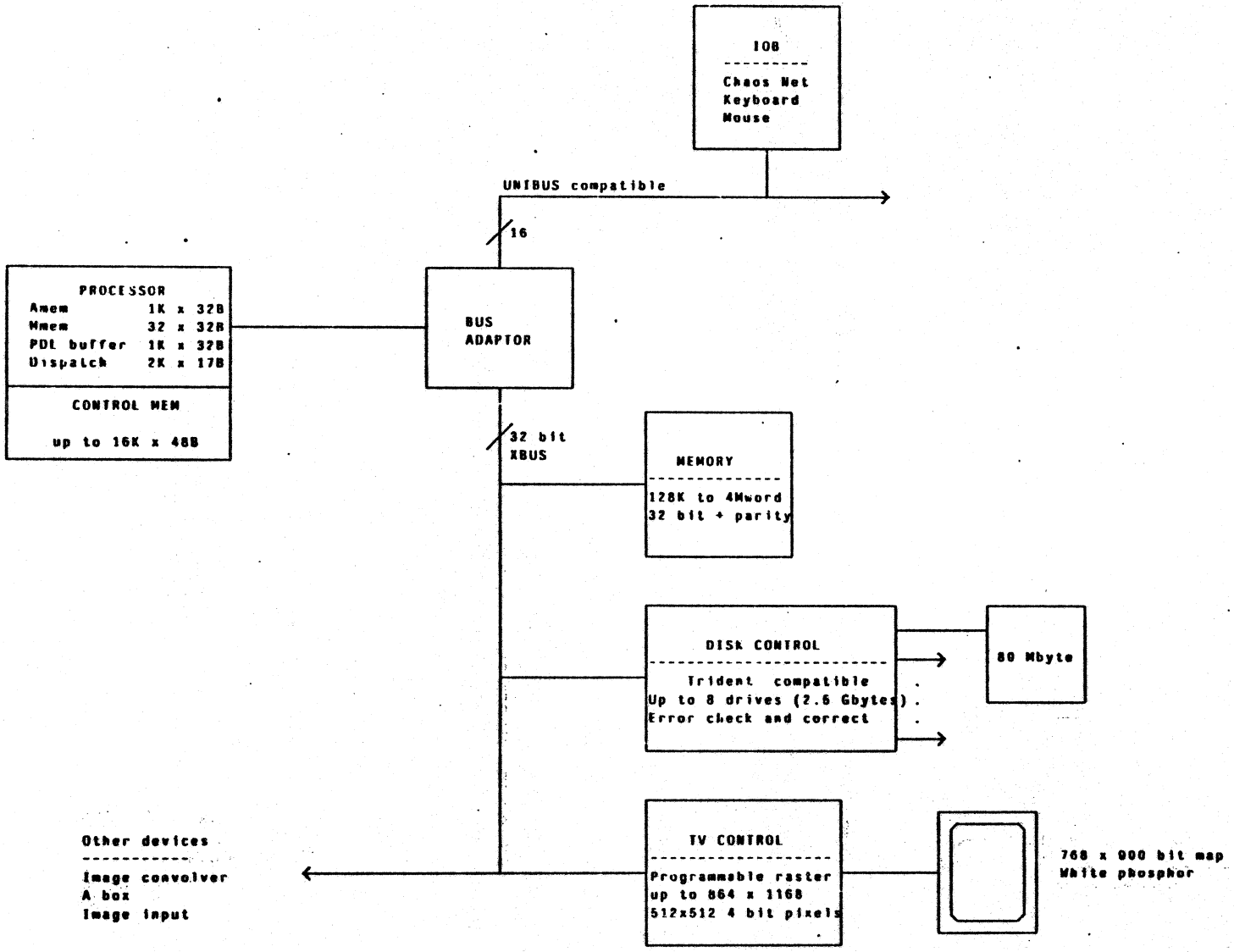
CADR

Thomas F. Knight, Jr.
David A. Moon
Jack Holloway
and Guy L. Steele, Jr.

The CADR machine, a revised version of the CONS machine, is a general-purpose, 32-bit microprogrammable processor which is the basis of the Lisp-machine system, a new computer system being developed by the Laboratory as a high-performance, economical implementation of Lisp. This paper describes the CADR processor and some of the associated hardware and low-level software.

Printed by Symbolics, Inc. under license from the Massachusetts Institute of Technology.



Lisp Machine	System Configuration	18-SEP-1981 08:42	AI: LMDOC; FIG1

Overview

The CADR microprocessor is a general purpose processor designed for convenient emulation of complex order codes, particularly those involving stacks and pointer manipulation. It is the central processor in the LISP machine project, where it interprets the bit-efficient 16-bit order code produced by the LISP machine compiler. (The terms "LISP machine" and "CADR machine" are sometimes confused. In this document, the CADR machine is a particular design of microprocessor, while the LISP machine is the CADR machine plus the microcode which interprets the LISP machine order code.)

The data paths of the CADR machine are 32 bits wide. Each 48-bit-wide microcode instruction specifies two 32-bit data sources from a variety of internal scratchpad registers; the two data-manipulation instructions can also specify a destination address. The internal scratchpads include a 1K pointer-addressable RAM intended for storing the top of the emulated stack, in a manner similar to a cache. Since in the LISP machine a large percentage of main memory references will be to the stack, this materially speeds up the machine.

The CADR machine has a 14-bit microprogram counter, which behaves much like that of a traditional processor, allowing up to 16K of writable microprogram memory. Also included is a 32-location microcode subroutine return stack.

Memory is accessed through a two-level virtual paging system, which maps 24-bit virtual addresses into 22-bit physical addresses.

There are four classes of micro-instructions. Each specifies two sources (A and M); the ALU and BYTE operations also specify a destination (A, or M plus functional). The A bus supplies data from the 1024-word A scratchpad memory, while the M bus supplies data from either the 32-word M scratchpad memory (a copy of the first 32 locations of the A scratchpad) or a variety of other internal registers. The four classes of microinstruction are:

ALU The destination receives the result of a boolean or arithmetic operation performed on the two sources.

BYTE The destination receives the result of a byte extraction, byte deposit, or selective field substitution from one source to the other. The byte so manipulated can be of any non-zero width.

JUMP A transfer of control occurs, conditional on the value of any bit accessible to the M bus, or on a variety of ALU and other internal conditions such as pending interrupts and page faults.

DISPATCH A transfer of control occurs to a location determined by a word from the dispatch memory selected by a byte of up to seven bits extracted from the M bus.

There are several sources and destinations whose loading and use invoke special action by the microprocessor. These include the memory address and memory data

registers, whose use initiates main memory cycles.

Some of the ALU operations are conditional, depending upon the low order bit in the Q register and the sign of A source. These operations are used for multiply and divide steps.

The main features of this machine which make it suitable for interpreting the LISP machine order code are its dynamically writable microcode, its very flexible dispatching and subroutining, its excellent byte manipulation abilities, and its internal stack storage. While the design of CADR was strongly influenced by the requirements of the LISP machine design, a conscious attempt was made to avoid features that are extremely special-purpose. The goal is a machine that happens to be good at interpreting the particular order code of the LISP machine, but which is general enough to interpret others almost as well. In particular, no critical parts of the LISP machine design (such as LISP machine instruction formats) are "wired in"; thus any changes to the LISP machine design can be easily accomodated by CADR. However, there are several "efficiency hacks" in the hardware, designed to speed up certain common operations of the LISP machine microcode, which might not be useful for other microcodes. These are described in later sections of this document.

Notational Conventions

All numbers used to describe bit positions, field widths, memory sizes, etc. are decimal. Octal is used only (and exclusively) to describe the values of fields. Bits within a word are consistently numbered from right to left, the least significant bit being bit <0>. Fields are described by the numbers of their most and least significant bits (e.g. "bits <22-10>").

Whenever a particular field value is described as "illegal", it does not mean that specifying that value will screw up the operation of the machine. It merely indicates a value which happens to have a certain function, not because it is considered directly useful, but because the internal workings of the machine may force certain selectors to that value for other reasons, and the user can select this value too even though it is not normally useful. These illegal values are described for the benefit of someone who may wish to fathom these inner workings.

A field value described as "unused" is reserved for possible design expansion and should not be used in programs. Bit fields described as "unused" should be zero in programs, for the sake of future compatibility.

Since the use of the term "micro" in referring to registers and instructions becomes redundant, its use will be dropped from here on in this part of the document. All instructions discussed are microinstructions.

The following bits are treated the same in every instruction. They will not be repeated in the individual instruction descriptions

IR<48> = Odd parity bit

IR<47> = Unused

IR<46> = Statistics (see the description of the Statistics Counter) This can be used to count how many times specified areas of the microcode are executed, to implement microcode breakpoints, or to stop the machine at a certain "time".

IR<45> = ILONG (1 means slow clock)

IR<44-43> = Opcode (0 ALU, 1 JUMP, 2 DISPATCH, 3 BYTE)

IR<42> = POPJ transfer. Causes a return from a micro subroutine, after executing one additional instruction.

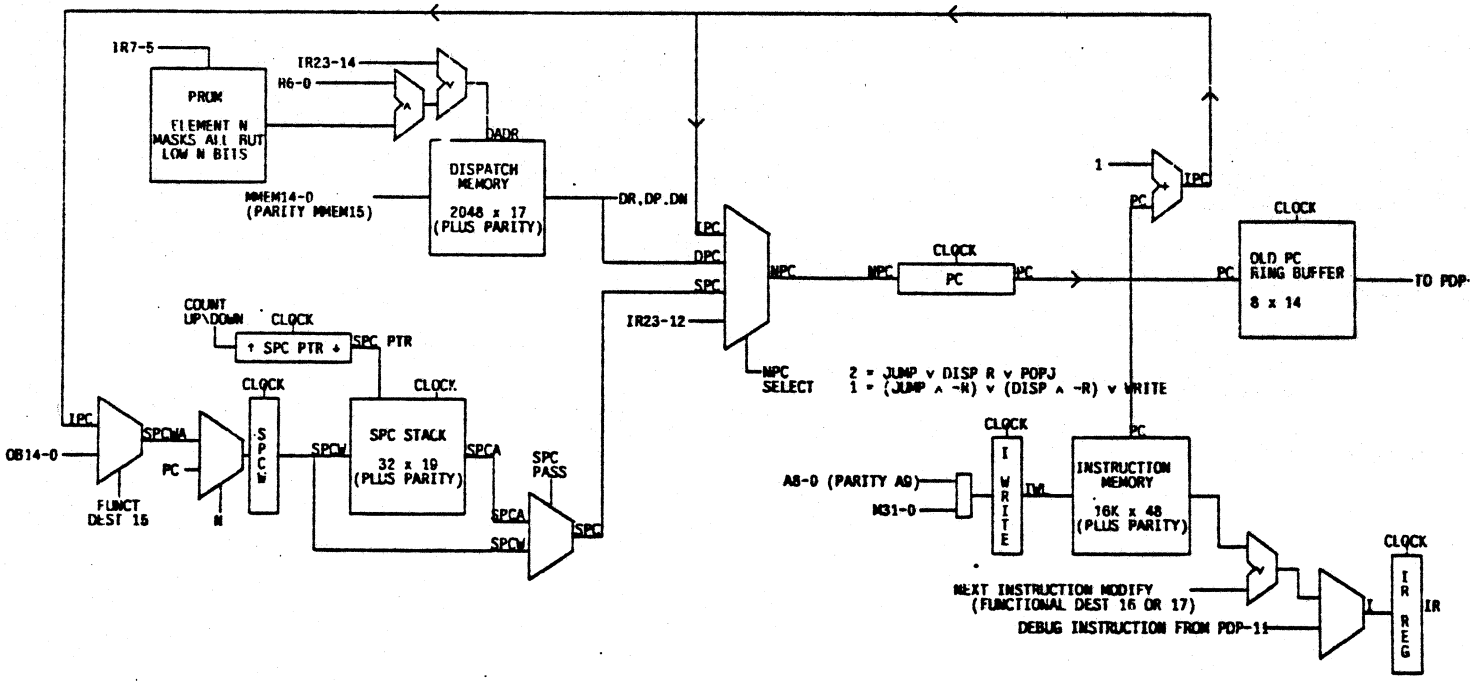
IR<11-10> = Miscellaneous Functions

0 Normal

1 Not used

2 Write dispatch memory, if opcode is DISPATCH.

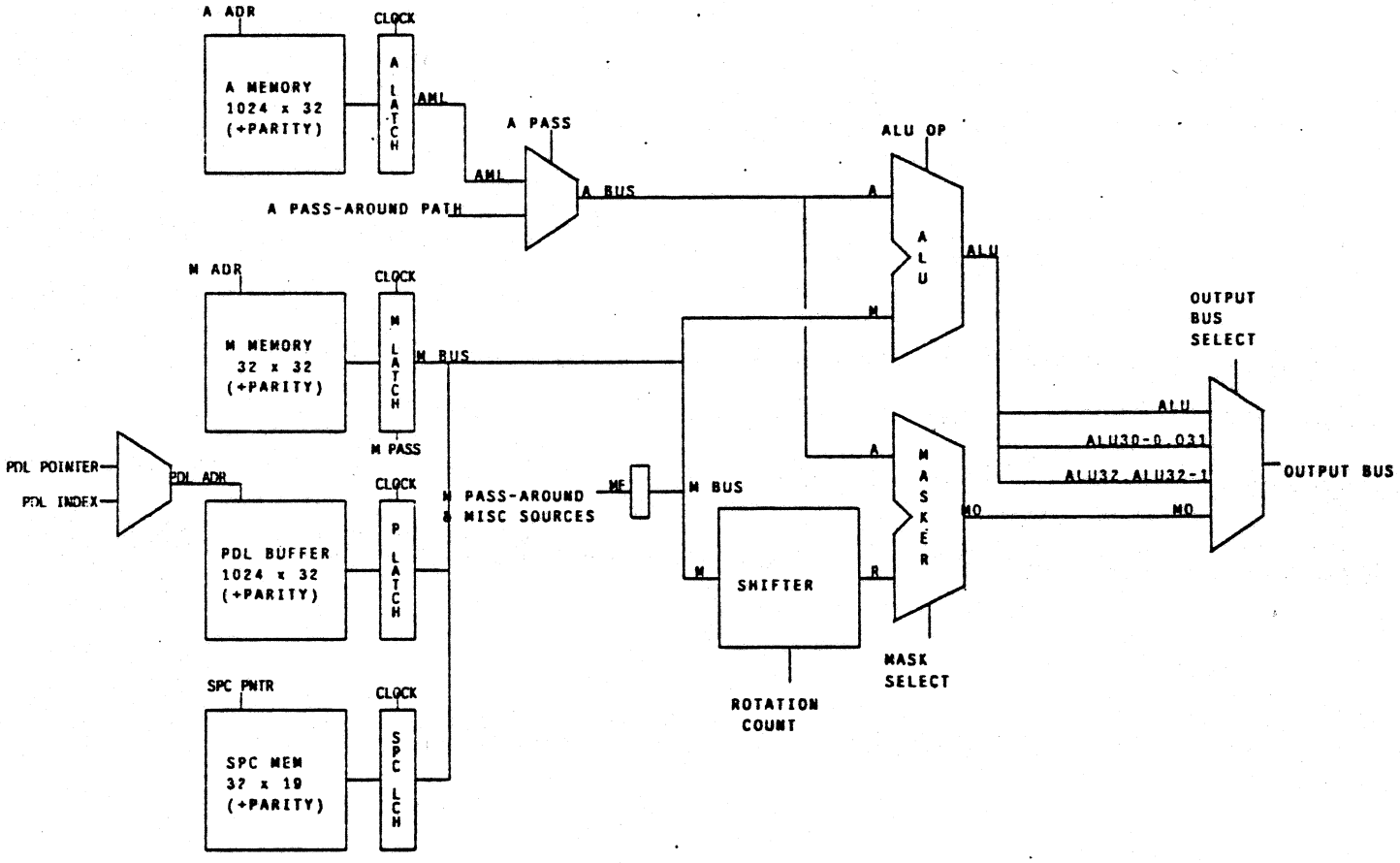
3 Enable modification of the M-ROTATE field by the location counter (LC). See the description of the instruction-stream hardware.



CADR Main Control Paths

Data Paths

The data paths of the machine consist of two source busses A and M, which provide data to the ALU and byte extractor, and an output bus OB, which is selected from the ALU (optionally shifted left or right) or the output of the byte extractor, and whose data can be routed to various destinations. We first describe the specification of the source busses, which are identically specified for all instructions; then the destination specifiers which control where results are stored; and finally the two instructions for controlling the ALU and the byte extractor.



CADR A and M Busses

Sources

All instructions specify sources in the same way. There are two source busses in the machine, the A bus and the M bus. The A bus is driven only from the A scratchpad memory of 1024 32-bit words. The M bus is driven from the M scratchpad of 32 32-bit words and a variety of other sources, including main memory data and control registers, the PC stack (for restoring the state of the processor after traps), the internal stack buffer and its pointer registers, the macrocode location counter, and the Q register. Addresses for the A and M scratchpads are taken directly from the instruction. The alternate sources of data for the M source are specified with an additional bit in the M source field.

```
IR<41-32> = A source address
IR<31-26> = M source address
        If IR<31> = 0,
                IR<30-26> = M scratchpad address
        If IR<31> = 1,
                IR<30-26> = M "functional" source
                                 0  Dispatch constant (see below)
                                 1  SPC pointer <28-24>, SPC data <18-0>
                                 2  PDL pointer <9-0>
                                 3  PDL index <9-0>
                                 5  PDL Buffer (addressed by Index)
                                 6  OPC registers (see below) <13-0>
                                 7  Q register
                                10  VMA register (memory address)
                                11  MAP[MD]
                                12  MD register (memory data)
                                13  LC (location counter)
                                14  SPC pointer and data, pop
                                24  PDL buffer, addressed by Pointer, pop
                                25  PDL buffer, addressed by Pointer
```

Functional sources not listed above should not be used and may have side effects. Sources 15, 16, and 17 are reserved for future expansion. Source 4 is the PDL buffer, indexed by the PDL Index, and the PDL pointer is decremented, presumably a useless operation.

Programming hint: it is often convenient to reserve one A memory word and one M memory word and fill them with constant zeros, to provide a zero source for each source bus. It is also convenient to have an M memory word containing all ones. These are particularly useful for byte extraction, masking, bit setting, and bit clearing operations. The CONSLP assembler in fact assumes that A memory location 2 and M memory location 2 are sources of zeros. The UCONS microcode stores all ones in location 3.

The M scratchpad normally contains a duplicate copy of the first 32 locations of the A scratchpad. The effect is as if there were a single scratchpad memory, the first 32 locations of which were dual-ported. This makes programming more convenient, since

these locations are accessible to both sides of the ALU and shifter.

Destinations

The 12-bit destination field in the BYTE and ALU instructions specifies where the result of the instruction is deposited. It is in one of two forms, depending upon the high-order bit. If the high-order bit is 1, then the low 10 bits are the address of an A memory location, and the remaining bit is unused. If the high order bit is 0, the low 10 bits are divided into a 5-bit "functional destination" field, and a 5-bit M scratchpad address, and both of the places specified by these fields get written into. The next-to-highest bit in the destination field is not used.

```
IR<25-14> = Destination
        If IR<25> = 1,
                IR<23-14> = A scratchpad write address
        If IR<25> = 0,
                IR<23-19> = Functional destination write address
                         0  None
                         1  LC (Location Counter)
                         2  Interrupt Control <29-26>
                                Bit 26 = Sequence-Break request
                                Bit 27 = Interrupt-Enable
                                Bit 28 = Bus-Reset
                                Bit 29 = LC Byte-mode
                        10  PDL (addressed by Pointer)
                        11  PDL (addressed by Pointer), push
                        12  PDL (addressed by Index)
                        13  PDL Index
                        14  PDL Pointer
                        15  SPC data, push
                        16  Next instruction modifier
                                ("OA register"), bits <25-0>
                        17  Next instruction modifier
                                ("OA register"), bits <47-26>
                        20  VMA register (memory address)
                        21  VMA register, start main memory read
                        22  VMA register, start main memory write
                        23  VMA register, write map.  The map is
                                addressed from MD and written from
                                VMA.  VMA<26>=1 writes the level 1
                                map from VMA<31-27>.  VMA<25>=1 writes
                                the level 2 map from VMA<23-0>.
                        30  MD register (memory data)
                        31  MD register, start main memory read
                        32  MD register, start main memory write
                        33  MD register, write map like 23
                IR<18-14> = M scratchpad write address
```

Functional destinations not listed may have strange results. Destinations 3-7 are reserved for expansion.

Note: If you write into the M-memory, the machine will also write into the corresponding A-memory address. Therefore you should never write into A-memory locations 0-37; this way the first 40 (octal) locations of A-memory "map into" the M-memory.

The full details of the more complicated functional destinations are described in later sections below. The Q register is loaded by using the Q-control field of the ALU instruction, not by using a functional destination. In addition, it loads from the ALU outputs, not the output bus. This means that the left and right shift operations are ineffective for data being loaded into Q.

Programming hint: if a functional destination is specified, an M scratchpad location must also be specified. It is convenient to reserve one location of the M scratchpad for "garbage"; this location can be specified when it is desired to write into a functional destination but not into any other M scratchpad location. Since the CONSLP assembler defaults the M write address to zero, it is best to let location 0 be the garbage location. Location 0 of the A scratchpad will also be written and is also reserved as a garbage location.

The ALU Instruction

The ALU operation performs most of the arithmetic in the machine. It specifies two sources of 32 bit numbers, and an operation to be performed by the ALU. The operation can be any of the 16 boolean functions on two variables, two's complement addition or subtraction, left shift, and several less useful operations. The carry into the ALU can be forced to be 0 or 1. The output of the ALU is optionally shifted one place, and then written into the specified destinations via the output bus. Additionally, the ALU instruction specifies one of four operations upon the Q register. These are do nothing, shift left, shift right, and load from the ALU outputs. An additional bit in the ALU operation field is decoded to indicate conditional operations; this is how the "multiply step" and "divide step" operations are specified. (Multiplication and division are explained in greater detail in another section.)

```
IR<44-43> = 0 (ALU opcode)
IR<41-32> = A source
IR<31-26> = M source
IR<25-14> = Destination
IR<13-12> = Output bus control
                0  Byte extractor output (illegal)
                1  ALU output
                2  ALU output shifted right one, with the correct
                   sign shifted in, regardless of overflow.
                3  ALU output shifted left one, shifting in Q<31>
                   from the right.
IR<9>     = not used
IR<8-4>   = ALU operation
                If IR<8> = 0,
                        IR<7-3> = ALU op code (see table)
                If IR<8> = 1,
                        IR<7-3> = Conditional ALU op code
                                  0  Multiply step
                                  1  Divide step
                                  5  Remainder correction
                                 11  Initial divide step
IR<2>     = Carry into low end of ALU
IR<1-0>   = Q control
                0  Do nothing
                1  Shift Q left, shifting in the inverse
                       of the sign of the ALU output (ALU<31>)
                2  Shift Q right, shifting in the low bit
                       of the ALU output (ALU<0>)
                3  Load Q from ALU output
```

ALU operation codes (from Table 1 of 74181 specifications). All arithmetic operations are two's complement. Note that the bits are permuted in such a way as to make the logical operations come out with the same opcodes as used by the Lisp BOOLE function. Names in square brackets are the CONSLP mnemonics for the operations.

	Boolean (IR<7>=1)		Arithmetic (IR<7>=0)			
IR<6-3>			Carry in = 0		Carry in = 1	
0	ZEROS	[SETZ]	-1		0	
1	M∧A	[AND]	(M∧A)-1		M∧A	
2	M∧¬A	[ANDCA]	(M∧¬A)-1		(M∧¬A)	
3	M	[SETM]	M-1		M	
4	¬M∧A	[ANDCM]	M∨¬A		(M∨¬A)+1	
5	A	[SETA]	(M∨¬A)+(M∧A)		(M∨¬A)+(M∧A)+1	
6	M⊕					
A	[XOR]	M-A-1	[M-A-1]	M-A	[SUB]	
7	M∨A	[IOR]	(M∨¬A)+M		(M∨¬A)+M+1	
10	¬A∧¬M	[ANDCB]	M∨A		(M∨A)+1	
11	M≡A	[EQV]	M+A	[ADD]	M+A+1	[M+A+1]
12	¬A	[SETCA]	(M∨A)+(M∧¬A)		(M∨A)+(M∧¬A)+1	
13	M∨¬A	[ORCA]	(M∨A)+M		(M∨A)+M+1	
14	¬M	[SETCM]	M		M+1	[M+1]
15	¬M∨A	[ORCM]	M+(M∧A)		M+(M∧A)+1	
16	¬M∨¬A	[ORCB]	M+(M∨¬A)		M+(M∨¬A)+1	
17	ONES	[SETO]	M+M	[M+M]	M+M+1	[M+M+1]

The BYTE Instruction

The BYTE instruction specifies two sources and a destination in the same way as the ALU instruction, but the operation performed is one of selective insertion of a byte field from the M source into an equal length field of the word from the A source. The rotation of the M source is specified by the SR bit as either zero or equal to the contents of the ROTATE field. The rotation of the mask used to select the bits replaced is specified by the MR bit as either zero or equal to the contents of the ROTATE field. The length of the mask field used for replacement is specified in the LENGTH MINUS 1 field. The four states of the SR and MR bits yield the following operations:

MR=0 SR=0 Not useful (This is a subset of other modes.)

MR=0 SR=1 LOAD BYTE PDP-10 LDB instruction (except the unmasked bits are from the A source). A byte of arbitrary position from the M source is right-justified in the output.

MR=1 SR=0 SELECTIVE DEPOSIT The masked field from the M source is used to replace the same length and position byte in the word from the A source.

MR=1 SR=1 DEPOSIT BYTE PDP-10 DPB instruction. A right-justified byte from the M source is used to replace a byte of arbitrary position in the word from the A source.

The BYTE instruction automatically makes the output of the byte extractor available by forcing the output bus select code to 0 (byte extractor output).

```
IR<44-43> = 3 (BYTE operation)
IR<41-32> = A source
IR<31-26> = M source
IR<25-14> = Destination
IR<13>    = MR = Mask Rotate (see above)
IR<12>    = SR = Source Rotate (see above)
IR<9-5>   = Length of byte minus 1 (0 means byte of length 1, etc.)
IR<4-0>   = Rotation count (to the left) of mask and/or M source
```

The byte operation rotates the M source by 0 (if SR=0) or by the rotation count (if SR=1), producing a result called R. It also uses the MR bit, the rotation count, and the length minus 1 field to produce a selector mask (see description below). This mask is all zeros except for a contiguous section of ones denoting the selected byte. This mask is used to merge the A source with R, bit by bit, selecting a bit from A if the mask is 0 and from R if the mask is 1. This result is then written into the specified destination(s).

Output of mask memories:

Right mask memory is indexed by 0 (MR=0) or by rotation count (MR=1).
Left mask memory is indexed by (the index into right mask memory) plus (the length minus 1 field), mod 32.

octal index	LEFT MASK MEMORY contents	RIGHT MASK MEMORY contents
0	00000000000000000000000000000001	11111111111111111111111111111111
1	00000000000000000000000000000011	11111111111111111111111111111110
2	00000000000000000000000000000111	11111111111111111111111111111100
3	00000000000000000000000000001111	11111111111111111111111111111000
4	00000000000000000000000000011111	11111111111111111111111111110000
5	00000000000000000000000000111111	11111111111111111111111111100000
6	00000000000000000000000001111111	11111111111111111111111111000000
7	00000000000000000000000011111111	11111111111111111111111110000000
10	00000000000000000000000111111111	11111111111111111111111100000000
11	00000000000000000000001111111111	11111111111111111111111000000000
12	00000000000000000000011111111111	11111111111111111111110000000000
13	00000000000000000000111111111111	11111111111111111111100000000000
14	00000000000000000001111111111111	11111111111111111111000000000000
15	00000000000000000011111111111111	11111111111111111110000000000000
16	00000000000000000111111111111111	11111111111111111100000000000000
17	00000000000000001111111111111111	11111111111111111000000000000000
20	00000000000000011111111111111111	11111111111111110000000000000000
21	00000000000000111111111111111111	11111111111111100000000000000000
22	00000000000001111111111111111111	11111111111111000000000000000000
23	00000000000011111111111111111111	11111111111110000000000000000000
24	00000000000111111111111111111111	11111111111100000000000000000000
25	00000000001111111111111111111111	11111111111000000000000000000000
26	00000000011111111111111111111111	11111111110000000000000000000000
27	00000000111111111111111111111111	11111111100000000000000000000000
30	00000001111111111111111111111111	11111111000000000000000000000000
31	00000011111111111111111111111111	11111110000000000000000000000000
32	00000111111111111111111111111111	11111100000000000000000000000000
33	00001111111111111111111111111111	11111000000000000000000000000000
34	00011111111111111111111111111111	11110000000000000000000000000000
35	00111111111111111111111111111111	11100000000000000000000000000000
36	01111111111111111111111111111111	11000000000000000000000000000000
37	11111111111111111111111111111111	10000000000000000000000000000000

After the two masks are selected, they are AND'ed together to get the final mask. This mask is all zeros, except for a field of contiguous ones defining the byte.

As an example, if MR=1, rotation count=5, and length minus 1=7, then the right mask index is 5 and the left mask index is 14 (octal). This results in a final mask as follows:

```
        Right mask 5    11111111111111111111111111100000
        Left mask 14    00000000000000000001111111111111
AND them together       --------------------------------
        Final mask      00000000000000000001111111100000
```

The byte is 8 bits wide, 5 positions from the right.

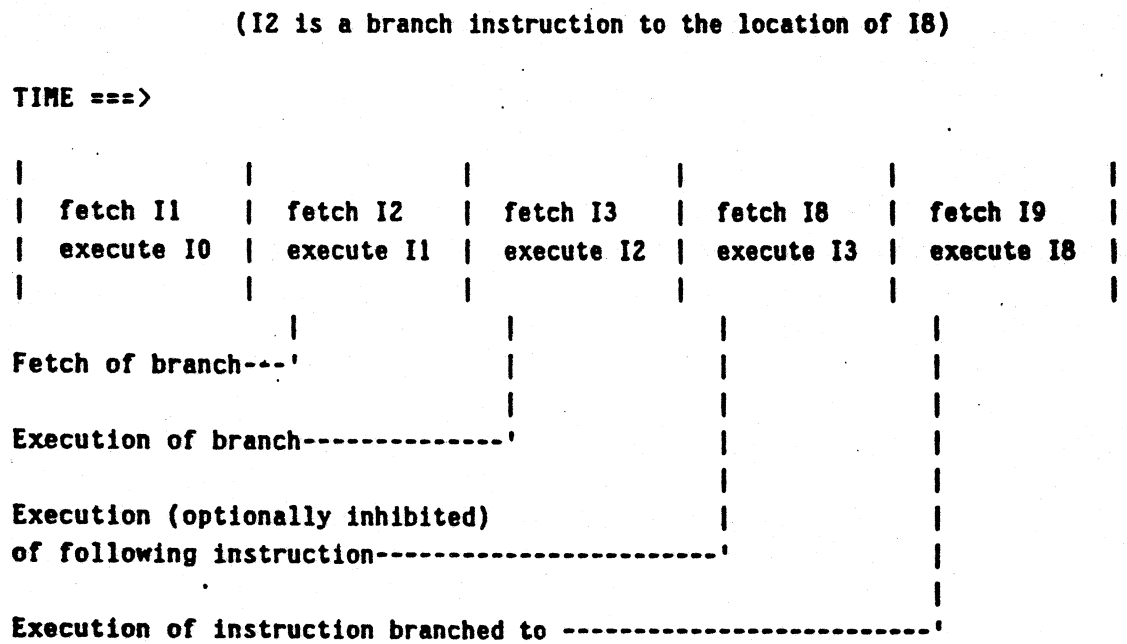
Programming hint: if the byte is "too large" (i.e. its position and size specifications cause it to hang over the left-hand edge of a word), then the masker does not truncate the byte at the left-hand edge. Instead, it produces a zero mask, selecting no byte at all; thus, the output of the byte operation equals the A source. The reason for this is that an overflow occurs in calculating the index into the left mask memory, and so the final mask is zero. For example, if MR=1, rotation count=20 (octal), and length minus 1=27 (octal), then the right mask index is 20 and the left mask index is 477 (mod 32). This results in a final mask as follows:

```
        Right mask 20   11111111111111110000000000000000
        Left mask 7     00000000000000000000000011111111
AND them together       --------------------------------
        Final mask      00000000000000000000000000000000
```

Control

The control section of the processor consists of a 14-bit program counter (the PC), a 32-location PC stack (SPC) and stack pointer (SPCPTR), and a 2K dispatch memory, used during the DISPATCH instruction. Unlike some microprocessors, and like most traditional machines, the normal mode of operation is to execute the next sequential instruction by incrementing the PC.

The processor uses single instruction look ahead, i.e. the lookup of the next instruction is overlapped with execution of the current one. This implies that after branching instructions the processor normally executes the following instruction, even if the branch was successful. Provision is made in these instructions to inhibit this execution (with the N bit), but the cycle it would have used will then be wasted.



Two types of instruction affect flow of control in the machine. The conditional JUMP specifies a new PC and transfer type in the instruction itself, while the DISPATCH instruction looks up the new PC and transfer type in the 2K dispatch memory. In either case, the new PC is loaded into the PC register, and the operation specified by the 3-bit transfer type is performed. These operations are:

N bit If on, inhibits execution of the next instruction, i.e. the instruction at the address one greater than that of the transfer instruction. (This instruction needn't actually be at the address one greater, if a transfer of control was

already in progress.) The cycle that would have executed that instruction is wasted.

The P and R bits are decoded as follows:

P=0 R=0	BRANCH	Normal program transfer.
P=1 R=0	CALL	Save the correct return address on the SPC stack, and jump to the new PC address.
P=0 R=1	RETURN	Ignore new PC; instead pop PC off the SPC stack.
P=1 R=1	FALL THROUGH	In a DISPATCH instruction, do not dispatch.
	I-MEM WRITE	In a JUMP instruction, write into the instruction memory, and do not jump.

The BRANCH transfer type is the normal program transfer, without saving a return address.

The CALL transfer type pushes the appropriate return address onto the SPC stack. This stack is 32 locations long. It is the responsibility of the programmer to avoid overflows. The return address is PC+2, or PC+1 if the N bit is also on. Actually, if the N bit is on the address of the instruction NOP'ed is saved, which may not be identical to PC+1 if a transfer of control is already in progress. If the N bit is not on, 1 + the address of that instruction is saved. In the case of a dispatch, if the N bit is on and bit 25 of the instruction is on, save PC, the address of the dispatch instruction itself; this allows the dispatch to be re-executed upon return. (Actually, due to pipelining, when the above paragraph says PC it doesn't really mean PC.)

The RETURN transfer type pops a return PC from the SPC stack, ignoring the PC specified in the instruction or dispatch table.

The FALL THROUGH transfer type for dispatches allows some entries in a dispatch table to specify that the dispatch should not occur after all. The following instruction is executed (unless inhibited), followed by the one after that (unless the first following one branches and inhibits it!).

The I-MEM WRITE transfer type is the mechanism for writing instructions into the microprogram instruction memory, and is described in a later section. (The dispatch memory, unlike the instruction memory, is <u>not</u> written into by setting the P and R bits (after all, in a dispatch instruction these bits come from the dispatch memory!); instead, the Miscellaneous Function field is used.)

An additional bit in every instruction, including ALU and BYTE instructions, called the POPJ bit, allows specification of simultaneous execution of a RETURN transfer type along with execution of any instruction. That is, it does the same thing as if this instruction, in addition to whatever else it does, had executed a RETURN transfer type jump without the N bit on. It is the responsibility of the programmer to avoid conflicts in the use of this bit simultaneously with other types of transfers.

The POPJ bit should be used in a JUMP instruction only in conjunction with the RETURN transfer type. This will cause a RETURN operation in either case, but execution of the following instruction is conditional, controlled by the N bit and the conditional JUMP instruction. The POPJ bit, when used in a DISPATCH instruction, is

specially over-ruled by the JUMP and CALL transfer types. This allows you to RETURN normally, but jump off to other code in exceptional cases, using the same dispatch table as other dispatch instructions which do not want to return. The POPJ bit should not be used in conjunction with writing of dispatch or instruction memory, nor with the SPC pop and push functional source and destination. The machine doesn't bother to do anything reasonable in these cases.

The DISPATCH Instruction

The dispatch instruction allows selection of any source available on the M bus [see description of M bus sources in the Data Path section], and the dispatch on any subfield of up to 7 bits from the selected word. The selected subfield is ORed with the "dispatch address" field of the instruction to produce an 11 bit address. This address is used to look up a 14 bit PC and 3 bit transfer type in the dispatch memory. The SPC-pointer-and-data-pop source will not operate reasonably in conjunction with the dispatch instruction.

```
IR<44-43> = 2 (DISPATCH operation)
IR<41-32> = Dispatch constant (also A source when writing D-MEM)
IR<31-26> = M source
IR<25>    = Alter return address pushed on SPC by the CALL transfer
            type, if the N bit is set, to be the address of this
            instruction rather than the next instruction.
IR<24>    = Enable instruction-stream hardware (described later).
IR<23>    = Unused
IR<22-12> = Address in dispatch memory
IR<9-8>   = Control dispatching off the map, see below.
IR<7-5>   = Length of byte (not minus 1!) from M source to dispatch on
IR<4-0>   = Rotation count (to the left) of M source
```

The dispatch operation takes the specified M source word and rotates it to the left as specified by the rotation count. All but the low K bits are masked out, where K is the contents of the length field. The result is OR'ed with the dispatch address, and this is used to address the 2K dispatch memory, which supplies the new PC and the R, P, and N bits.

If bits 8 and 9 of IR are not zero, the bottom bit of the dispatch address comes from the virtual memory map rather than the rotator and masker. The address inputs to the map in this case come from MD. This is primarily useful for testing pointers just fetched from main memory for validity with respect to the garbage collector's conventions. IR<8> selects bit 14 of the second level map, and IR<9> selects bit 15. Selecting both bits ORs them together.

The dispatch constant field is loaded into the DISPATCH CONSTANT register on every dispatch instruction. This register is accessible as an M source. The dispatch constant field has nothing whatsoever to do with the operation of dispatching; it is merely a convenient device for loading a completely random register while doing something else. (Uses for this feature are discussed in a later section.)

Miscellaneous function 2 inhibits the normal action of the instruction and instead loads the dispatch memory with the low order contents of the A memory scratchpad location specified in the A source. Note that the A source address is the same field as the dispatch constant field. The dispatch constant is loaded anyway, but this can be ignored. The parity bit (bit 17) is also loaded, and it is the responsibility of the

programmer to load correct (odd) parity into the memory. Normal addressing of the dispatch memory is in effect, so it is advisable to have the length field contain 0 so that the dispatch memory location to modify is uniquely specified by the dispatch address in the instruction.

The JUMP Instruction

The JUMP instruction allows conditional branching based on any bit of any M source or on a variety of internal processor conditions, including ALU output. (While DISPATCH could also be used to test single M source bits, the use of JUMP saves dispatch memory.) The JUMP operation is also used, by means of a trick, to write into the instruction memory.

```
IR<44-43> = 1 (JUMP operation)
IR<41-32> = A source
IR<31-26> = M source
IR<25-12> = New PC
IR<9>     = R bit (1 means pop new PC off SPC stack)
IR<8>     = P bit (1 means push return PC onto SPC stack)
IR<7>     = N bit (1 means inhibit next instruction if jump successful)
IR<6>     = If 1, invert sense of jump condition
IR<5>     = If 0, test bit of M source; if 1, test internal condition
IR<4-0>   = If IR<5>=0, rotation count for M source.
            If IR<5>=1, condition number:
                0  Low bit of shifter output (illegal)
                1  M source < A source
                2  M source ≤ A source
                3  M source = A source
                4  Page fault
                5  Page fault or interrupt pending
                6  Page fault or interrupt pending or sequence break flag
                7  Unconditionally true
```

Page faults, interrupts, and sequence breaks are documented in later sections.

The jump condition is determined as follows. If IR<5>=0, then the M source is rotated left by the rotation count; the low-order bit of the result is then tested. Thus, to test the sign bit, a rotation count of 1 should be used. The jump condition is true if the low-order bit is 1. If IR<5>=1, then the specified internal condition is tested. In either case, the sense of the jump condition is inverted if IR<6>=1. In particular, this allows testing of all six arithmetic relations between the M and A sources.

If the final jump condition, possibly after inversion, is true, then the new PC field and the R, P, and N bits are used to determine the new contents of the PC. If the condition is not true, execution continues with the next instruction, modulo the POPJ bit.

If both the R and P bits are set (WRITE), then A and M sources are (conditionally!) written into the instruction memory. Bits <47-32> are taken from A source bits <15-0>; bits <31-0> are taken from M source <31-0>. Notice that this is not the same alignment of bits as is used for the "next instruction modify" functional destinations (16 and 17). The reason for the odd location of WRITE in the instruction

set is due to the way in which it operates. It causes the same operations as the CALL transfer type, resulting in the the old PC plus 1 or 2 being saved on the SPC stack and the PC register being loaded with the address to be modified. Then, when the instruction memory would normally be fetching the instruction to be executed from that location, a write pulse is generated, causing the saved data from the A and M sources to be written into the instruction memory. Meanwhile, the machine simulates a RETURN transfer instruction, causing the SPC stack to be popped back into the PC and instruction execution to proceed from where it left off. Note that this instruction requires use of a word on the SPC stack and requires an extra cycle. It is highly recommended that the N bit also be on in the JUMP instruction, since the processor will be executing a RETURN transfer type unconditionally during what should be the execution of the instruction following the write. If, however, this does not conflict with other things that this following instruction specifies, then the following instruction may be executed. Care is required.

Program Modification

A novel technique is used for variabilizing fields in the program instruction. Two of the "functional destinations" of the output bus are (conceptual) registers (sometimes collectively referred to as the OA register), whose contents get OR'ed with the next instruction executed. Combined with the shifter/masker ability to move any contiguous set of bits into an arbitrary field, this feature provides, for example, variable rotation counts and the ability to use program determined addresses of registers; for example, it can be used to index into the A scratchpad memory.

Functional destination 16 (OA-REG-LOW), when written into, effectively OR's bits <25-0> into bits <25-0> of the next instruction; functional destination 17 (OA-REG-HIGH) effectively OR's bits <21-0> into bits <47-26> of the next instruction. The place between bits <26> and <25> is a natural dividing line for all classes of instructions. Note that only one half of a particular instruction can be modified, since it is impossible to write into both functional destinations simultaneously.

When this feature is used, parity checking is disabled for the word fetched from the instruction memory, since the OA "register" is OR'ed into the output of the memory before parity is checked.

This feature is particularly useful for supplying the address of a location of instruction memory or dispatch memory to be written into, for specifying variable addresses in the A and M memories, and for operations on bytes of variable length or position. Examples of these are detailed in a later section.

Clocks

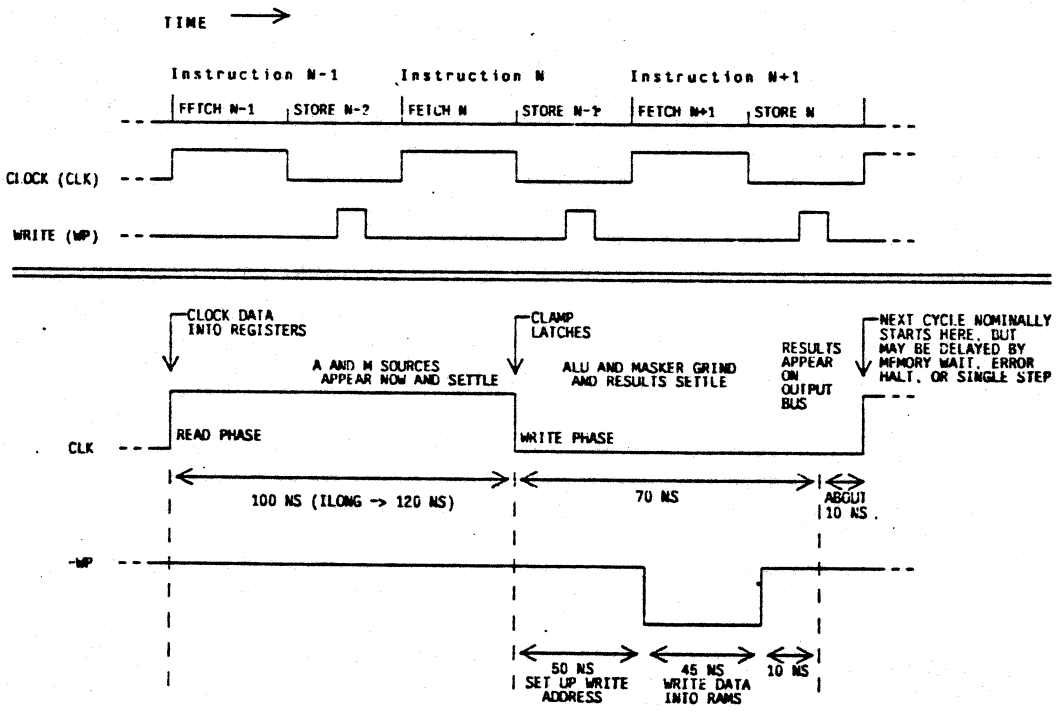
The CADR processor uses only one clock signal. This clock loads output data into the designated registers, and a new PC and instruction are also loaded. The only events which do not take place synchronous with the clock are the control signals for the A, M, and PDL scratchpads and the SPC stack. For these devices, a two stage cycle is performed. During the first phase, the source addresses of the respective devices are gated into the address inputs. After the output data has settled, the outputs of these devices are latched. Then, the address is changed to that specified as the write location from the previous instruction. After the address has settled, a write pulse is generated for the scratchpad memory to perform the write. Pass-around paths are provided (invisibly to the programmer) for the A and M memories, which notice and correct read references to a location which was written into on the previous cycle but has not yet actually been written into the scratchpad. No such pass-around path is provided for the PDL memory, because on any cycle in which the PDL memory is written into, the M scratchpad must also be written into, and so the next instruction can refer to that M scratchpad location, thereby using the M pass-around path. The SPC stack has a pass-around path when used by the RETURN transfer type, but does not have a pass-around path when used as an M source. The RETURN pass-around path makes it possible to have a subroutine only two instructions long. It would take extra hardware to provide the missing pass-around paths, and examination of actual microprograms showed that they would be very rarely used.

The clock cycle is of variable length. The duration of the first half of the cycle (the "read phase") is controlled by both the ILONG bit of the instruction (IR<45>) and by two "speed" bits from the diagnostic interface. The duration of the second half (the "write phase") is normally fixed. This clock serves as both the processor clock and a clock for the bus interface, memory, and external devices.

The clock can be stopped at the end of either phase, for several reasons. Usually the clock stops at the end of the read phase, referred to as "wait". This leaves the clock in the inactive high state, and leaves the latches on the memories open. The clock can wait because the machine was commanded to halt by the diagnostic interface, because a single-step commanded by the diagnostic interface has completed, because of an error such as a parity error, because of the statistics counter overflowing, or because of a memory-wait condition. This latter condition happens if a main memory cycle is initiated while a previous cycle is still in progress, or if the program calls for the result of a main memory read before the bus controller has granted the bus access needed to perform that read cycle. During a clock wait, the processor clock stops, but the clock to the rest of the system (the bus interface and XBUS devices), continues to run, allowing them to operate. When the processor finishes waiting the processor clock starts up in synchrony with the external clock.

The clock can also stop at the end of the write phase, referred to as "hang". This is used only during memory reads. If the processor calls for the result of a read which is in progress but has not yet completed, it hangs until the data has arrived from memory and sufficient time has passed for the data to flow through the data paths and appear on the output bus. This is also sufficient time for the parity of the data to be checked. In the case of a hang, both clocks stop, which allows them to restart synchronously without any extra delay. In this way, the speed of the processor is adjusted

to exactly match the speed of the memory.



CADR Cycle Timing

Accessing Memory

Access to main memory is accomplished through use of several functional sources and destinations. These perform three functions; first, they allow access to two registers, VMA (virtual memory address) and MD (memory data). Secondly, they can initiate memory operations. Thirdly, they can wait for a memory operation to be completed. Actually, this facility is not just for accessing main memory; it is used to access any device on the Xbus or the Unibus, which includes not only memory but peripheral equipment. For simplicity the term "memory" will be used, however.

There are eight functional destinations associated with the memory system. Four of these load data into the VMA, the other four load data into the MD. Each group of four consists of one with no other side effects, one which starts a read cycle, one which starts a write cycle, and one which writes into the virtual address map.

In a memory read operation, data from memory is placed in the MD register when it arrives, and can then be picked up by the program (using a functional source). In a memory write operation, the program places the data to be written into the MD register (by using a functional destination), whence it is passed to the memory.

The VMA register contains the virtual address of the location to be referenced. This is 24 bits long; the high 8 bits of the register exist but are ignored by the hardware. The VMA contains a "virtual" address; before being sent to the memory it is passed through the "map", which produces a 22 bit physical address, controls whether permission for the read or write operation requested is allowed, and remembers 8 bits which the software (microcode) can use for its own purposes.

Except when starting a memory cycle, the address to be mapped comes from bits <23-0> of the MD register, rather than the VMA register. The reason for this is to simplify the use of the map for checking what "space" a pointer being read from or written into memory points at, a frequently-needed operation in the Lisp machine garbage-collection algorithm.

The map consists of two scratchpad memories. The First Level Map contains 2048 5-bit locations, and is addressed by bits <23-13> of the VMA or MD. The Second Level map contains 1024 24-bit locations, and is addressed by the concatenation of the output from the First Level Map and bits <12-8> of the VMA or MD. The virtual address space consists of 2048 blocks, each containing 32 pages. Each page contains 256 words (of 32 bits, of course). Each block of virtual address space has a corresponding location in the First Level Map. Locations in the Second Level Map are not permanently allocated to particular addresses; instead, the First Level Map location for a block of virtual addresses indicates where in the Second Level Map those addresses are currently described. The Second Level Map contains sufficient space to describe 32 blocks, so at any given time most blocks must be described as "no information available." This done by reserving the last 32 locations in the Second Level Map for this purpose and filling them with "no information available" page descriptors; most First Level Map locations will point here.

The output of the Second Level Map consists of:

```
MAP<23>    = access permission
MAP<22>    = write permission
MAP<21-14> = available to software.  Note that bits 15 and 14 can
             be tested by the DISPATCH instruction.
MAP<13-0>  = physical page number
```

The physical address sent to memory is the concatenation of the physical page number and bits 7-0 of the VMA.

The two maps can be read by putting an appropriate address in the MD, and reading the functional source MEMORY-MAP-DATA (11):

```
MAP<31> = 1 if the most recent memory cycle was not performed because it
          was an attempt to write without write permission, i.e. a 1 in
          bit 22 of the second level map.
MAP<30> = 1 if the most recent memory cycle was not performed because there
          was no access permission, i.e. a 1 in bit 23 of the second level map.
          MAP<30> is 0 if no access fault exists, although a write fault may
          exist.  Note that bits <31-30> apply to the last attempted memory
          cycle, and have nothing to do with the map locations addressed by
          the contents of MD.
MAP<29> = 0 always.
MAP<28-24> = First Level Map
MAP<23-0> = Second Level Map
```

The maps can be written by using one of the functional destinations VMA-WRITE-MAP (23), MEMORY-DATA-WRITE-MAP (33). The MD supplies the address of the map location to be written, and the VMA supplies the data to be written, and tells which level of the map is being written. One register must be set up in a previous instruction, the other is written via the functional destination, and the actual writing into the map happens on the following cycle. There is no pass-around path and no latch for the map, so the following instruction must not use it.

The first level map is written from bits <31-27> of the VMA, if VMA<26> is a 1. (These are not the same bits as it reads into when using the MEMORY-MAP-DATA functional source.) The second level map is written from VMA<23-0>, if VMA<25> is a 1. Note that when writing the second level map the first level map supplies part of the address, and must have been written previously. Therefore it is not useful to write both at the same time, although it is possible to set both bits to 1.

Main memory operations are initiated by using one of the functional destinations VMA-START-READ (21), VMA-START-WRITE (22), and MEMORY-DATA-START-WRITE (32). There is also MEMORY-DATA-START-READ (31), but it is probably useless. In the case of a write, the VMA supplies the address and the MD supplies the data, so one register must be set up in advance and the other is set up by the functional destination that starts the operation. A main memory read can also be started by the

macro instruction-stream hardware, described later.

The register named (VMA or MD) is loaded with the result of the instruction (from the Output Bus) at the end of the cycle during which that instruction is executed. During the following cycle, the map is read. The instruction executed during this cycle should be a JUMP instruction which checks for a page fault condition. At the end of this cycle, if no page fault occurs, the memory operation begins. The processor continues executing while the memory operation happens, but if any operation which conflicts with the memory being busy is attempted, the machine waits or hangs until the memory operation has been completed. Such references include asking for the results of a read cycle by using the MEMORY-DATA (12) functional source, using any functional destination that refers to the VMA, MD, or MAP, or attempting to start a read cycle via the instruction-stream hardware.

The presence or absence of a page fault is remembered until the next time a memory cycle is started, so it is not strictly necessary to check for page fault immediately after starting a cycle, but is good practice.

The MEMORY-DATA-START-WRITE destination is useful for doing the second half of a read-followed-by-write operation, since the correct value is still in the VMA. Note that it is still necessary to check for a page fault after starting the write, since you may have read permission but not write permission.

There is a feature by which main memory parity errors can be trapped to the microcode. A bit in the diagnostic interface controls whether or not this is enabled. When the MEMORY-DATA functional source is used, and the last thing to be loaded into the MD was data from memory which had even parity, a main memory parity error has occurred. If trapping is enabled, the current instruction is NOPed and a CALL transfer to location 0 is forced. The following instruction is also NOPed. The trap routine must use the OPC registers to determine just where to return to if it plans to return, since if a transfer operation was in progress the address pushed on the SPC stack by the trap may have nothing to do with the address of the instruction which caused the trap. This is also true of the error-handler for microcode-detected programming errors. If a main memory parity error occurs, and trapping is not enabled, the machine halts if error-halting is enabled, just as it does in response to a parity error in an internal memory.

When using semiconductor main memory, which has single-bit error correction, a parity error trap indicates that an uncorrectable multiple-bit error occurred. Single-bit errors are corrected automatically by the hardware, and cause an interrupt so that the processor may, at its leisure, log the error and attempt to rewrite the contents of the bad location.

The Instruction-Stream Feature

The CADR processor contains a small amount of hardware to aid in the interpretation of an instruction stream which comes in units smaller than the CADR word size. For example, the Lisp-machine macrocompiled instruction set uses 16-bit units. The hardware speeds up both fetching and decoding of instructions by relieving the microcode of some routine bookkeeping.

Both 8-bit (byte) and 16-bit (halfword) instructions are supported, depending on a mode bit (bit 29 of the "Interrupt Control" register, functional destination 2.) The hardware decides when it is time to fetch a new main-memory word, containing the next 2 or 4 units of the instruction stream, and alters the flow of microprogram control. The hardware provides a feature by which the rotator control can be made to select the current unit of the instruction stream; this is used when dispatching on the instruction being interpreted, and when extracting fields of the instruction via the BYTE microinstruction.

There is a 26-bit register called the Location Counter (LC), which can be read by functional source 13 and written by functional destination 1. It always contains the address of the next instruction stream unit, in terms of 8-bit bytes. In halfword mode LC<0> is forced to zero. The LC is capable of counting by 1 or 2 (depending on byte vs. halfword mode) and has a special connection to the VMA; the VMA is loaded from the LC, divided by 4, when an instruction-fetch occurs.

The high 6 bits of functional source 13 are not part of the LC per se, but contain various associated status, as follows:

31 Need Fetch. This is 1 if the next time the instruction stream is advanced, a new word will be fetched from main memory. This is a function of the low 2 bits of LC, of byte mode, and of whether the LC has been written into since an instruction word was last fetched from main memory.

30 not used, zero.

29 LC Byte Mode. 1 if the instruction stream is in 8-bit units, 0 if it is in 16-bit units. This reflects bit 29 of the Interrupt Control register.

28 Bus Reset. This reflects bit 28 of the Interrupt Control register, which is set to 1 to reset the bus interface, the Unibus, and the Xbus.

27 Interrupt Enable. 1 if external interrupt requests are allowed to contribute to the JUMP condition. This reflects bit 27 of the Interrupt Control register.

26 Sequence Break. 1 if a sequence break (macrocode interrupt signal) is pending. This flag does nothing except contribute to the JUMP condition. This reflects bit 26 of the Interrupt Control register.

Bit 14 of the SPC stack is used to flag the return address containing it as the address of the main instruction-interpretation loop. The hardware recognizes a RETURN transfer with SPC<14>=1 as completing the interpretation of one instruction and initiating the interpretation of the next. The instruction stream will be advanced to its next unit (byte or halfword) in the cycle following the RETURN transfer. (It is delayed one cycle for obscure timing reasons.) This cycle is free to also execute a useful microinstruction, provided it does not use the LC, VMA, MD, and associated hardware.

Advancing the instruction stream increments the LC, by 1 or 2. If a new word needs to be fetched from main memory, the unincremented LC, divided by 4, is transferred to the VMA and a read cycle is started. A fetch can be required either because the LC points at the first unit of a word or because the LC has been modified since the last instruction stream advance (a branch occurred). It is legal for the instruction which does the RETURN transfer to modify the LC, and a fetch will always be required. If no fetch is required, the RETURN transfer is altered by forcing SPC<1> to 1, skipping over two microinstructions which, in the fetch case, check for a page fault (or interrupt or sequence break) and transfer the new instruction stream word from MD into a scratchpad location.

The instruction stream can also be advanced by a DISPATCH instruction with bit 24 set. In this case, no alteration of the SPC return address occurs. The dispatch should check the NEEDFETCH signal, which is available as bit 31 of the LC functional source, to determine whether a new word is going to be fetched. If a fetch occurs, the DISPATCH should call a subroutine to check for page fault and transfer the new instruction stream word from MD to a scratchpad location. If no fetch occurs, the DISPATCH should drop through. The instruction after the DISPATCH may then operate on the next unit of the instruction stream. This feature is provided to facilitate the use of multi-unit instructions.

The remaining hardware associated with the instruction stream feature implements miscellaneous function 3, which alters the M-rotate field to select the current unit of the instruction stream from the current word, which should be supplied as the M-source. This applies to any operation which uses the rotator: BYTE instructions, DISPATCH instructions, and JUMP instructions which test a bit. The instruction should be coded for the unit (byte or halfword) at the right-hand end of the word. In half-word mode, IR<4> is XOR'ed with LC<1> to produce the high-order bit of the rotate count. In byte mode, IR<4> is XOR'ed with (LC<1> XOR LC<0>), and IR<3> is XOR'ed with LC<0>. The effect, since the LC always has the address of the next instruction, and the bits are numbered from right to left, is as desired. In halfword mode, the low half of the M source is accessed for the even instruction, when LC<1>=1, and the high half is accessed for the odd instruction, when LC<1>=0.

Multiplication, Division, and the Q register

The Q register is provided in CADR primarily for multiplication and division. It is occasionally useful for other things because it is an extra place to put the results of an ALU instruction, and because it can be used to collect the bits which are shifted out when the OUTPUT-SELECTOR-RIGHTSHIFT-1 operation is used in an ALU instruction.

The Q register is controlled by two bits (IR<1-0>) in the ALU instruction. The operations are do nothing, shift it left, shift it right, and load it from the output of the ALU. (It loads from the ALU rather than the Output Bus for electrical reasons.) When the Q register shifts left, Q<0> receives -ALU<31>, the complement of the sign of the ALU output. When the Q register shifts right, Q<31> receives ALU<0>, the low bit of the ALU output. The Q register is also connected to the Output Bus shifter; when the Output Bus is shifted left, OB<0> receives Q<31>, the sign of the Q. These interconnections are dictated by the needs of multiplication and division.

Multiplication in CADR is a simple, 1 bit at a time, shift-and-add affair. The hardware provides a conditional-ALU operation, MULTIPLY-STEP, which is ADD if Q<0>=1, and SETM otherwise. This is used in combination with SHIFT-Q-RIGHT and OUTPUT-SELECTOR-RIGHTSHIFT-1. Initially the multiplicand is placed in an A-scratchpad location and the multiplier is placed in Q. 32 MULTIPLY-STEP operations are executed; as Q shifts to the right each of the bits of the multiplier appear in Q<0>. If the bit is 1, the multiplicand gets added in. The results of each operation go into an M-scratchpad location, which is fed back into the next step. The low bit of each result is shifted into Q. Thus, when the 32 steps have been completed, the Q contains the low 32 bits of the product, and the M-scratchpad location contains the high 32 bits.

This algorithm needs a slight modification to deal with 2's complement numbers. The sign bit of a 2's complement number has negative weight, so in the last step if Q<0>=1, i.e. the multiplier is negative, a subtraction should be done instead of an addition. The hardware does not provide this, so instead we do a subtraction after the last step, which is adding and then subtracting twice as much, which has the effect of subtracting. Note that this correction only affects the high 32 bits of the product, and can be omitted if we are only looking for a single-precision result. Consider the following code. (The CONSLP assembler format used is explained later in this document.)

```
; Multiply Subroutine.  A-MPYR times Q-R, low product to Q-R, high to M-AC.

MPY       ((M-AC) MULTIPLY-STEP M-ZERO A-MPYR)          ;Partial result = 0 in first step
(REPEAT 30. ((M-AC) MULTIPLY-STEP M-AC A-MPYR))         ;Do 30 steps
          (POPJ-IF-BIT-CLEAR-XCT-NEXT                   ;Return after next if A-MPYR positive
                    (BYTE-FIELD 1 0) Q-R)
         ((M-AC) MULTIPLY-STEP M-AC A-MPYR)             ;The final step
          (POPJ-AFTER-NEXT
           (M-AC) SUB M-AC A-MPYR)                      ;Correction for negative multiplier
```

```
(NO-OP)                                    ;Jump delay
```

To multiply numbers of less than 32 bits is also possible. With the same initial conditions, after n steps the high n bits of the Q contain the low n bits of the product, and the remaining bits of the product are in the low bits of the M-scratchpad location. Two BYTE instructions can be used to extract and combine these bits to produce a right-adjusted product, if the numbers are unsigned.

Division is a little more complex than multiplication. It too goes a bit at a time, using a non-restoring algorithm which either adds or subtracts at each stage. The basic idea is to keep subtracting the divisor from the dividend, shifted over by different amounts, as in long-division by hand. If the subtraction produces a positive result, it "goes in" and a quotient bit of 1 is produced. If the subtraction produces a negative result, it "fails to go in" and a quotient bit of 0 is produced. Instead of backing up and not doing the subtraction, we set a flag that too much has been subtracted, and add instead the next time. This works since the weight of the divisor on the next step is half as much, and B-(A/2) = B-A+(A/2). The "flag" is simply the complement of the quotient bit produced, except for the first step when the flag must be forced to OFF.

Division does not handle 2's complement numbers as easily as multiplication does. The algorithm essentially requires all positive numbers, however the hardware automatically takes the absolute value of the divisor by interchanging addition and subtraction if the divisor is negative. It is up to the microcode to make the dividend positive beforehand, and to determine the correct signs for the quotient and remainder afterward. The sign of the quotient should be the XOR of the signs of dividend and divisor. The sign of the remainder should be the same as the sign of the dividend.

Initially the positive dividend is in the Q register and the signed divisor is in an A-scratchpad location. Appropriate conditional-ALU operations are used in conjunction with the SHIFT-Q-LEFT and OUTPUT-SELECTOR-LEFTSHIFT-1 functions. An M-scratchpad location receives the result of each step, and is fed back to the next step. This location initially contains the high 32 bits of the double-length dividend, or 0 if the dividend is single-precision. At each step, the OUTPUT-SELECTOR-LEFTSHIFT-1 operation brings the high bit of the Q into the low bit of the M-scratchpad, bringing up another bit of the dividend. At each step, the complement of the sign of the ALU output represents a bit of the quotient and is shifted into the low end of Q. After 33 steps, Q contains the positive quotient (which is why it is called the Q-for-quotient register). The reason why it takes 33 steps rather than 32 is a little difficult to explain. The quotient bit produced by the first step, if 1, indicates "divide overflow", and is not really part of the quotient. When using a single-precision dividend, "divide overflow" can only happen if the divisor is zero, since the initial operation is zero minus the absolute value of the divisor, which is negative unless the divisor is zero.

What is left of the dividend after all the subtractions is the positive remainder. The last step does not use OUTPUT-SELECTOR-LEFTSHIFT-1, so that the M-scratchpad will receive the remainder rather than the remainder times 2. If the "too much has been subtracted" flag is set, it is necessary to do one final addition to correct

the remainder. This addition simply undoes the previous subtraction, not also doing a new subtraction, because of the omission of the left shift.

The ALU operations for division are:

DIVIDE-STEP The conditional add or subtract described above, SHIFT-Q-LEFT, and OUTPUT-SELECTOR-LEFTSHIFT-1. Q<0>=0 serves as the "too much has been subtracted" flag.

DIVIDE-FIRST-STEP Identical to DIVIDE-STEP except that the "too much has been subtracted" flag is forced to be off.

DIVIDE-LAST-STEP Identical to DIVIDE-STEP except that the OUTPUT-SELECTOR-LEFTSHIFT-1 is omitted.

DIVIDE-REMAINDER-CORRECTION-STEP The conditional add or subtract logic is used, except subtract is turned into SETM by invoking part of the multiply logic. The exchanging of add and subtract if the divisor is negative then applies, doing the right thing. No shifting occurs and Q is unchanged.

Division of numbers smaller than 32 bits can be accomplished in less than 33 steps by sufficiently careful shifting of the inputs and outputs.

To illustrate how it all fits together, and show how to do the sign-correction, here is the code for 32-bit division, with a double-precision dividend, in the CONSLP format explained later in this document:

```
; Division Subroutine.
; M-AC and M-1 are the high and low words of the dividend, respectively.
; M-2 is the divisor.  The quotient is in M-AC, the remainder in M-1.

DIV     (JUMP-GREATER-OR-EQUAL M-AC A-ZERO DIV1)                ;Check for negative dividend
        (JUMP-NOT-EQUAL-XCT-NEXT M-1 A-ZERO DIV0)               ;If so, change sign
       ((M-1 Q-R) SUB M-ZERO A-1)
        ((M-AC) SUB M-AC (A-CONSTANT 1))                ;Borrow from high if low is zero
DIV0    ((M-AC) SETCM M-AC)                             ;1's complement high dividend
        (CALL DIV2)                                     ;Now, call positive-dividend case
        (POPJ-AFTER-NEXT (M-1) SUB M-ZERO A-1)          ;Make the remainder negative,
       ((M-AC) SUB M-ZERO A-AC)                         ;and change the sign of the quotient

; Divide routine for positive dividend.
DIV1    ((Q-R) M-1)                                     ;Low dividend to Q-R
DIV2    ((M-1) DIVIDE-FIRST-STEP M-AC A-2)              ;First division step
        (JUMP-IF-BIT-SET (BYTE-FIELD 1 0) Q-R DIVIDE-OVERFLOW) ;Error check
(REPEAT 31. ((M-1) DIVIDE-STEP M-1 A-2)                 ;Middle division steps
        ((M-1) DIVIDE-LAST-STEP M-1 A-2)                ;Final step, quotient in Q-R
        ((M-1) DIVIDE-REMAINDER-CORRECTION-STEP M-1 A-2) ;M-1 gets remainder
        ((M-AC) Q-R)                                    ;Extract quotient from Q-R
        (POPJ-AFTER-NEXT                                ;Return after next, but if
         POPJ-GREATER-OR-EQUAL M-2 A-ZERO)              ; divisor is negative,
```

```
((M-AC) SUB M-ZERO A-AC)                ; change sign of quotient
```

The Bus Interface

The Bus Interface connects the CADR machine to two busses, the Unibus and the Xbus. The Unibus is a regular pdp11 bus, used to attach peripheral devices, especially commercial devices designed for the PDP11 line. The Xbus is a 32-bit bus used to attach memory and high-performance peripheral devices, such as disk. The bus interface also includes the diagnostic interface, which allows a unibus operator, such as a pdp10, a pdp11, or another lisp machine, to control the operation of the machine, hardware to pass interrupts from the Unibus and the Xbus to the processor, the logic which arbitrates the Xbus, and the logic which arbitrates the Unibus in the absence of a pdp11 on that bus.

The Bus Interface allows the CADR machine to access memory on the Xbus and devices on the Unibus, allows independent devices on the Xbus to access the Xbus (only), and allows Unibus devices to access Xbus memory (through a map since the Unibus address space is not big enough.) Buffering is provided when the Unibus accesses the Xbus, to convert a 32-bit word into a pair of 16-bit words.

The CADR machine sees a 22-bit physical address space of 32-bit words. The top 128K of this, locations 17400000-17777777, reference the Unibus. Each 32-bit word has a 16-bit Unibus word in bits 0-15, and zero in bits 16-31. There is no provision for using byte addressing on the Unibus, nor for read-pause-write cycles. The 128K immediately below the Unibus, locations 17000000-17377777, are reserved for Xbus I/O devices. Locations 0-16777777 are for Xbus memory.

The bus interface includes a number of Unibus registers which control its various functions:

Spy Feature

Unibus locations 766000-766036 are used for the Spy feature, which is described in detail elsewhere. These locations read and write various internal signals in the CADR machine, and provide the necessary hook for microcode loading and diagnostics.

Two-Machine Lashup

Two bus interfaces may be cabled together with a single 50-wire flat cable for maintenance purposes. One machine, the debugger, is able to perform reads and writes on the other machine's, the debuggee's, Unibus. Through registers on the Unibus (such as the Spy feature), the debuggee may be diagnosed and exercised. By using the debuggee's Unibus map (described below), the debuggee's Xbus can be exercised. The following locations on the debugger's Unibus control this feature:

766100 Reads or writes the debuggee-Unibus location addressed by the registers below.

766114 (Write only) Contains bits 1-16 of the debuggee-Unibus address to be accessed. Bit 0 of the address is always zero.

766110 (Write only) Contains additional modifier bits, as follows. These bits are reset to zero when the debuggee's Unibus is reset.

1 Bit 17 of the debuggee-Unibus address.
2 Resets the debuggee's Unibus and bus interface. Write a 1 here then write a 0.
4 Timeout inhibit. This turns off the NXM timeout for all Xbus and Unibus cycles done by the debuggee's bus interface (not just those commanded by the debugger).

766104 (Read only) These contain the status for bus cycles executed on the debuggee's busses. These bits are cleared by writing into location 766044 (Error Status) on the debuggee's Unibus. They are not cleared by power up. The bits are documented below under "Error Status".

Error Status

766044 Reading this location returns accumulated error status bits from previous bus cycles. Writing this location ignores the data written and clears the status bits. Note that these bits are not cleared by power up.

1 Xbus NXM Error. Set when an Xbus cycle times out for lack of response.
2 Xbus Parity Error. Set when an Xbus read receives a word with bad parity, and the Xbus ignore-parity line was not asserted. Parity Error is also set by Xbus NXM Error.
4 CADR Address Parity Error. Set when an address received from the processor has bad parity. Indicates trouble in the communication between the processor and the bus interface.
10 Unibus NXM Error. Set when a Unibus cycle times out for lack of response.
20 CADR Parity Error. Set when data received from the processor has bad parity. Indicates trouble in the communication between the processor and the bus interface.
40 Unibus Map Error. Set when an attempt to perform an Xbus cycle through the Unibus map is refused because the map specifies invalid or write-protected.

The remaining bits are random (not necessarily zero).

Interrupts

The bus interface allows the CADR machine to field interrupts on the Unibus, if no pdp11 is present. If a pdp11 is present, its program can forward interrupts to the CADR machine in a transparent way. The Xbus also can interrupt the CADR machine. The following Unibus locations control interrupts and the Unibus arbitrator:

766040 Reading this location returns interrupt status, as follows:

1 Disable Interrupt Grant. If this is set, the Unibus arbitrator will not grant BR4, BR5, BR6, and BR7 requests. It will continue to grant NPR

requests. Powers up to zero.

2 Local Enable (read only). 1 means that the bus interface is arbitrating the Unibus. 0 means that a pdp11 is present on the bus and is doing the arbitration.

1774 Bits 9-2 contain the vector address of the last Unibus interrupt accepted by the bus interface or simulated by the pdp11 program.

2000 Enable Unibus Interrupts. A 1 here causes bit 15 (Unibus interrupt) to be set when the bus interface accepts a Unibus interrupt. This bit is not reset by power-up.

4000 Interrupt Stops Grants. A 1 here causes bit 0 (Disable Interrupt Grant) to be set when the bus interface accepts a Unibus interrupt, thus preventing further interrupts until the CADR machine has processed the first interrupt. This bit is not reset by power-up.

30000 Bits 13-12 are the "interrupt level" for purposes of Unibus granting. The mapping to normal pdp11 levels is: 0->0, 1->4, 2->5, 3->6. To simulate level 7, turn on Disable Interrupt Grant. These bits are not reset by power-up.

40000 Xbus Interrupt (read only). This bit is the interrupt-request line on the Xbus.

100000 Unibus Interrupt. A 1 indicates that a Unibus interrupt has been accepted by the bus interface or simulated by a pdp11 program, and is awaiting processing by the CADR program. This bit clears on power-up. Note that the interrupt-request signal to the CADR machine is the OR of bits 14 and 15.

766040 Writing this location writes into bits 0 and 10-13 (mask 36001) of the above register. This is used to change the "interrupt level" and to re-enable acceptance of Unibus interrupts after processing an interrupt.

766042 Writing this location writes into bits 2-9 and 15 (mask 101774) of the above register. This is used to simulate Unibus interrupts and to clear bit 15 (Unibus Interrupt) after processing an interrupt.

Locations between 766040 and 766136 not mentioned above are duplicates of other locations, and should not be used.

Unibus Map

Unibus locations 140000-177777 are divided into 16 pages which can be mapped anywhere in Xbus physical address space. Each page is 512 16-bit words or 256 32-bit words long, the same size as the pages of the CADR virtual memory. The first 8 pages can be addressed by a pdp11, while the second 8 are hidden under the pdp11 I/O space. The Unibus map is intended to be used both as a diagnostic path to the Xbus and for operating Unibus peripherals that access memory.

Each Xbus location occupies 4 Unibus byte addresses. It takes two 16-bit Unibus cycles to read or write one 32-bit Xbus location. 16 buffers (one for each page) are provided to hold the data between the two Unibus cycles. As long as each page is only in

use by a single bus-master, the right thing will happen.

An additional feature is that writing an Xbus address of 17400000 or higher through the Unibus map writes into CADR's MD register. This provides a 32-bit parallel data path into the processor for diagnostic purposes. These Xbus addresses are otherwise unusable, because they are used by the processor to address the Unibus.

Unibus locations 766140-766176 contain the 16 mapping registers. Note that these power up to random contents, and should be cleared by an initialization routine. The bit layout is:

100000 Bit 15 is the map-valid bit. If this is 0, this mapping register is not set up, and will not respond to the Unibus; NXM timeout will occur and an Error Status bit will be set.

40000 Bit 14 is the write-permit bit. If this is 0, this mapping register will not respond to Unibus writes; NXM timeout will occur and an Error Status bit will be set.

37777 Bits 13-0 contain the Xbus page number. These bits are concatenated with bits 9-2 of the Unibus address to produce the mapped Xbus address.

The Xbus

The Xbus is the standard 32 bit wide data bus for the CADR processor. Main memory and high speed peripherals such as the disk control and TV display are interfaced to the Xbus. Control of the Xbus is similar to the Unibus, in that transfers are positively timed and (as far as the devices are concerned) asynchronous. The bus is terminated at both ends with resistive pullups of 390 ohms to ground and 180 ohms to +5 volts, for an effective 123 ohm termination to +3.42 volts. At ground, each termination draws 28 ma. for a total load of 56 ma. The bus is open collector, and may be driven with any device capable of handling the 56 ma. load. The recommended driver is the AMD 26S10, which also provides bus receivers.

A typical read cycle begins with placing the address for the transfer on the -XADDR lines and the parity of the address on the -XBUS.ADDRPAR line. The -XBUS.RQ line is then lowered, initiating the request. The responding device places the requested data on the 32 -XBUS lines and the parity of the data on the -XBUS.PAR line. Should it not be convenient for the device to produce parity (as in the case of I/O registers), the device may assert -XBUS.IGNPAR to notify the bus master that the transfer should not be checked for correct parity. The responding device then asserts -XBUS.ACK, which remains asserted until the -XBUS.RQ signal is removed by the master.

Write requests proceed identically, except that the master asserts -XBUS.WR and the data to be written on the -XBUS lines along with the address lines. All bus masters are required to produce good parity data on writes.

Deskewing delays are the responsibility of the bus master. In particular, it is the responsibility of the bus master to assert good address, write, and data lines 80 ns. prior to asserting -XBUS.RQ, and these lines must be held until the -XBUS.ACK signal drops in response to the master dropping -XBUS.RQ. Responding devices are allowed to assert -XBUS.ACK at the same time they drive read data onto the -XBUS lines. Thus, masters should delay 50 ns. after receiving -XBUS.ACK before dropping -XBUS.RQ and strobing the data. Responding devices are required to drop -XBUS.ACK immediately after -XBUS.RQ is no longer asserted.

Normal bus master arbitration between the CADR processor and the Unibus requests is handled by the bus interface. Devices on the Xbus which must become bus master, such as the disk control, do so by asserting the -XBUS.EXTRQ signal. When the bus becomes free, the bus interface responds by asserting -XBUS.EXTGRANT. This signal is daisy chained between bus master devices on the Xbus, coming in on the -XBUS.EXTGRANT.IN pin and leaving on the -XBUS.EXTGRANT.OUT pin. Within each device, the decision is made whether or not to pass the grant onto the next device. Unlike the Unibus structure, the decision on whether to pass grant and the act of becoming bus master happen synchronously with a master clock signal distributed on the -XBUS.SYNC line.

When a device initiates a request, it immediately asserts -XBUS.EXTRQ. At the falling edge of -XBUS.SYNC it clocks the request signal into a D flip flop which we will call REQ.SYNC. When -XBUS.EXTGRANT.IN goes low, the device asserts -XBUS.EXTGRANT.OUT unless it has either the REQ.SYNC flip flop set, or is already the bus master. At the next falling edge of -XBUS.SYNC the device which has both -XBUS.EXTGRANT.IN and REQ.SYNC set becomes bus master. The device should

immediately assert -XBUS.BUSY and may immediately begin asserting address lines for a transfer. -XBUS.BUSY may be dropped asynchronously, after the slave device drops -XBUS.ACK in response to the master's request.

The -XBUS.EXTGRANT.IN signal must be terminated with a resistive pullup of 180 ohms to +5 volts within each device which does not simply connect it to -XBUS.EXTGRANT.OUT.

XBUS Signal review:

Data lines:

-XBUS<31:0> 32 data lines, low when data is a one.
-XBUS.PAR Parity of the 32 data lines. Required for writes.
-XBUS.IGNPAR Ignore parity signal, may be asserted by any device for a read.

Address lines:

-XADDR<21:0> 22 address lines, low for address bit a one.
-XADDR.PAR Odd parity for the address.

Cycle control lines:

-XBUS.RQ Asserted by the master to request a read or write Minimum of 80 ns following stable -XADDR, -XBUS.WRITE, and -XBUS data.
-XBUS.ACK Asserted by the slave in response to -XBUS.RQ No delay necessary following assertion of good read data.
-XBUS.WR Asserted by the master during a write cycle.

