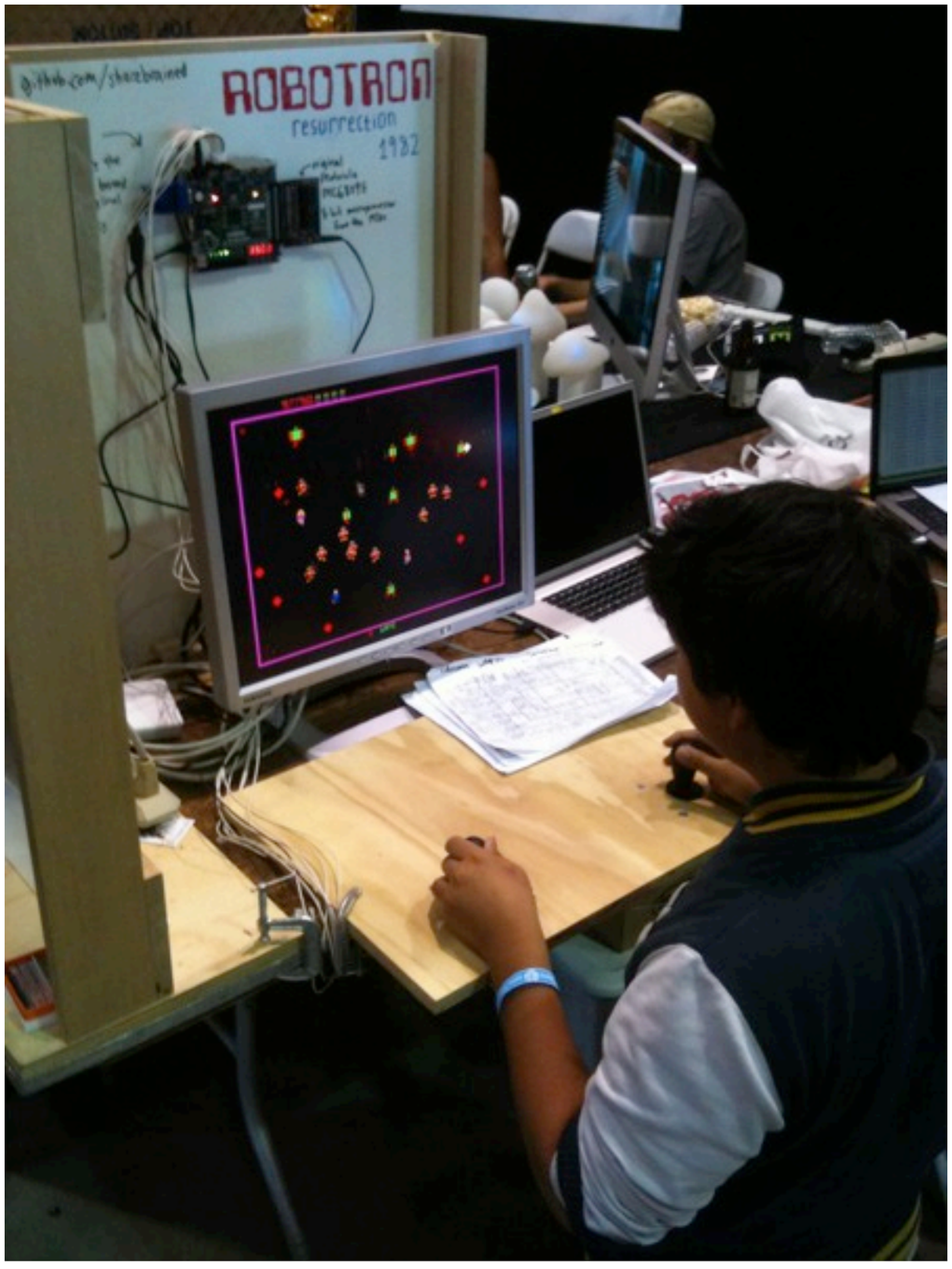
As I studied the schematic, I implemented the circuitry I found by describing it in HDL code.

One of Many Misunderstandings...



There were a few mis-steps along the way. The most visible were when I was trying to understand how the video hardware worked.

But Eventually...



But eventually, I figured it out and got a playable machine!

All my HDL is up on GitHub, if you're at all interested in how it all works...

Links

Incredible reverse-engineering of Williams arcade machines

seanriddle.com/willy.html

My FPGA-based Robotron hardware implementation

github.com/sharebrained/robotron-fpga

All Must Be Tested!!!

churchofrobotron.com