SIMPLE GEMINI SPACECRAFT COMPUTER

revision 2.0

John Pultorak
January 2010

Abstract

This describes the architecture and serial logic design of the Simple Gemini Spacecraft Computer (SGSC).

It is based upon "NASA Project Gemini Familiarization Manual, Rendezvous and Docking Configurations" SEDR 300, Vol. 2, suppl, July 1, 1966, publicly available from NASA CASI/STI (79N76135) and downloadable from various internet sites.

I interpolated some architectural details, simplified some features, and developed my own logic design.

Simple Gemini Computer Facts

serial computation and data transfer

11 instructions

13-bit instruction word

26-bit data word

4096 words of random access memory

instruction cycle 140 uSec

all instructions: 1 cycle

arithmetic bit rate: 500 KHz

Why 'Simple'?

I eliminated a few features from the Gemini computer architecture to make the computer easier and cheaper to build:

- I removed one syllable from the memory word. The Gemini had a 39-bit memory word composed of three 13-bit syllables. I changed it to a 26-bit memory word with two 13-bit syllables. The 3rd syllable was deleted from the Gemini's successor: the Saturn V LVDC.

- I eliminated the fractional multiply (MPY), divide (DIV), and store product/quotient (SPQ) instructions. These were implemented in a hardware multiply/divide “element” of discouraging complexity. It's more fun to program multiply/divide functions in software, anyway.

- I eliminated the hardware subtraction (SUB) and reverse subtraction (RSU) instructions. You can use the ADD instruction with negative numbers to subtract.

- I simplified the I/O.

- I simplified the timing. Since I use shift registers instead of delay lines, I don't need timing (gate) signals for every bit position in a 13-bit word to serialize data.

- I somewhat simplified operand fields for the SHF, CLD, and PRO instructions.

If you like, you may restore the missing elements. In particular, you'll find a description of the multiply/divide element in “Laboratory Maintenance Instructions, Saturn V Launch Vehicle Digital Computer (LVDC) 1965 (available on the web).

Instruction Set

Opcodes are octal.

HOP 00 Fields in the 26-bit memory word referenced by the operand (see HOP word) are used to change the next instruction address.

```
MSB        LSB      MSB                          LSB
13  12  11  10  09  08  07  06  05  04  03  02  01
 0   0   0   0  A9  A8  A7  A6  A5  A4  A3  A2  A1
```

DIV 01 **Not implemented.**

PRO 02 Process input or output. The input or output specified by the operand address is read into, or written from, the accumulator. For output, the accumulator is cleared if operand A9=1.

The A9 bit specifies whether ACC contents are recirculated during the PRO operation:

If A9=0 during PRO, each shifted-out accumulator bit is recirculated back to the accumulator input through a 2-input OR gate. The other OR gate input is driven by the input channel for that address so the accumulator contents are OR'ed with new input data. If the input isn't connected, it's OR-gate input assumes a zero state so the accumulator just recirculates it's current value.

If A9=1 during PRO, shifted-out bits are not recirculated so incoming data just replaces the outgoing data. If the input is not connected, zeros are clocked in as the original contents are clocked out, causing the accumulator to be cleared.

```
MSB        LSB      MSB                          LSB
13  12  11  10  09  08  07  06  05  04  03  02  01
 0   0   1   0  A9  --  --  --  --  --  A3  A2  A1

                                        A3  A2  A1
DO01  Reset DI01, DI02, DI03, DI04      0   0   1
DO02  Write MDIU Digit select           0   1   0
DO03  Write MDIU Digit magnitude        0   1   1
DI05  Read MDIU insert data             1   0   1
DI06  Read TIME code                    1   1   0

Note: DI01-DI04 are the "Data Ready", "Enter", "Readout",
and "Clear" flags set by the keyboard entry or depressing
three MDIU buttons bearing those names.
See the CLD instruction.
```

RSU 03 **Not implemented.**

ADD 04 Add. The contents of the memory location referenced by the operand is added to the contents of the accumulator. The result is placed in the accumulator.

A9 is the "residual" bit.

```
A9=1  Use memory sector 17 (the "residual" sector)
A9=0  Use sector register to reference memory.
```

```
MSB        LSB      MSB                          LSB
13  12  11  10  09  08  07  06  05  04  03  02  01
 0   1   0   0  A9  A8  A7  A6  A5  A4  A3  A2  A1
```

SUB 05 **Not implemented.**

CLA 06 Clear and add. The accumulator is loaded with the contents of the memory location referenced by the operand.

A9 is the "residual" bit.

```
A9=1  Use memory sector 17 (the "residual" sector)
A9=0  Use sector register to reference memory.
```

```
MSB        LSB      MSB                          LSB
13  12  11  10  09  08  07  06  05  04  03  02  01
 0   1   1   0  A9  A8  A7  A6  A5  A4  A3  A2  A1
```

AND 07 Bitwise AND. The contents of the memory location referenced by the operand are logically ANDed, bit-by-bit, with the accumulator. The result is placed in the accumulator.

A9 is the "residual" bit.

```
A9=1  Use memory sector 17 (the "residual" sector)
A9=0  Use sector register to reference memory.
```

```
MSB        LSB      MSB                          LSB
13  12  11  10  09  08  07  06  05  04  03  02  01
 0   1   1   1  A9  A8  A7  A6  A5  A4  A3  A2  A1
```

MPY 10 **Not implemented.**

TRA 11 Transfer. Transfers execution to the address specified in the operand field. Syllable and sector are unchanged. A9 bit is ignored.

```
MSB        LSB      MSB                          LSB
13  12  11  10  09  08  07  06  05  04  03  02  01
 1   0   0   1  --  A8  A7  A6  A5  A4  A3  A2  A1
```

SHF 12 Shift. Shifts the contents of the accumulator left or right one or two places, as specified by the operand. An invalid code in the operand clears the accumulator.

Left-shift enters zeroes in low order positions. Right-shift copies the sign bit in high-order positions.

Valid codes:

```
                              A6          A5-A4
        shift left 1 bit      0           3
        shift left 2 bits     0           2
        shift right 1 bit     0           1
        shift right 2 bits    0           0
        zero accumulator      1           x
```

```
        MSB        LSB      MSB                          LSB
        13  12  11  10  09  08  07  06  05  04  03  02  01
         1   0   1   0  --  --  --  A6  A5  A4  --  --  --
         1   0   1   0  --  --  --   0   1   1  --  --  -- (SL 1 bit)
         1   0   1   0  --  --  --   0   1   0  --  --  -- (SL 2 bits)
         1   0   1   0  --  --  --   0   0   1  --  --  -- (SR 1 bit)
         1   0   1   0  --  --  --   0   0   0  --  --  -- (SR 2 bits)
         1   0   1   0  --  --  --   1  --  --  --  --  -- (zero acc)
```

Example:

```
26  25  24  23  22  21      06  05  04  03  02  01   (bit position)
S   M1  M2  M3  M4  M5  ... M20 M21 M22 M23 M24 M25  (initial)
M1  M2  M3  M4  M5  ... M20 M21 M22 M23 M24 M25 0    (SL 1 bit)
M2  M3  M4  M5  ... M20 M21 M22 M23 M24 M25 0   0    (SL 2 bits)
S   S   M1  M2  M3  M4  M5  ... M20 M21 M22 M23 M24  (SR 1 bit)
S   S   S   M1  M2  M3  M4  M5  ... M20 M21 M22 M23  (SR 2 bits)
```

TMI 13 Transfer on minus accumulator sign. If the sign bit in the accumulator is negative, execution transfers to the address specified by the operand. Syllable and sector are unchanged. A9 bit is ignored.

```
        MSB        LSB      MSB                          LSB
        13  12  11  10  09  08  07  06  05  04  03  02  01
         1   0   1   1  --  A8  A7  A6  A5  A4  A3  A2  A1
```

STO 14 Store. The accumulator is stored in the memory location referenced by the operand. The accumulator is unchanged.

A9 is the "residual" bit.

```
A9=1  Use memory sector 17 (the "residual" sector)
A9=0  Use sector register to reference memory
```

```
        MSB        LSB      MSB                          LSB
        13  12  11  10  09  08  07  06  05  04  03  02  01
         1   1   0   0  A9  A8  A7  A6  A5  A4  A3  A2  A1
```

SPQ 15 **Not implemented.**

CLD 16 Clear and add discrete. The discrete input selected by the operand is read into all accumulator bit positions.

The DATA READY flag is set by a 0-9 keypress from the MDIU. Use PRO 5 to read the MDIU digit. Then clear the DATA READY flag with PRO 1.

MSB			LSB	MSB								LSB
13	12	11	10	09	08	07	06	05	04	03	02	01
1	**1**	**1**	**0**	**--**	**--**	**--**	**--**	**--**	**--**	**A3**	**A2**	**A1**

		A3	A2	A1
DI01	Read DATA READY flag	0	0	1
DI02	Read ENTER key press	0	1	0
DI03	Read READ OUT key press	0	1	1
DI04	Read CLEAR key press	1	0	0
DI07	Compliment Accumulator	1	1	1

TNZ 17 Transfer on non-zero. If the contents of the accumulator are non-zero, execution transfers to the address specified by the operand. Syllable and sector are unchanged. A9 bit is ignored.

MSB			LSB	MSB								LSB
13	12	11	10	09	08	07	06	05	04	03	02	01
1	**1**	**1**	**1**	**--**	**A8**	**A7**	**A6**	**A5**	**A4**	**A3**	**A2**	**A1**

MDIU

DESCRIPTION

The MDIU is the pilot interface to the computer. It consists of Manual Data Keyboard and a Manual Data Readout units.

The Manual Data Keyboard is a keypad with 10 buttons:

```
ZERO, 1,2,3,4,5,6,7,8,and 9
```

The Manual Data Readout has 3 buttons:

```
READOUT, CLEAR, ENTER
```

The Manual Data Readout also has a 7 digit decimal display. The first 2 digits show an address, and the last 5 digits show data.

The pilot can enter or display data for up to 99 addresses. Each address identifies a type of data. A display of zero address and data indicates pilot error.

MDIU INTERFACES (CLD)

The following are accessed by the CLD instruction.

```
DI01  Data Ready  A digit (0-9) key has been pressed.
DI02  Enter       The ENTER key was pressed.
DI03  Readout     The READ OUT key was pressed.
DI04  Clear       The CLEAR key was pressed.
```

MDIU INTERFACES (PRO)

The following are accessed by the PRO instruction.

```
DO01  Reset DI01, DI02, DI03, DI04.
DO02  Digit select.
DO03  Digit magnitude.
DI05  Read MDIU insert data.
DI06  Read TIME code.
DI07  Compliment Accumulator.
```

Memory

Memory is 4096 words. Each word is 26 bits, consisting of two 13-bit syllables.

```
     -------------- SYLLABLE --------------
      13 12 11 10 09 08 07 06 05 04 03 02 01    (bit position)

      --------------- WORD -----------------
         SYLLABLE 1             SYLLABLE 0
      26    ...     14      13    ...     01    (bit position)
```

```
          ----- 26 bit word -----
          26      14   13      01
sector    SYLLABLE 0   SYLLABLE 1          8-bit addr      12-bit address
0               (256 words)                 000 - 377       0000 - 0377
1               (256 words)                 000 - 377       0400 - 0777
2               (256 words)                 000 - 377       1000 - 1377
3               (256 words)                 000 - 377       1400 - 1777
4               (256 words)                 000 - 377       2000 - 2377
5               (256 words)                 000 - 377       2400 - 2777
6               (256 words)                 000 - 377       3000 - 3377
7               (256 words)                 000 - 377       3400 - 3777
10              (256 words)                 000 - 377       4000 - 4377
11              (256 words)                 000 - 377       4400 - 4777
12              (256 words)                 000 - 377       5000 - 5377
13              (256 words)                 000 - 377       5400 - 5777
14              (256 words)                 000 - 377       6000 - 6377
15              (256 words)                 000 - 377       6400 - 6777
16              (256 words)                 000 - 377       7000 - 7377
17       (256 words; "residual sector")     000 - 377       7400 - 7777
```

The memory is RAM. It is initialized from EPROM at startup.

During power up initialization, the entire contents of the EPROM is copied to RAM by a hardware “bootstrap” circuit. The EPROM is then disabled and the computer runs entirely from RAM.

The 4 EPROM are located on the Memory Board (U34-U37). They are 27C128 16K x 8 bit EPROMs. Only the lowest 4K is used in each EPROM. The table below shows how bits of the 26-bit word are assigned to the EPROMs.

```
IC    D7 D6 D5 D4 D3 D2 D1 D0
U34    x  x  x  x  x  x 26 25
U35   24 23 22 21 20 19 18 17
U36   16 15 14 13 12 11 10 09
U37   08 07 06 05 04 03 02 01
```

Word Representation

Instruction word

Instruction words are 1 syllable long. Instruction words can be read from any syllable of memory.

```
      MSB        LSB      MSB                          LSB
      13  12  11  10  09  08  07  06  05  04  03  02  01   (bit position)
      OP4 OP3 OP2 OP1 A9  A8  A7  A6  A5  A4  A3  A2  A1   (fields)

A8-A1 is the operand address. The low-order bit is A1.

A9 is the "residual" bit.
      A9=1  Use memory sector 17 (the "residual" sector)
      A9=0  Use sector register to reference memory.

OP4-OP1 is the opcode field.  The low-order bit is OP1.
```

Data word

Data words are 2 syllables long. Numbers are 2's compliment, with 25 magnitude bits and 1 sign bit. The low-order bit is M25. The high-order bit is M1. 'S' is the sign.

Data words are read from syllables 0 and 1 of memory.

```
          MSB                                                  (SYLLABLE 1)
      26  25  24  23  22  21  20  19  18  17  16  15  14   (bit position)
      S   M1  M2  M3  M4  M5  M6  M7  M8  M9  M10 M11 M12  (bit code)

                                                   LSB   (SYLLABLE 0)
      13  12  11  10  09  08  07  06  05  04  03  02  01   (bit position)
      M13 M14 M15 M16 M17 M18 M19 M20 M21 M22 M23 M24 M25  (bit code)
```

HOP word

The HOP word transfers execution to a different sector in memory.

```
                                                           (SYLLABLE 1)
      26  25  24  23  22  21  20  19  18  17  16  15  14   (bit position)
      --  --  --  --  --  --  --  --  --  --  --  SYL --   (bit code)

      MSB         LSB      MSB                      LSB   (SYLLABLE 0)
      13  12  11  10  09  08  07  06  05  04  03  02  01   (bit position)
      S4  S3  S2  S1  A9  A8  A7  A6  A5  A4  A3  A2  A1   (bit code)

A8-A1 identifies the address of the next instruction to execute in the new
sector.

A9 is the "residual" bit.
      A9=1  Use memory sector 17 (the "residual" sector) to fetch
      instructions. Note that the address of the operand of each instruction
      will still be controlled by the sector register or A9 bit for that
      instruction. Therefore, instructions fetched from the residual sector
      will reference operands at the sector defined by the sector register
      or operands in the residual sector if the instruction's A9 bit is set.
```

S4-S1 loads the sector register.

SYL selects the new syllable.
SYL=0 Use syllable 0.
SYL=1 Use syllable 1.

Registers

OPR Operation Register (4 bit)

```
01  02  03  04   (bit position)
OP1 OP2 OP3 OP4
LSB         MSB

Control Inputs:
SLOPR       Shift OPR left.  Takes effect at CLK2.
```

IRES Instruction Residual Bit Register (1 bit)

```
A9 from the HOP word feeds the instruction "residual" bit.
A9=0  Use sector bits S1-S4 to select the new sector.
A9=1  Use memory sector 17 (the "residual" sector) for the new sector.

Control Inputs:
WIRES       Write contents of MB bit 09 into IRES (parallel load).
            Takes effect at CLK2. Taken from A9 of the HOP word.
```

ORES Operand Residual Bit Register (1 bit)

```
A9 from the instruction word feeds the operand "residual" bit.
0     Use sector bits S1-S4 to select the new sector.
1     Use memory sector 17 (the "residual" sector) for the new sector.

Control Inputs:
SLORES      Shift ORES left.  Takes effect at CLK2.
```

SCR Sector Register (4 bit)

```
01  02  03  04   (bit position)
S1  S2  S3  S4
LSB         MSB

Control Inputs:
WSCR        write the contents of MB bit 10-13 into SCR (parallel load).
            Takes effect at CLK2. Taken from S1-S4 of the HOP word.
```

MAR Memory Address Register (8 bit)

```
01  02  03  04  05  06  07  08   (bit position)
A1  A2  A3  A4  A5  A6  A7  A8
LSB                     MSB

A1-A8 identifies a memory address inside a sector.

Control Inputs:
SLMAR       Shift MAR left.  Takes effect at CLK2.
```

SYR Syllable Register (1 bit)

SYR=0 Use syllable 0.
SYR=1 Use syllable 1.

Control Inputs:
WSYR write contents of MB bit 15 into SYR (parallel load).
Takes effect at CLK2. Data will be field SYL of HOP word.

MB Memory Buffer (26 bit)

Reads/writes both syllables of a memory word to/from 2 13-bit registers:
MB0 holds syllable 0 (bits 01-13)
MB1 holds syllable 1 (bits 14-26)

Control Inputs:
WMB write contents of the currently addressed memory word into
MB0 and MB1 (parallel load). Takes effect at CLK2.

CLMB Clear MB0 and MB1.

SLMB Shift MB0 and MB1 left. Takes effect at CLK2.

WMEM Write contents of MB0 and MB1 to memory syllables 0 and 1.

IAR Instruction Address Register (8 bit)

01	02	03	04	05	06	07	08	(bit position)
A1	**A2**	**A3**	**A4**	**A5**	**A6**	**A7**	**A8**	
LSB							**MSB**	

A1-A8 identifies the "word position" of the next instruction to execute in the new sector.

Control Inputs:
SLIAR Shift IAR left. Takes effect at CLK2.