Mastership control lines:

-XBUS.BUSY Asserted when a device other than the bus interface is bus master. Only the bus interface examines this line. Asserted on a -XBUS.SYNC clock edge, dropped asynchronously after -XBUS.ACK drops.
-XBUS.EXTRQ Asserted when a device other than the bus interface wishes to become bus master. Asserted asynchronously, may be removed asynchronously after the device becomes master, but before dropping -XBUS.BUSY.
-XBUS.EXTGRANT.IN The daisy-chained mastership grant signal. Must be pulled up to +5V with a 180 ohm resistor. -XBUS.EXTGRANT.OUT Asserted initially by the bus interface, synchronously with the -XBUS.SYNC edge. The signal may be subject to synchronizer lossage, since it is a clocked version of -XBUS.EXTRQ which is not synchronous

with -XBUS.SYNC

Miscellaneous:

-XBUS.INIT — When low, resets all devices. This is low during power on and off, and when the machine is reset.

-XBUS.SYNC — Synchronization clock for mastership passing and other desired purposes. Devices become bus master synchronous with the edge of this signal. The request will normally follow the edge by 80 ns.

-XBUS.INTR — Driving this low requests an interrupt. All devices are required to initialize to a non-interrupt enable condition, and are required to have interrupt enable and disable bits which can selectively enable interrupts from that device. The "requesting interrupt" state must be readable in one of the device control register bits.

XBUS.POWER.OK — This line is HIGH when power is stable. It remains low for 3 seconds after power comes on, and goes low 3 seconds before power is turned off.

Error Checking

All internal memories in the CADR machine have parity checking. If bad parity is detected, the machine is halted, if that is enabled. The processor always completes the current instruction, and clocks the next one into the IR, before halting. This is done to simplify the timing and to ensure that it halts with the scratchpad memory latches open. It means that the data with bad parity will no longer be on the busses once the machine stops. Furthermore, one incorrect instruction will have been executed. The OPC registers can be helpful in reconstructing what must have happened.

Upon initial power-on, error halting is disabled, but it is expected that as soon as the bootstrap program has initialized all internal memories it will enable error halting.

Main memory parity is checked and can either halt the machine, cause a microcode trap, or be ignored, depending on mode flags in the diagnostic interface.

The data paths do not have any redundant checking. When the machine is bootstrapped it runs some simple diagnostics designed to detect solid failures in the memories and data paths.

Self Bootstrapping

When the machine is powered on it resets itself and the Unibus but does not automatically start up. A bootstrap sequence can be initiated in any of several ways. The diagnostic interface can command one. The diagnostic display panel, by grounding one wire, can start one. This is intended to be connected to a push button. The bus interface can start a bootstrap by grounding one wire. The chaos network interface, if it receives a certain sequence of messages from the network, will "push the boot button." The I/O board recognizes a special set of keyboard commands (left and right control-meta) as a boot signal. The character typed along with the left-right control-meta is available to the bootstrap for selection of software options.

The bootstrap sequence starts by resetting the machine, which will halt it if it is running. It turns on RUN, which will not do anything yet since the clock is stopped. It sets the machine to its slowest speed, disables parity traps, error halts, and the statistics counter, and enables the PROM (read-only) control memory. The trailing edge of the boot signal allows the clock to start, causing a trap to microcode location 0, just like the memory parity error trap. Location 0 of the PROM receives control. It must clear all internal memories (filling them with good parity), reset the Unibus (before first using it), enable error halts, set the machine speed to its normal value, run some diagnostic checks to be sure the machine is working to some extent, load the microcode from the disk, load the initial contents of main memory from the disk, and transfer control to the normal microcode at its start address by going over the Unibus and manipulating the diagnostic interface.

If the diagnostic self-test fails, the microcode goes into a loop, and the value of the PC can be read from the diagnostic display to determine what the problem seemed to be.

Interrupts and Sequence Breaks

Interrupts are hardware signals to the microcode - typically the microcode transfers data in or out of a buffer in main memory. When the signal requires the attention of full Lisp code, a sequence break is triggered. This consists of setting a sequence-break pending flag in A-memory, and, if a defer-sequence-break flag (also in A-memory) is not set, setting the hardware sequence-break flag. This flag is tested at various convenient points such as macroinstruction fetch, and causes the microcode to turn off the flag and enter the sequence-break routines. The sequence-break flag is tested by the same jump instruction that tests for page faults and interrupts.

Interrupts can be generated by both the Xbus and the Unibus. The exact protocol is documented in the section on the bus interface.

Sequence-breaks are software signals indicating the need to run the scheduler (a Lisp program). A sequence-break suggests that the condition for which some process is waiting may have become true. The scheduler checks all processes for runnability, and also checks if it is time to perform periodic actions which are not full processes. Lisp programs can defer sequence-breaks to protect critical areas, while still allowing interrupts so that real-time response at the lowest level is preserved.

Access to virtual memory in the Lisp Machine software environment is viewed as a primitive operation. Regardless of the actual location of a memory datum, the fetch of that item is continued. This view considerably simplifies coding of the system, but imposes moderately high potential latencies in responding to sequence breaks. Interrupts are handled entirely at the microcode level, and the response time for these will be quite short.

The interrupt-control register, writable by functional destination 2, and readable in the high bits of LC (functional source 13), contains three bits relevant to interrupts. Bit <27>, INTERRUPT ENABLE, allows the external interrupt signal from the bus interface to be seen by the JUMP instruction. Bit <26>, SEQUENCE-BREAK, is the sequence-break flag which is testable by the JUMP instruction.

Bit <28>, BUS-RESET, generates a RESET signal on the Unibus (BUS INIT L) and on the Xbus (XBUS.INIT L), and resets the bus interface, when it is written 1 and then 0. The machine also resets the busses when it is powered up.

Bit <29> is used by the Instruction-Stream feature.

The Statistics Counter

The statistics counter is a 32-bit counter, which increments whenever an instruction with bit 46 = 1 is executed. When the counter overflows from -1 to 0 the machine stops, after completing execution of the instruction which caused the overflow. (The stopping is under control of an enable bit in the diagnostic interface.) Bit 46 is always 0 in instructions from the PROM.

The statistics counter can be read and written using the diagnostic interface. It provides several facilities.

It can be used for metering, to measure how many instructions are executed, possibly restricted to a certain subset of the microprogram. The microcode debugger and console program has commands to set and clear the statistics bits in areas of control memory.

It can be used for breakpointing, by setting the counter to -1 and turning on the statistics bit in those instructions which have breakpoints set on them.

It can be used to find obscure bugs, by setting the statistics bit in all locations of control memory, and setting the appropriate number in the statistics counter to cause the machine to halt just before the point where the error appears, so that it can be single-stepped through the suspect microcode.

The statistics counter is loaded from the Instruction Write Register, rather than the normal diagnostic bus, because of its 32-bit width. Effectively it loads from the M bus with a 1-cycle delay. It is probably not possible for the machine to use the statistics counter on itself, although clever ways might be found.

The Diagnostic Interface

The diagnostic interface occupies 16 Unibus addresses. It includes a 16-bit diagnostic bus which can be used to read and write various portions of the machine. There are 16 readable locations, and 8 writable locations. A readable location and a writable location at the same address have no relation to each other. The diagnostic bus is used by debugging and maintenance programs, including the "console" program, and in a few cases by the machine itself during bootstrapping.

First we will describe the readable locations. These are sometimes called the "spy feature." Naturally, most of these are somewhat meaningless if read while the machine is running.

766000 IR<15-0>. The low 16 bits of the currently-executing instruction.

766002 IR<31-16>. The middle 16 bits of the currently-executing instruction.

766004 IR<47-32>. The high 16 bits of the currently-executing instruction.

766006 not used

766010 OPC. The OPCs are described below.

766012 PC. The current program counter, which is the address of the next instruction to be executed.

766014 OB<15-0>. The low half of the output bus.

766016 OB<31-16>. The high half of the output bus.

766020 Flag Register 1. This provides various signals associated with starting and stopping the machine. When the machine stops due to a hardware error, this register tells what happened. The bits are:

- <15> = -WAIT. 1 if the machine is running or runnable, 0 if it is waiting for memory. See the discussion of Clocks for the exact meaning of WAIT.
- <14> = -V1PE. Normally 1, 0 if the level-2 map had a parity error at the last clock.
- <13> = -V0PE. Normally 1, 0 if the level-1 map had a parity error at the last clock.
- <12> = HIGHOK. 1 if the high runs in the machine are all valid, 0 if some are not. This is essentially a power-supply check, and a check for broken wires.
- <11> = -STATHALT. Normally 1, 0 if the machine has been stopped by the statistics counter.
- <10> = ERR. 1 if an error condition is present. If ERRSTOP is on in the mode register, the machine is stopped.

<9> = SSDONE. 1 if a single-step operation has been completed.
<8> = SRUN. 1 if the machine is trying to run (but it may be stopped by a parity error, by a wait condition, or by the statistics counter).
<7> = -HIGHERR. 1 if there was HIGHOK at the last clock.
<6> = -MEMPE. Normally 1, 0 if there was a main memory parity error that was not caught by a trap at the last clock.
<5> = -IPE. Normally 1, 0 if there was a control memory parity error at the last clock.
<4> = -DPE. Normally 1, 0 if there was a dispatch memory parity error at the last clock.
<3> = -SPE. Normally 1, 0 if there was an SPC stack parity error at the last clock.
<2> = -PDLPE. Normally 1, 0 if there was a PDL-buffer parity error at the last clock.
<1> = -MPE. Normally 1, 0 if there was an M-scratchpad parity error at the last clock.
<0> = -APE. Normally 1, 0 if there was an A-scratchpad parity error at the last clock.

766022 Flag Register 2. This register contains flags associated with pipelining and some miscellaneous control signals which the debugging program likes to see. The bits are:

<15> = unused
<14> = unused
<13> = WMAPD. The previous cycle said to write the map, and this cycle will.
<12> = DESTSPCD. The previous cycle wrote into the SPC stack by using a functional destination (as opposed to a CALL transfer).
<11> = IWRITED. The previous cycle did an I-MEM WRITE type of JUMP instruction, and this cycle will write control memory, do a RETURN transfer, and NOP the following cycle.
<10> = IMODD. The previous cycle used the "OA register" to modify this cycle's instruction, or this cycle's instruction came from the DEBUG-IR (see below). This flag inhibits parity checking of the IR.
<9> = PDLWRITED. The previous cycle caused a write into the PDL-buffer, and this cycle will do it.
<8> = SPUSHD. The previous cycle caused a write into the SPC stack, and this cycle will do it.
<7> = unused
<6> = unused.
<5> = IR<48>. This is the parity bit of the IR.
<4> = NOP. The instruction currently in the IR is not really being executed; this cycle is a NOP cycle.
<3> = -VMAOK. The last attempt to start a main memory cycle was not successful because the map indicated a page fault.

<2> = JCOND. 1 if the jump-condition is satisfied. Meaningless if the instruction in IR is not a JUMP instruction.
<1-0> = PCS1-0. These 2 bits select the next PC (the address of the instruction after next.) The encoded values are:
0 = SPC<13-0> the SPC stack.
1 = IR<25-12> the address specified by a JUMP instruction.
2 = DPC<13-0> the dispatch memory.
3 = IPC<13-0> the PC+1.

766024 M<15-0>. The low half of the M-source selected by the instruction currently in IR.

766026 M<31-16>. The high half of the M-source.

766030 A<15-0>. The low half of the A-source selected by the instruction currently in IR.

766032 A<31-16>. The high half of the A-source.

766034 ST<15-0>. The low half of the statistics counter.

766036 ST<31-16>. The high half of the statistics counter.

Here is a description of the writable registers of the diagnostic interface.

766000 DEBUG-IR<15-0>. The low 16 bits of an instruction supplied by the diagnostic interface.

766002 DEBUG-IR<31-16>. The middle 16 bits.

766004 DEBUG-IR<47-32>. The high 16 bits.

766006 Clock control register. Resetting the machine sets this to zero. The following bits exist:
<4> = LDSTAT. Setting this to 1, then clocking the machine, causes the statistics counter to load from IWR<31-0>, which loaded from the M bus on the previous clock.
<3> = IDEBUG. Setting this to 1 causes the IR to load from the DEBUG-IR instead of the PROM or the control memory, when the machine is clocked. The primary way that the machine can be manipulated through the diagnostic interface is by executing instructions using this mechanism.
<2> = NOP11. Setting this to 1 forces NOP. This allows you to clock the machine, for instance to transfer DEBUG-IR into IR, without the present contents of the IR causing unwanted side-effects by getting

executed as an instruction. NOP11 does not prevent the PC from getting changed (in fact it will be incremented), and it does not prevent previously-scheduled pipelined writes from happening.

<1> = STEP. Setting this to 1, when SSDONE is 0, causes the processor clock to run for one cycle, and then set SSDONE. Setting STEP to 0 clears SSDONE. (Both of these operations really take several cycles of the clock to complete.) STEP is the way that the diagnostic interface "clocks" the machine. Note that the main clock is running all the time, even when the machine is stopped. STEP generates a single processor clock, in synchronism with the main clock.

<0> = RUN. Setting this to 1 causes the machine to start running. You first use STEP to set up the state of all the registers and memories, the PC, and the IR, then turn on RUN. The first instruction executed is the one you left in the IR.

766010 OPC control register. Resetting the machine sets this to zero. This register contains some bits which need to be used by the console program in order to completely restore the state of the machine from a saved state. The bits are:

<2> = OPCINH. Setting this to 1 inhibits the OPCs from being clocked by the processor clock. This bit must not be changed except when the clock is high (i.e. the machine is stopped). The process of restoring the OPCs consists of setting OPCINH, then getting the 8 values into the PC by executing JUMP instructions, and transferring those values into the OPCs via the OPCCLK bit. Once the OPCs have been restored, OPCINH remains set so that they will be undisturbed while the rest of the machine state is restored. Just before starting the machine, set OPCINH to 0.

<1> = OPCCLK. Setting this to 1 and then to 0 generates a clock to just the OPCs. This is used to read out the 8 OPC registers without disturbing the state of the rest of the machine.

<0> = LPC.HOLD. Setting this to 1 prevents the LPC register from loading from the PC register when the machine is clocked. This is used in restoring the LPC. The LPC is a duplicate copy of the first OPC register, used by the IR<25> feature of the DISPATCH instruction.

766012 Mode register. Resetting the machine sets this to zero. This register enables various features and controls the speed of the clock. The bits are:

<7> = PROG.BOOT. Setting this to 1 starts a bootstrap sequence.

<6> = PROG.RESET. Setting this to 1 resets the machine. Reset stops the machine by clearing RUN, forces the clock to stop until the RESET operation is over, clears the pipeline flags which cause things to happen in the next instruction, and clears the Clock, Mode, and OPC registers of the diagnostic interface.

<5> = PROMDISABLE. A 1 here disables the PROM. A 0 here replaces the first 1K locations of control memory with the PROM.

<4> = TRAPENB. A 1 here enables main memory parity errors to cause

microcode traps to location 0. A 0 here causes main memory parity errors to be treated the same as other parity errors.

<3> = STATHENB. A 1 here enables overflow of the statistics counter to halt the machine.

<2> = ERRSTOP. A 1 here enables hardware errors (HIGHERR and various parity errors) to halt the machine. A 0 causes it to continue blithely on.

<1-0> = SPEED<1-0>. These bits control the speed of the clock. The ILONG bit in the microinstruction also affects the speed, slowing it down by 40 nanoseconds. The speed codes are:

0 = Extra Slow
1 = Slow
2 = Normal
3 = Fast

766014 not used.

766016 not used.

The OPCs are a set of 8 registers which remember the last 8 values of the PC. This provides a useful history for debugging. It is also used by the microcode itself in certain trap-handling routines. You can only read the last of the 8 OPCs, which is what the PC was 8 clocks ago. Special control is provided over the clocking of the OPCs so that they can be read out without di so that they can be saved and restored by the microcode debugger. This is described above under 766010.

The OPCs can be read both by the diagnostic interface and as a functional source, for maximum flexibility.

The bus interface provides a special path by which the MD register may be loaded. This provides a parallel source of diagnostic input data. After loading MD, instructions can be executed via the DEBUG-IR to transfer the data to the desired destination.

There are several maintenance indicators (light-emitting diodes) scattered around the machine. Inside the front door, near the lower-left-hand corner, are 5 octal displays. These show the current value of the PC. The decimal points on these displays show various interesting conditions. From left to right:

1 - PROMENABLE. Indicates that the current instruction is coming from the PROM rather than the writable control memory.

2 - IPE. Indicates that control memory had a parity error at the last clock.

3 - DPE. Indicates that dispatch memory had a parity error at the last clock.

4 - TILT0. Indicates that the map or main memory had a parity error at the last clock.

5 - TILT1. Indicates that the A-scratchpad, the M-scratchpad, the PDL-buffer, or the SPC stack had a parity error at the last clock.

There is also provision for indicators for the various error conditions, "the machine is really running," and the status of the disk interface. The location of this indicator panel, and whether or not all machines will have one, is not yet determined.

The Disk Controller

The Lisp machine disk controller attaches from 1 to 8 disk units of the "Trident" family to the CADR machine's XBUS. The 1-unit version consists of one board, and a second board is added when more than one disk unit is to be used. The two versions are almost program compatible.

Interface Registers

The disk controller is operated by reading and writing four 32-bit registers which are on the XBUS. These are normally at physical addresses 17377774-17377777, which is just below the Unibus. The address can be changed by changing jumpers. Many bits in these registers refer to the "selected unit", which is that disk unit whose number is currently in bits <30:28> of the disk-address register.

When read, the registers are:

0 STATUS

<24:31> The block-counter of the selected unit. This tells you its current rotational position. Reading of this register is not synchronized to its incrementation, so you must read it twice and check that it came out the same both times.

<23> Internal Parity Error. This indicates that parity of the bits seen at the disk and parity of the bits seen at the memory failed to agree; something must have been lost inside the controller someplace. The Read All and Write All commands cause spurious internal parity errors. The Read Compare command causes a spurious internal parity error if it sets Read Compare Difference (bit 22) and the the disk data and the memory data differ in parity. This error does not stop the transfer.

<22> Read Compare Difference. This indicates that data from memory and data from the disk failed to agree. This bit is undefined unless the command is read-compare. This error does not stop the transfer.

<21> CCW Cycle. This bit being on in combination with Memory Parity Error or Nonexistent Memory Error indicates that the error happened while fetching a CCW, rather than while reading or writing data.

<20> Nonexistent Memory Error. Indicates that memory (or other XBUS device) failed to respond within 15 microseconds. This error stops the transfer.

<19> Memory Parity Error. Indicates that even parity was read from memory (or other XBUS device). This error stops the transfer.

<18> Header Compare Error. Indicates that a block-header read from disk failed to have the expected value. This may be because the disk head is not positioned at the proper place, because the disk is not correctly formatted, or because the header wasn't read correctly. This error stops the transfer.

<17> Header ECC Error. Indicates that the error-correcting code of a block header failed to check. Unfortunately most header ECC errors show up as header compare errors instead. Maybe this can be fixed? This error stops the transfer. Header ECC Error also happens if an attempt is made to continue a

read or write operation past the end of the disk.

<16> ECC Hard. Indicates that the error correcting code discovered an error, and was unable to correct it. The data read from disk is wrong, try reading again. This error stops the transfer.

<15> ECC Soft. Indicates that the error correcting code discovered an error, and was able to determine which data bits were in error. The program can correct it, see the ECC Register for how. The error correcting code will correct any single burst of up to 11 erroneous bits. This error stops the transfer.

<14> Read Overrun. Indicates that data arrived from the disk faster than it could be stored into memory. This error stops the transfer.

<13> Write Overrun. Indicates that memory did not supply data fast enough for the disk. This error stops the transfer.

<12> Start Block Error. Indicates that a start-of-block (sector pulse) happened at a time when it should not have. Either the disk is incorrectly formatted or it is generating spurious sector pulses. This error stops the transfer.

<11> Timeout Error. Indicates that a disk operation took longer than 2.5 seconds. This error stops the transfer.

<10> Selected Unit Seek Error. The selected unit is reporting failure of a seek operation. This error stops the transfer. Reset the error by using the Recalibrate command.

<9> Selected Unit not On-line. The heads are not loaded, the disk is not powered on, or there is no disk at the specified unit number. This error stops the transfer.

<8> Selected Unit not On-Cylinder. Generally indicates that a seek is in progress on the selected unit. Not an error. If the disk goes off-cylinder during a write operation, a fault will occur. If it goes off cylinder during a read, presumably a header-compare error or an ECC error will occur.

<7> Selected Unit Read-Only. The status of a switch on the disk. Note that the read-only status can only change to reflect a change in the switch when the drive is not selected. Storing into the Disk Address register momentarily deselects the current unit so that it may update its read-only status from the switch. Writing while the disk is read-only causes a fault.

<6> Selected Unit Fault. Indicates either trouble with the disk or a programming error, see the Trident manual. This error stops the transfer. Reset by using the Fault Clear and/or Recalibrate commands. This error lights the Device Check light on the drive.

<5> No Unit Selected. This error stops the transfer. Happens if no disk is plugged into the selected unit number, or the disk unit is powered off or "degated".

<4> Multiple Units Selected. This error stops the transfer. This indicates that more than one disk drive is selected, or the wrong drive is selected.

<3> Interrupt Request. 1 means the disk controller is asserting -XBUS.INTR.

<2> Selected Unit Attention. Reset using the At Ease command. Attention indicates seek completion, recalibrate completion, initial loading of

the heads, seek incomplete error, or an emergency head retract. "Implicit" seeks do not cause attention.

<1> Any Attention. Some unit has an attention, you have to select them one after another to find out which.

<0> Not Active. 0 means the controller is busy, 1 means it is ready to accept a command.

1 MEMORY ADDRESS

<31:24> not used

<23:22> Disk type. 00 Trident 01 Marksman 10 unused 11 Trident (old control)

<21:0> the address of the last memory reference made by the disk control. This is the address of a CCW if CCW Cycle is on in the status register, otherwise the address of a data word.

2 DISK ADDRESS

<31> not used

<30:28> Unit number. In the 1-unit version, always zero.

<27:16> Cylinder number. A T-80 has 815. cylinders.

<15:8> Head number. A T-80 has 5 heads. As it turns out, only the bottom 6 bits of the head number can work (this is a feature of the Trident.)

<7:0> Block number. A T-80 is usually formatted with 17. blocks per track. "Block" is mostly synonymous with "sector".

When a transfer is terminated by an error, the disk address register contains the address of the block being transferred when the error occurred. When a transfer terminates normally, the disk address register has the address of the last block transferred.

3 ERROR CORRECTION REGISTER

<31:16> Error pattern bits.

<15:0> Error bit position+1.

When a soft ECC error occurs, this register tells where in the last block transferred the error was. The disk address register has the disk address of the block containing the error, and the command list pointer points to the CCW which points to the memory page containing the error. The error pattern should be XOR'ed into the contents of memory at the specified bit address; it may overlap across a word boundary. Note that the bit position is off by 1; the first bit in the block is bit 1.

You should not write any register while a transfer is active, except for using the Reset command to stop a hung transfer, and even then you should expect to lose.

When written, the registers are:

0 COMMAND

Writing the command register does NOT initiate a transfer, unlike most disk

controllers. Use register 3 (START) to initiate a transfer, after setting up the other registers. However, writing the command register does reset the various error flags. Note that the command register cannot be read back.

<31:12> not used

<11> Done Interrupt Enable. Enables not-active (bit 0 of the status register) to cause an interrupt. The interrupt will keep happening until you clear this bit. (This is really an idle interrupt rather than a done interrupt.)

<10> Attention Interrupt Enable. Enables any-attention (bit 1 of the status register) to cause an interrupt. (The interrupt will only happen if the controller is not active. While the controller is active you couldn't do anything about it anyway.) The interrupt will keep happening until you select the drive and give an at-ease command, or clear this bit.

<9> Recalibrate. In combination with command 5, causes the disk to return the heads to cylinder 0.

<8> Fault Clear. In combination with command 5, resets most fault conditions in the disk.

<7> Data Strobe Late. For recovery of marginal data.

<6> Data Strobe Early. For recovery of marginal data.

<5> Servo Offset. For recovery of marginal data, offsets the heads slightly. Bit 4 controls which direction. Note that this is somewhat kludgey, if you try to seek while the heads are offset you get a fault.(use command 6 first to clear the offset.) Transferring more than one block at a time while in servo offset mode, or even retrying a transfer without first doing an offset clear, will probably cause a fault. Of questionable worth anyway. Writing while the heads are offset causes a fault.

<4> Offset forward. 1 means offset forward, 0 means offset backward.

<3> I/O Direction. 1 means from-memory, 0 means to-memory. See below for valid combinations.

<2:0> Command code. The following combinations of bits are valid commands (here expressed in octal). Note that bits 10 and 11 may always be turned on, and bits 4 through 7 may be turned on in any reading command.

0000 Read.

0010 Read-compare. Reads from both disk and memory, and sets bit 22 of the status register if they don't agree.

0011 Write.

0002 Read All. Reads all bits of the disk starting at the specified rotational position. Note that internal parity errors will occur spuriously during this command, and that it will not automatically advance heads and cylinders. See the description of disk formatting below.

0013 Write All. Writes all bits of the disk starting at the specified rotational position. This is intended for formatting the disk, see below. The caveats under READ ALL apply to WRITE ALL also. In addition, it doesn't really write quite all of the last page; somewhere between zero and seventeen words will be lost.

0004 Seek. Initiates a seek to the cylinder specified in the disk address register. An attention will occur when the seek completes. Note that this command is not logically necessary; the controller always initiates a seek if necessary at the start of a data transfer command. The read, read-compare, and write commands also will seek in the middle of a transfer when necessary. The seek command is provided so you can overlap seeks on multiple units.

0005 At ease. Resets attention on the selected unit.

1005 Recalibrate. Seek to cylinder 0, without assuming the current position of the heads is correct. This is used to correct a seek error, and as part of error recovery. Recalibrate resets some error conditions in the drive, and causes an attention when complete.

0405 Fault clear. Resets most error conditions in the drive.

1405 This probably does both a Recalibrate and a Fault Clear.

0006 Offset clear. Take the heads out of the offset state. This does not wait for completion, but the next command will.

xxx7 This is a reserved command, and will currently hang the controller, causing a timeout error (bit 11 in the status register.)

0016 Reset. This stops the current transfer and resets the controller. This command takes effect as soon as it is stored in the command register; no store in START is required. After storing a Reset command you should store 0 in the command register to turn off the reset condition. Use of Reset while a transfer is in progress isn't guaranteed not to do strange things.

All commands except for the xxx5 group and Reset wait for completion of any previous seek operation on the selected unit before starting. Thus even the Seek and Offset Clear commands can take finite time before the controller is ready for the next command.

1 COMMAND LIST POINTER

This is the address of a vector of Channel Command Words (CCWs) which specify what memory pages, and how many, are to be transferred to/from disk. Only bits <15:0> of the CLP can count, so if you try to carry across this boundary your command list will wrap around.

The format of a CCW is:

<31:24> not used

<23:8> Main memory address of a page

<7:1> not used

<0> More flag. If this bit is 0, this is the last CCW in the list. If this bit is 1, there is another CCW in the following location.

2 DISK ADDRESS

See the description of the disk address register under reading. Note that in the 1-unit version, the unit number bits <30:28> are ignored and regarded as always zero.

3 START:
Writing anything at this address initiates the operation specified in the command, disk address, and command list pointer registers.

Disk Structure

Each disk block contains one Lisp machine page worth of data, i.e. 256. words or 1024. bytes. You can transfer up to 65536. consecutive disk blocks to non-consecutive memory locations in a single operation, or you could if the machine supported that much main memory. A T-80 has 815. cylinders, each with 5 heads (tracks), each with 16. or 17. blocks depending on how you feel like formatting it. A T-300 is the same except it has 19. heads.

Formatting

The format is determined by the program that uses the Write All operation to format the disk, within the constraints determined by the hardware. A track contains (approximately) 20160. bytes (on a T-80 or a T-300). Jumpers in the disk are set to give 17. sector pulses per track, or one every 1164. bytes, with a little left over at the end of the track.

Everything goes low-order bit first and low-order byte first. Note that bits in the disk controller are the complement of bits seen by the drive. Thus all bits in the Trident manual should be thought of as complemented.

The format of a block is:

```
(sector pulse here)
PREAMBLE - 53. bytes of ones.
VFO LOCK - 8. bytes of ones.
SYNC - a byte containing octal 177
HEADER - a 32-bit word as follows:
   <31:30> next block address code:
        0 following block on same track
        1 block 0 on next track (next head)
        2 block 0 on head 0 of next cylinder
        3 end of disk
   <29:28> not used, should be zero
   <27:16> cylinder number, used to verify that the
           disk is positioned to the correct cylinder.
   <15:8> head number, used to verify the head selection.
   <7:0> block number, used to verify the rotational position.
HEADER ECC - a 32-bit checkword.
VFO RELOCK - 20. bytes of ones.
SYNC - a byte containing octal 177
```

```
PAD - a byte containing octal 377, which is here to fix
        a bug in the logic for read-compare.  (Ugh)
DATA - 1024. bytes of whatever you want.
DATA ECC - a 32-bit checkword.
POSTAMBLE - 44. bytes of ones.
```

To format the disk, you should do it one track at a time. Lay out in memory the bits to be written on the track. Truncate the length to a multiple of a page, but make sure that the last 17. words don't matter (in general you will be writing 19. pages, or 19456. bytes, leaving about 771. bytes at the end of the track which may not get written, depending on how full the fifo is when the operation terminates. Depending on the block length chosen, you may not get a chance to fully write the last block, but as long as you get into the data area it will be all right. Do a WRITE ALL command of this data, with a disk address whose block-number field (bits <7:0>) is zero. Ignore any internal parity error (bit 23 of the status register.) You can verify it by using the Read All command (but the internal parity and read-compare features will not work), or you can use the ordinary write and read commands. You must compute the ECC check-words manually. The polynomial is x^31+x^29+x^20+x^10+x^8+1 [if I understand this logic correctly.]

Note that, when using Read All, there is some ambiguity as to precisely where the data read starts. It is unlikely to line up the bytes on byte boundaries. The first several microseconds worth of data will be missing or corrupted.

Debugging

Connector J11 is provided for a flat cable to an LED display, with the following useful signals on it. These are ground when inactive, 15 milliamps at +3 volts or so when active.

1 Read Active. The controller is active and bit 0 of the command register is 0.

2 Write Active. The controller is active and bit 0 of the command register is 1.

3 Seek. The selected unit is not on-cylinder.

4 Transfer Lossage. This is the IOR of Timeout, Read Overrun, Write Overrun, Memory Parity Error, and Nonexistent Memory Error.

5 Format Lossage. This is the IOR of Start Block Error, Header Compare Error, Header ECC Error, and Reset.

6 ECC Lossage. This is the IOR of Hard ECC Error and Soft ECC Error.

7 Disk Lossage. This is the IOR of Multiple Units Selected, No Units Selected, Selected Unit Fault, Selected Unit not On-Line, and Selected Unit Seek Error.

8 Spare. This probably does not light up.

<<Here insert a one-page table of instruction formats and so forth>>

the expression program is evaluated to produce the symbol's value (which may be conditional on the context in which it appears). Expression programs are discussed in a later section.

Instructions

In general, CONSLP assembles a list into a data item by evaluating all the elements of the list and adding them up. There is a fairly rich language for specifying complex expression programs and assigning symbolic names to them; for now, however, we will merely use the symbols predefined by CONSLP. CONSLP also allows the fields of an instruction to be written in almost any order, but we will describe only the conventional order for writing them.

The general form of an I-MEM instruction is:

```
(<popj> (<destinations>) <operation> <condition>
   <M-source> <byte-descriptor> <A-source> <target-tag> <other fields>)
```

The <popj> field is POPJ-AFTER-NEXT to specify that the POPJ bit be set.
The <destinations> field may be an A or M memory tag, or the name of a functional destination, or both an M memory tag and a functional destination.
The <operation> specifies the instruction type, and possibly other fields (such as the jump condition) as well.
The <condition> may also be a separate field, though it usually is encoded as part of the operation.
The <byte-descriptor> describes the byte to be used in a BYTE or DISPATCH instruction.
The <M-source> and <A-source> specify the sources; these may be tags in the appropriate memories, or, for the <M-source>, the name of an M multiplexor source.
The <target-tag> is an I-MEM tag for JUMP instructions, or a D-MEM tag for DISPATCH instructions.
The <other fields> can be such things as the Q control and Miscellaneous Functions.

Many of these fields can be omitted, and CONSLP will default them appropriately. If the <operation> is omitted, then ALU is assumed, unless a <byte descriptor> is present either implicitly or explicitly, in which case BYTE is assumed. If only one source is present in an ALU instruction, then an opcode of SETA is supplied for an A source, and SETM for an M source, thus causing a simple movement of data. If the A source is omitted in a BYTE instruction, then location 2 in A memory is assumed (which is supposed to contain zero).

Here are some examples of instructions, with commentary. We assume the convention described above for A and M memory tags.

```
((A-FOO) M-BAR)      ;move from BAR in M-MEM to FOO in A-MEM

(CALL ZAP)           ;do a CALL transfer to instruction ZAP (M bit set)
```

```
((A-FOO) SUB M-BAR A-BAZ)
                    ;subtract A-BAZ from M-BAR, put result in A-FOO

(JUMP-EQUAL-XCT-NEXT M-BAR A-FOO LOSE)
                    ;jump to LOSE if M-BAR equals A-FOO; N bit is clear,
                    ; so instruction after the JUMP is executed
                    ; whether or not the JUMP succeeds

(POPJ-AFTER-NEXT (M-FOO) MEMORY-DATA)
                    ;put data from memory into M-FOO,
                    ; and also POPJ after next instruction

((M-SAVE MEMORY-DATA-START-WRITE)
        ADD MEMORY-DATA A-ZERO ALU-CARRY-IN-ONE)
                    ;add one to the read memory data,
                    ; transfer to write memory data and M-SAVE,
                    ; and begin writing the data into main memory
                    ; at the address already in the VMA
```

Literals

CONSLP provides a facility for specifying literals in the A and M memories. The constructs

```
(A-CONSTANT <expression>)   and   (M-CONSTANT <expression>)
```

may appear as an A source or M source specification, causing CONSLP to allocate a word in the appropriate memory, assemble the literal expression there, and use the address of that location as the source location. If the same constant in the same memory is referenced many times, CONSLP will assemble only one copy of it. Two constants are considered the same if their final binary values are identical, regardless of the source expressions which reduced to those values. The zero constant is treated specially, and made to refer to location 2 of the appropriate memory (hence the user should reserve these locations as constant sources of zeros). Similarly the -1 constant is made to refer to location 3 of the appropriate memory.

Byte Specifications

Rather than requiring the user to calculate the rotation count and length (minus 1) fields for BYTE and DISPATCH instructions, CONSLP provides a uniform method for specifying a byte in terms of its size and position in the word; CONSLP then calculates the fields appropriately.

The simplest way to describe a byte is with the BYTE-FIELD construct:

(BYTE-FIELD <size in bits> <position from right>)

For example, (BYTE-FIELD 5 0) is the low five bits of a word, and (BYTE-FIELD 7 5) is the seven bits above them. The two arguments to BYTE-FIELD must be constant integers.

Another way to describe a byte is:

(LISP-BYTE <ppss>)

where the low two octal digits of <ppss> are the size and the next two are the position. The argument <ppss> is evaluated as a LISP form (see below under "Expression Programs").

When a byte specifier appears in an instruction, the op-code is defaulted to BYTE, and the type of byte instruction defaulted to "load byte". If specified elsewhere in the instruction, the op-code may be DISPATCH instead; the dispatch is based on the specified byte. The op-code may also be JUMP, but only if the byte is one bit wide; this means that the jump will test the specified bit of the M source.

When CONSLP assembles the final instruction, it constructs the rotation count and length minus 1 fields on the basis of the byte specifier and the operation to be performed. For JUMP, DISPATCH, and "load byte" type BYTE instructions, this involves subtracting the byte position from 32 to obtain the correct rotation count. (Recall that CADR rotates words to the left.) If Miscellaneous Function 3 (LOW PC BIT specifies half word) is enabled, then the position (which should be less than 16) is subtracted from 16 instead. For "deposit byte" and "selective deposit" type BYTE instructions, the byte position itself is used as the rotation count. The length minus 1 field for BYTE and JUMP is computed by subtracting 1 from the byte length, unless the byte length is zero, in which case zero is used. (Note that CADR cannot really handle zero-length bytes, but CONSLP allows them to be defined on the theory that the "next instruction modify" feature may be in use. Programs which use this feature must be aware of the hackery which the assembler pulls, and allow for the actual values of the fields at run time.) The DISPATCH instruction has a length field instead of a length minus 1 field, and so no subtraction of 1 is performed for it.

Here are some examples of the use of byte specifiers:

```
((M-X) (BYTE-FIELD 7 4) M-Y)
                    ;extracts a 7-bit byte, 4 bits from
                    ; the right, from M-Y, and puts this
                    ; byte right-justified in M-X.  The
                    ; A source is defaulted to 1, which
                    ; should be a constant zero so that the
                    ; other bits in M-X will be zero.

(JUMP-IF-BIT-SET (BYTE-FIELD 1 3) M-ZAP QUUX)
                    ;jump to QUUX if the "10" bit is set in M-ZAP

(DISPATCH (BYTE-FIELD 3 0) M-ZAP DTABLE)
```

```
;use the low three bits of M-ZAP to index
; into the dispatch table DTABLE
```

It is possible to create a symbolic name for a byte field by using the ASSIGN pseudo-operation:

```
(ASSIGN LOW-HEX-DIGIT (BYTE-FIELD 4 0))
```

Since this is a common operation, another pseudo-op exists for the purpose:

```
(DEF-DATA-FIELD <symbol> <byte size> <byte position>)
```

For example:

```
(DEF-DATA-FIELD LOW-HEX-DIGIT 4 0)
```

It is also possible to associate a name with a byte field in a particular register. One way to do this is to sum the byte specifier and the name of the register:

```
(ASSIGN CONDITION-CODES (PLUS (BYTE-FIELD 4 0.) PDP-11-PS))
(ASSIGN TRACE-TRAP-BIT (PLUS (BYTE-FIELD 1 4) PDP-11-PS))
(ASSIGN PRIORITY (PLUS (BYTE-FIELD 3 5) PDP-11-PS))
```

This case too is common enough to warrant a special pseudo-operation for the purpose:

```
(DEF-BIT-FIELD-IN-REG <symbol> <byte size> <byte position> <register>)
```

For example:

```
(DEF-BIT-FIELD-IN-REG CONDITION-CODES 4 0 PDP-11-PS)
(DEF-BIT-FIELD-IN-REG TRACE-TRAP-BIT 1 4 PDP-11-PS)
(DEF-BIT-FIELD-IN-REG PRIORITY 3 5 PDP-11-PS)
```

Note that the <register> had better be in the M-scratchpad. With this definition, it is only necessary to mention, say, PRIORITY, in an instruction to cause an appropriate byte reference to occur:

```
((A-PRIORITY) PRIORITY)      ;extract the PRIORITY byte from PDP-11-PS
                             ; and place it right-justified in A-PRIORITY
```

By special dispensation, it also works to use such symbols in the destination field. The appropriate DPB is assembled.

Two more pseudo-operations make it easy to define names for many consecutive bits or fields in a register.

(DEF-NEXT-FIELD <symbol> <byte size> <register>) .sp This defines <symbol> to be a byte of the speicified size, in a position to the left of any fields already defined by DEF-NEXT-FIELD. If this is the first DEF-NEXT-FIELD for the specified register, then the field position is zero (at the low end of the word). For example:

```
(DEF-NEXT-FIELD REL-OFFSET 8 IBM-1130-INSTRUCTION)
(DEF-NEXT-FIELD TAG-FIELD 2 IBM-1130-INSTRUCTION)
(DEF-NEXT-FIELD FORMAT-BIT 1 IBM-1130-INSTRUCTION)
(DEF-NEXT-FIELD OP-CODE 5 IBM-1130-INSTRUCTION)
```

would be entirely equivalent to:

```
(DEF-BIT-FIELD-IN-REG REL-OFFSET 8 0 IBM-1130-INSTRUCTION)
(DEF-BIT-FIELD-IN-REG TAG-FIELD 2 8 IBM-1130-INSTRUCTION)
(DEF-BIT-FIELD-IN-REG FORMAT-BIT 1 10. IBM-1130-INSTRUCTION)
(DEF-BIT-FIELD-IN-REG OP-CODE 5 11. IBM-1130-INSTRUCTION)
```

The pseudo-operation:

```
(DEF-NEXT-BIT <symbol> <register>)
```

is entirely equivalent to:

```
(DEF-NEXT-FIELD <symbol> 1 <register>)
```

and so allocates a single bit. It may be intermixed freely with DEF-NEXT-FIELD. For example:

```
(DEF-NEXT-FIELD CONDITION-CODES 4 PDP-11-PS)
(DEF-NEXT-BIT TRACE-TRAP-BIT PDP-11-PS)
(DEF-NEXT-FIELD PRIORITY 3 PDP-11-PS)
```

The construct:

```
(RESET-BIT-POINTER <register>)
```

may be used to reset the pointer into <register> used by DEF-NEXT-FIELD and DEF-NEXT-BIT. This is useful if the data in <register> can have several different formats. For example:

```
(DEF-NEXT-BIT C PDP-11-PS)
(DEF-NEXT-BIT V PDP-11-PS)
(DEF-NEXT-BIT Z PDP-11-PS)
(DEF-NEXT-BIT N PDP-11-PS)
(RESET-BIT-POINTER PDP-11-PS)
(DEF-NEXT-FIELD CONDITION-CODES 4 PDP-11-PS)
```

```
(DEF-NEXT-BIT TRACE-TRAP-BIT PDP-11-PS)
(DEF-NEXT-FIELD PRIORITY 3 PDP-11-PS)

(DEF-NEXT-FIELD DST-REG 3 PDP-11-INSTRUCTION)
(DEF-NEXT-FIELD DST-MODE 3 PDP-11-INSTRUCTION)
(DEF-NEXT-FIELD SRC-REG 3 PDP-11-INSTRUCTION)
(DEF-NEXT-FIELD SRC-REG 3 PDP-11-INSTRUCTION)
(DEF-NEXT-FIELD OP-CODE 4 PDP-11-INSTRUCTION)
(RESET-BIT-POINTER PDP-11-INSTRUCTION)
(DEF-NEXT-FIELD BRANCH-OFFSET 8 PDP-11-INSTRUCTION)
(DEF-NEXT-FIELD BRANCH-CONDITION 3 PDP-11-INSTRUCTION)
(RESET-BIT-POINTER PDP-11-INSTRUCTION)
```

Dispatch Tables

When assembling into the dispatch memory (i.e. (LOCALITY D-MEM)) it is necessary to use two special pseudo-operations, START-DISPATCH and END-DISPATCH, to allocate blocks of dispatch memory. These pseudo-operations specify the length of the block required, and CONSLP undertakes to pack the various odd-sized blocks into the dispatch memory in an appropriate manner.

The typical form for a dispatch block is:

```
(START-DISPATCH <log2 of size> <constant data>)
<dispatch table tag>
        <first word of table>
        ...
        <last word of table>
(END-DISPATCH)
```

The <log2 of size> is the number of bits that will be dispatched on, that is, the logarithm base 2 of the size of the dispatch block. The <constant data> will be added into each of the words of the dispatch table; this is useful for the P, R, and N bits (which in CONSLP are called P-BIT, R-BIT, and INHIBIT-XCT-NEXT-BIT). The END-DISPATCH is logically not necessary, but is used for error checking. Exactly the correct number of words must be assembled between the START-DISPATCH and END-DISPATCH, or CONSLP will give an error message.

As an example of a dispatch table, consider this code:

```
(LOCALITY M-MEM)
PDP-11-INSTRUCTION          (0)      ;HOLDS SIMULATED PDP-11 INSTRUCTION
(DEF-NEXT-FIELD DST-REG 3 PDP-11-INSTRUCTION)
(DEF-NEXT-FIELD DST-MODE 3 PDP-11-INSTRUCTION)
(DEF-NEXT-FIELD SRC-REG 3 PDP-11-INSTRUCTION)
(DEF-NEXT-FIELD SRC-REG 3 PDP-11-INSTRUCTION)
(DEF-NEXT-FIELD OP-CODE 4 PDP-11-INSTRUCTION)
```

```
        ...

(LOCALITY I-MEM)
        (DISPATCH-CALL-XCT-NEXT DST-MODE D-DST-MODE)
        ...

(LOCALITY D-MEM)
(START-DISPATCH 3 P-BIT)
D-DST-MODE
        (DST-REGISTER)                  ;R0
        (DST-REG-INDIRECT)              ;@R0
        (DST-AUTO-INCREMENT)            ;(R0)+
        (DST-AUTO-INC-INDIRECT)         ;@(R0)+
        (DST-AUTO-DECREMENT)            ;-(R0)
        (DST-AUTO-DEC-INDIRECT)         ;@-(R0)
        (DST-INDEXED)                   ;N(R0)
        (DST-INDEXED-INDIRECT)          ;@N(R0)
(END-DISPATCH)
```

Note that the use in I-MEM of the op-code DISPATCH-CALL-XCT-NEXT is purely for cosmetic purposes, to indicate that the P bit but not the N bit is a constant in all of the dispatch table entries; it is otherwise identical to the DISPATCH op-code.

Standard Operation Codes

CONSLP supplies a large number of initial symbols for various operations, particularly for the various conditional jumps. While it is possible to define different ones, use of these standard ones is naturally encouraged. (These symbols are defined in the file LISPM; CONSYM >.)

ALU Operations

The standard ALU operations supplied by CONSLP are:

Boolean

SETCM	set to complement of M
ANDCB	AND together complements of both M and A
ANDCM	AND complement of M with A
SETZ	set to zeros
ORCB	OR together complements of both M and A
SETCA	set to complement of A
XOR	XOR (exclusive OR) M and A
ANDCA	AND M with complement of A
ORCM	OR complement of M with A

```
    EQV             EQV M and A (complement of XOR)
    SETA            set to A
    AND             AND together M and A
    SETO            set to ones
    ORCA            OR M with complement of A
    IOR             OR M and A (inclusive OR)
    SETM            set to M
```

Arithmetic

```
    ADD             M plus A (two's complement addition)
    SUB             M minus A (two's complement subtraction)
    M+M             M plus M (two's complement addition)
    M+M+1           M plus M plus 1
    M+A+1           M plus A plus 1
    M-A-1           M minus A minus 1
    M+1             M plus 1
```

Conditional Arithmetic

```
    MULTIPLY-STEP
    DIVIDE-FIRST-STEP
    DIVIDE-STEP
    DIVIDE-LAST-STEP
    DIVIDE-REMAINDER-CORRECTION-STEP
```

The conditional ALU operations for multiplication and division are explained in detail in a later section.

The output bus selector field defaults to 1 (output bus gets ALU output). The other two choices must be specified explicitly:

```
OUTPUT-SELECTOR-RIGHTSHIFT-1
OUTPUT-SELECTOR-LEFTSHIFT-1
```

The Q control field of an ALU instruction may be specified by using one of these symbols:

```
SHIFT-Q-LEFT    shift Q left (shifts inverse of ALU<31> into Q<0>)
SHIFT-Q-RIGHT   shift Q right (shifts ALU<0> into Q<31>)
LOAD-Q          load Q from output bus
```

If none of these is present, the default is to do nothing to Q. (Instead of writing LOAD-Q, one may write Q-R in the destination portion of the instruction. This does not mean that Q is a functional destination; it merely forces the operation to be ALU, and forces the Q control field to be LOAD-Q.)

The carry field may be specified by ALU-CARRY-IN-ZERO or ALU-CARRY-IN-ONE. Note that the SUB, M+M+1, M+A+1, and M+1 operations have ALU-CARRY-IN-ONE as part of their definitions, so it is not necessary to specify it explicitly.

BYTE operations

If a byte specifier is present in an instruction and the op-code is not explicitly forced to be JUMP or DISPATCH, then the op-code is BYTE by default, performing a "load byte" type of operation.

To get a "deposit byte" type operation, the symbol DPB is used; similarly, to get a "selective deposit", SELECTIVE-DEPOSIT is used. For example:

```
((A-FOO) DPB M-BAR (BYTE-FIELD 3 6) A-FOO)
                    ;a true PDP-10 style DPB; the low octal
                    ; digit of M-BAR replaces the third lowest
                    ; octal digit of A-FOO.

((A-ZAP) DPB M-BAR (BYTE-FIELD 3 6) A-FOO)
                    ;similar, but the result is placed in
                    ; A-ZAP.  A-FOO is not altered.

((A-ZAP) SELECTIVE-DEPOSIT M-FOO (BYTE-FIELD 16. 8) (A-CONSTANT -1))
                    ;A-ZAP gets a copy of M-FOO with the high eight
                    ; bits and the low eight bits replaced with all ones
                    ; (alternatively, it gets a copy of the -1
                    ; with the middle 16. bits replaced with
                    ; the corresponding bits from M-FOO)
```

DISPATCH Operations

Four op-codes are defined in CONSLP for dispatching:

```
DISPATCH
DISPATCH-CALL
DISPATCH-XCT-NEXT
DISPATCH-CALL-XCT-NEXT
```

These are provided purely for cosmetic purposes, since the actual dispatch action is controlled by the dispatch table. CONSLP makes no attempt to check that the "correct" op-code is used with a given dispatch table. By convention, the XCT-NEXT versions are used iff the instruction following the dispatch instruction will be executed (N bit <u>not</u> set), and the CALL versions are used if the P bit is set.

To specify the value of the 10-bit "immediate argument" which is loaded into the DISPATCH CONSTANT register, one may use

```
(I-ARG <expression>)                ;immediate argument
```

in the dispatch instruction.

There is a special pseudo-op to facilitate use of the DISPATCH CONSTANT to pass a small, constant number as an argument to a subroutine. The form

```
((ARG-CALL FOO) (I-ARG BAR))
```

generates a DISPATCH instruction to a one-word table containing a CALL-type transfer to FOO, and puts BAR in the dispatch constant field of the dispatch instruction. FOO may then use the READ-I-ARG functional source to pick up and act on the argument.

Miscellaneous Function 2 (write into the dispatch memory) is specified by the symbol WRITE-DISPATCH-RAM.

JUMP Operations

CONSLP defines a large number of names for the various JUMP operations. These are all built out of a logical progression of pieces:

```
<type>  <condition>  <xct next>
```

The <type> may be either JUMP, CALL, or POPJ, meaning that no bits, the P bit, or the R bit is set. The <condition> may be one of the following:

```
IF-BIT-SET
IF-BIT-CLEAR
EQUAL
NOT-EQUAL
LESS-THAN
GREATER-THAN
GREATER-OR-EQUAL
LESS-OR-EQUAL
IF-PAGE-FAULT
IF-NO-PAGE-FAULT
IF-PAGE-FAULT-OR-INTERRUPT
IF-NO-PAGE-FAULT-OR-INTERRUPT
IF-PAGE-FAULT-OR-INTERRUPT-OR-SEQUENCE-BREAK
IF-NO-PAGE-FAULT-OR-INTERRUPT-OR-SEQUENCE-BREAK
```

If omitted, the <condition> is assumed to be "always". The <xct next>, if present, is XCT-NEXT; its absence denotes the presence of the N bit, which inhibits the instruction after the jump if the jump is successful. The three parts are connected by "-".

Examples of these operations:

```
CALL-LESS-THAN
```

```
JUMP-LESS-THAN-XCT-NEXT
CALL
POPJ-IF-BIT-SET
CALL-IF-PAGE-FAULT-OR-INTERRUPT
CALL-IF-BIT-CLEAR-XCT-NEXT
JUMP-XCT-NEXT
POPJ-XCT-NEXT
```

The POPJ-XCT-NEXT operation is not to be confused with POPJ-AFTER-NEXT, which may be used in any instruction to set the POPJ bit.

Jump instructions which perform an arithmetic comparison should have both an A and an M source; the sources are compared. Jump instructions which test a bit should have an M source and a byte specifier for a 1-bit byte to test.

Functional Sources

The following names are supplied by CONSLP for the various functional sources:

0	READ-I-ARG	The dispatch constant
1	MICRO-STACK-PNTR-AND-DATA	SPCPTR and SPC contents
	MICRO-STACK-POINTER	Byte specifier for bits <28-24>
	MICRO-STACK-DATA	Byte specifier for bits <18-0>
14	MICRO-STACK-PNTR-AND-DATA-POP	Like 1, but pops SPC stack
	MICRO-STACK-POINTER-POP	Like 1, but pops SPC stack
	MICRO-STACK-DATA-POP	Like 1, but pops SPC stack
2	PDL-BUFFER-POINTER	PDL-pointer register
3	PDL-BUFFER-INDEX	PDL-index register
5	C-PDL-BUFFER-INDEX	PDL-buffer addressed by index
25	C-PDL-BUFFER-POINTER	PDL-buffer addressed by pointer
24	C-PDL-BUFFER-POINTER-POP	PDL-buffer addressed by pointer, pop
6	OPC-REGISTER	The OPCs
7	Q-R	Q register
10	VMA	VMA register
11	MEMORY-MAP-DATA	MAP[MD]
12	MEMORY-DATA	MD
13	LOCATION-COUNTER	LC

Functional Destinations

The following names are provided by CONSLP for functional destinations. Note that some of them are the same names used for sources; CONSLP distinguishes usage by context.

1	LOCATION-COUNTER	LC
2	INTERRUPT-CONTROL	Interrupt Control Register
10	C-PDL-BUFFER-POINTER	Pdl location addressed by PDL POINTER
11	C-PDL-BUFFER-POINTER-PUSH	Push data onto pdl, increment PDL POINTER
12	C-PDL-BUFFER-INDEX	Pdl location addressed by PDL INDEX
13	PDL-BUFFER-INDEX	PDL INDEX register
14	PDL-BUFFER-POINTER	PDL POINTER register
15	MICRO-STACK-DATA-PUSH	Push data onto SPC stack
16	OA-REG-LOW	Next instruction modify, bits <25-0>
17	OA-REG-HI	Next instruction modify, bits <47-26>
20	VMA	VMA register
21	VMA-START-READ	VMA, initiate read cycle
22	VMA-START-WRITE	VMA, initiate write cycle
23	VMA-WRITE-MAP	VMA, MAP[MD] ← VMA
30	MEMORY-DATA	MD register
31	MEMORY-DATA-START-READ	MD, initiate read cycle
32	MEMORY-DATA-START-WRITE	MD, initiate write cycle
33	MEMORY-DATA-WRITE-MAP	MD, MAP[MD] ← VMA

The symbol Q-R may also be used as a destination; it causes an ALU instruction to have its Q control field to be set to "load Q from ALU output"; this is equivalent to specifying LOAD-Q in the instruction. Do not use the output bus shifter in connection with Q-R as a destination!

Operations Common to All Instructions

The symbol for the POPJ bit is POPJ-AFTER-NEXT.

Miscellaneous Function 3 is denoted by LOW-PC-BIT-SELECTS-HALF-WD. (This feature is described in greater detail in an earlier and a later section.)

Expression Programs in CONSLP

Wherever an expression may be used in CONSLP, the following arcane forms may be used. In particular, the value of a symbol is normally an expression instead of a simple number. Whenever an expression (or a symbol with an expression as its definition) is encountered, it is evaluated according to the following rules:

<number> Evaluates to itself.

(PLUS <exp1> <exp2>) Adds together the two expressions, and combines their properties (such as byte-specifier-ness).

(DESTINATION-P <exp>) A conditional: if encountered while assembling a destination, returns the value of <exp>, and otherwise NIL.

(SOURCE-P <exp>) A conditional: if encountered while assembling a source (M or A), returns the value of <exp>, and otherwise NIL.

(DISPATCH-INSTRUCTION-P <exp>) A conditional: if encountered while assembling a DISPATCH instruction, returns the value of <exp>, and otherwise NIL.

(JUMP-INSTRUCTION-P <exp>) A conditional: if encountered while assembling a JUMP instruction, returns the value of <exp>, and otherwise NIL.

(ALU-INSTRUCTION-P <exp>) A conditional: if encountered while assembling an ALU instruction, returns the value of <exp>, and otherwise NIL.

(BYTE-INSTRUCTION-P <exp>) A conditional: if encountered while assembling a BYTE instruction, returns the value of <exp>, and otherwise NIL.

(NOT <conditional>) Negation. <conditional> must be one of the above conditionsl forms.

(OR <cond1> ... <condn>) Like a LISP OR, returns the first non-NIL conditional.

(BYTE-FIELD <size> <pos>) As described earlier, defines a byte with the given size and position from the right.

(LISP-BYTE <ppss>) As described earlier; if ppss is written in octal, then this is like (BYTE-FIELD ss pp). If <ppss> is not a number, then it is a LISP expression (not a CONSLP expression!), and is evaluated in LISP.

(BYTE-MASK <byte specifier>) Value is a word which is zero everywhere except for being all ones in the specified byte. This is a kind of conditional, in that it returns NIL if the byte specifier doesn't really specify a byte.

(BYTE-VALUE <byte specifier> <value>) Value is a word which is zero everywhere, except that it contains <value> in the specified byte. This is a kind of conditional, in that it returns NIL if the byte specifier doesn't really specify a byte.

(OA-HIGH-CONTEXT <word>) Assembles <word> as an instruction, and returns the high half (bits <47-26>), as if for use by the OA register feature (next instruction modify, functional destination 17).

(OA-LOW-CONTEXT <word>) Assembles <word> as an instruction, and returns the low half (bits <25-0>), as if for use by the OA register feature (next instruction modify, functional destination 16).

(FORCE-DISPATCH <exp>) Returns value of <exp>, but also forces the instruction to be a DISPATCH instruction. A conflict causes an error.

(FORCE-JUMP <exp>) Returns value of <exp>, but also forces the instruction to be a JUMP instruction.

(FORCE-ALU <exp>) Returns value of <exp>, but also forces the instruction to be an ALU instruction.

(FORCE-BYTE <exp>) Returns value of <exp>, but also forces the instruction to be a BYTE instruction.

(FORCE-DISPATCH-OR-BYTE <exp>) Returns value of <exp>, but also forces the instruction to be a DISPATCH or BYTE instruction.

(FORCE-ALU-OR-BYTE <exp>) Returns value of <exp>, but also forces the instruction to be an ALU or BYTE instruction.

(I-MEM-LOC <tag>) Returns the address represented by <tag> in locality I-MEM as a right-justified value.

(D-MEM-LOC <tag>) Returns the address represented by <tag> in locality D-MEM as a right-justified value.

(A-MEM-LOC <tag>) Returns the address represented by <tag> in locality A-MEM as a right-justified value.

(M-MEM-LOC <tag>) Returns the address represented by <tag> in locality M-MEM as a right-justified value.

(EVAL <lisp exp>) Returns the result of evaluating in LISP the S-expression <exp>.

(FIELD <name> <value>) Makes a note that the field <name> has been specified, then multiplies together the values of <name> and <value>; if <name> has a LISP CONS-LAP-ADDITIVE-CONSTANT property, this is then added in. (This obscurity is the primitive from which all field specifications are made.)

(ERROR) Error if this is assembled. Useful in conditionals.

As examples of how conditionals might be used in expressions, consider these definitions (which are similar (but not identical) to the ones actually used in CONSLP):

```
(ASSIGN Q-R (OR (SOURCE-P (FIELD M-SOURCE 7))
                (FORCE-ALU 3)))

(ASSIGN MEMORY-DATA
        (OR (SOURCE-P (FIELD M-SOURCE 12))
            (FIELD FUNCTIONAL-DESTINATION 30)))

(ASSIGN MEMORY-DATA-START-WRITE
        (OR (SOURCE-P (ERROR))
            (FIELD FUNCTIONAL-DESTINATION 32)))
```

Miscellaneous Pseudo-Operations

Several identical words may be assembled consecutively by saying:

```
(REPEAT <count> <word>)
```

The location counter within the current locality may be set by

```
(LOC <value>)          ;sets it to <value>
(MODULO <n>)           ;advances it to the next multiple of <n>
```

If the MODULO operation is used in A-memory, wastage is avoided by filling in the skipped-over locations with constants.

CADR Features and Programming Examples

In this section the various features of the CADR machine are examined and discussed in detail. An attempt is made to give some feeling for how each feature fits into the overall structure of the machine, and the purposes for which the feature is intended. Short programming examples using each feature are presented.

Timing - The N Bit and the POPJ Bit

Because CADR fetches the next instruction at the same time it is executing the current one, by the time the effect of a JUMP or DISPATCH is known the instruction following the JUMP or DISPATCH has already been fetched. Unless suppressed by the N bit, this instruction is executed before the instruction branched to. The effect of this on programming is that one should "code the branch one instruction sooner". The mnemonics CONSLP provides for the various branching operations normally set the N bit, thus doing the straightforward thing at the cost of wasted cycles; one must append "-XCT-NEXT" to the mnemonic to clear the N bit and so bum the code.

For example, consider these two pieces of code:

```
((A-FOO) XOR M-BAR A-FOO)                    ;XOR M-BAR into A-FOO
(JUMP-IF-BIT-SET MUMBLE MUMBLIFY)            ;branch on MUMBLE bit

(JUMP-IF-BIT-SET-XCT-NEXT MUMBLE MUMBLIFY)   ;branch on MUMBLE bit
((A-FOO) XOR M-BAR A FOO)                    ;XOR M-BAR into A-FOO
```

These both perform an XOR and conditionally jump to MUMBLIFY, but the first one wastes a cycle if the JUMP is successful. Notice the convention of "exdenting" an instruction which is under the influence of an XCT-NEXT to make it more visible.

If a CALL transfer type is executed, the return address saved on the SPC stack depends on the N bit:

```
(CALL THE-SUBROUTINE)                ;call, N bit set
((A-FOO) XOR M-BAR A-FOO)            ;return here after call

(CALL-XCT-NEXT THE-SUBROUTINE)       ;call, N bit clear
((A-ARGUMENT) ADD M-BAZ A-FOO)       ;do this before entering the subroutine
((A-FOO) XOR M-BAR A-FOO)            ;return here after call
```

If the N bit is set, PC+1 is pushed on the SPC stack; otherwise PC+2 is pushed.

The POPJ bit may be set in any instruction. It causes a RETURN transfer, but only after the next instruction has

also been executed:

```
ADD-THREE-WORDS                 ;subroutine to add together A-1, A-2, and A-3
        ((M-RESULT) A-1)
        (POPJ-AFTER-NEXT (M-RESULT) ADD M-RESULT A-2)
       ((M-RESULT) ADD M-RESULT A-3)
```

Again, the idea is to specify the desired control "one instruction early".

Consider the following program:

```
START   (JUMP-XCT-NEXT FOO)
       (JUMP-XCT-NEXT BAR)
        ...
FOO     (JUMP-XCT-NEXT FOO)
        ...
BAR     (JUMP-XCT-NEXT BAR)
```

When started at START, it will go into an infinite loop alternately executing FOO and BAR. Effectively it is in two "jump point" loops at the same time!

Byte Manipulation

By using M location 2 (by convention a source of zeros) with a BYTE instruction, one can clear any bit or field of bits in any A memory location:

```
        ((A-FOO) DPB M-ZERO A-FOO (BYTE-FIELD 1 31.))           ;clear sign bit
```

It is often convenient to reserve another M memory location to contain -1 (all ones), in order to be able to set bits easily:

```
        ((A-FOO) DPB M-ONES A-FOO (BYTE-FIELD 1 31.))           ;set sign bit
```

In a similar manner one can write a routine to extend a signed 24-bit number to 32 bits:

```
SIGN-EXTEND                                     ;extend 24-bit number in M-NUM
        (POPJ-AFTER-NEXT POPJ-IF-BIT-CLEAR M-NUM (BYTE-FIELD 1 23.))
       ((M-NUM) SELECTIVE-DEPOSIT M-NUM (BYTE-FIELD 24. 0) (A-CONSTANT -1))
```

Another way to do this, which doesn't require the use of POPJ, is to use OA modification to select whether the M source is M-ZERO or M-ONES:

```
        ((OA-REG-HI) (BYTE-FIELD 1 23.) M-NUM)          ;low M-source bit gets sign
       ((M-NUM) SELECTIVE-DEPOSIT M-ZERO (BYTE-FIELD 8 24.) A-NUM)
```

This requires that M-ZERO and M-ONES be an even/odd pair.

Normally bytes can only be loaded from an M source. However, it is possible to load a byte from A-memory, provided that it is at one end of the word, by the following trick:

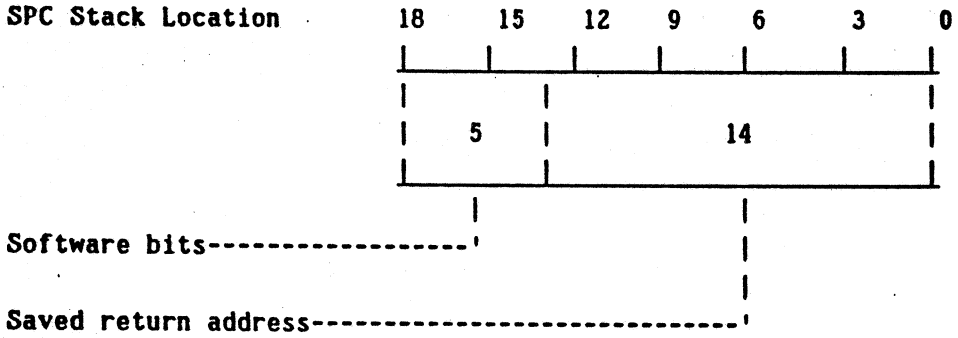
```
(DEF-DATA-FIELD X-FIELD 6 0)
(DEF-DATA-FIELD ALL-BUT-X-FIELD 32 6)

        ((DEST) SELECTIVE-DEPOSIT M-ZERO ALL-BUT-X-FIELD A-FOO)
```

The Instruction Stream

<<Some new stuff should be written for this>>

The SPC Stack

The SPC stack is 32 locations long, each location containing 19 bits (plus parity). It is indexed by SPCPTR, a 5-bit up/down counter. It is used primarily as a microcode subroutine return stack, but besides the 14 bits needed to save a microcode PC there are 5 bits for software use, one of which is the bit used for the macroinstruction pair fetch feature mentioned above.



There are two ways in which to write into the SPC stack memory; both of them also increment SPCPTR, thus causing a push operation. A JUMP or DISPATCH performing a CALL transfer type (P bit set, R bit clear) causes a return address to be pushed on the stack as described earlier. The five software bits are set to zero. Writing into functional destination 15 (MICRO-STACK-DATA-PUSH) pushes the low 19 bits of the output bus data onto the SPC stack.

The SPC stack is read by a JUMP or DISPATCH performing a RETURN transfer type (R bit set, P bit clear); the low 14 bits popped off the stack are put in the PC, and the software bits are ignored, except for bit 14 which causes NEXT-INSTR. It

can also be read as M functional sources 1 and 14. The first (MICRO-STACK-PNTR-AND-DATA) merely reads the data (and SPCPTR) on the top of the stack, while the second (MICRO-STACK-PNTR-AND-DATA-POP) pops the stack after reading the data.

There is no way to explicitly set the contents of SPCPTR. However, a good trick is to use the following loop:

```
FOO     ((M-TEMP) MICRO-STACK-POINTER-POP)          ;get just SPCPTR
        (JUMP-IF-EQUAL M-TEMP A-ZERO FOO)
```

A better trick is to use the following loop, which not only is shorter, but is recursive rather than iterative, and has the important advantage of being more obscure:

```
FOO     (CALL-NOT-EQUAL MICRO-STACK-PNTR-AND-DATA
                (A-CONSTANT (PLUS 1 (I-MEM-LOC FOO))) FOO)
```

This is a good thing to do on initialization so that the stack will begin in a known place, thus aiding debugging via the diagnostic interface.

There is no provision for detection of SPC stack overflow or underflow. It is the responsibility of the programmer to avoid nesting subroutines to a depth greater than 32.

The PDL BUFFER Memory

The PDL BUFFER is intended to be used as a special-purpose cache in the Lisp machine to contain the top portion of the Lisp pushdown stack. It has 1024 locations of 32 bits, and can be indexed by either the PDL POINTER or the PDL INDEX. PDL POINTER is a 10-bit up/down counter, while PDL INDEX is simply a 10-bit register.

The PDL BUFFER is manipulated through various functional sources and functional destinations. The PDL POINTER and PDL INDEX registers may be read and written. (On CONS, these could only be read together, but on CADR they are read separately to facilitate doing arithmetic with them without the need to extract a byte first.) The contents of the PDL BUFFER location addressed by the contents of PDL INDEX may be read and written. The contents of the location addressed by the contents of PDL POINTER may also be read and written, and in this case the PUSH and POP operations may optionally be done by incrementing or decrementing the PDL POINTER. The pointer decrements after reading and increments before writing, so it always points to the topmost valid location.

It doesn't work to specify both C-PDL-BUFFER-POINTER-PUSH and C-PDL-BUFFER-POINTER-POP in the same instruction. On the other hand, the same effect can always be achieved simply by using C-PDL-BUFFER-POINTER for both source and destination instead.

There is no provision for automatic overflow or underflow detection on pushes and pops of the PDL BUFFER. In the Lisp machine, the PDL POINTER is checked on entry to every function, and at a few other necessary places. If there is insufficient room left within the PDL BUFFER for a maximum size frame, some of the PDL BUFFER is stored into main memory to make room. If there is also insufficient space left within the

virtual memory allocated to the PDL, a PDL-OVERFLOW error is signalled. Similarly, the function exit code decides whether to pull some stack back in from main memory.