ACC Accumulator (26 bit)

01	02	03	04	05	06	07	08	09	10	11	12	13	(bit position)
M25	**M24**	**M23**	**M22**	**M21**	**M20**	**M19**	**M18**	**M17**	**M16**	**M15**	**M14**	**M13**	**(bit code)**
LSB													

14	15	16	17	18	19	20	21	22	23	24	25	26	(bit position)
M12	**M11**	**M10**	**M9**	**M8**	**M7**	**M6**	**M5**	**M4**	**M3**	**M2**	**M1**	**S**	**(bit code)**
											MSB		

Least significant bit (M25) comes first because arithmetic operations are serial beginning with low order bits.

Control Inputs:
SLACC Shift ACC left. Takes effect at CLK2.
WACCSL1 Load 0 into ACC[01]. Takes effect at CLK2.
WACCSL2 Load 0s into ACC[01-02]. Takes effect at CLK2.

```
WACCSR1     Load sign bit (from MIR) in ACC[25].  Takes effect at CLK2.
WACCSR2     Load sign bits (from MIR) in ACC[24-25].  Effective at CLK2.
```

NZR Non-zero Register (1 bit)

```
Set at the end of Phase B if the Accumulator is non-zero.
0     Accumulator is zeroed.
1     Accumulator is non-zero.

Control Inputs:
CLNZR       Clear NZR.
SETNZR      Set NZR.
```

MIR Minus Sign Register (1 bit)

```
Set at the end of Phase B if sign bit in Accumulator is negative.
0     Accumulator sign bit is positive (0).
1     Accumulator sign bit is negative (1).

Control Inputs:
CLMIR       Clear MIR.
SETMIR      Set MIR.
```

KBDR MDIU Keyboard Register (8 bit)

```
01  02  03  04   (bit position)
A1  A2  A3  A4
LSB       MSB

A1-A4 identifies a BCD keyboard code.

Control Inputs:
WKBDR Write a BCD digit into KBDR (parallel load).
SLKBDR      Shift KBDR left.  Takes effect at CLK2.
```

DSELR MDIU Digit Select Register (4 bit)

```
01  02  03  04   (bit position)
A1  A2  A3  A4
LSB       MSB

A1-A4 identifies selected digit where 0 is rightmost digit.

Control Inputs:
SLDSELR     Shift DSELR left.  Takes effect at CLK2.
```

DMAGR MDIU Digit Magnitude Register (4 bit)

```
01  02  03  04   (bit position)
A1  A2  A3  A4
```

LSB MSB

A1-A4 contains BCD code for digit selected by DSELR.

Control Inputs:
SLDMAGR Shift DMAGR left. Takes effect at CLK2.

TIMR Time Register (4 bit)

01 02 03 04 (bit position)
A1 A2 A3 A4
LSB MSB

A1-A4 contains a BCD digit incremented at 10 Hz.

Control Inputs:
WTIMR Write a BCD digit into TIMR (parallel load).
SLTIMR Shift TIMR left. Takes effect at CLK2.

Timing

The computer instruction cycle is 5 phases long: (PA, PB, PC, PD, PE).

All instructions execute in 1 cycle.

Description of the instruction cycle:

Prior to the start of the instruction cycle: The 26-bit memory word containing the next instruction will be read into MB0 and MB1. The address of the instruction word will be in MAR.

PA Phase A: Shift the 13-bit instruction word from MB into OPR, ORES, and IAR. OPR holds the 4-bit instruction code, ORES holds the operand residual bit, and IAR holds the operand address. The address of the current instruction remains in MAR.

PB Phase B: Shift the 8-bit operand address, currently in IAR, into MAR so we can read the operand word. Simultaneously, shift the address of the current instruction, currently in MAR, into IAR, incrementing it by 1 through the “plus 1 adder”. At the end of the phase, MAR will hold the address of the operand, and IAR will hold the address for the next instruction. At the very end of phase B (bit 14), use MAR to load the memory word containing the operand into MB.

PC/D Phases C and D: Perform the instruction specified in the operation register (OPR).

PE Phase E: Shift the address of the next instruction from IAR to MAR. At the end of phase E (bit 14), use MAR to load the memory word containing the next instruction into MB.

Register Transfers

The register transfers herein are my interpretation of SEDR 300.

```
      KEY:
      []    specifies an address for a memory transfer
      -     specifies a range of bits
      :     (colon) terminates a control function
      *     logical AND
      +     logical OR
      '     logical invert (NOT)
      <-    denotes transfer of information
      ()    denotes a portion of a register
      plus  arithmetic addition operator
      minus arithmetic subtraction operator
      #     comment follows


PRIOR TO EXECUTION:

            # Makes initial instruction a HOP with the HOP word at
            # sector 0, address 0.
      MB0 <- 0
      MB1 <- 0

      NZR <- 0
      MIR <- 0

      ----------------------------------------------------------------
PHASE A
            # Phase A begins with the 26-bit instruction word already
            # loaded into MBR0 and MBR1. The address for the
            # current instruction (the program counter) is in MAR

            # Phase A: Move the instruction word from MBx, where x is
            # the syllable selected by SYR, to IAR, ORES, and OPR.
            # Also, rotate the accumulator in phases A and B so all bits
            # pass by the NZR register; by the end of Phase B, NZR detects
            # whether the accumulator is zero or non-zero.

      IAR  <- MB[SYR, 01-08]
      ORES <- MB[SYR, 09]
      OPR  <- MB[SYR, 10-13]
      ACC[14-26] <- ACC[1-13]; ACC[1-13] <- ACC[14-26]

      ----------------------------------------------------------------
PHASE B
            # Phase B: prepare to access the operand by moving the
            # operand address from IAR into MAR. Simultaneously, move
            # the address for the current instruction into IAR,
            # incrementing by 1 during the move so it references the
            # next instruction. At the end of Phase B, load MB0,1
            # from memory, using the operand address now in MAR.
            # Set the NZR and NIR flags by the end of Phase B so they
            # can be used to decide whether to conditionally branch.
```

```
      MAR <- IAR;  IAR <- (MAR plus 1) (simultaneously)
      ACC[14-26] <- ACC[01-13]; ACC[01-13] <- ACC[14-26]
      NZR <- 1 if accumulator is nonzero.
      MIR <- 1 if accumulator sign is minus


ORES':
            # Phase B: construct 12-bit effective address (ea) for
            # the operand from the sector register and the memory
            # address register.
      ea(AD9-AD12) <- SCR, ea(AD1-AD8) <- MAR


ORES:
            # Phase B: construct 12-bit effective address (ea) for
            # the operand from the residual sector (17 octal) and the
            # memory address register.
      ea(AD9-AD12) <- 17, ea(AD1-AD8) <- MAR


GBIT14:
            # Phase B: Use the operand effective address, now valid
            # by bit 14, to fetch the operand word.
      MBR0,1 <- MEM[ea]  # read syllables 0,1

      ----------------------------------------------------------------
```

PHASE C/D

```
PC * HOP:
            # Phase C: execute HOP instruction.
            # Load IRES, SCR, SYR from MB at the start of Phase C.
            # Then, shift instruction into IAR
      IRES <- MB0[09]
      SCR <- MB0[10-13]
      SYR <- MB1[02]

      IAR <- MB[01-08]


PC * HOP:
            # End of Phase D: execute HOP instruction.


PC/D * CLA:
            # Phases C and D: execute CLA instruction.
      ACC <- operand


PC/D * ADD:
            # Phases C and D: execute ADD instruction.
      ACC <- ACC plus operand


PC/D * AND:
            # Phases C and D: execute AND instruction. Does a bitwise
            # logical and of the operand with the accumulator.
```

```
      ACC <- ACC * operand


PC/D * SHF * MAR(A4-A6)=3:
            # Phases C and D: execute SHF instruction (shift left one place).
      ACC(S-M24) <- ACC(M1-M25)
      ACC(M25) <- 0


PC/D * SHF * MAR(A4-A6)=4:
            # Phases C and D: execute SHF instruction (shift left two
places).
      ACC(S-M23) <- ACC(M2-M25)
      ACC(M24) <- 0
      ACC(M25) <- 0


PC/D * SHF * MAR(A1-A3)=1 * MAR(A4-A6)=2:
            # Phases C and D: execute SHF instruction (shift right one
place).
      ACC(M2-M25) <- ACC(M1-M24)
      ACC(M1) <- ACC(S)


PC/D * SHF * MAR(A1-A3)=0 * MAR(A4-A6=2):
            # Phases C and D: execute SHF instruction
            # (shift right two places).
      ACC(M3-M25) <- ACC(M1-M23)
      ACC(M1) <- ACC(S)
      ACC(M2) <- ACC(S)


PC/D * STO:
            # Phases C and D: execute STO.
      MEM[ea, 01-26] <- ACC(S-M25)


PC/D * CLD:
            # Phases C and D: execute CLD. Copied to all bits in the ACC.
      ACC(S-M25) <- discrete input selected by operand.


PC/D * PRO * input * ORES':
            # Phases C and D: execute PRO. OR inputs with ACC if A9 is a 0.
      ACC <- ACC + input


PC/D * PRO * input * ORES:
            # Phases C and D: execute PRO.
      ACC <- input


PC/D * PRO * output * ORES':
            # Phases C and D: execute PRO.
      output <- ACC
```

```
PC/D * PRO * output * ORES:
            # Phases C and D: execute PRO. Clear ACC if A9 is a 1.
      output <- ACC
      ACC <- 0

      ------------------------------------------------------------
```

PHASE E

```
IRES' * (not a branch):
            # Phase E: construct 12-bit effective address (ea) for
            # the operand from the sector register and the memory
            # address register.
      ea(AD9-AD12) <- SCR, ea(AD1-AD8) <- MAR


IRES * (not a branch):
            # Phase E: construct 12-bit effective address (ea) for
            # the operand from the residual sector (17 octal) and the
            # memory address register.
      ea(AD9-AD12) <- 17, ea(AD1-AD8) <- MAR


(not a branch):
            # Phase E: prepare to fetch the next instruction. Move the
            # address for the next instruction from IAR to MAR.
            # If it's a branch, the branch address is already in MAR.
      MAR <- IAR(A1-A8)

PE * GBIT14:
            # Phase E: Use the instruction effective address, now valid
            # by bit 14, to fetch the next memory word
      MBR0,1 <- MEM[ea]  # read syllables 0,1
      NZR <- 0
      MIR <- 0
```

Logic Signals

All signals shown here are positive logic. The actual circuit implementation is mostly negative logic so floating inputs will be inactive during assembly and checkout. The signals in the actual implementation which are negative logic have a “N' prefix added.

```
CLMB        = PURST     # forces a HOP 0 for the first instruction.

WMB         = (PB * GBIT14) + (PE * GBIT14)

SLMB        = (PA * G13) + (PC * G13) + (PD * G13)

SLOPR       =  PA * G13

SLORES      =  PA * G13

SLIAR       = (PA * G13) + (PB * G8)
            + (PE * G8 * (TRA + (TNZ * NZR) + (TMI * MIR))')
            + (PC * HOP * G8)

SLMAR       = (PB * G8)
            + (PE * G8 * (TRA + (TNZ * NZR) + (TMI * MIR))')

SLACC       = (PA * G13) + (PB * G13)
            + (PC * SHF' * G13) + (PD * SHF' * G13)
            + (PC * SL1  * G12) + (PD * SL1  * G13) # SHF left 1 bit
            + (PC * SL2  * G12) + (PD * SL2  * G12) # SHF left 2 bits
            + (PC * SR1  * G1)                      # SHF right 1 bit
            + (PC * SR2  * G1)  + (PD * SR2  * G1)  # SHF right 2 bits
            + (PC * ISHF * G13) + (PD * ISHF * G13) # invalid SHF, zero

            # Decode the shift operand
SL1         = shift left 1 bit
SL2         = shift left 2 bits
SR1         = shift right 1 bit
SR2         = shift right 2 bits
ISHF        = invalid shift code, zero the accumulator

            # Insert the 0's or sign bits into the shifted (SHF) word
WACCSL1     = PD * SL1 * GBIT14
WACCSL2     = PD * SL2 * GBIT14
WACCSR1     = PD * SR1 * GBIT14
WACCSR2     = PD * SR2 * GBIT14

CLNZR       = PE * GBIT14
SETNZR      = (ACCOUT * PA * G13) + (ACCOUT * PB * G13)

CLMIR       = PE * GBIT14
SETMIR      = MB[26] * PB * GBIT14

            # Load other HOP word fields before shifting A1-A8 into IAR
WIRES       = PC * HOP * G1         # residual bit from HOP instruction
WSCR        = PC * HOP * G1         # sector from HOP instruction
WSYR        = PC * HOP * G1         # syllable from HOP instruction

WMEM        = PD * STO * GBIT14     # writes MB0 and MB1 to memory
```

IAR SELECT

```
ISOUT       = (IN1 * PA) + (IN2 * (PB + (PC * HOP'))) + (IN3 * (PC * HOP))

where:
IN1         = output of ORES
IN2         = output of PLUS 1 ADDER
IN3         = output of INSTRUCTION SELECT (ISEL)
ISOUT       = shift left data input to IAR
```

AS SELECT

```
Control codes for shift left data input to ACC:
AS7         = A (ACCOUT) default
AS6         = M (MB0OUT)
AS5         = A * M
AS4         = A + I
AS3         = I (IMUXOUT)
AS2         = ADDOUT
AS1         = SUBOUT
AS0         = 0 (LOGICAL ZERO)


AS6         = (PC + PD) * CLA
AS5         = (PC + PD) * AND
AS4         = (PC + PD) * (PRO * ORES')
AS3         = (PC + PD) * ((PRO * ORES) + CLD)
AS2         = (PC + PD) * ADD
AS1         = (PC + PD) * (SUB + RSU)
AS0         = (PC + PD) * ISHF
```

PLUS 1 ADDER

```
IN2         = PB * G1
```

MDIU

```
SLKBDR      = PRO5 * (PC + PD) * G13
SLTIMR      = PRO6 * (PC + PD) * G13

WTIMR       = PB * GBIT14
WKBDR       = PB * GBIT14 * KPRESS


            # Signal to clear the MDIU DI01-DI04 latches & KBDR:
RESET       = PRO * DO01 * PD * GBIT14

SLDSELR     = PRO * DO02 * PC * G4
SLDMAGR     = PRO * DO03 * PC * G4
SLDDVR      = PRO * DO04 * PC * G1
```

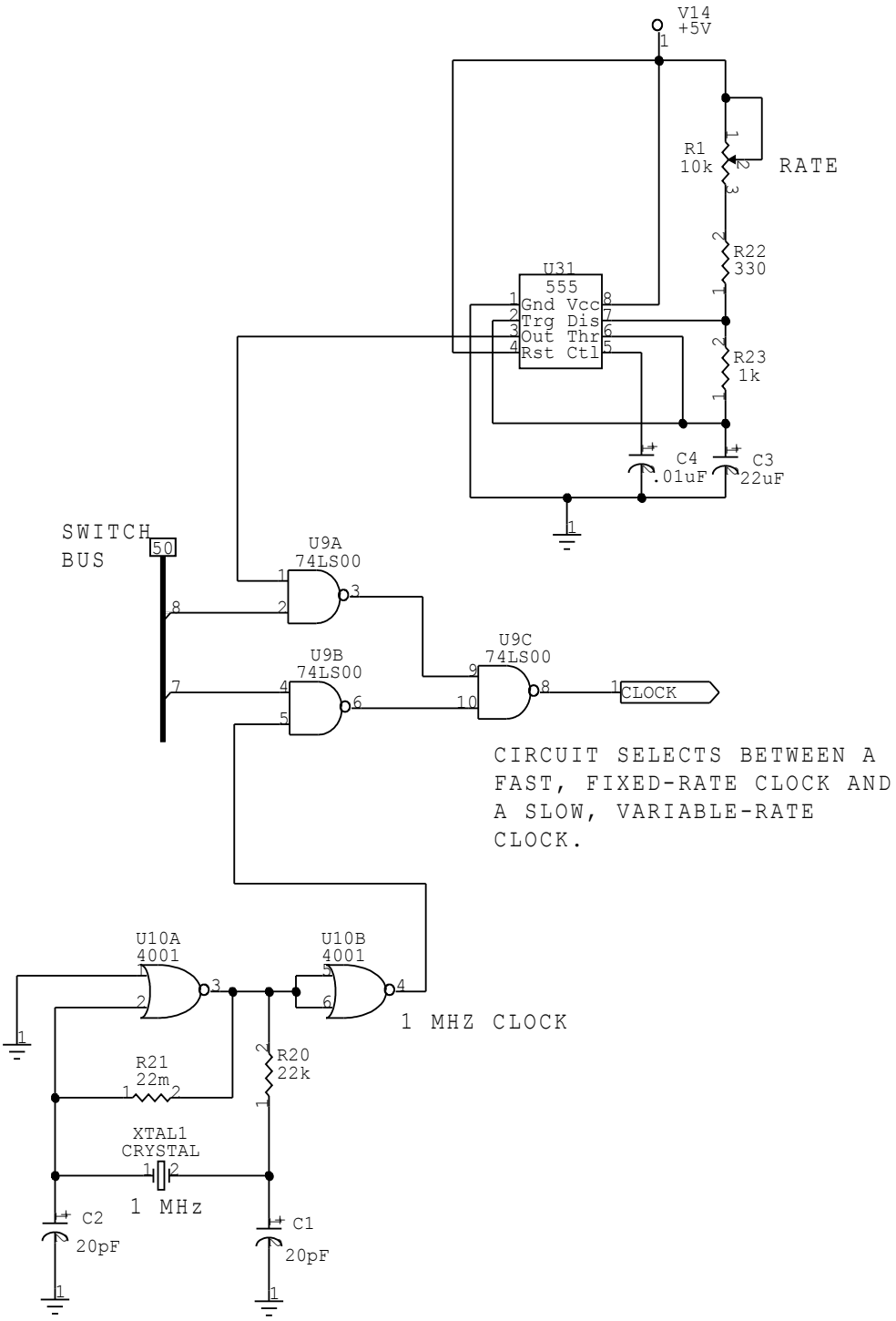
Timing Board

TBS.

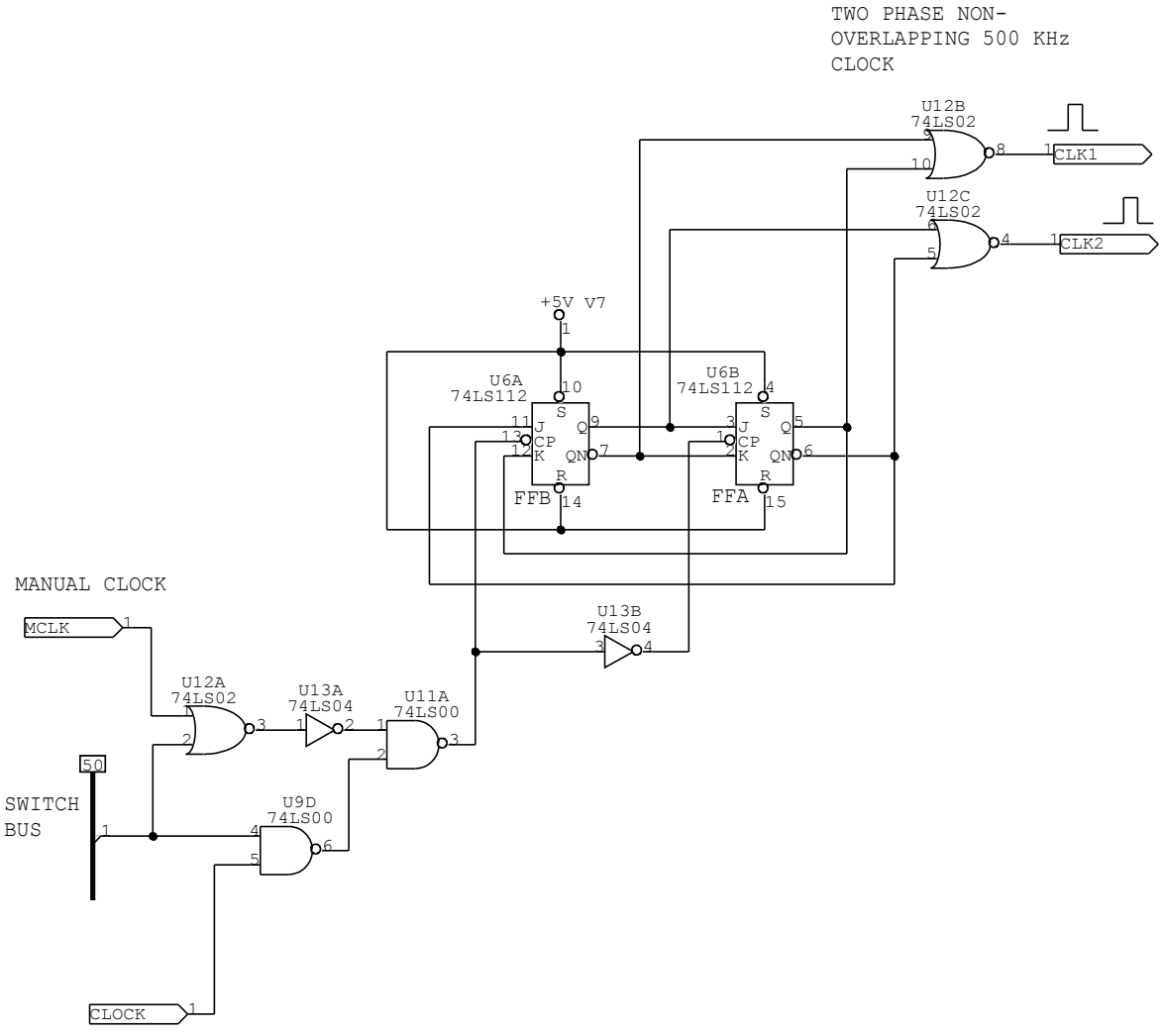
Timing Board IC List

DESIGNATION	PART	QTY
U16:	74LS08	(1)
U5,U4:	74LS05	(2)
U18:	74LS194	(1)
U19,U20:	74LS244	(2)
U13,U15,U14:	74LS04	(3)
U12,U1:	74LS02	(2)
U11,U9,U7:	74LS00	(3)
U6:	74LS112	(1)
U21,U22,U23,U24,U25,U26:	74LS160A	(6)
U8,U3,U17:	74LS20	(3)
U2:	74LS109	(1)
U27:	74LS138	(1)
U28,U30:	74LS161A	(2)
U29:	74LS154	(1)
U10:	4001	(1)
U31:	555	(1)

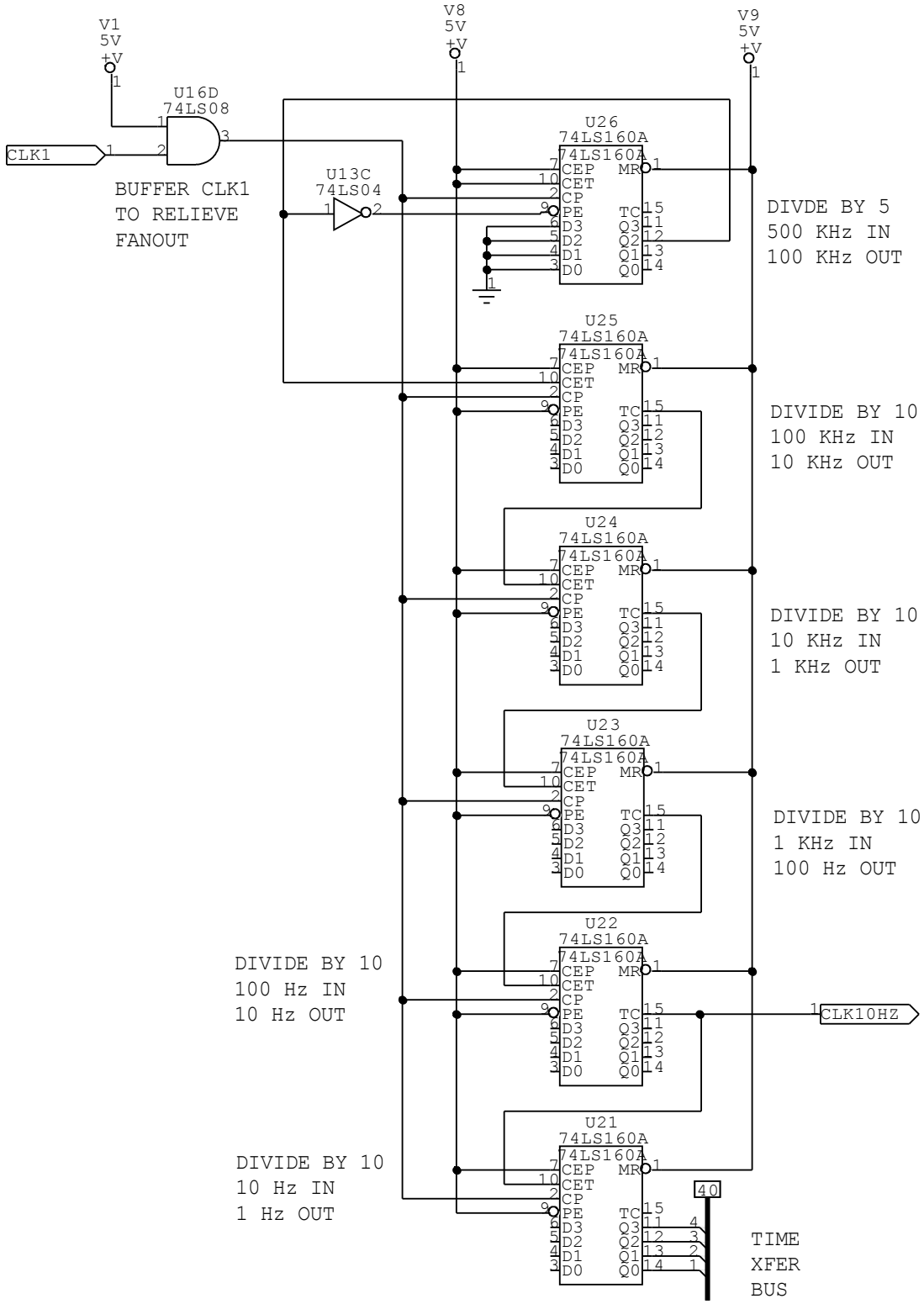
GEMINI TIMING #1
CLOCK OSCILLATOR



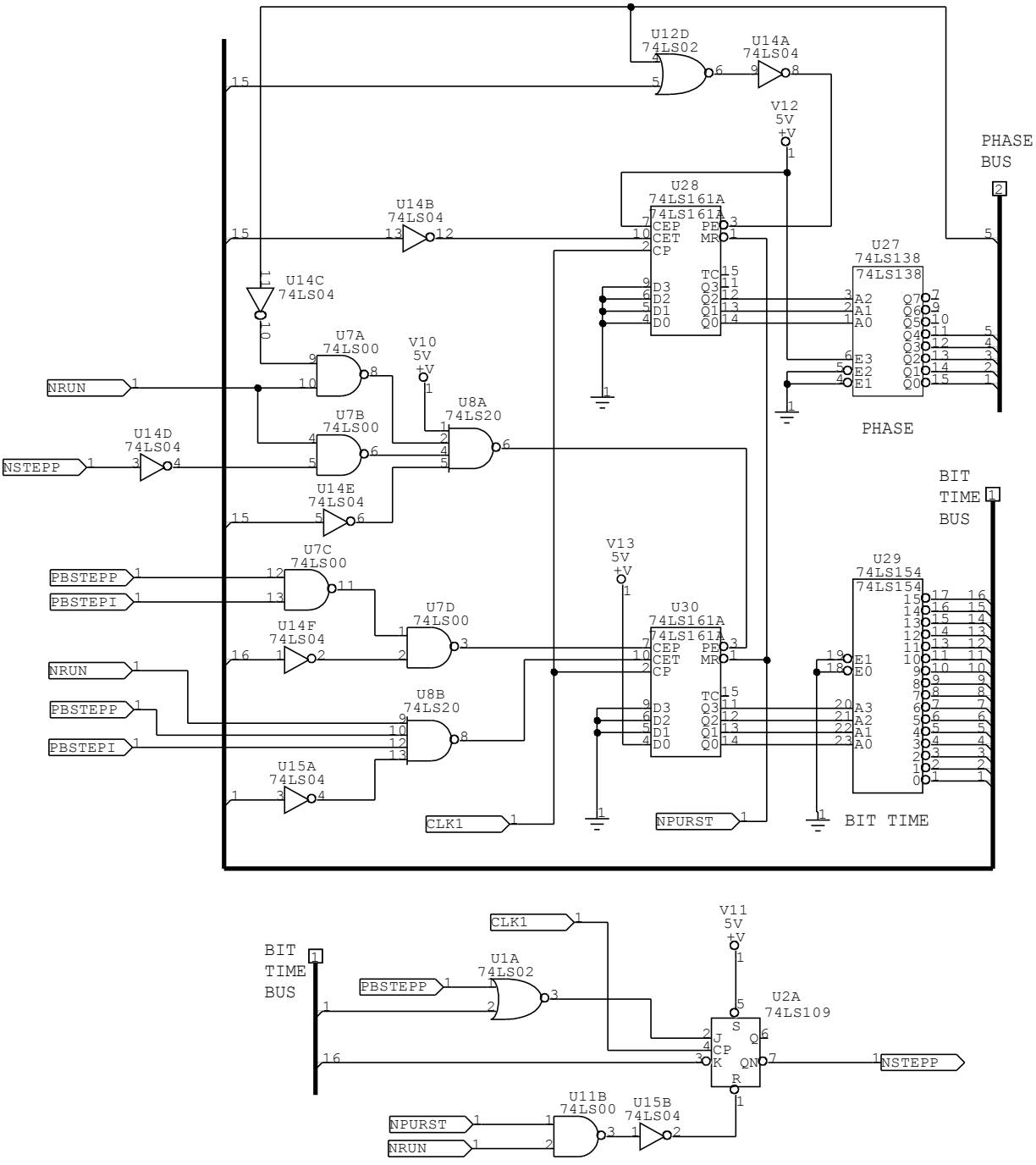
GEMINI TIMING #2
2 PHASE CLOCK



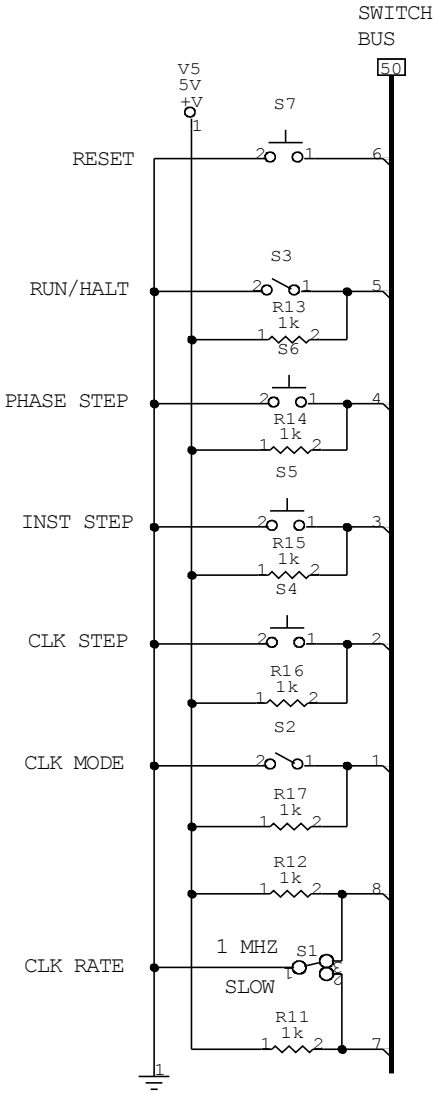
GEMINI TIMING #3
TIME BASE

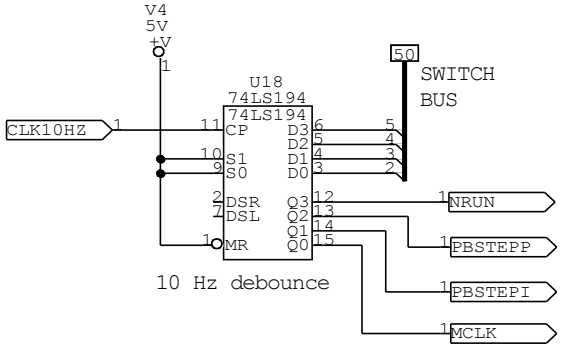


GEMINI TIMING #4
BIT-TIME AND
PHASE SEQUENCERS

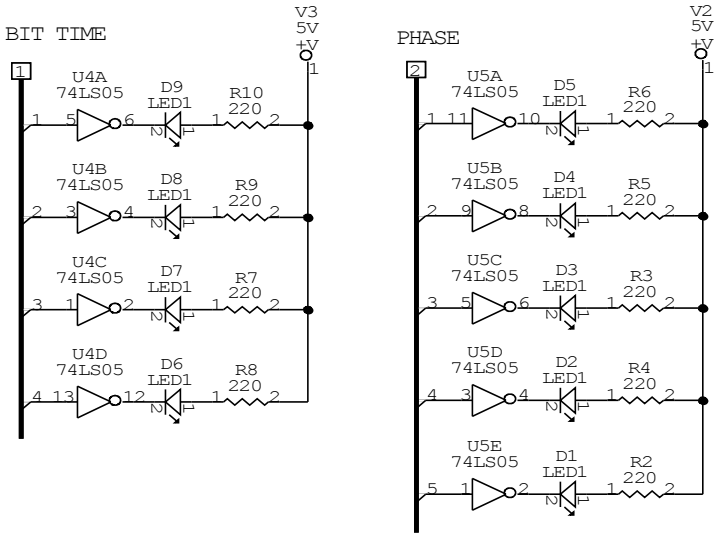


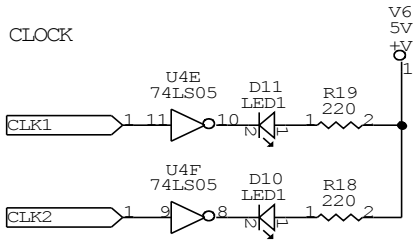
GEMINI TIMING #6
SWITCHES





GEMINI TIMING #7
DISPLAYS

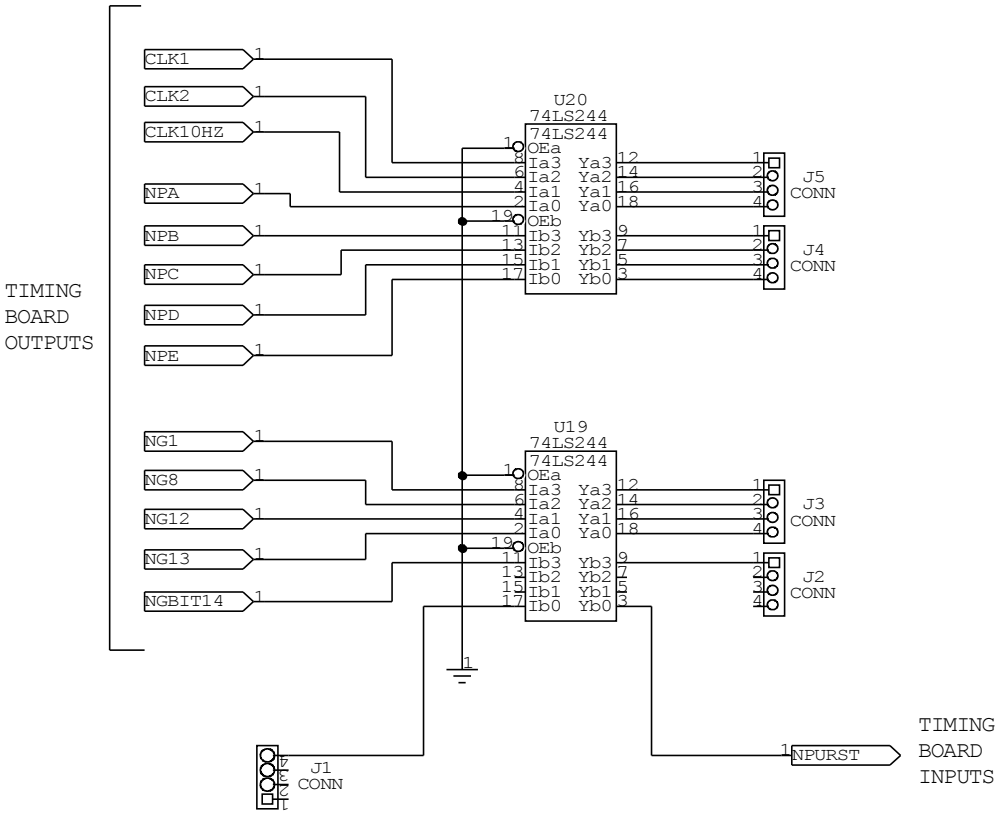




GEMINI TIMING #8
EXTERNAL INTERFACES

"SWITCH BUS" 6 IS OUTPUT
TO THE MEMORY BOARD
BUT NOT BUFFERED.