	REGISTER MEMORY J1			ALU J2			PROGRAM COUNTER J3			INST MEMORY ONE J4			INST MEMORY TWO J5			MAIN MEMORY INTER J6		
		1	2 ●M0			00			●		1	2 ○ M0						●
		3	4 ●M1			00			●		3	4 ○ M1						●
30		5	6 ●M2	○		00			●		5	6 ○ M2					●	●
		7	8 ●M3	○		00			●		7	8 ○ M3					●	●
		9	10 GND				●		●		9	10 GND					○	
		11	12 ●M16	●		00	○		●		11	12 ○ M16						●
29		13	14 ●M17	●		00	○		●		13	14 ○ M17						●
		15	16 ●M18	○		00	○		●		15	16 ○ M18						●
		17	18 ●M19	○		00	○		●		17	18 ○ M19						●
		19	20 GND								19	20 GND						
		21	22 IR.DISPM	○			○		●		21	22 ○ IRDISP 24						○
28		23	24 NEXT INSTR	○					●		23	24 NEXT INSTR						○
		25	26 ○ IR32				○		○		25	26 ○ -VMD16	●					●
		27	28 ○ IR33						○		27	28 ○ -VMD19	●					●
		29	30 GND								29	30 GND						
		31	32 ○ OB0			●	○		○		31	32 ○ OB0	●					○
27		33	34 ○ OB1			●	○		○		33	34 ○ OB1	●					○
		35	36 ○ OB2	●		●	○		○		35	36 ○ OB2		○				○
		37	38 ○ OB3	●		●	○		○		37	38 ○ OB3		○				○
		39	40 GND								39	40 GND						
		41	42 ○ OB16	●		●	○		○		41	42 ○ OB16	○●		○		○●	○
26		43	44 ○ OB17	●		●	○		○		43	44 ○ OB17	○●		○		○●	○
		45	46 ○ OB18	●		●	○		○		45	46 ○ OB18	○●		○		○●	○
		47	48 ○ OB19	●		●	○				47	48 ○ OB19	○●		○		○●	○
		49	50 GND								49	50 GND						
		51 ○ IR16	52 ●A0			○	○		○		51	52 A0		○				
		53 ○ IR17	54 ●A1			○	○		○		53	54 A1		○				
25		55 ○ IR18	56 ●A2			○	○		○		55	56 A2		○				
		57 ○ IR19	58 ●A3			○	○		○		57	58 A3		○			○	
		59	60 GND								59	60 GND						
		61 ○ -STATBIT	62 ●A16			○	○		○		61	-STATBIT	62					
24		63	64 ●A17	●		○	○		○		63	-STC20	64 ○ -LDADDRL	○				●
		65 ○ IR6	66 ●A18	●		○	○				65	-STC24	66 ○ -LDADDRM	●	○			●
		67 ○ IR8	68 ●A19	●		○	○				67	-JPARITY	68 -LDDDRM	●	○		○	●
		69	-MAMS 70 GND				○				69	-MAMS	70 GND				●	
		71	-PC0 72 -SPY.FLAG2	●		●	○		○		71	-PC0	72 ○ -SPY.IRL	○	●			
		73	-PC1 74 "ASSIGNED"	●			○				73	-PC1	74 ○ -SPY.IRM	○	●			
23		75	-PC2 76 -NEEDFRESH	●			○		○		75	-PC2	76 -NEEDFRESH	○				●
		77	-PC3 78 PDLWRITED	●			○				77	-PC3	78 PDLWRITED	○				○
		79	+5 80 GND								79	+5	80 GND					
		81 ○ IR20	82 ●M4			00	○		●		81	IR20	82 ○ M4	●			○	●
22		83 ○ IR21	84 ●M5			00	○		●		83	IR21	84 ○ M5	●			○	●
		85 ○ IR22	86 ●M6			00	○		●		85	IR22	86 ○ M6	●			○	●
		87 ○ IR23	88 ●M7			00	○		●		87	IR23	88 ○ M7	●			○	●
		89	-MACHRUNA 90 GND				○				89	-MACHRUNA	90 GND				●	
		91	SPY4 92 ●M20	●		00	○		●		91	SPY4	92 M20	00	○		00	●
21		93	SPY5 94 ●M21	●		00	○		●		93	SPY5	94 M21	00	○		00	●
		95	SPY6 96 ●M22	●		00	○		●		95	SPY6	96 M22	00	○		00	●
		97	SPY7 98 ●M23	●		00	○		●		97	SPY7	98 M23	00	○		00	●
		99	+5 100 GND								99	+5	100 GND					
		101	IR4 102 ○ OB4	○		●	○		●		101	IR4	102 ○ OB4		○			○
20		103	IR5 104 ○ OB5	○		●	○		●		103	IR5	104 ○ OB5		○			○
		105	IR6 106 ○ OB6	○		●	○		●		105	IR6	106 ○ OB6		○			○
		107	IR7 108 ○ OB7	○		●	○		●		107	IR7	108 ○ OB7		○			○
		109	-CLOCKRST.B 110 GND				○				109	-CLOCKRST.B	110 GND				●	
		111 ○ IR36	112 ○ OB20			●	○		●		111	IR36	112 OB20	●	○			○
19		113 ○ IR37	114 ○ OB21			●	○		●		113	IR37	114 OB21	●	○			○
		115 ○ IR38	116 ○ OB22			●	○		●		115	IR38	116 OB22	●	○			○
		117 ○ IR39	118 ○ OB23			●	○		●		117	IR39	118 OB23	●	○			○
		119	+5 120 GND								119	+5	120 GND					
		121	●-DESTARC 122 ●A4			○	○		○		121		122 A4		○			
18		123	-JWRITED 124 ●A5			○	○		○		123	-JWRITED	124 A5		○		○	
		125	-JDWRITE 126 ●A6			○	○		○		125	-JDWRITE	126 A6		○		●	
		127	-JDREAD 128 ●A7	○		○	○		○		127	-JDREAD	128 A7	●	○		●	
		129	-MCLK 130 GND				●		●		129	-MCLK	130 GND				○	
		131	-PC4 132 ●A10			○	●		●		131	-PC4	132 ○ -DEST2MOD0	○	○			●
		133	-PC5 134 ●A11			○	●		●		133	-PC5	134 ○ -DEST2MOD1	○	○			●
17		135	-PC6 136 ●A12			○	●		●		135	-PC6	136 ○ -SPY STL	○				●
		137	-PC7 138 ●A13			○	●		●		137	-PC7	138 ○ -SPY STR	○	○			●
		139	+5 140 GND				●		●		139	+5	140 GND					
15,16																		
		1 ○ CLKN	2 ●M8	○		00	●		●		1	○ CLKN	2 ○ M8	○			○	●
14		3	-JWE 4 ●M9			00	●		●		3	○ -JWE	4 ○ M9	○				●
		5 ○ -WP	6 ●M10			00	●		●		5	-WP	6 ○ M10				○	●
		7 ○ -TSE	8 ●M11	○		00	●		●		7	-TSE	8 ○ M11				○	●
		9	SR.PARDR 10 GND								9	SR.PARDR	10 GND				○	
		11 ○ -NOP	12 ●M24	○		00	●		●		11	-NOP	12 ○ M24					●
13		13 ○ RESET	14 ●M25			00	●		●		13	RESET	14 ○ M25				●	●
		15 IR8	16 ●M26	○		00	●		●		15	○ IR8	16 ○ M26					●
		17 IR9	18 ●M27	○		00	●		●		17	○ IR9	18 ○ M27					●
		19	+5 20 GND								19	+5	20 GND					
		21 IRID	22 EADRD			○	○		○		21	○ IRID	22 EADRD		○		○	●
12		23 IR11	24 EADR1			○	○		○		23	○ IR11	24 EADR1		○		○	●
		25 IR40	26 EADR2			○	○		○		25	IR40	26 EADR2	●	○			●
		27 IR41	28 EADR3			●	●		●		27	IR41	28 EADR3	●	○			●
		29 DPARDR	30 GND				●		●		29	DPARDR	30 GND				○	
		31 IR42	32 ○ OB8			●	○		○		31	IR42	32 ○ OB8	●	○			○
11		33 IR43	34 ○ OB9	○		●	○		○		33	IR43	34 ○ OB9	●	○			○
		35 ○ DEST	36 ○ OB10	●		●	○		○		35	DEST	36 ○ OB10		○		○	○
		37 SPY8	38 ○ OB11	●		●	○		○		37	○ SPY8	38 ○ OB11	○●	○		○●	○
		39	+5 40 GND								39	+5	40 GND					
		41	-SPY.DR 42 ○ OB24	●		●	○		○		41		42 OB24					○
10		43	-SPY PC 44 ○ OB25	●		●	○		○		43		44 OB25					○
		45 SPY9	46 ○ OB26	●		●	○		○		45	○ SPY9	46 OB26	○●			○●	○
		47 SPY10	48 ○ OB27	●		●	○		○		47	○ SPY10	48 OB27	○●			○●	○
		49	-NOPA 50 GND				●		●		49	-NOPA	50 GND				○	
		51 SPY11	52 ●A8	●	○		○	○			51	○ SPY11	52 A8	○●	○		○●	
9		53 ●-ERCR	54 ●A9	○	○		○	○			53		54 A9		○			
		55 IR24	56 ●A10		○		○	○			55	○ IR24	56 A10		○			
		57 ○ IR25	58 ●A11		○		○	○			57	○ IR25	58 A11		○		○	
		59	+5 60 GND								59	+5	60 GND					
		61	-RC8 62 ●A24	●	○		○	●			61	○ -RC8	62 ○ JUMPI	○	○			
8		63	-RC9 64 ●A25	●	○		○	●			63	○ -RC9	64 ○ IRINKI	○	○			
		65	-RC10 66 ●A26	●	○		○	●			65	○ -RC10	66 ○ -STC26	○	○			
		67	-RC11 68 ●A27	●	○		○	●			67	○ -RC11	68 -STC32	○	●			○
		69	-IDEBUS 70 GND				○				69	○ -IDEBUS	70 GND	○				
		71	-MPH 72 ●M12			00	○	○	●		71		72 ○ M12				●	●
7		73	-VMACK 74 ●M13			00	○	○	●		73		74 ○ M13				●	●
		75 SPY12	76 ●M14	●		00	○	○	●		75	○ SPY12	76 ○ M14	○●			○●	●
		77 SPY13	78 ●M15	●		00	○	○	●		77	○ SPY13	78 ○ M15	○●			○●	●
		79	+5 80 GND								79		80 GND					
		81 SPY14	82 ●M28	●		00	○	○	●		81	○ SPY14	82 M28	○●	○		○●	●
6		83 SPY15	84 ●M29	●		00	○	○	●		83	○ SPY15	84 M29	○●	○		○●	●
		85 ○ IR26	86 ●M30			00	○	○	●		85	○ IR26	86 M30		○		○	●
		87 ○ IR27	88 ●M31			00	○	○	●		87	○ IR27	88 M31		○		○	●
		89	RAMDISABLE 90 GND				○				89	○ RAMDISABLE	90 GND	○				
		91 ○ IR28	92 PROMENABLE				○		●		91	IR28	92 ○ PROMENABLE		○			
5		93 ○ IR29	94 MMEMPARDR				○				93	IR29	94 MMEMPARDR	●			○	
		95 ○ IR30	96 PDLPARDR								95	IR30	96 PDLPARDR	●			○	
		97 ○ IR31	98 APARDR				○				97	IR31	98 APARDR	●			○	
		99	+5 100 GND								99	+5	100 GND					
		101 S3NT	102 ○ OB12			●	○		●		101	S3NT	102 ○ OB12		○		●	○
4		103 SEQUENCE BRK	104 ○ OB13			●	○		●		103	SEQUENCE BRK	104 ○ OB13		○		●	○
		105 -PC12	106 ○ OB14			●	○		●		105	-PC12	106 ○ OB14	○	○			○
		107 -PC13	108 ○ OB15			●	○		●		107	-PC13	108 ○ OB15	○	○			○
		109 -PROMDISABLED	110 GND				○				109	-PROMDISABLED	110 GND				●	
		111 IR44	112 ○ OB28	○		●	○		●		111	IR44	112 OB28	●				○
3		113 IR45	114 ○ OB29			●	○		●		113	IR45	114 OB29	●				○
		115 IR46	116 ○ OB30			●	○		●		115	IR46	116 OB30	●				○
		117 IR47	118 ○ OB31			●	○		●		117	IR47	118 OB31	●				○
		119	+5 120 GND								119	+5	120 GND					
		121 IR12	122 ●A12	○		○	○		○		121	IR12	122 A12		○			
2		123 ○ IR13	124 ●A13	○		○	○		○		123	○ IR13	124 A13		○			
		125 ○ IR14	126 ●AN			○	○		○		125	○ IR14	126 AN		○			
		127 ○ IR15	128 ●A15			○	○		○		127	○ IR15	128 A15		○			
		129 -HALT	130 GND				●		●		129	○ -HALT	130 GND				○	
		131 ARM	132 ●A28	●		○	●		●		131		132 ○ ERR					●
1		133 ALU31	134 ●A29	●		○	●		●		133		134	○				
		135 -TRAP	136 ●A30			○	●		●		135	-TRAP	136				●	
		137	138 ●A31			○	●		●		137		138 IR48		●			○
		139 +5	140 GND				●		●		139	+5	140 GND					

R

L

ELECTRICAL CONNECTIVITY OF J1, J2 & J3 IS IDENTICAL
ELECTRICAL CONNECTIVITY OF J4, J5 & J6 IS IDENTICAL

TYPICAL PIN DESIGNATION IS J2L36
- J: J1 THRU J6
- 2: 1 THRU 140
- L: L OR R
- 36: 1 THRU 6

Ⓜ INST IN 2ND OR 4TH QUARTER
Ⓛ INST IN LEFT HALF

○ USE OF SIGNAL
● SOURCE OF SIGNAL

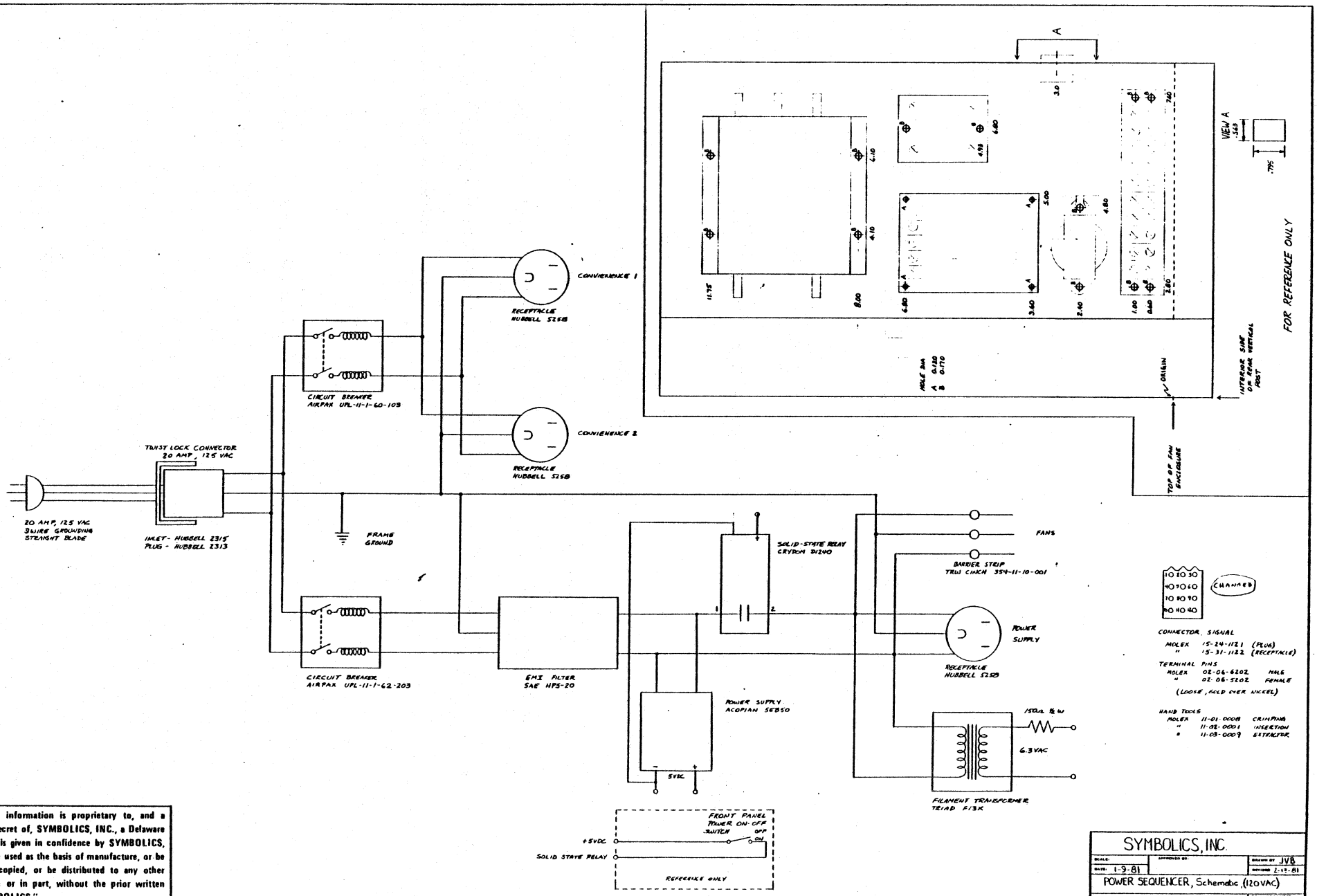
REVISIONS
A R14 OF J1,J2,J3 ASSIGNED 9-2-81 JVB

SYMBOLICS INC
855 HAMILTON ROAD SANTA MONICA CALIF 90291

SCHEMATIC, BACKPANEL

DRAWN 11-2-80 JVB

0009903 A

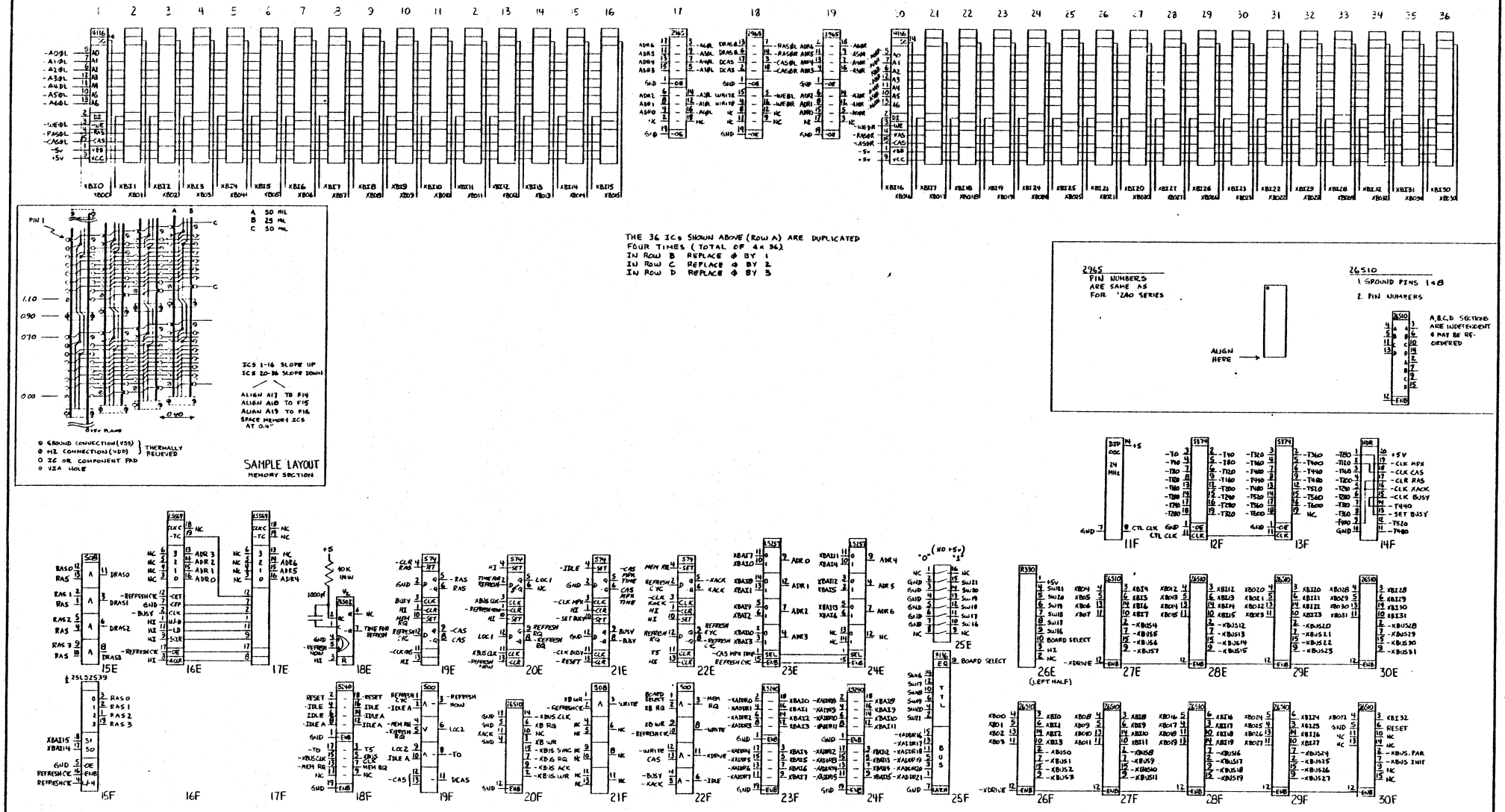


SYMBOLICS, INC.

SCALE:	APPROVED BY:	DRAWN BY JVB
DATE: 1-9-81		REVISED 2-17-81

POWER SEQUENCER, Schematic (120VAC)

DRAWING NUMBER 101091



PC BOARD 101201
ASSY 101202

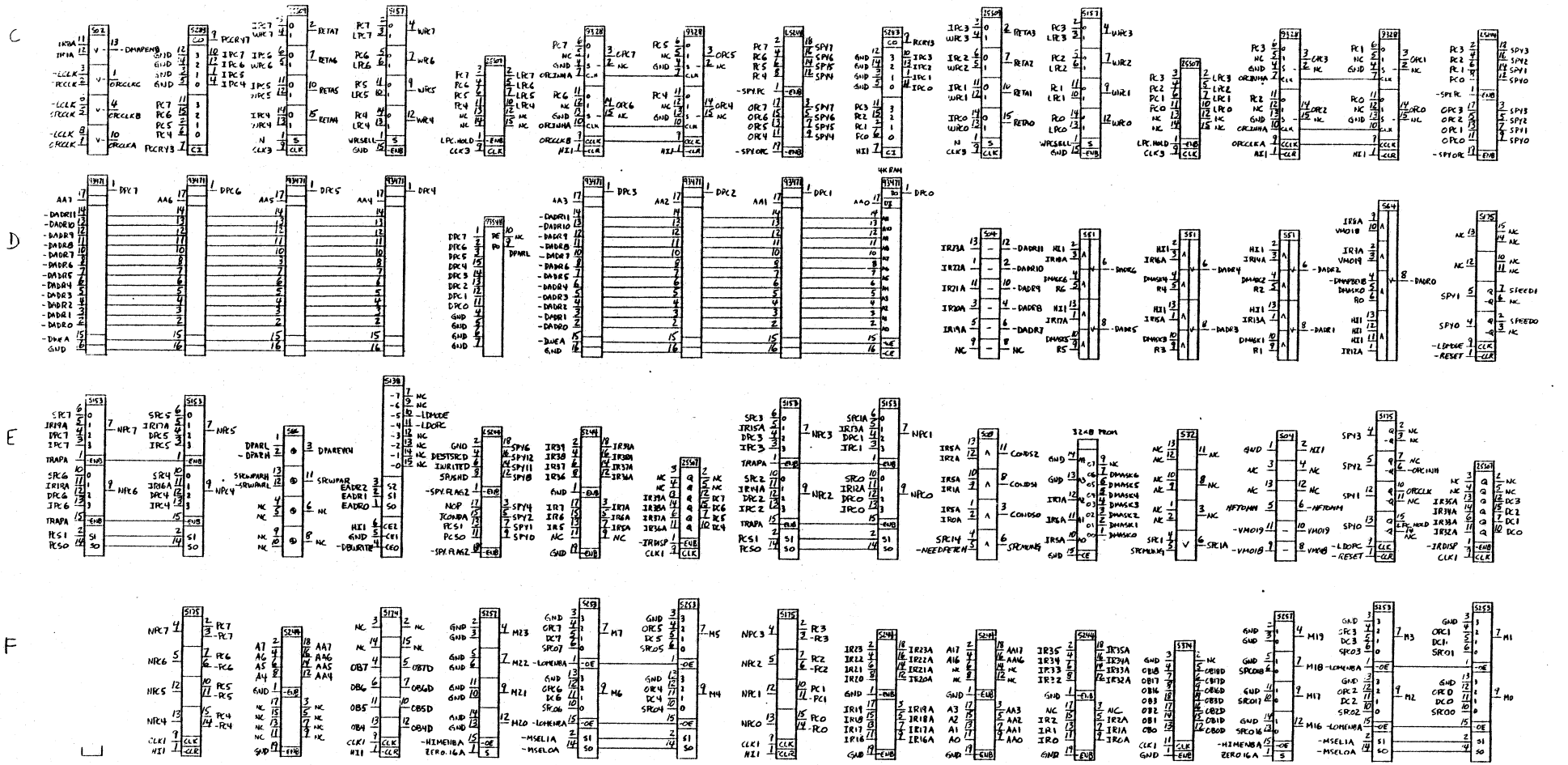
LOC 1E34561

SYMBOLICS INC
605 NIGHTREE ROAD SANTA MONICA, CA 90402

SCALE	APPROVED BY:	DRAWN BY JVD
DATE 1-29-81		REVISED D

MEMORY, Schematic

DRAWING NUMBER 101200



NOTES:

1

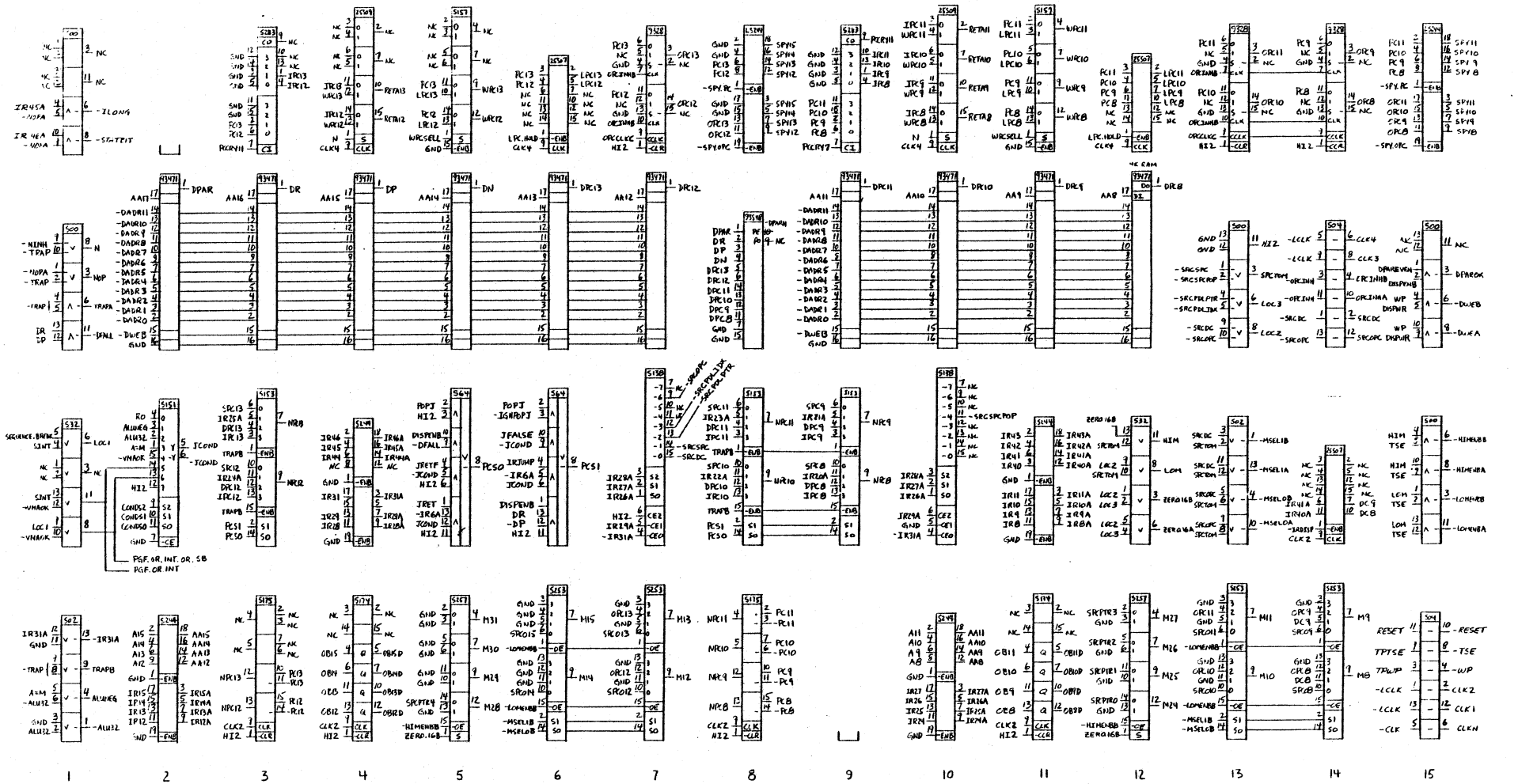
PC BOARD 009191
ASSY 009192

SYMBOLICS, INC.

PROGRAM COUNTER Sheet 1 of

Schematic 009190

John V. Blankenship
November 11, 1980
Rev B



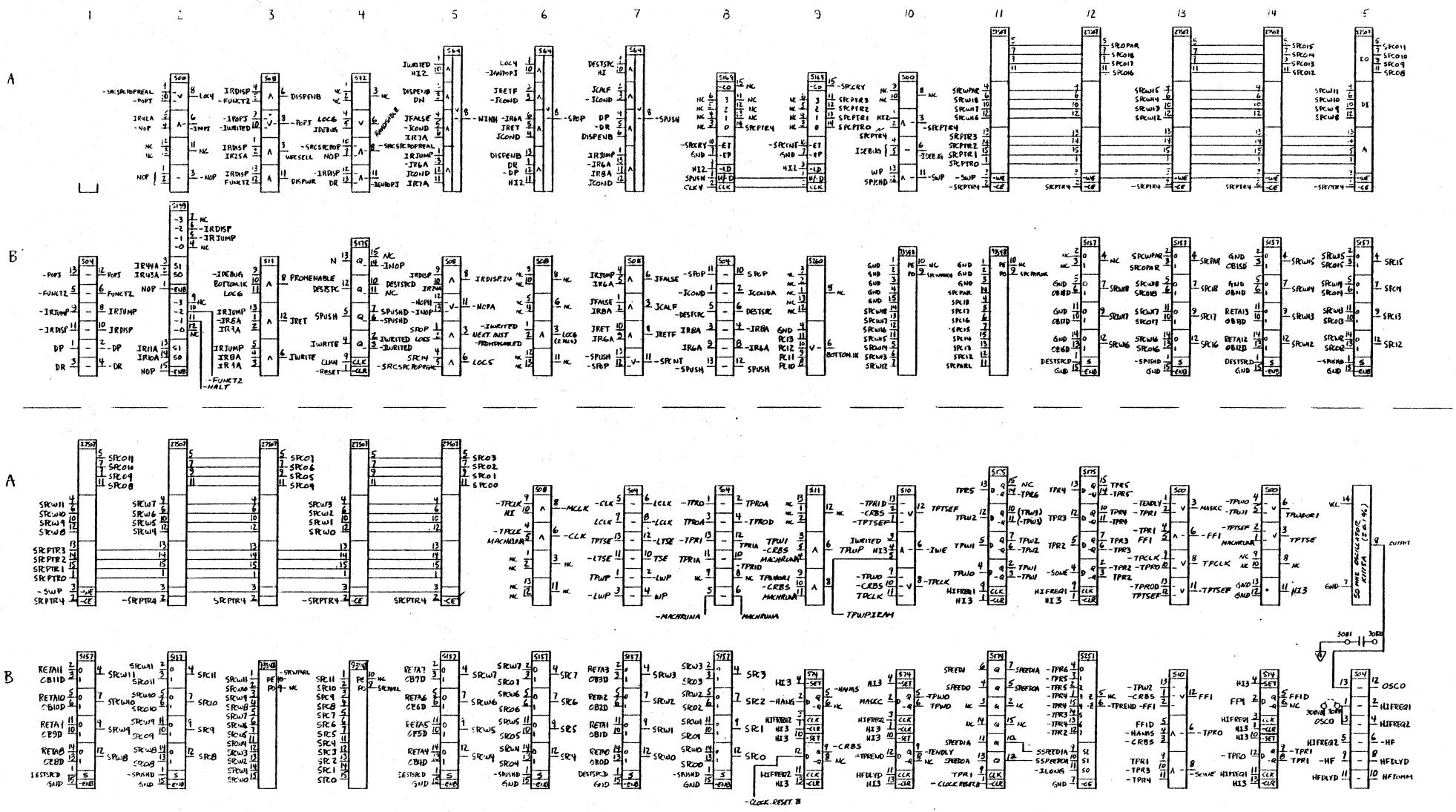
SYMBOLICS, INC.

PROGRAM COUNTER

Sheet 2 of 3

Schematic 0091190

John V Blankenbaker
November 11, 1980
Rev B



SYMBOLICS, INC.

PROGRAM COUNTER

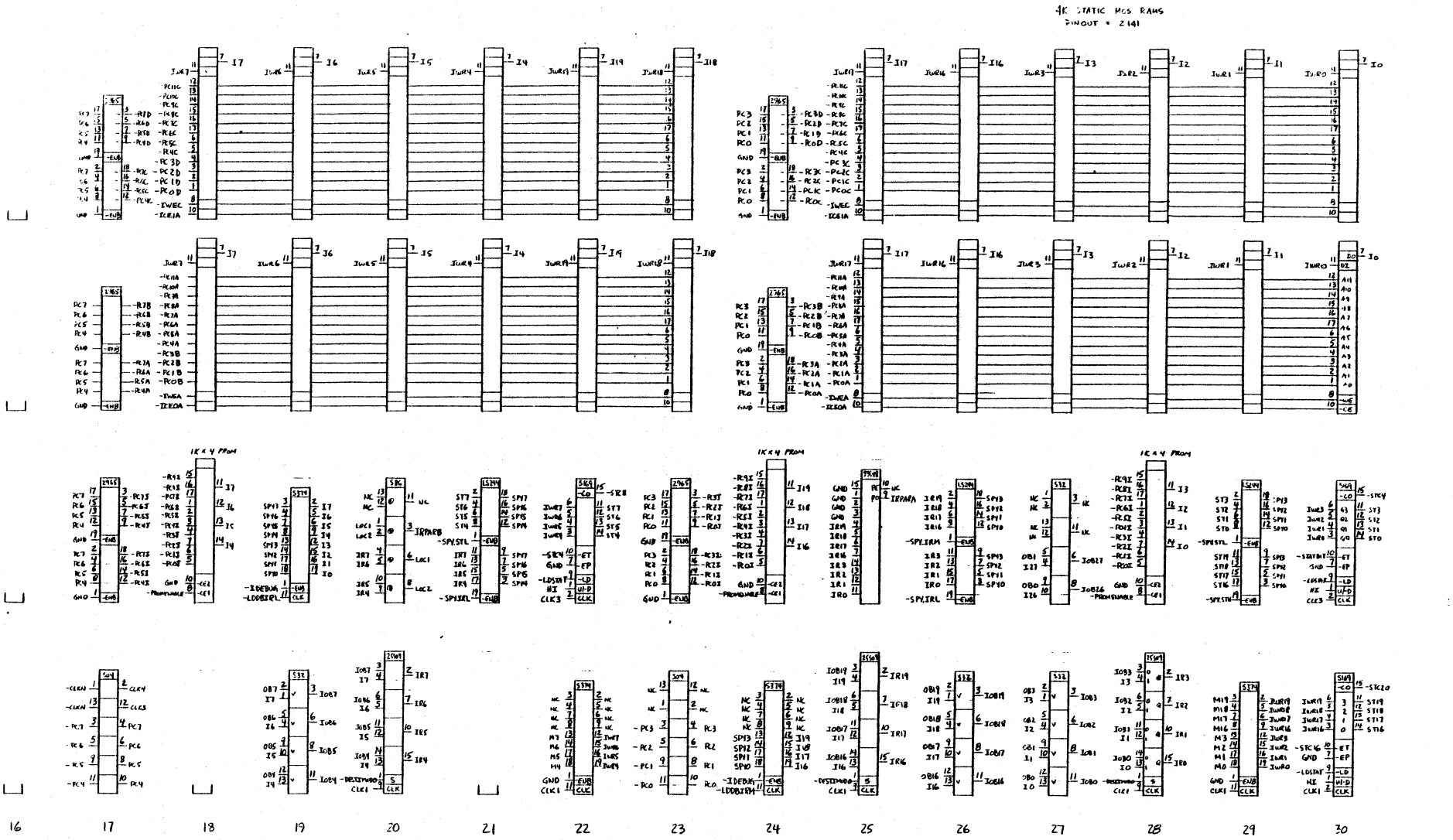
Sheet 3 of 3

Schematic 009190

John V Blankenbaker

November 11, 1980

Rev B



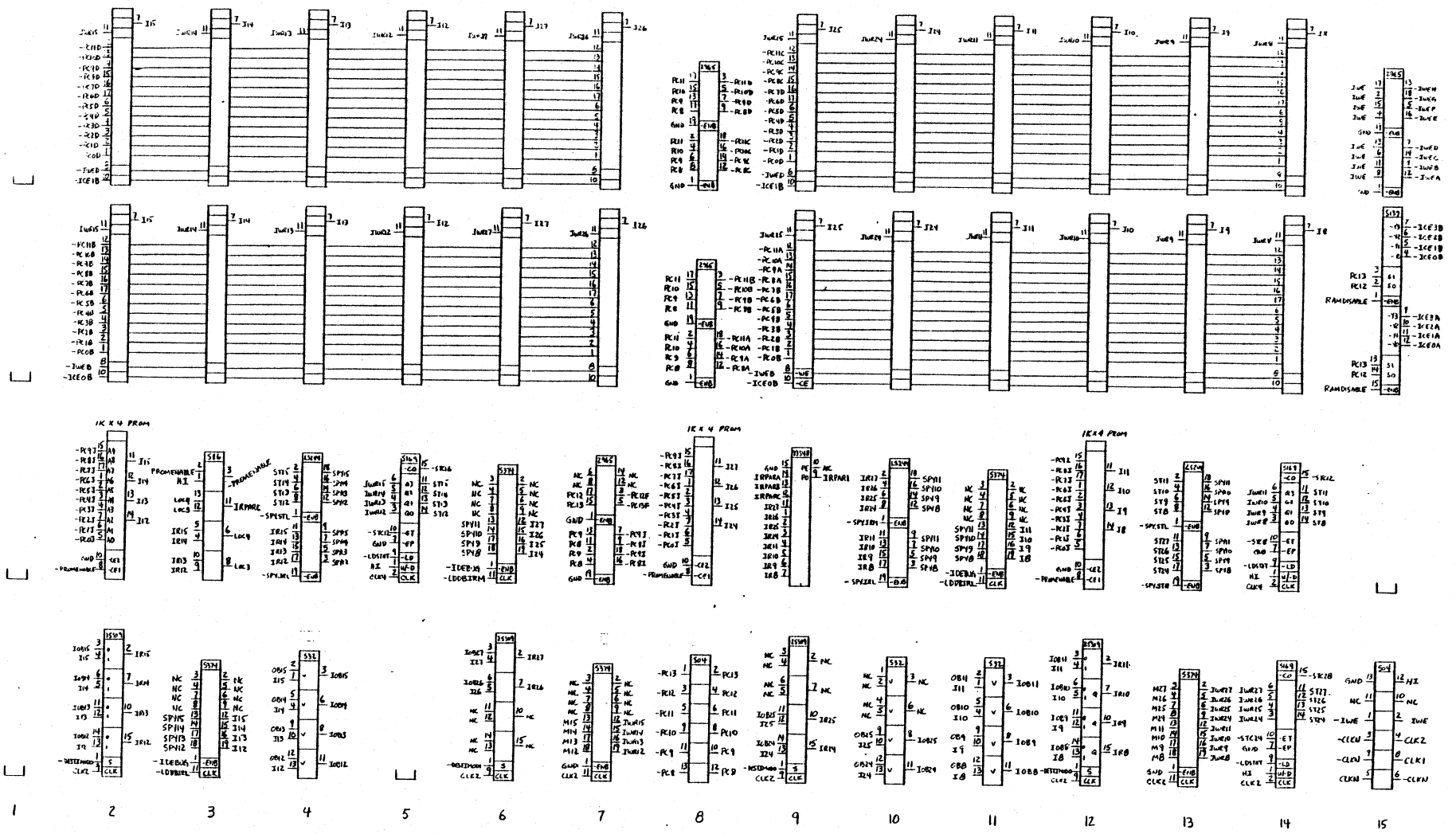
SYMBOLICS, INC.

INSTRUCTION MEMORY ONE Sheet 1 of 5

007200

John V. Blankenbaker
Sept 22, 1980

LOC XXXXXX789



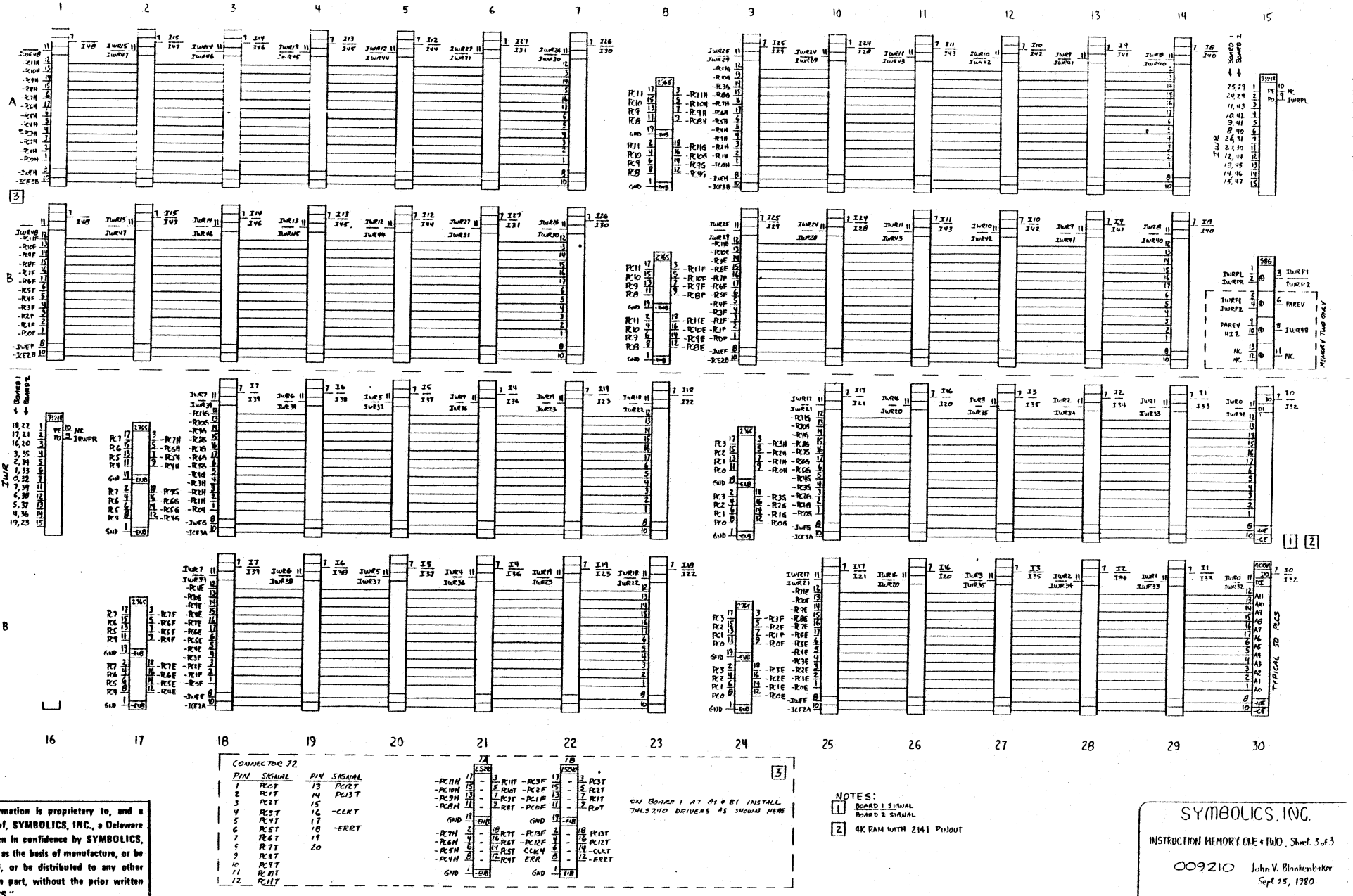
LOC X X X X 5 6 7 8 9

SYMBOLICS, INC

INSTRUCTION MEMORY ONE Sheet 2 of 3

009200 John V Blankenbaker Sept 22, 1980

NOTE: SEE 009210 FOR SHT 3



CONNECTOR J2

PIN	SIGNAL	PIN	SIGNAL
1	PC0T	13	PC12T
2	PC1T	14	PC13T
3	PC2T	15	
4	PC3T	16	-CLKT
5	PC4T	17	
6	PC5T	18	-ERRT
7	PC6T	19	
8	PC7T	20	
9	PC8T		
10	PC9T		
11	PC10T		
12	PC11T		

NOTES:

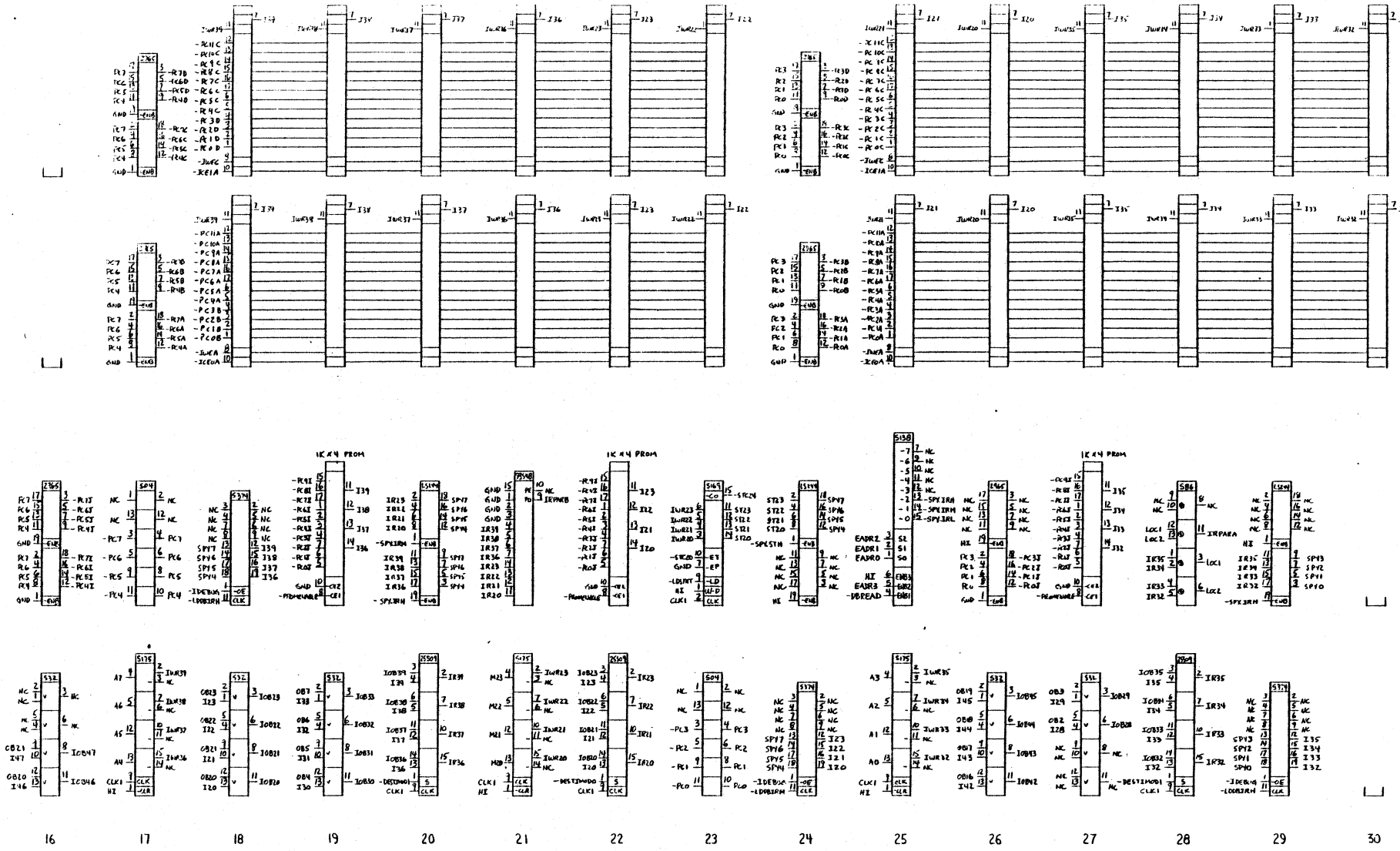
[1] BOARD 1 SIGNAL
BOARD 2 SIGNAL

[2] 4K RAM WITH 2141 PINOUT

SYMBOLICS, INC.

INSTRUCTION MEMORY ONE & TWO, Sheet 3 of 3

009210 John V. Blankenbaker
Sept 25, 1980



SYMBOLICS, INC.

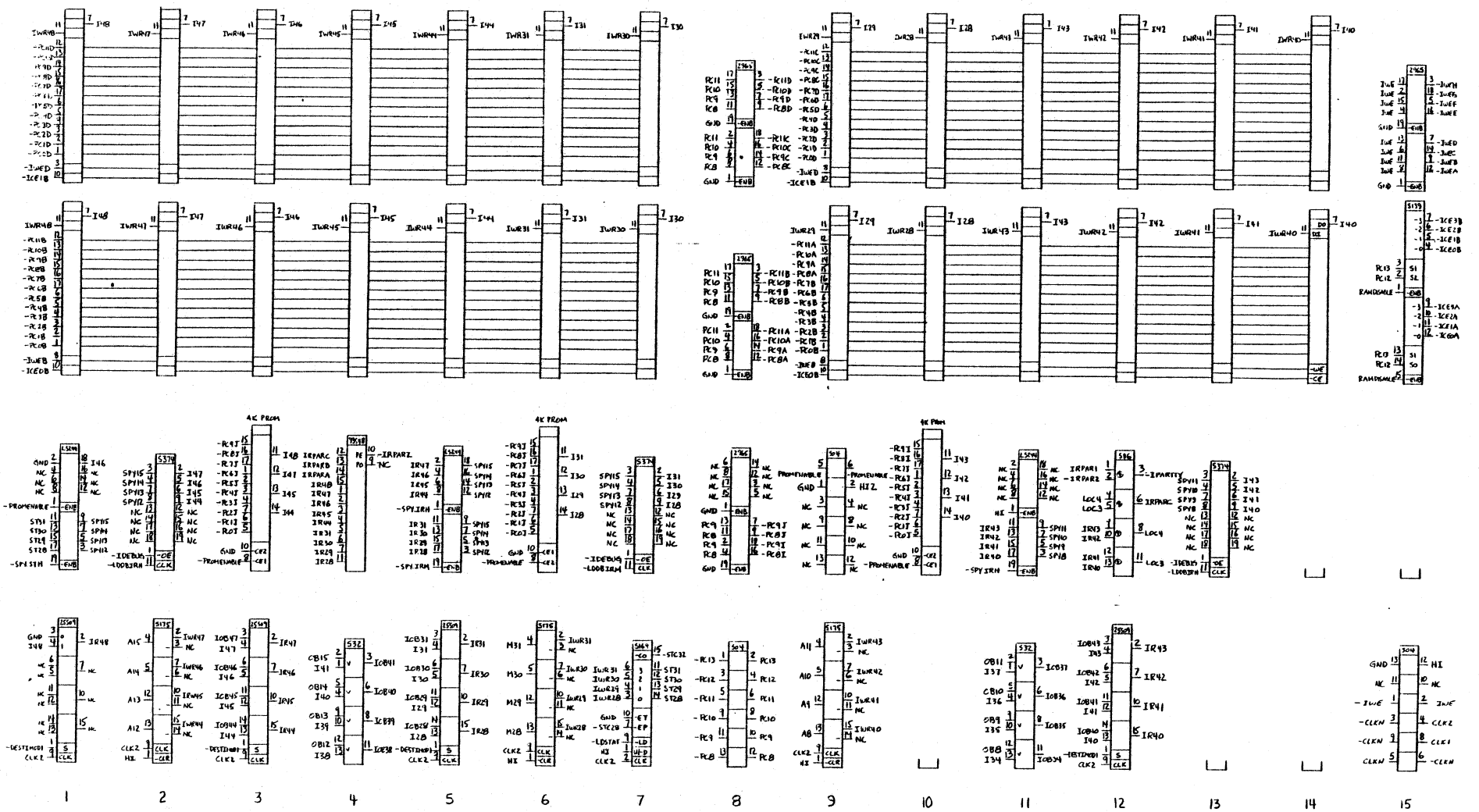
INSTRUCTION MEMORY TWO, Sheet 1 of 5

009210 John V Blankenbaker

Sept 25, 1980

Rev A

LOC 1 2 3 4 5 6 7 8 9

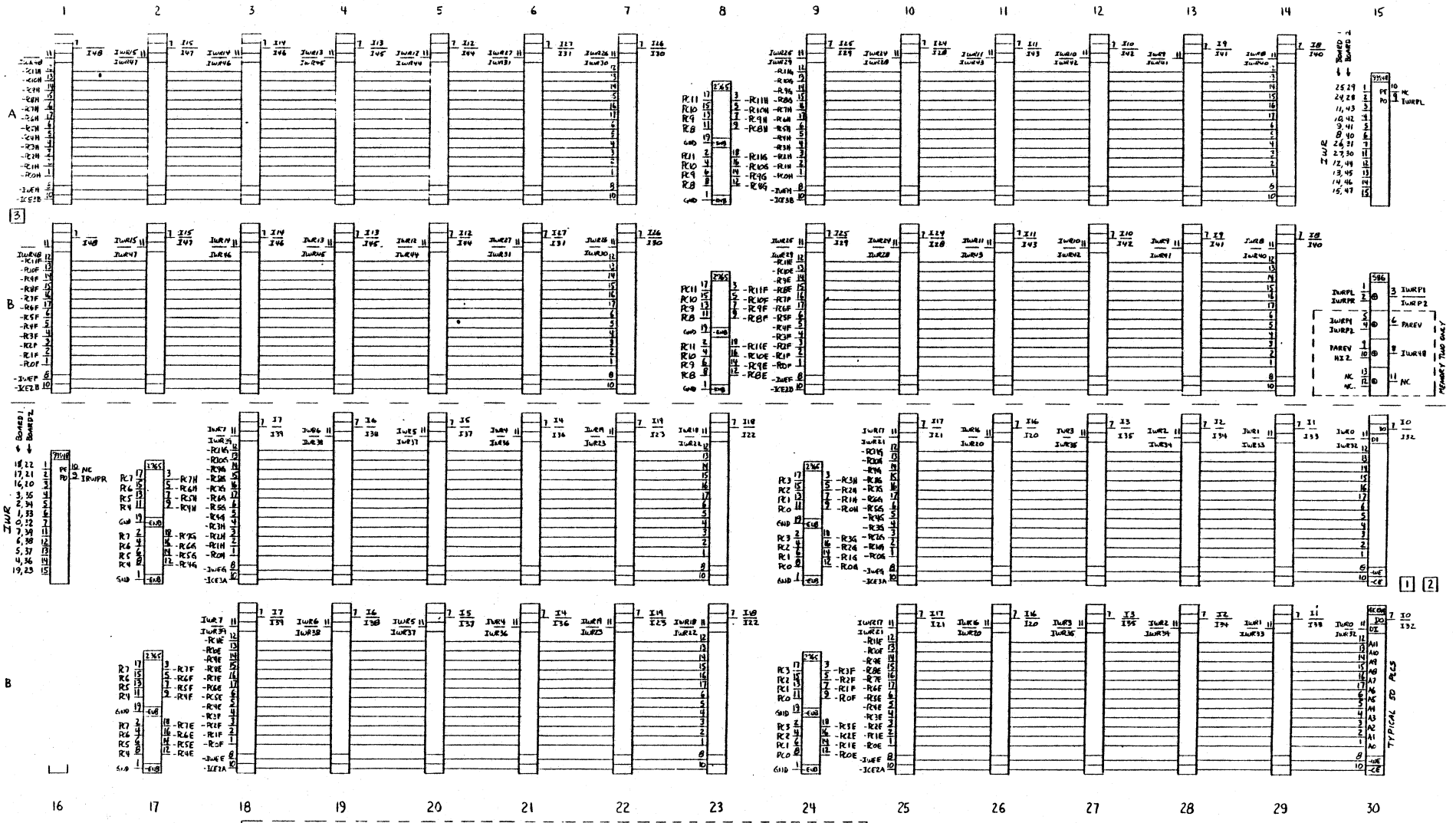


SYMBOLICS, INC.

INSTRUCTION MEMORY TWO, Sheet 2 of 5

009210

John V. Blankenbaker
Sept 25, 1980
Rev A



CONNECTOR J2

PIN	SIGNAL	PIN	SIGNAL
1	PC0T	13	PC12T
2	PC1T	14	PC13T
3	PC2T	15	
4	PC3T	16	-CLKT
5	PC4T	17	
6	PC5T	18	-ERRT
7	PC6T	19	
8	PC7T	20	
9	PC8T		
10	PC9T		
11	PC10T		
12	PC11T		

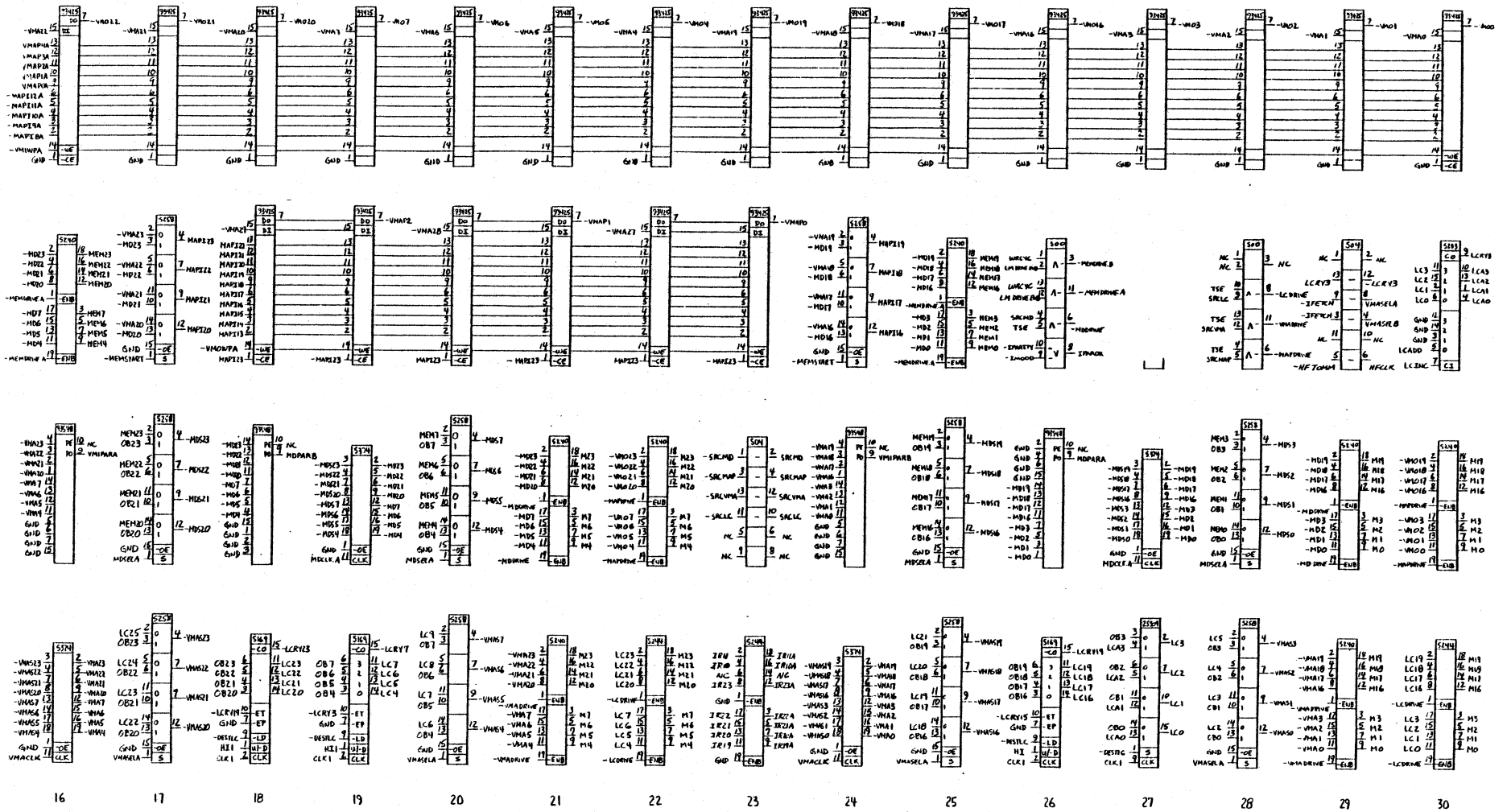
ON BOARD 1 AT A1 & B1 INSTALL 74LS240 DRIVERS AS SHOWN HERE

NOTES:
1. BOARD 1 SIGNAL BOARD 2 SIGNAL
2. 4K RAM WITH 2141 PINOUT

SYMBOLICS, INC.

INSTRUCTION MEMORY ONE & TWO, Sheet 3 of 3

009210 John V. Blankenbaker
Sept 25, 1980



NOTE 1. 74S02 PINOUTS SHOWN AS BOARD IS BUILT. TO USE 74S02 ADAPTOR _____ IS REQUIRED SEE 2A, 6A, 27A, 16B & 30B

ASSY 009222
FC BD 009221

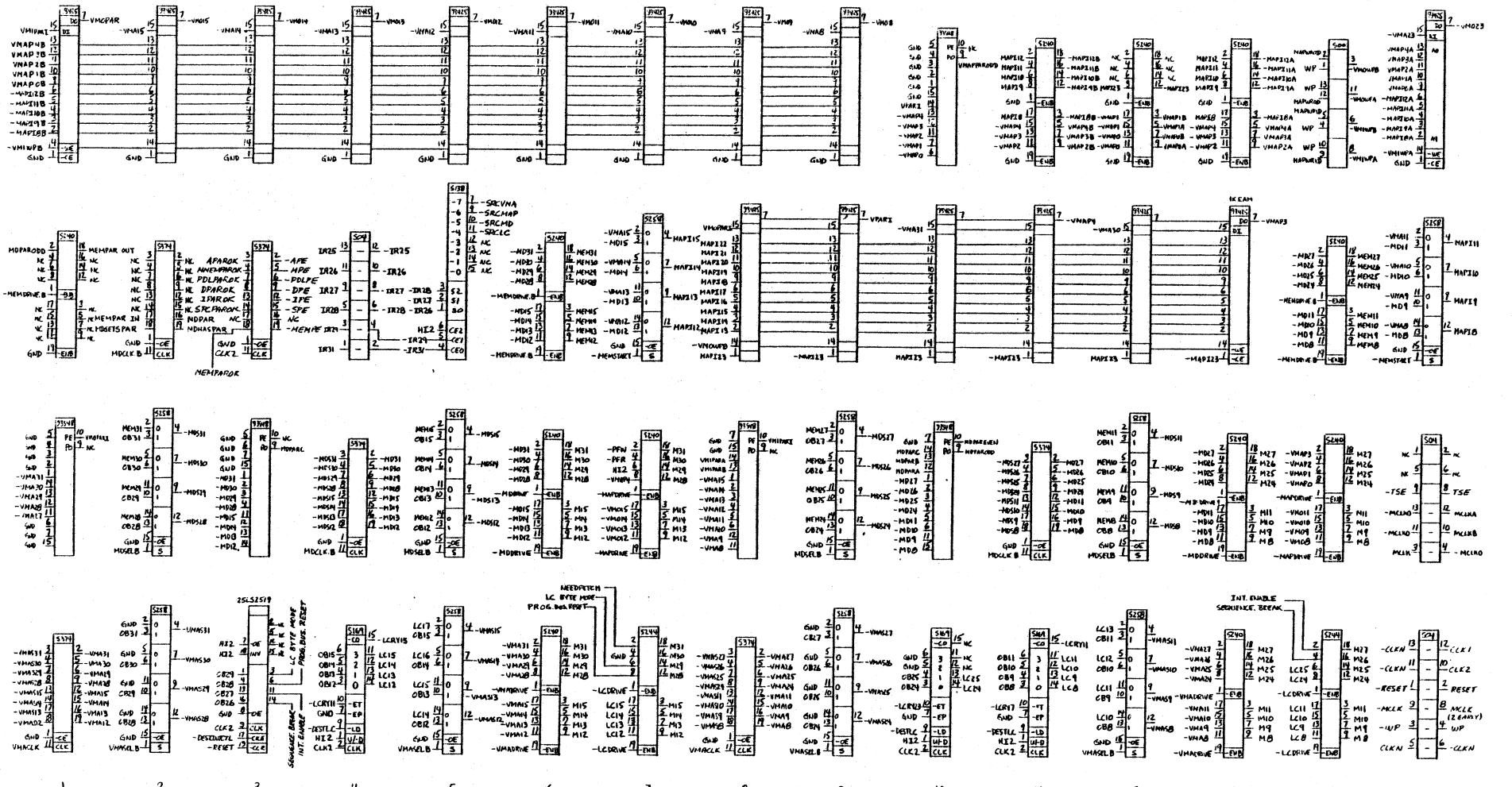
ECC XXXYXXXXX

SYMBOLICS, INC

MAIN MEMORY INTERFACE, Sheet 1 of 3

Schematic 009220

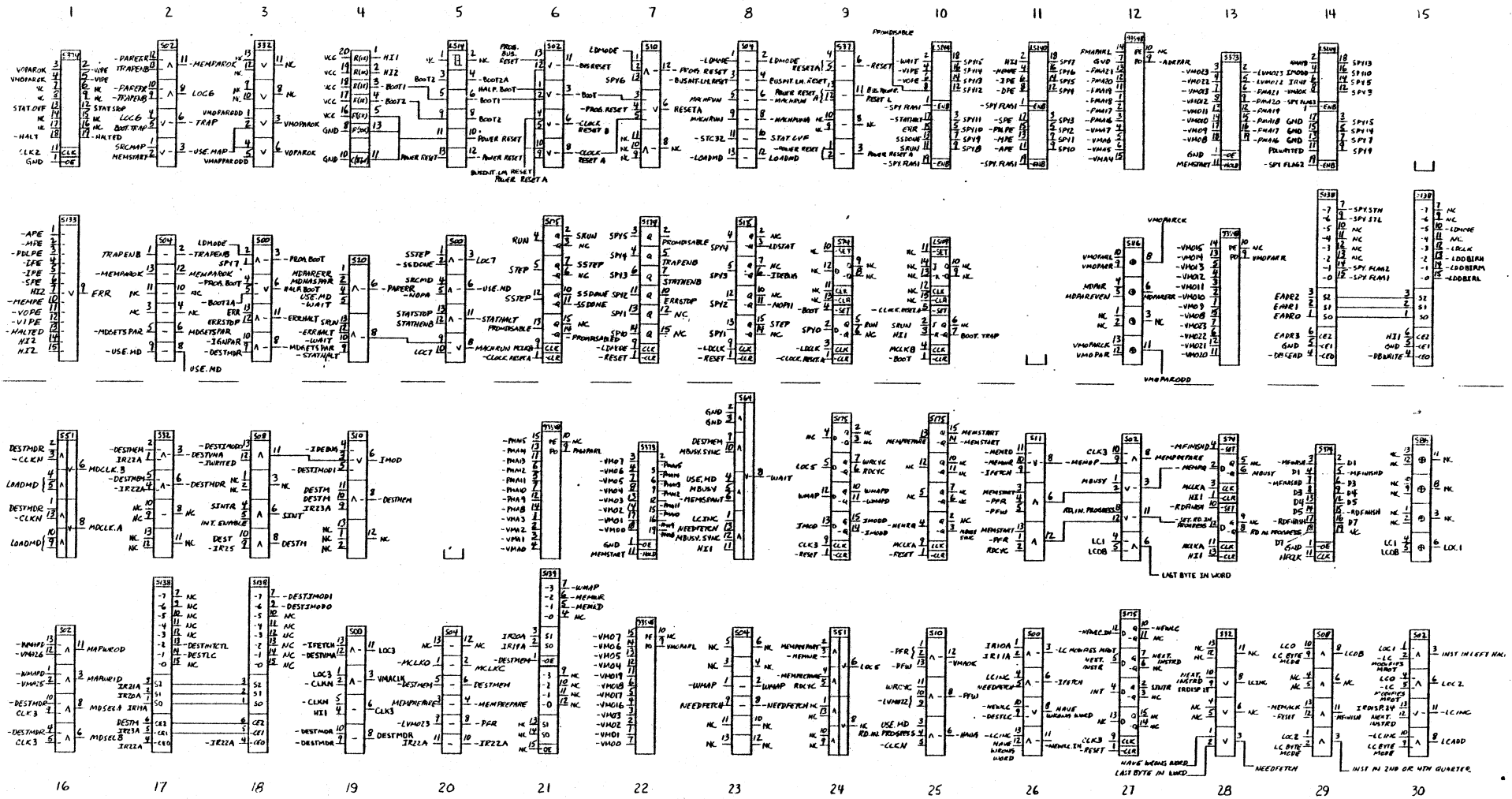
John V. Blankerbaker
November 13, 1980
Rev A



SYMBOLICS, INC

MAIN MEMORY INTERFACE, Sheet 2 o

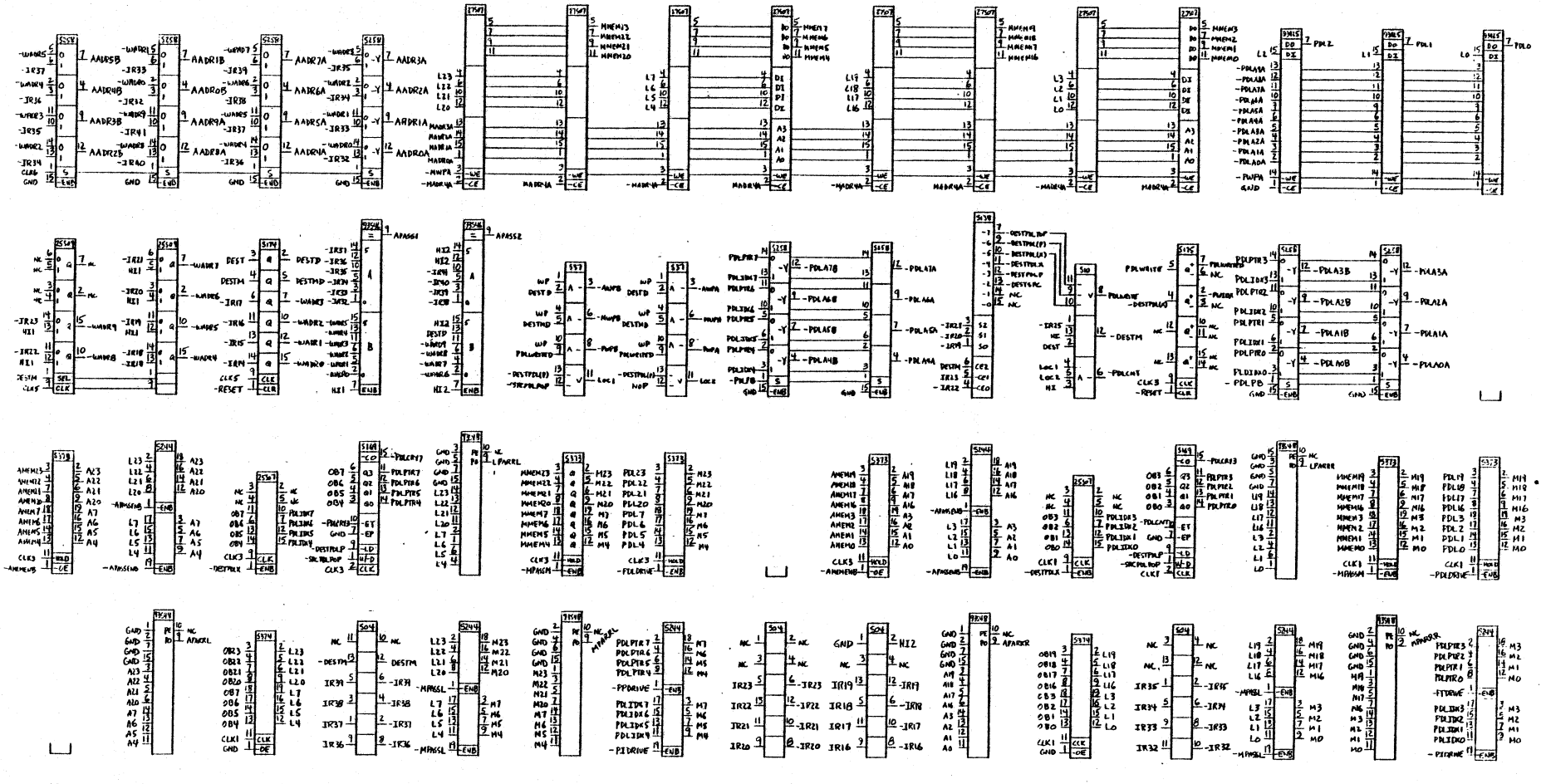
Schematic 009220 John V. Blankenbake
November 13, 1980
Rev A



SYMBOLICS, INC

MAIN MEMORY INTERFACE, Sheet 3 of 3

Schematic 009220 | John V Ekanzenbaker
November 13, 1980
Rev A



LOC 1 2 3 4 5 6

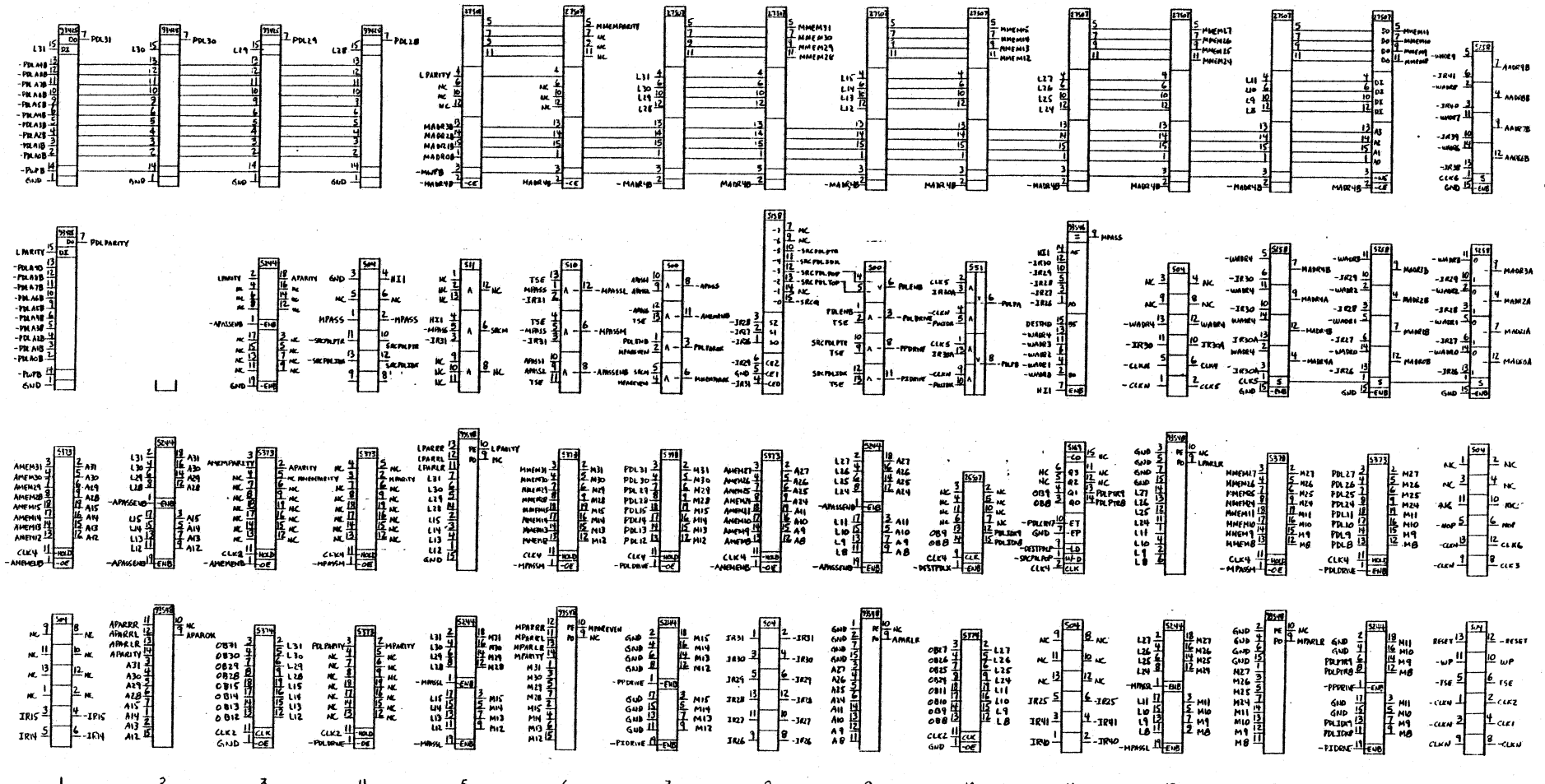
ASSY 009172
PC BOARD 009171

SYMBOLICS

REGISTER MEMORY Sheet 1 of

Schematic 009170

John V Blankenbaker
October 17, 1980

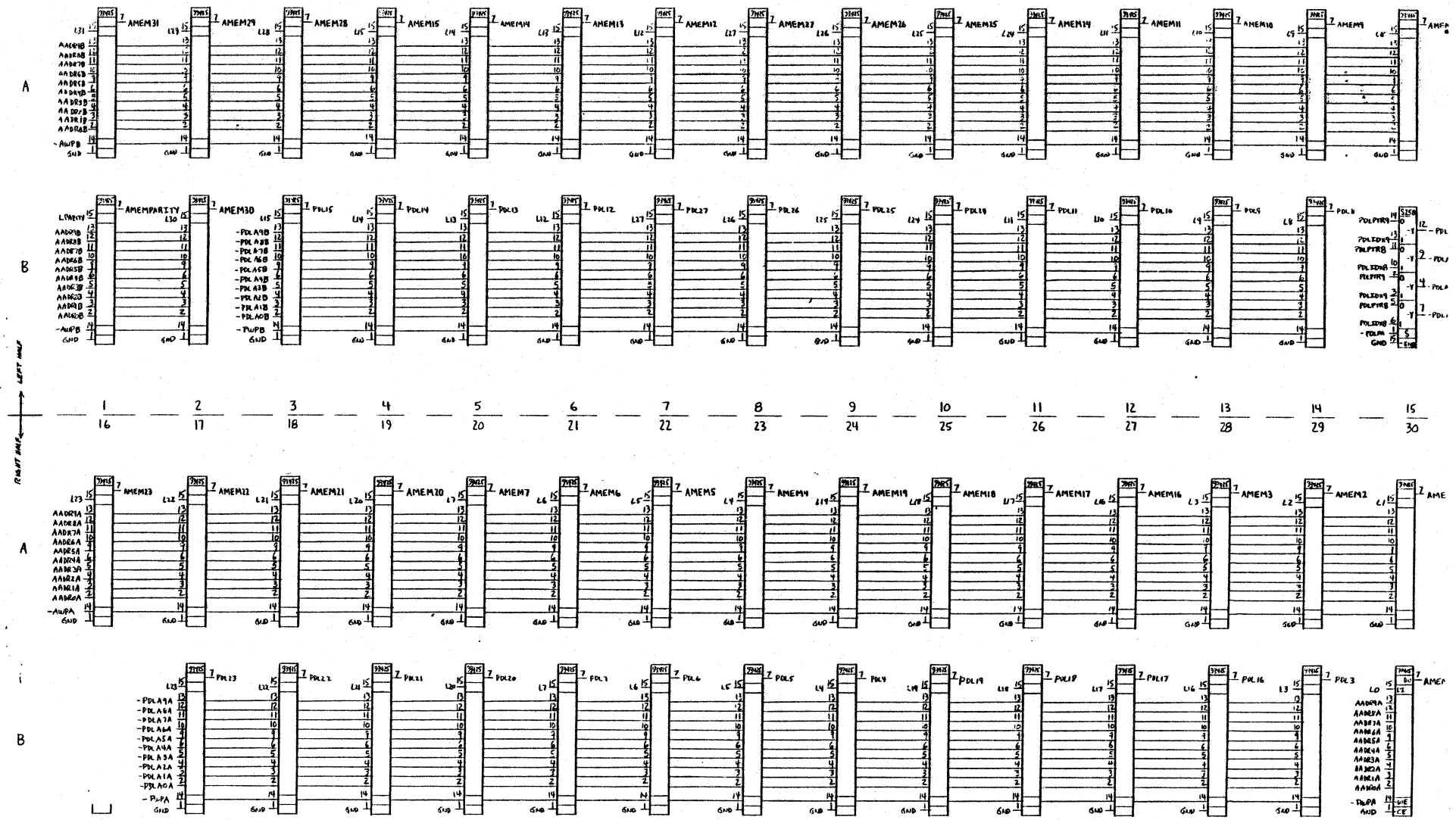


SYMBOLICS, INC

REGISTER MEMORY Sheet 2 of 3

Schematic 009170

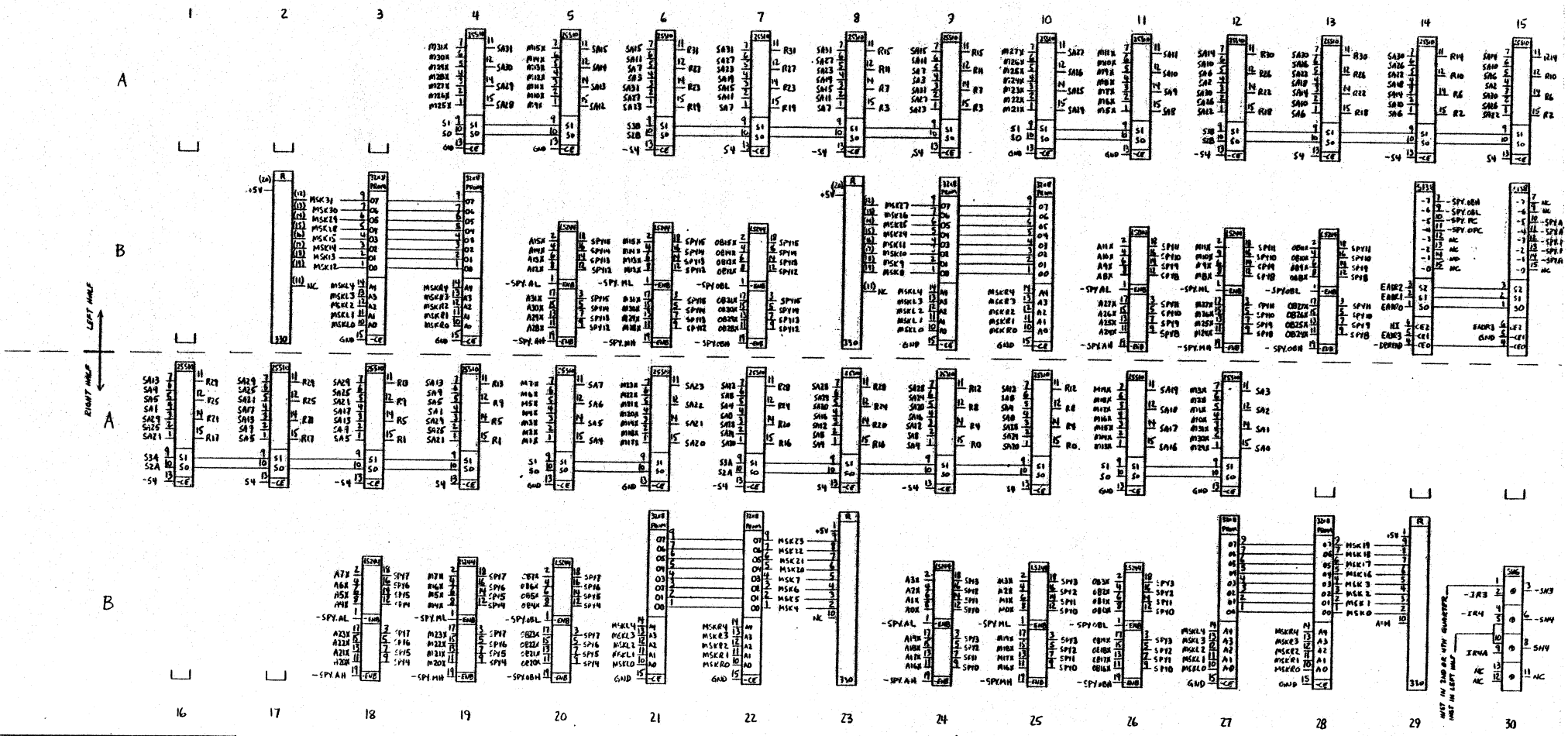
John V Blankenbaker
October 17, 1980
A



SYMBOLICS, INC.

REGISTER MEMORY Sheet 3 of 3

Schematic 009170 John V. Blankenbaker
October 17, 1980



SYMBOLICS, INC

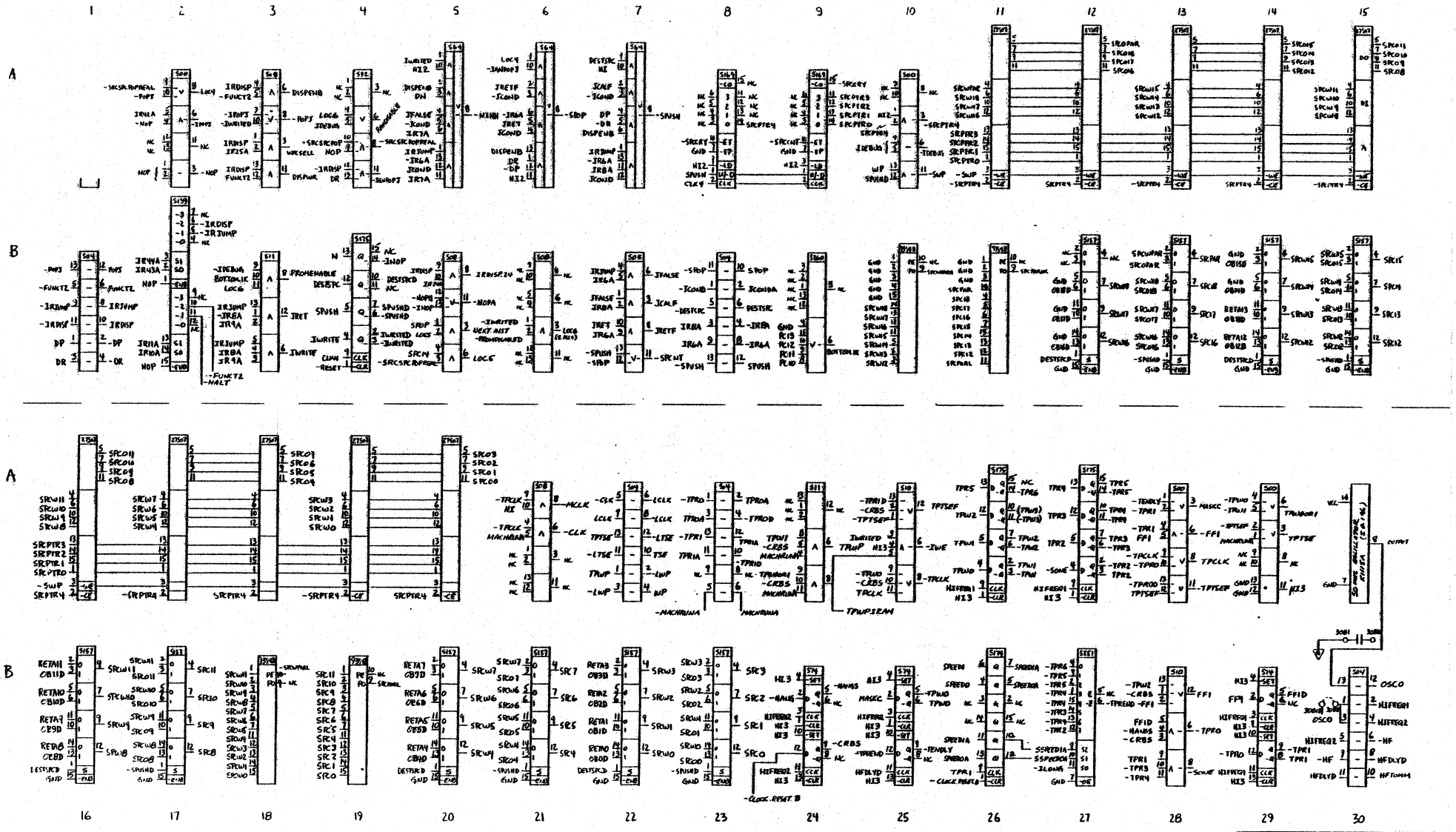
ARITHMETIC UNIT Sheet 3 of 3

Schematic 009180

John V. Blankenbaker

October 6, 1980

Rev A



SYMBOLICS, INC.