"TIME XFER BUS" 1-4 IS
OUTPUT TO THE MDIU BOARD
BUT NOT BUFFERED.



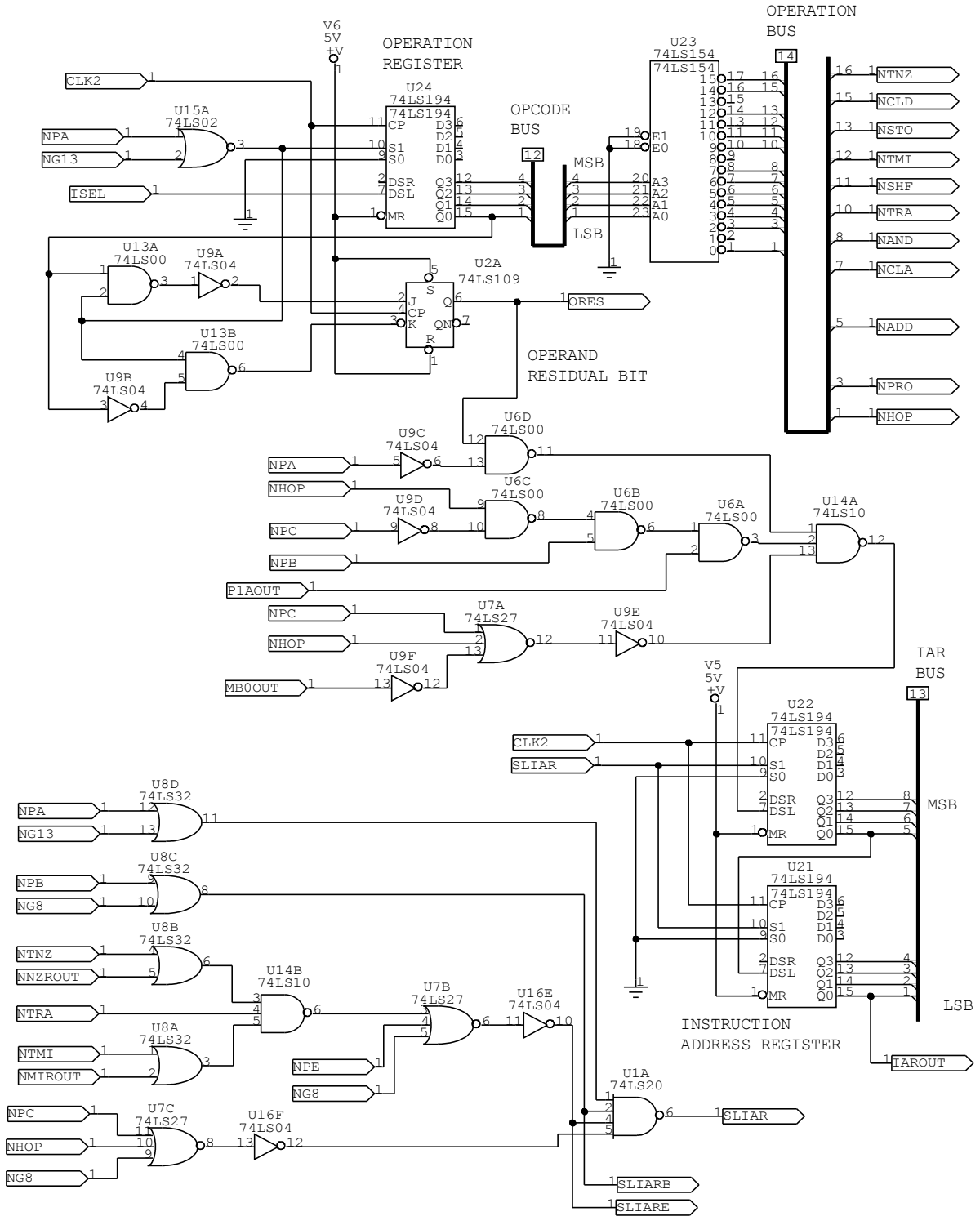
Control Board

TBS.

Control Board IC List

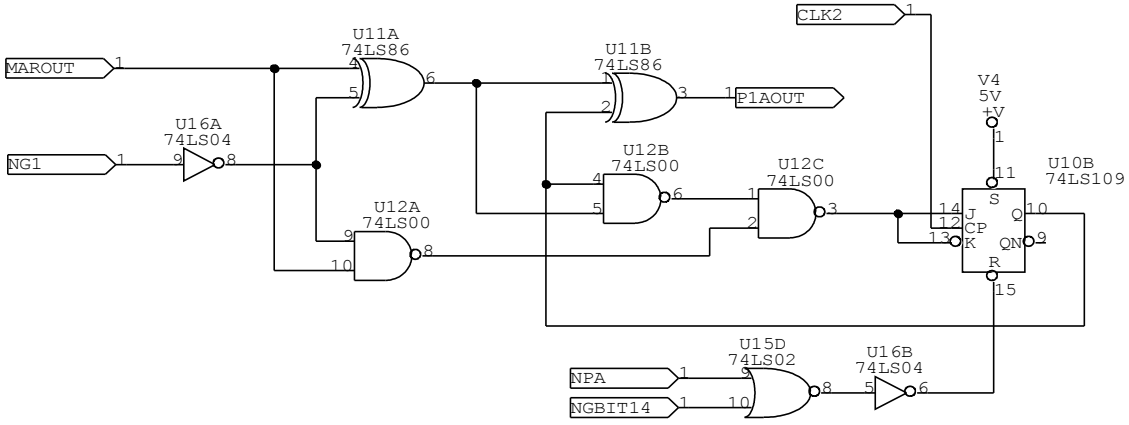
DESIGNATION	PART	QTY
U17,U18,U19,U20:	74LS244	(4)
U3,U4,U5:	74LS05	(3)
U16,U9:	74LS04	(2)
U15:	74LS02	(1)
U10,U2:	74LS109	(2)
U12,U6,U13:	74LS00	(3)
U11:	74LS86	(1)
U7:	74LS27	(1)
U1:	74LS20	(1)
U14:	74LS10	(1)
U8:	74LS32	(1)
U21,U22,U24:	74LS194	(3)
U23:	74LS154	(1)

GEMINI CONTROL #1
INSTRUCTION REGISTERS

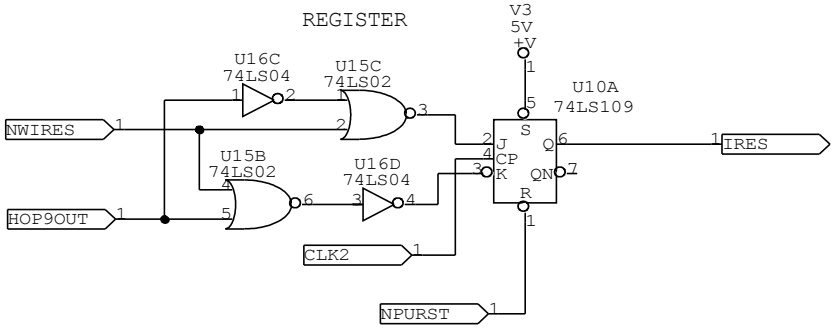


GEMINI CONTROL #2
ADDER/IRES LOGIC

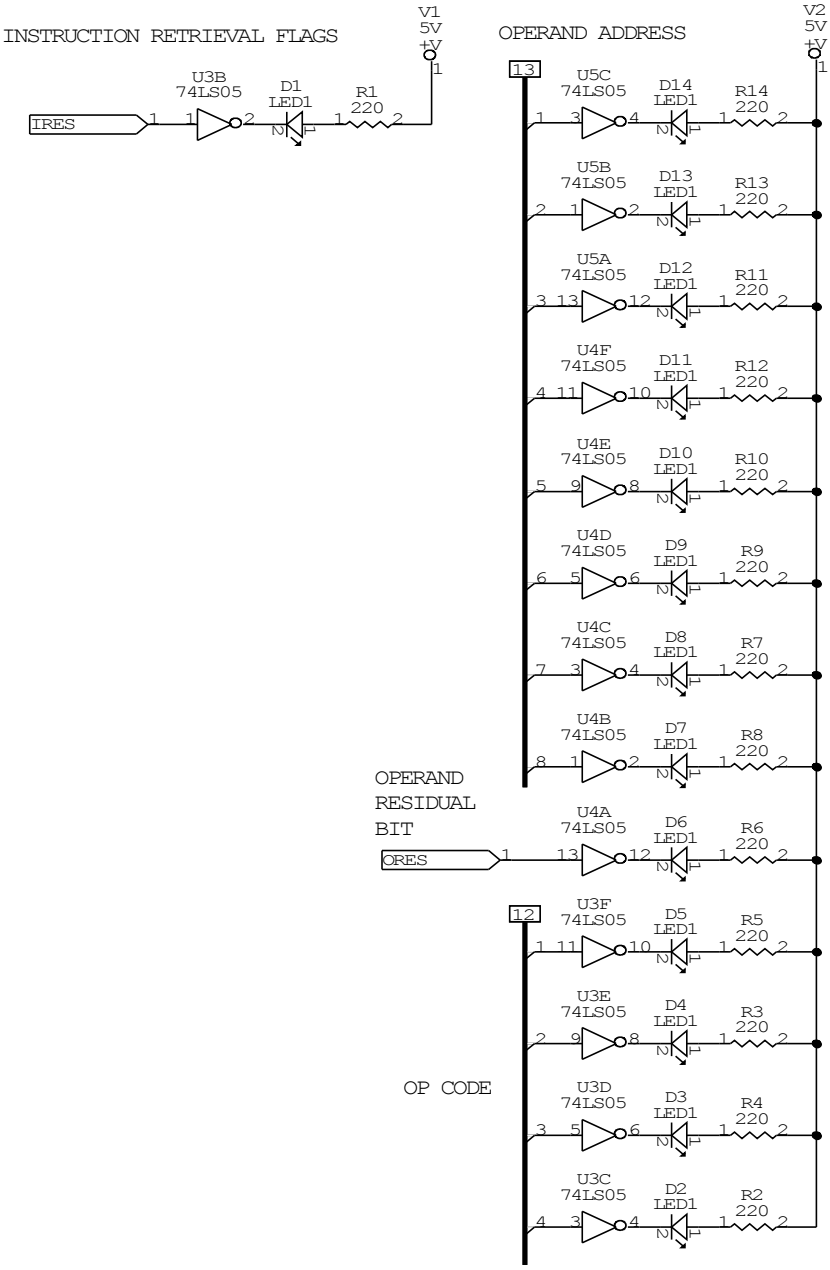
PLUS 1 ADDER



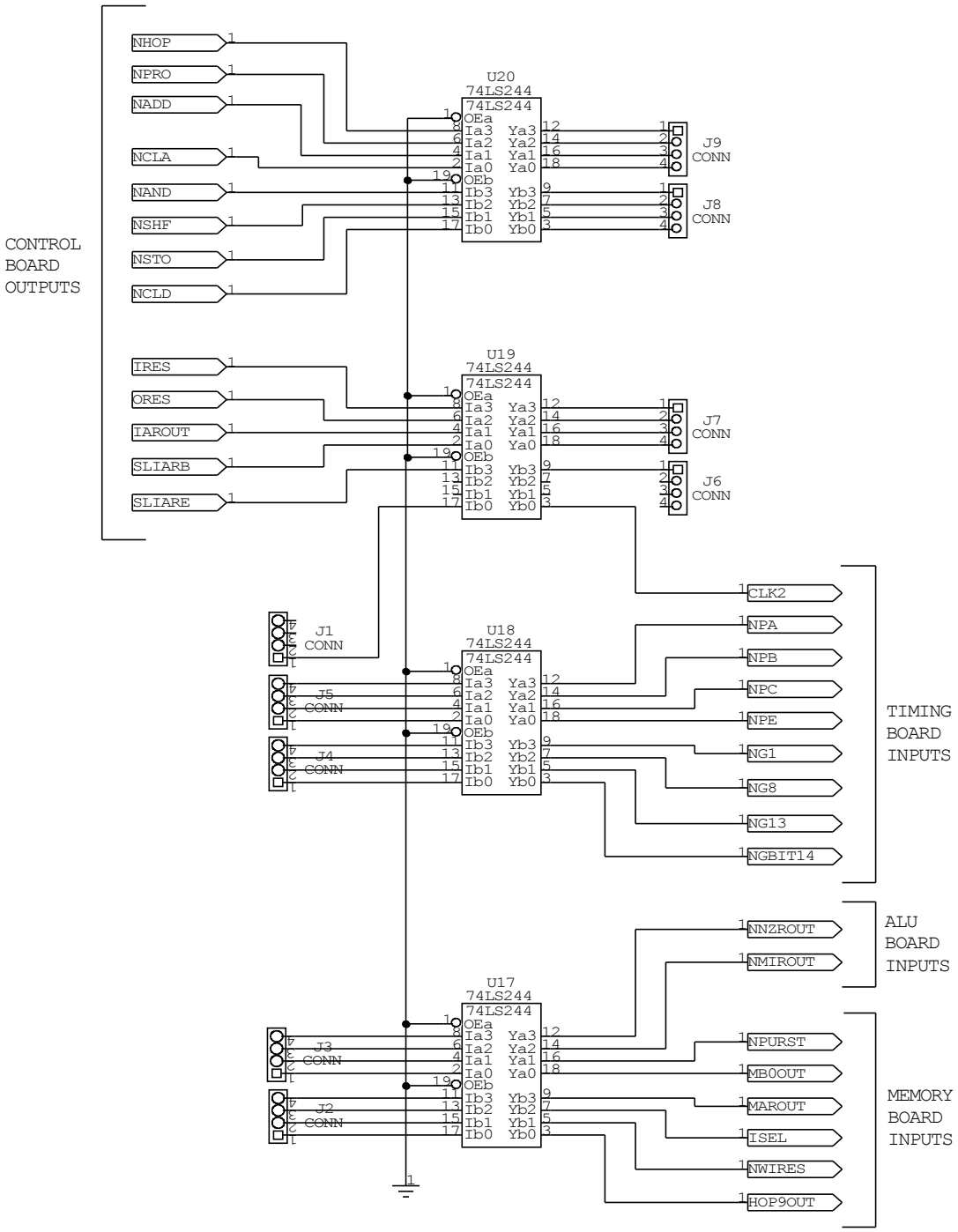
INSTRUCTION
RESIDUAL BIT
REGISTER



GEMINI CONTROL #3
DISPLAYS



GEMINI CONTROL #4
EXTERNAL INTERFACES



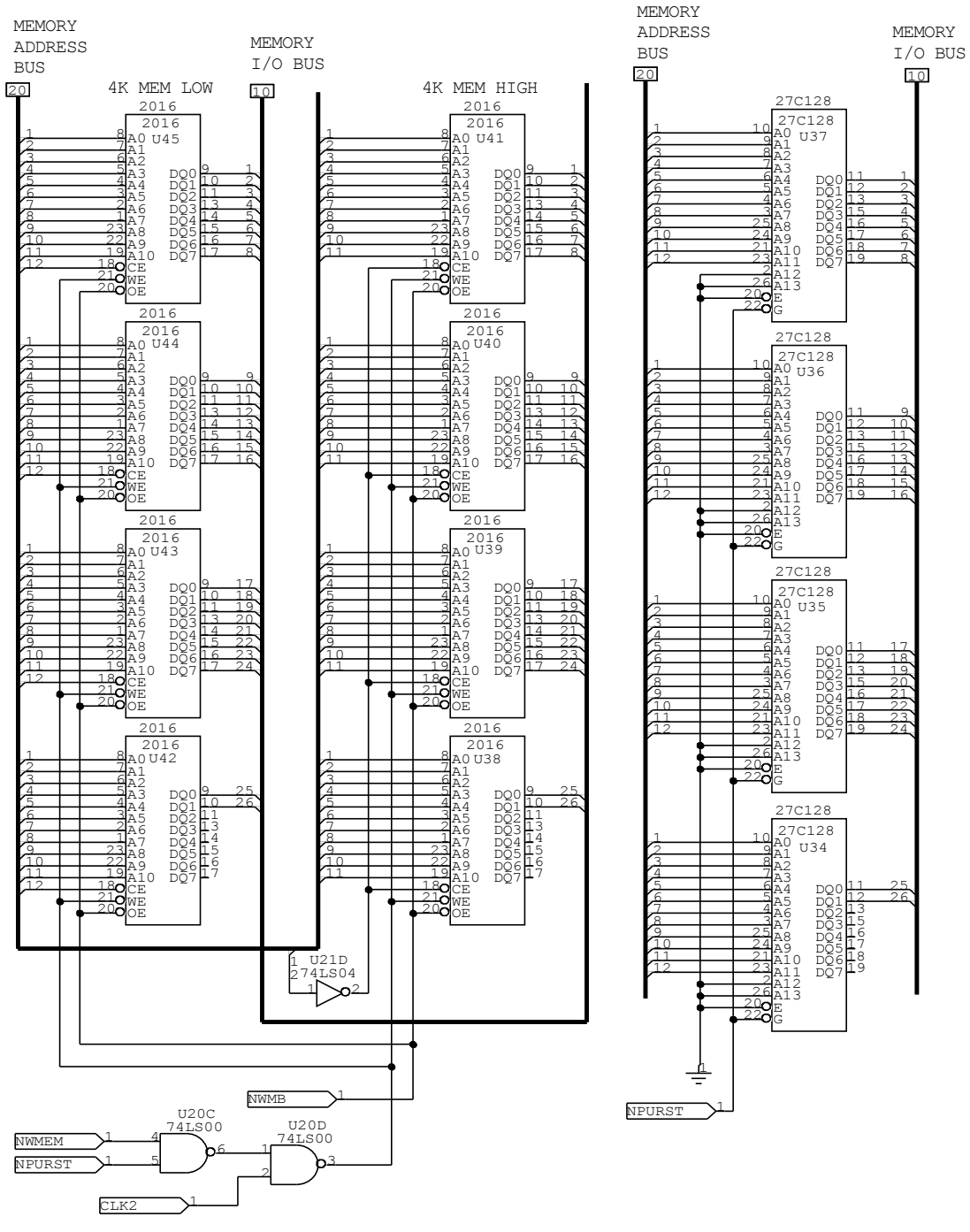
Memory Board

TBS.