PROGRAM COUNTER

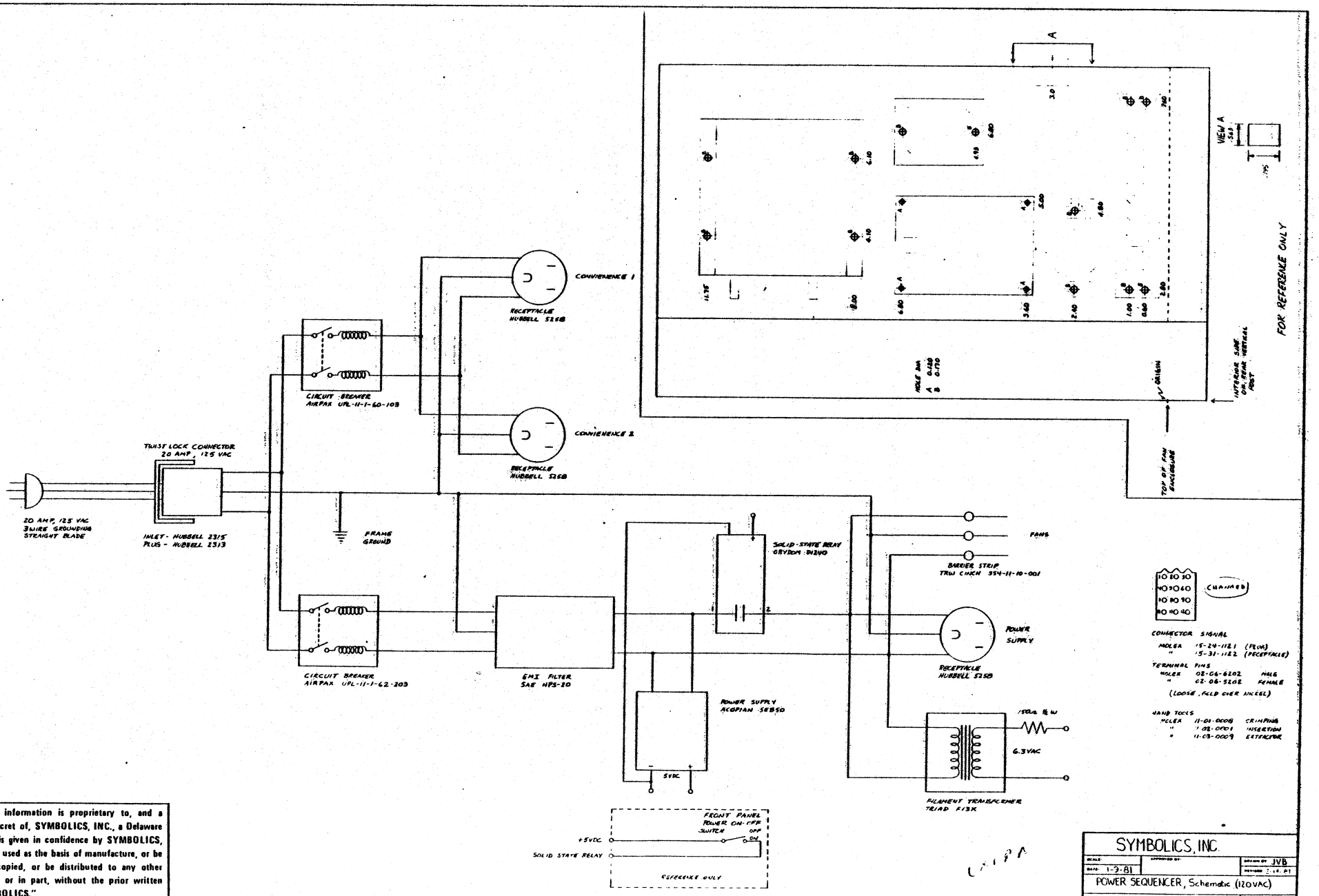
Sheet 3 of 3

Schematic 009190

John V Blankenbaker

November 11, 1980

Rev B



SYMBOLICS, INC.

SCALE:	APPROVED BY:	DRAWN BY: JVB
DATE: 1-9-81		REVISED: 5-18-81

POWER SEQUENCER, Schematic (120VAC)

DRAWING NUMBER 101091

The clock can be stopped at the end of either phase, for several reasons. Usually the clock stops at the end of the read phase, referred to as "wait". This leaves the clock in the inactive high state, and leaves the latches on the memories open. The clock can wait because the machine was commanded to halt by the diagnostic interface, because a single-step commanded by the diagnostic interface has completed, because of an error such as a parity error, because of the statistics counter overflowing, or because of a memory-wait condition. This latter condition happens if a main memory cycle is initiated while a previous cycle is still in progress, or if the program calls for the result of a main memory read before the bus controller has granted the bus access needed to perform that read cycle. During a clock wait, the processor clock stops, but the clock to the rest of the system (the bus interface and XBUS devices), continues to run, allowing them to operate. When the processor finishes waiting the processor clock starts up in synchrony with the external clock.

The clock can also stop at the end of the write phase, referred to as "hang". This is used only during memory reads. If the processor calls for the result of a read which is in progress but has not yet completed, it hangs until the data has arrived from memory and sufficient time has passed for the data to flow through the data paths and appear on the output bus. This is also sufficient time for the parity of the data to be checked. In the case of a hang, both clocks stop, which allows them to restart synchronously without any extra delay. In this way, the speed of the processor is adjusted to exactly match the speed of the memory.

ε4The Bus Interfaceε0

The Bus Interface connects the CADR machine to two busses, the Unibus and the Xbus. The Unibus is a regular pdp11 bus, used to attach peripheral devices, especially commercial devices designed for the PDP11 line. The Xbus is a 32-bit bus used to attach memory and high-performance peripheral devices, such as disk. The bus interface also includes the diagnostic interface, which allows a unibus operator, such as a pdp10, a pdp11, or another lisp machine, to control the operation of the machine, hardware to pass interrupts from the Unibus and the Xbus to the processor, the logic which arbitrates the Xbus, and the logic which arbitrates the Unibus in the absence of a pdp11 on that bus.

The Bus Interface allows the CADR machine to access memory on the Xbus and devices on the Unibus, allows independent devices on the Xbus to access the Xbus (only), and allows Unibus devices to access Xbus memory (through a map since the Unibus address space is not big enough.) Buffering is provided when the Unibus accesses the Xbus, to convert a 32-bit word into a pair of 16-bit words.

<<More to come>>

Cover how to program the various frobs from the Unibus, initialization, map structure.

.page
.sect
ε4The Xbusε0

The Xbus is the standard 32 bit wide data bus for the CADR processor. Main memory and high speed peripherals such as the disk control and TV display are interfaced to the Xbus. Control of the Xbus is similar to the Unibus, in that transfers are positively timed and (as far as the devices are concerned) asynchronous. The bus is terminated at both ends with resistive pullups of 390 ohms to ground and 180 ohms to +5 volts, for an effective 123 ohm termination to +3.42 volts. At ground, each termination draws 28 ma. for a total load of 56 ma. The bus is open collector, and may be driven with any device capable of handling the 56 ma. load. The recommended driver is the AMD 26S10, which also provides bus receivers.

A typical read cycle begins with placing the address for the transfer on the -XADDR lines and the parity of the address on the -XBUS.ADDRPAR line. The -XBUS.RQ line is then lowered, initiating the request. The responding device places the requested data on the 32 -XBUS lines and the parity of the data on the -XBUS.PAR line. Should it not be convenient for the device to produce parity (as in the case of I/O registers), the device may assert -XBUS.IGNPAR to notify the bus master that the transfer should not be checked for correct parity. The responding device then asserts -XBUS.ACK, which remains asserted until the -XBUS.RQ signal is removed by the master.

Write requests proceed identically, except that the master asserts -XBUS.WR and the data to be written on the -XBUS lines along with the address lines. All bus masters are required to produce good parity data on writes.

Deskewing delays are the responsibility of the bus master. In particular, it is the responsibility of the bus master to assert good address, write, and data lines 80 ns. prior to asserting -XBUS.RQ, and these lines must be held until the -XBUS.ACK signal drops in response to the master dropping -XBUS.RQ. Responding devices are allowed to assert -XBUS.ACK at the same time they drive read data onto the -XBUS lines. Thus, masters should delay 50 ns. after receiving -XBUS.ACK before dropping -XBUS.RQ and strobing the data. Responding devices are required to drop -XBUS.ACK immediately after -XBUS.RQ is no longer asserted.

Normal bus master arbitration between the CADR processor and the Unibus requests is handled by the bus interface. Devices on the Xbus which must become bus master, such as the disk control, do so by asserting the -XBUS.EXTRQ signal. When the bus becomes free, the bus interface responds by asserting -XBUS.EXTGRANT. This signal is daisy chained between bus master devices on the Xbus, coming in on the -XBUS.EXTGRANT.IN pin and leaving on the -XBUS.EXTGRANT.OUT pin. Within each device, the decision is made whether or not to pass the grant onto the next device. Unlike the Unibus structure, the decision on whether to pass grant and the act of becoming bus master happen synchronously with a master clock signal distributed on the -XBUS.SYNC line.

When a device initiates a request, it immediately asserts -XBUS.EXTRQ. At the falling edge of -XBUS.SYNC it clocks the request signal into a D flip flop which we will call RFQ.SYNC. When -XBUS.EXTGRANT.IN goes low, the device asserts -XBUS.EXTGRANT.OUT unless it has either the RFQ.SYNC flip flop set, or is already the bus master. At the next falling edge of -XBUS.SYNC the device which has both -XBUS.EXTGRANT.IN and RFQ.SYNC set becomes bus master. The device should immediately assert -XBUS.BUSY and may immediately begin asserting address lines for a transfer. -XBUS.BUSY may be dropped asynchronously, after the slave device drops -XBUS.ACK in response to the master's request.

The -XBUS.EXTGRANT.IN signal must be terminated with a resistive pullup of 180 ohms to +5 volts within each device which does not simply connect it to -XBUS.EXTGRANT.OUT.

Signal review:

data lines:

-XBUS0 through -XBUS31	32 data lines, low when data is a one
-XBUS.PAR	parity of the 32 data lines. Required for writes
-XBUS.IGNPAR	ignore parity signal, may be asserted by any device for a read

address lines:

-XADDR0 through -XADDR21	22 address lines, low for address bit a one
-XADDR.PAR	<<this needs to be decided..."is this required?>>

cycle control lines:

-XBUS.RQ	Asserted by the master to request a read or write Minimum of 80 ns following stable -XADDR, -XBUS.WRITE and -XBUS data
-XBUS.ACK	Asserted by the slave in response to -XBUS.RQ No delay necessary following assertion of good read data
-XBUS.WR	Asserted by the master during a write cycle.

mastership control lines:

-XBUS.BUSY	Asserted when a device other than the bus interface is bus master. Only the bus interface examines this line. Asserted on a -XBUS.SYNC clock edge, dropped asynchronously after -XBUS.ACK drops
-XBUS.EXTRQ	Asserted when a device other than the bus interface wishes to become bus master. Asserted asynchronously, may be removed asynchronously after the device becomes master, but before dropping -XBUS.BUSY
-XBUS.EXTGRANT.IN	The daisy-chained mastership grant signal. Must be pulled up with 180 ohms to VCC in the device.
-XBUS.EXTGRANT.OUT	Asserted initially by the bus interface, synchronously with the -XBUS.SYNC edge. The signal may be subject to synchronizer lossage, since it is a clocked version of -XBUS.EXTRQ which is not synchronous with -XBUS.SYNC

Miscellaneous:

-XBUS.INIT	When low, resets all devices. This is low during power on and off, and when the machine is reset.
-XBUS.SYNC	Synchronization clock for mastership passing and other desired purposes. Devices become bus master synchronous with the edge of this signal. The request will normally follow the edge by 80 ns.
-XBUS.INTR	Driving this low requests an interrupt. All devices are required to initialize to a non-interrupt enable condition, and are required to have interrupt

enable and disable bits which can selectively enable interrupts from that device. The "requesting interrupt" state must be readable in one of the device control register bits.

XBUS.POWER.OK

This line is HIGH when power is stable. It remains low for <<xx>> seconds after power comes on, and goes low <<xx>> seconds before power is turned off.

LISPM Bus Interface CADR1:BUSINT UML 25-MAR-81 0822
******* DIP MAP *******

26S10 XD x F30	26S10 XD x E30	74LS244 XBD x D30	93S48 BUSPAR x C30	8304 LMDATA x B30	SIP180/3 LMDATA x A30
26S10 XD x F29	26S10 XD x E29	74LS244 XBD x D29	93S48 BUSPAR x C29	8304 LMDATA x B29	SIP180/3 LMDATA x A29
26S10 XD x F28	26S10 XD x E28	74LS244 XBD x D28	93S48 BUSPAR x C28	8304 LMDATA x B28	SIP180/3 LMDATA x A28
26S10 XD x F27	26S10 XD x E27	74LS244 XBD x D27	29701 WBUF x C27	8304 LMDATA x B27	74LS240 LMADR x A27
26S10 XA x F26	26S10 XA x E26	29701 RBUF x D26	29701 WBUF x C26	74LS244 BUSSEL x B26	74LS240 LMADR x A26
26S10 XA x F25	26S10 XA x E25	29701 RBUF x D25	29701 WBUF x C25	74LS244 BUSSEL x B25	74LS240 LMADR x A25
26S10 XA x F24	93S48 XAPAR x E24	29701 RBUF x D24	29701 WBUF x C24	74LS244 BUSSEL x B24	74LS240 LMADR x A24
26S10 XA x F23	93S48 XAPAR x E23	29701 RBUF x D23	74LS244 RBUF x C23	74LS244 BUSSEL x B23	74LS240 LMADR x A23
26S10 XA x F22	74LS244 UBXA x E22	74LS244 RBUF x D22	74LS244 RBUF x C22	8304 DBGOUT x B22	SIP180/3 DBGIN x A22
26S10 XA x F21	74LS244 UBXA x E21	74LS244 RBUF x D21	74LS244 BUSSEL x C21	8304 DBGOUT x B21	8304 DIAG x A21
26S10 XD x F20	74LS244 UBXA x E20	74S86 REQERR xxxx D20	74LS244 BUSSEL x C20	74S00 DATCTL xxxx B20	8304 DIAG x A20

LISPM Bus Interface CADR1;BUSINT UML 25-MAR-81 0823
******* DIP MAP *******

DM8838 UBD x F19	74LS244 UBD x E19	74LS244 BUSSEL x D19	74LS244 BUSSEL x C19	74S04 DATCTL xxxxxx B19	74LS374 DBGIN x A19
DM8838 UBD x F18	74LS244 UBD x E18	74LS240 UBINTC x D18	74S51 DATCTL xx C18	74S32 DATCTL xxxx B18	74LS374 DBGIN x A18
DM8838 UBD x F17	74LS244 UBMAP x E17	74LS374 UBINTC x D17	74S51 DATCTL xx C17	74S02 DATCTL xxxx B17	74S241 DBGOUT x A17
DM8838 UBD x F16	74LS244 UBMAP x E16	25LS2519 UBINTC x D16	74LS244 REQERR x C16	74S08 REQTIM xxxx B16	25LS2519 DBGIN x A16
DM8838 UPRIOR x F15	29701 UBMAP x E15	74LS74 UBINTC xx D15	74S64 DATCTL x C15	8304 REQERR x B15	74S139 DBGIN xo A15
74S38 UPRIOR xxoo F14	29701 UBMAP x E14	74LS74 UBINTC xx D14	74S51 DATCTL xx C14	74S74 RQSYNC xx B14	74S10 DBGIN xxx A14
74S38 UPRIOR xxxx F13	29701 UBMAP x E13	74LS74 UBINTC xx D13	74S10 RQSYNC xxx C13	74S04 XA xxxxxx B13	74S04 DBGIN xxxxxx A13
DM8838 UBA x F12	29701 UBMAP x E12	74S133 REQU x D12	74S260 RQSYNC xx C12	74S02 DBGOUT xxxx B12	74S08 DBGOUT xxxx A12
DM8838 UBA x F11	74S258 UBMAP x E11	74S04 UBA xxxxxx D11	74S64 REQLM x C11	74S51 REQLM xx B11	74S00 DIAG xxxx A11
DM8838 UBA x F10	74S258 UBMAP x E10	74S174 UPRIOR x D10	74S64 REQLM x C10	74S74 REQUB xx B10	MTD100 DBGOUT xoo A10
DM8838 UBA x F09	74S08 REQLM xxoo E09	74S472 UPRIOR x D09	TD100 REQLM x C09	TD250 REQUB x B09	74S32 DBGIN xxxx A09

LISPM Bus Interface CADR1:BUSINT UML 25-MAR-81 0823
******* DIP MAP *******

DM8838 UBA x	74S133 UBCYC x	25LS2519 UPRIOR x	74S04 REQLM xxxxxx	74S74 UBCYC xx	74S175 RQSYNC x
F08	E08	D08	C08	B08	A08
DM8838 UBMAST x	74S139 UBCYC xx	25LS2519 UPRIOR x	MTD100 REQU xxx	74S04 CLM xxxxxx	74S260 RQSYNC xx
F07	E07	D07	C07	B07	A07
DM8838 UPRIOR x	74S260 UBCYC xx	74S10 UBMAST xxx	74S51 REQUB xx	TD100 RQSYNC x	74S175 RQSYNC x
F06	E06	D06	C06	B06	A06
74S133 UBCYC x	74S04 UBINTC xxxxxx	74S00 UBINTC xxxx	74S20 REQU xo	74S11 RQSYNC xxx	74S02 RQSYNC xxxx
F05	E05	D05	C05	B05	A05
TD250 UBCYC x	74S32 UBINTC xxxx	74S74 UBMAST xx	74S11 DBGIN xxx	74S241 DBGOUT x	74S00 REQU xxxx
F04	E04	D04	C04	B04	A04
74S00 REQU xxxx	74S138 UBCYC x	74LS74 REQU xx	74LS27 UPRIOR xxx	74S04 REQTIM xxxxxx	74LS112 REQERR ox
F03	E03	D03	C03	B03	A03
74S02 REQU xxxx	74S08 UBINTC xxxx	74LS74 UBMAST xx	74LS74 UBMAST xx	74 276 REQERR x	74S288 REQTIM x
F02	E02	D02	C02	B02	A02
TD100 REQU x	74S00 UBMAST xxxx	MTD100 UBMAST xxx	74LS163 UPRIOR x	74LS273 REQTIM x	74LS124 REQTIM xo
F01	E01	D01	C01	B01	A01

LISPM Bus Interface CADR1;BUSINT UML 25-MAR-81 0823
******* EDGE CONNECTIONS Flags: (# Output, @ Terminator, ---- Dedicated ground, ++++ Dedicated power) *******

-A-	-B-	-C-	-D-
A1 -XBUS35 A2 +5.0V++++++++++++++++++	A1 -XBUS5 A2 +5.0V++++++++++++++++++ #	A1 UB NPG IN A2 +5.0V++++++++++++++++++ #	A1 A2 +5.0V++++++++++++++++++
B1 -XBUS34 B2 -5.0V	B1 -XBUS4 B2 -5.0V	B1 UB NPG OUT B2 -5.0V #	B1 B2 -5.0V
C1 -XBUS33 C2 GND----------	C1 -XBUS3 C2 GND---------- #	C1 -XBUS RQ C2 GND---------- #	C1 C2 GND----------
D1 -XBUS32 D2 -XBUS31	D1 -XBUS2 D2 -XBUS1 #	D1 -XBUS ACK D2 -UBD15 H #	D1 D2 -UB BR7 H #
E1 -XBUS30 E2 -XBUS29 #	E1 -XBUS0 E2 -XBUS PAR #	E1 -XBUS WR E2 -UBD14 H #	E1 E2 -UB BR6 H #
F1 GND F2 -XBUS28	F1 GND F2 -XADDR PAR #	F1 GND F2 -UBD13 H #	F1 GND F2 -UB BR5 H #
H1 -XBUS27 H2 -XBUS26 #	H1 -XADDR21 H2 -XADDR20 #	H1 -UBD11 H H2 -UBD12 H #	H1 H2 -UB BR4 H #
J1 -XBUS25 J2 -XBUS24 #	J1 -XADDR19 J2 -XADDR18 #	J1 -XBUS IGNPAR J2 -UBD10 H #	J1 J2
K1 -XBUS23 K2 -XBUS22 #	K1 -XADDR17 K2 -XADDR16 #	K1 -XBUS INIT K2 -UBD9 H #	K1 K2 UB BG7 IN #
L1 -XBUS21 L2 -XBUS20 #	L1 -XADDR15 L2 -XADDR14 #	L1 -XBUS EXTRQ L2 -UBD8 H #	L1 -UB INIT H # L2 UB BG7 IN #
M1 -XBUS19 M2 -XBUS18 #	M1 -XADDR13 M2 -XADDR12 #	M1 -XBUS BUSY M2 -UBD7 H #	M1 M2 UB BG6 IN #
N1 GND N2 -XBUS17	N1 GND N2 -XADDR11 #	N1 GND N2 -UBD4 H #	N1 GND N2 UB BG6 IN #
P1 -XBUS16 P2 -XBUS15 #	P1 -XADDR10 P2 -XADDR9 #	P1 -XBUS SYNC P2 -UBD5 H #	P1 +12.0V P2 UB BG5 IN #
R1 -XBUS14 R2 -XBUS13 #	R1 -XADDR8 R2 -XADDR7 #	R1 -LM BOOT H R2 -UBD1 H #	R1 +12.0V R2 UB BG5 IN #
S1 -XBUS12 S2 -XBUS11 #	S1 -XADDR6 S2 -XADDR5 #	S1 -XBUS INTR S2 -UBD0 H #	S1 +12.0V S2 UB BG4 IN #
T1 GND----------	T1 GND----------	T1 GND----------	T1 GND----------
T2 -XBUS10 #	T2 -XADDR4 #	T2 -UBD3 H #	T2 UB BG4 IN #
U1 -XBUS9 U2 -XBUS8 #	U1 -XADDR3 U2 -XADDR2 #	U1 -XBUS POWER RESET H U2 -UBD2 H #	U1 U2
V1 -XBUS7 V2 -XBUS6 #	V1 -XADDR1 V2 -XADDR0 #	V1 -XBUS EXTGRANT OUT V2 -UBD6 H #	V1 V2

LISPM Bus Interface CADR1;BUSINT UML 25-MAR-81 0823
******* EDGE CONNECTIONS Flags: (# Output, @ Terminator, ---- Dedicated ground, ++++ Dedicated power) *******

-E-		-F-		-J01-		-J02-	
A1		A1		01 CLKO	#	01	
A2 +5.0V++++++++++++++++++		A2 +5.0V++++++++++++++++++		02 -MEMRQ H	@	02	
B1		B1		03 -LM ACK H	#	03	
B2 -5.0V		B2 -5.0V		04 LMX GRANT	#	04	
C1 -UB ADR12 H	#	C1		05 LMUB GRANT	#	05	
C2 GND----------		C2 GND----------		06 XBUS REQUEST	#	06	
D1 -UB ADR17 H	#	D1 -UB BBSY H	#	07 LMUB MASTER	#	07	
D2 -UB ADR15 H	#	D2		08 C1 OUT	#	08	
E1 -UB MSYN H	#	E1		09 XWR	#	09	
E2 -UB ADR16 H	#	E2		10 -FREE H	#	10	
F1 -UB ADR2 H	#	F1 GND		11 NXM TIMEOUT	#	11	
F2 -UB C1 H	#	F2		12 -ANY PAR ERROR H	#	12	
H1 -UB ADR1 H	#	H1		13 ANY GRANT DLYD	#	13	
H2 -UB ADR0 H	#	H2		14 MSYN IN	#	14	
J1 -UB SSYN H	#	J1 -UB NPR H	#	15 MSYN OUT	#	15	
J2 -UB C0 H		J2		16 SSYN IN	#	16	
K1 -UB ADR14 H	#	K1		17 SSYN OUT	#	17	
K2 -UB ADR13 H	#	K2		18 UB REG CYC TO	#	18	
L1 -UB ADR11 H	#	L1		19 UBXRQ	#	19	
L2		L2		20 UBX GRANT	#	20	
M1		M1 -UB INTR H	#	21 -DEBUG IN REQ H	@	21	
M2		M2		22 DEBUG ACK	#	22	
N1 GND		N1 GND		23 DBUB MASTER	#	23	
N2 -UB ADR8 H	#	N2		24 NC	#	24	
P1 -UB ADR10 H	#	P1		25 NC	#	25	
P2 -UB ADR7 H	#	P2		26 ----------		26 ----------	
R1 -UB ADR9 H	#	R1		27 ----------		27 ----------	
R2		R2		28 ----------		28 ----------	
S1		S1		29 ----------		29 ----------	
S2		S2		30 ----------		30 ----------	
T1 GND----------		T1 GND----------		31 ----------		31 ----------	
T2		T2 -UB SACK H	#	32 ----------		32 ----------	
U1 -UB ADR6 H	#	U1		33 ----------		33 ----------	
U2 -UB ADR4 H	#	U2		34 ----------		34 ----------	
V1 -UB ADR5 H	#	V1		35 ----------		35 ----------	
V2 -UB ADR3 H	#	V2 LOCAL ENABLE	@	36 ----------		36 ----------	
				37 ----------		37 ----------	
				38 ----------		38 ----------	
				39 ----------		39 ----------	
				40 ----------		40 ----------	
				41 ----------		41 ----------	
				42 ----------		42 ----------	
				43 ----------		43 ----------	
				44 ----------		44 ----------	
				45 ----------		45 ----------	
				46 ----------		46 ----------	
				47 ----------		47 ----------	
				48 ----------		48 ----------	
				49 ----------		49 ----------	
				50 ----------		50 ----------	

LISPM Bus Interface CADR1;BUSINT UML 25-MAR-81 0823
******* EDGE CONNECTIONS Flags: (# Output, @ Terminator, ---- Dedicated ground, ++++ Dedicated power) *******

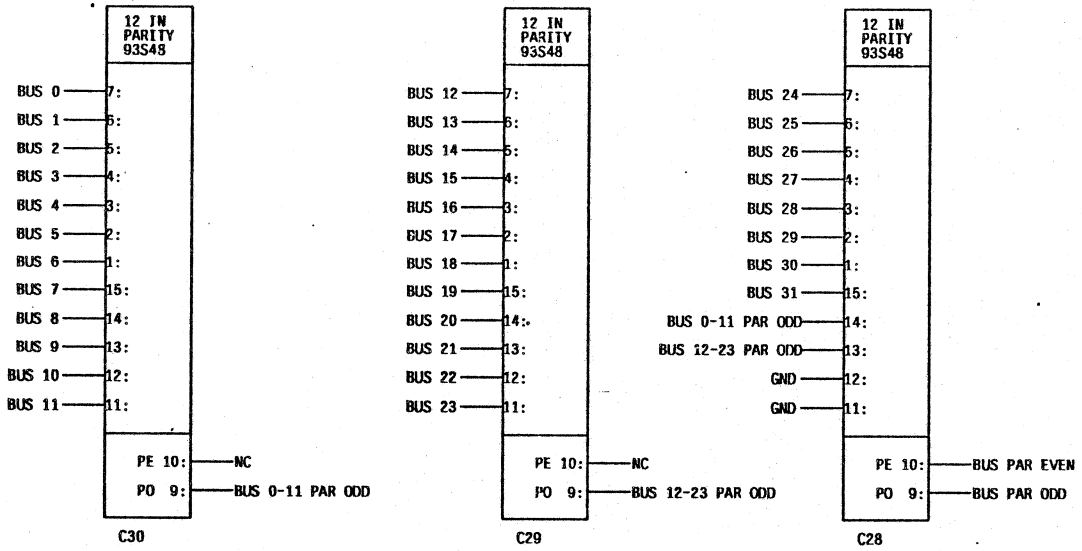
-J03-	-J04-	-J05-		-J06-	
01	01	01 DBD0	#	01 DBD0	#
02	02	02 DBD1	#	02 DBD1	#
03	03	03 DBD2	#	03 DBD2	#
04	04	04 DBD3	#	04 DBD3	#
05	05	05 DBD4	#	05 DBD4	#
06	06	06 DBD5	#	06 DBD5	#
07	07	07 DBD6	#	07 DBD6	#
08	08	08 DBD7	#	08 DBD7	#
09	09	09 DBD8	#	09 DBD8	#
10	10	10 DBD9	#	10 DBD9	#
11	11	11 DBD10	#	11 DBD10	#
12	12	12 DBD11	#	12 DBD11	#
13	13	13 DBD12	#	13 DBD12	#
14	14	14 DBD13	#	14 DBD13	#
15	15	15 DBD14	#	15 DBD14	#
16	16	16 DBD15	#	16 DBD15	#
17	17	17 DEBUG IN A0	@	17 DEBUG OUT A0	#
18	18	18 DEBUG IN A1	@	18 DEBUG OUT A1	#
19	19	19 DEBUG IN WR	@	19 DEBUG OUT WR	#
20	20	20 -DEBUG IN REQ H	@	20 -DEBUG OUT REQ H	#
21 ----------	21 ----------	21 DEBUG IN ACK	#	21 DEBUG OUT ACK	@
22 ----------	22 ----------	22 NC	#	22 NC	#
23 ----------	23 ----------	23 NC	#	23 NC	#
24 ----------	24 ----------	24 NC	#	24 NC	#
25 ----------	25 ----------	25 NC	#	25 NC	#
26 ----------	26 ----------	26 ----------		26 ----------	
27 ----------	27 ----------	27 ----------		27 ----------	
28 ----------	28 ----------	28 ----------		28 ----------	
29 ----------	29 ----------	29 ----------		29 ----------	
30 ----------	30 ----------	30 ----------		30 ----------	
31 ----------	31 ----------	31 ----------		31 ----------	
32 ----------	32 ----------	32 ----------		32 ----------	
33 ----------	33 ----------	33 ----------		33 ----------	
34 ----------	34 ----------	34 ----------		34 ----------	
35 ----------	35 ----------	35 ----------		35 ----------	
36 ----------	36 ----------	36 ----------		36 ----------	
37 ----------	37 ----------	37 ----------		37 ----------	
38 ----------	38 ----------	38 ----------		38 ----------	
39 ----------	39 ----------	39 ----------		39 ----------	
40 ----------	40 ----------	40 ----------		40 ----------	
		41 ----------		41 ----------	
		42 ----------		42 ----------	
		43 ----------		43 ----------	
		44 ----------		44 ----------	
		45 ----------		45 ----------	
		46 ----------		46 ----------	
		47 ----------		47 ----------	
		48 ----------		48 ----------	
		49 ----------		49 ----------	
		50 ----------		50 ----------	

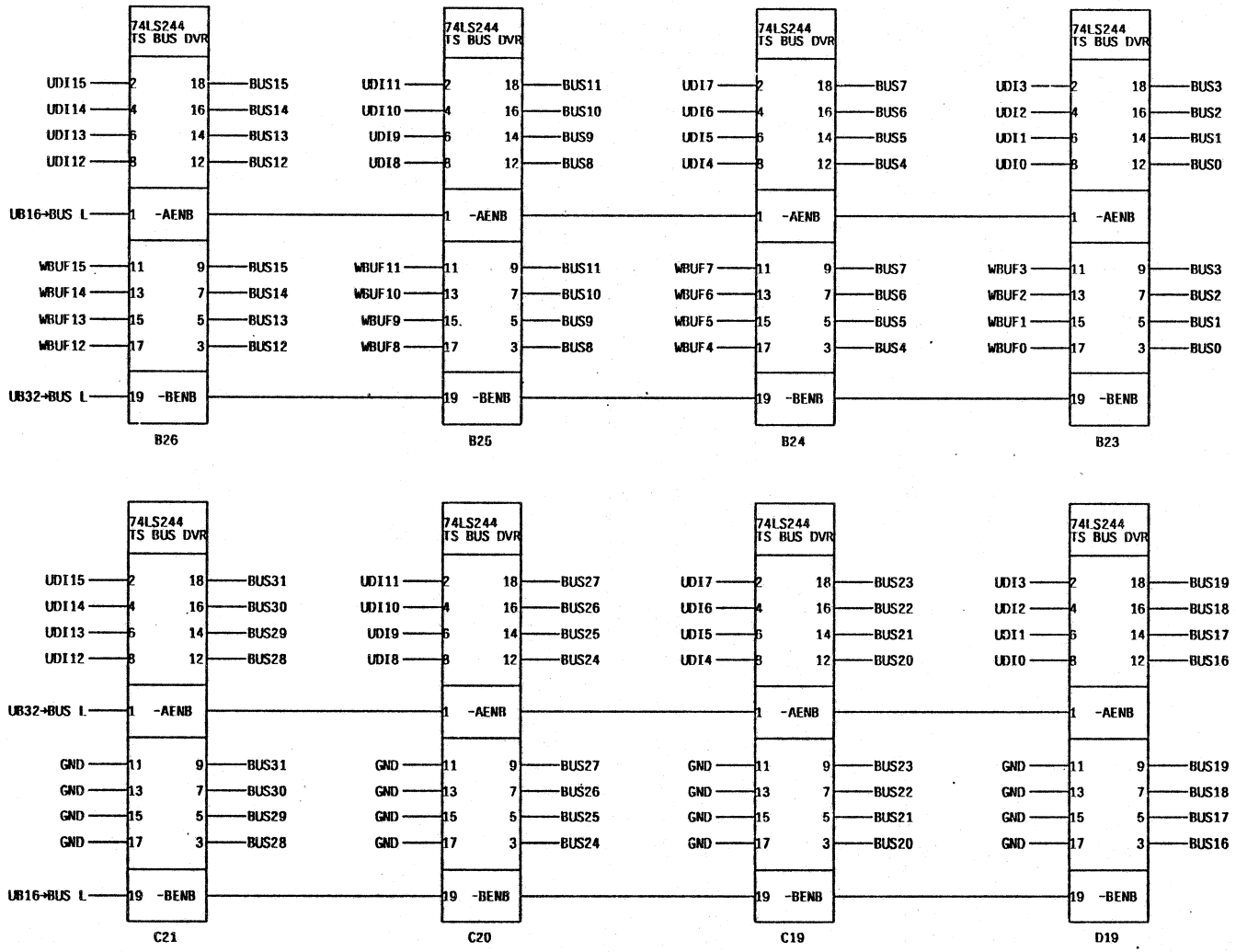
LISPM Bus Interface CADR1;BUSINT UML 25-MAR-81 0823
******* EDGE CONNECTIONS Flags: (# Output, @ Terminator, ---- Dedicated ground, ++++ Dedicated power) *******

-J07-		-J08-		-J09-		-J10-
01 SPY0	#	01 GND		01 -ADR15 H		01
02 SPY1	#	02 -MCLK7 H	@	02 -ADR14 H		02
03 SPY2	#	03 GND		03 -ADR13 H		03
04 SPY3	#	04 GND		04 -ADR12 H		04
05 SPY4	#	05 SPY ADR 1	#	05 -ADR11 H		05
06 SPY5	#	06 SPY ADR 2	#	06 -ADR10 H		06
07 SPY6	#	07 SPY ADR 3	#	07 -ADR9 H		07
08 SPY7	#	08 SPY ADR 4	#	08 -ADR8 H		08
09 SPY8	#	09 -SPY READ H	#	09 -ADR7 H		09
10 SPY9	#	10 -SPY WRITE H	#	10 -ADR6 H		10
11 SPY10	#	11 -BUSINT LM RESET H	#	11 -ADR5 H		11
12 SPY11	#	12 -LM BOOT H		12 -ADR4 H		12
13 SPY12	#	13 -LM UNIBUS RESET H		13 -ADR3 H		13
14 SPY13	#	14 LM MEMDRIVE ENB	#	14 -ADR2 H		14
15 SPY14	#	15 -LM POWER RESET H		15 -ADR1 H		15
16 SPY15	#	16 NC	#	16 -ADR0 H		16
17 -LM GRANT H	#	17 NC	#	17 -MEMRQ H	@	17
18 WRCYC	@	18 NC	#	18 -LM ACK H	#	18
19 LM INT	#	19 NC	#	19 -LOADMD H	#	19
20 MEMPAR FROM LM	@	20 NC	#	20 -LM IGNPAR H	#	20
21 ----------		21 ----------		21		21
22 ----------		22 ----------		22		22
23 ----------		23 ----------		23		23
24 ----------		24 ----------		24		24
25 ----------		25 ----------		25		25
26 ----------		26 ----------		26 ----------		26 ----------
27 ----------		27 ----------		27 ----------		27 ----------
28 ----------		28 ----------		28 ----------		28 ----------
29 ----------		29 ----------		29 ----------		29 ----------
30 ----------		30 ----------		30 ----------		30 ----------
31 ----------		31 ----------		31 ----------		31 ----------
32 ----------		32 ----------		32 ----------		32 ----------
33 ----------		33 ----------		33 ----------		33 ----------
34 ----------		34 ----------		34 ----------		34 ----------
35 ----------		35 ----------		35 ----------		35 ----------
36 ----------		36 ----------		36 ----------		36 ----------
37 ----------		37 ----------		37 ----------		37 ----------
38 ----------		38 ----------		38 ----------		38 ----------
39 ----------		39 ----------		39 ----------		39 ----------
40 ----------		40 ----------		40 ----------		40 ----------
				41 ----------		41 ----------
				42 ----------		42 ----------
				43 ----------		43 ----------
				44 ----------		44 ----------
				45 ----------		45 ----------
				46 ----------		46 ----------
				47 ----------		47 ----------
				48 ----------		48 ----------
				49 ----------		49 ----------
				50 ----------		50 ----------

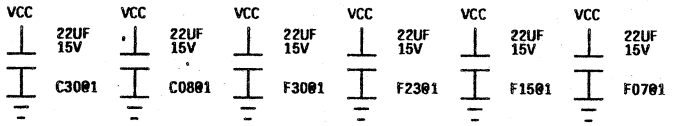
LISPM Bus Interface CADR1;BUSINT UML 25-MAR-81 0823
******* EDGE CONNECTIONS Flags: (# Output, @ Terminator, ---- Dedicated ground, ++++ Dedicated power) *******

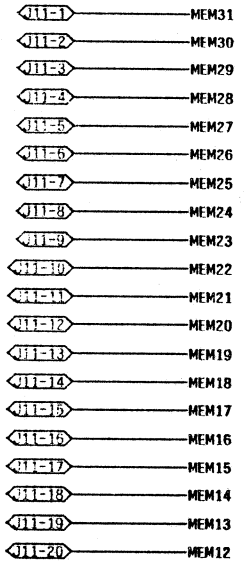
-J11-		-J12-	
01 MEM31	#	01 MEM11	#
02 MEM30	#	02 MEM10	#
03 MEM29	#	03 MEM9	#
04 MEM28	#	04 MEM8	#
05 MEM27	#	05 MEM7	#
06 MEM26	#	06 MEM6	#
07 MEM25	#	07 MEM5	#
08 MEM24	#	08 MEM4	#
09 MEM23	#	09 MEM3	#
10 MEM22	#	10 MEM2	#
11 MEM21	#	11 MEM1	#
12 MEM20	#	12 MEM0	#
13 MEM19	#	13 MEMPAR TO LM	#
14 MEM18	#	14 -ADRPAR H	
15 MEM17	#	15 -ADR21 H	
16 MEM16	#	16 -ADR20 H	
17 MEM15	#	17 -ADR19 H	
18 MEM14	#	18 -ADR18 H	
19 MEM13	#	19 -ADR17 H	
20 MEM12	#	20 -ADR16 H	
21 ----------		21 ----------	
22 ----------		22 ----------	
23 ----------		23 ----------	
24 ----------		24 ----------	
25 ----------		25 ----------	
26 ----------		26 ----------	
27 ----------		27 ----------	
28 ----------		28 ----------	
29 ----------		29 ----------	
30 ----------		30 ----------	
31 ----------		31 ----------	
32 ----------		32 ----------	
33 ----------		33 ----------	
34 ----------		34 ----------	
35 ----------		35 ----------	
36 ----------		36 ----------	
37 ----------		37 ----------	
38 ----------		38 ----------	
39 ----------		39 ----------	
40 ----------		40 ----------	


12 IN PARITY 93S48
BUS 0
BUS 1
BUS 2
BUS 3
BUS 4
BUS 5
BUS 6
BUS 7
BUS 8
BUS 9
BUS 10
BUS 11
7:
6:
5:
4:
3:
2:
1:
15:
14:
13:
12:
11:
PE 10:
PO 9:
NC
BUS 0-11 PAR ODD
C30
12 IN PARITY 93S48
BUS 12
BUS 13
BUS 14
BUS 15
BUS 16
BUS 17
BUS 18
BUS 19
BUS 20
BUS 21
BUS 22
BUS 23
7:
6:
5:
4:
3:
2:
1:
15:
14:
13:
12:
11:
PE 10:
PO 9:
NC
BUS 12-23 PAR ODD
C29
12 IN PARITY 93S48
BUS 24
BUS 25
BUS 26
BUS 27
BUS 28
BUS 29
BUS 30
BUS 31
BUS 0-11 PAR ODD
BUS 12-23 PAR ODD
GND
GND
7:
6:
5:
4:
3:
2:
1:
15:
14:
13:
12:
11:
PE 10:
PO 9:
BUS PAR EVEN
BUS PAR ODD
C28

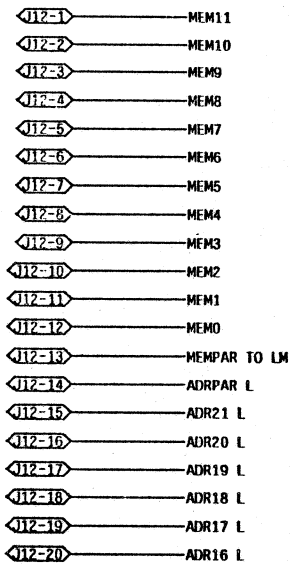


CBUS	BUS FROM UNIBUS	10-DEC-1980 10:05	AI: CADR1; BUSSEL

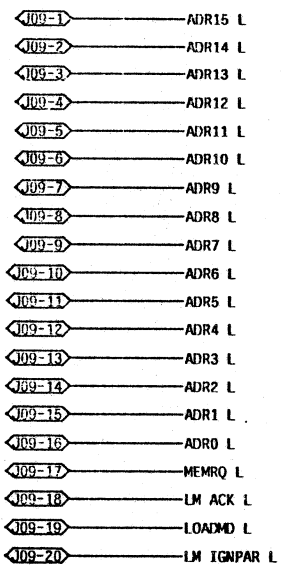

VCC
22UF
15V
C3001
VCC
22UF
15V
C0801
VCC
22UF
15V
F3001
VCC
22UF
15V
F2301
VCC
22UF
15V
F1501
VCC
22UF
15V
F0701



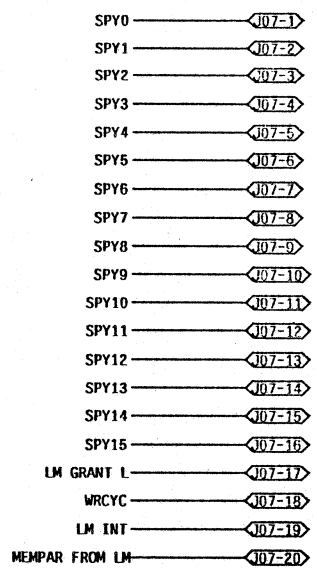
CADR; BCPINS
to 1AJ1



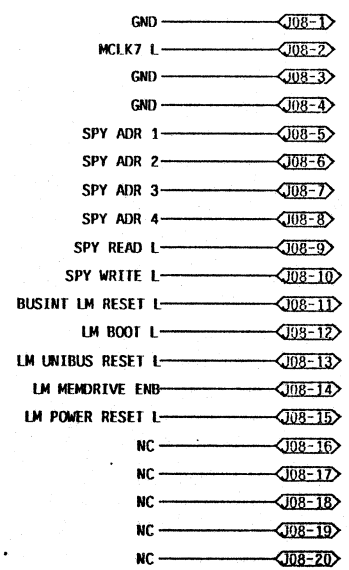
CADR; BCPINS
to 1BJ1



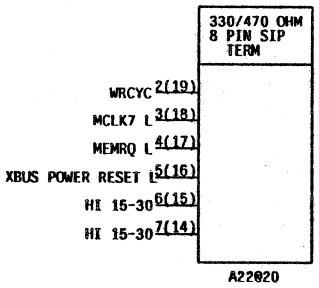
CADR; BCPINS
to 1CJ1

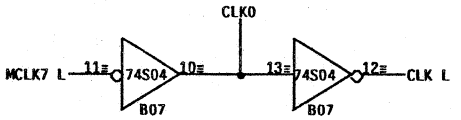


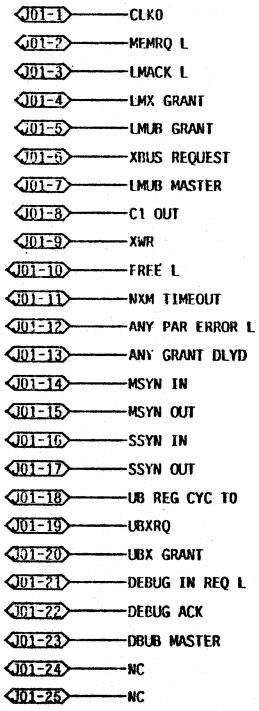
TO 3AJ1
CADR; BCPINS

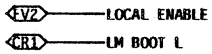


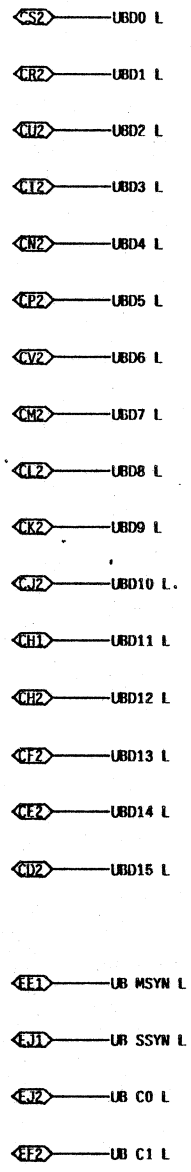
TO 5AJ1
CADR; MBCPIN

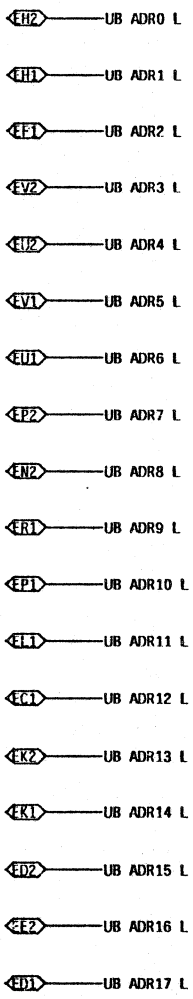


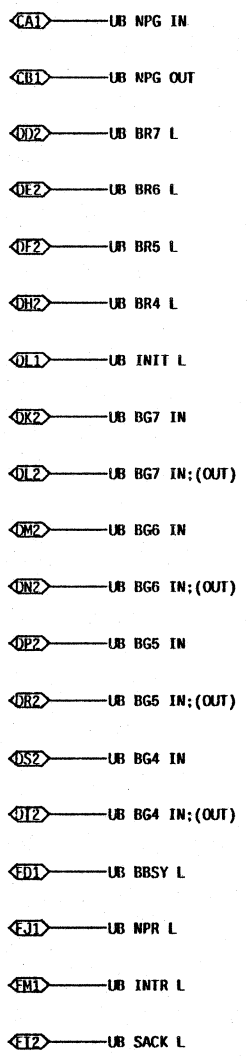


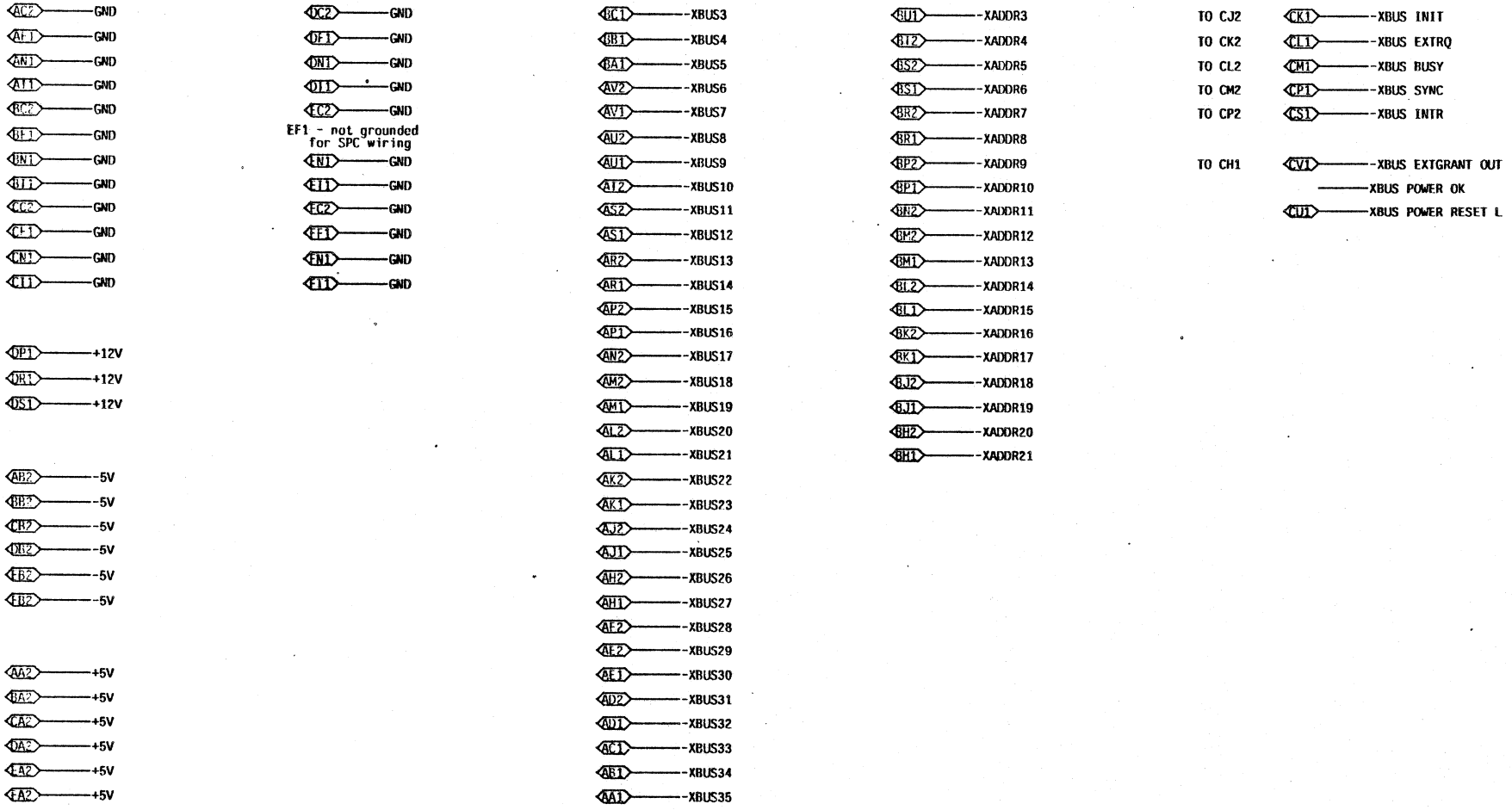

J01-1
CLK0
J01-2
MEMRQ L
J01-3
LMACK L
J01-4
LMX GRANT
J01-5
LMUB GRANT
J01-6
XBUS REQUEST
J01-7
LMUB MASTER
J01-8
C1 OUT
J01-9
XWR
J01-10
FREE L
J01-11
NXM TIMEOUT
J01-12
ANY PAR ERROR L
J01-13
ANY GRANT DLYD
J01-14
MSYN IN
J01-15
MSYN OUT
J01-16
SSYN IN
J01-17
SSYN OUT
J01-18
UB REG CYC TO
J01-19
UBXRQ
J01-20
UBX GRANT
J01-21
DEBUG IN REQ L
J01-22
DEBUG ACK
J01-23
DBUB MASTER
J01-24
NC
J01-25
NC


EV2 LOCAL ENABLE
CR1 LM BOOT L

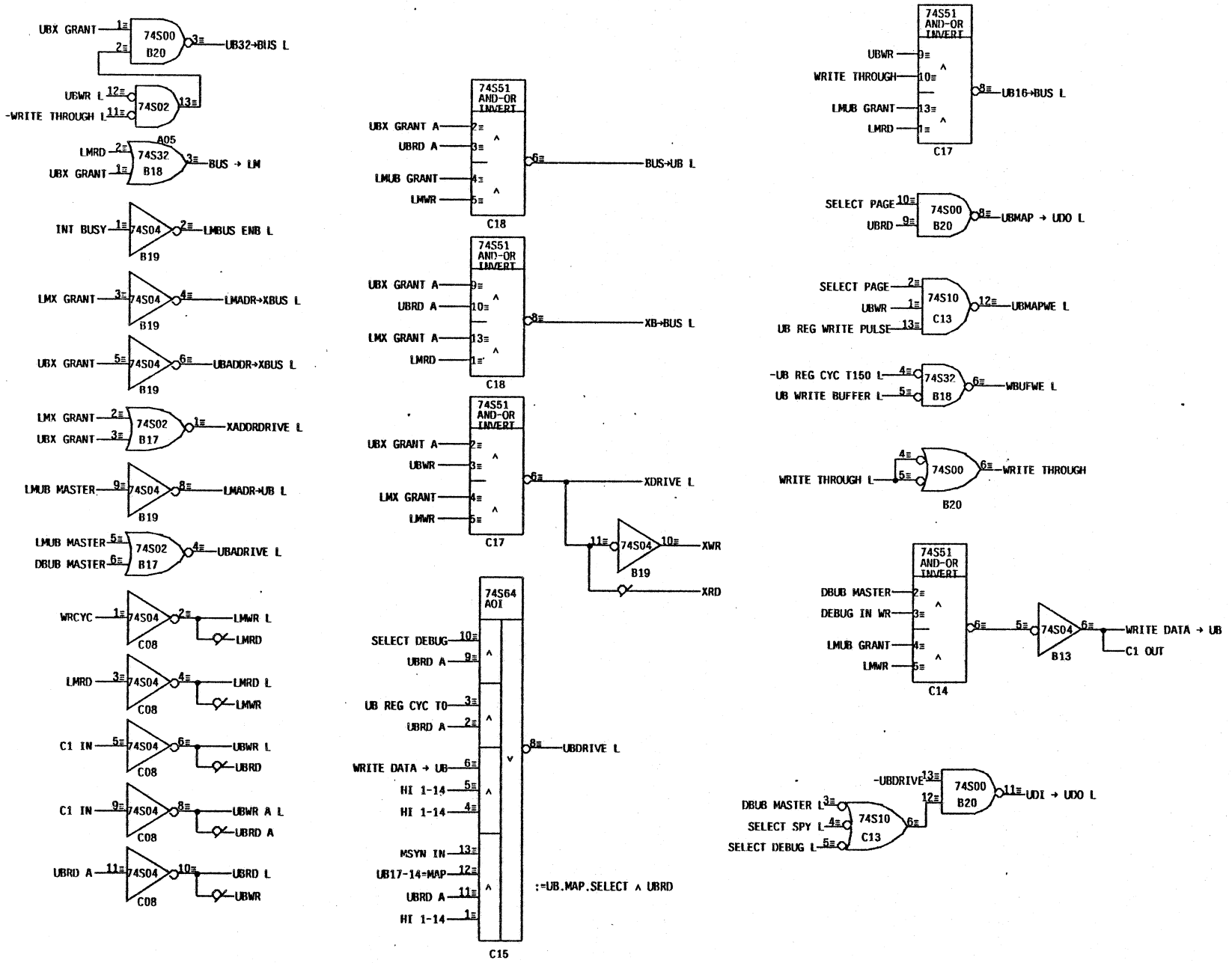

CS2 UBD0 L
CR2 UBD1 L
CU2 UBD2 L
CT2 UBD3 L
CN2 UBD4 L
CP2 UBD5 L
CV2 UBD6 L
CM2 UBD7 L
CL2 UBD8 L
CK2 UBD9 L
CJ2 UBD10 L
CH1 UBD11 L
CH2 UBD12 L
CF2 UBD13 L
CE2 UBD14 L
CD2 UBD15 L
EE1 UB MSYN L
EJ1 UB SSYN L
EJ2 UB C0 L
EF2 UB C1 L


EH2 UB ADR0 L
EH1 UB ADR1 L
EF1 UB ADR2 L
EV2 UB ADR3 L
EU2 UB ADR4 L
EV1 UB ADR5 L
EU1 UB ADR6 L
EP2 UB ADR7 L
EN2 UB ADR8 L
ER1 UB ADR9 L
EP1 UB ADR10 L
EL1 UB ADR11 L
EC1 UB ADR12 L
EK2 UB ADR13 L
EK1 UB ADR14 L
ED2 UB ADR15 L
EE2 UB ADR16 L
ED1 UB ADR17 L

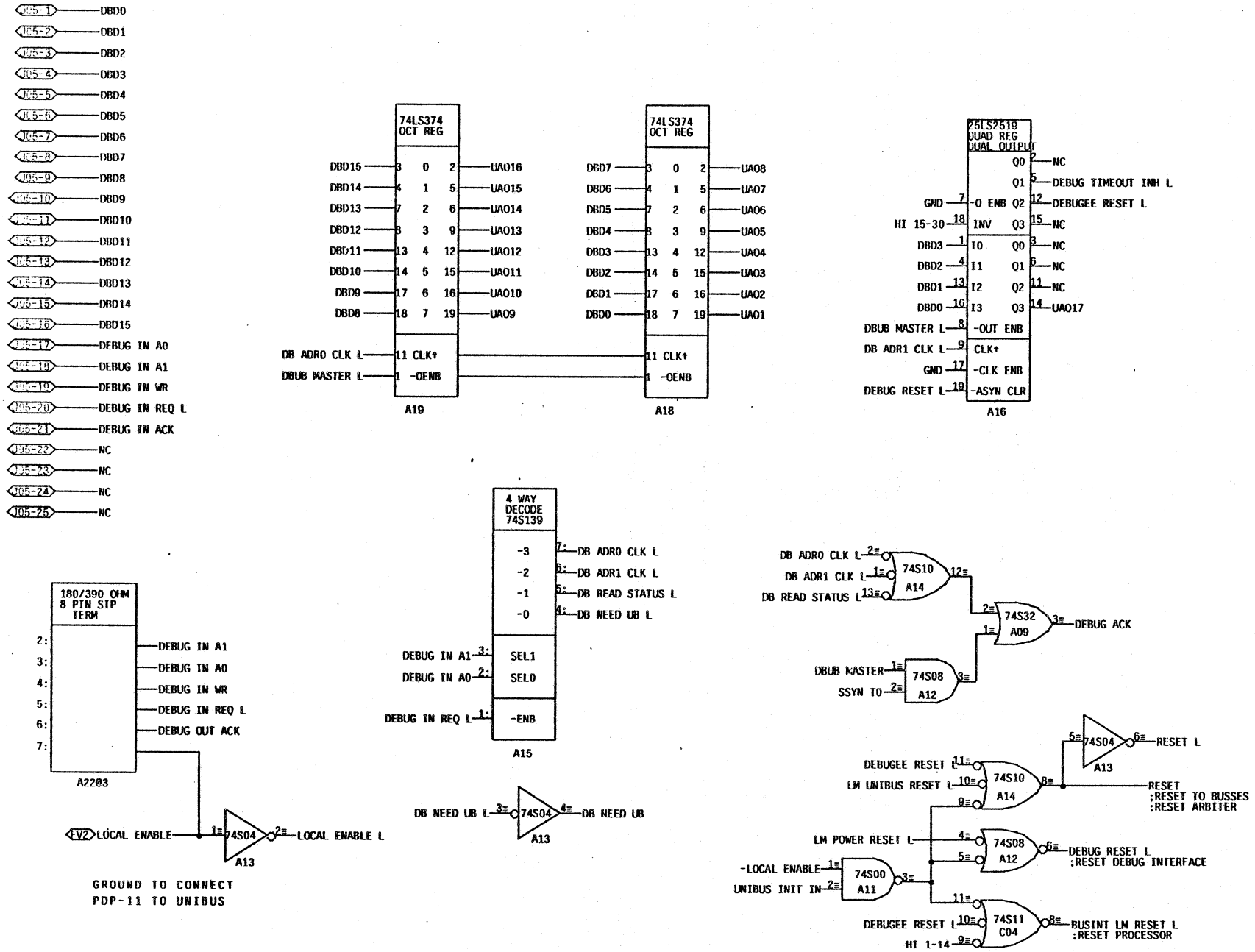

CA1 UB NPG IN
CB1 UB NPG OUT
DD2 UB BR7 L
DE2 UB BR6 L
DF2 UB BR5 L
DH2 UB BR4 L
DL1 UB INIT L
DK2 UB BG7 IN
DL2 UB BG7 IN;(OUT)
DM2 UB BG6 IN
DN2 UB BG6 IN;(OUT)
DP2 UB BG5 IN
DR2 UB BG5 IN;(OUT)
DS2 UB BG4 IN
DT2 UB BG4 IN;(OUT)
ED1 UB BBSY L
EJ1 UB NPR L
EM1 UB INTR L
ET2 UB SACK L



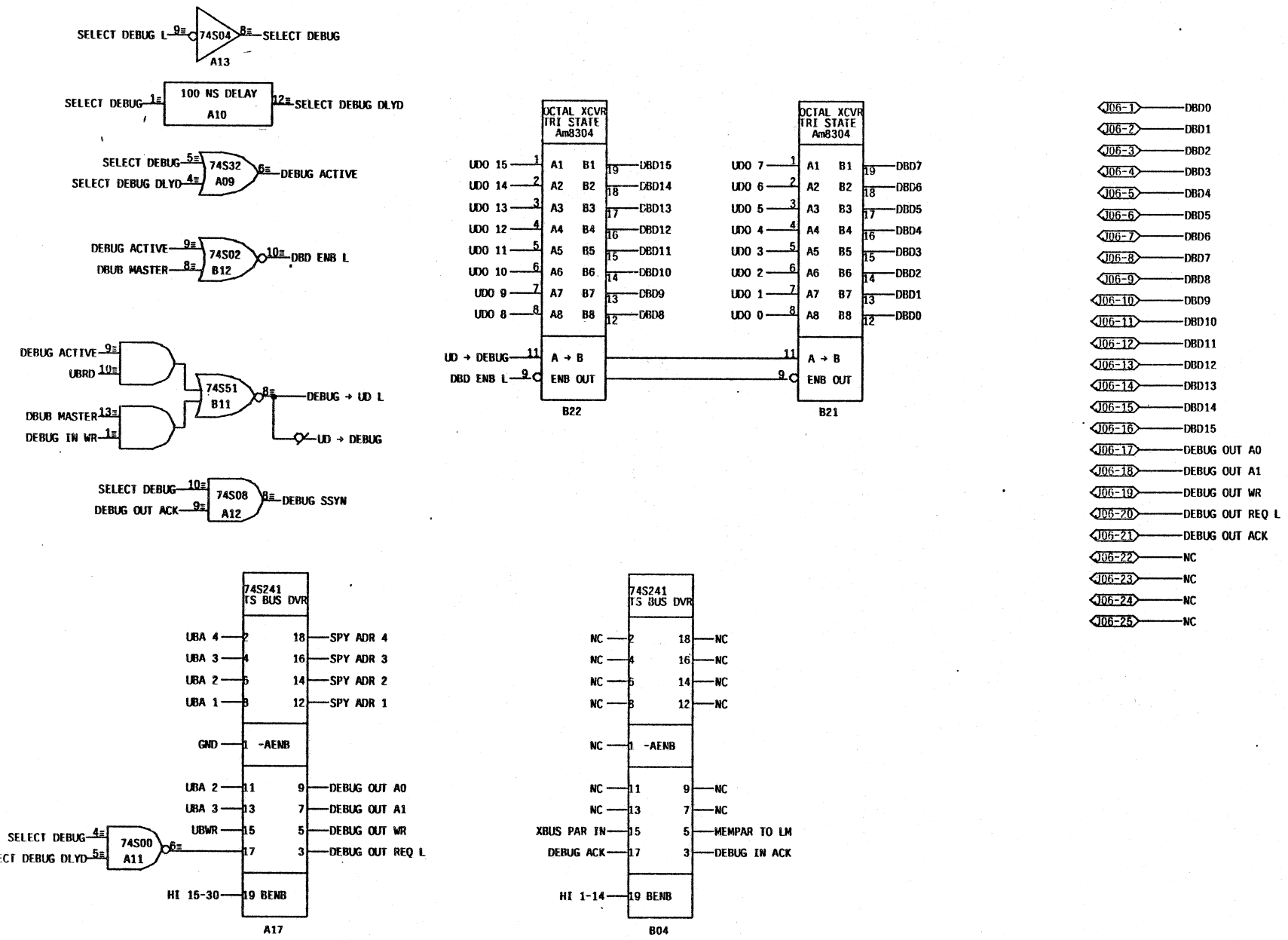
CBUS	XBUS BACKPLANE CONNS	10-DEC-1980 10:09	AI: CADR1; CXBUS



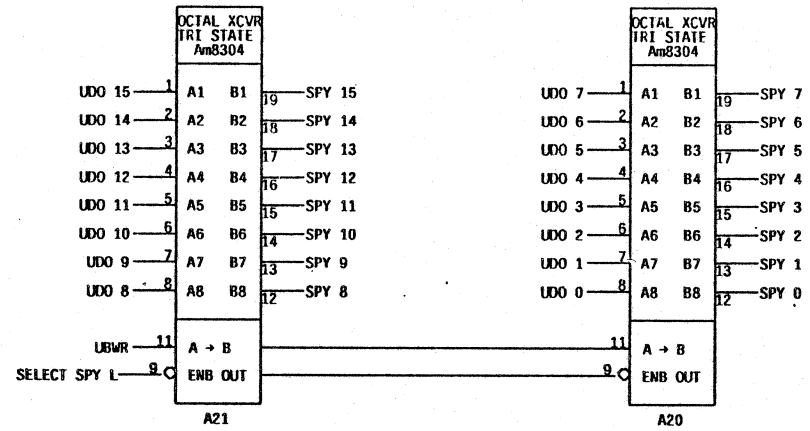
CBUS	DATA PATH CONTROL	10-DEC-1980 17:33	AI: CADR1; DATCTL

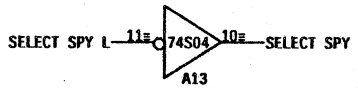


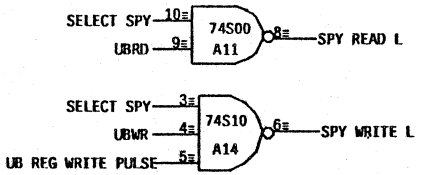
CBUS	DEBUGEE DATA PATH	10-DEC-1980 10:25	AI: CADR1; DBGIN



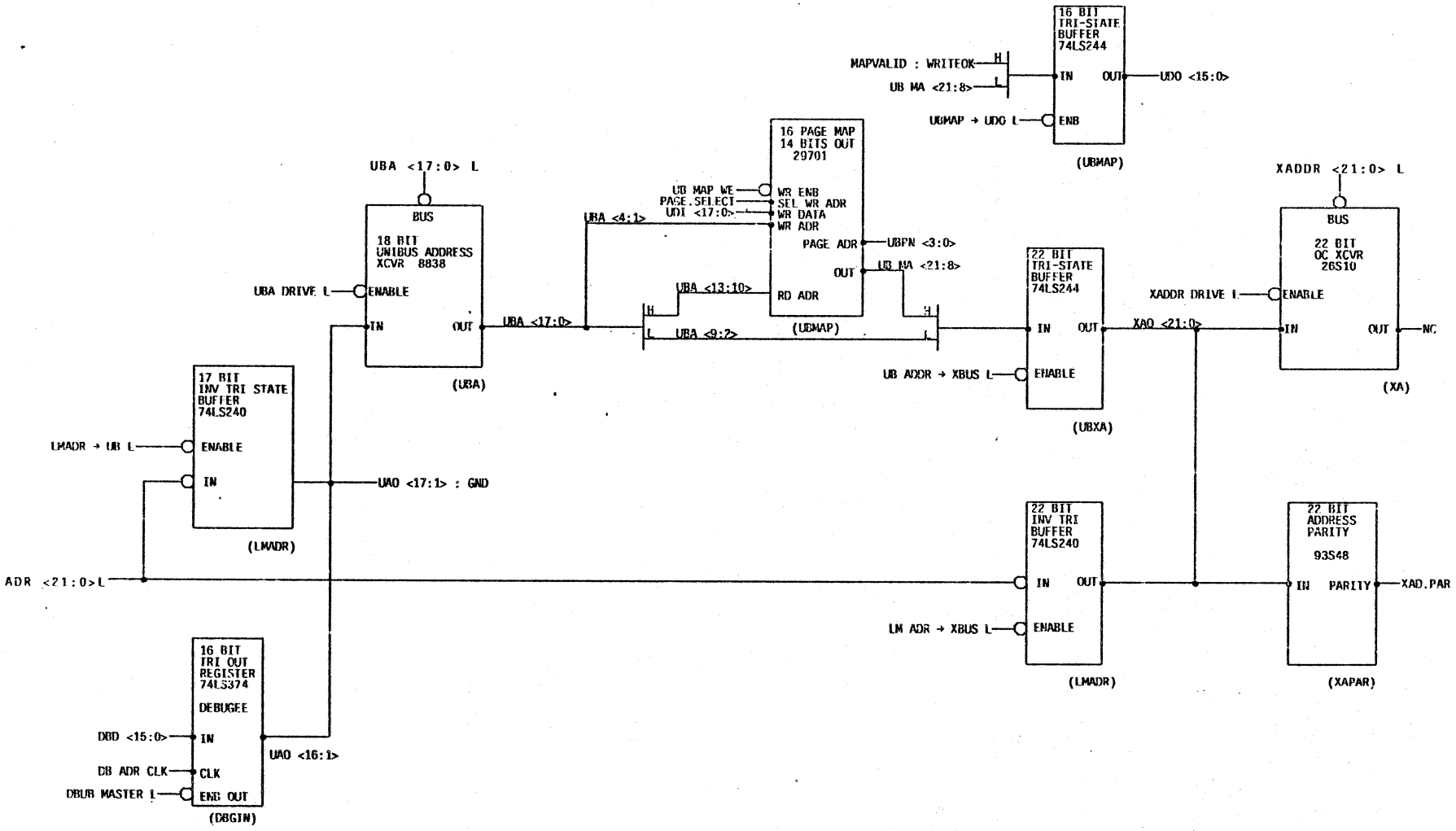
CBUS	DEBUGER DATA PATH	10-DEC-1980 10:26	AI: CADR1; DBGOUT



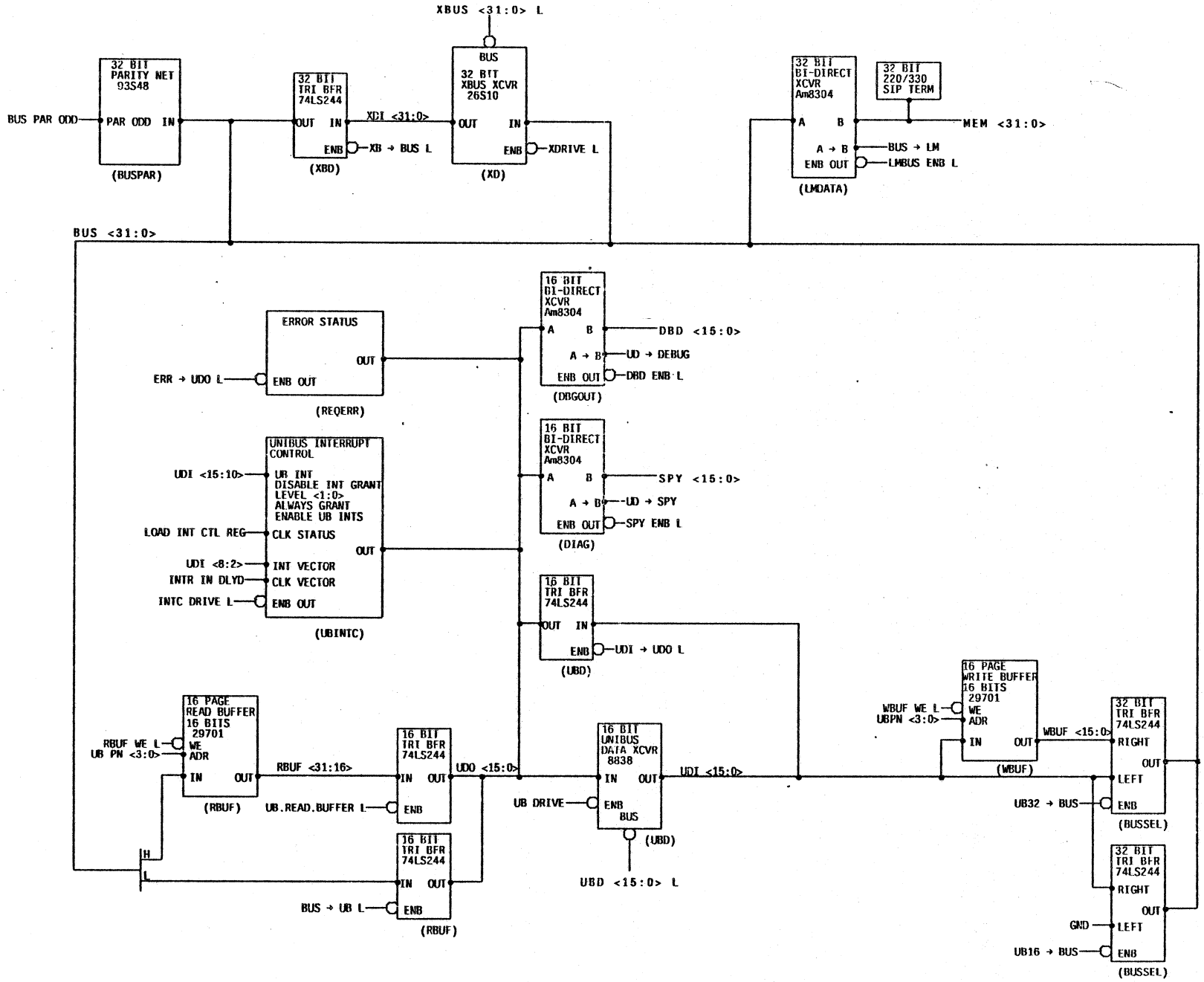




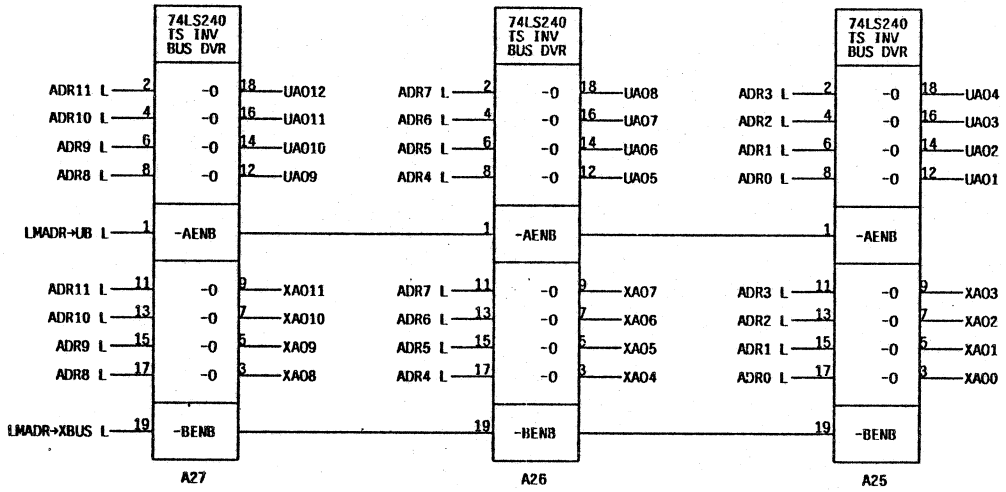
CBUS	DIAGNOSTIC BUS	10-DEC-1980 10:26	AI: CADR1; DIAG