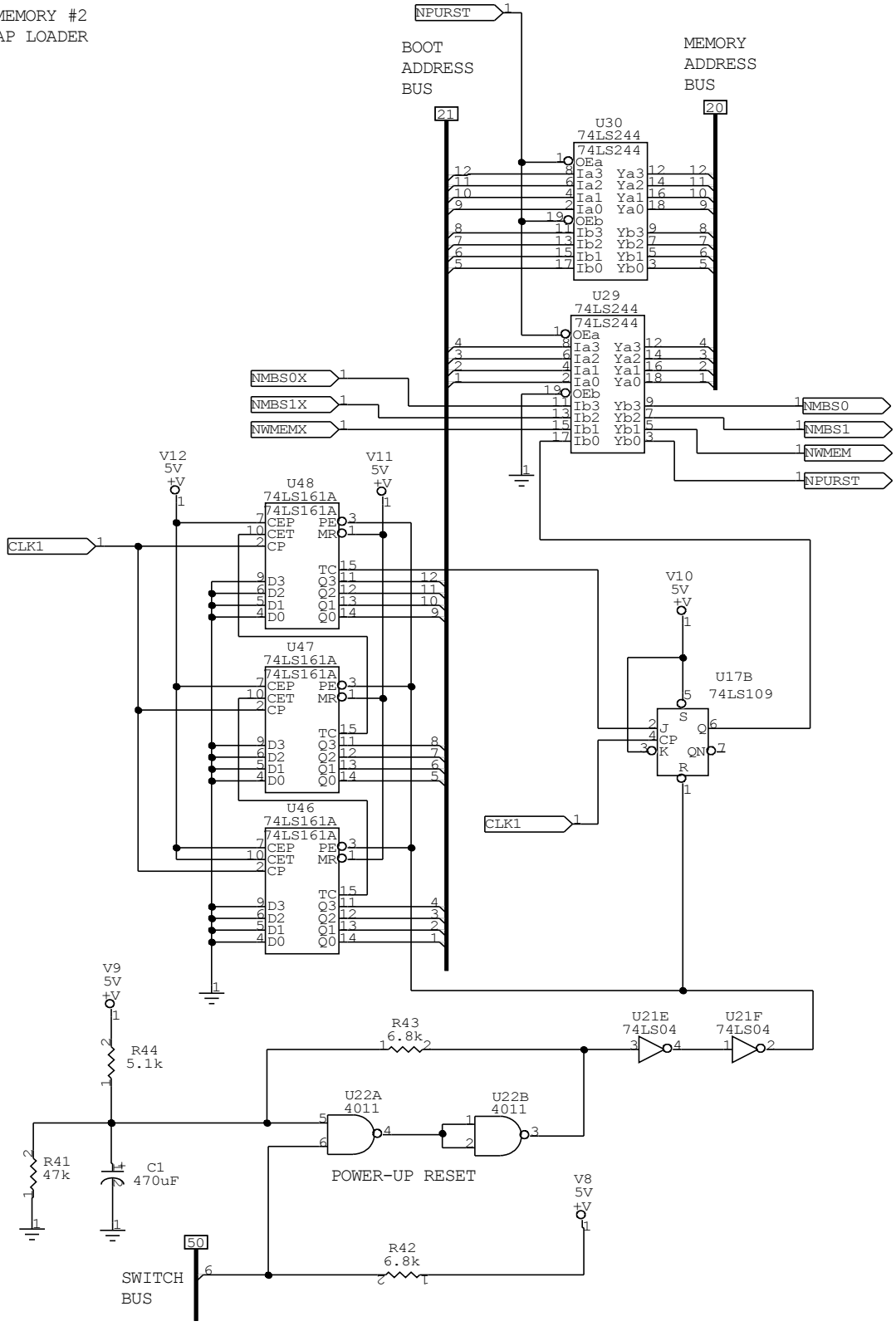
Memory Board IC List

DESIGNATION	PART	QTY
U21,U15,U16:	74LS04	(3)
U19:	74LS27	(1)
U14,U20,U13,U12,U8:	74LS00	(5)
U23,U24,U29,U30,U31,U32,U33,U49,U50,U54,U55:	74LS244	(11)
U25:	74LS157	(1)
U26,U46,U47,U48:	74LS161A	(4)
U27,U28,U51,U52,U53,U56,U57,U58:	74LS194	(8)
U7,U2,U1,U6,U5,U4,U3:	74LS05	(7)
U34,U35,U36,U37:	27C128	(4)
U38,U39,U40,U41,U42,U43,U44,U45:	2016	(8)
U22:	4011	(1)
U17,U9:	74LS109	(2)
U11,U10:	74LS10	(2)
U18:	74LS02	(1)

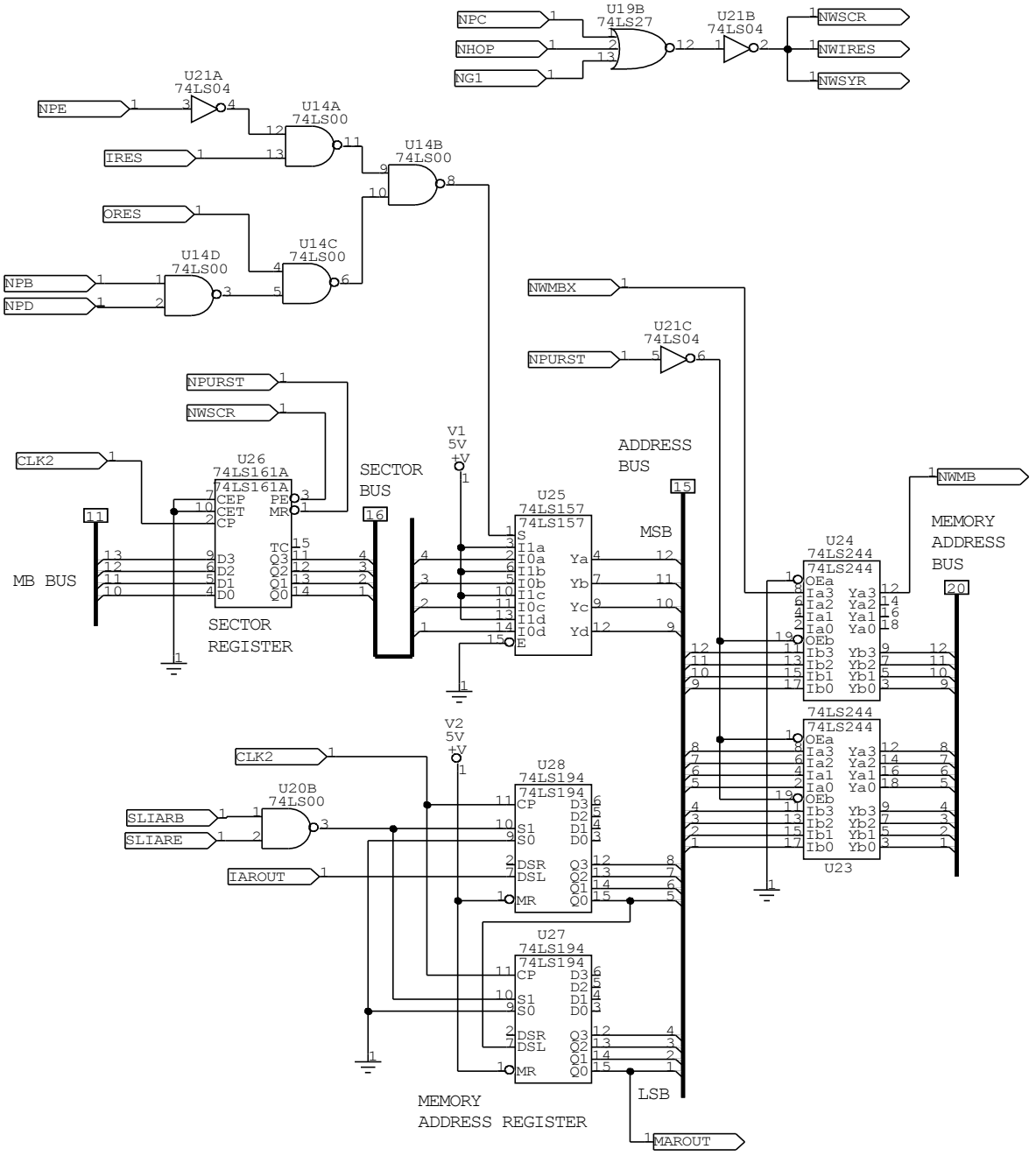
GEMINI MEMORY #1



GEMINI MEMORY #2
BOOTSTRAP LOADER

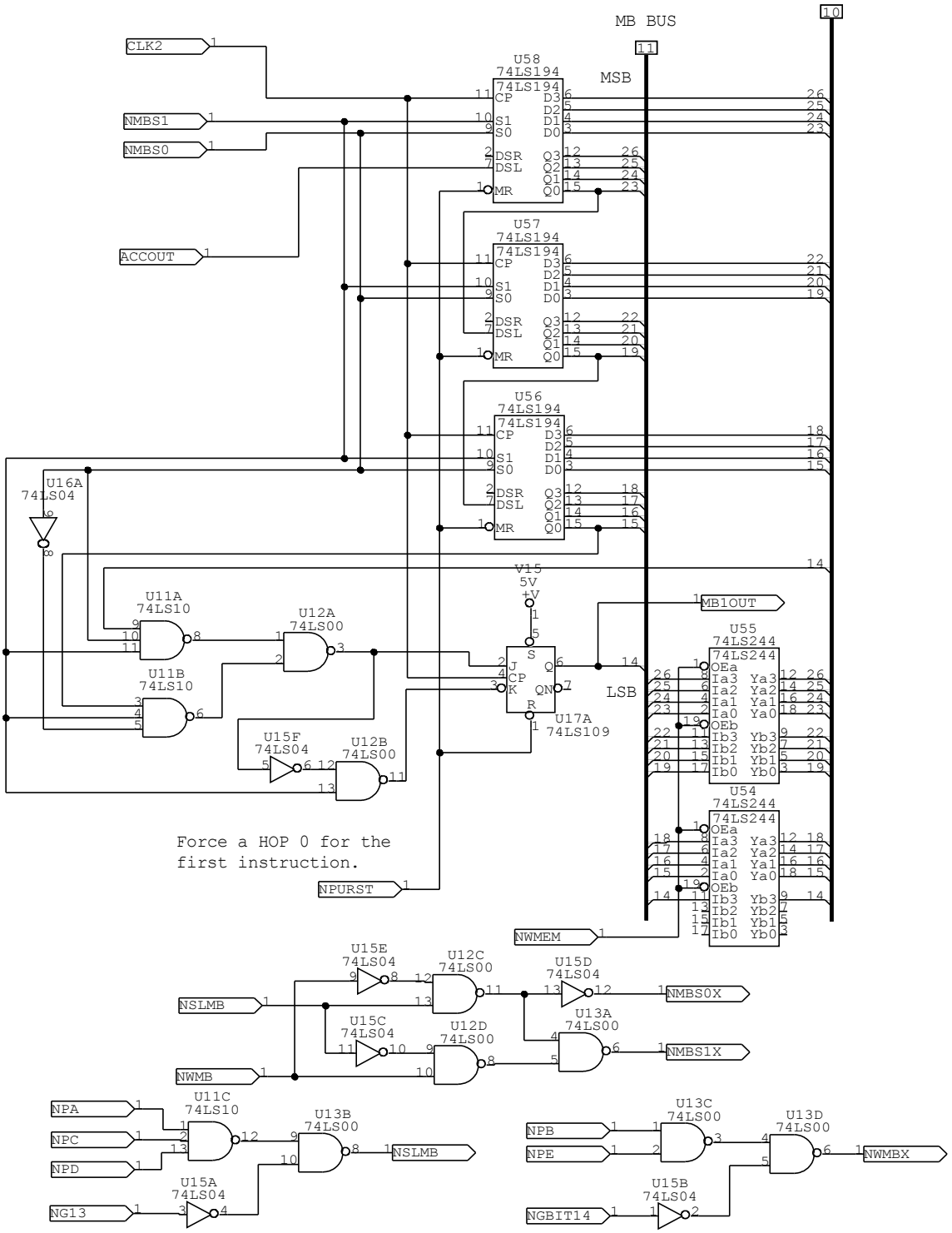


GEMINI MEMORY #3
MEMORY ADDRESS LOGIC

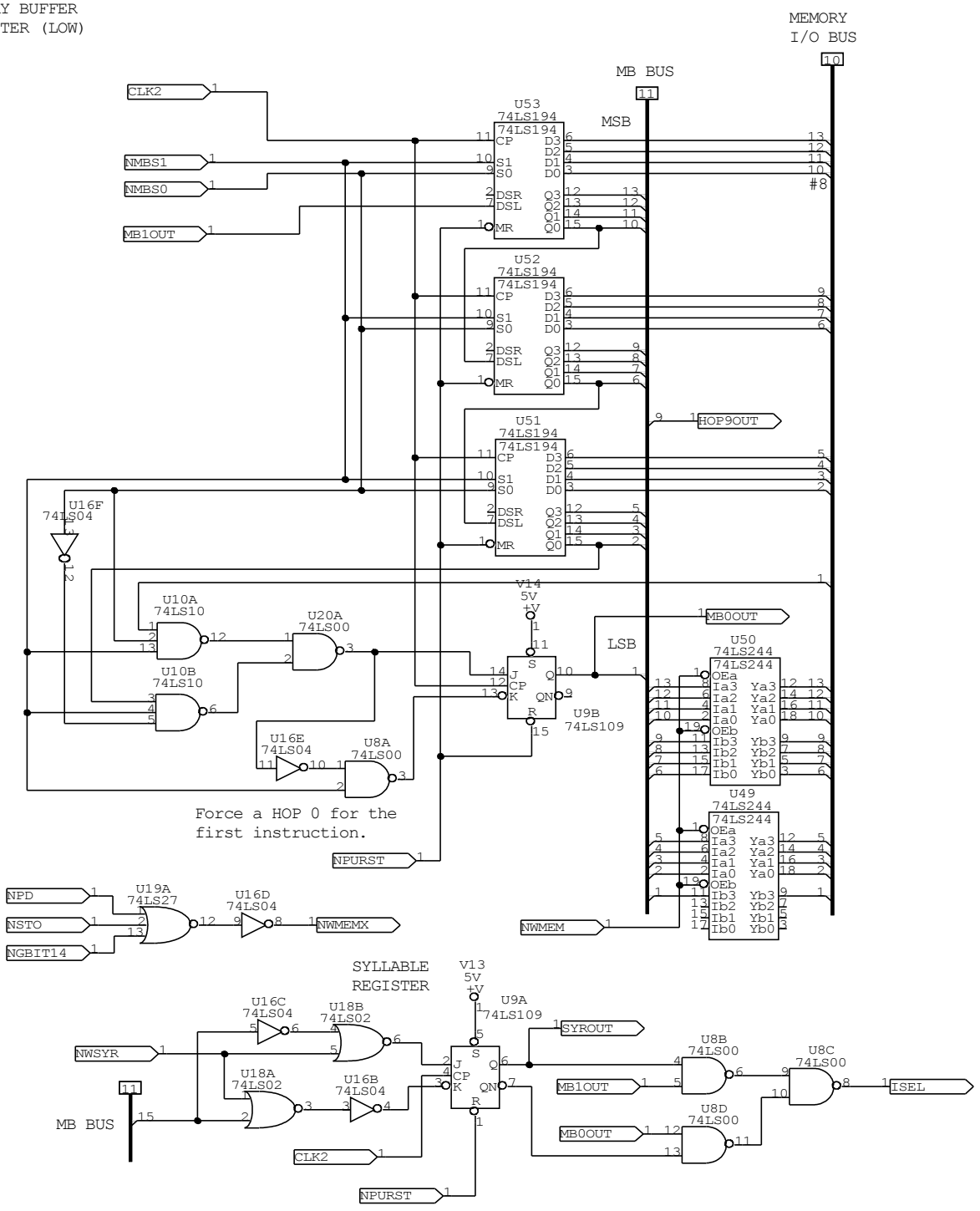


GEMINI MEMORY #4
MEMORY BUFFER
REGISTER (HIGH)