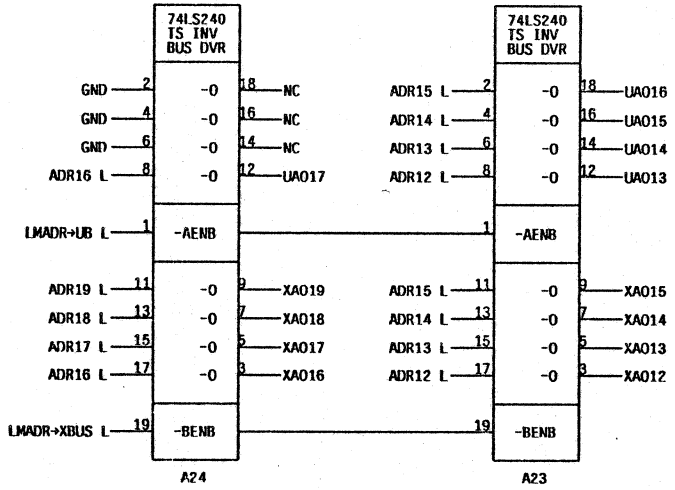


CBUS	DATA PATH - ADDRESS	25-MAY-1978 16:41	AI: CADR1; DPADR



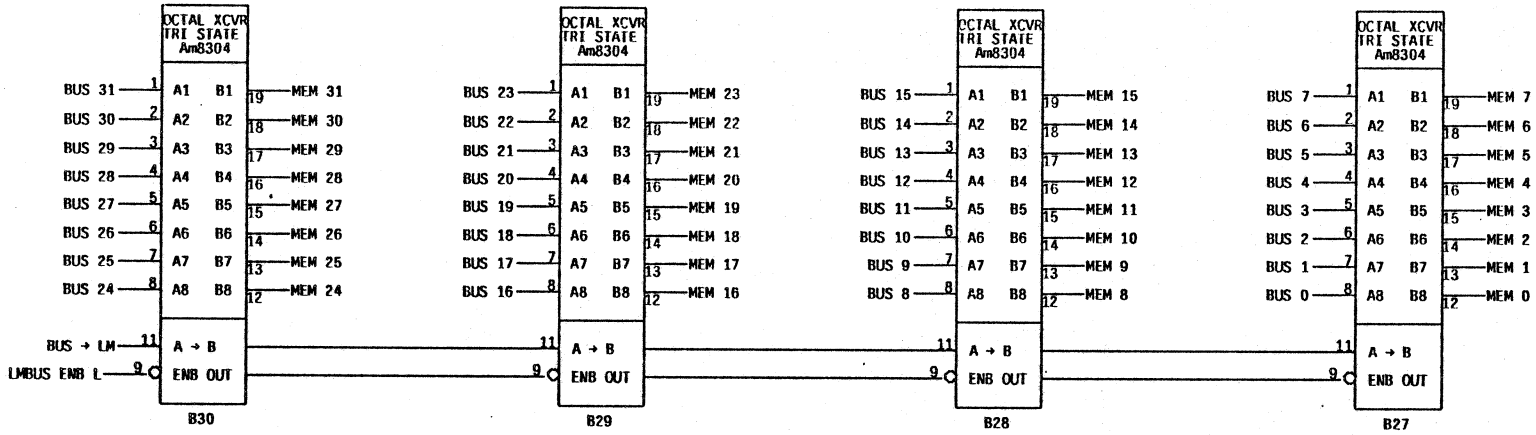
CBUS	DATA PATH - XBUS DATA	25-MAY-1978 16:47	AI: CADR1; DPDATA

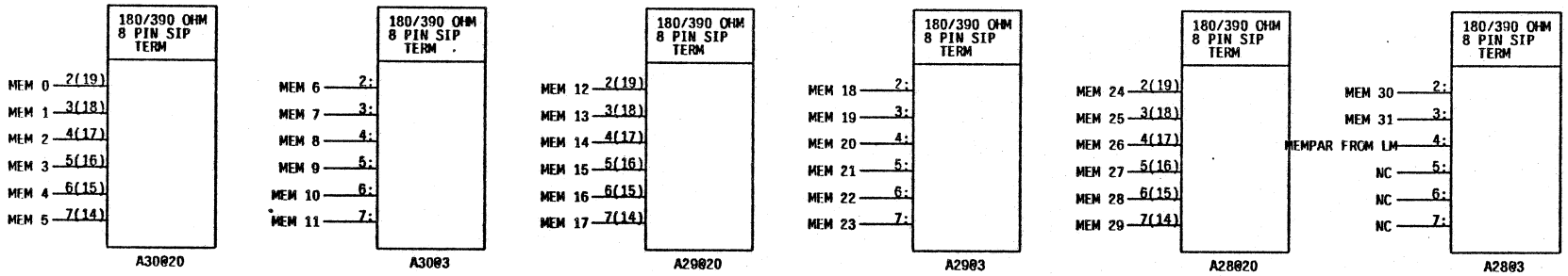




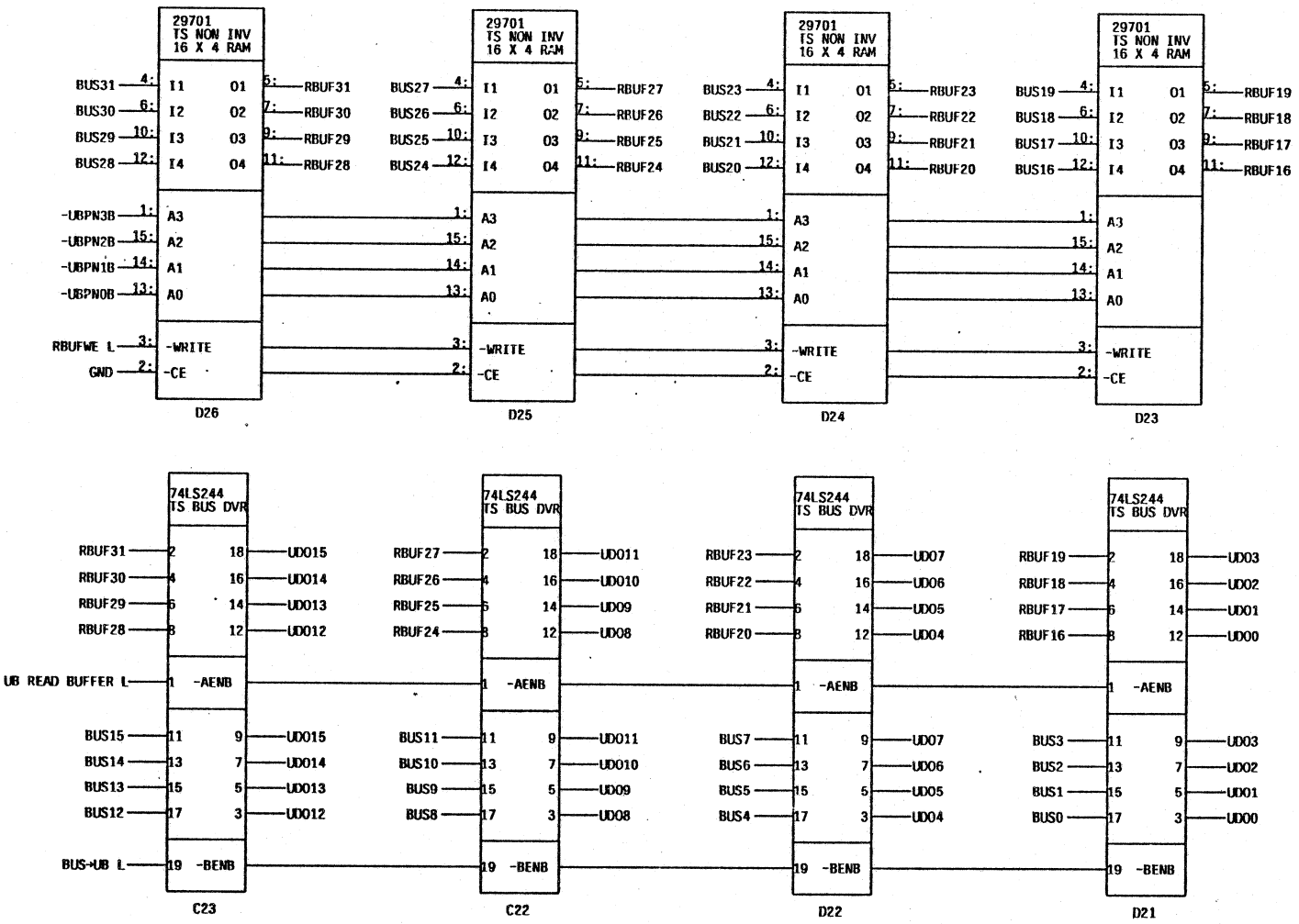
ADR <21:20> APPEAR
ON CADR1; UBINTC

CBUS	ADDRESS FROM LISP MACHINE	10-DEC-1980 10:27	AI: CADR1; LMADR

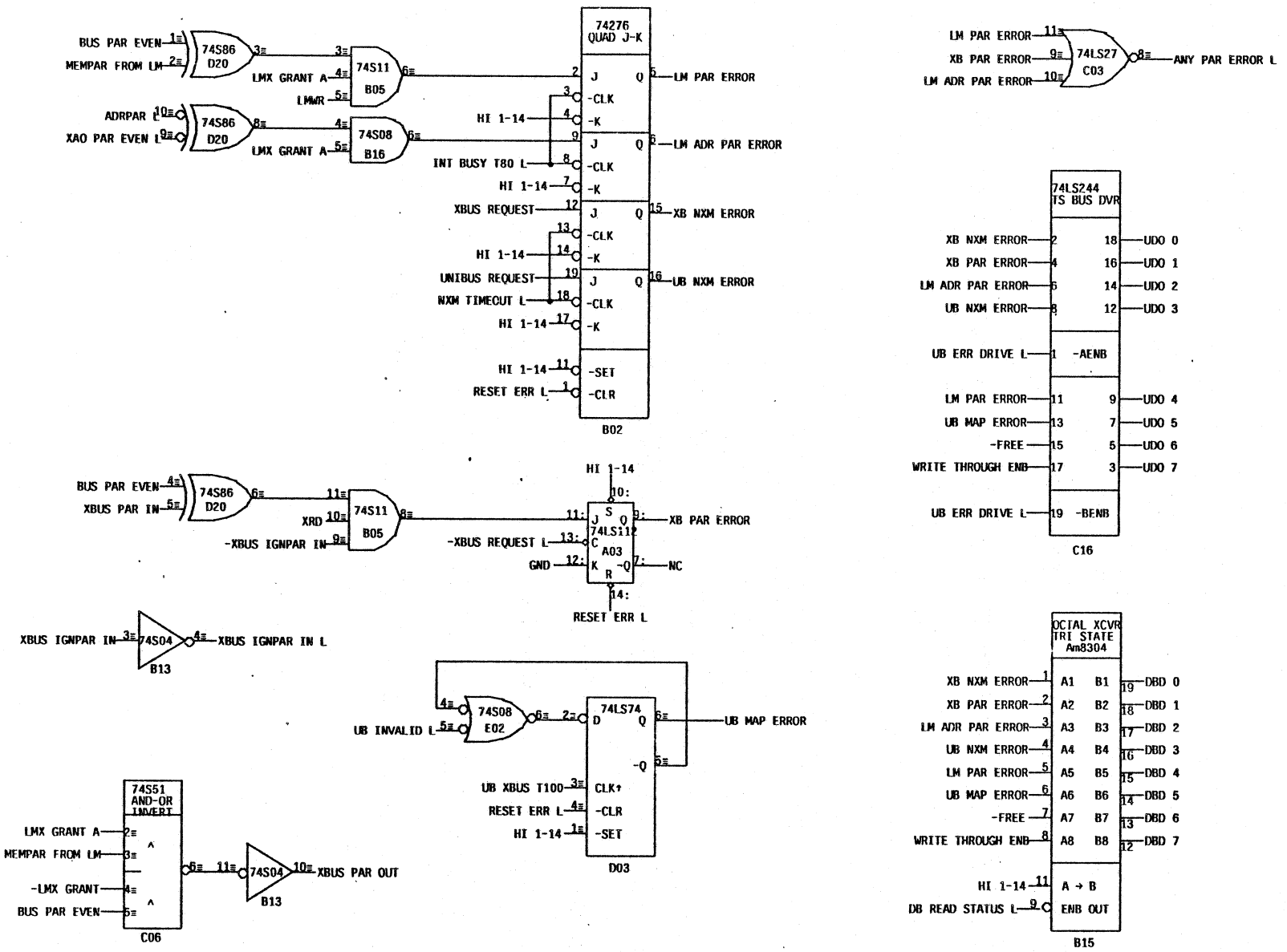




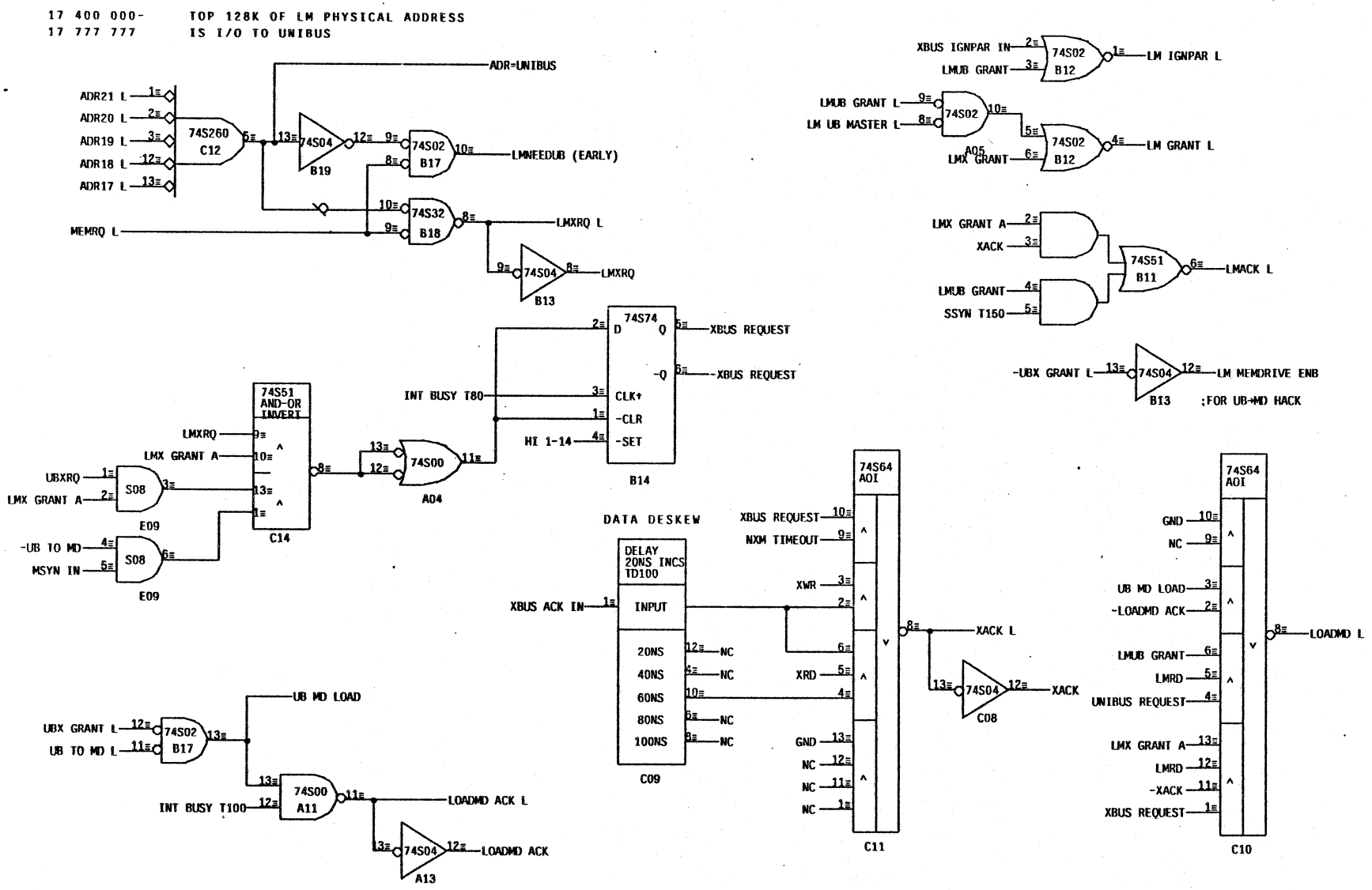
NOTE: MEMPAR TO LM is on DBGOUT

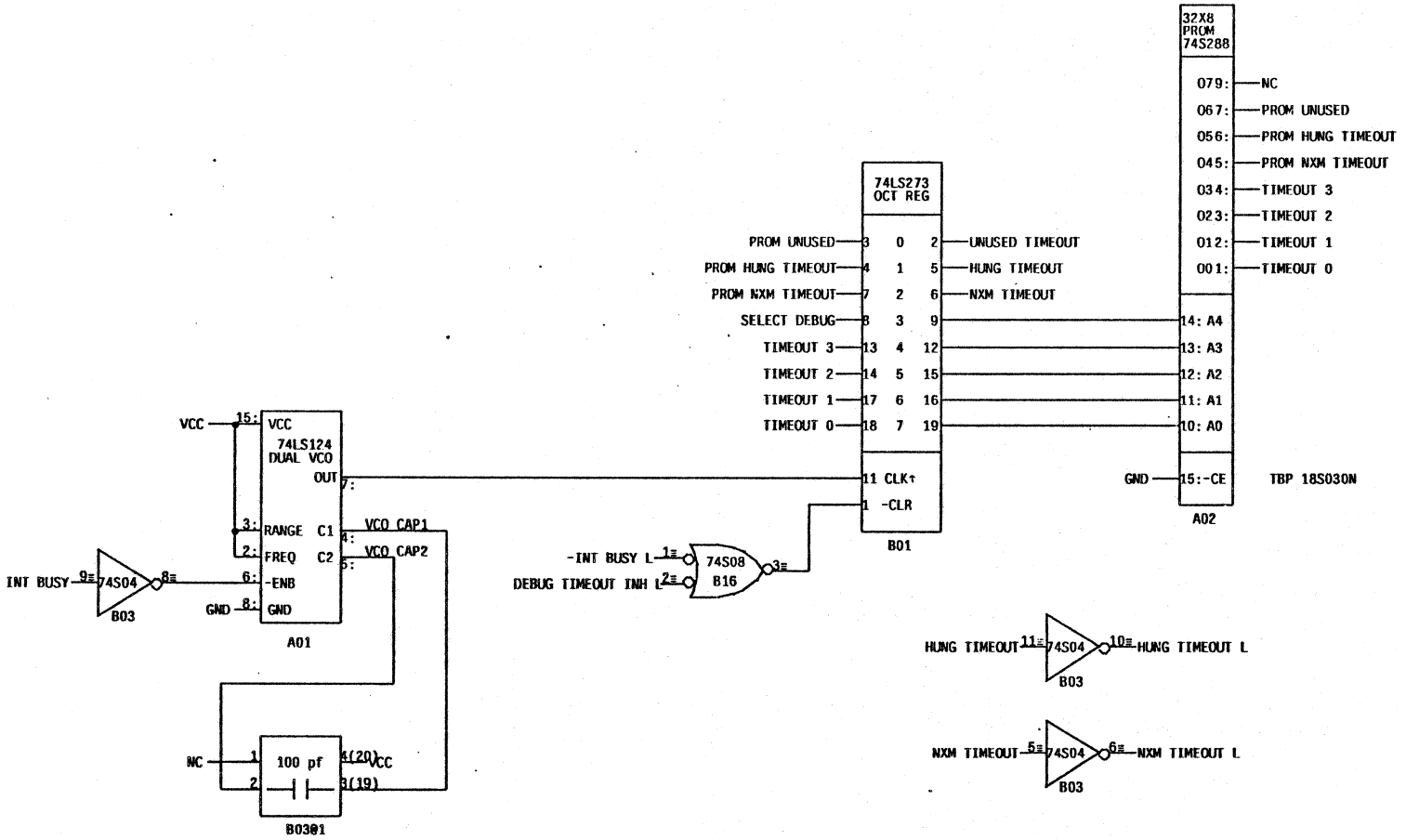


CBUS	READ BUFFER	10-DEC-1980 10:28	AI: CADR1; RBUF



CBUS	ERROR LOGIC	10-DEC-1980 10:40	AI: CADR1; REQERR

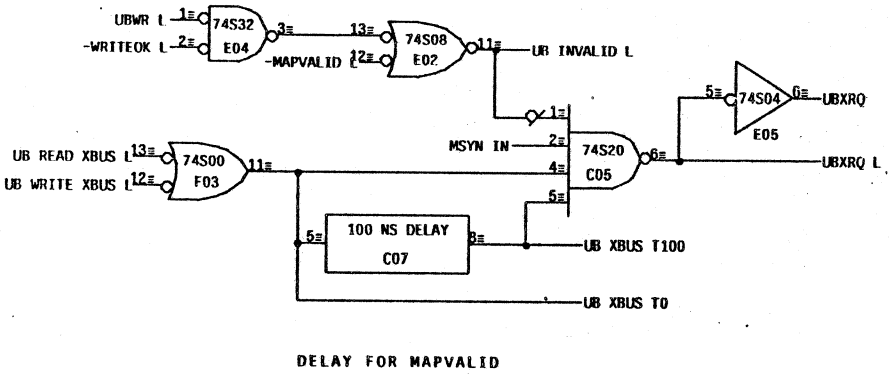

17 400 000-
17 777 777
TOP 128K OF LM PHYSICAL ADDRESS
IS I/O TO UNIBUS
ADR=UNIBUS
ADR21 L
ADR20 L
ADR19 L
ADR18 L
ADR17 L
74S260
C12
74S04
B19
74S02
B17
LMNEEDUB (EARLY)
MEMRQ L
74S32
B18
LMXRQ L
74S04
B13
LMXRQ
XBUS IGNPAR IN
LMUB GRANT
74S02
B12
LM IGNPAR L
LMUB GRANT L
LM UB MASTER L
74S02
A05
74S02
B12
LM GRANT L
LMX GRANT
LMX GRANT A
XACK
74S51
B11
LMACK L
LMUB GRANT
SSYN T150
-UBX GRANT L
74S04
B13
LM MEMDRIVE ENB
;FOR UB→MD HACK
74S74
D Q
-Q
XBUS REQUEST
-XBUS REQUEST
INT BUSY T80
CLK↑
-CLR
-SET
HI 1-14
B14
74S51
AND-OR
INVERT
LMXRQ
LMX GRANT A
UBXRQ
S08
E09
-UB TO MD
MSYN IN
S08
E09
C14
74S00
A04
DATA DESKEW
DELAY
20NS INCS
TD100
INPUT
20NS
40NS
60NS
80NS
100NS
XBUS ACK IN
C09
74S64
AOI
XBUS REQUEST
NXM TIMEOUT
XWR
XRD
GND
NC
NC
NC
XACK L
74S04
C08
XACK
C11
74S64
AOI
GND
NC
UB MD LOAD
-LOADMD ACK
LMUB GRANT
LMRD
UNIBUS REQUEST
LMX GRANT A
LMRD
-XACK
XBUS REQUEST
LOADMD L
C10
UBX GRANT L
UB TO MD L
74S02
B17
UB MD LOAD
INT BUSY T100
74S00
A11
LOADMD ACK L
74S04
A13
LOADMD ACK



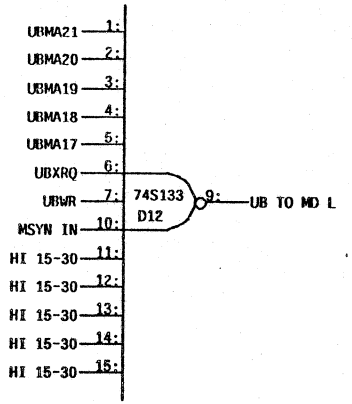
CBUS	XBUS & UNIBUS TIMEOUT	10-DEC-1980 15:26	AI: CADR1; REQTIM

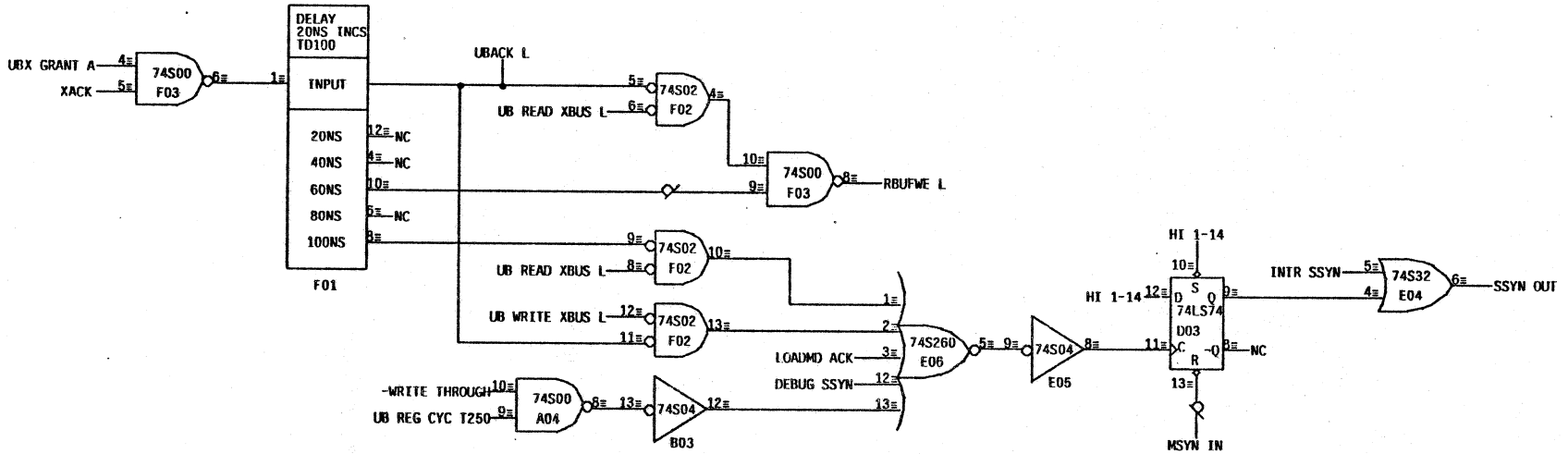
17 400 000 to
17 777 777

THESE ARE LM ADDRESSES THAT ARE MAPPED ONTO UNIBUS
THEY DON'T EXIST ON XBUS
THEREFORE THEY'LL MEAN UNIBUS TO MEMORY DATA REGISTER

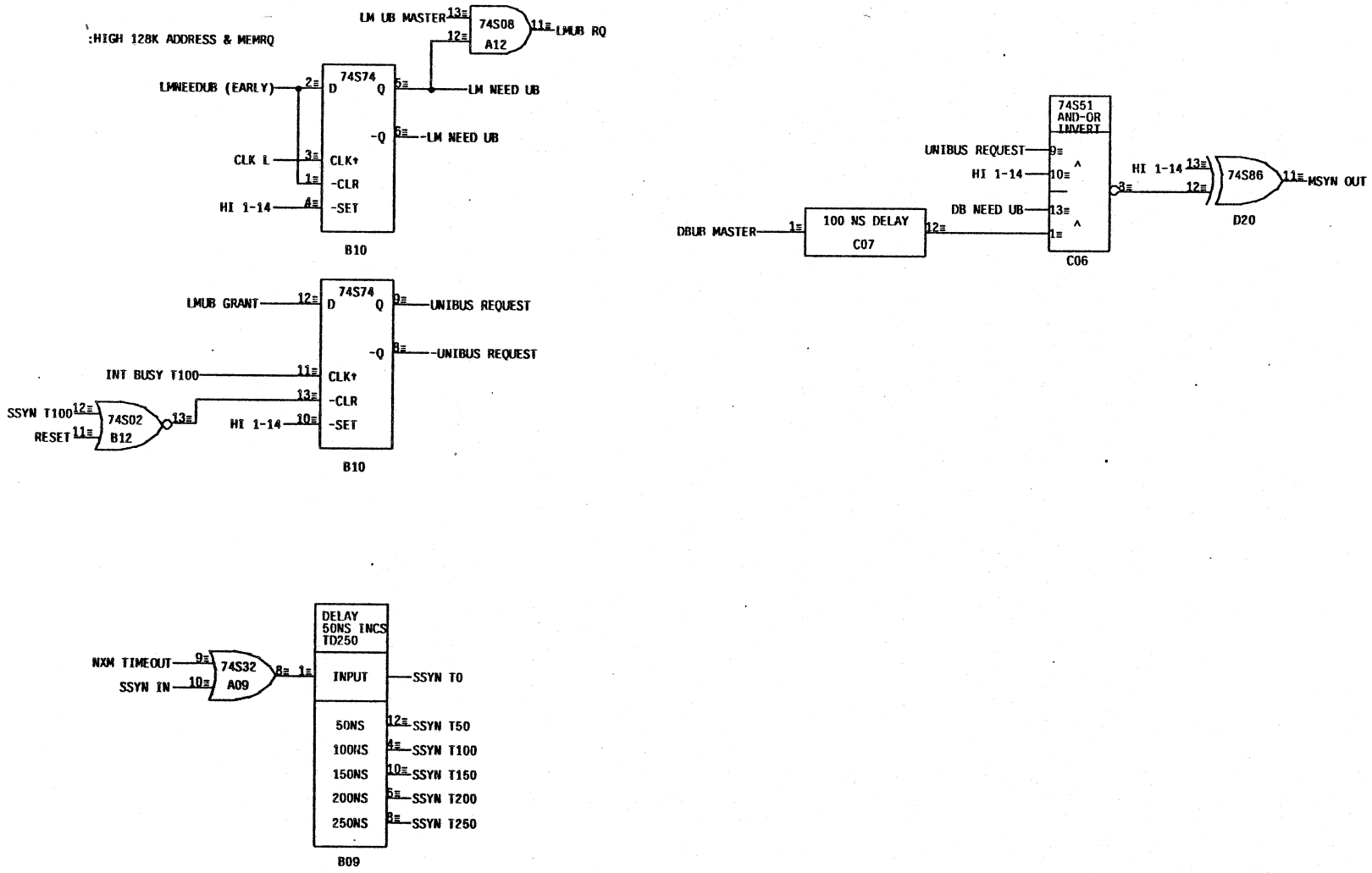


DELAY FOR MAPVALID

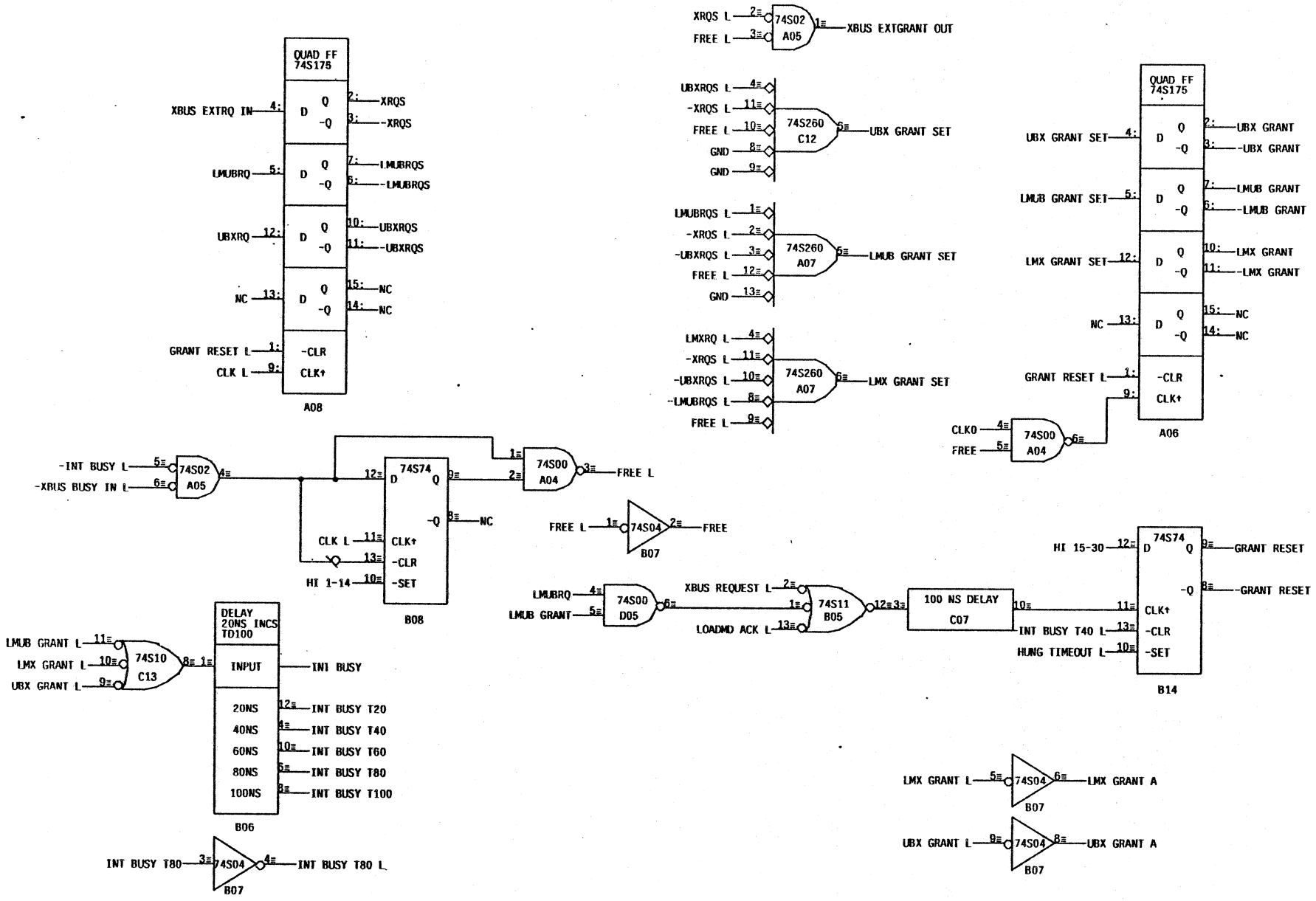




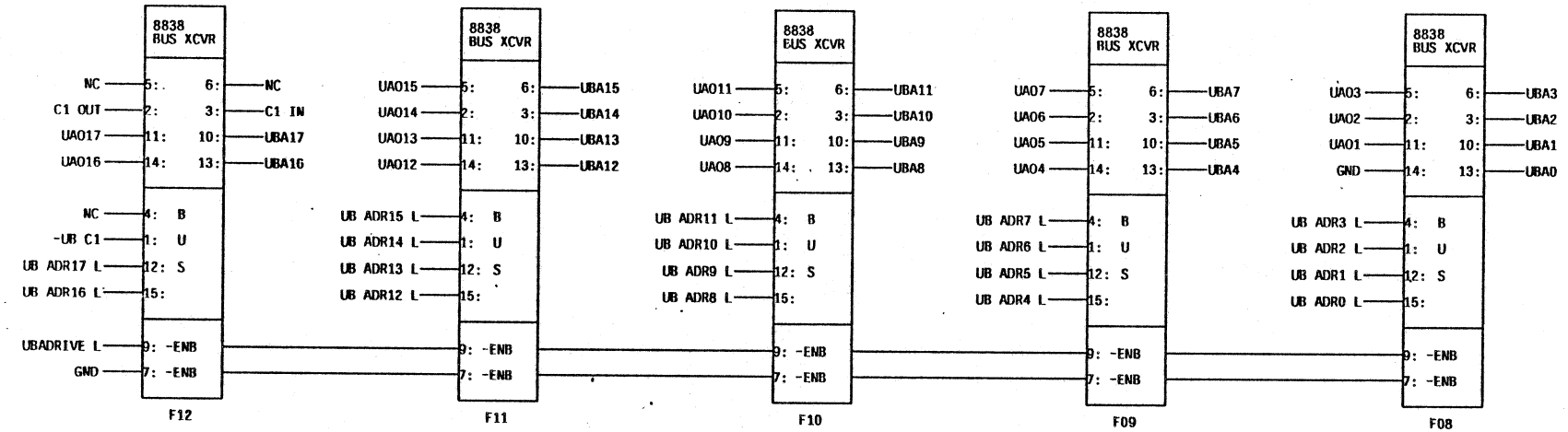
CBUS	XBUS REQUEST FROM UNIBUS	10-DEC-1980 15:36	AI: CADR1; REQU

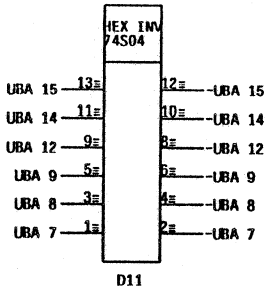


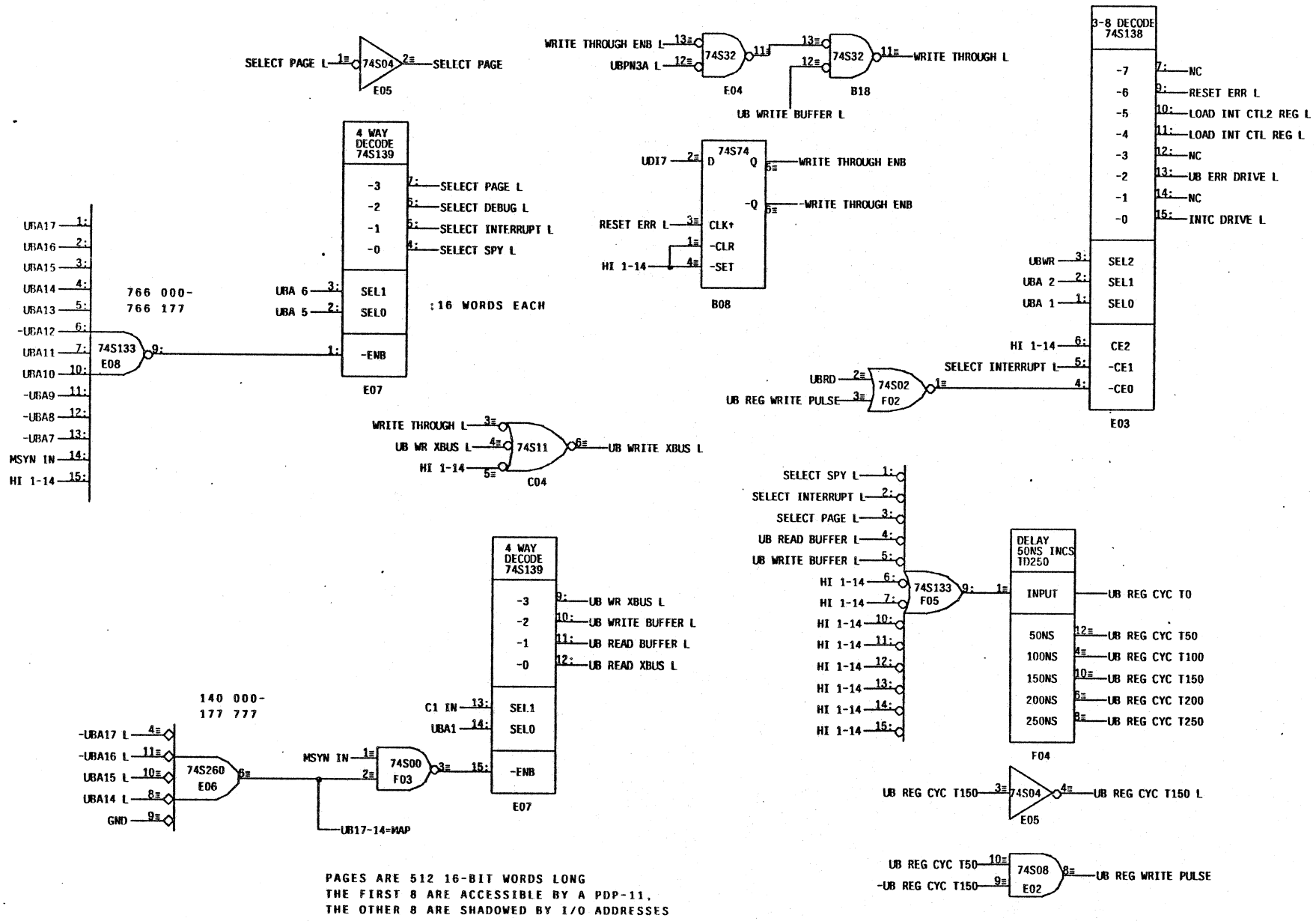
CBUS	LM & DEBUG TO UNIBUS	10-DEC-1980 15:23	AI: CADR1; REQUB

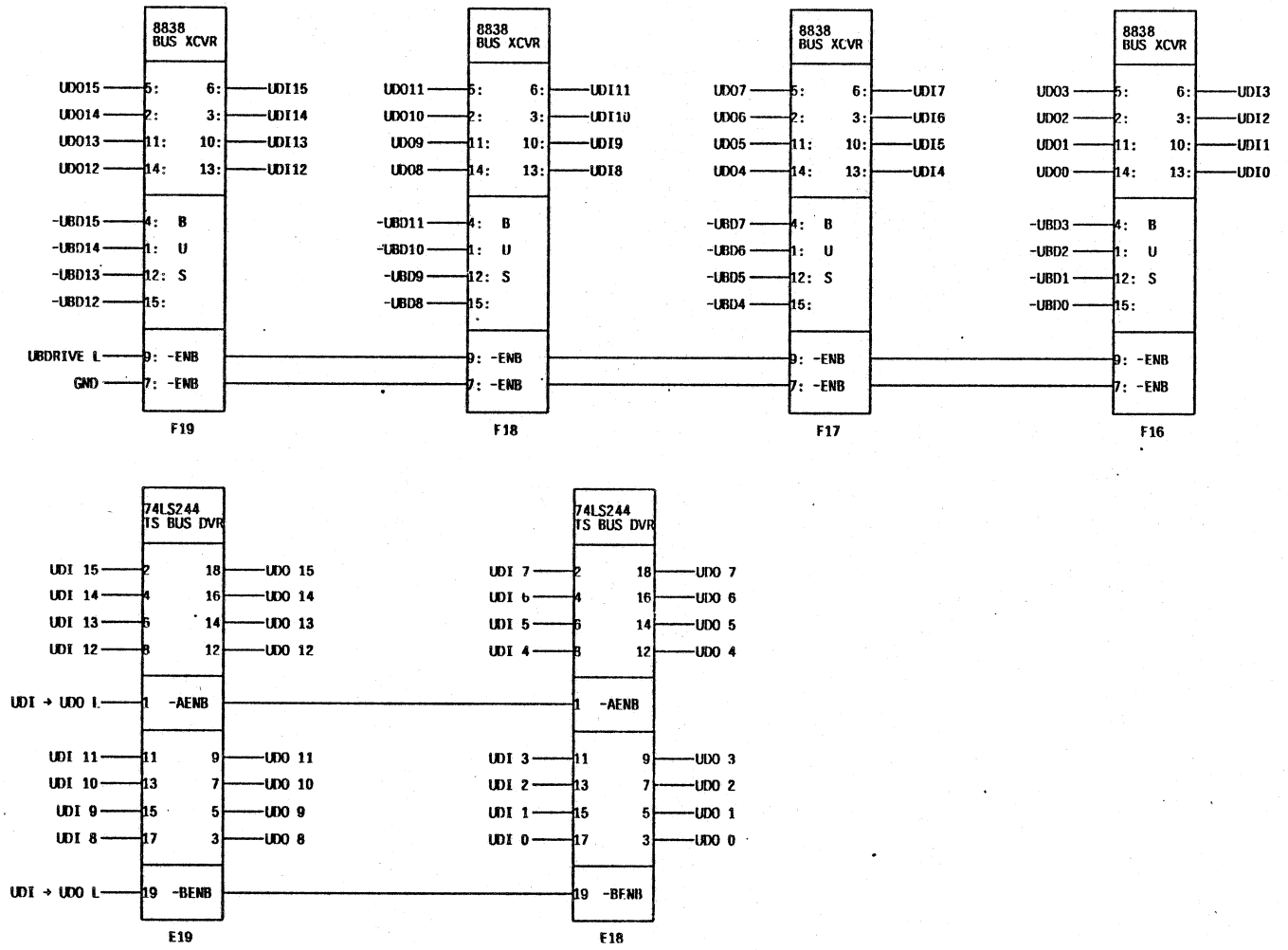


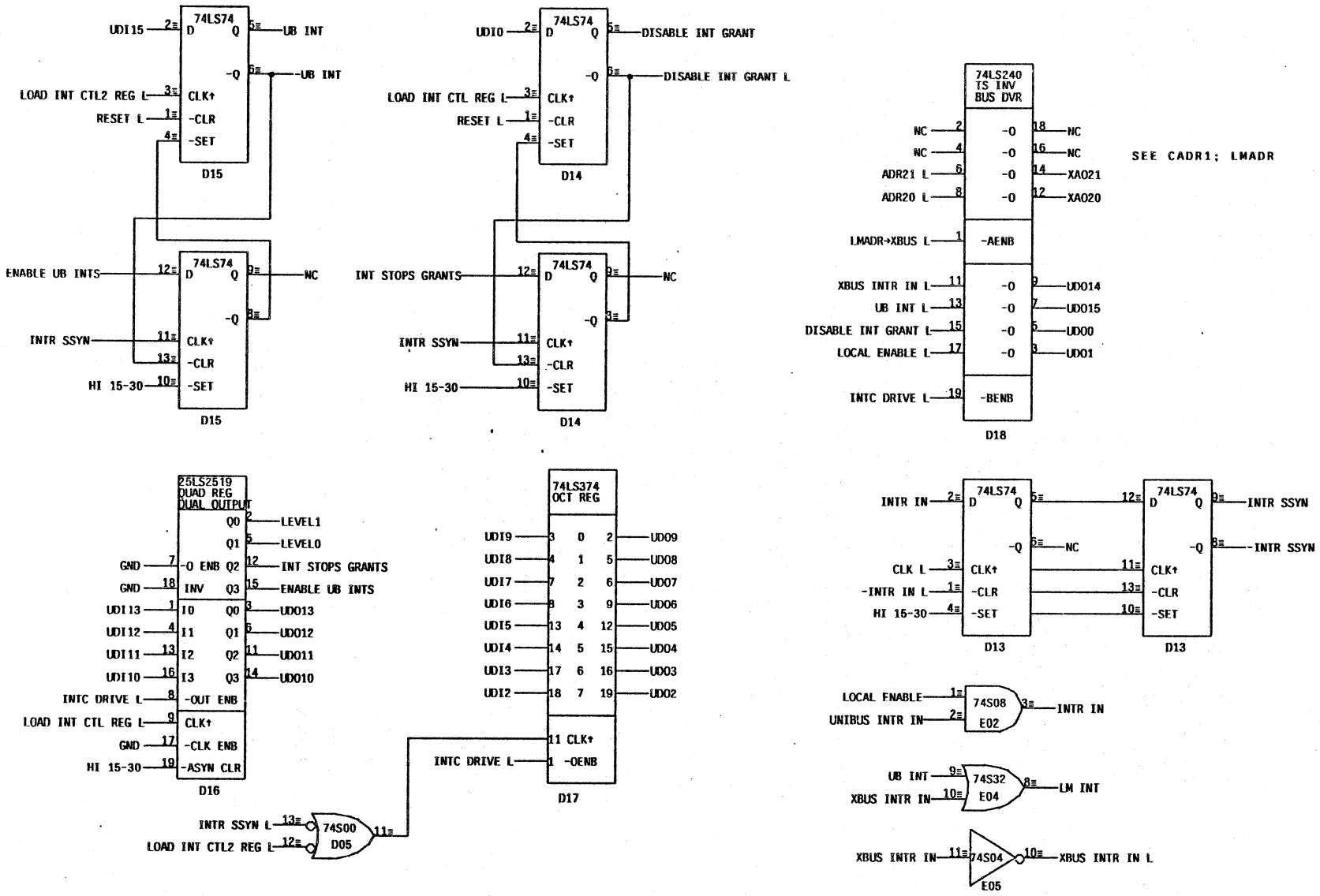
CBUS	REQUEST SYNCHRONIZER	10-DEC-1980 11:32	AI: CADR1; RQSYNC


8838
BUS XCVR
NC
C1 OUT
UA017
UA016
NC
-UB C1
UB ADR17 L
UB ADR16 L
UBADRIVE L
GND
-ENB
NC
C1 IN
UBA17
UBA16
F12
UA015
UA014
UA013
UA012
UB ADR15 L
UB ADR14 L
UB ADR13 L
UB ADR12 L
UBA15
UBA14
UBA13
UBA12
F11
UA011
UA010
UA09
UA08
UB ADR11 L
UB ADR10 L
UB ADR9 L
UB ADR8 L
UBA11
UBA10
UBA9
UBA8
F10
UA07
UA06
UA05
UA04
UB ADR7 L
UB ADR6 L
UB ADR5 L
UB ADR4 L
UBA7
UBA6
UBA5
UBA4
F09
UA03
UA02
UA01
GND
UB ADR3 L
UB ADR2 L
UB ADR1 L
UB ADR0 L
UBA3
UBA2
UBA1
UBA0
F08

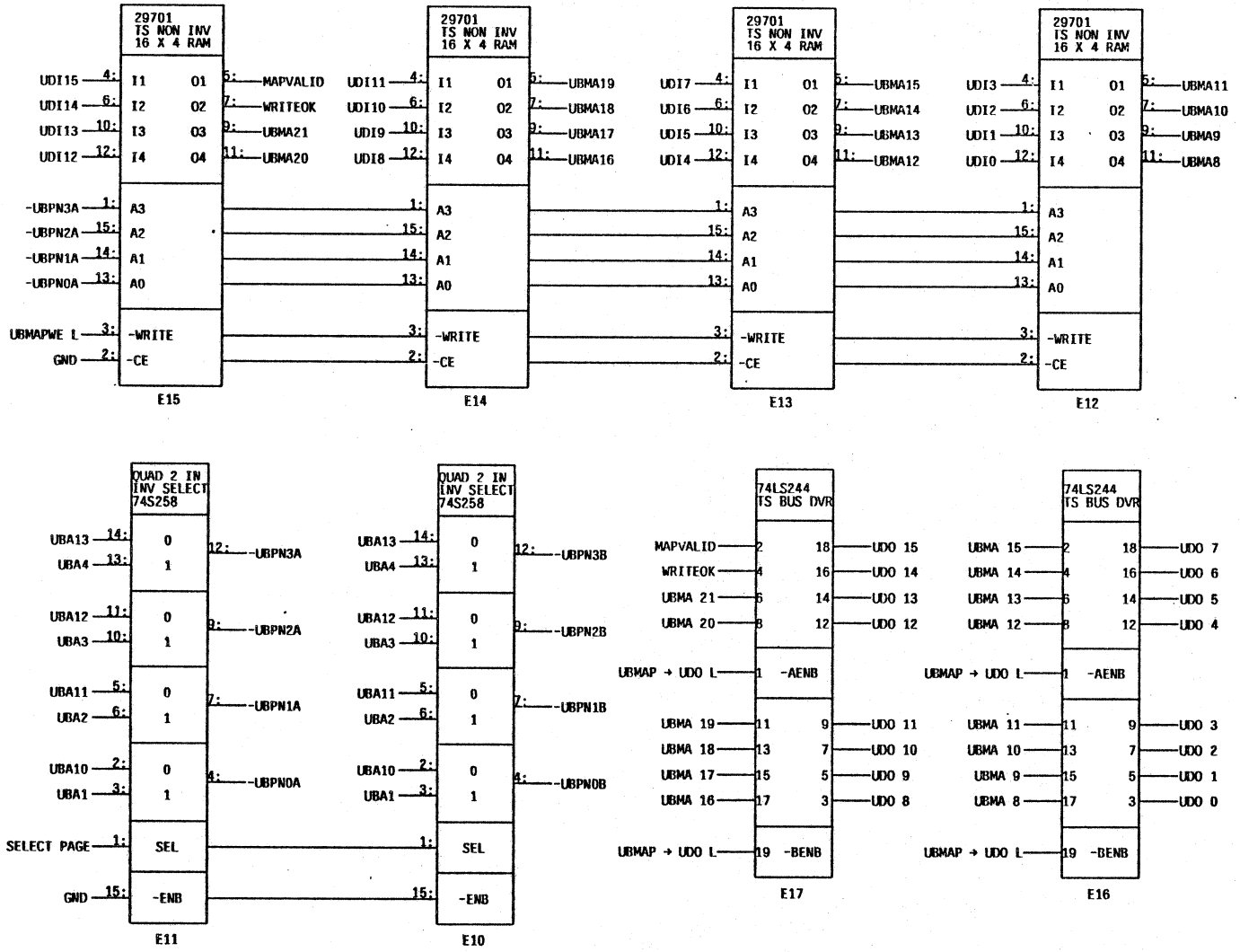

HEX INV
74S04
UBA 15
UBA 14
UBA 12
UBA 9
UBA 8
UBA 7
-UBA 15
-UBA 14
-UBA 12
-UBA 9
-UBA 8
-UBA 7
D11

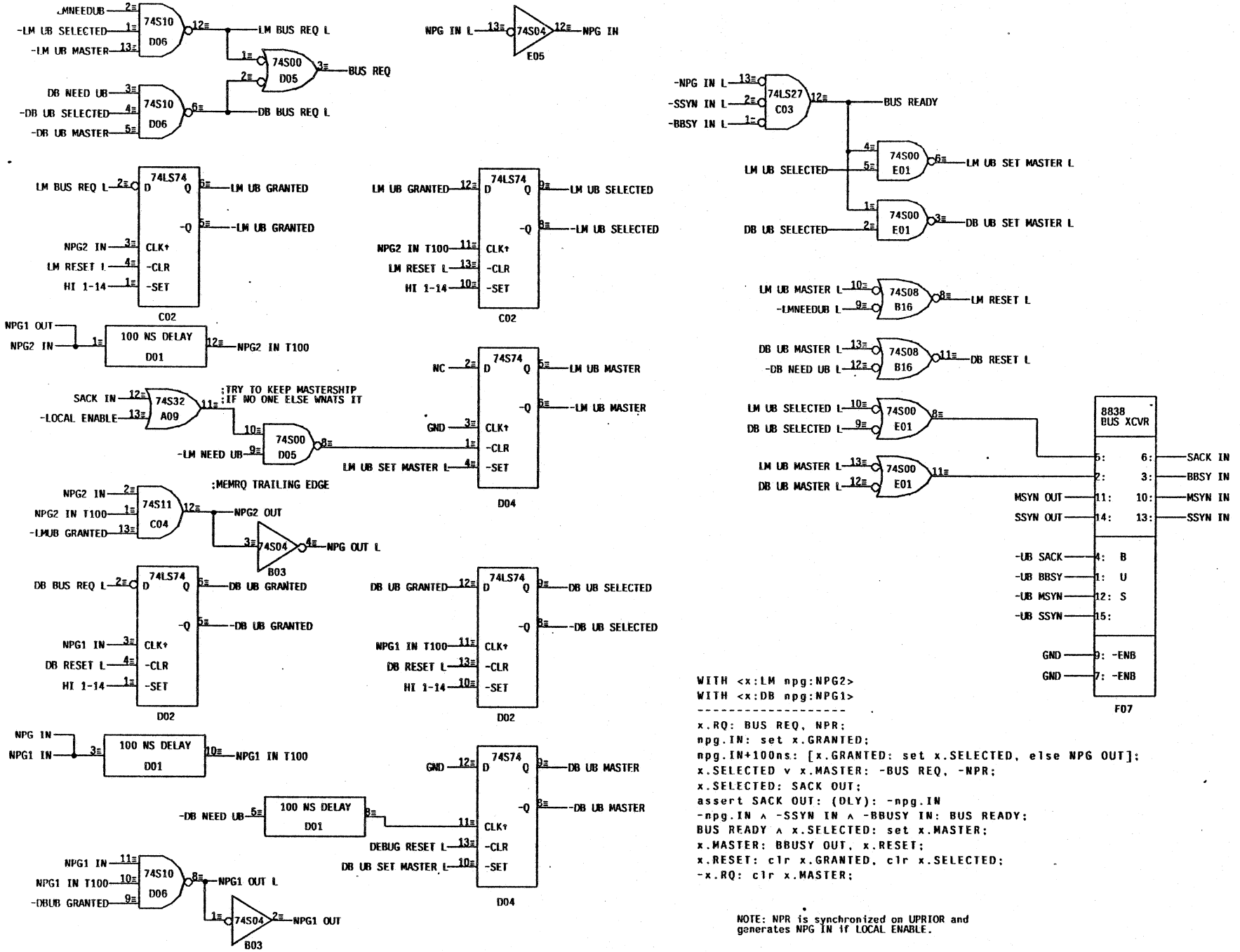

SELECT PAGE L
74S04
E05
SELECT PAGE
WRITE THROUGH ENB L
UBPN3A L
74S32
E04
74S32
B18
WRITE THROUGH L
UB WRITE BUFFER L
3-8 DECODE
74S138
-7
-6
-5
-4
-3
-2
-1
-0
NC
RESET ERR L
LOAD INT CTL2 REG L
LOAD INT CTL REG L
NC
UB ERR DRIVE L
NC
INTC DRIVE L
SEL2
SEL1
SEL0
UBWR
UBA 2
UBA 1
CE2
-CE1
-CE0
HI 1-14
SELECT INTERRUPT L
E03
4 WAY
DECODE
74S139
-3
-2
-1
-0
SELECT PAGE L
SELECT DEBUG L
SELECT INTERRUPT L
SELECT SPY L
SEL1
SEL0
-ENB
UBA 6
UBA 5
E07
;16 WORDS EACH
UBA17
UBA16
UBA15
UBA14
UBA13
-UBA12
UBA11
UBA10
-UBA9
-UBA8
-UBA7
MSYN IN
HI 1-14
766 000-
766 177
74S133
E08
UDI7
74S74
D
Q
-Q
CLK↑
-CLR
-SET
WRITE THROUGH ENB
WRITE THROUGH ENB
RESET ERR L
HI 1-14
B08
UBRD
UB REG WRITE PULSE
74S02
F02
WRITE THROUGH L
UB WR XBUS L
HI 1-14
74S11
C04
UB WRITE XBUS L
SELECT SPY L
SELECT INTERRUPT L
SELECT PAGE L
UB READ BUFFER L
UB WRITE BUFFER L
HI 1-14
HI 1-14
HI 1-14
HI 1-14
HI 1-14
HI 1-14
HI 1-14
HI 1-14
74S133
F05
DELAY
50NS INCS
TD250
INPUT
50NS
100NS
150NS
200NS
250NS
UB REG CYC T0
UB REG CYC T50
UB REG CYC T100
UB REG CYC T150
UB REG CYC T200
UB REG CYC T250
F04
4 WAY
DECODE
74S139
-3
-2
-1
-0
UB WR XBUS L
UB WRITE BUFFER L
UB READ BUFFER L
UB READ XBUS L
SEL1
SEL0
-ENB
C1 IN
UBA1
E07
140 000-
177 777
-UBA17 L
-UBA16 L
UBA15 L
UBA14 L
GND
74S260
E06
MSYN IN
74S00
F03
UB17-14=MAP
UB REG CYC T150
74S04
E05
UB REG CYC T150 L
UB REG CYC T50
-UB REG CYC T150
74S08
E02
UB REG WRITE PULSE
PAGES ARE 512 16-BIT WORDS LONG
THE FIRST 8 ARE ACCESSIBLE BY A PDP-11,
THE OTHER 8 ARE SHADOWED BY I/O ADDRESSES


8838
BUS XCVR
UDO15
UDO14
UDO13
UDO12
UDI15
UDI14
UDI13
UDI12
-UBD15
-UBD14
-UBD13
-UBD12
UBDRIVE L
GND
-ENB
F19
UDO11
UDO10
UDO9
UDO8
UDI11
UDI10
UDI9
UDI8
-UBD11
-UBD10
-UBD9
-UBD8
F18
UDO7
UDO6
UDO5
UDO4
UDI7
UDI6
UDI5
UDI4
-UBD7
-UBD6
-UBD5
-UBD4
F17
UDO3
UDO2
UDO1
UDO0
UDI3
UDI2
UDI1
UDI0
-UBD3
-UBD2
-UBD1
-UBD0
F16
74LS244
TS BUS DVR
UDI 15
UDI 14
UDI 13
UDI 12
UDO 15
UDO 14
UDO 13
UDO 12
UDI → UDO L
-AENB
UDI 11
UDI 10
UDI 9
UDI 8
UDO 11
UDO 10
UDO 9
UDO 8
-BENB
E19
UDI 7
UDI 6
UDI 5
UDI 4
UDO 7
UDO 6
UDO 5
UDO 4
UDI 3
UDI 2
UDI 1
UDI 0
UDO 3
UDO 2
UDO 1
UDO 0
E18

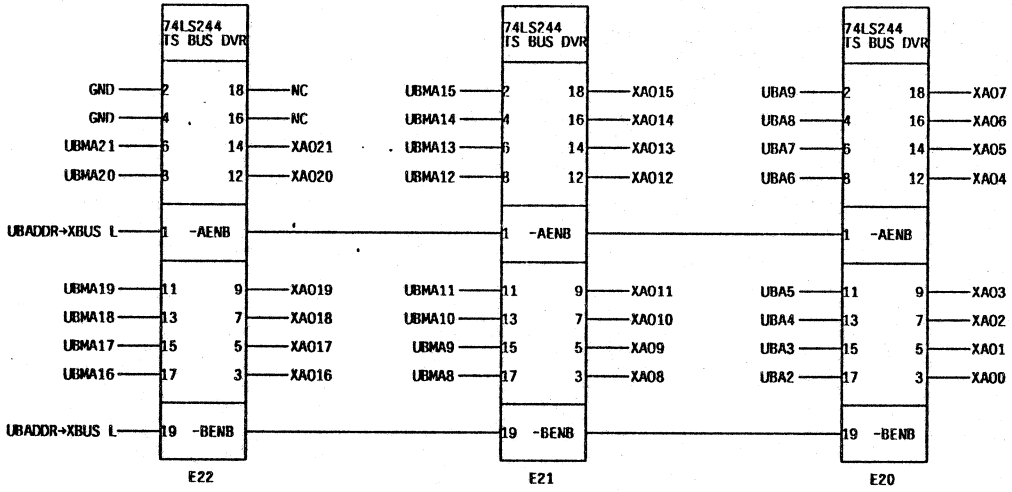


CBUS	UNIBUS INTERRUPT CONTROL	10-DEC-1980 11:34	AI: CADR1; UBINTC

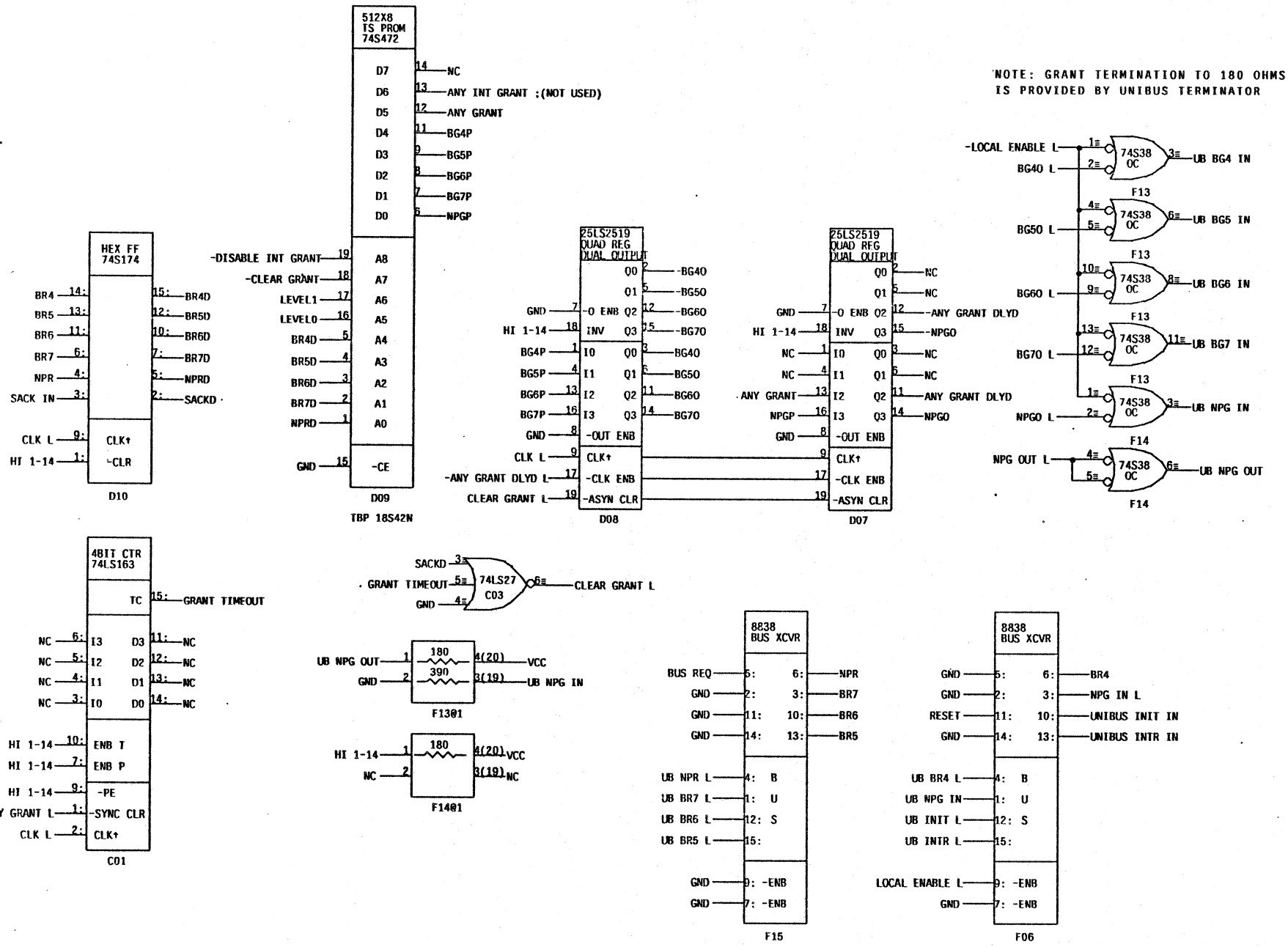

29701
TS NON INV
16 X 4 RAM
UDI15 4: I1 O1 5: MAPVALID
UDI14 6: I2 O2 7: WRITEOK
UDI13 10: I3 O3 9: UBMA21
UDI12 12: I4 O4 11: UBMA20
-UBPN3A 1: A3
-UBPN2A 15: A2
-UBPN1A 14: A1
-UBPNOA 13: A0
UBMAPWE L 3: -WRITE
GND 2: -CE
E15
UDI11 4: I1 O1 5: UBMA19
UDI10 6: I2 O2 7: UBMA18
UDI9 10: I3 O3 9: UBMA17
UDI8 12: I4 O4 11: UBMA16
E14
UDI7 4: I1 O1 5: UBMA15
UDI6 6: I2 O2 7: UBMA14
UDI5 10: I3 O3 9: UBMA13
UDI4 12: I4 O4 11: UBMA12
E13
UDI3 4: I1 O1 5: UBMA11
UDI2 6: I2 O2 7: UBMA10
UDI1 10: I3 O3 9: UBMA9
UDI0 12: I4 O4 11: UBMA8
E12
QUAD 2 IN
INV SELECT
74S258
UBA13 14: 0
UBA4 13: 1
12: -UBPN3A
UBA12 11: 0
UBA3 10: 1
9: -UBPN2A
UBA11 5: 0
UBA2 6: 1
7: -UBPN1A
UBA10 2: 0
UBA1 3: 1
4: -UBPNOA
SELECT PAGE 1: SEL
GND 15: -ENB
E11
UBA13 14: 0
UBA4 13: 1
12: -UBPN3B
UBA12 11: 0
UBA3 10: 1
9: -UBPN2B
UBA11 5: 0
UBA2 6: 1
7: -UBPN1B
UBA10 2: 0
UBA1 3: 1
4: -UBPN0B
1: SEL
15: -ENB
E10
74LS244
TS BUS DVR
MAPVALID 2 18 UDO 15
WRITEOK 4 16 UDO 14
UBMA 21 6 14 UDO 13
UBMA 20 8 12 UDO 12
UBMAP → UDO L 1 -AENB
UBMA 19 11 9 UDO 11
UBMA 18 13 7 UDO 10
UBMA 17 15 5 UDO 9
UBMA 16 17 3 UDO 8
UBMAP → UDO L 19 -BENB
E17
UBMA 15 2 18 UDO 7
UBMA 14 4 16 UDO 6
UBMA 13 6 14 UDO 5
UBMA 12 8 12 UDO 4
UBMAP → UDO L 1 -AENB
UBMA 11 11 9 UDO 3
UBMA 10 13 7 UDO 2
UBMA 9 15 5 UDO 1
UBMA 8 17 3 UDO 0
UBMAP → UDO L 19 -BENB
E16



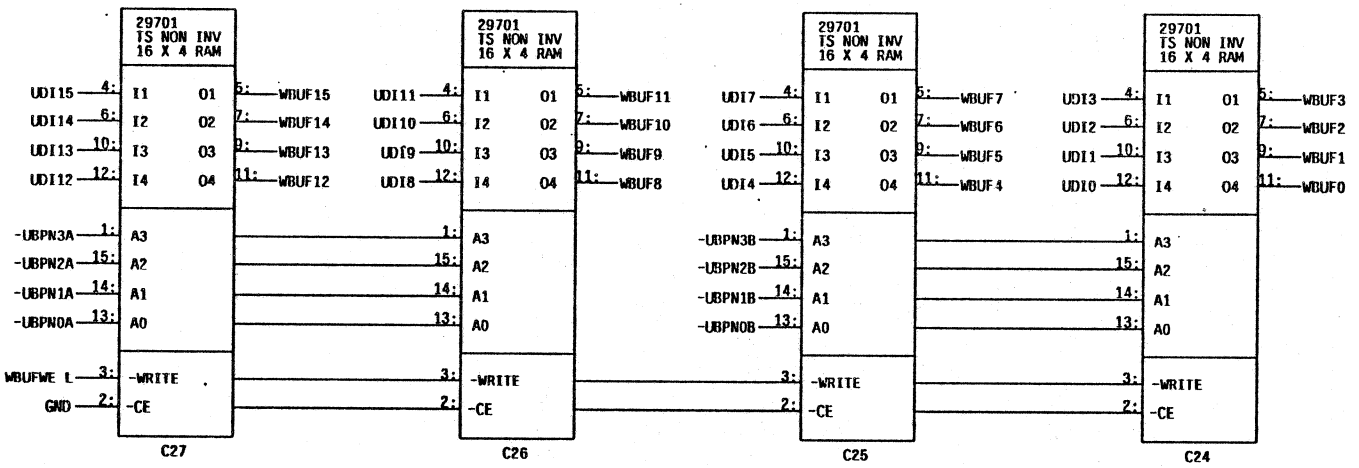
CBUS	UNIBUS MASTERSHIP	10-DEC-1980 13:21	AI: CADR1; UBMAST



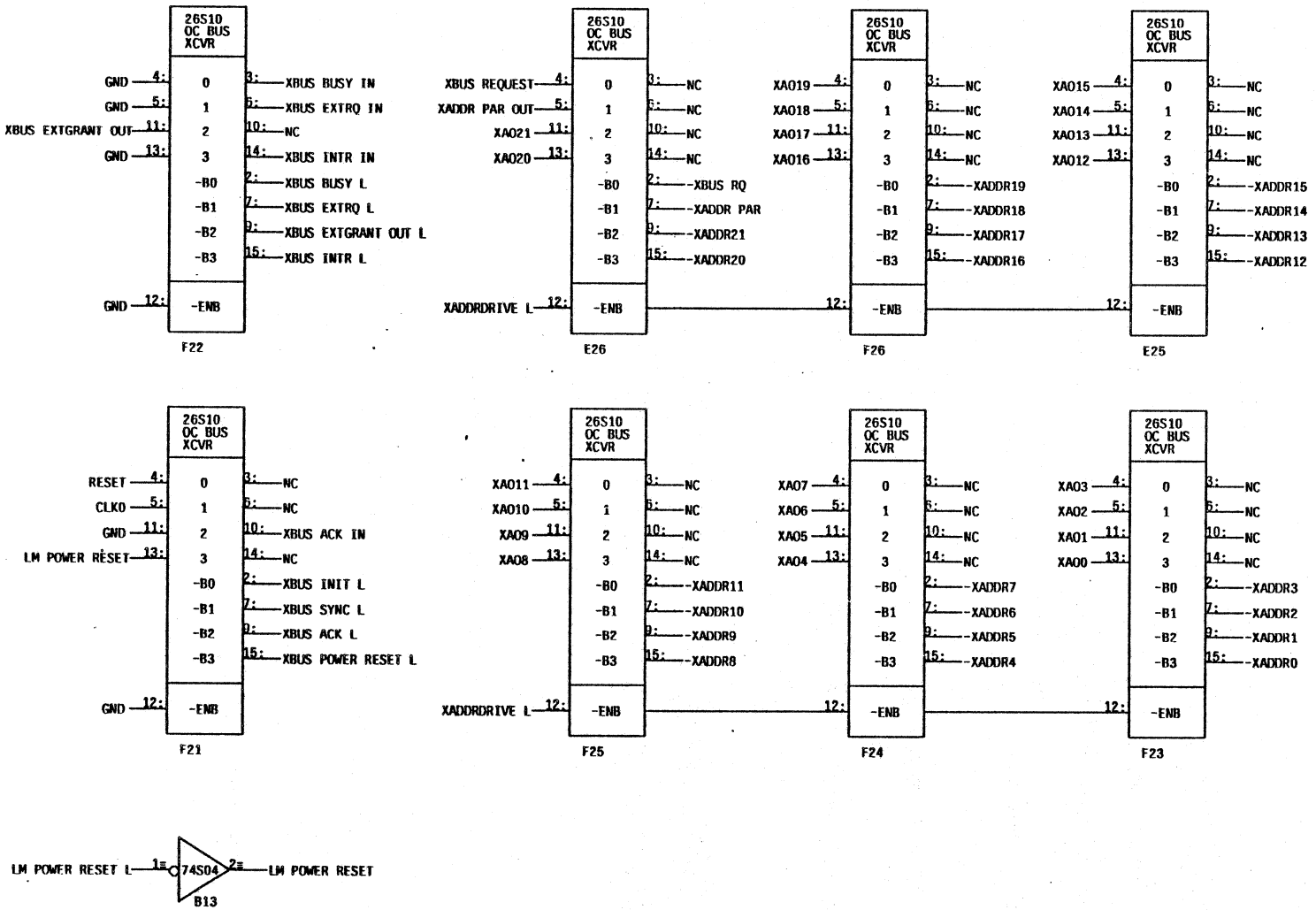
CBUS	UNIBUS ADDRESS TO XBUS	10-DEC-1980 11:36	AI: CADR1; UBXA

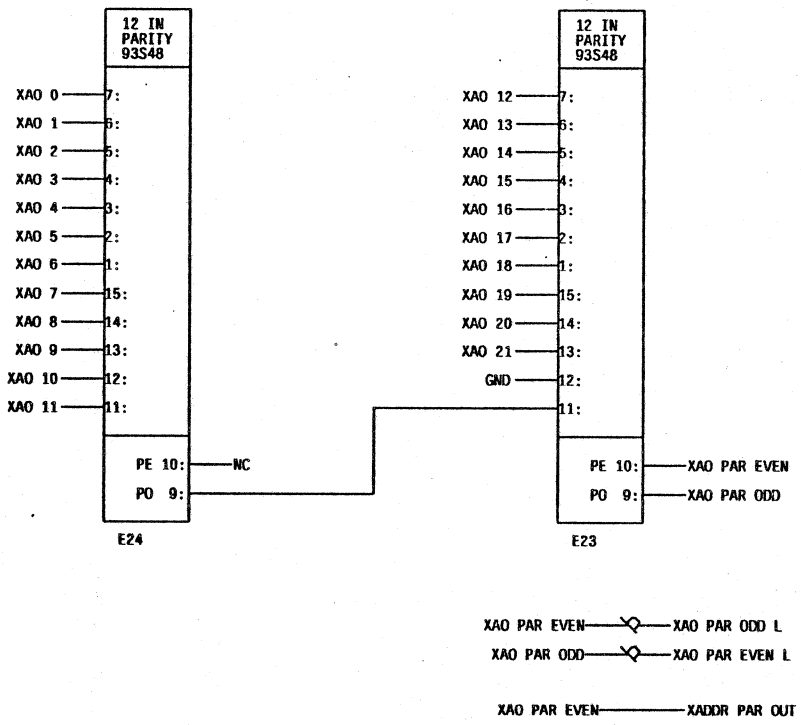


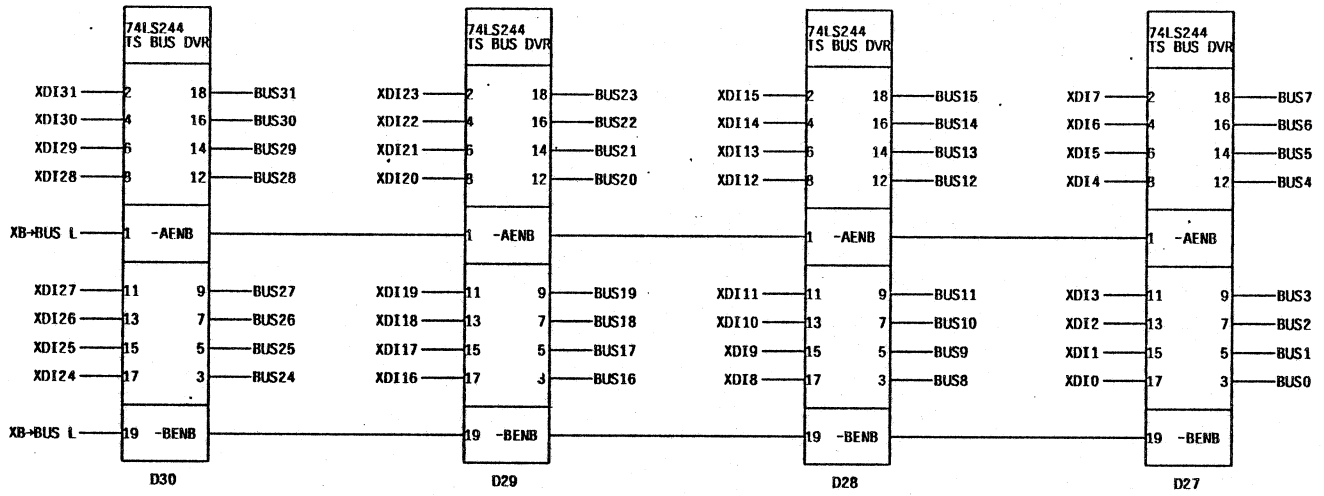
CBUS	UNIBUS BUS GRANT	10-DEC-1980 13:15	AI: CADR1; UPRIOR



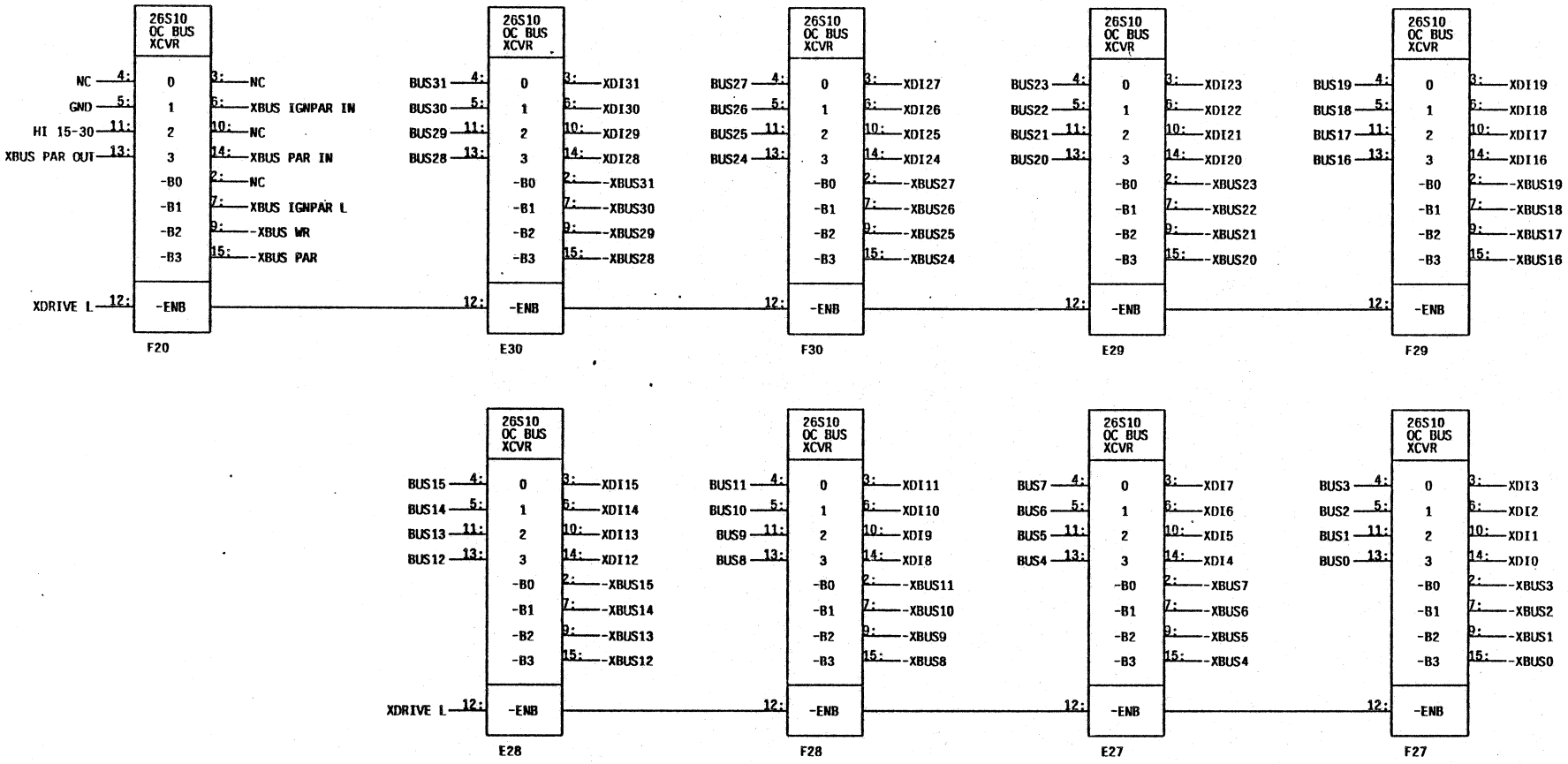
CBUS	WRITE BUFFER	10-DEC-1980 11:37	AI: CADR1; WBUF


26S10
OC BUS
XCVR
GND 4:
GND 5:
XBUS EXTGRANT OUT 11:
GND 13:
0
1
2
3
-B0
-B1
-B2
-B3
-ENB
GND 12:
3: XBUS BUSY IN
6: XBUS EXTRQ IN
10: NC
14: XBUS INTR IN
2: XBUS BUSY L
7: XBUS EXTRQ L
9: XBUS EXTGRANT OUT L
15: XBUS INTR L
F22
XBUS REQUEST 4:
XADDR PAR OUT 5:
XA021 11:
XA020 13:
XADDRDRIVE L 12:
3: NC
6: NC
10: NC
14: NC
2: -XBUS RQ
7: -XADDR PAR
9: -XADDR21
15: -XADDR20
E26
XA019 4:
XA018 5:
XA017 11:
XA016 13:
12:
3: NC
6: NC
10: NC
14: NC
2: -XADDR19
7: -XADDR18
9: -XADDR17
15: -XADDR16
F26
XA015 4:
XA014 5:
XA013 11:
XA012 13:
12:
3: NC
6: NC
10: NC
14: NC
2: -XADDR15
7: -XADDR14
9: -XADDR13
15: -XADDR12
E25
RESET 4:
CLK0 5:
GND 11:
LM POWER RESET 13:
3: NC
6: NC
10: XBUS ACK IN
14: NC
2: XBUS INIT L
7: XBUS SYNC L
9: XBUS ACK L
15: XBUS POWER RESET L
F21
XA011 4:
XA010 5:
XA09 11:
XA08 13:
3: NC
6: NC
10: NC
14: NC
2: -XADDR11
7: -XADDR10
9: -XADDR9
15: -XADDR8
F25
XA07 4:
XA06 5:
XA05 11:
XA04 13:
3: NC
6: NC
10: NC
14: NC
2: -XADDR7
7: -XADDR6
9: -XADDR5
15: -XADDR4
F24
XA03 4:
XA02 5:
XA01 11:
XA00 13:
3: NC
6: NC
10: NC
14: NC
2: -XADDR3
7: -XADDR2
9: -XADDR1
15: -XADDR0
F23
LM POWER RESET L 1
74S04
2 LM POWER RESET
B13


12 IN
PARITY
93S48
XAO 0
XAO 1
XAO 2
XAO 3
XAO 4
XAO 5
XAO 6
XAO 7
XAO 8
XAO 9
XAO 10
XAO 11
7:
6:
5:
4:
3:
2:
1:
15:
14:
13:
12:
11:
PE 10:
PO 9:
NC
E24
12 IN
PARITY
93S48
XAO 12
XAO 13
XAO 14
XAO 15
XAO 16
XAO 17
XAO 18
XAO 19
XAO 20
XAO 21
GND
7:
6:
5:
4:
3:
2:
1:
15:
14:
13:
12:
11:
PE 10:
PO 9:
XAO PAR EVEN
XAO PAR ODD
E23
XAO PAR EVEN
XAO PAR ODD L
XAO PAR ODD
XAO PAR EVEN L
XAO PAR EVEN
XADDR PAR OUT



CBUS	XBUS TO BUS	10-DEC-1980 16:06	AI: CADR1; XBD



CBUS	XBUS DATA TRANSCEIVERS	10-DEC-1980 11:50	AI: CADR1; XD

LISPM I/O BOARD CADRIO;IOB UML 18-DEC-80 1049
******* DIP MAP *******

75118 IOBKBD x F30	75118 IOBKBD x E30	74LS374 IOBKBD x D30	74LS374 IOBKBD x C30	74LS74 IOBKBD xx B30	74LS164 IOBKBD x A30
74LS74 IOBSER xx F29	74LS244 IOBSER x E29	74LS244 IOBCSR x D29	74LS374 IOBKBD x C29	74LS569 IOBMS2 x B29	74LS164 IOBKBD x A29
74S374 CLK60H x F28	74S374 CLK60H x E28	74LS244 CLKTOD x D28	74LS10 IOBKBD xxx C28	74LS569 IOBMS2 x B28	74LS164 IOBKBD x A28
74LS244 CLK60H x F27	74LS244 CLK60H x E27	74LS175 IOBCSR x D27	74LS74 IOBCSR xx C27	74LS569 IOBMS2 x B27	74LS14 IOBKBD oxxxxx A27
74LS374 CLKTOD x F26	74LS374 CLKTOD x E26	74LS08 IOBKBD xxxx D26	74LS109 IOBKBD xx C26	74LS569 IOBMS2 x B26	74LS86 IOBMS2 xxxo A26
74LS374 CLKTOD x F25	74LS374 CLKTOD x E25	74LS32 IOBCLK xxxx D25	74 37 IOBCLK xxoo C25	74LS569 IOBMS2 x B25	74LS14 IOBMSE xxxxxx A25
74 393 CLKTOD xx F24	74 393 CLKTOD xx E24	74LS163 IOBCLK x D24	74LS244 IOBMS2 x C24	74LS569 IOBMS2 x B24	74LS374 IOBMSE x A24
74 393 CLKTOD xx F23	74 393 CLKTOD xx E23	74 393 CLKTOD xx D23	74LS74 IOBCLK xx C23	74LS86 IOBMS2 xxxx B23	74LS374 IOBMSE x A23
74LS193 CLKTIM x F22	74LS193 CLKTIM x E22	74 393 CLKTOD xx D22	74LS138 IOBCLK x C22	74LS244 CLKTOD x B22	74LS374 IOBMSE x A22
74LS193 CLKTIM x F21	74LS193 CLKTIM x E21	74LS74 IOBSER xx D21	74S163 CLKTIM x C21	74LS138 CLK60H x B21	25LS2521 IOBMSE x A21
DM8838 IOBXCV x F20	74LS138 IOBADR x E20	74LS14 CLK60H xxxooo D20	DUMMY20 CLK60H x C20	74LS174 LMUCON x B20	25LS2521 IOBCSR x A20

LISPM I/O BOARD CADRIO;IOB UML 18-DEC-80 1050
******* DIP MAP *******

DM8838 IOBXCV x F19	74S133 IOBADR x E19	74LS244 LMDATP x D19	74S133 LMUCON, x C19	74LS164 LMTURN x B19	74LS193 LMTURN x A19
DM8838 IOBXCV x F18	74LS244 LMDATP x E18	74LS244 LMDATP x D18	74LS138 LMUCON x C18	74LS164 LMTURN x B18	74LS193 LMTURN x A18
DM8838 IOBXCV x F17	74LS244 LMDATP x E17	74LS244 LMDATP x D17	74LS244 LMDATP x C17	MC1488L IOBSER x B17	74LS193 LMTURN x A17
74S38 IOBINT xxxx F16	-------- -------- E16	74LS244 LMDATP x D16	74LS244 LMDATP x C16	MC1489L IOBSER xxxx B16	74LS161 LMTURN x A16
74S38 IOBCSR xxxx F15	-------- -------- E15	74S32 LMRCTL xxxx D15	74S00 LMTBFC xxxx C15	74S74 LMMODU xx B15	EXAR-CL IOBSER x A15
74S175 IOBINT x F14	74S260 IOBINT xx E14	74S112 LMTURN xx D14	74S00 LMTBFC xxxx C14	74S04 LMTBFC xxxxxx B14	-------- -------- A14
TD100 IOBINT x F13	74S04 IOBADR xxxxxx E13	74S112 LMTBFC xx D13	25LS193 LMTBUF x C13	74 165 LMTBUF x B13	-------- -------- A13
74S00 IOBINT xxxx F12	74LS00 IOBINT xxxx E12	DUMMY LMMYNM x D12	25LS193 LMTBUF x C12	74 165 LMTBUF x B12	2651 IOBSER x A12
74LS10 IOBINT xxx F11	74LS02 IOBSER xxxx E11	P SIP100 LMMYNM x D11	25LS193 LMTBUF x C11	TD100 LMDETC x B11	TD25 LMDETC x A11
DM8838 IOBINT x F10	74S08 LMRCTL xxxx E10	DUMMY LMMYNM x D10	2147 LMTBUF x C10	74S08 LMTBFC xxxx B10	74S288 LMMODU x A10
DM8136 IOBADR x F09	TD250 IOBADR x E09	74 279 LMRCTL xxox D09	9401 LMTBUF x C09	74S04 LMDETC xxxxxx B09	74S112 LMMODU xx A09

LISPM I/O BOARD

******* CADRIO;IOB UML 18-DEC-80 1052 ******* DIP MAP *******

DM8136 IOBADR x	74S74 IOBADR xx	74S74 LMRCTL xx	74S51 LMTBUF xx	74S02 LMTBFC xxxx	74S374 LMMODU x
F08	E08	D08	C08	B08	A08
DM8837A IOBXCV x	74S37 IOBADR xxxx	74S00 LMRBUF xxxx	9401 LMRBUF x	74LS164 LMRBUF x	74LS164 LMRBUF x
F07	E07	D07	C07	B07	A07
DM8837A IOBXCV x	74S251 LMMYNM x	25LS193 LMRBUF x	74S74 LMRCLK xx	74LS161 LMRCLK x	74S112 LMTCLK xx
F06	E06	D06	C06	B06	A06
74S251 LMMYNM x	74S51 LMRBUF xx	25LS193 LMRBUF x	74S37 LMRBUF xxxx	74LS161 LMRCLK x	EXAR-CL LMTCLK x
F05	E05	D05	C05	B05	A05
74LS161 LMMYNM x	74S00 LMUCON xxxx	25LS193 LMRBUF x	2147 LMRBUF x	TD100 LMDETC x	26S02 LMRCTL xx
F04	E04	D04	C04	B04	A04
74LS161 LMMYNM x	74S00 LMRCTL xxxx	74S04 LMTCLK xxxxxx	TD250 LMRCLK x	74S163 LMTCLK x	DUMMY LMLNDR x
F03	E03	D03	C03	B03	A03
74LS161 LMMYNM x	74S158 LMLNDR x	74S10 LMMODU xxx	74S08 LMRBUF xxxx	74S74 LMDETC xx	26LS31 LMLNDR xxxx
F02	E02	D02	C02	B02	A02
74S11 LMMODU oxo	74S174 LMMYNM x	74S287 LMMYNM x	74S74 LMDETC xx	74S04 LMRCLK xxxxxx	26LS33 LMLNDR xxxx
F01	E01	D01	C01	B01	A01

LISPM I/O BOARD CADRIO;IO8 UML 18-DEC-80 1053
******* EDGE CONNECTIONS Flags: (# Output, @ Terminator, ---- Dedicated ground, ++++ Dedicated power) *******

-A-	-B-	-C-	-D-
A1	A1	A1 NPG.IN*	A1
A2 ++++++++++++++++++++++++	A2 ++++++++++++++++++++++++	A2 +5.0V++++++++++++++++++++	A2 +5.0V++++++++++++++++++++
B1	B1	B1 NPG.IN*	B1
B2	B2	B2 -5.0V	B2 -5.0V
C1	C1	C1	C1
C2 ----------	C2 ----------	C2 GND---------- #	C2 GND---------- #
D1	D1	D1	D1
D2	D2	D2 -D15* #	D2 -BR7*
E1	E1	E1	E1
E2	E2	E2 -D14* #	E2 -BR6*
F1	F1	F1 NC #	F1 NC #
F2	F2	F2 -D13* #	F2 -BR* #
H1	H1	H1 -D11* #	H1
H2	H2	H2 -D12* #	H2 -BR4*
J1	J1	J1	J1
J2	J2	J2 -D10* #	J2
K1	K1	K1	K1
K2	K2	K2 -D9* #	K2 BG7.IN*
L1	L1	L1	L1 -INIT*
L2	L2	L2 -D8* #	L2 BG7.IN*
M1	M1	M1	M1
M2	M2	M2 -D7* #	M2 BG6.IN*
N1	N1	N1 NC #	N1 NC #
N2	N2	N2 -D4* #	N2 BG6.IN*
P1	P1	P1 -BOOT* #	P1 +12.0V
P2	P2	P2 -D5* #	P2 BG.IN*
R1	R1	R1	R1
R2	R2	R2 -D1* #	R2 BG.OUT* #
S1	S1	S1	S1
S2	S2	S2 -D0* #	S2 BG4.IN*
T1 ----------	T1 ----------	T1 GND---------- #	T1 GND---------- #
T2	T2	T2 -D3* #	T2 BG4.IN*
U1	U1	U1	U1
U2	U2	U2 -D2* #	U2
V1	V1	V1	V1
V2	V2	V2 -D6* #	V2

LISPM I/O BOARD CADRIO;IOB UML 18-DEC-80 1053
******* EDGE CONNECTIONS Flags: (# Output, @ Terminator, ---- Dedicated ground, ++++ Dedicated power) *******

-E-	-F-	-J01-	-J02-
A1 GND #	A1	01 GND #	01
A2 +5.0V++++++++++++++++++	A2 +5.0V++++++++++++++++++	02 INTERFERE+	02
B1	B1	03 INTERFERE-	03
B2 -5.0V	B2 -5.0V	04 GND #	04
C1 -A12*	C1	05 RCVR.DATA+	05
C2 GND---------- #	C2 GND---------- #	06 RCVR.DATA-	06
D1 -A17*	D1 -BBSY* #	07 GND #	07
D2 -A15*	D2	08 TRANS.DATA+ #	08
E1 -MSYN*	E1 H SYNC	09 TRANS.DATA- #	09
E2 -A16*	E2 V SYNC	10 GND #	10
F1 -A2*	F1 NC #	11	11
F2 -C1*	F2	12	12
H1 -A1*	H1	13	13
H2 -A0*	H2	14	14
J1 -SSYN* #	J1 -NPR*	15	15
J2 -C0*	J2	16	16
K1 -A14*	K1	17	17
K2 -A13*	K2	18	18
L1 -A11*	L1	19	19
L2	L2	20	20
M1	M1 -INTR* #	21	21
M2 GND #	M2	22	22
N1 GND #	N1 NC #	23	23
N2 -A8*	N2	24	24
P1 -A10*	P1	25	25
P2 -A7*	P2	26 GND---------- #	26 ----------
R1 -A9*	R1	27 GND---------- #	27 ----------
R2	R2	28 GND---------- #	28 ----------
S1	S1	29 GND---------- #	29 ----------
S2 GND #	S2	30 GND---------- #	30 ----------
T1 GND---------- #	T1 GND---------- #	31 GND---------- #	31 ----------
T2	T2 -SACK* #	32 GND---------- #	32 ----------
U1 -A6*	U1	33 GND---------- #	33 ----------
U2 -A4*	U2	34 GND---------- #	34 ----------
V1 -A5*	V1	35 GND---------- #	35 ----------
V2 -A3*	V2 POWER LINE ^↓	36 ----------	36 ----------
		37 ----------	37 ----------
		38 ----------	38 ----------
		39 ----------	39 ----------
		40 ----------	40 ----------
		41 ----------	41 ----------
		42 ----------	42 ----------
		43 ----------	43 ----------
		44 ----------	44 ----------
		45 ----------	45 ----------
		46 ----------	46 ----------
		47 ----------	47 ----------
		48 ----------	48 ----------
		49 ----------	49 ----------
		50 ----------	50 ----------

LISPM I/O BOARD CADRIO;IOB UML 18-DEC-80 1053
******* EDGE CONNECTIONS Flags: (# Output, @ Terminator, ---- Dedicated ground, ++++ Dedicated power) *******

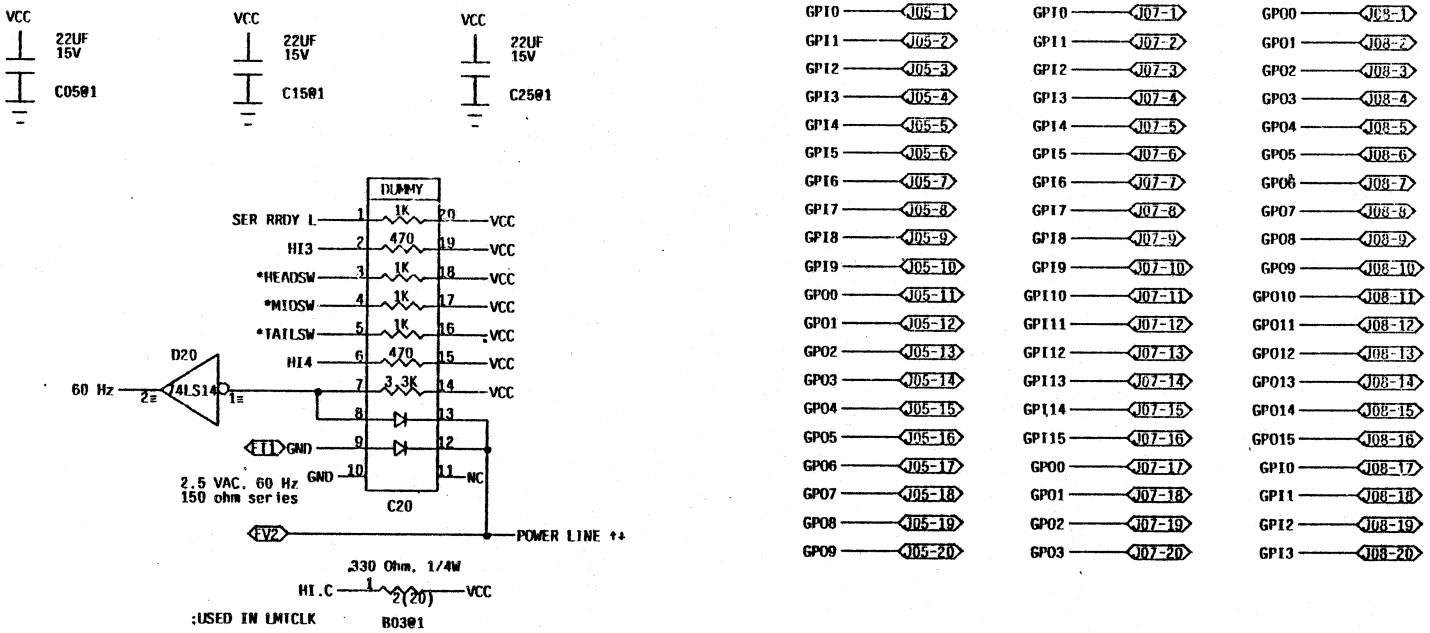
-J03-	-J04-	-J05-	-J06-
01 H SYNC	01	01 GPI0	01
02 V SYNC	02	02 GPI1	02
03 FH1	03	03 GPI2	03
04	04	04 GPI3	04
05	05	05 GPI4	05
06	06	06 GPI5	06
07	07	07 GPI6	07
08	08	08 GPI7	08
09	09	09 GPI8	09
10 AUDIO+ *	10	10 GPI9	10
11 KBDCLK+ *	11	11 GPO0 #	11
12 KBDIN+ *	12	12 GPO1 #	12
13 *VERB	13	13 GPO2 #	13
14 *VERA	14	14 GPO3 #	14
15 *HORB	15	15 GPO4 #	15
16 *HORA	16	16 GPO5 #	16
17 *HEADSW	17	17 GPO6 #	17
18 *MIDSW	18	18 GPO7 #	18
19 *TAILSW	19	19 GPO8 #	19
20	20	20 GPO9 #	20
21 ----------	21 ----------	21	21
22 ----------	22 ----------	22	22
23 FH2----------	23 ----------	23	23
24 ----------	24 ----------	24	24
25 ----------	25 ----------	25	25
26 ----------	26 ----------	26 ----------	26 ----------
27 ----------	27 ----------	27 ----------	27 ----------
28 ----------	28 ----------	28 ----------	28 ----------
29 ----------	29 ----------	29 ----------	29 ----------
30 AUDIO- *----------	30 ----------	30 ----------	30 ----------
31 KBDCLK- *----------	31 ----------	31 ----------	31 ----------
32 KBDIN- *----------	32 ----------	32 ----------	32 ----------
33 GND---------- #	33 ----------	33 ----------	33 ----------
34 GND---------- #	34 ----------	34 ----------	34 ----------
35 GND---------- #	35 ----------	35 ----------	35 ----------
36 GND---------- #	36 ----------	36 ----------	36 ----------
37 GND---------- #	37 ----------	37 ----------	37 ----------
38 GND---------- #	38 ----------	38 ----------	38 ----------
39 GND---------- #	39 ----------	39 ----------	39 ----------
40 ----------	40 ----------	40 ----------	40 ----------
		41 ----------	41 ----------
		42 ----------	42 ----------
		43 ----------	43 ----------
		44 ----------	44 ----------
		45 ----------	45 ----------
		46 ----------	46 ----------
		47 ----------	47 ----------
		48 ----------	48 ----------
		49 ----------	49 ----------
		50 ----------	50 ----------

LISPM I/O BOARD CADRIO;IOB UML 18-DEC-80 1054
******* EDGE CONNECTIONS Flags: (# Output, @ Terminator, ---- Dedicated ground, ++++ Dedicated power) *******

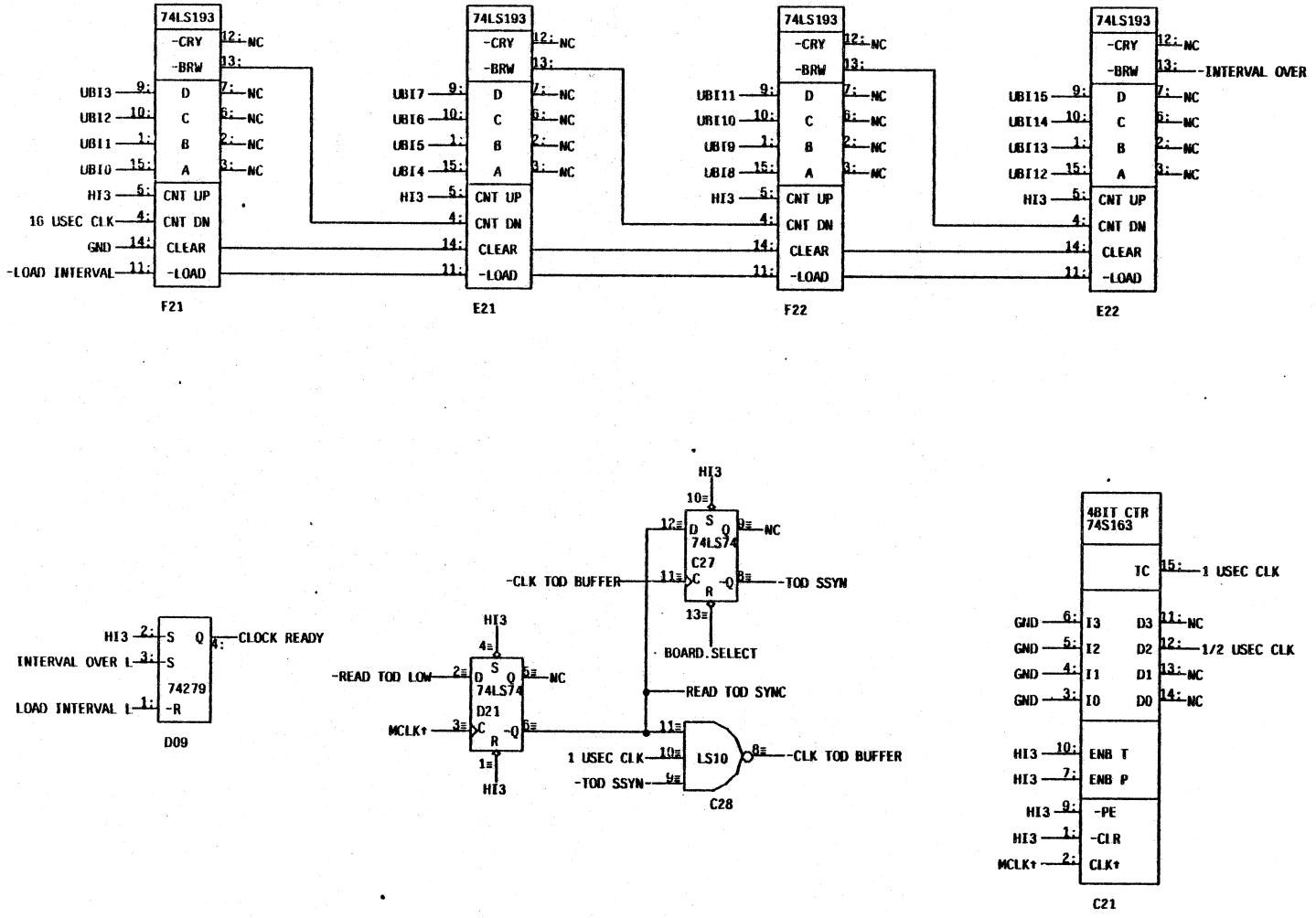
-J07-	-J08-	-J09-	-J10-
01 GPI0	01 GPO0	#|01 EIA DATA OUT	#|01
02 GPI1	02 GPO1	#|02 EIA DATA IN	|02
03 GPI2	03 GPO2	#|03 EIA RTS OUT	#|03
04 GPI3	04 GPO3	#|04 EIA CTS IN	|04
05 GPI4	05 GPO4	#|05 EIA DSR IN	|05
06 GPI5	06 GPO5	#|06 EIA DCD IN	|06
07 GPI6	07 GPO6	#|07 EIA DTR OUT	#|07
08 GPI7	08 GPO7	#|08	|08
09 GPI8	09 GPO8	#|09	|09
10 GPI9	10 GPO9	#|10	|10
11 GPI10	11 GPO10	#|11	|11
12 GPI11	12 GPO11	#|12	|12
13 GPI12	13 GPO12	#|13	|13
14 GPI13	14 GPO13	#|14	|14
15 GPI14	15 GPO14	#|15	|15
16 GPI15	16 GPO15	#|16	|16
17 GPO0 #	17 GPI0	|17	|17
18 GPO1 #	18 GPI1	|18	|18
19 GPO2 #	19 GPI2	|19	|19
20 GPO3 #	20 GPI3	|20	|20
21 ----------	21 ----------	|21	|21
22 ----------	22 ----------	|22	|22
23 ----------	23 ----------	|23	|23
24 ----------	24 ----------	|24	|24
25 ----------	25 ----------	|25	|25
26 ----------	26 ----------	|26 GND---------- #	|26 ----------
27 ----------	27 ----------	|27 GND---------- #	|27 ----------
28 ----------	28 ----------	|28 ----------	|28 ----------
29 ----------	29 ----------	|29 ----------	|29 ----------
30 ----------	30 ----------	|30 ----------	|30 ----------
31 ----------	31 ----------	|31 ----------	|31 ----------
32 ----------	32 ----------	|32 ----------	|32 ----------
33 ----------	33 ----------	|33 ----------	|33 ----------
34 ----------	34 ----------	|34 ----------	|34 ----------
35 ----------	35 ----------	|35 ----------	|35 ----------
36 ----------	36 ----------	|36 ----------	|36 ----------
37 ----------	37 ----------	|37 ----------	|37 ----------
38 ----------	38 ----------	|38 ----------	|38 ----------
39 ----------	39 ----------	|39 ----------	|39 ----------
40 ----------	40 ----------	|40 ----------	|40 ----------
		|41 ----------	|41 ----------
		|42 ----------	|42 ----------
		|43 ----------	|43 ----------
		|44 ----------	|44 ----------
		|45 ----------	|45 ----------
		|46 ----------	|46 ----------
		|47 ----------	|47 ----------
		|48 ----------	|48 ----------
		|49 ----------	|49 ----------
		|50 ----------	|50 ----------

LISPM I/O BOARD CADRIO;IOB UML 18-DEC-80 1054
******* EDGE CONNECTIONS Flags: (# Output, @ Terminator, ---- Dedicated ground, ++++ Dedicated power) *******

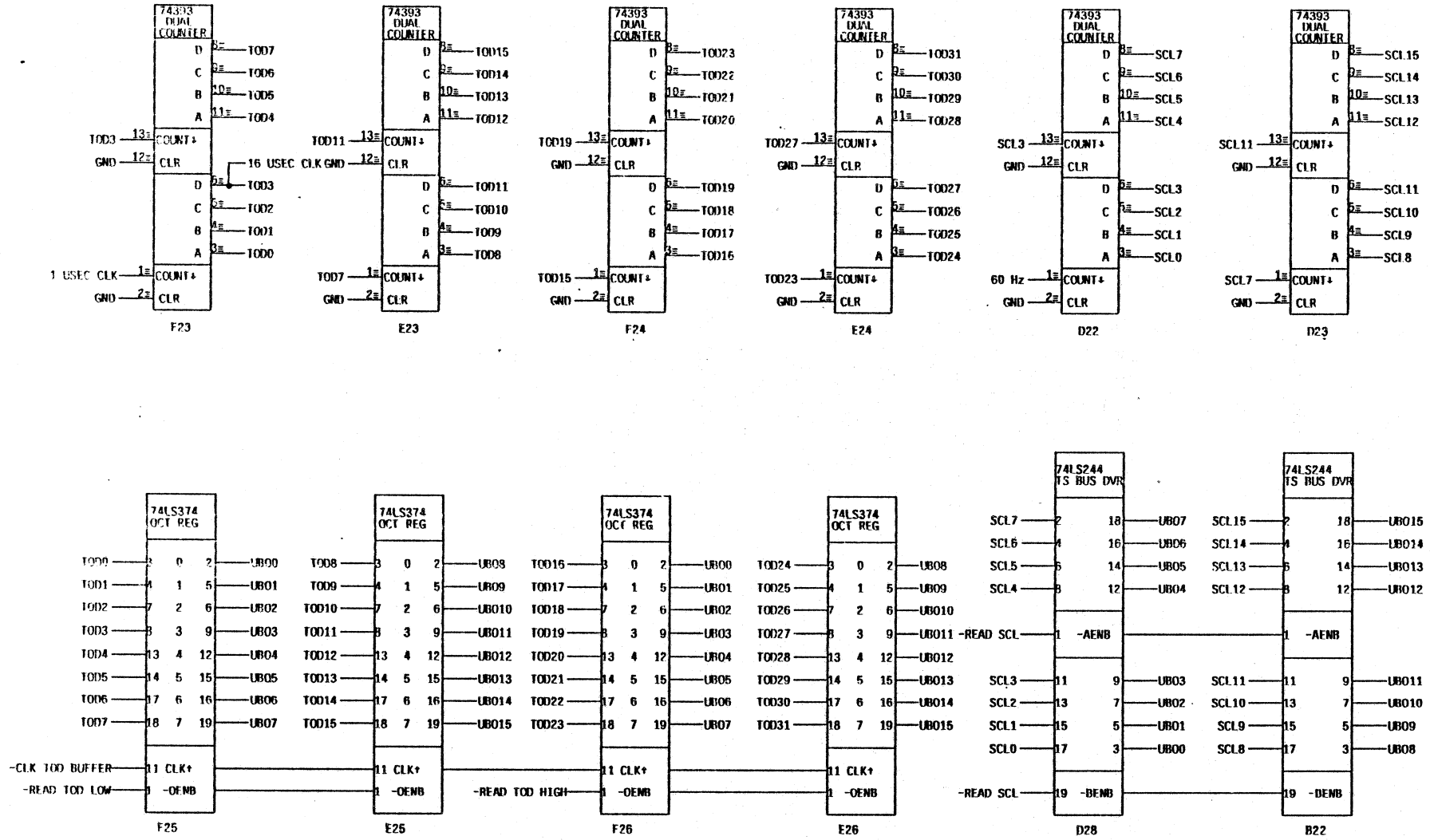
-J11-		-J12-	
01	02	01	02
03	04	03	04
05	06	05	06
07	08	07	08
09	10	09	10
11	12	11	12
13	14	13	14
15	16	15	16
17	18	17	18
19	20	19	20
21 ----------	22 ----------	21 ----------	22 ----------
23 ----------	24 ----------	23 ----------	24 ----------
25 ----------	26 ----------	25 ----------	26 ----------
27 ----------	28 ----------	27 ----------	28 ----------
29 ----------	30 ----------	29 ----------	30 ----------
31 ----------	32 ----------	31 ----------	32 ----------
33 ----------	34 ----------	33 ----------	34 ----------
35 ----------	36 ----------	35 ----------	36 ----------
37 ----------	38 ----------	37 ----------	38 ----------
39 ----------	40 ----------	39 ----------	40 ----------



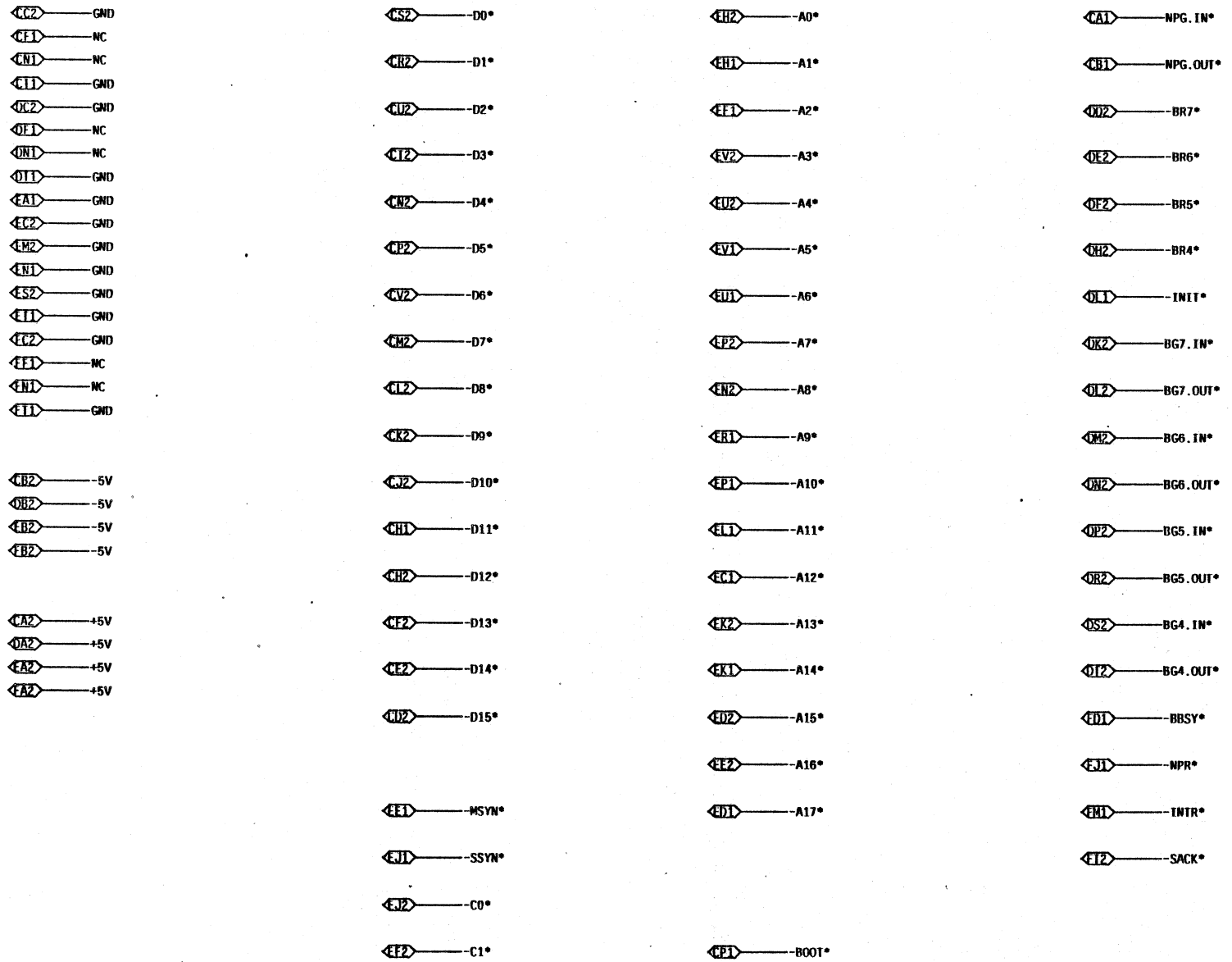
I/O BOARD	60 CYCLE CLOCK & GPIO	11-DEC-1980 19:30	AI: CADRIO; CLK60H



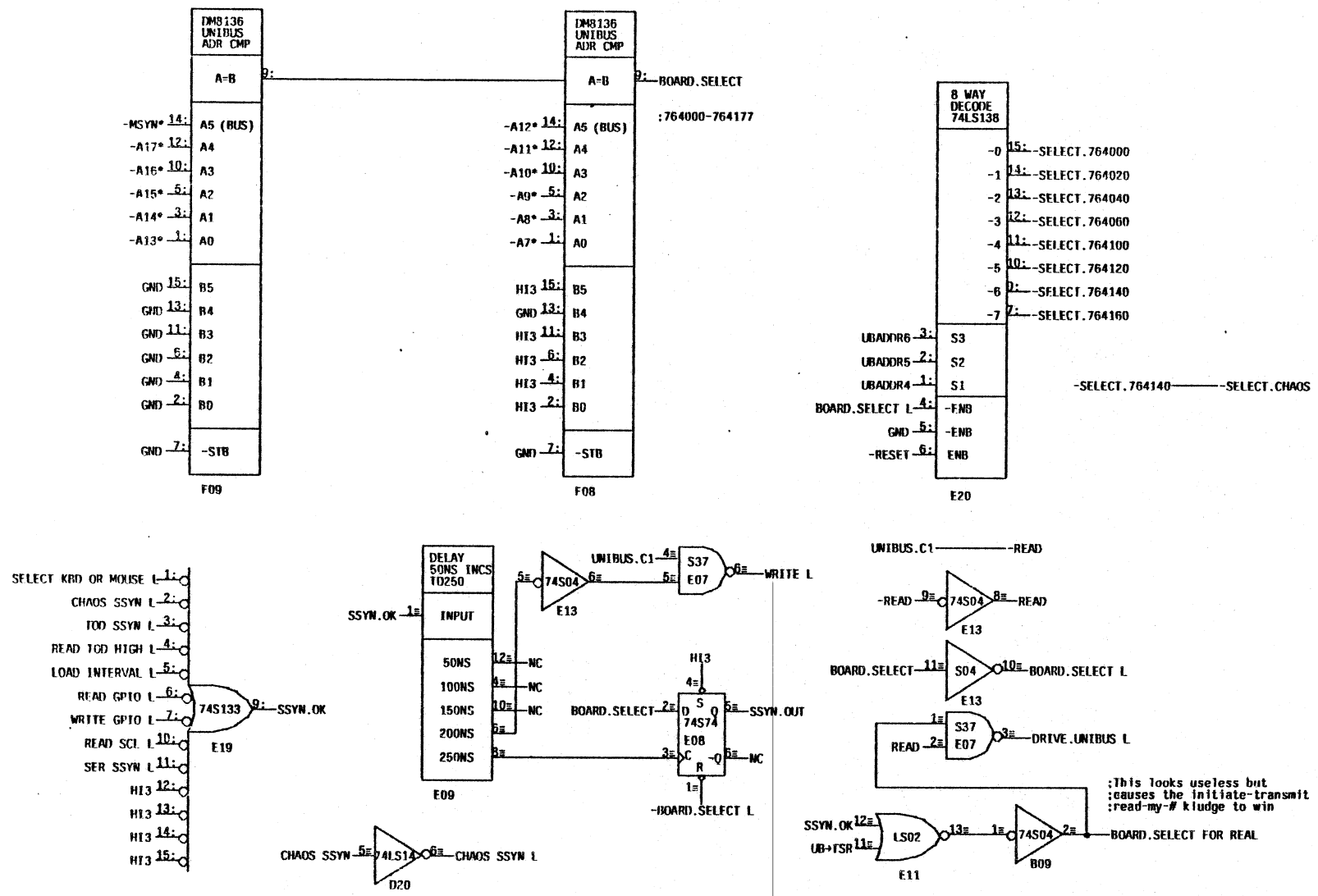
I/O BOARD	INTERVAL TIMER	11-DEC-1980 18:40	AI: CADRIO; CLKTIM

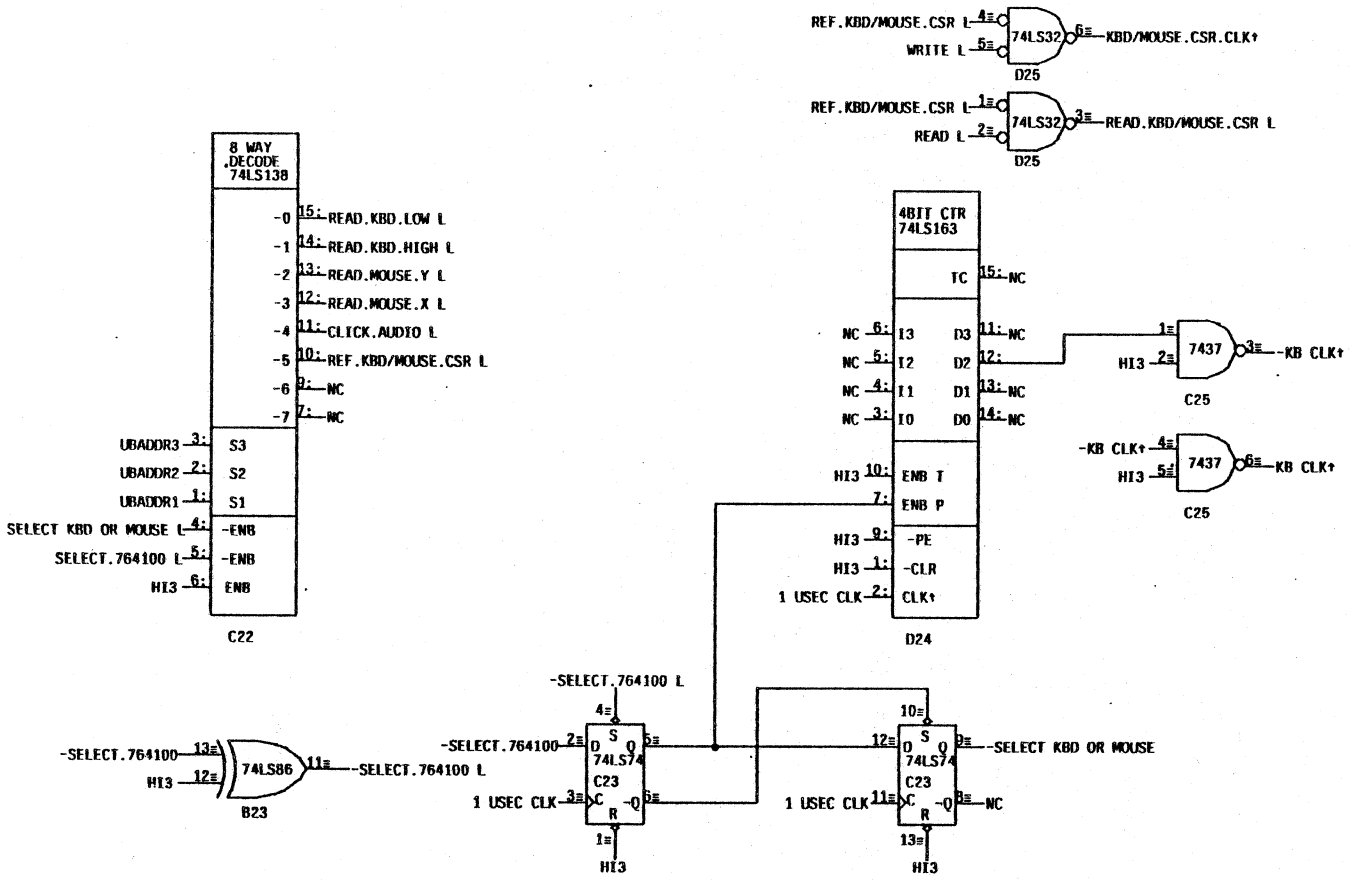


I/O BOARD	TIME OF DAY CLOCKS	11-DEC-1980 10:50	AI: CADRIO; CLKTOD

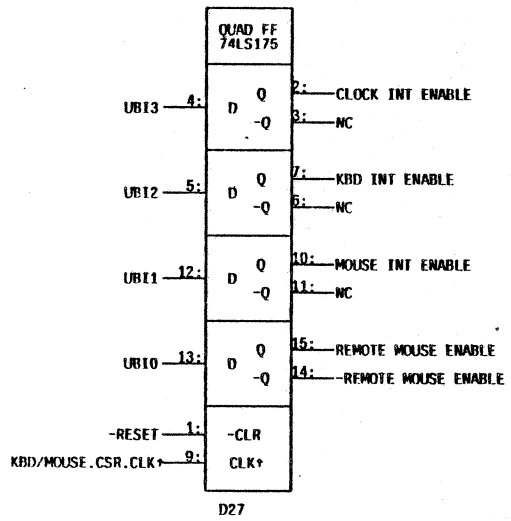


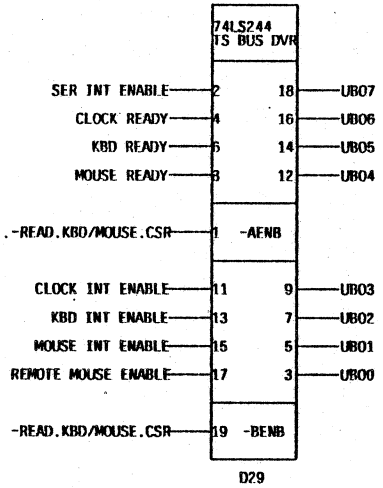
HEX SPC SLOT	BACKPLANE CONNECTIONS	15-SEP-1979 10:41	AI: CADRIO; HEXSPC

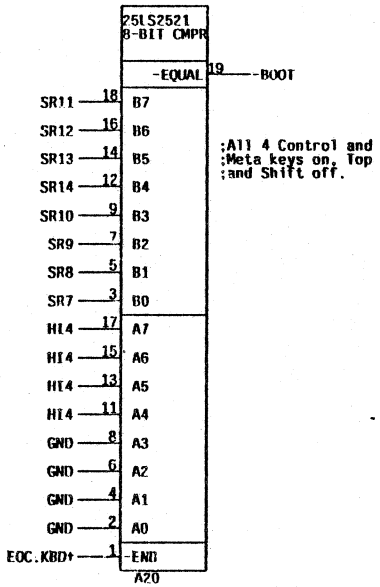

DM8136
UNIBUS
ADR CMP
A=B
A5 (BUS)
A4
A3
A2
A1
A0
B5
B4
B3
B2
B1
B0
-STB
-MSYN*
-A17*
-A16*
-A15*
-A14*
-A13*
GND
F09
DM8136
UNIBUS
ADR CMP
A=B
BOARD.SELECT
:764000-764177
A5 (BUS)
A4
A3
A2
A1
A0
B5
B4
B3
B2
B1
B0
-STB
-A12*
-A11*
-A10*
-A9*
-A8*
-A7*
HI3
GND
F08
8 WAY
DECODE
74LS138
-0
-1
-2
-3
-4
-5
-6
-7
-SELECT.764000
-SELECT.764020
-SELECT.764040
-SELECT.764060
-SELECT.764100
-SELECT.764120
-SELECT.764140
-SELECT.764160
S3
S2
S1
-ENB
-ENB
ENB
UBADDR6
UBADDR5
UBADDR4
BOARD.SELECT L
GND
-RESET
-SELECT.764140
-SELECT.CHAOS
E20
SELECT KBD OR MOUSE L
CHAOS SSYN L
TOD SSYN L
READ TOD HIGH L
LOAD INTERVAL L
READ GPIO L
WRITE GPIO L
READ SCL L
SER SSYN L
HI3
HI3
HI3
HI3
74S133
SSYN.OK
E19
DELAY
50NS INCS
TD250
INPUT
50NS
100NS
150NS
200NS
250NS
SSYN.OK
NC
NC
NC
E09
74S04
E13
UNIBUS.C1
S37
E07
WRITE L
HI3
74S74
E08
BOARD.SELECT
SSYN.OUT
NC
-BOARD.SELECT L
CHAOS SSYN
74LS14
D20
CHAOS SSYN L
UNIBUS.C1
-READ
-READ
READ
BOARD.SELECT
S04
BOARD.SELECT L
S37
E07
DRIVE.UNIBUS L
;This looks useless but
;causes the initiate-transmit
;read-my-# kludge to win
SSYN.OK
UB→TSR
LS02
E11
74S04
B09
BOARD.SELECT FOR REAL



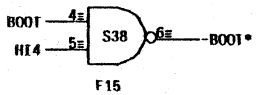
I/O BOARD	KEYBOARD, MOUSE CLOCK SYNC	11-DEC-1980 11:09	AI: CADRIO; IOBCLK

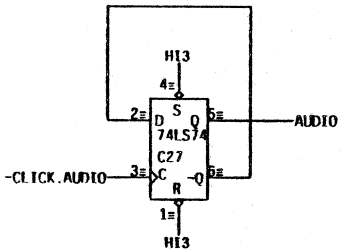

QUAD FF
74LS175
UBI3
4:
D
Q
-Q
2:
CLOCK INT ENABLE
3:
NC
UBI2
5:
D
Q
-Q
7:
KBD INT ENABLE
6:
NC
UBI1
12:
D
Q
-Q
10:
MOUSE INT ENABLE
11:
NC
UBI0
13:
D
Q
-Q
15:
REMOTE MOUSE ENABLE
14:
-REMOTE MOUSE ENABLE
-RESET
1:
-CLR
KBD/MOUSE.CSR.CLK↑
9:
CLK↑
D27

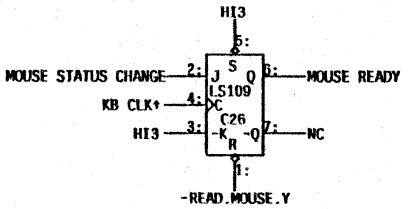

74LS244
TS BUS DVR
SER INT ENABLE
2
18
UB07
CLOCK READY
4
16
UB06
KBD READY
6
14
UB05
MOUSE READY
8
12
UB04
.-READ.KBD/MOUSE.CSR
1
-AENB
CLOCK INT ENABLE
11
9
UB03
KBD INT ENABLE
13
7
UB02
MOUSE INT ENABLE
15
5
UB01
REMOTE MOUSE ENABLE
17
3
UB00
-READ.KBD/MOUSE.CSR
19
-BENB
D29

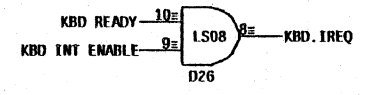

25LS2521
8-BIT CMPR
-EQUAL
19
-BOOT
SR11
18
B7
SR12
16
B6
SR13
14
B5
SR14
12
B4
SR10
9
B3
SR9
7
B2
SR8
5
B1
SR7
3
B0
HI4
17
A7
HI4
15
A6
HI4
13
A5
HI4
11
A4
GND
8
A3
GND
6
A2
GND
4
A1
GND
2
A0
EOC.KBD↑
1
-END
A20
;All 4 Control and
;Meta keys on, Top
;and Shift off.

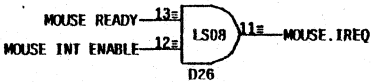

-BOOT
13≡
74S04
12≡
BOOT
E13

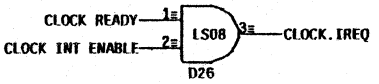

BOOT
4≡
HI4
5≡
S38
6≡
-BOOT*
F15

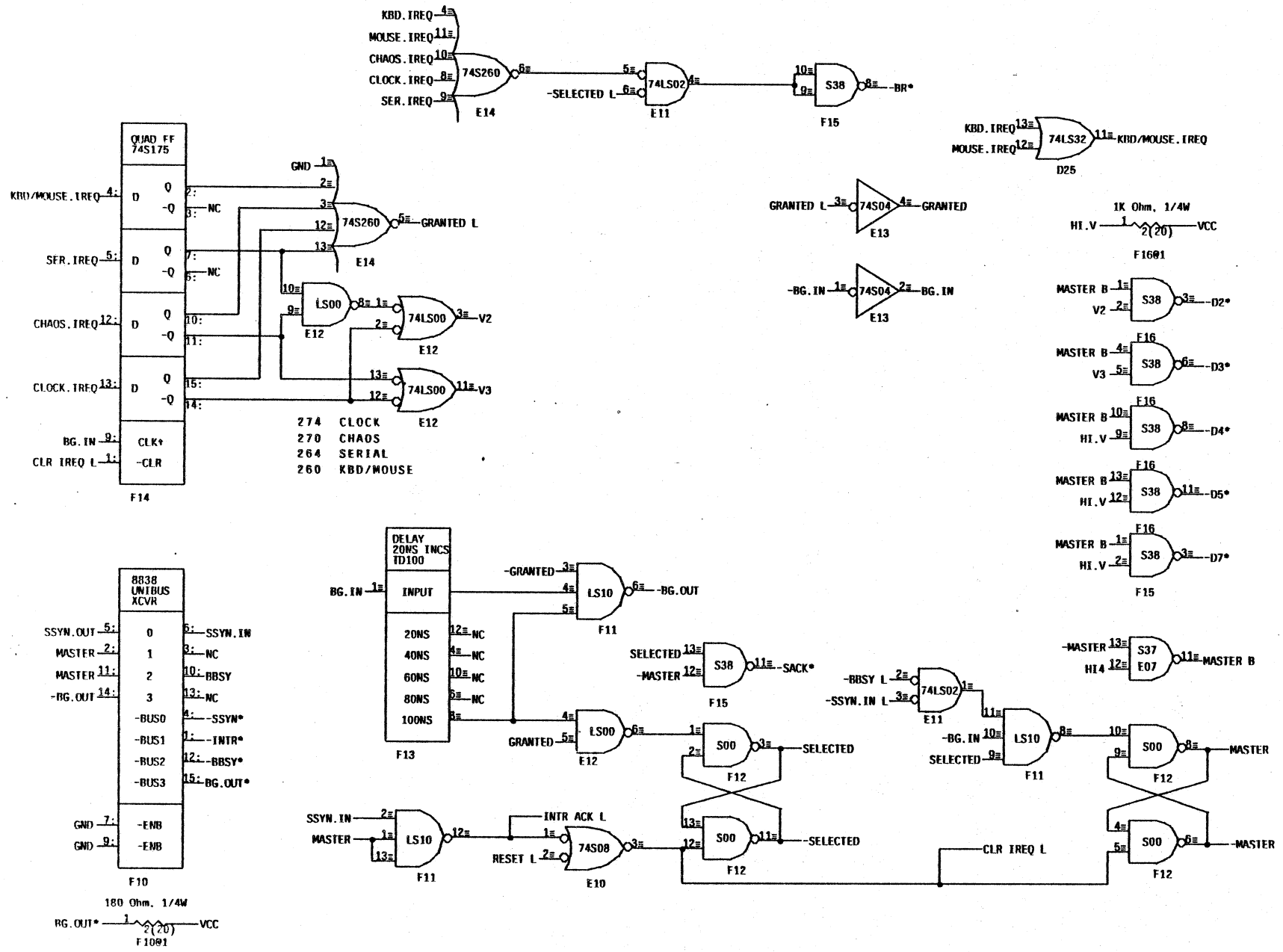

HI3
4≡
2≡
D
S
Q
74LS74
C27
C
R
-Q
5≡
6≡
AUDIO
-CLICK.AUDIO
3≡
1≡
HI3


HI3
5:
MOUSE STATUS CHANGE
2:
J
S
Q
LS109
C
C26
-K
R
-Q
6:
MOUSE READY
KB CLK↑
4:
HI3
3:
7:
NC
1:
-READ.MOUSE.Y

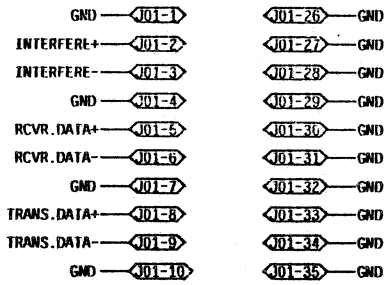

KBD READY
10≡
KBD INT ENABLE
9≡
LS08
8≡
KBD.IREQ
D26


MOUSE READY
13≡
MOUSE INT ENABLE
12≡
LS08
11≡
MOUSE.IREQ
D26

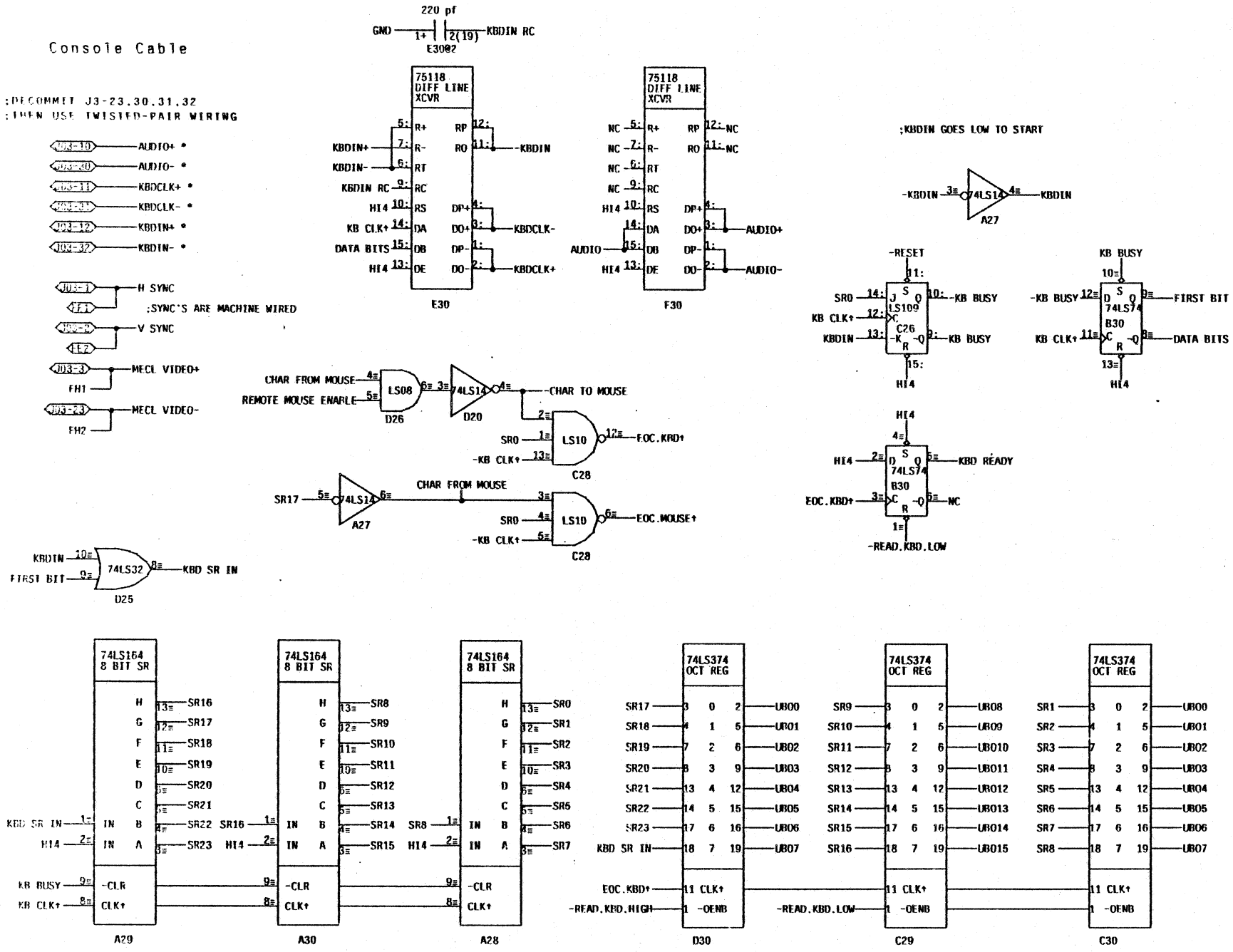

CLOCK READY
1≡
CLOCK INT ENABLE
2≡
LS08
3≡
CLOCK.IREQ
D26



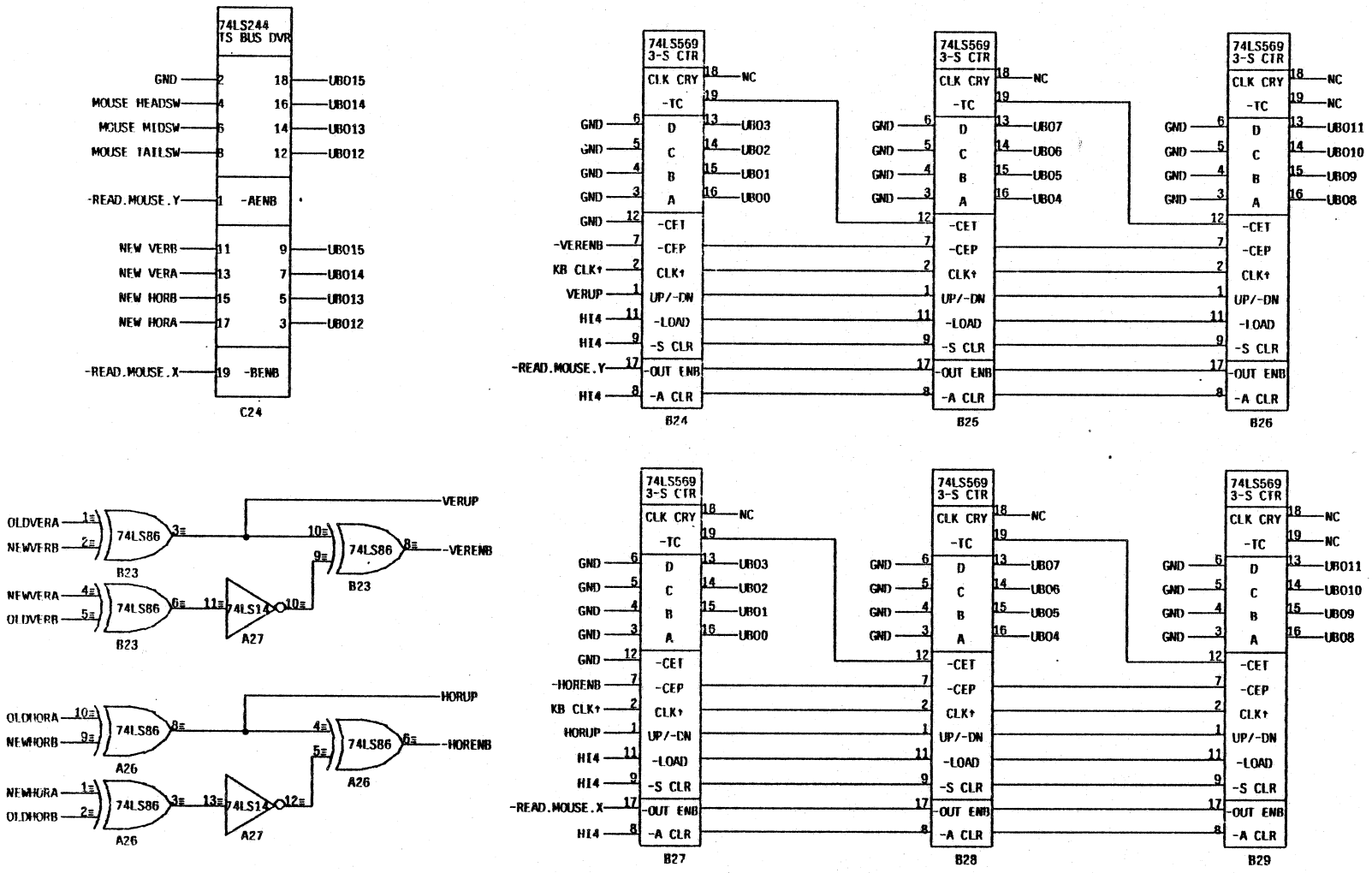
I/O BOARD	UNIBUS INTERRUPT	11-DEC-1980 10:39	AI: CADRIO; IOBINT



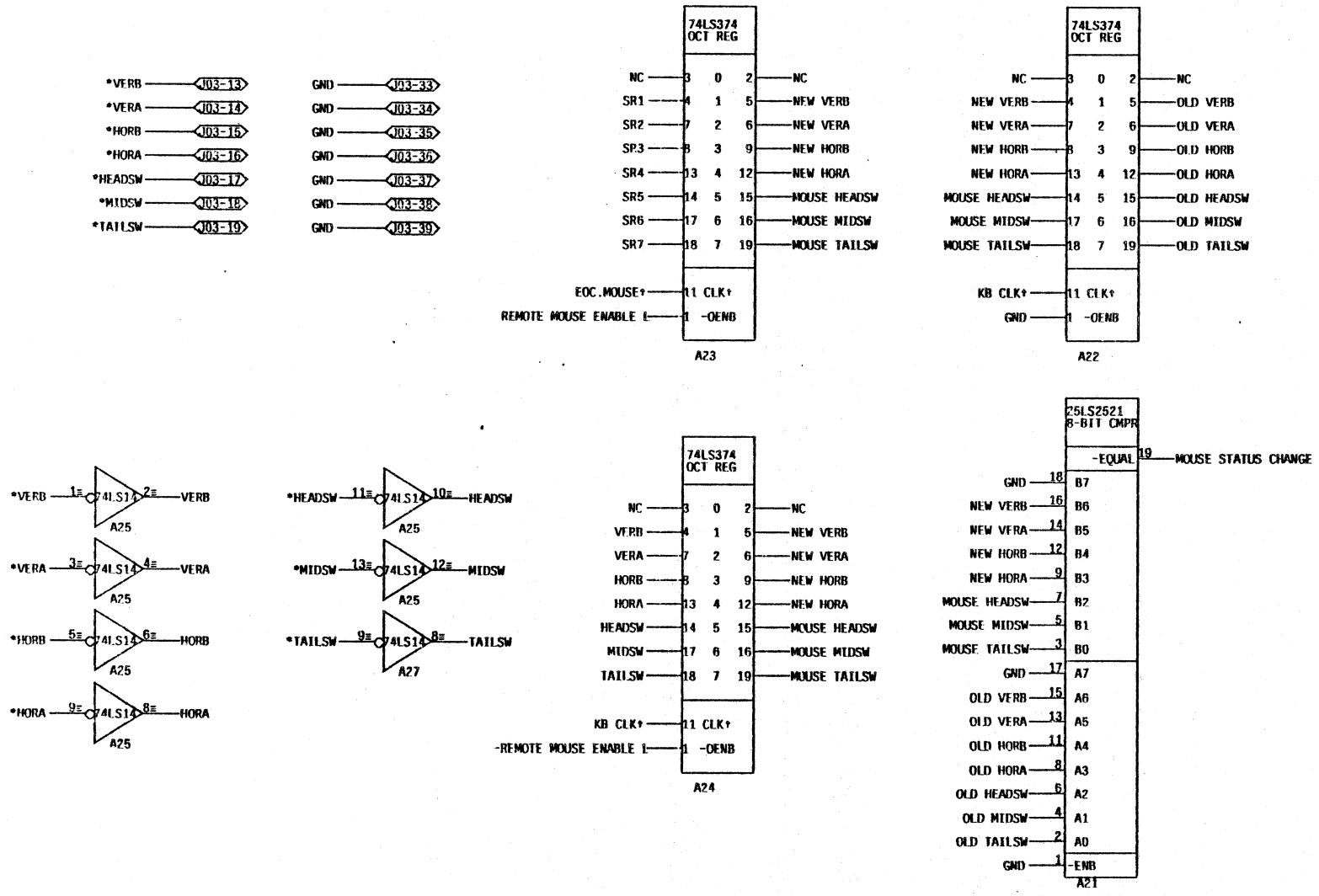
LISP MACHINE CHAOS NET CONN TO TRANSCEIVER	17-DEC-1980 11:10	AI: CADRIO; IOBJPS



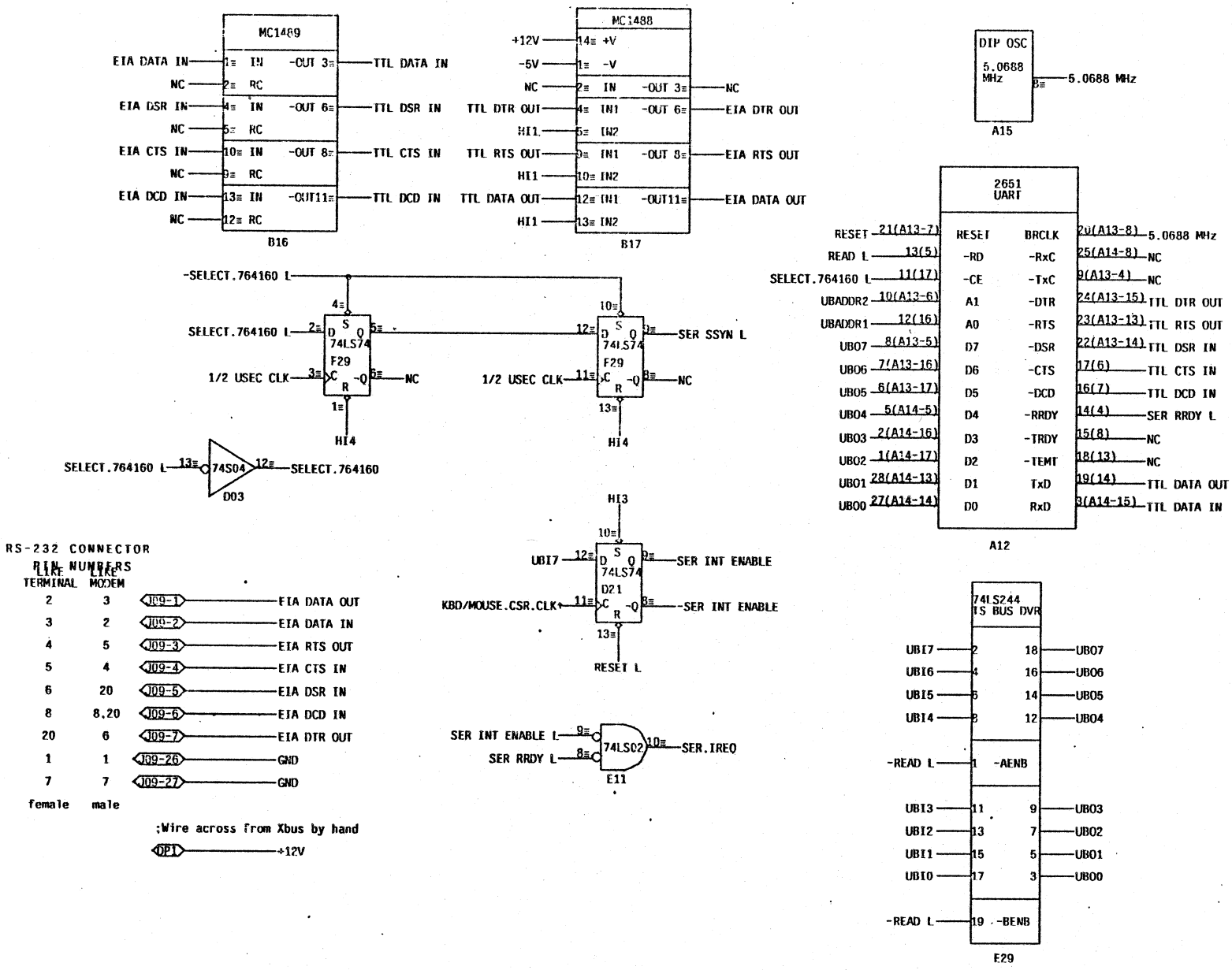
I/O BOARD	KEYBOARD CONTROL	12-DEC-1980 02:16	AI: CADRIO; IOBKBD

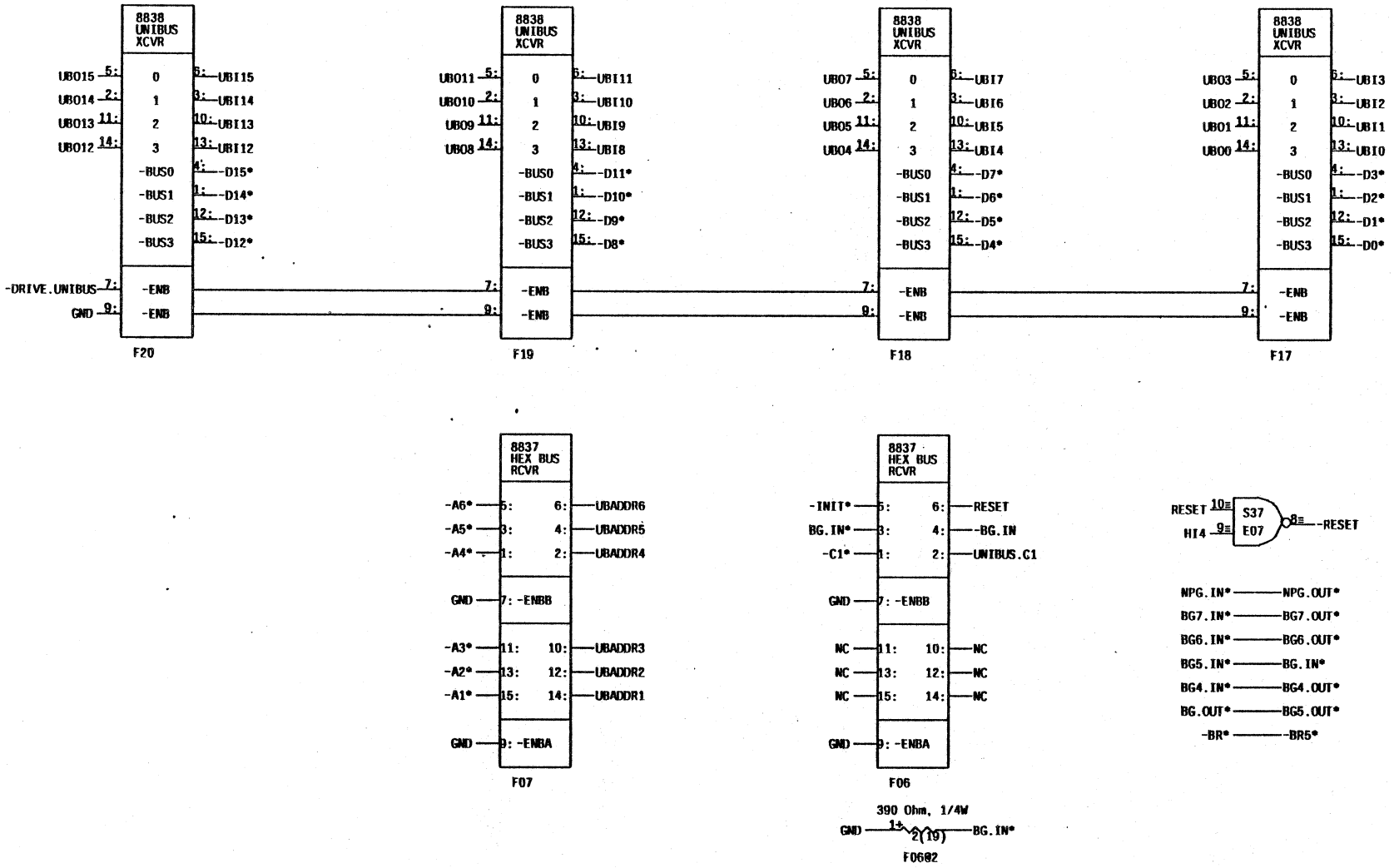


I/O BOARD	MOUSE 2	11-DEC-1980 18:46	AI: CADRIO; IOBMS2

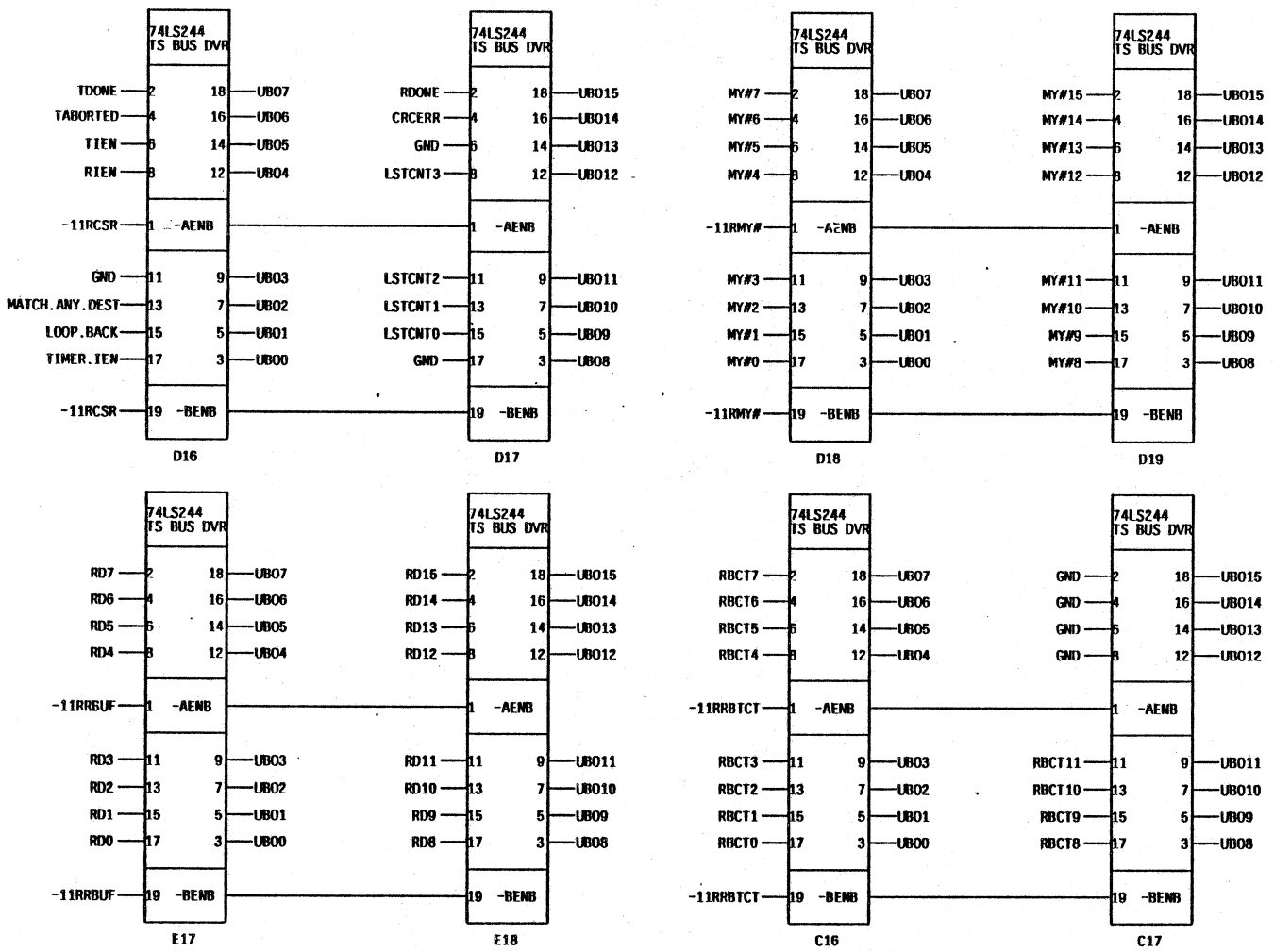


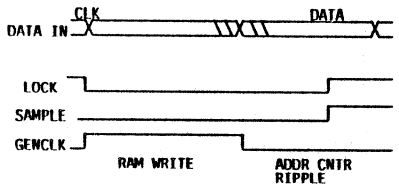
I/O BOARD	MOUSE INTERFACE	11-DEC-1980 11:43	AI: CADRIO; IOBMSE

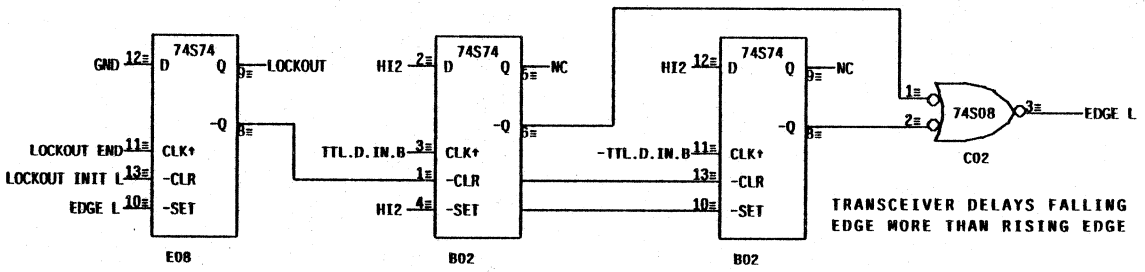

MC1489
EIA DATA IN
NC
EIA DSR IN
NC
EIA CTS IN
NC
EIA DCD IN
NC
-OUT 3
-OUT 6
-OUT 8
-OUT11
TTL DATA IN
TTL DSR IN
TTL CTS IN
TTL DCD IN
B16
MC1488
+12V
-5V
NC
TTL DTR OUT
HI1
TTL RTS OUT
HI1
TTL DATA OUT
HI1
-OUT 3
-OUT 6
-OUT 8
-OUT11
NC
EIA DTR OUT
EIA RTS OUT
EIA DATA OUT
B17
DIP OSC
5.0688 MHz
5.0688 MHz
A15
2651 UART
RESET
-RD
-CE
A1
A0
D7
D6
D5
D4
D3
D2
D1
D0
BRCLK
-RxC
-TxC
-DTR
-RTS
-DSR
-CTS
-DCD
-RRDY
-TRDY
-TEMT
TxD
RxD
5.0688 MHz
NC
NC
TTL DTR OUT
TTL RTS OUT
TTL DSR IN
TTL CTS IN
TTL DCD IN
SER RRDY L
NC
NC
TTL DATA OUT
TTL DATA IN
A12
-SELECT.764160 L
SELECT.764160 L
1/2 USEC CLK
HI4
SER SSYN L
74LS74
F29
74S04
SELECT.764160
D03
HI3
UBI7
KBD/MOUSE.CSR.CLK
SER INT ENABLE
-SER INT ENABLE
D21
RESET L
SER INT ENABLE L
SER RRDY L
74LS02
SER.IREQ
E11
74LS244
TS BUS DVR
-READ L
-AENB
-BENB
E29
RS-232 CONNECTOR
PIN NUMBERS
LINE TERMINAL
LINE MODEM
female
male
J09-1
J09-2
J09-3
J09-4
J09-5
J09-6
J09-7
J09-26
J09-27
EIA DATA OUT
EIA DATA IN
EIA RTS OUT
EIA CTS IN
EIA DSR IN
EIA DCD IN
EIA DTR OUT
GND
GND
;Wire across from Xbus by hand
DP1
+12V



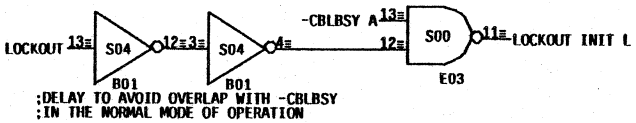
I/O BOARD	UNIBUS TRANSCEIVER	11-DEC-1980 11:16	AI: CADRIO; IOBXCV

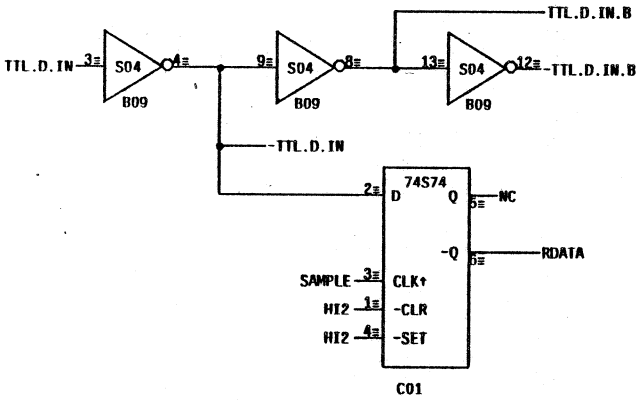

74LS244
TS BUS DVR
TDONE
TABORTED
TIEN
RIEN
-11RCSR
GND
MATCH.ANY.DEST
LOOP.BACK
TIMER.IEN
-11RCSR
UB07
UB06
UB05
UB04
-AENB
UB03
UB02
UB01
UB00
-BENB
D16
RDONE
CRCERR
GND
LSTCNT3
LSTCNT2
LSTCNT1
LSTCNT0
GND
UB015
UB014
UB013
UB012
UB011
UB010
UB09
UB08
D17
MY#7
MY#6
MY#5
MY#4
-11RMY#
MY#3
MY#2
MY#1
MY#0
-11RMY#
D18
MY#15
MY#14
MY#13
MY#12
MY#11
MY#10
MY#9
MY#8
D19
RD7
RD6
RD5
RD4
-11RRBUF
RD3
RD2
RD1
RD0
-11RRBUF
E17
RD15
RD14
RD13
RD12
RD11
RD10
RD9
RD8
E18
RBCT7
RBCT6
RBCT5
RBCT4
-11RRBTCT
RBCT3
RBCT2
RBCT1
RBCT0
-11RRBTCT
C16
GND
GND
GND
GND
RBCT11
RBCT10
RBCT9
RBCT8
C17

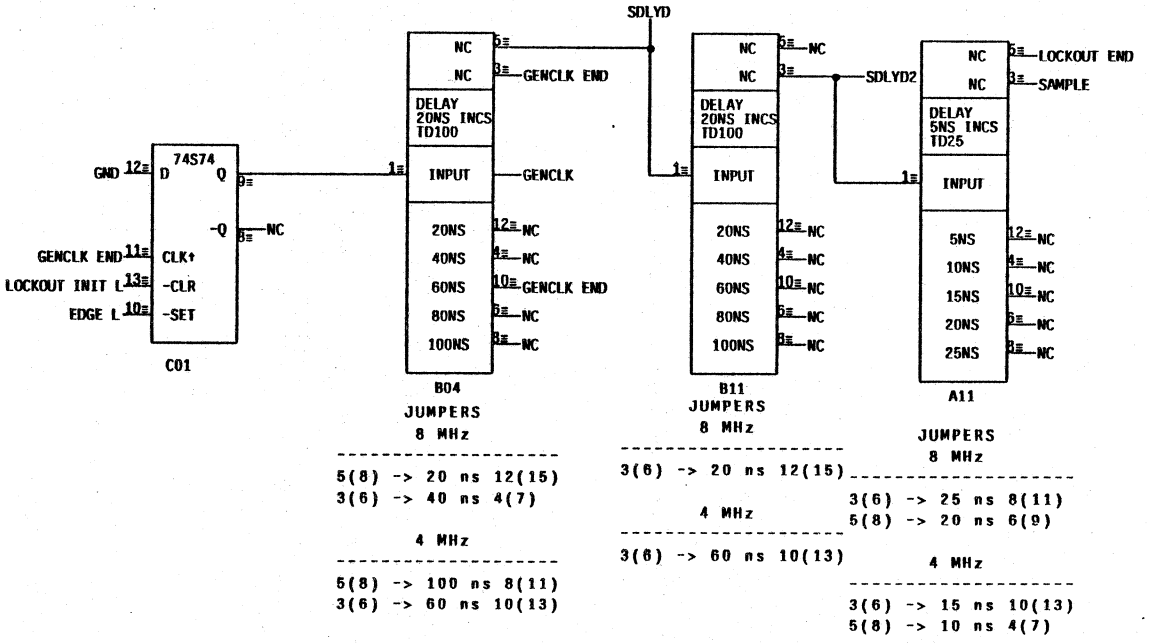




SAMPLE should be timed to occur in the middle
of the data part of the cycle.
Normally, this will be .75 of the cycle time
less 15-25 ns. LOCKOUT END should be 5 ns earlier.