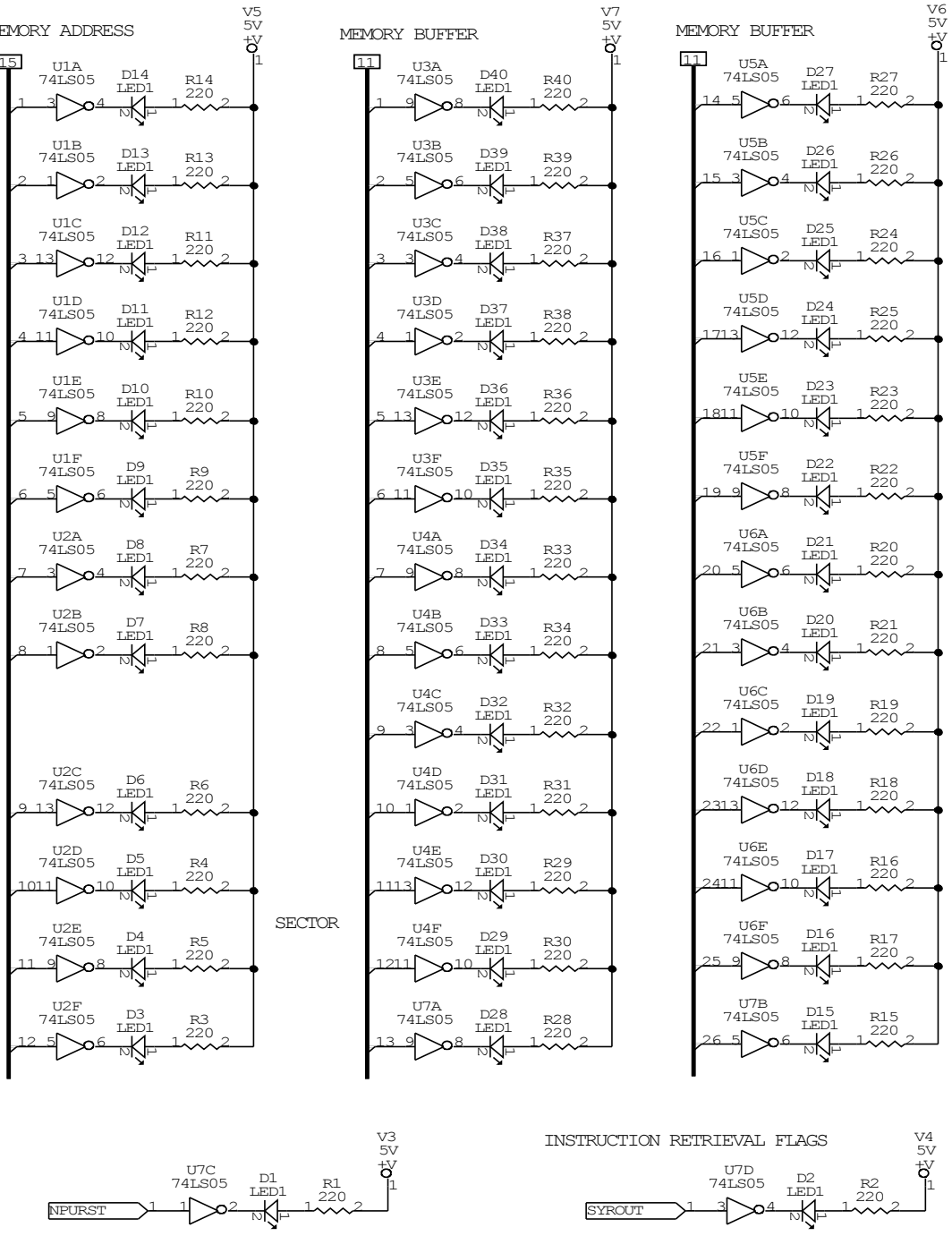
MEMORY
I/O BUS



GEMINI MEMORY #5
MEMORY BUFFER
REGISTER (LOW)



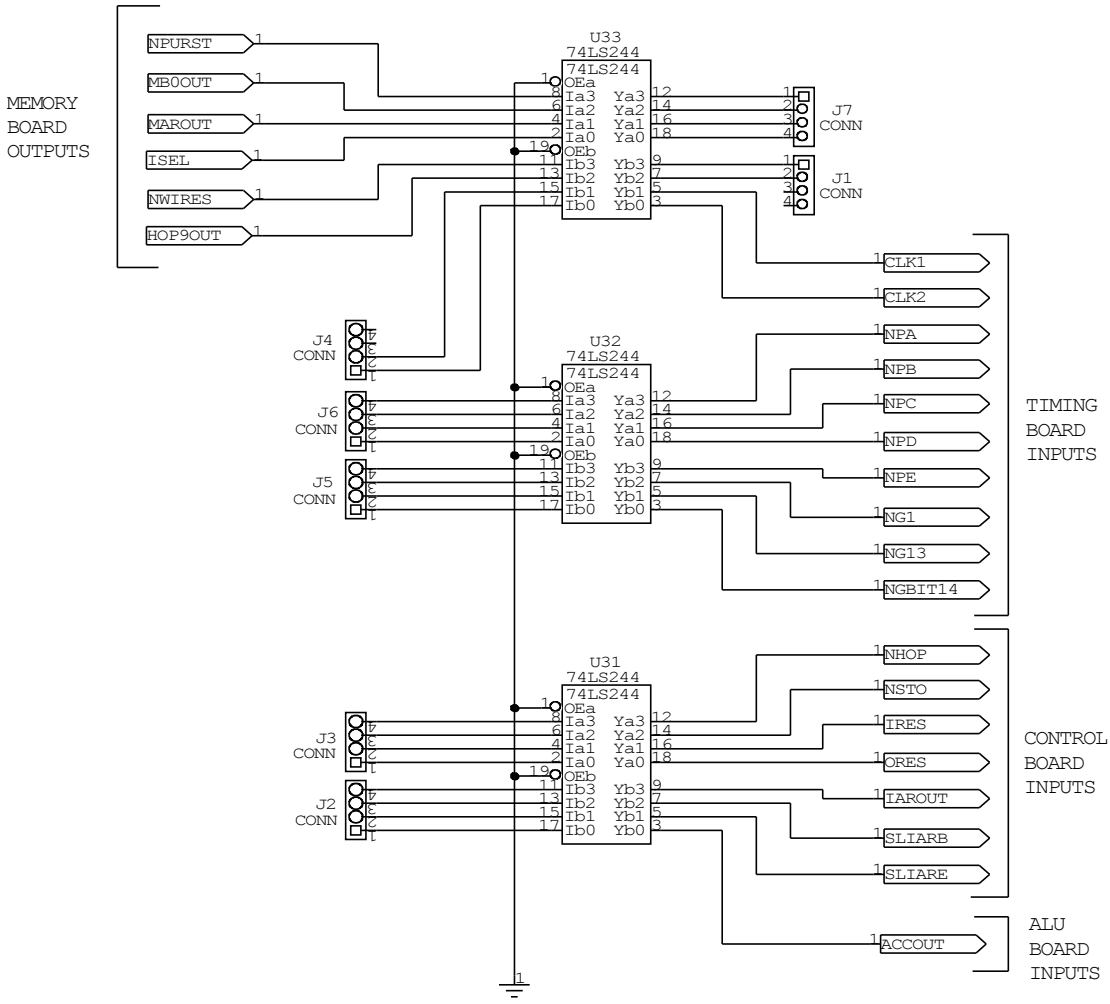
GEMINI MEMORY #6
DISPLAYS



GEMINI MEMORY #7
EXTERNAL INTERFACES

"SWITCH BUS" 6 IS INPUT
TO THE MEMORY BOARD
BUT NOT BUFFERED.

"MEMORY ADDRESS BUS" 1-6 IS
ALSO OUTPUT AND IS ALREADY
BUFFERED.



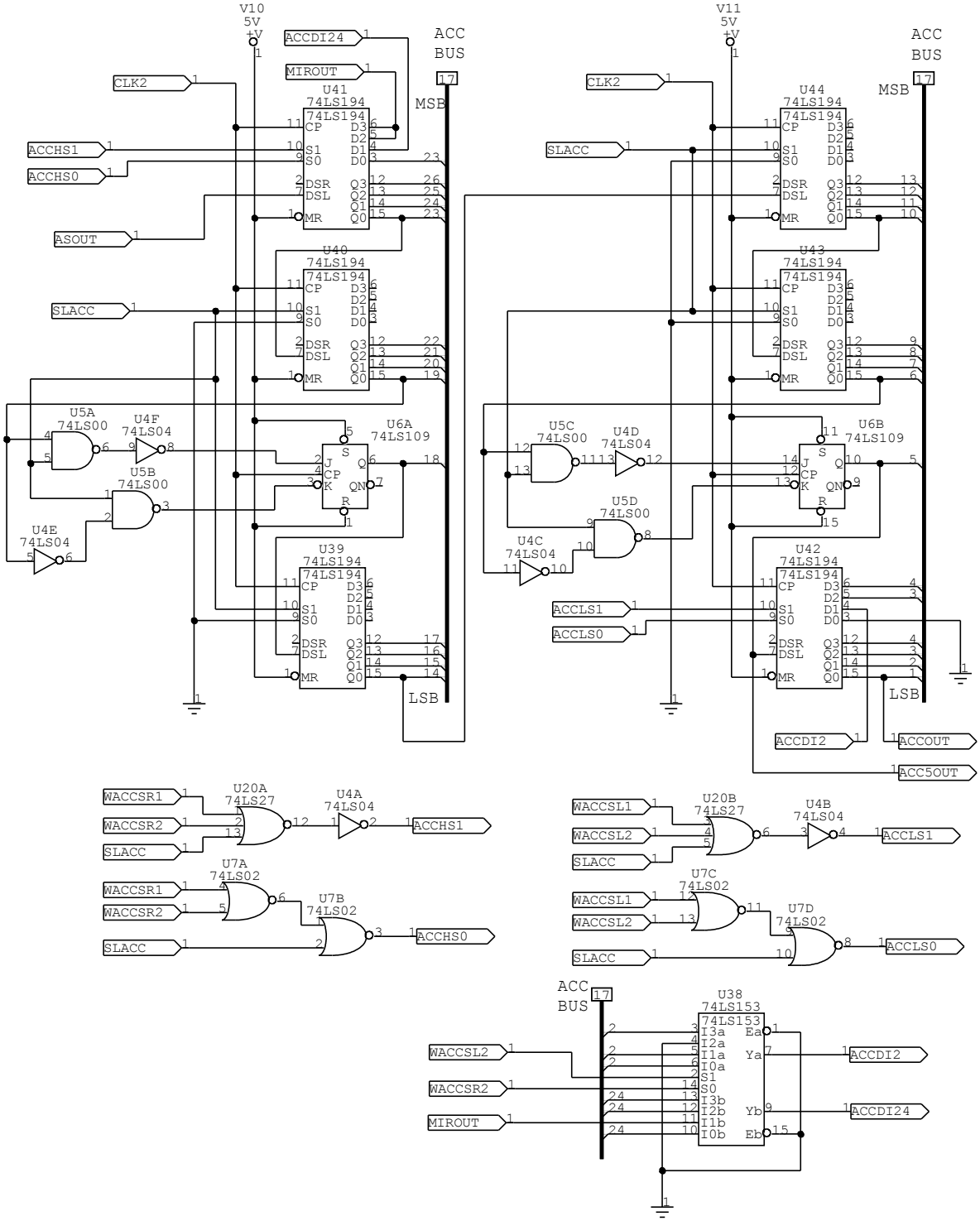
ALU Board

TBS.

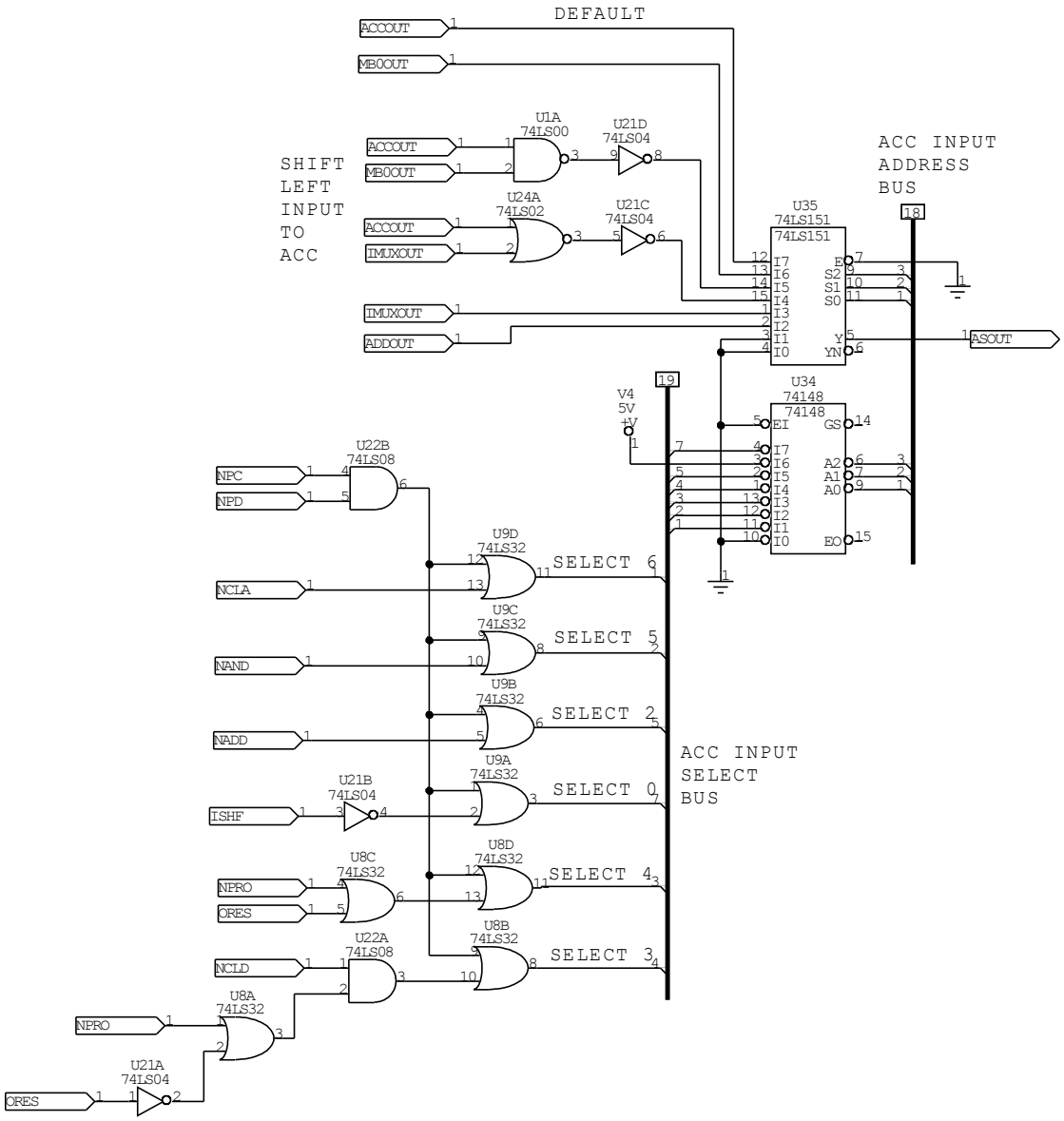
ALU Board IC List

DESIGNATION	PART	QTY
U30,U31,U32,U33:	74LS244	(4)
U19,U16,U15,U18,U17:	74LS05	(5)
U21,U29,U11,U4:	74LS04	(4)
U8,U9,U25,U13:	74LS32	(4)
U22,U23:	74LS08	(2)
U24,U26,U10,U7:	74LS02	(4)
U1,U27,U5:	74LS00	(3)
U34:	74148	(1)
U35:	74LS151	(1)
U2,U14,U6:	74LS109	(3)
U28:	74LS86	(1)
U36:	74LS30	(1)
U20,U12,U3:	74LS27	(3)
U37:	74LS138	(1)
U38:	74LS153	(1)
U39,U40,U41,U42,U43,U44:	74LS194	(6)