JUMPERS
8 MHz

5(8) -> 20 ns 12(15)
3(6) -> 40 ns 4(7)

4 MHz

5(8) -> 100 ns 8(11)
3(6) -> 60 ns 10(13)

JUMPERS
8 MHz

3(6) -> 20 ns 12(15)

4 MHz

3(6) -> 60 ns 10(13)

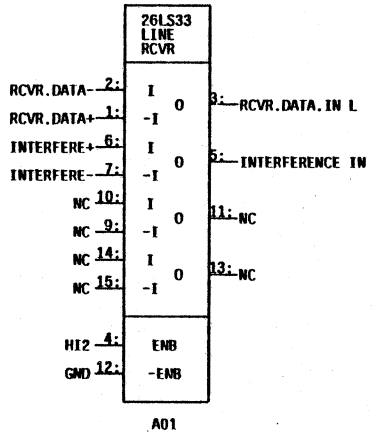
JUMPERS
8 MHz

3(6) -> 25 ns 8(11)
5(8) -> 20 ns 6(9)

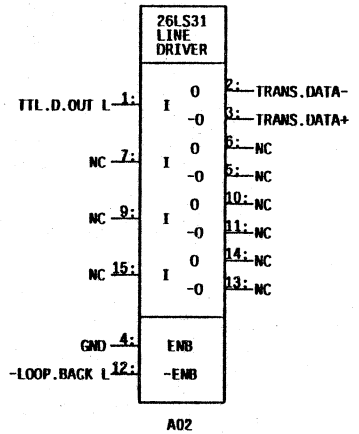
4 MHz

3(6) -> 15 ns 10(13)
5(8) -> 10 ns 4(7)

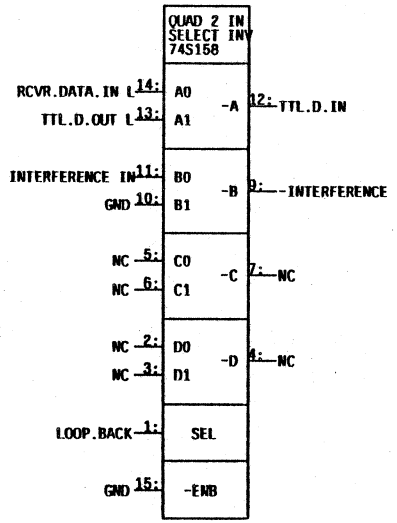
;=26LS32



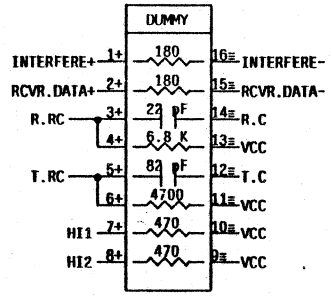
A01



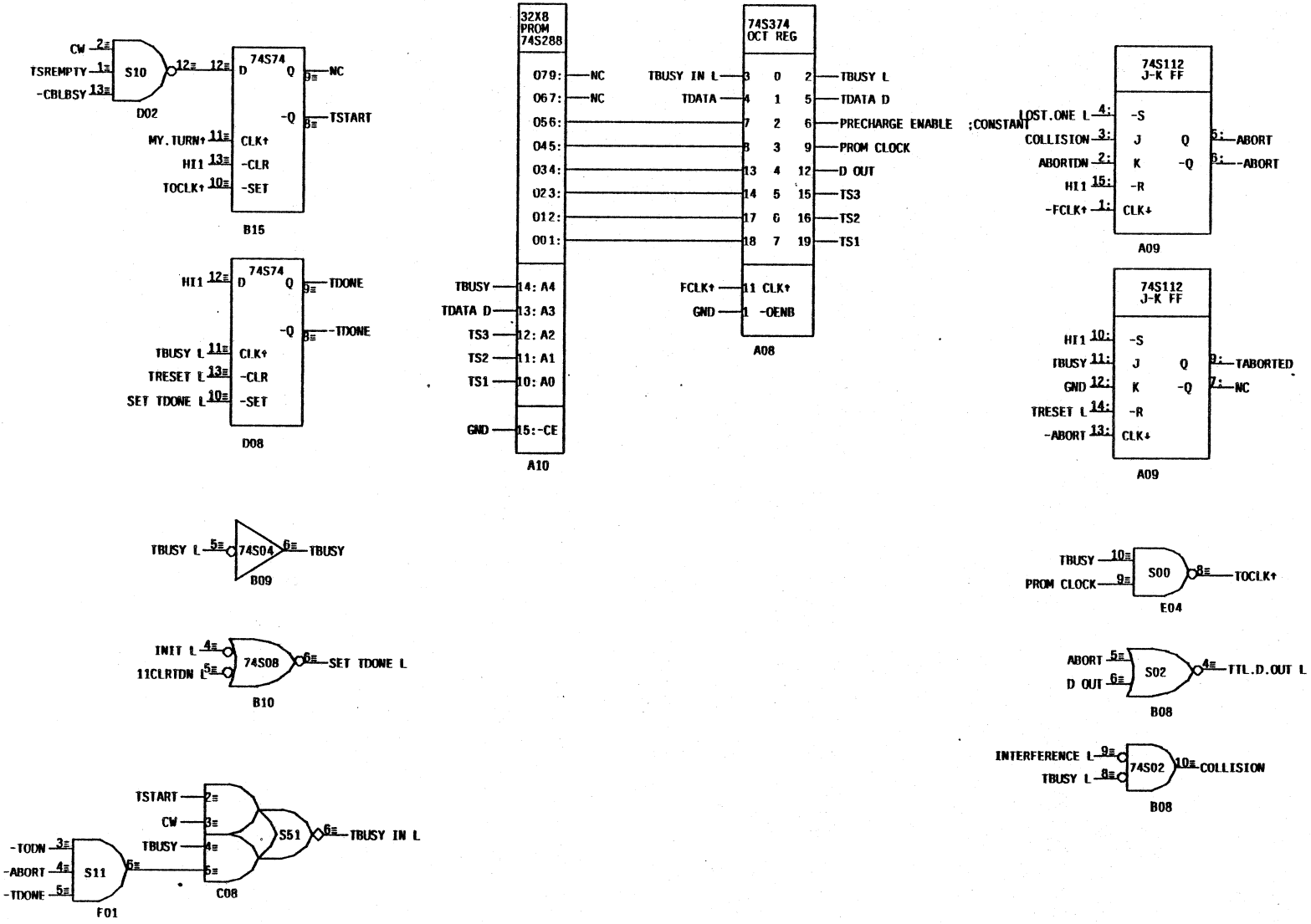
A02



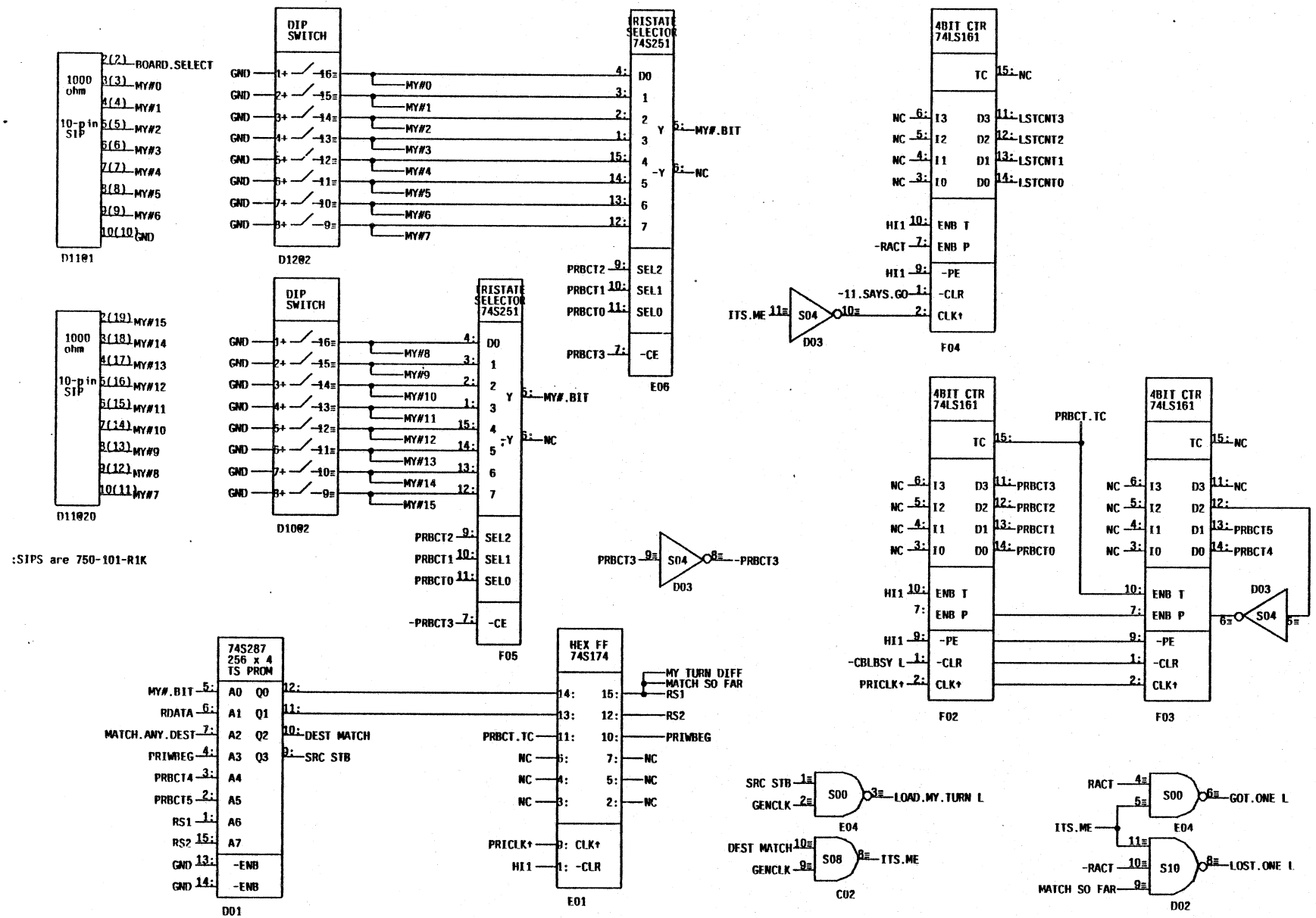
E02



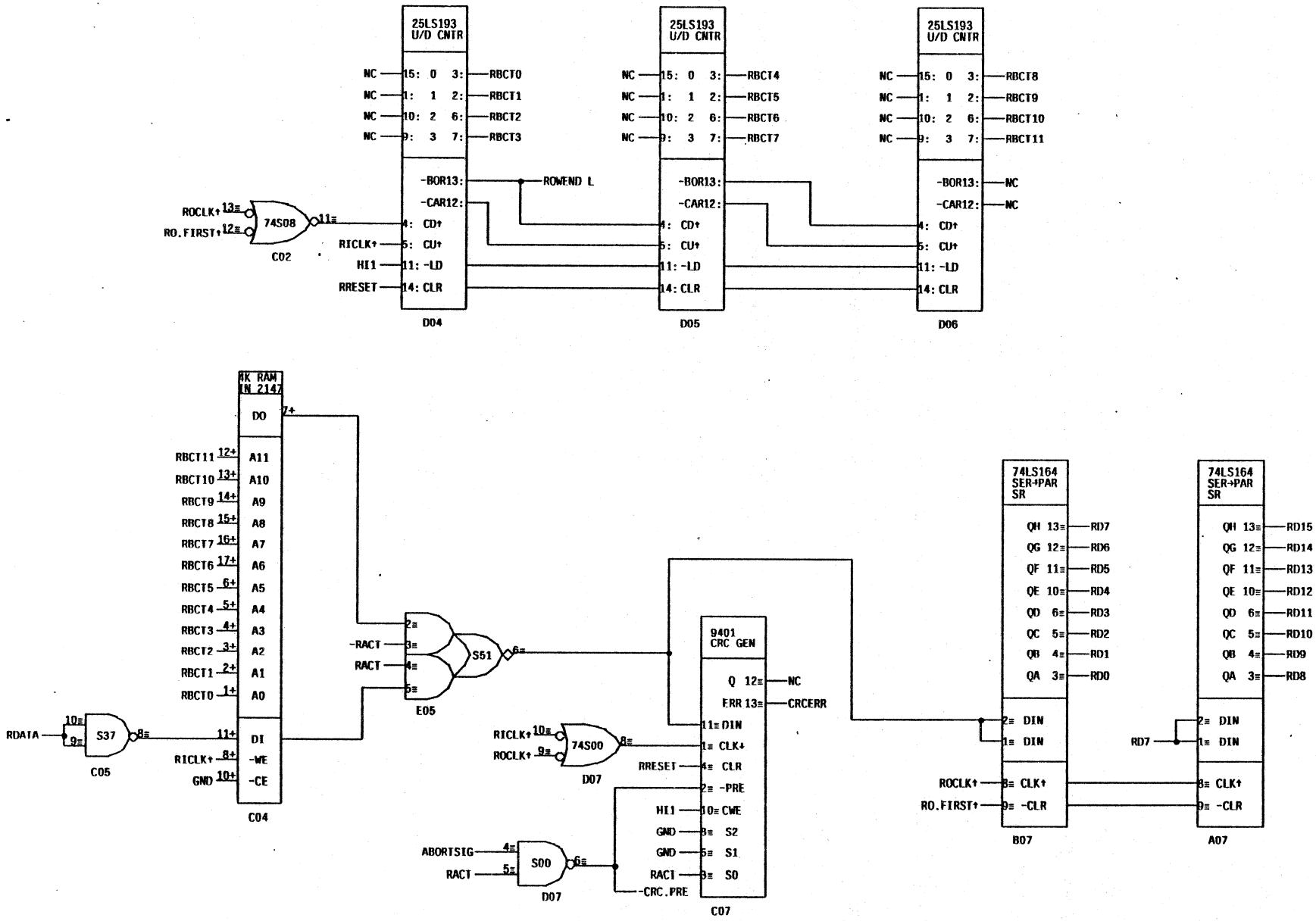
A0302



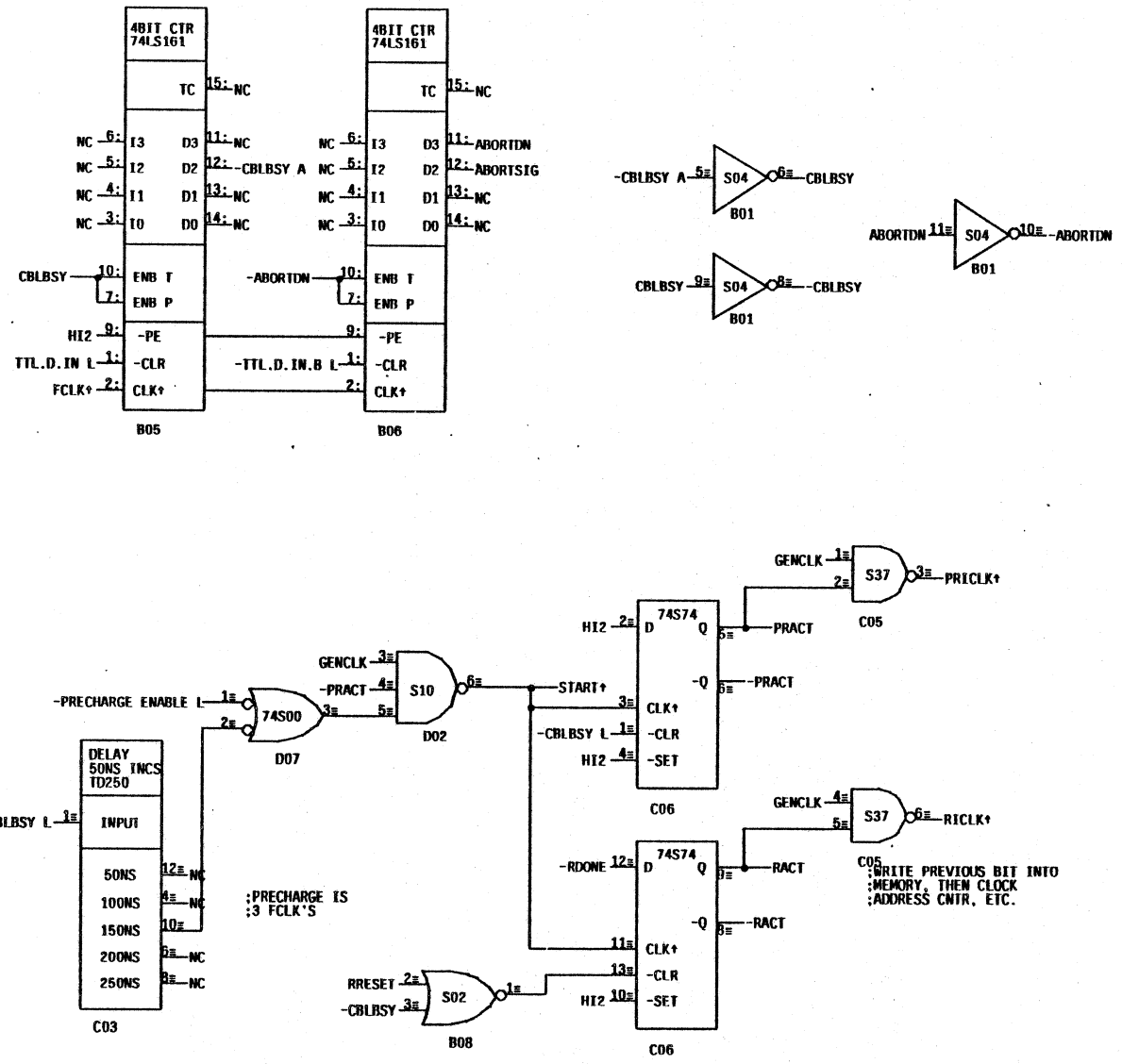
LISP MACHINE CHAOS NET SERIAL MODULATOR	13-AUG-1980 09:46	AI: CADRIO; LMMODU



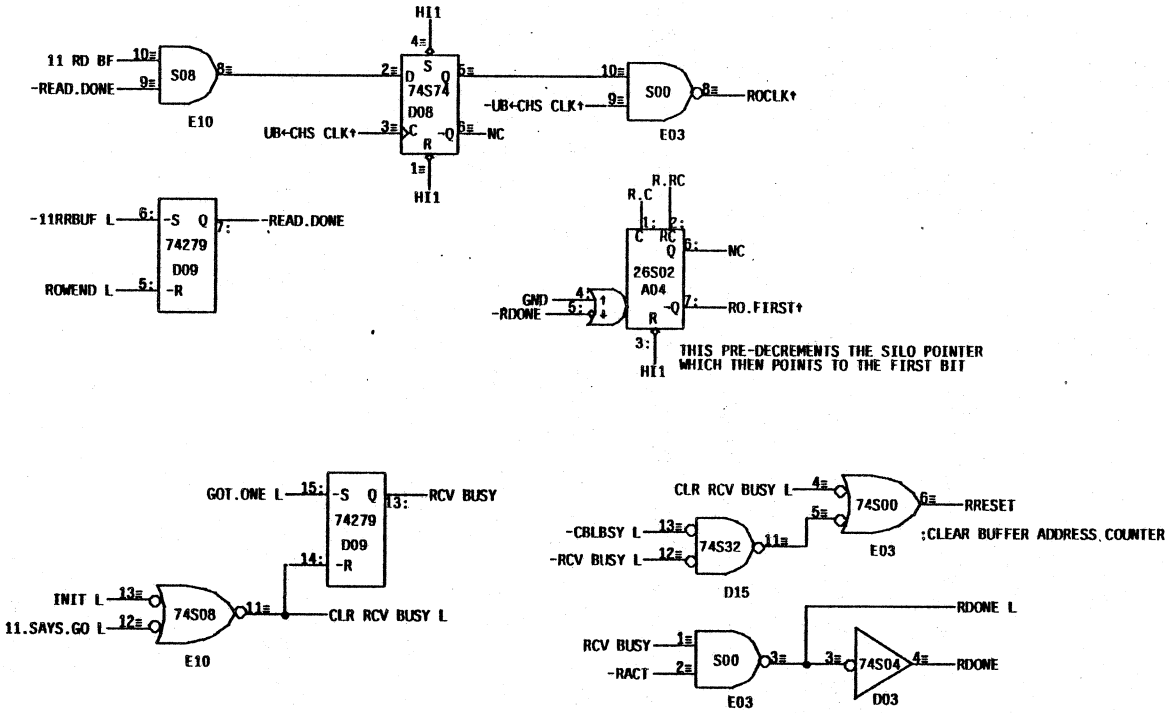
LISP MACHINE CHAOS NET FIND MSGS DESTINED FOR ME	13-AUG-1980 09:54	AI: CADRIO; LMMYNM



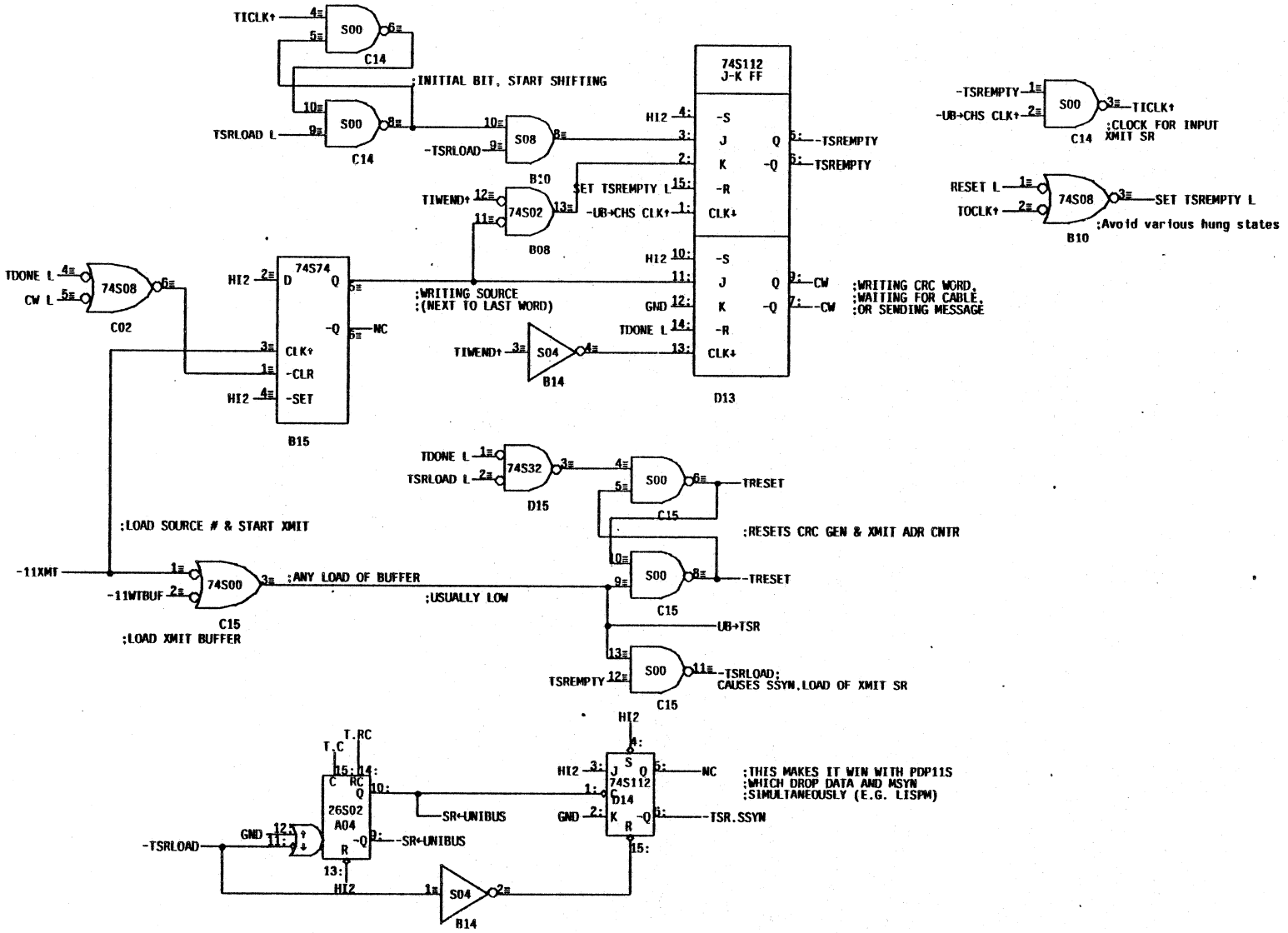
LISP MACHINE CHAOS NET RECEIVER BUFFER	13-AUG-1980 09:47	AI: CADRIO; LMRBUF



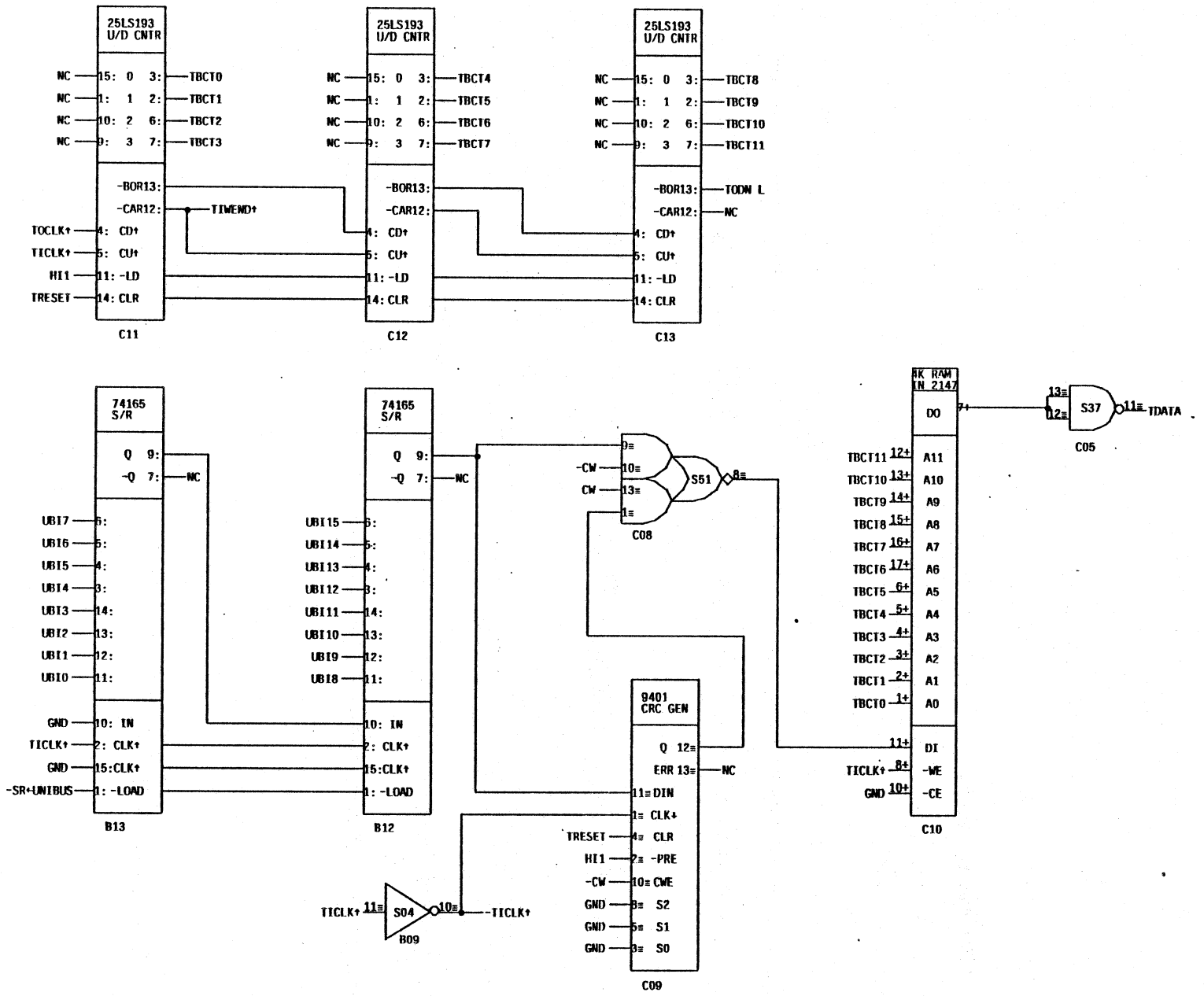
LISP MACHINE CHAOS NET RECEIVE CLOCK	13-AUG-1980 09:48	AI: CADRIO; LMRCLK



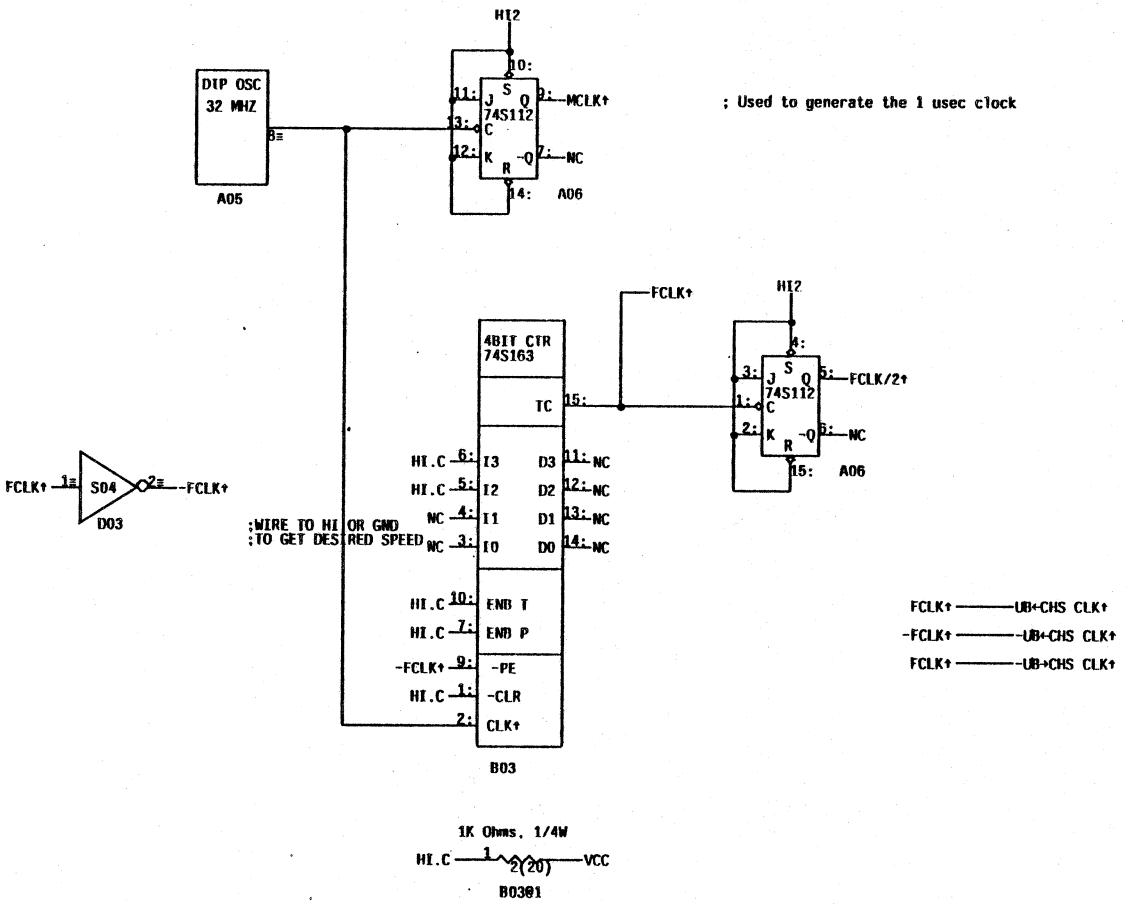
LISP MACHINE CHAOS NET RECEIVE CONTROL	13-AUG-1980 09:48	AI: CADRIO; LMRCTL



LISP MACHINE CHAOS NET TRANSMIT CONTROL	13-AUG-1980 09:49	AI: CADRIO; LMTBFC

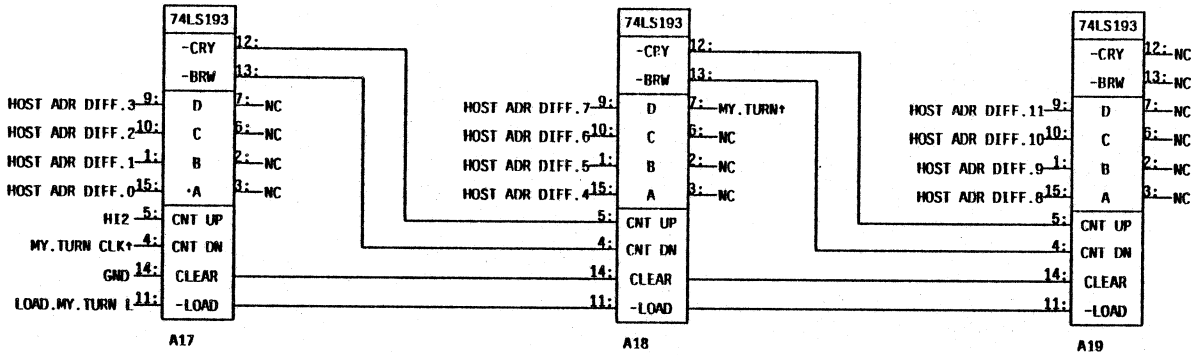


LISP MACHINE CHAOS NET TRANSMIT BUFFER	13-AUG-1980 09:49	AI: CADRIO; LMTBUF

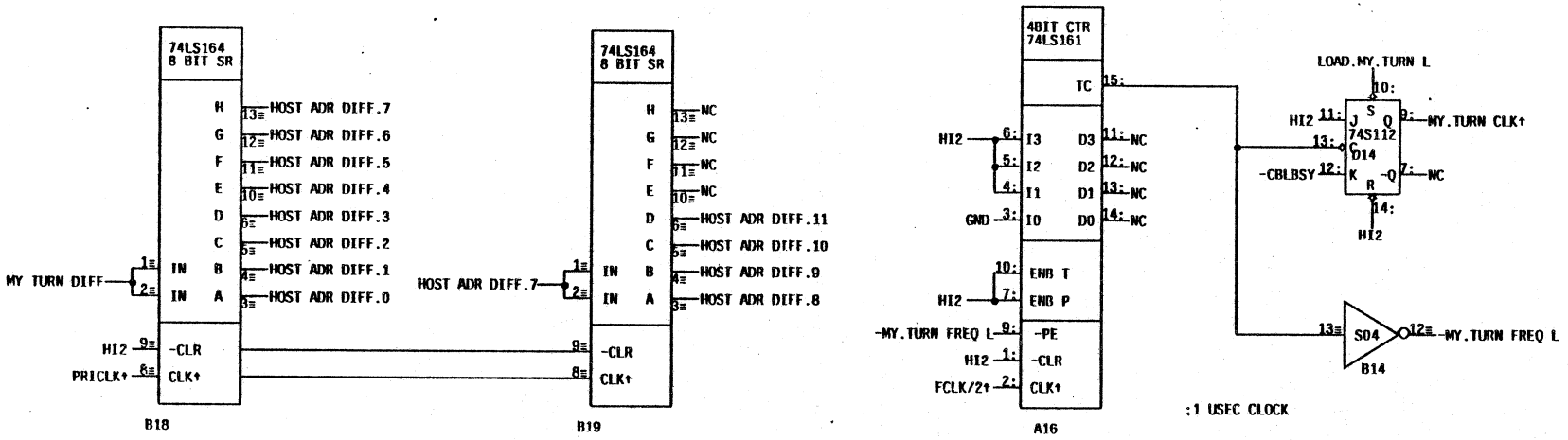


LISP MACHINE CHAOS NET TRANSMIT CLOCK	17-DEC-1980 11:19	AI: CADRIO; LMTCLK

"My Turn" Counter

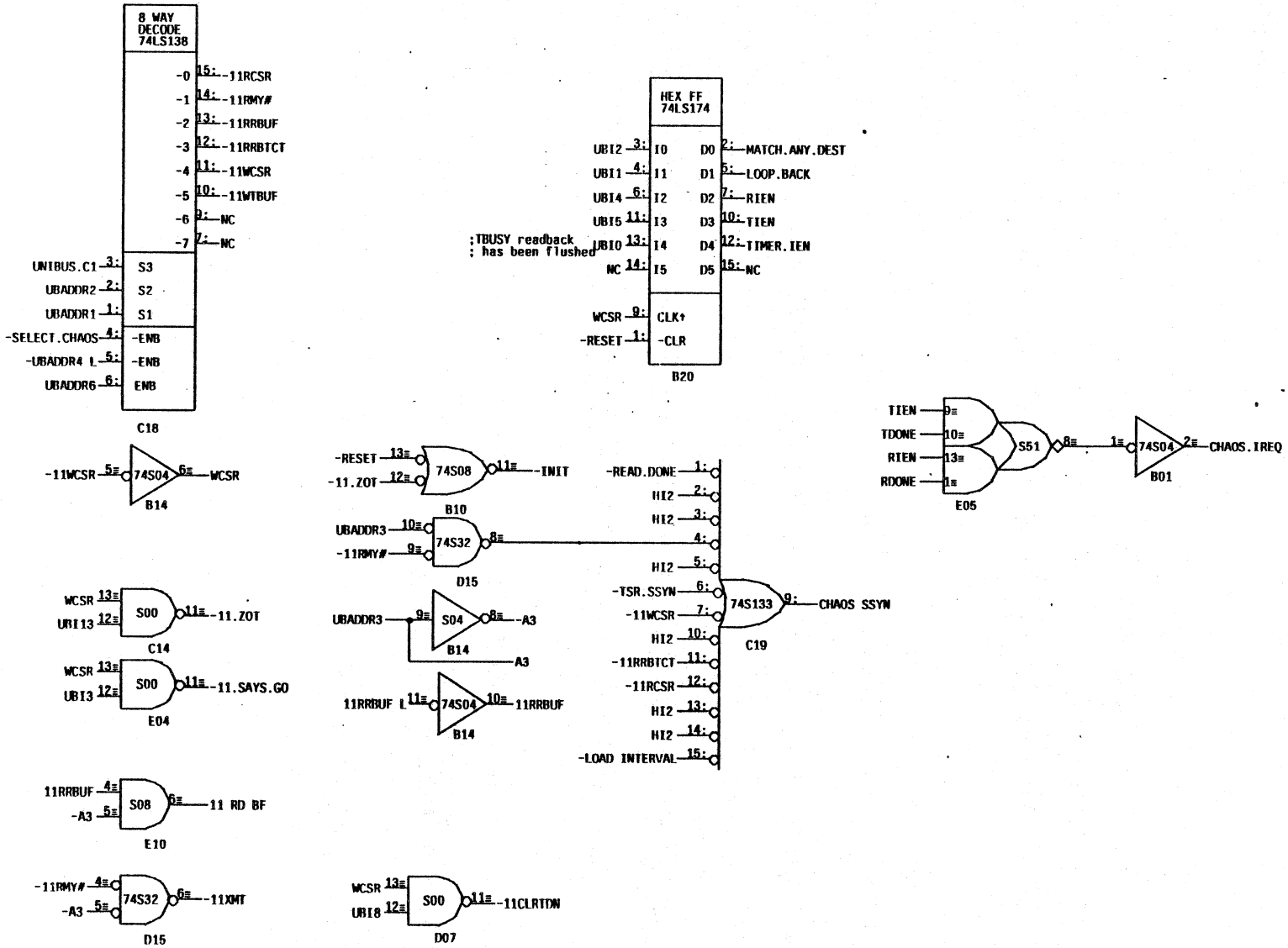


:DEFAULT IS MYTURN.MOD 2+7 OR 64 USECS



:1 USEC CLOCK

LISP MACHINE CHAOS NET MY TURN TIMER	13-AUG-1980 09:51	AI: CADRIO; LMTURN



LISP MACHINE CHAOS NET UNIBUS CONTROL	16-DEC-1980 22:05	AI: CADRIO; LMUCON

LISP Machine TV Board CADRTV;LMTV4B UML 07-DEC-80 2352
******* DIP MAP *******

26S10 XBDATA x F30	74LS299 RAMSHF x E30	2118 RAMD x D30	2118 RAMC x C30	2118 RAMB x B30	2118 RAMA x A30
26S10 XBDATA x F29	74LS374 RAMREG x E29	2118 RAMD x D29	2118 RAMC x C29	2118 RAMB x B29	2118 RAMA x A29
26S10 XBDATA x F28	74LS299 RAMSHF x E28	2118 RAMD x D28	2118 RAMC x C28	2118 RAMB x B28	2118 RAMA x A28
26S10 XBDATA x F27	74LS374 RAMREG x E27	2118 RAMD x D27	2118 RAMC x C27	2118 RAMB x B27	2118 RAMA x A27
26S10 XBDATA x F26	74LS299 RAMSHF x E26	2118 RAMD x D26	2118 RAMC x C26	2118 RAMB x B26	2118 RAMA x A26
26S10 XBDATA x F25	74LS374 RAMREG x E25	2118 RAMD x D25	2118 RAMC x C25	2118 RAMB x B25	2118 RAMA x A25
26S10 XBDATA x F24	74LS299 RAMSHF x E24	2118 RAMD x D24	2118 RAMC x C24	2118 RAMB x B24	2118 RAMA x A24
26S10 XBDATA x F23	74LS374 RAMREG x E23	2118 RAMD x D23	2118 RAMC x C23	2118 RAMB x B23	2118 RAMA x A23
25LS2521 XBADR x F22	74LS299 RAMSHF x E22	2118 RAMD x D22	2118 RAMC x C22	2118 RAMB x B22	2118 RAMA x A22
25LS2521 XBADR x F21	74LS374 RAMREG x E21	2118 RAMD x D21	2118 RAMC x C21	2118 RAMB x B21	2118 RAMA x A21
25LS2521 XBADR x F20	74LS299 RAMSHF x E20	2118 RAMD x D20	2118 RAMC x C20	2118 RAMB x B20	2118 RAMA x A20

LISP Machine TV Board CADRTV;LMTV4B UML 07-DEC-80 2353
******* DIP MAP *******

25LS2521 XBADR x F19	74LS374 RAMREG x E19	2118 RAMD x D19	2118 RAMC x C19	2118 RAMB x B19	2118 RAMA x A19
74LS240 XBADR x F18	74LS299 RAMSHF x E18	2118 RAMD x D18	2118 RAMC x C18	2118 RAMB x B18	2118 RAMA x A18
74LS240 XBADR x F17	74LS374 RAMREG x E17	2118 RAMD x D17	2118 RAMC x C17	2118 RAMB x B17	2118 RAMA x A17
74LS240 XBADR x F16	74LS299 RAMSHF x E16	2118 RAMD x D16	2118 RAMC x C16	2118 RAMB x B16	2118 RAMA x A16
26S10 XBDATA x F15	74LS374 RAMREG x E15	2118 RAMD x D15	2118 RAMC x C15	2118 RAMB x B15	2118 RAMA x A15
26S10 XBDATA x F14	74LS74 XBCTL xx E14	898-3-R2 RAMBUF x D14	898-3-R2 RAMBUF x C14	898-3-R2 RAMBUF x B14	898-3-R2 RAMBUF x A14
74S138 XBCTL x F13	74S00 XBCTL xxxx E13	74LS244 COLOR x D13	74S37 RAMCAS xxxx C13	898-3-R2 RAMCAS x B13	74S241 RAMBUF x A13
25LS2519 XBCTL x F12	74S374 XBCTL x E12	74S00 SYNCLK xxxx D12	25LS2539 RAMCAS xx C12	74S253 RAMADR x B12	74S253 RAMADR x A12
74LS244 COLOR x F11	74S32 COLOR xxxo E11	TD100 RAMCAS x D11	74S37 SYNCLK xxxx C11	74S253 RAMADR x B11	74S253 RAMADR x A11
74S04 XBDATA xxxxxx F10	74S139 XBCTL xx E10	74S08 SYNCLK xxxx D10	74LS163 TVMA x C10	74LS569 RAMADR x B10	74LS569 RAMADR x A10
74S257 COLOR x F09	74S241 COLOR x E09	74S37 SYNCLK xxxx D09	74LS163 TVMA x C09	74LS163 TVMA x B09	74LS163 TVMA x A09

LISP Machine TV Board CADRTV;LMTV4B UML 07-DEC-80 2353
******* DIP MAP *******

10105 ECLCLK oox F08	74128 ECLVID xxoo E08	74S04 RAMADR xxxxxx D08	EXAR-CL ECLCLK x C08	74S283 TVINC x B08	74S283 TVINC x A08
10124 ECLCLK x F07	10124 ECLVID x E07	74S374 SYNCLK x D07	74LS377 SYNADR x C07	74LS377 SYNADR x B07	74LS273 TVINC x A07
10124 ECLVID x F06	10124 ECLVID x E06	74S288 SYNCLK x D06	74LS569 SYNADR x C06	74LS569 SYNADR x B06	74LS569 SYNADR x A06
DUAL-SIP ECLSIP x F05	DUAL-SIP ECLSIP x E05	74LS244 SYNRAM x D05	74S472 SYNRAM x C05	74LS374 SYNADR x B05	74LS374 SYNADR x A05
10102 ECLVID xxxx F04	10141 ECLVID x E04	74S51 TVMA xx D04	25LS2539 SYNREG xx C04	2147 SYNRAM x B04	2147 SYNRAM x A04
10212 ECLVID xo F03	10141 ECLVID x E03	74S04 TVMA xxxxxx D03	74S374 SYNREG x C03	2147 SYNRAM x B03	2147 SYNRAM x A03
10136 ECLCLK x F02	10125 ECLCLK x E02	74LS175 SYNREG x D02	74LS669 SYNREG x C02	2147 SYNRAM x B02	2147 SYNRAM x A02
10121 ECLVID x F01	DUAL-SIP ECLSIP x E01	74S10 SYNREG xxx D01	74LS669 SYNREG x C01	2147 SYNRAM x B01	2147 SYNRAM x A01

LISP Machine TV Board CADRTV;LMTV4B UML 07-DEC-80 2353
******* EDGE CONNECTIONS Flags: (# Output, @ Terminator, ---- Dedicated ground, ++++ Dedicated power) *******

-A-		-B-		-C-	-D-
A1 -XBUS35		A1 -XBUS5	#	A1	A1
A2 +5.0V++++++++++++++++++++		A2 +5.0V++++++++++++++++++++		A2 +5.0V++++++++++++++++++++	A2 +5.0V++++++++++++++++++++
B1 -XBUS34		B1 -XBUS4	#	B1	B1
B2		B2		B2	B2
C1 -XBUS33		C1 -XBUS3	#	C1	C1
C2 DEVADR 17----------		C2 DEVADR 17----------		C2 DEVADR 17----------	C2 DEVADR 17----------
D1 -XBUS32		D1 -XBUS2	#	D1	D1
D2 -XBUS31	#	D2 -XBUS1	#	D2 -XBUS.RQ H	# D2
E1 -XBUS30	#	E1 -XBUS0	#	E1	E1 -COLOR CLK H #
E2 -XBUS29	#	E2 -XBUS.PAR		E2 -XBUS.ACK H	# E2
F1 DEVADR 17		F1 DEVADR 17		F1 DEVADR 17	F1 DEVADR 17
F2 -XBUS28	#	F2 -XADDR.PAR		F2 -XBUS.WR H	# F2
H1 -XBUS27	#	H1 -XADDR 21 H		H1 -XBUS.EXTGRANT.IN	H1 -LOAD COLOR 0 H #
H2 -XBUS26	#	H2 -XADDR 20 H		H2 -XBUS.IGNPAR H	# H2
J1 -XBUS25	#	J1 -XADDR 19 H		J1 -XBUS.EXTGRANT.OUT	J1 -LOAD COLOR 1 H #
J2 -XBUS24	#	J2 -XADDR 18 H		J2 -XBUS.INIT H	# J2
K1 -XBUS23	#	K1 -XADDR 17 H		K1 XBUS.POWER.OK	K1 -LOAD COLOR 2 H #
K2 -XBUS22	#	K2 -XADDR 16 H		K2 -XBUS.EXTRQ	K2
L1 -XBUS21	#	L1 -XADDR 15 H		L1	L1
L2 -XBUS20	#	L2 -XADDR 14 H		L2 -XBUS.BUSY	L2
M1 -XBUS19	#	M1 -XADDR 13 H		M1	M1
M2 -XBUS18	#	M2 -XADDR 12 H		M2 -XBUS.SYNC	M2
N1 DEVADR 17		N1 DEVADR 17		N1 DEVADR 17	N1 DEVADR 17
N2 -XBUS17	#	N2 -XADDR 11 H		N2	N2
P1 -XBUS16	#	P1 -XADDR 10 H		P1	P1
P2 -XBUS15	#	P2 -XADDR 9 H		P2 -XBUS.INTR H	# P2
R1 -XBUS14	#	R1 -XADDR 8 H		R1	R1
R2 -XBUS13	#	R2 -XADDR 7 H		R2	R2
S1 -XBUS12	#	S1 -XADDR 6 H		S1	S1
S2 -XBUS11	#	S2 -XADDR 5 H		S2	S2
T1 DEVADR 17----------		T1 DEVADR 17----------		T1 DEVADR 17----------	T1 DEVADR 17----------
T2 -XBUS10	#	T2 -XADDR 4 H		T2	T2
U1 -XBUS9	#	U1 -XADDR 3 H		U1 -XBUS POWER RESET H	# U1
U2 -XBUS8	#	U2 -XADDR 2 H		U2	U2
V1 -XBUS7	#	V1 -XADDR 1 H		V1	V1
V2 -XBUS6	#	V2 -XADDR 0 H		V2	V2

LISP Machine TV Board CADRTV;LMTV4B UML 07-DEC-80 2353
******* EDGE CONNECTIONS Flags: (# Output, @ Terminator, ---- Dedicated ground, ++++ Dedicated power) *******

-E-		-F-		-J01-	-J02-
A1		A1		01	01
A2 +5.0V++++++++++++++++++		A2 +5.0V++++++++++++++++++		02	02
B1		B1		03	03
B2		B2		04	04
C1		C1		05	05
C2 DEVADR 17----------		C2 DEVADR 17----------		06	06
D1		D1 COMP VIDEO OUT		07	07
D2		D2		08	08
E1		E1 -BLANKING (FUDGED) H	#	09	09
E2		E2 TTL VIDEO DRIVE	#	10	10
F1 DEVADR 17		F1 DEVADR 17		11	11
F2		F2		12	12
H1 COLOR VALUE 0	#	H1 MECL VIDEO OUT	#	13	13
H2		H2		14	14
J1 COLOR VALUE 1	#	J1 -MECL VIDEO OUT H	#	15	15
J2		J2		16	16
K1 COLOR VALUE 2	#	K1 COLOR 0	#	17	17
K2		K2		18	18
L1 COLOR VALUE 3	#	L1 COLOR 1	#	19	19
L2		L2		20	20
M1 COLOR VALUE 4	#	M1 COLOR 2	#	21	21
M2		M2		22	22
N1 DEVADR 17		N1 DEVADR 17		23	23
N2		N2		24	24
P1 COLOR VALUE 5	#	P1 COLOR 3	#	25	25
P2		P2		26 ----------	26 ----------
R1 COLOR VALUE 6	#	R1 COLOR 4	#	27 ----------	27 ----------
R2		R2		28 ----------	28 ----------
S1 COLOR VALUE 7	#	S1 COLOR 5	#	29 ----------	29 ----------
S2		S2		30 ----------	30 ----------
T1 DEVADR 17----------		T1 DEVADR 17----------		31 ----------	31 ----------
T2		T2		32 ----------	32 ----------
U1 HSYNC OUT	#	U1 COLOR 6	#	33 ----------	33 ----------
U2		U2		34 ----------	34 ----------
V1 VSYNC OUT	#	V1 COLOR 7	#	35 ----------	35 ----------
V2		V2		36 ----------	36 ----------
				37 ----------	37 ----------
				38 ----------	38 ----------
				39 ----------	39 ----------
				40 ----------	40 ----------
				41 ----------	41 ----------
				42 ----------	42 ----------
				43 ----------	43 ----------
				44 ----------	44 ----------
				45 ----------	45 ----------
				46 ----------	46 ----------
				47 ----------	47 ----------
				48 ----------	48 ----------
				49 ----------	49 ----------
				50 ----------	50 ----------

LISP Machine TV Board CADRTV;LMTV4B UML 07-DEC-80 2354
******* EDGE CONNECTIONS Flags: (# Output, @ Terminator, ---- Dedicated ground, ++++ Dedicated power) *******

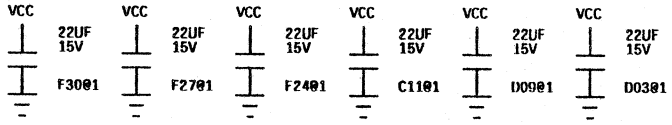
-J03-	-J04-	-J05-	-J06-
01 02	01 02	01 02	01 02
03 04	03 04	03 04	03 04
05 06	05 06	05 06	05 06
07 08	07 08	07 08	07 08
09 10	09 10	09 10	09 10
11 12	11 12	11 12	11 12
13 14	13 14	13 14	13 14
15 16	15 16	15 16	15 16
17 18	17 18	17 18	17 18
19 20	19 20	19 20	19 20
21 ---------- 22 ----------	21 ---------- 22 ----------	21 22	21 22
23 ---------- 24 ----------	23 ---------- 24 ----------	23 24	23 24
25 ---------- 26 ----------	25 ---------- 26 ----------	25 26 ----------	25 26 ----------
27 ---------- 28 ----------	27 ---------- 28 ----------	27 ---------- 28 ----------	27 ---------- 28 ----------
29 ---------- 30 ----------	29 ---------- 30 ----------	29 ---------- 30 ----------	29 ---------- 30 ----------
31 ---------- 32 ----------	31 ---------- 32 ----------	31 ---------- 32 ----------	31 ---------- 32 ----------
33 ---------- 34 ----------	33 ---------- 34 ----------	33 ---------- 34 ----------	33 ---------- 34 ----------
35 ---------- 36 ----------	35 ---------- 36 ----------	35 ---------- 36 ----------	35 ---------- 36 ----------
37 ---------- 38 ----------	37 ---------- 38 ----------	37 ---------- 38 ----------	37 ---------- 38 ----------
39 ---------- 40 ----------	39 ---------- 40 ----------	39 ---------- 40 ----------	39 ---------- 40 ----------
		41 ---------- 42 ----------	41 ---------- 42 ----------
		43 ---------- 44 ----------	43 ---------- 44 ----------
		45 ---------- 46 ----------	45 ---------- 46 ----------
		47 ---------- 48 ----------	47 ---------- 48 ----------
		49 ---------- 50 ----------	49 ---------- 50 ----------

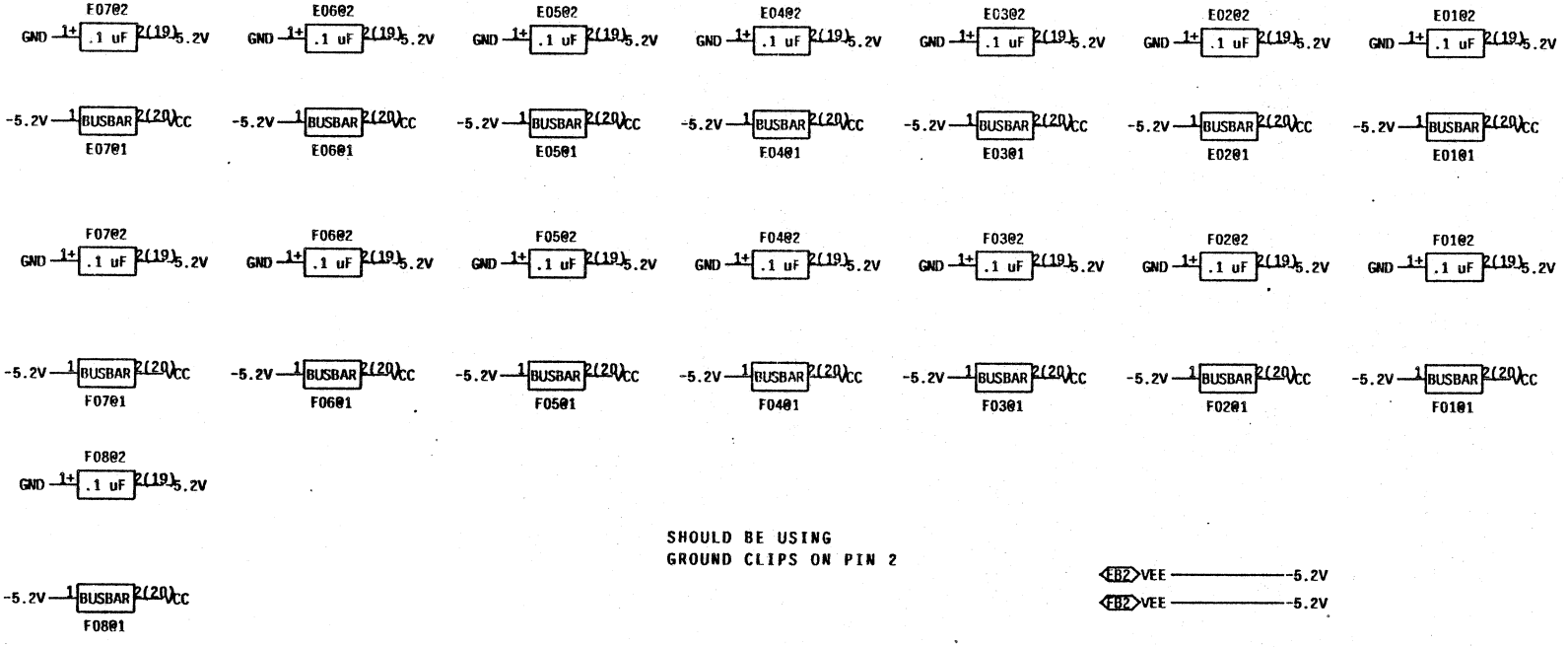
LISP Machine TV Board CADRTV;LMTV4B UML 07-DEC-80 2354
******* EDGE CONNECTIONS Flags: (# Output, @ Terminator, ---- Dedicated ground, ++++ Dedicated power) *******

-J07-	-J08-	-J09-	-J10-
01 02	01 02	01 02	01 02
03 04	03 04	03 04	03 04
05 06	05 06	05 06	05 06
07 08	07 08	07 08	07 08
09 10	09 10	09 10	09 10
11 12	11 12	11 12	11 12
13 14	13 14	13 14	13 14
15 16	15 16	15 16	15 16
17 18	17 18	17 18	17 18
19 20	19 20	19 20	19 20
21 ---------- 22 ----------	21 ---------- 22 ----------	21 22	21 22
23 ---------- 24 ----------	23 ---------- 24 ----------	23 24	23 24
25 ---------- 26 ----------	25 ---------- 26 ----------	25 26 ----------	25 26 ----------
27 ---------- 28 ----------	27 ---------- 28 ----------	27 ---------- 28 ----------	27 ---------- 28 ----------
29 ---------- 30 ----------	29 ---------- 30 ----------	29 ---------- 30 ----------	29 ---------- 30 ----------
31 ---------- 32 ----------	31 ---------- 32 ----------	31 ---------- 32 ----------	31 ---------- 32 ----------
33 ---------- 34 ----------	33 ---------- 34 ----------	33 ---------- 34 ----------	33 ---------- 34 ----------
35 ---------- 36 ----------	35 ---------- 36 ----------	35 ---------- 36 ----------	35 ---------- 36 ----------
37 ---------- 38 ----------	37 ---------- 38 ----------	37 ---------- 38 ----------	37 ---------- 38 ----------
39 ---------- 40 ----------	39 ---------- 40 ----------	39 ---------- 40 ----------	39 ---------- 40 ----------
		41 ---------- 42 ----------	41 ---------- 42 ----------
		43 ---------- 44 ----------	43 ---------- 44 ----------
		45 ---------- 46 ----------	45 ---------- 46 ----------
		47 ---------- 48 ----------	47 ---------- 48 ----------
		49 ---------- 50 ----------	49 ---------- 50 ----------

LISP Machine TV Board CADRTV;LMTV4B UML 07-DEC-80 2354
******* EDGE CONNECTIONS Flags: (# Output, @ Terminator, ---- Dedicated ground, ++++ Dedicated power) *******

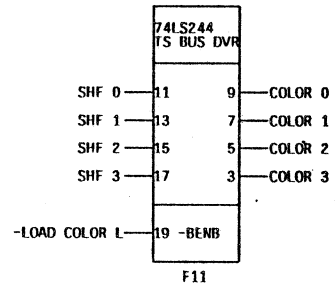
-J11-		-J12-	
01	02	01	02
03	04	03	04
05	06	05	06
07	08	07	08
09	10	09	10
11	12	11	12
13	14	13	14
15	16	15	16
17	18	17	18
19	20	19	20
21 ----------	22 ----------	21 ----------	22 ----------
23 ----------	24 ----------	23 ----------	24 ----------
25 ----------	26 ----------	25 ----------	26 ----------
27 ----------	28 ----------	27 ----------	28 ----------
29 ----------	30 ----------	29 ----------	30 ----------
31 ----------	32 ----------	31 ----------	32 ----------
33 ----------	34 ----------	33 ----------	34 ----------
35 ----------	36 ----------	35 ----------	36 ----------
37 ----------	38 ----------	37 ----------	38 ----------
39 ----------	40 ----------	39 ----------	40 ----------

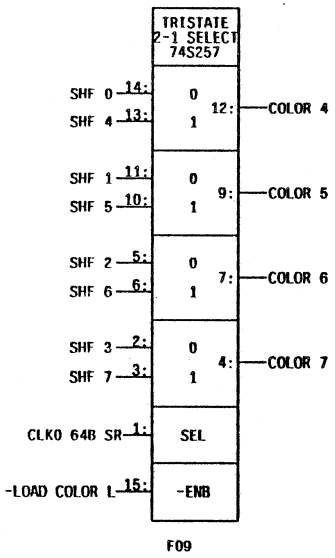


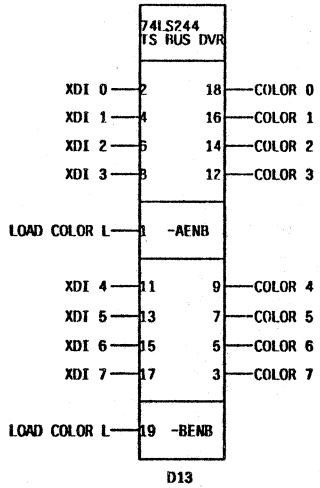


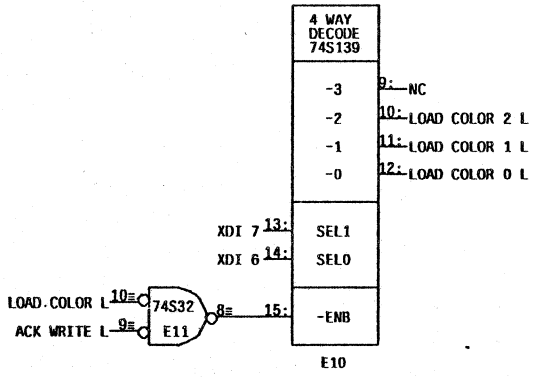
SHOULD BE USING
GROUND CLIPS ON PIN 2

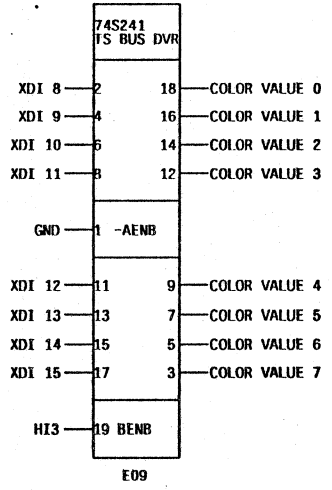
USE .1uF DIP CAPS NEXT TO EVERY OTHER MEMORY CHIP

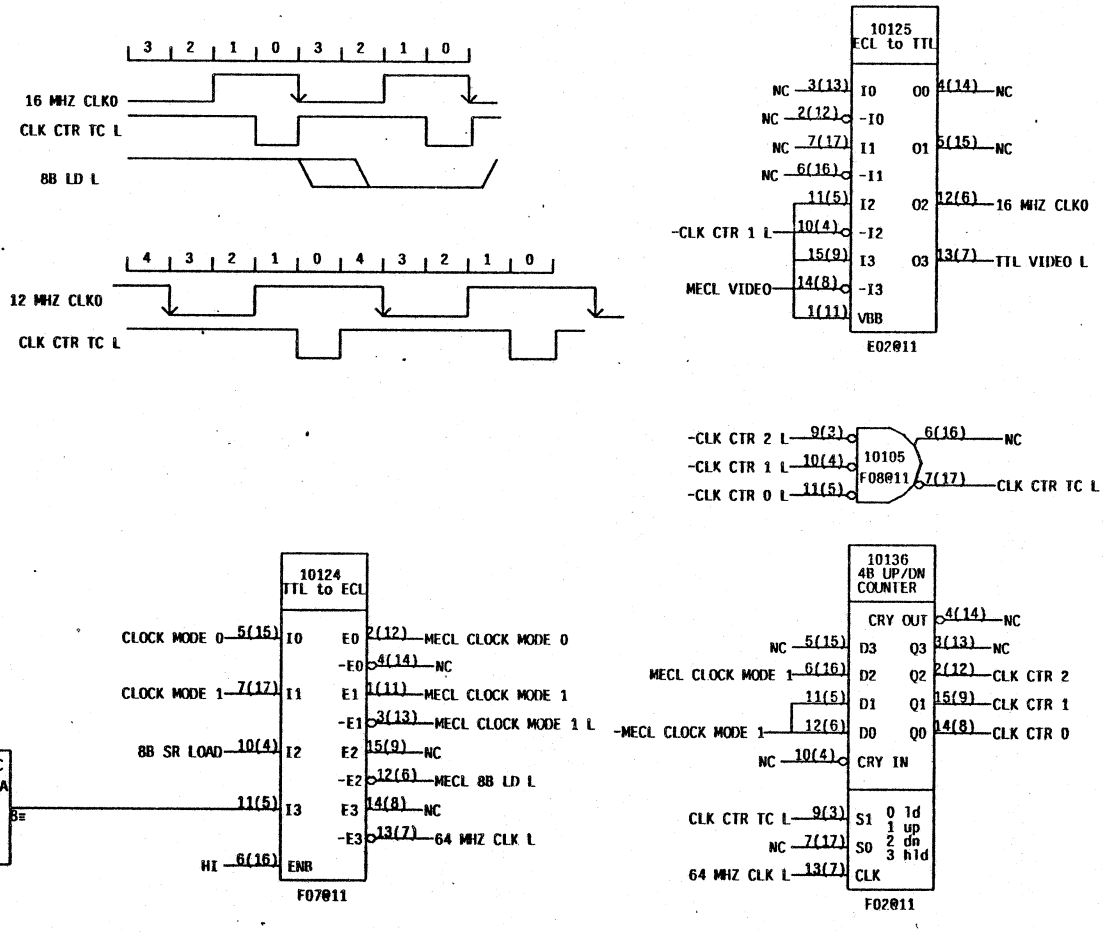

74LS244
TS BUS DVR
SHF 0
SHF 1
SHF 2
SHF 3
-LOAD COLOR L
11
13
15
17
19 -BENB
9
7
5
3
COLOR 0
COLOR 1
COLOR 2
COLOR 3
F11


TRISTATE
2-1 SELECT
74S257
SHF 0
SHF 4
SHF 1
SHF 5
SHF 2
SHF 6
SHF 3
SHF 7
CLK0 64B SR
-LOAD COLOR L
14:
13:
11:
10:
5:
6:
2:
3:
1:
15:
0
1
SEL
-ENB
12:
9:
7:
4:
COLOR 4
COLOR 5
COLOR 6
COLOR 7
F09


74LS244
TS BUS DVR
XDI 0
XDI 1
XDI 2
XDI 3
LOAD COLOR L
XDI 4
XDI 5
XDI 6
XDI 7
LOAD COLOR L
2
4
6
8
1 -AENB
11
13
15
17
19 -BENB
18
16
14
12
9
7
5
3
COLOR 0
COLOR 1
COLOR 2
COLOR 3
COLOR 4
COLOR 5
COLOR 6
COLOR 7
D13

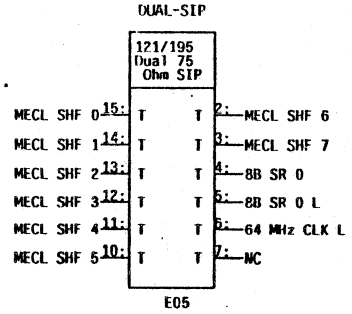

4 WAY
DECODE
74S139
-3
-2
-1
-0
9:
10:
11:
12:
NC
LOAD COLOR 2 L
LOAD COLOR 1 L
LOAD COLOR 0 L
XDI 7
XDI 6
13:
14:
SEL1
SEL0
-ENB
15:
LOAD COLOR L
ACK WRITE L
10≡
9≡
74S32
E11
8≡
E10


74S241
TS BUS DVR
XDI 8
XDI 9
XDI 10
XDI 11
GND
XDI 12
XDI 13
XDI 14
XDI 15
HI3
2
4
6
8
1 -AENB
11
13
15
17
19 BENB
18
16
14
12
9
7
5
3
COLOR VALUE 0
COLOR VALUE 1
COLOR VALUE 2
COLOR VALUE 3
COLOR VALUE 4
COLOR VALUE 5
COLOR VALUE 6
COLOR VALUE 7
E09