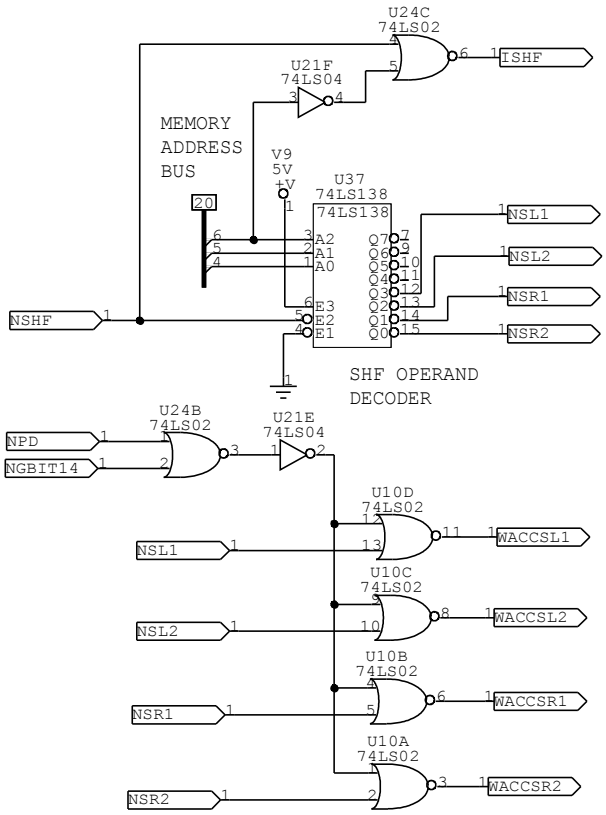
GEMINI ALU #1
ACCUMULATOR REGISTER



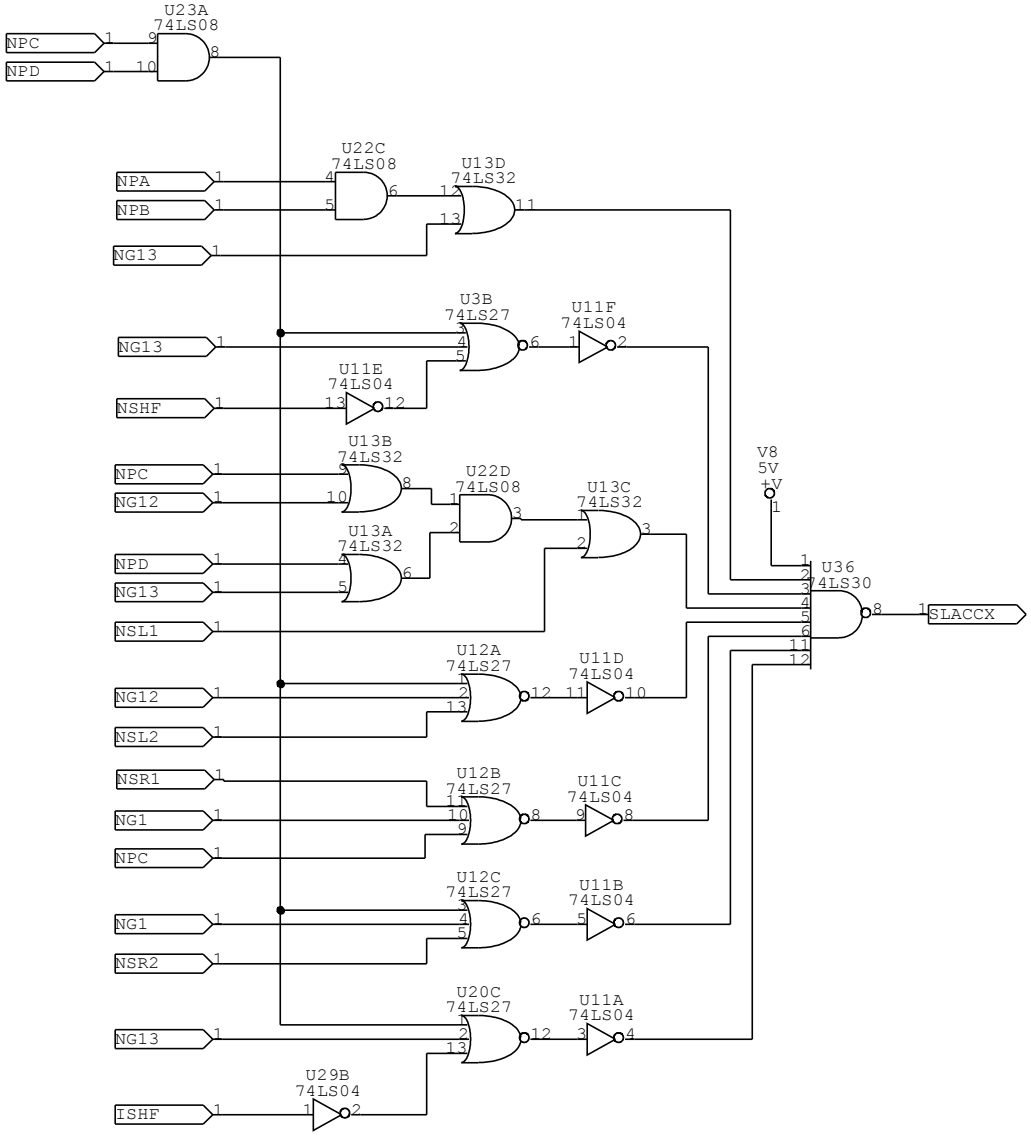
GEMINI ALU #2
ACCUMULATOR INPUT SELECTOR



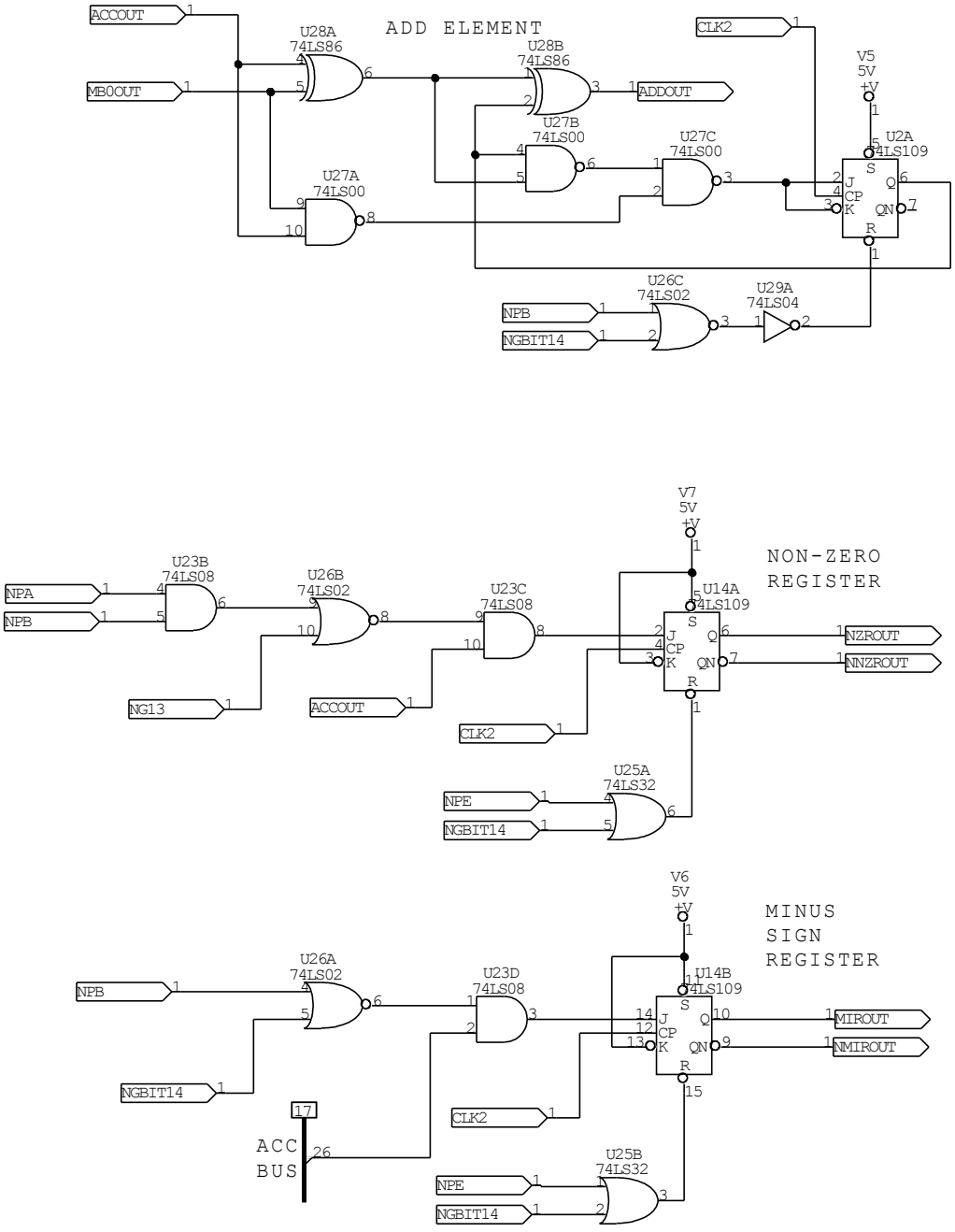
GEMINI ALU #3
SHF INSTRUCTION LOGIC



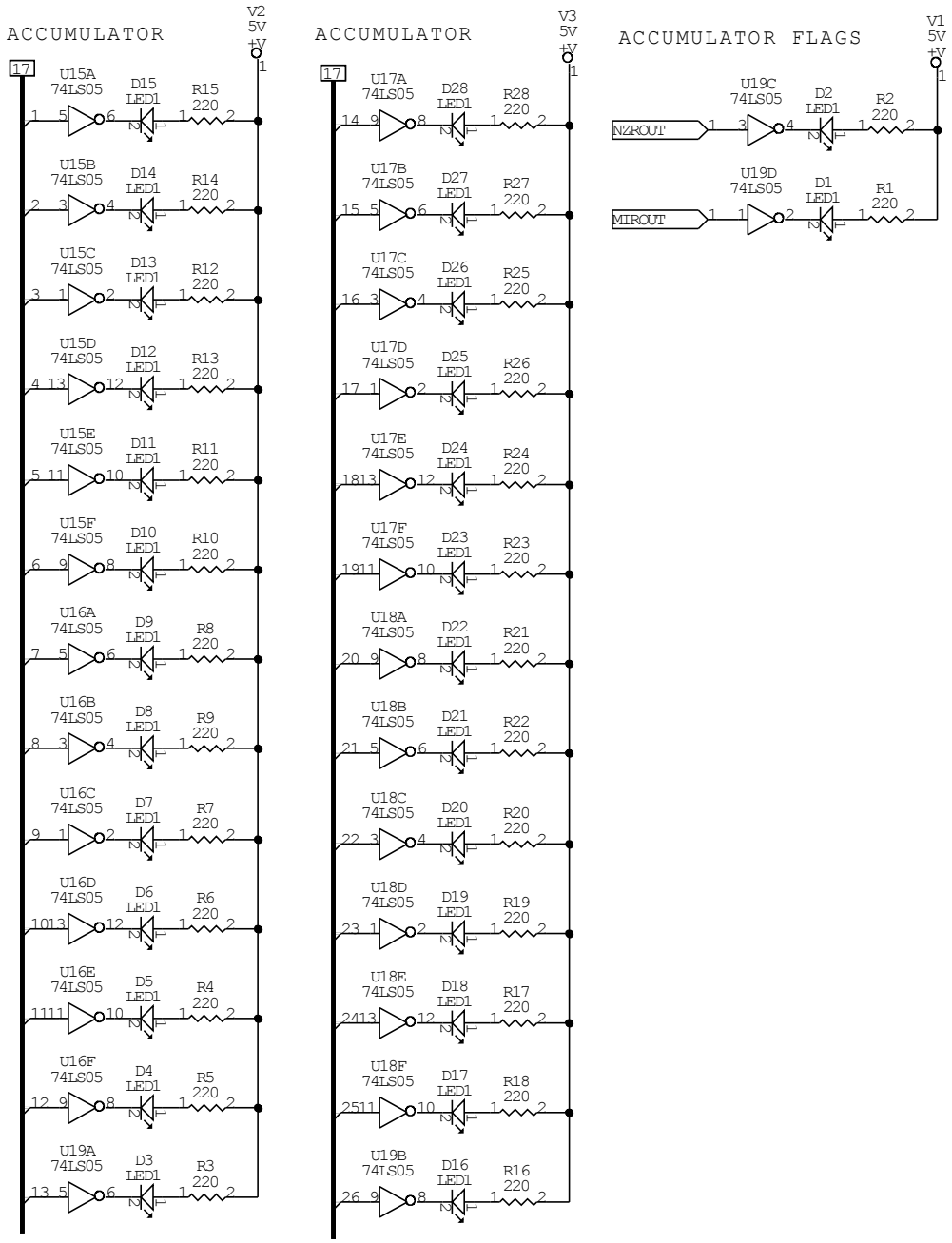
GEMINI ALU #4
ACC SHIFT LOGIC



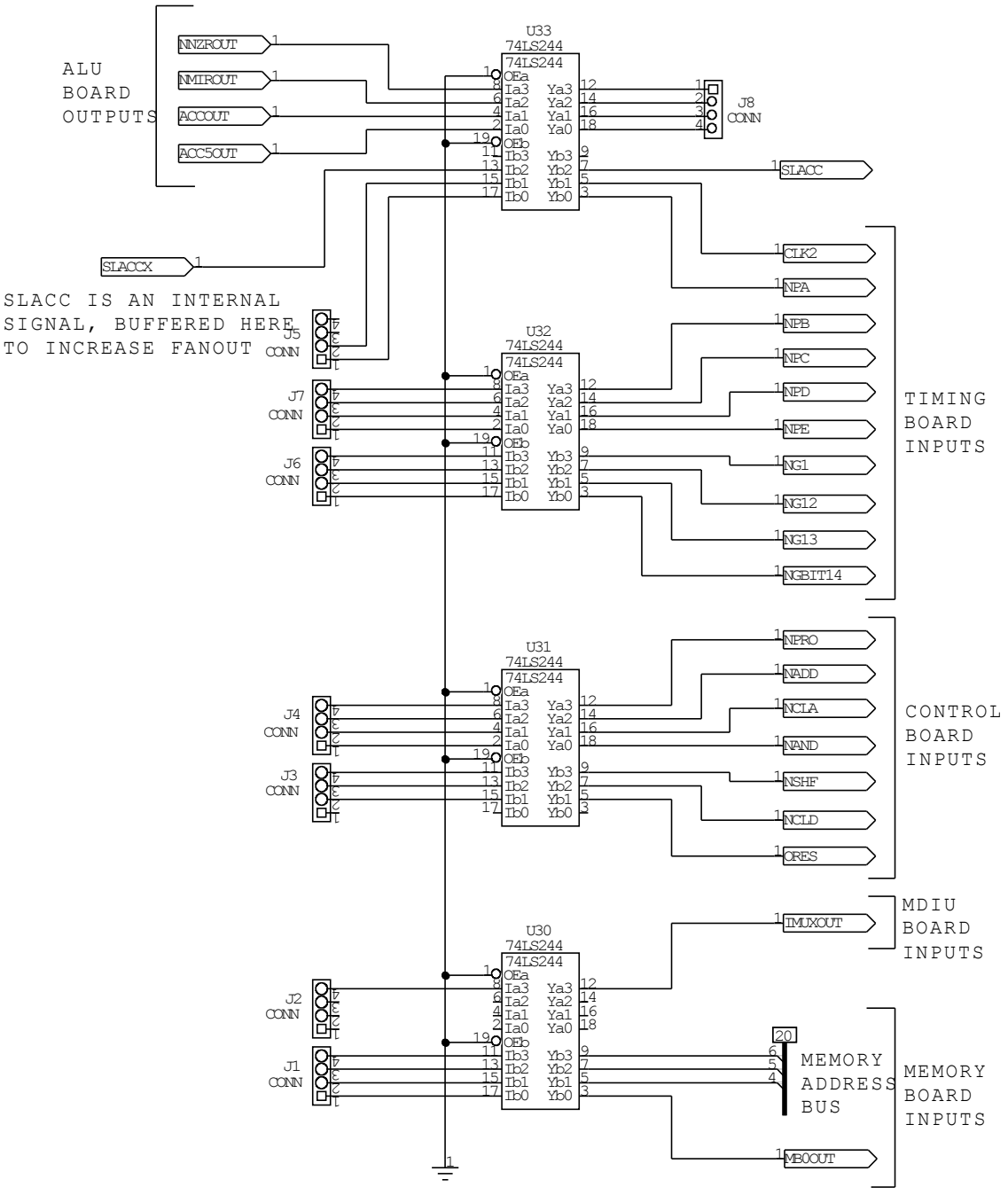
GEMINI ALU #5
ADDER AND CONDITIONAL FLAGS



GEMINI ALU #6
DISPLAYS



GEMINI ALU #7
EXTERNAL INTERFACES



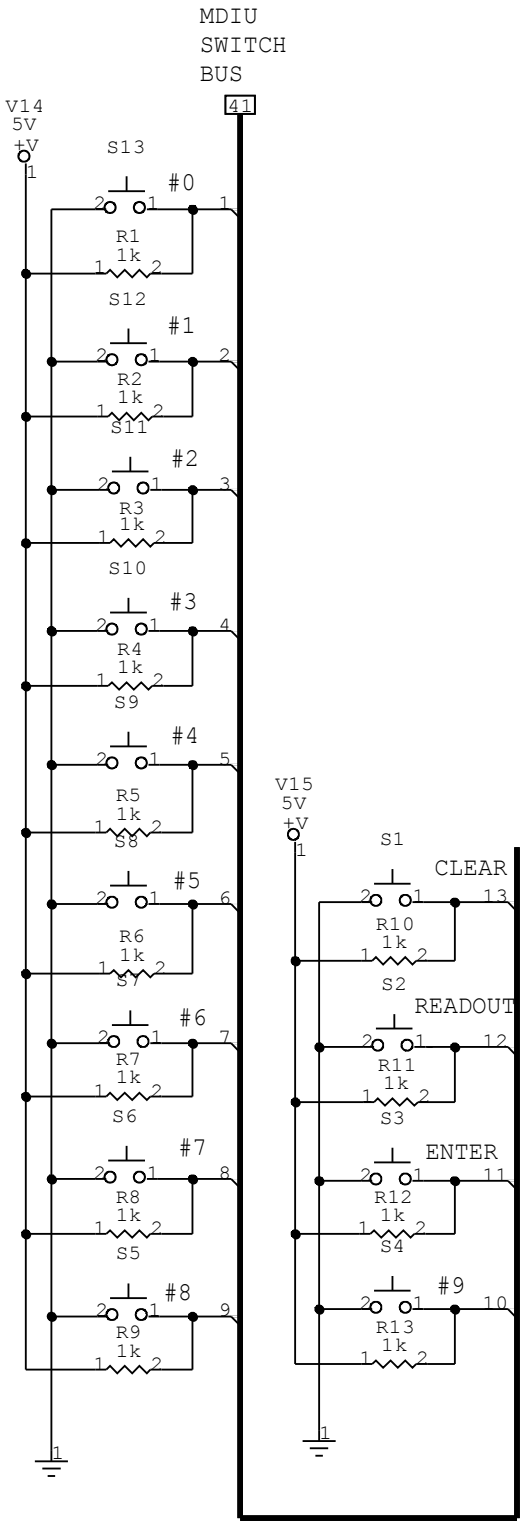
MDIU Board

TBS.

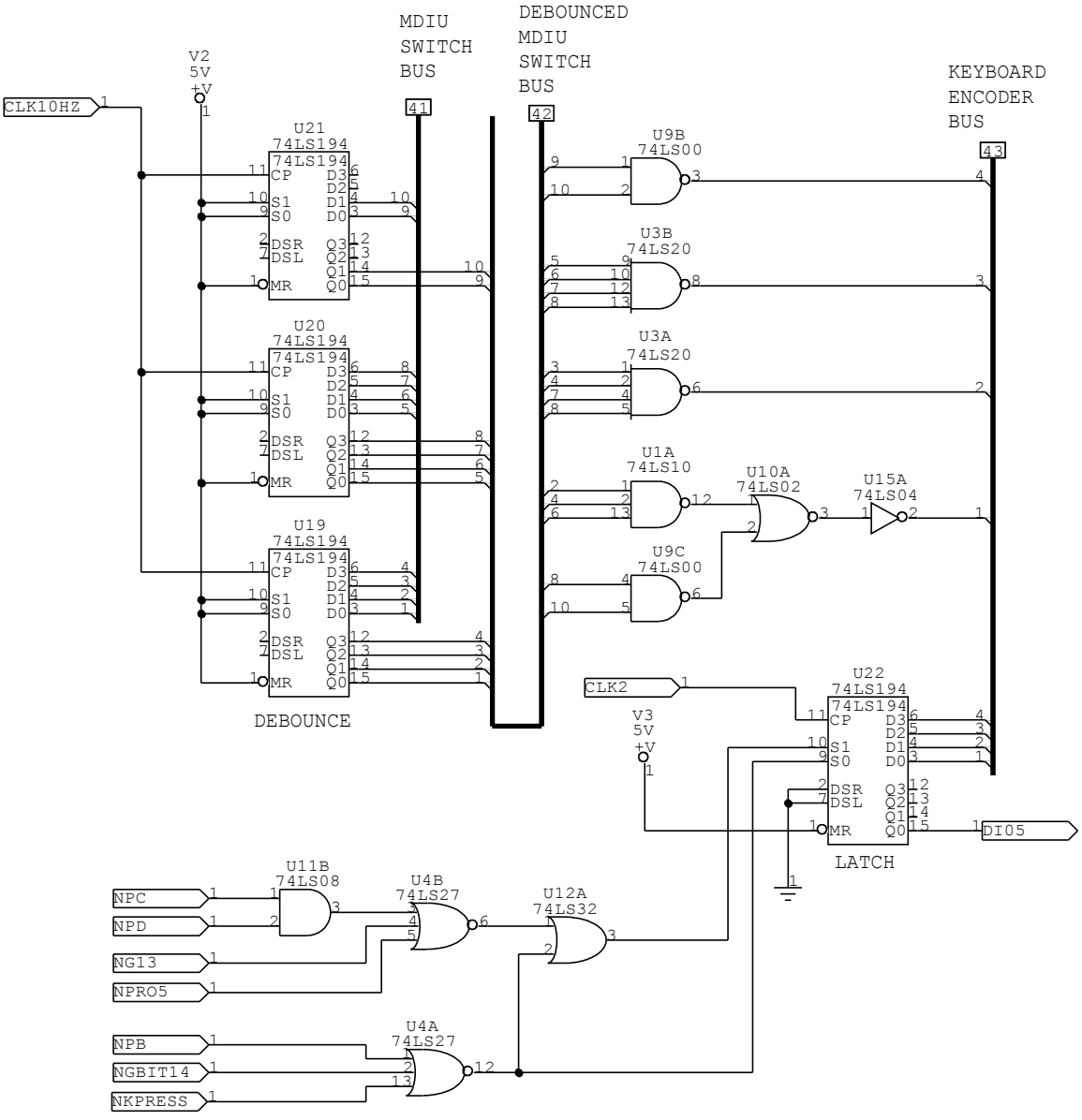
MDIU Board IC List

DESIGNATION	PART	QTY
U10:	74LS02	(1)
U12:	74LS32	(1)
U14,U4,U13,U2:	74LS27	(4)
U11:	74LS08	(1)
U16,U19,U20,U21,U22,U23,U24,U33:	74LS194	(8)
U17,U18:	74LS244	(2)
U9:	74LS00	(1)
U1:	74LS10	(1)
U15,U8:	74LS04	(2)
U3,U5:	74LS20	(2)
U25:	74LS154	(1)
U26,U27,U28,U29,U30,U31,U32:	74LS161A	(7)
U6,U7:	74LS109	(2)
U34:	74LS138	(1)
U35:	74LS151	(1)
U36,U37,U38,U39,U40,U41,U42:	74LS47	(7)

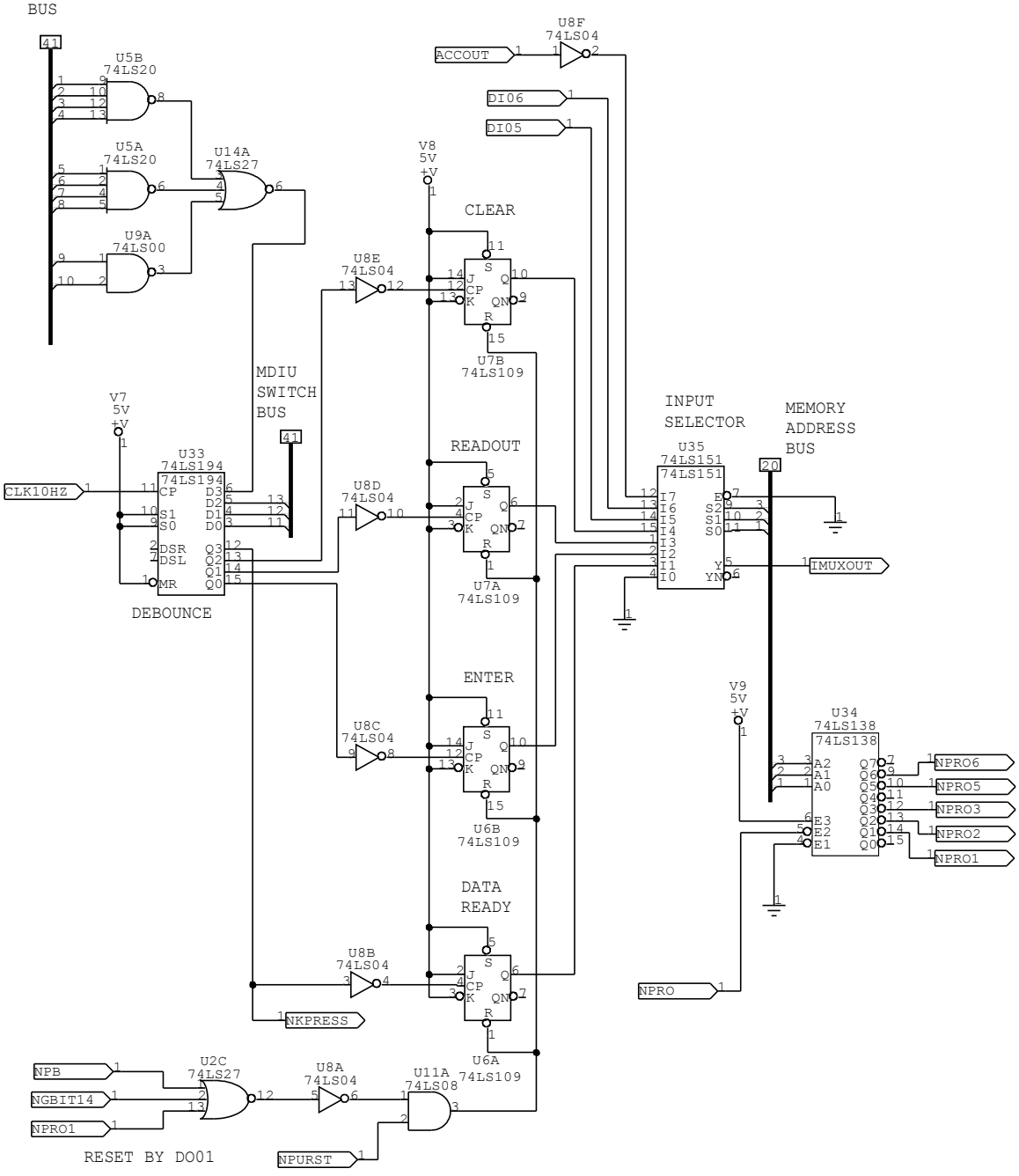
GEMINI MDIU #1
KEYBOARD



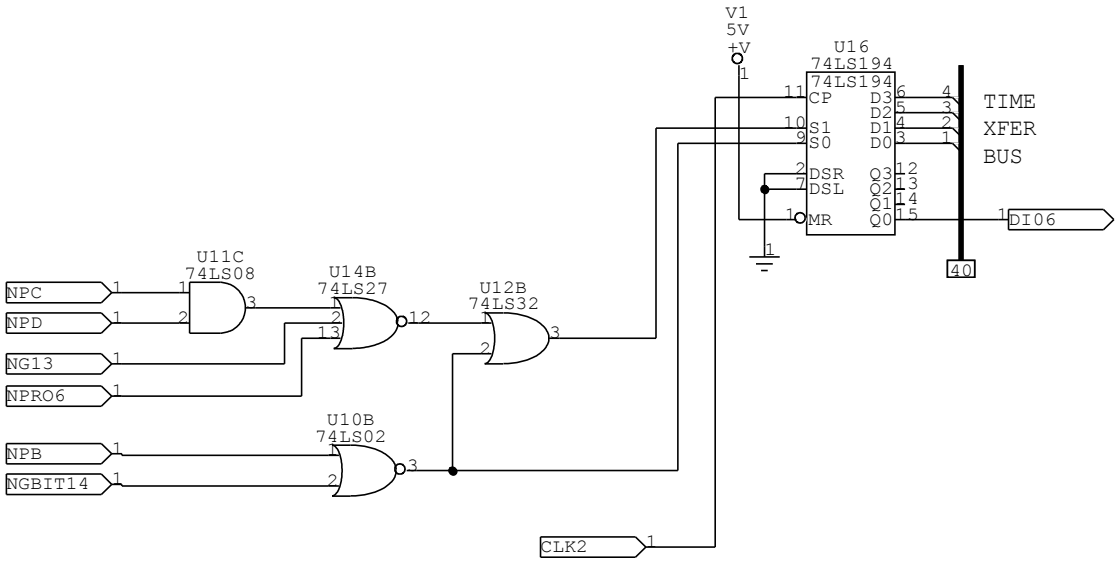
GEMINI MDIU #2
KEYBOARD ENCODER



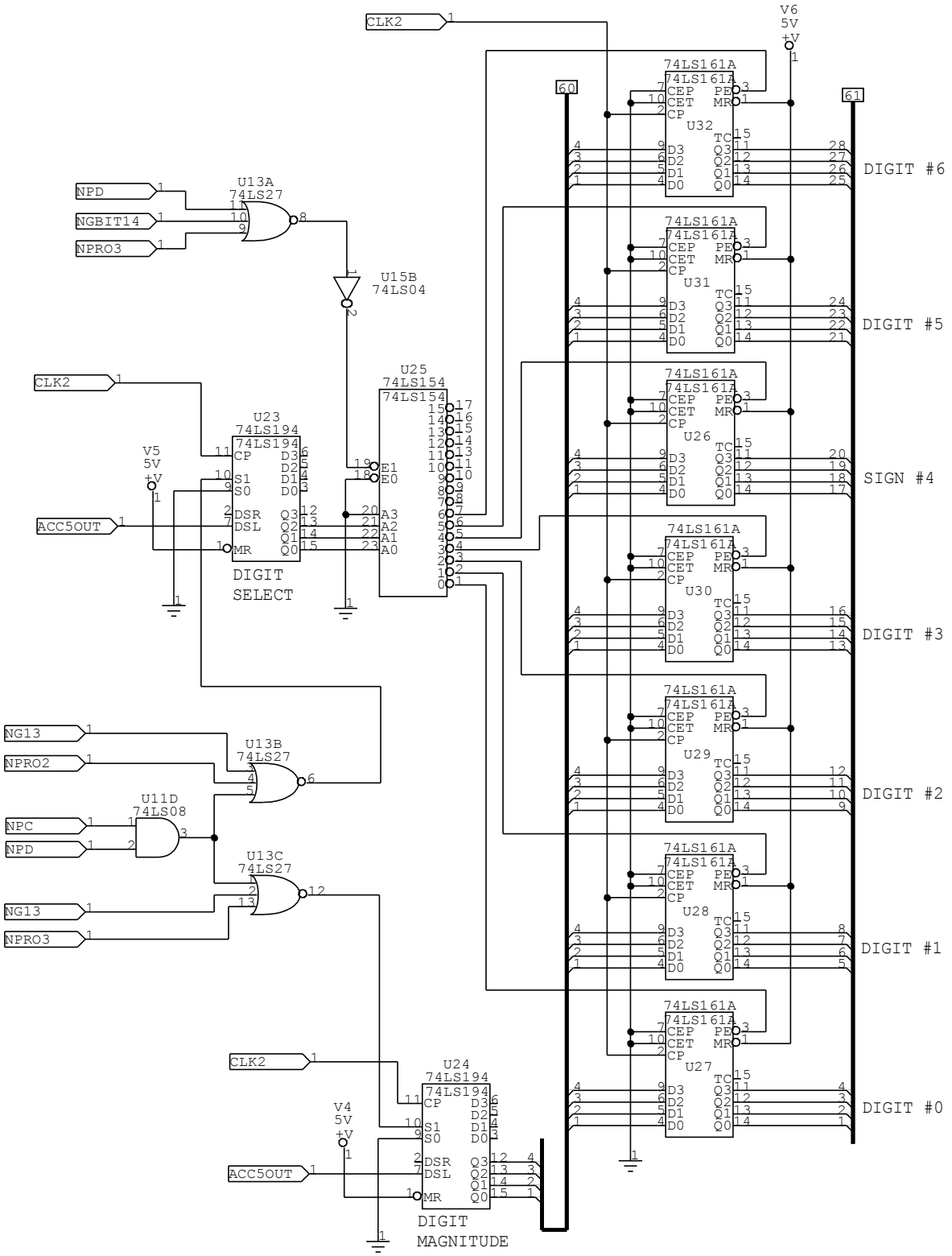
GEMINI MDIU #3
PUSHBUTTON LOGIC



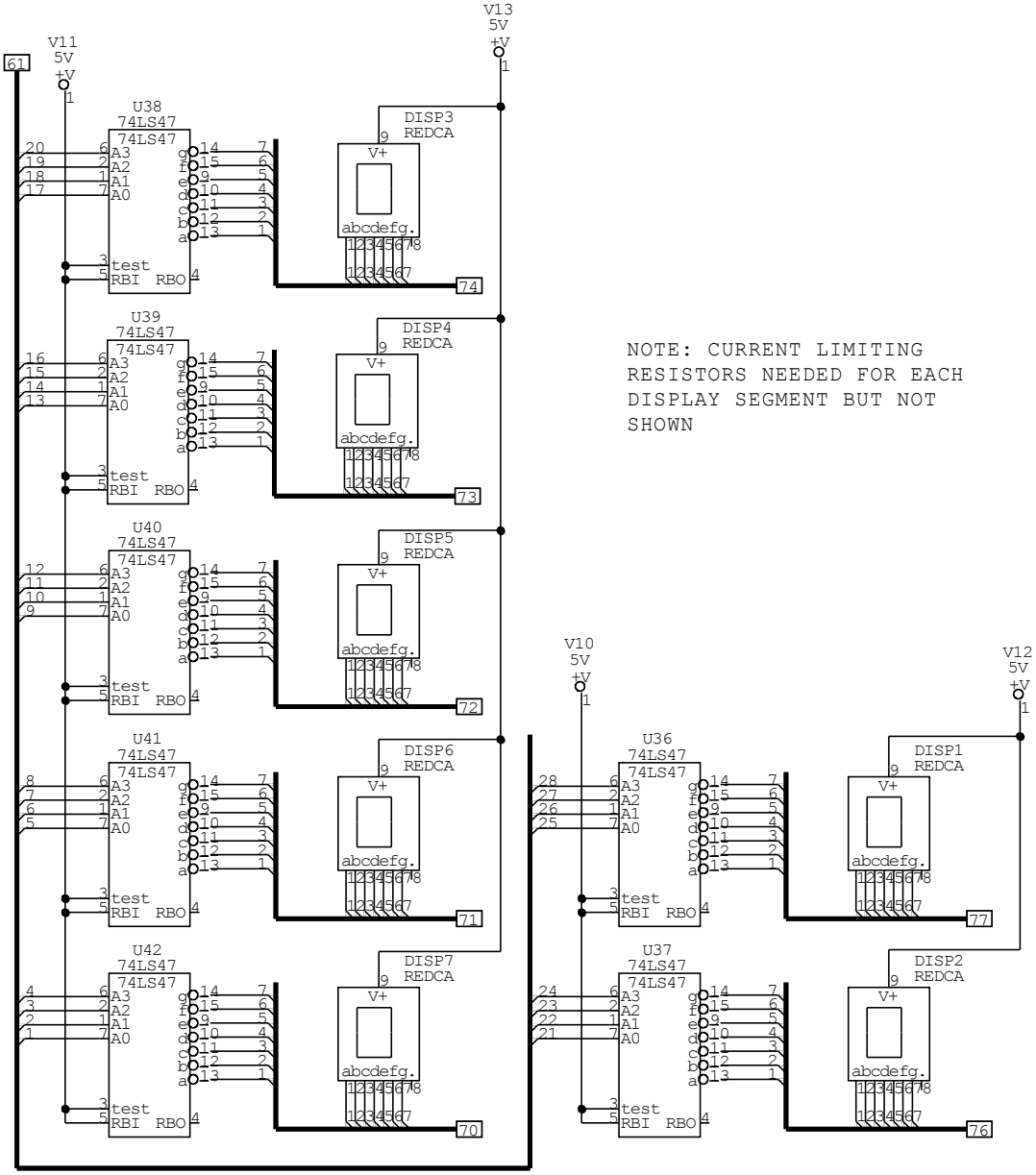
GEMINI MDIU #4
TIME ENCODER



GEMINI MDIU #5
DISPLAY LATCHES



GEMINI MDIU #6
DIGIT DISPLAYS



GEMINI MDIU #7
EXTERNAL INTERFACES

"TIME XFER BUS" 1-4 IS INPUT
TO THE MDIU BOARD
BUT NOT BUFFERED.