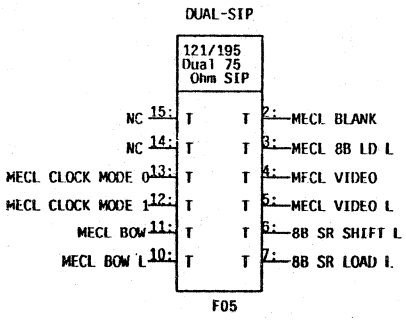


LISPM TV	MECL Clock	7-DEC-1980 22:49	AI: CADRTV; ECLCLK

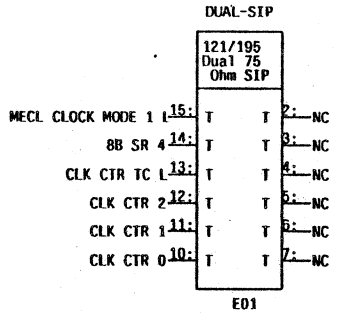
NOTE: Install with pin 1 down



NOTE: Install with pin 1 down

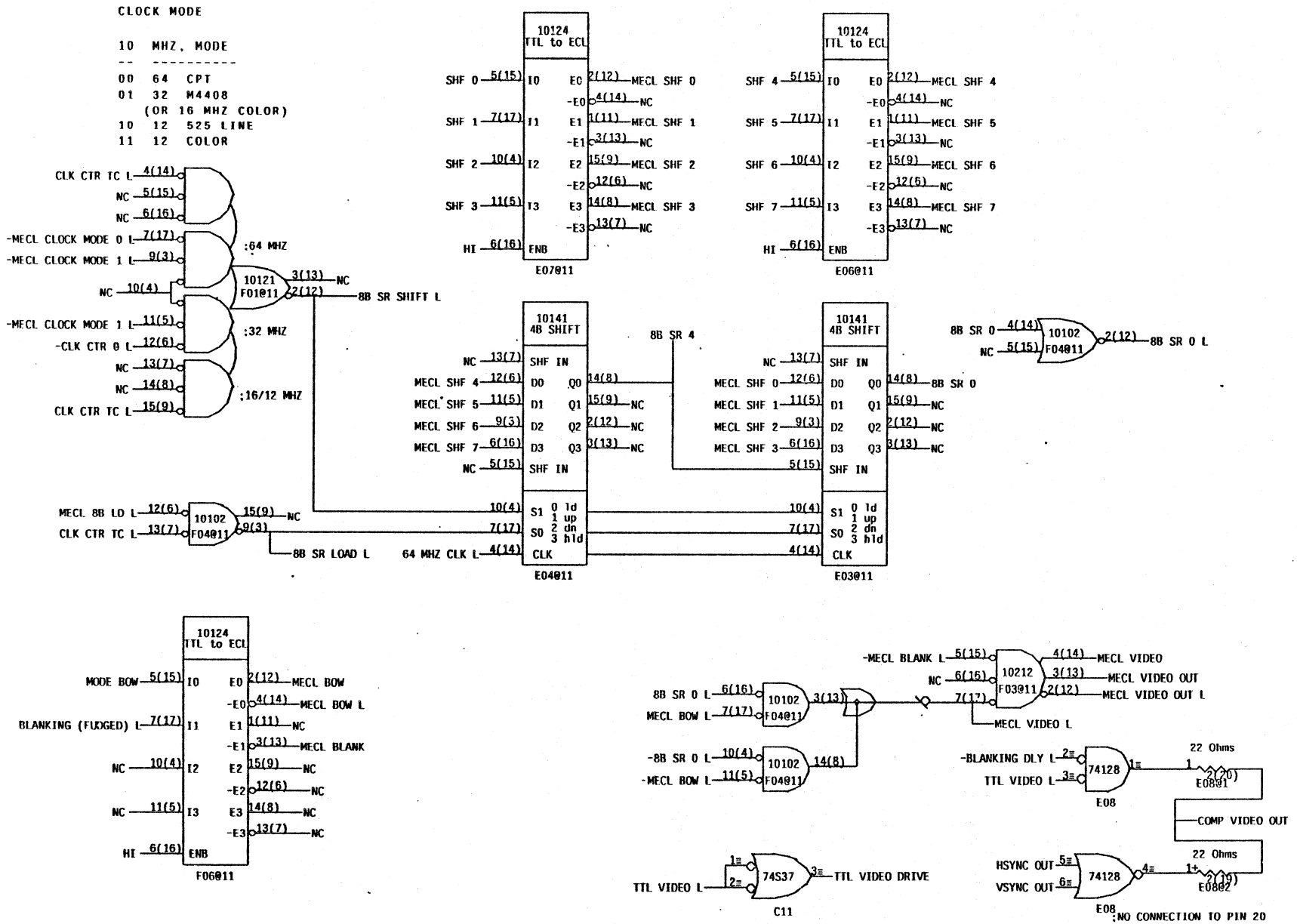


NOTE: Install with pin 1 down

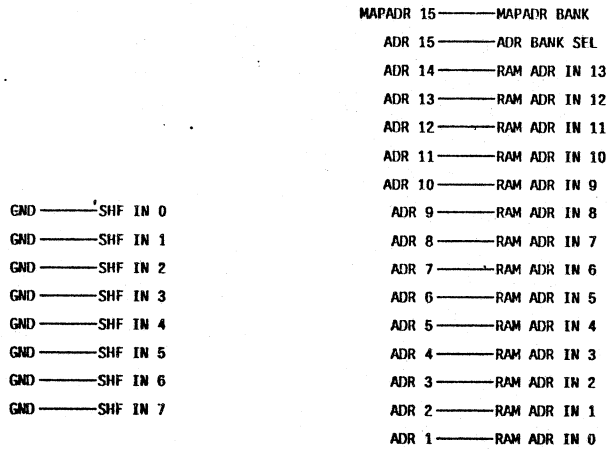


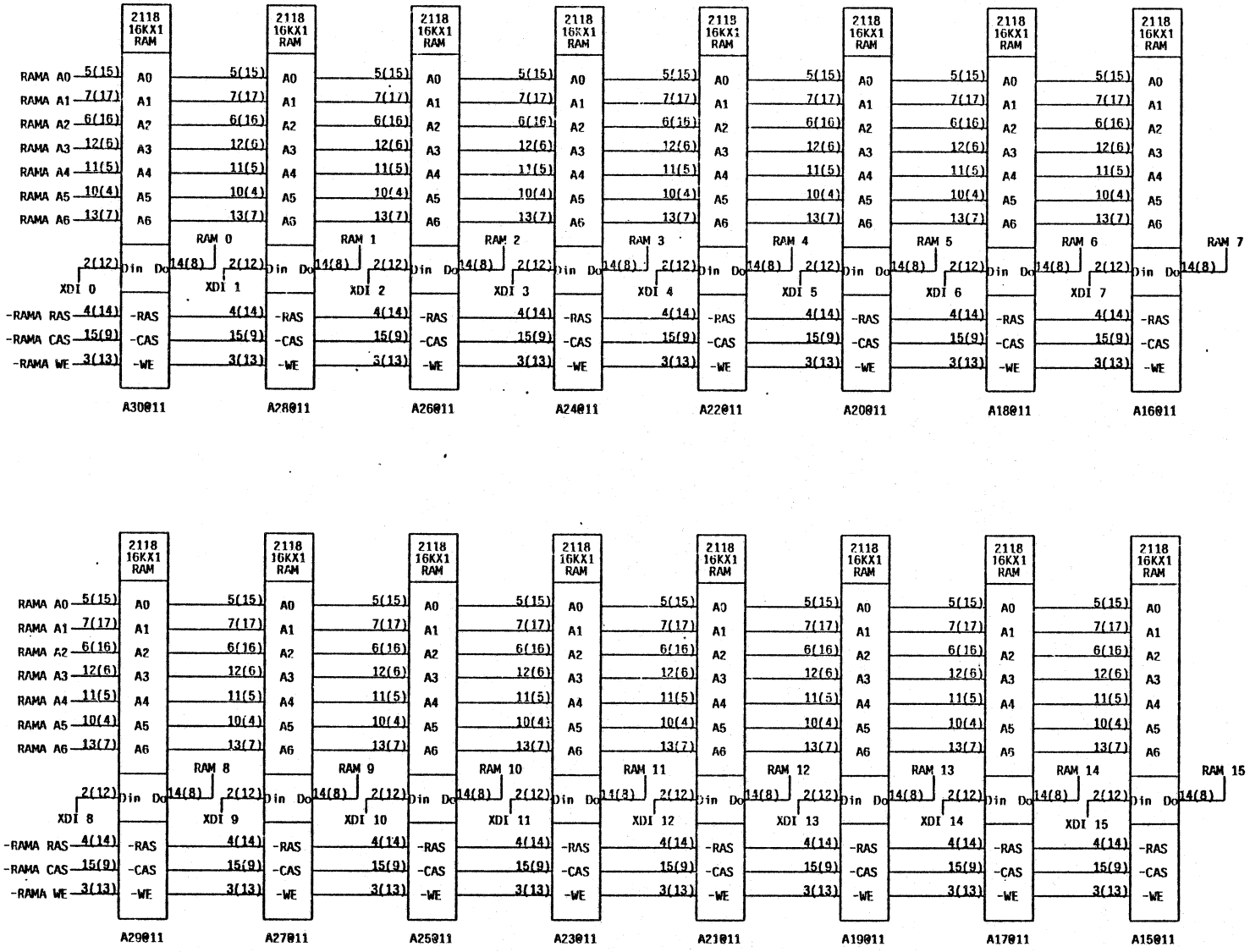
INSTALL 121 OHM SIDE OF SIP CONNECTED TO PIN 10

THE "NC" SIP PINS SHOULD MEASURE -2 VOLTS

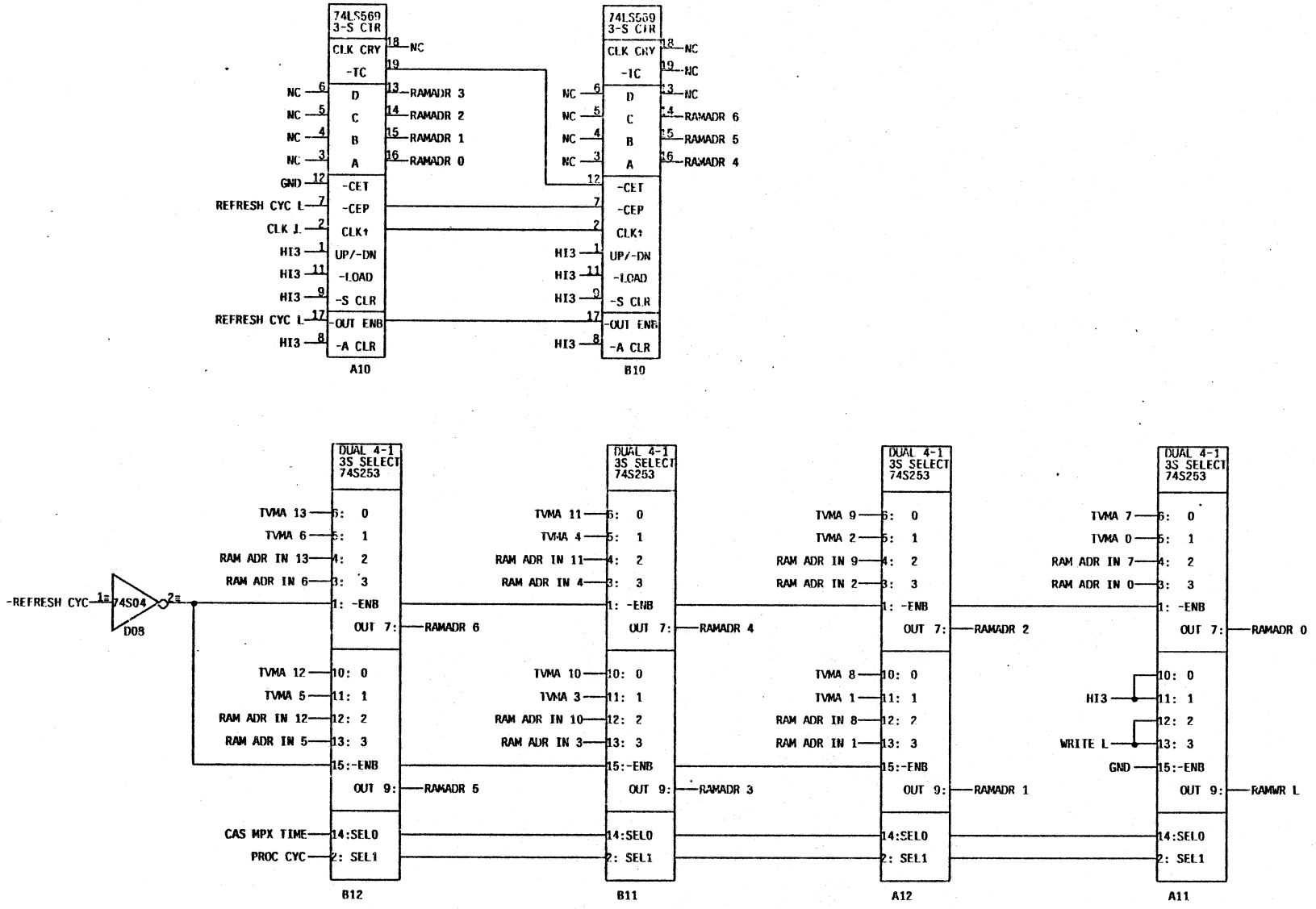


LISPM TV	MECL Video	7-DEC-1980 22:44	AI: CADRTV; ECLVID

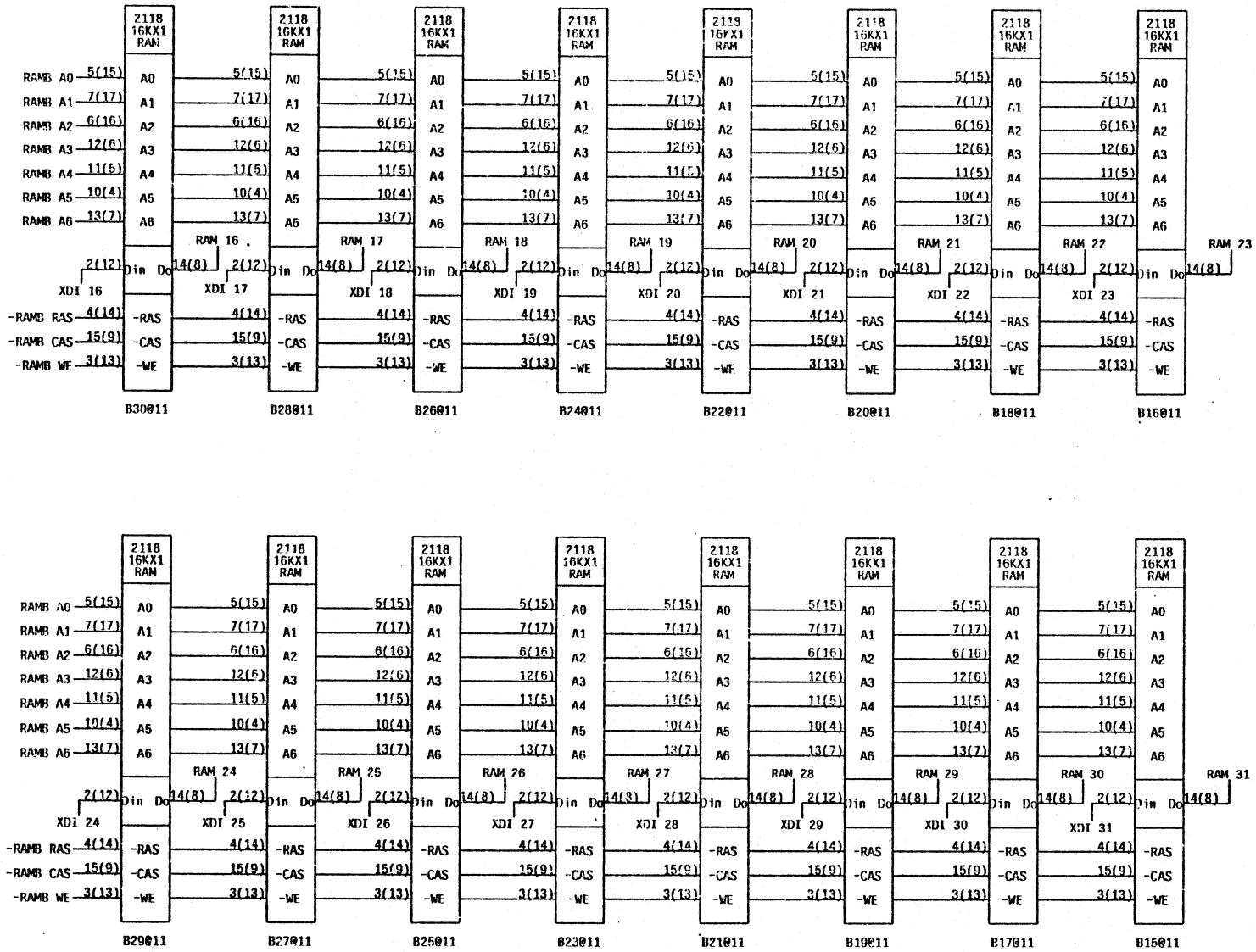

MAPADR 15 MAPADR BANK
ADR 15 ADR BANK SEL
ADR 14 RAM ADR IN 13
ADR 13 RAM ADR IN 12
ADR 12 RAM ADR IN 11
ADR 11 RAM ADR IN 10
ADR 10 RAM ADR IN 9
ADR 9 RAM ADR IN 8
ADR 8 RAM ADR IN 7
ADR 7 RAM ADR IN 6
ADR 6 RAM ADR IN 5
ADR 5 RAM ADR IN 4
ADR 4 RAM ADR IN 3
ADR 3 RAM ADR IN 2
ADR 2 RAM ADR IN 1
ADR 1 RAM ADR IN 0
GND SHF IN 0
GND SHF IN 1
GND SHF IN 2
GND SHF IN 3
GND SHF IN 4
GND SHF IN 5
GND SHF IN 6
GND SHF IN 7



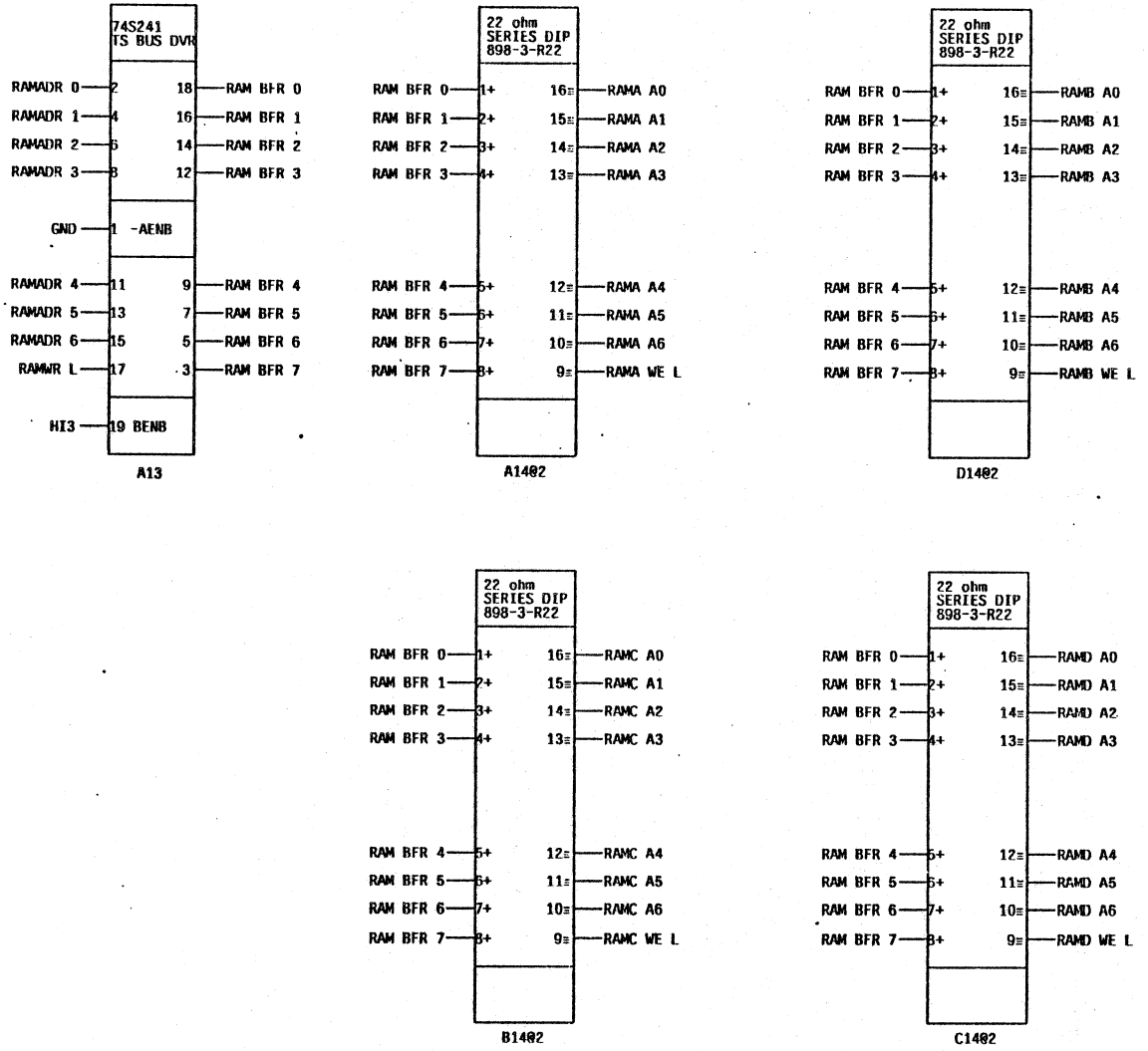
LISPM TV	Video Memory: Row A	7-DEC-1980 19:49	AI: CADRTV; RAMA

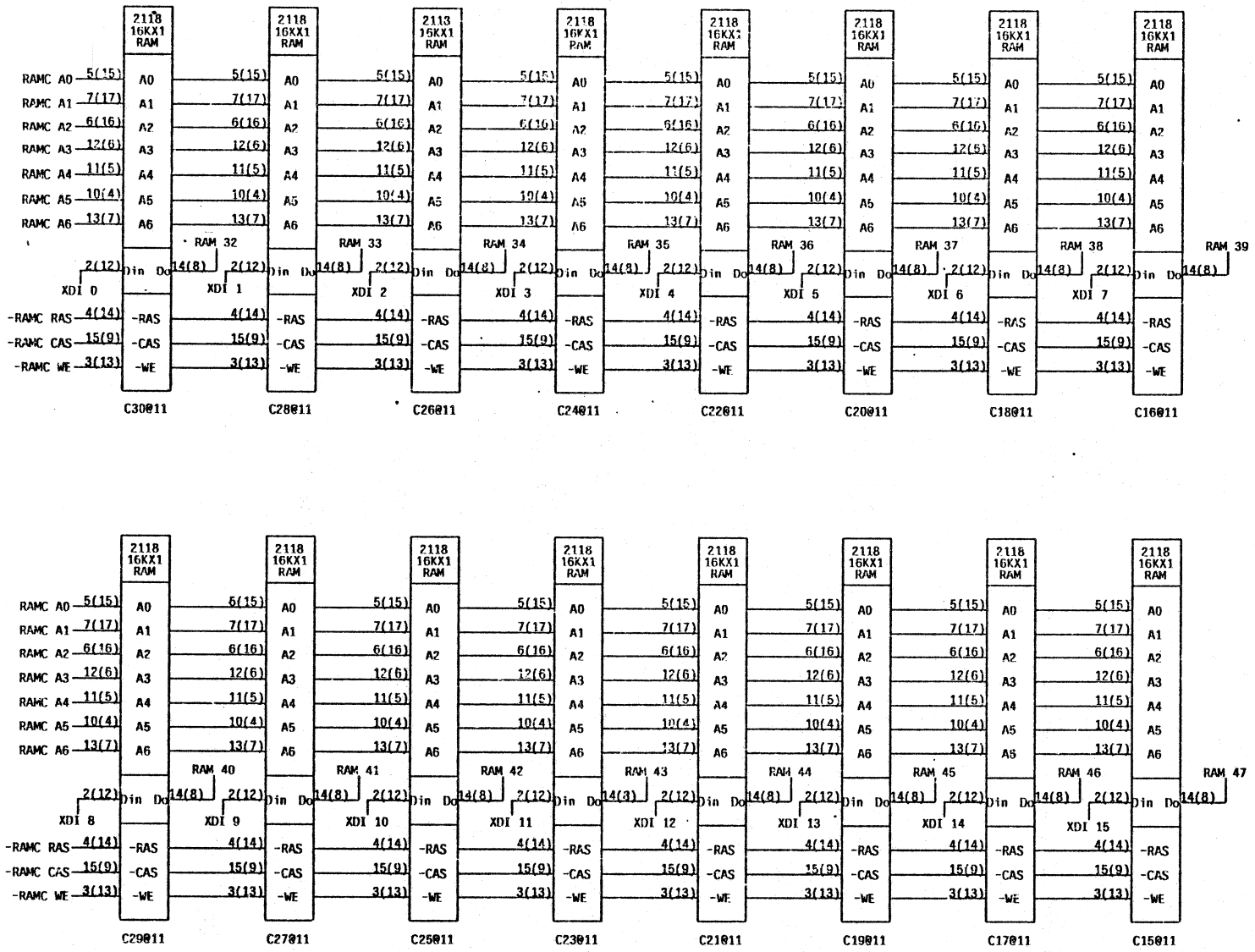


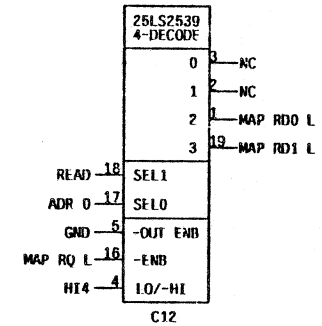
LISPM TV	Ram Address Logic	7-DEC-1980 22:50	AI: CADRTV; RAMADR

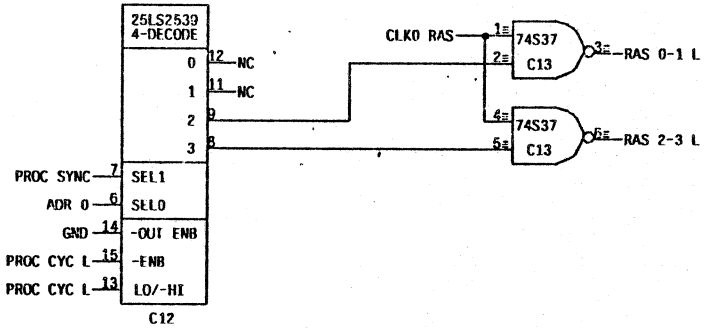


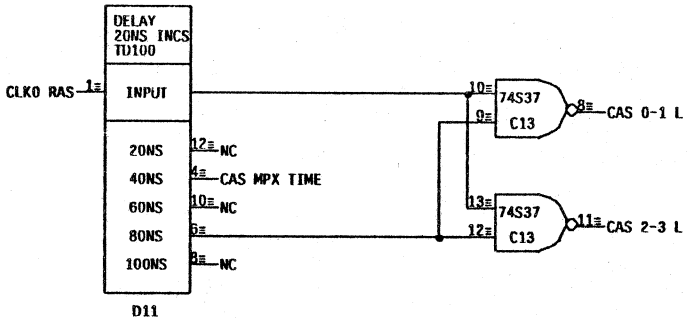
LISPM TV	Video Memory: Row B	7-DEC-1980 19:50	AI: CADRTV; RAMB

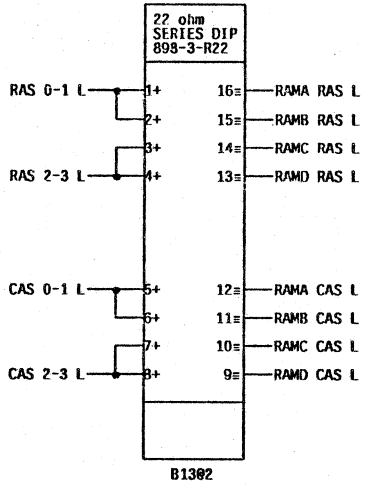

74S241
TS BUS DVR
RAMADR 0
RAMADR 1
RAMADR 2
RAMADR 3
GND
RAMADR 4
RAMADR 5
RAMADR 6
RAMWR L
HI3
-AENB
BENB
RAM BFR 0
RAM BFR 1
RAM BFR 2
RAM BFR 3
RAM BFR 4
RAM BFR 5
RAM BFR 6
RAM BFR 7
A13
22 ohm
SERIES DIP
898-3-R22
RAMA A0
RAMA A1
RAMA A2
RAMA A3
RAMA A4
RAMA A5
RAMA A6
RAMA WE L
A1402
RAMB A0
RAMB A1
RAMB A2
RAMB A3
RAMB A4
RAMB A5
RAMB A6
RAMB WE L
D1402
RAMC A0
RAMC A1
RAMC A2
RAMC A3
RAMC A4
RAMC A5
RAMC A6
RAMC WE L
B1402
RAMD A0
RAMD A1
RAMD A2
RAMD A3
RAMD A4
RAMD A5
RAMD A6
RAMD WE L
C1402

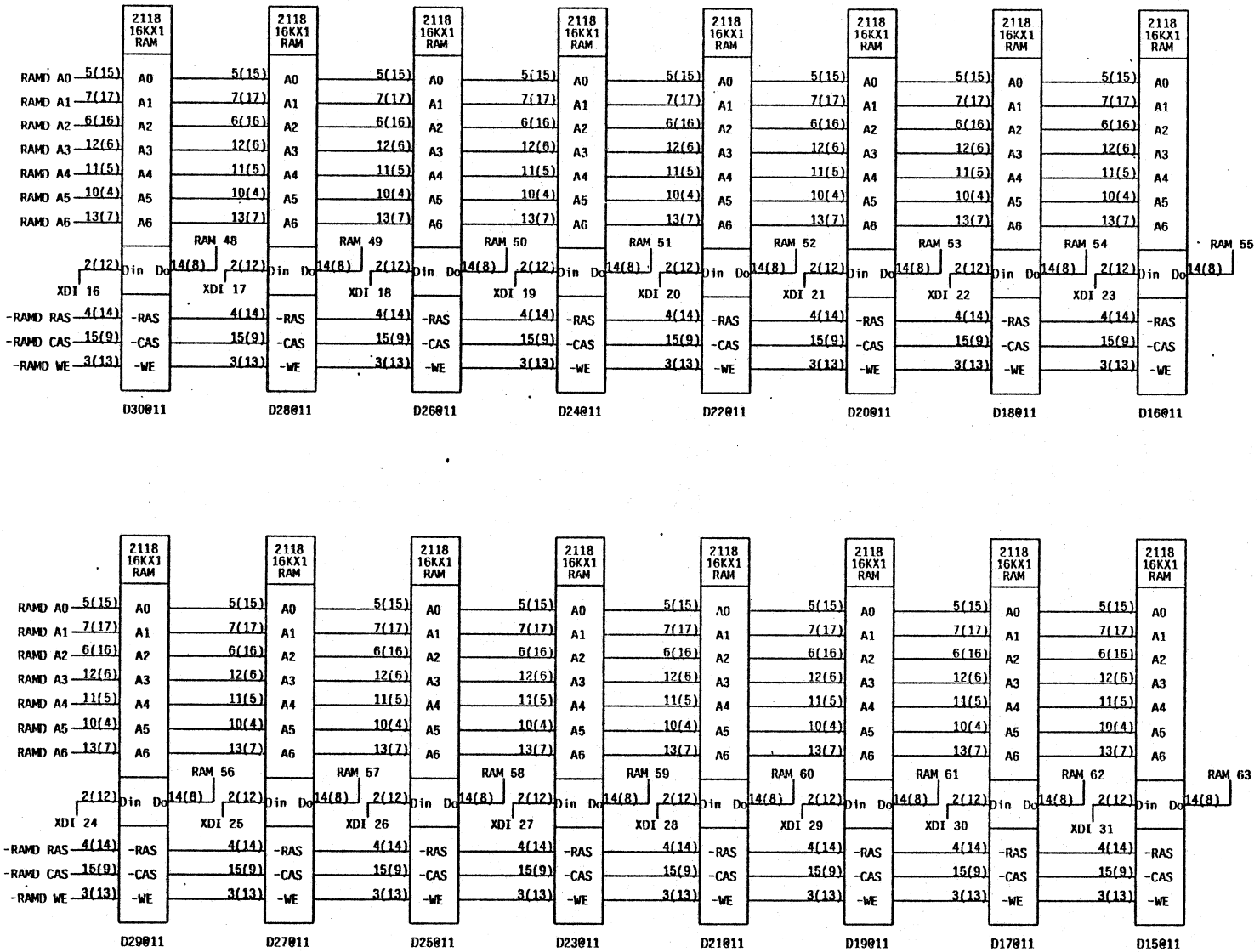

2118 16KX1 RAM
RAMC A0
RAMC A1
RAMC A2
RAMC A3
RAMC A4
RAMC A5
RAMC A6
-RAMC RAS
-RAMC CAS
-RAMC WE
A0
A1
A2
A3
A4
A5
A6
-RAS
-CAS
-WE
Din
Do
XDI 0
XDI 1
XDI 2
XDI 3
XDI 4
XDI 5
XDI 6
XDI 7
XDI 8
XDI 9
XDI 10
XDI 11
XDI 12
XDI 13
XDI 14
XDI 15
RAM 32
RAM 33
RAM 34
RAM 35
RAM 36
RAM 37
RAM 38
RAM 39
RAM 40
RAM 41
RAM 42
RAM 43
RAM 44
RAM 45
RAM 46
RAM 47
C30@11
C28@11
C26@11
C24@11
C22@11
C20@11
C18@11
C16@11
C29@11
C27@11
C25@11
C23@11
C21@11
C19@11
C17@11
C15@11


25LS2539
4-DECODE
0
1
2
3
NC
NC
MAP RD0 L
MAP RD1 L
READ
ADR 0
SEL1
SEL0
GND
-OUT ENB
MAP RQ L
-ENB
HI4
LO/-HI
C12

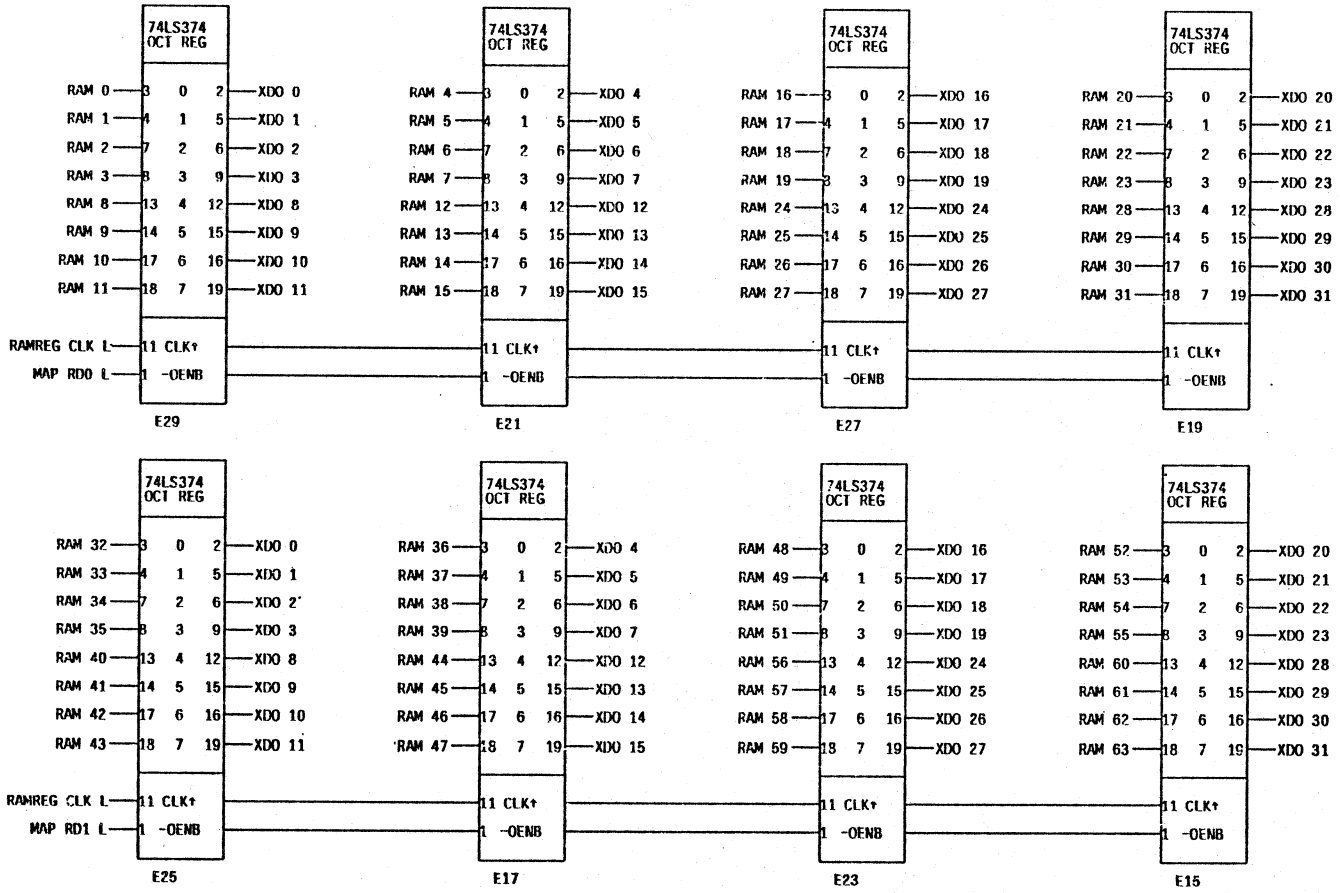

25LS2539
4-DECODE
0
1
2
3
NC
NC
PROC SYNC
SEL1
ADR 0
SEL0
GND
-OUT ENB
PROC CYC L
-ENB
PROC CYC L
LO/-HI
C12
CLK0 RAS
74S37
C13
RAS 0-1 L
74S37
C13
RAS 2-3 L


DELAY
20NS INCS
TD100
CLK0 RAS
INPUT
20NS
40NS
60NS
80NS
100NS
NC
CAS MPX TIME
NC
NC
D11
74S37
C13
CAS 0-1 L
74S37
C13
CAS 2-3 L

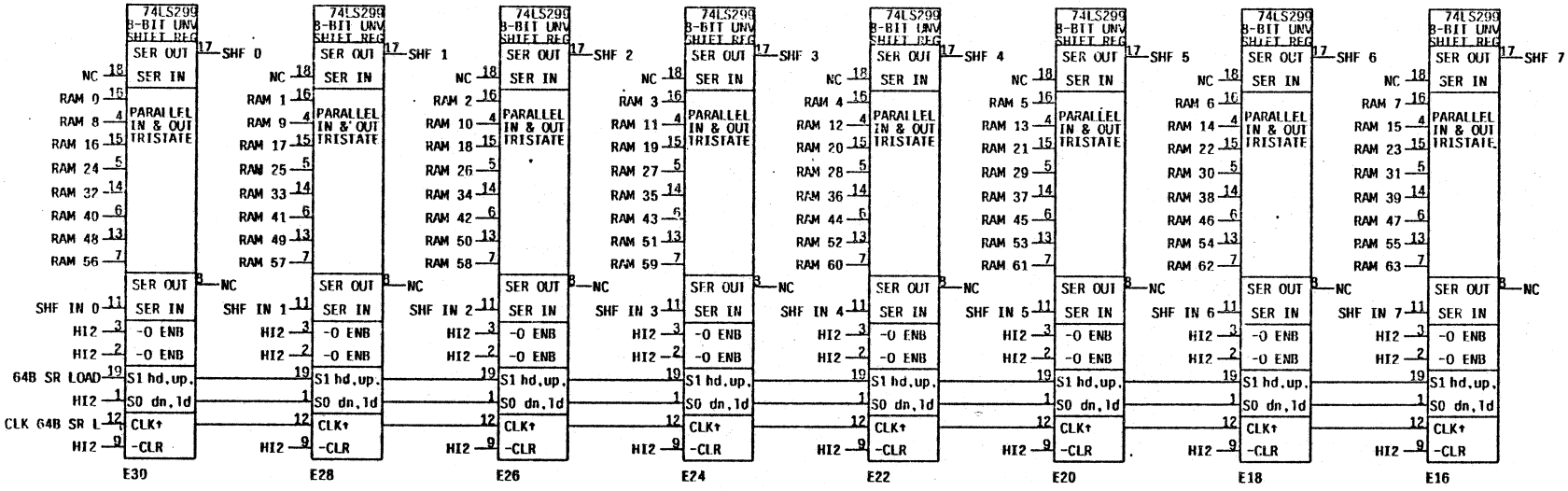

22 ohm
SERIES DIP
898-3-R22
RAS 0-1 L
RAS 2-3 L
CAS 0-1 L
CAS 2-3 L
RAMA RAS L
RAMB RAS L
RAMC RAS L
RAMD RAS L
RAMA CAS L
RAMB CAS L
RAMC CAS L
RAMD CAS L
B1302


2118 16KX1 RAM
RAMD A0
RAMD A1
RAMD A2
RAMD A3
RAMD A4
RAMD A5
RAMD A6
-RAMD RAS
-RAMD CAS
-RAMD WE
A0
A1
A2
A3
A4
A5
A6
-RAS
-CAS
-WE
Din
Do
RAM 48
RAM 49
RAM 50
RAM 51
RAM 52
RAM 53
RAM 54
RAM 55
XDI 16
XDI 17
XDI 18
XDI 19
XDI 20
XDI 21
XDI 22
XDI 23
D30@11
D28@11
D26@11
D24@11
D22@11
D20@11
D18@11
D16@11

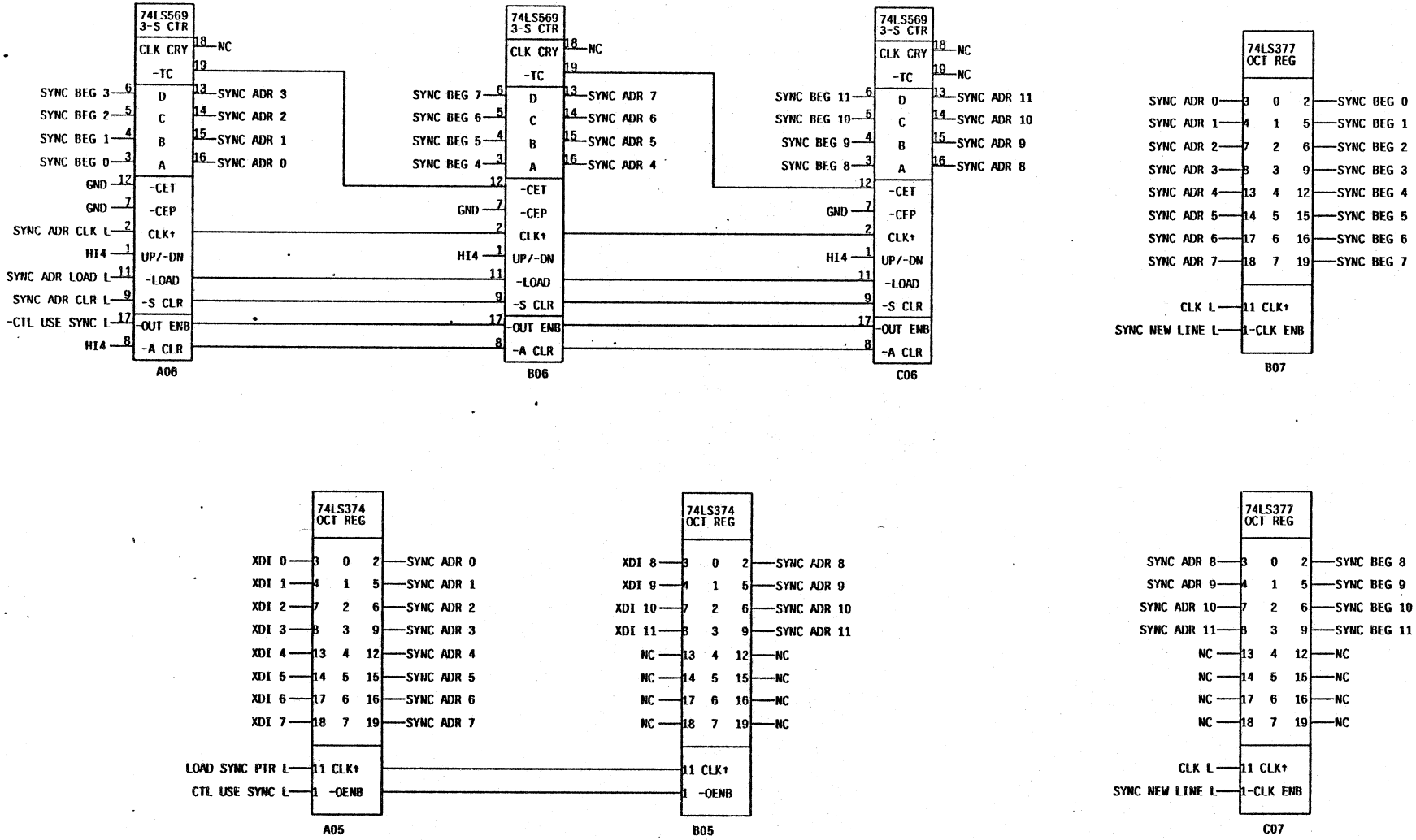
2118 16KX1 RAM
RAMD A0
RAMD A1
RAMD A2
RAMD A3
RAMD A4
RAMD A5
RAMD A6
-RAMD RAS
-RAMD CAS
-RAMD WE
A0
A1
A2
A3
A4
A5
A6
-RAS
-CAS
-WE
Din
Do
RAM 56
RAM 57
RAM 58
RAM 59
RAM 60
RAM 61
RAM 62
RAM 63
XDI 24
XDI 25
XDI 26
XDI 27
XDI 28
XDI 29
XDI 30
XDI 31
D29@11
D27@11
D25@11
D23@11
D21@11
D19@11
D17@11
D15@11



LISPM TV	Ram Output Registers	7-DEC-1980 23:02	AI: CADRTV; RAMREG



LISPM TV	64-bit Video Shift Register	7-DEC-1980 22:52	AI: CADRTV; RAMSHF

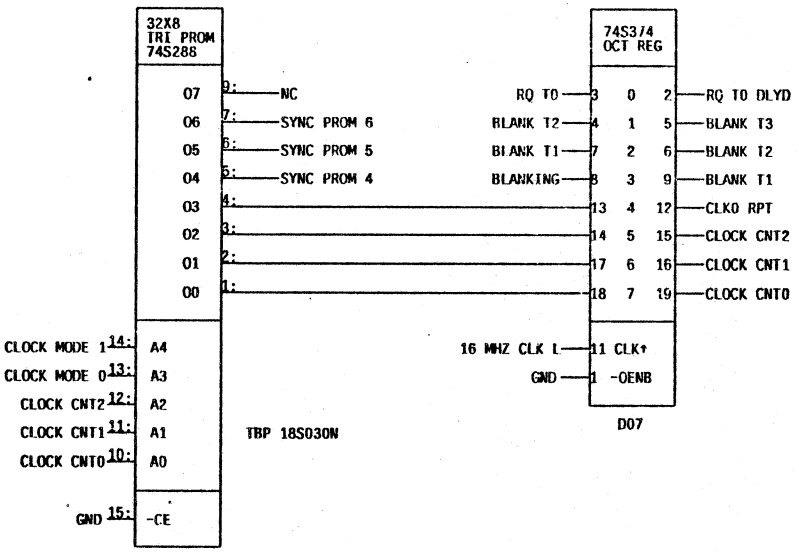


LISPM TV	Sync Program Address	7-DEC-1980 22:53	AI: CADRTV; SYNADR

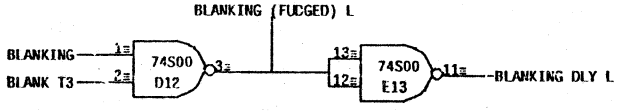
ON CADRTV;NXBCTL

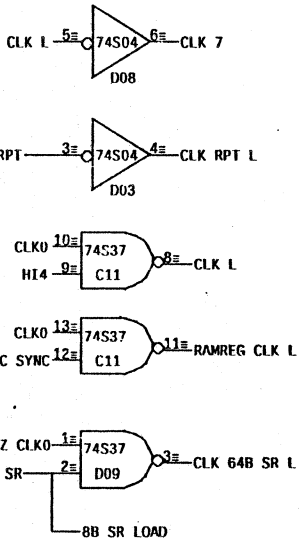
SYNC PROM

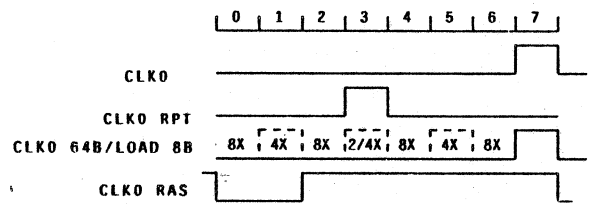
4 CLKO
5 CLKO 65B SR
7 CLKO RAS
7 ---



LMTV4B OR LMTV8B



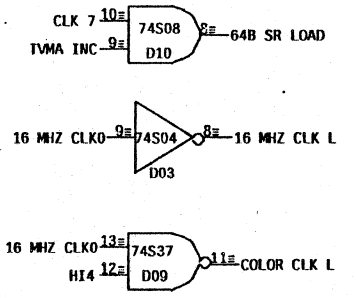




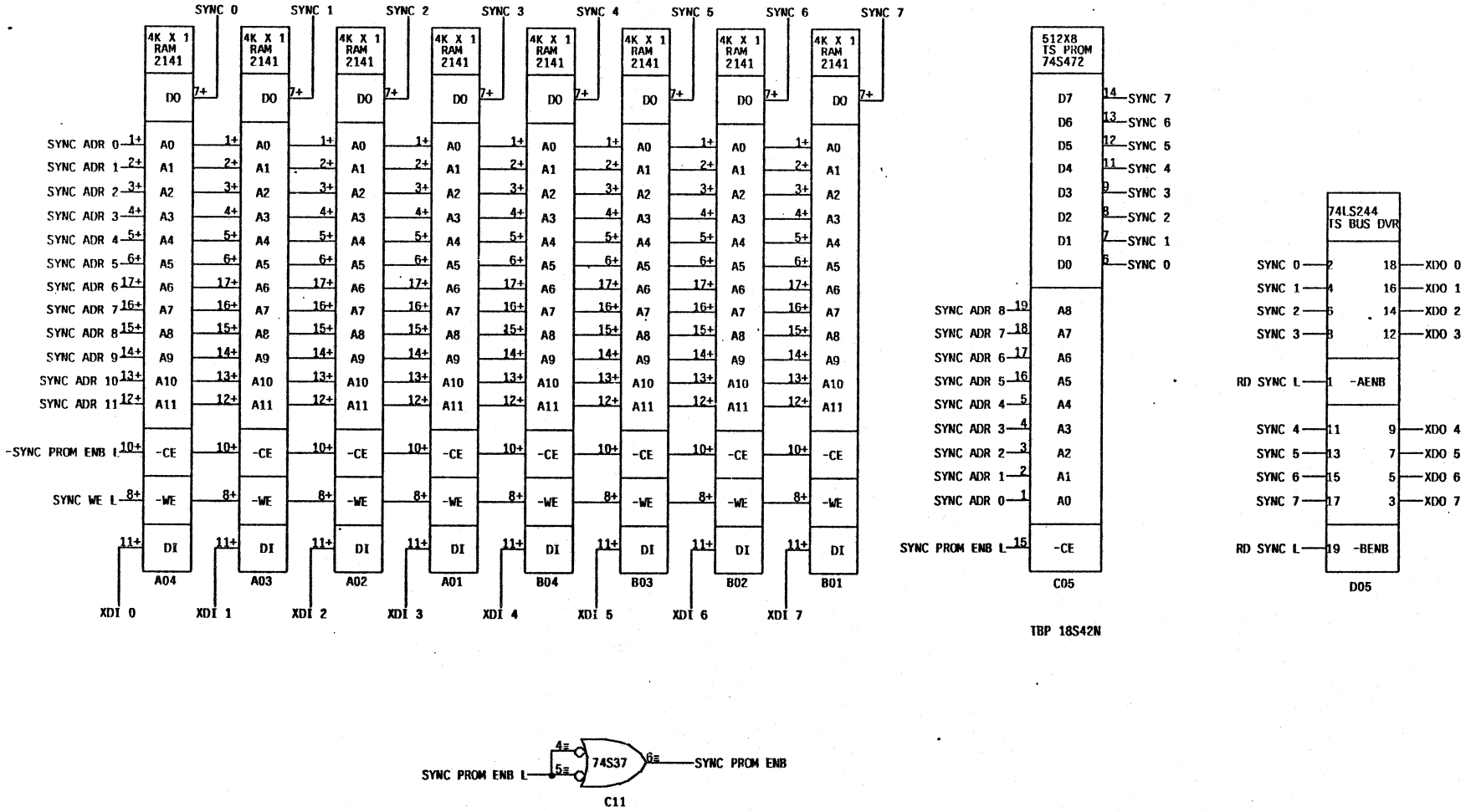
CLOCK MODE

10 MHZ,	MODE
00 64	CPT
01 32	M4408
(OR 16 MHZ COLOR)	
10 12	525 LINE
11 12	COLOR

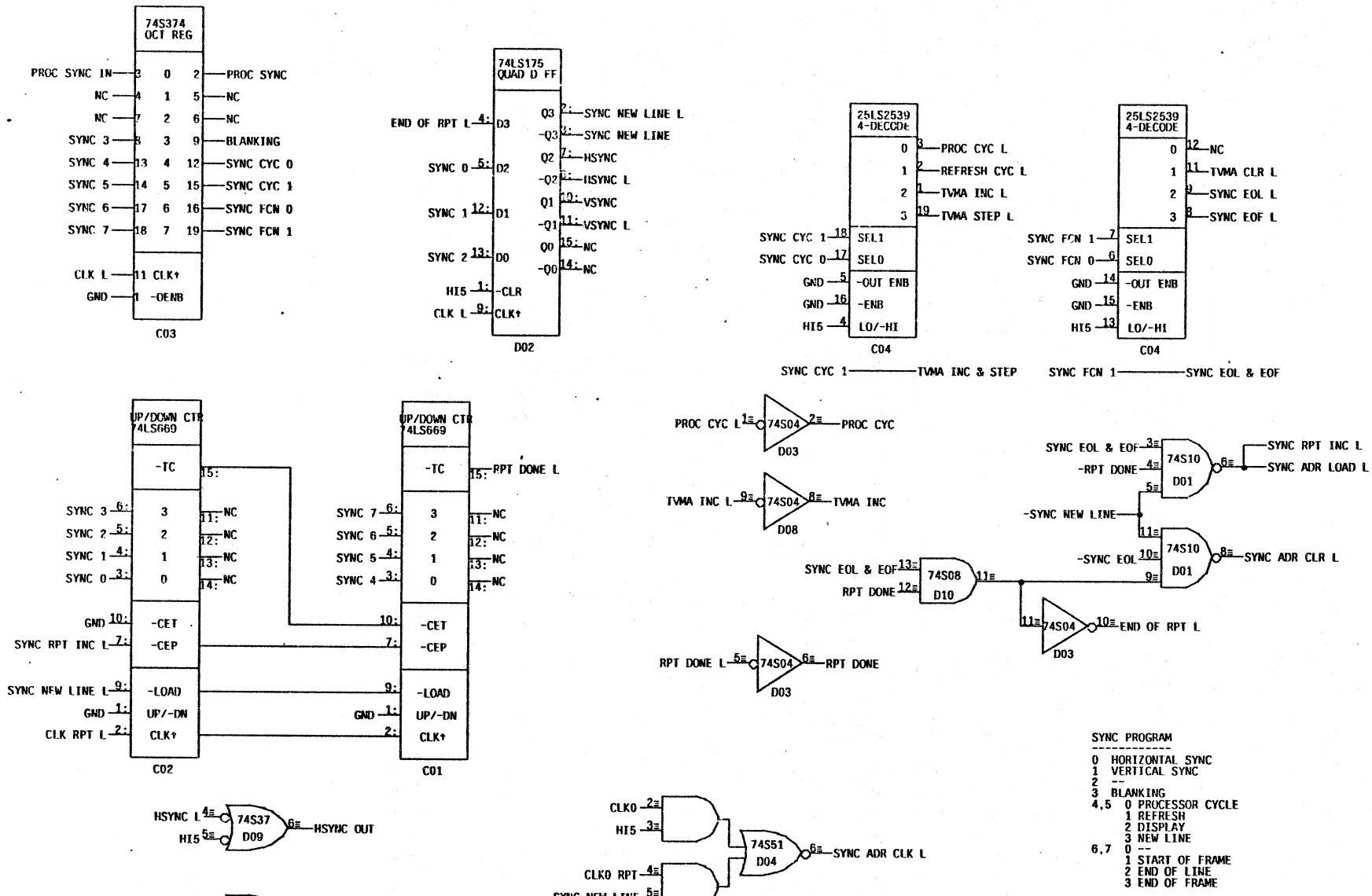
CLK 7 DELAYED FROM CLKO
ALLOW FOR HOLD TIME OF 74S299



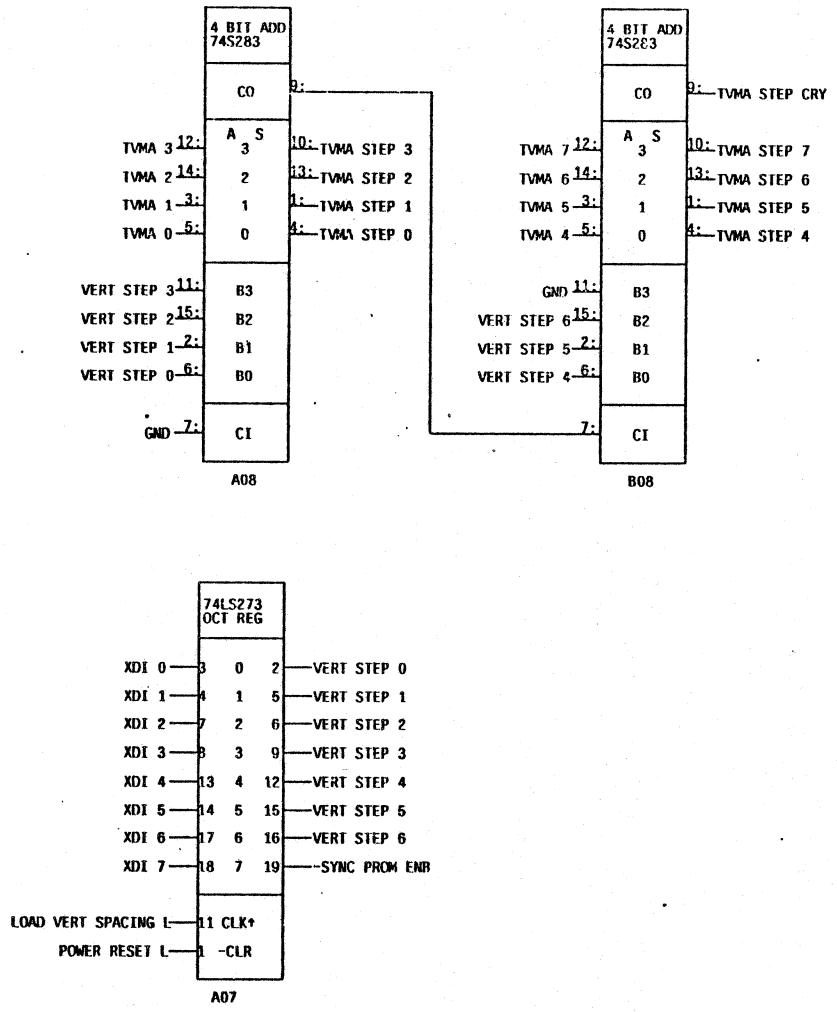
LISPM TV	Sync Clocking	7-DEC-1980 23:27	AI: CADRTV; SYNCLK



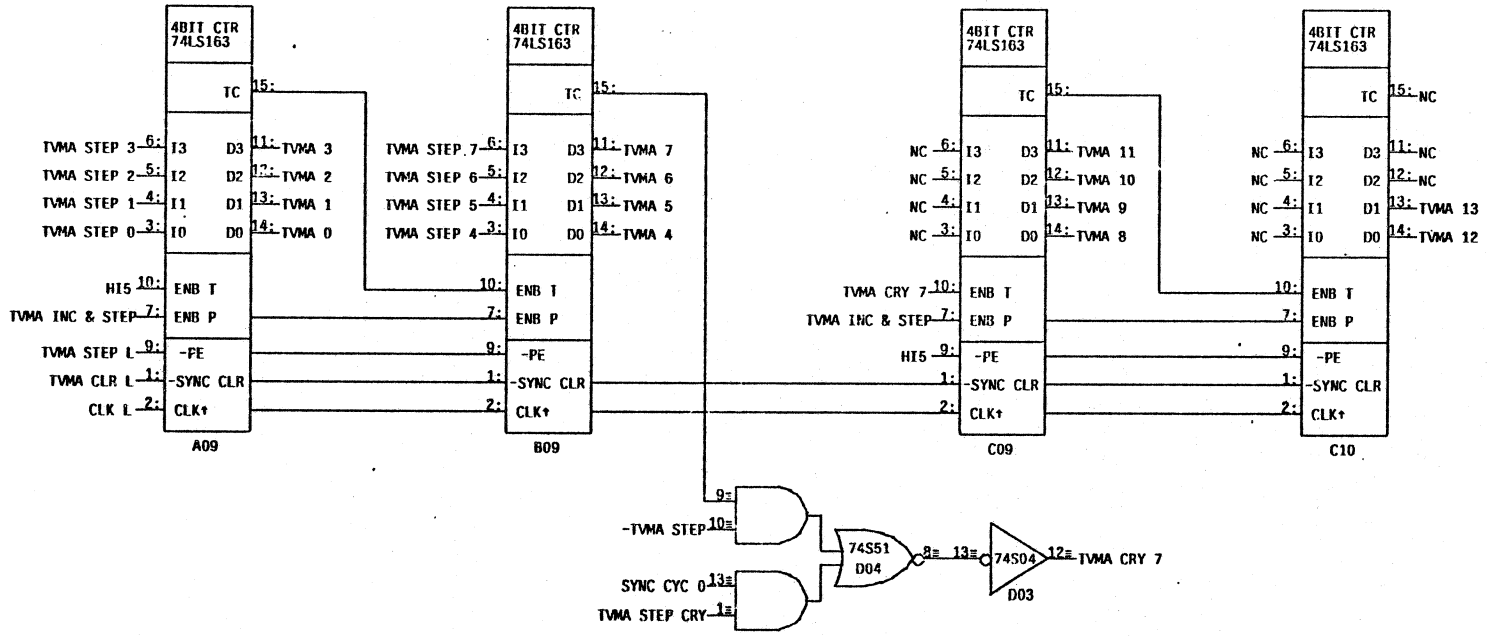
LISPM TV	Sync Program Memory	7-DEC-1980 23:31	AI: CADRTV; SYNRAM



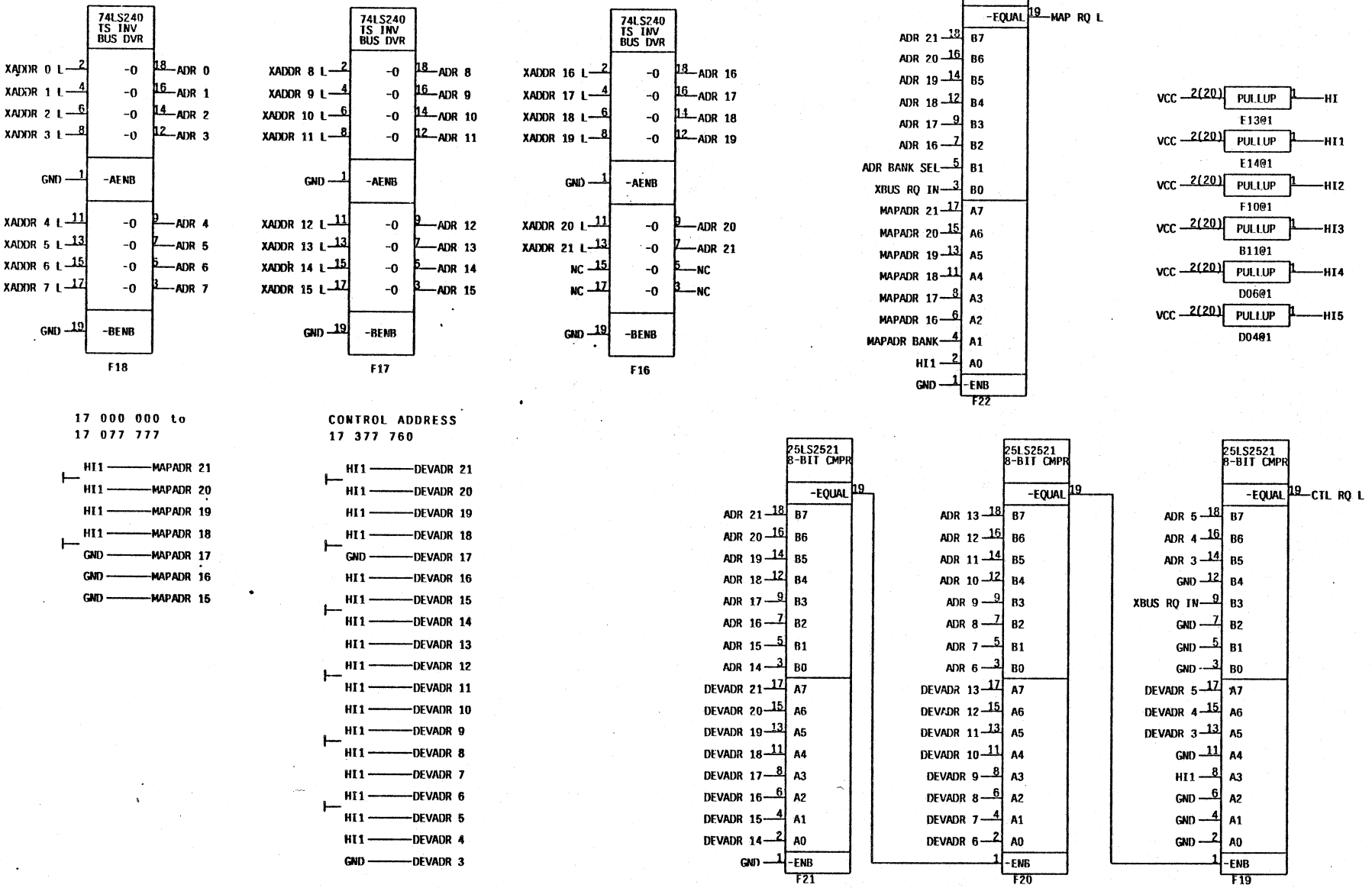
LISPM TV	Sync Program Reg & Repeat	7-DEC-1980 22:57	AI: CADRTV; SYNREG



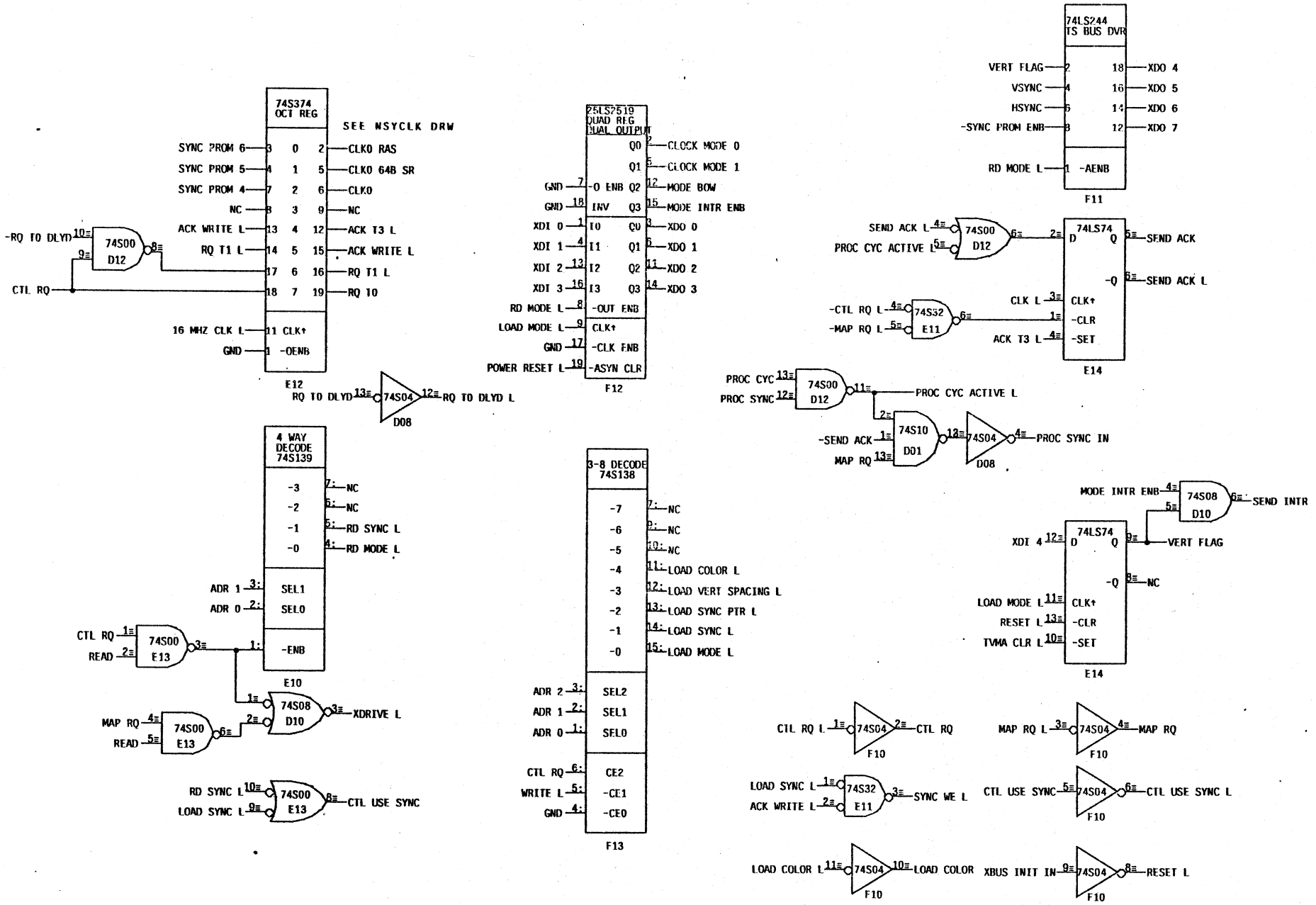
LISPM TV	TV Mem Address Adder	7-DEC-1980 22:58	AI: CADRTV; TVINC



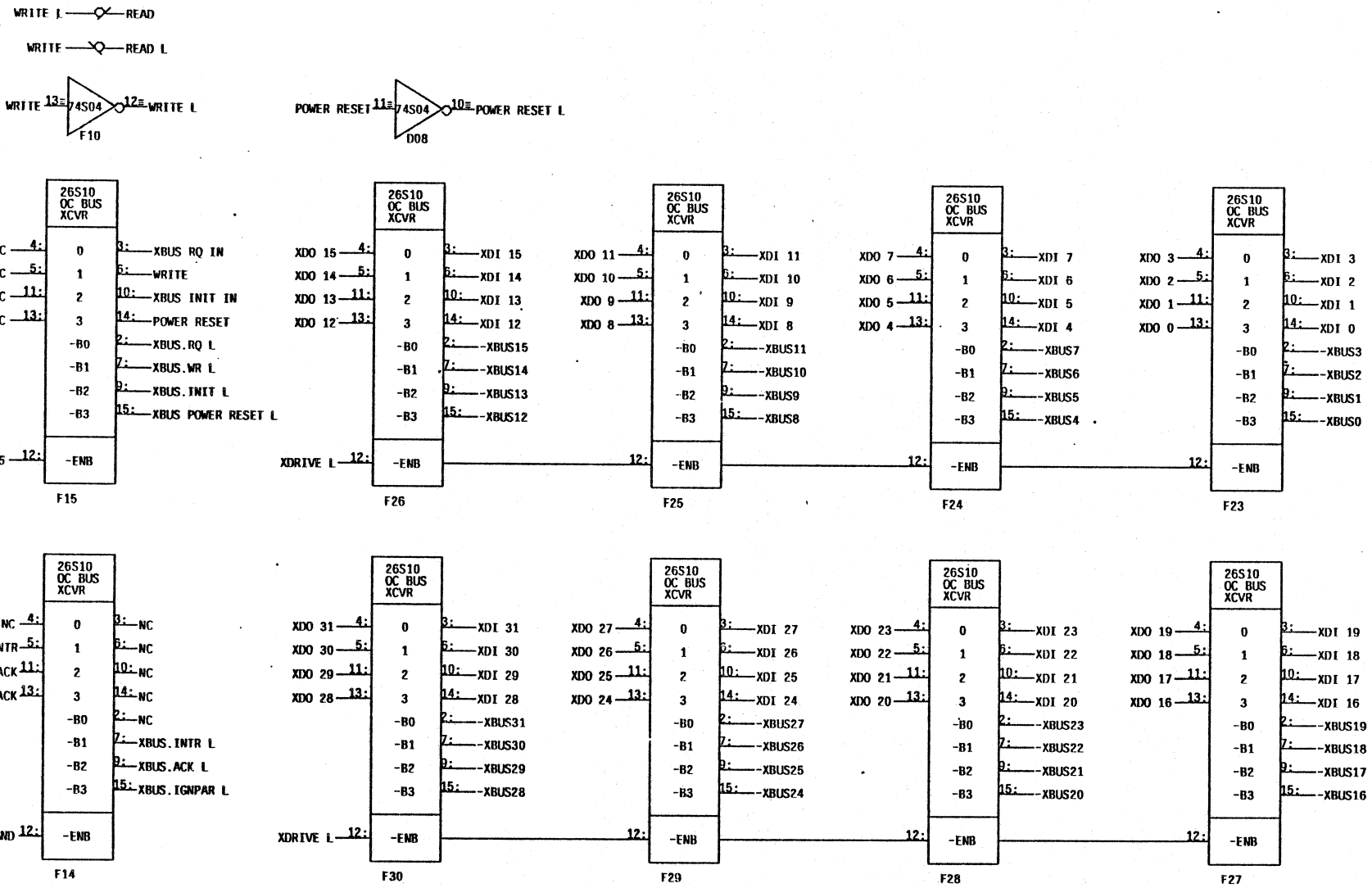
LISPM TV	TV Mem Address Counter	7-DEC-1980 23:02	AI: CADRTV; TVMA



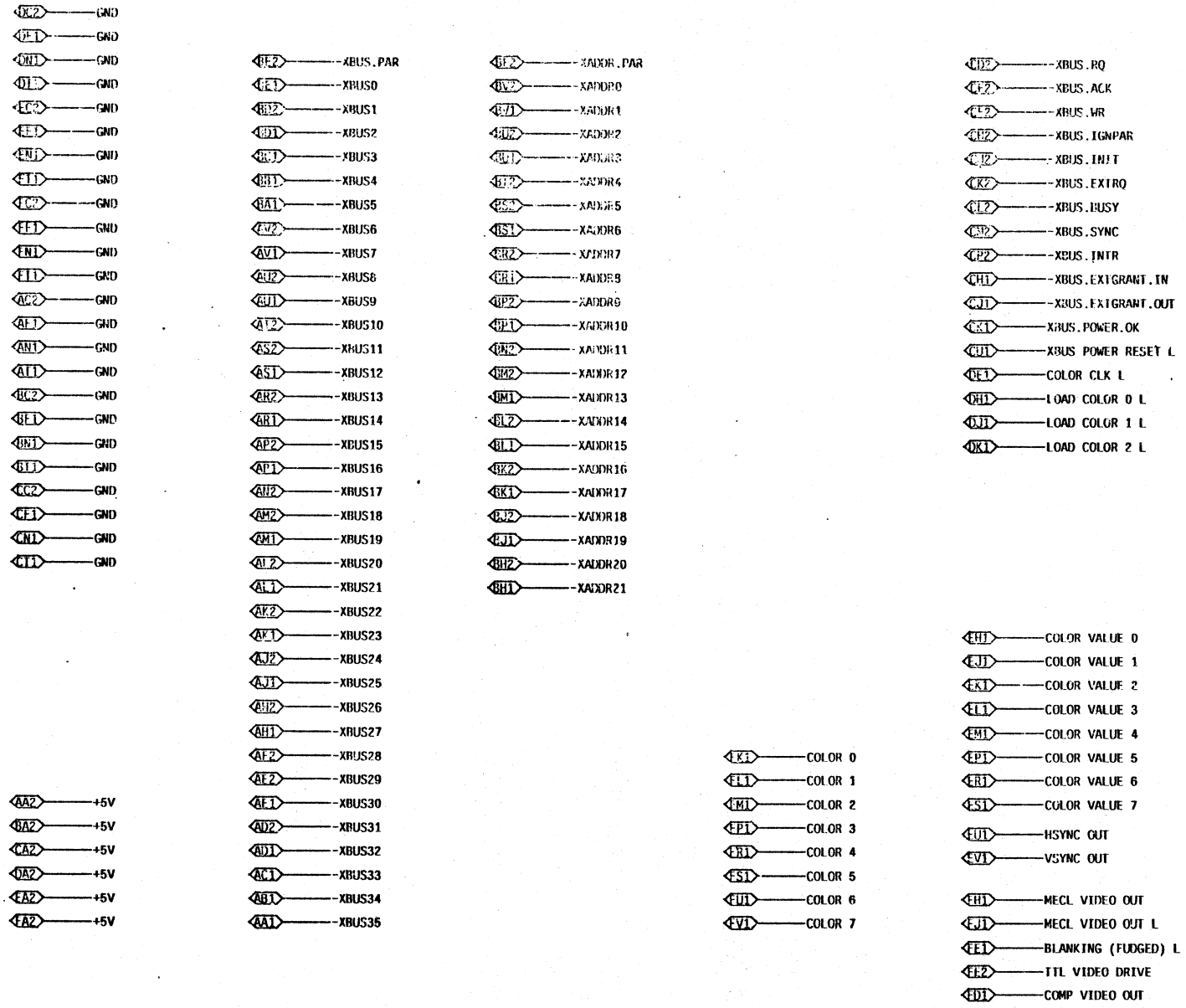
LISPM TV	XBus Address Selector	10-DEC-1980 10:51	AI: CADRTV; XBADR



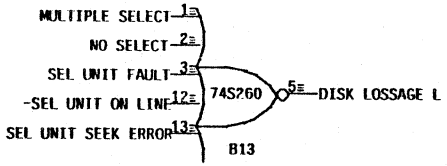
LISPM TV	Mode Reg and Xbus Control	7-DEC-1980 23:05	AI: CADRTV; XBCTL

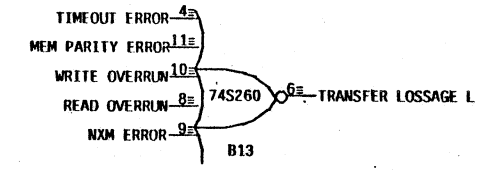


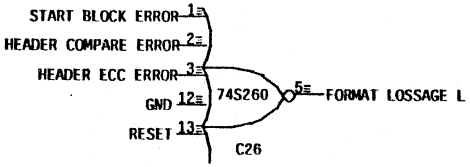
LISPM TV	Xbus Data Transceiver	7-DEC-1980 23:04	AI: CADRTV; XBDATA

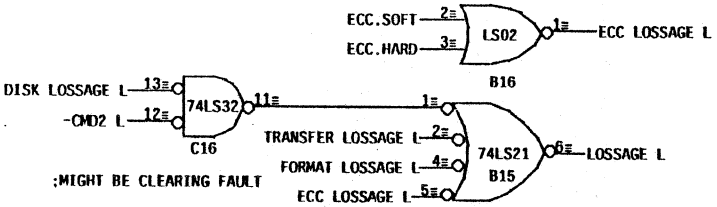


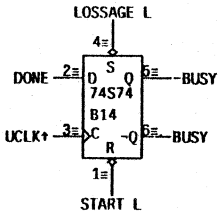
LISPM TV	Backplane Connections	7-DEC-1980 23:01	AI: CADRTV; XBUS

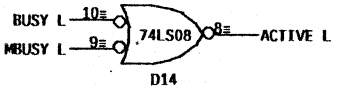


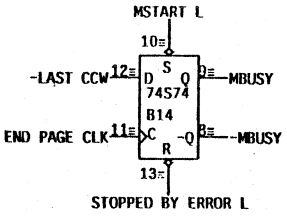


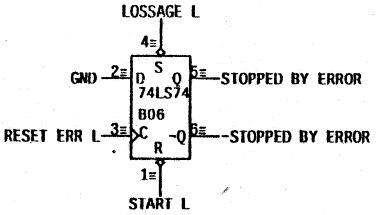




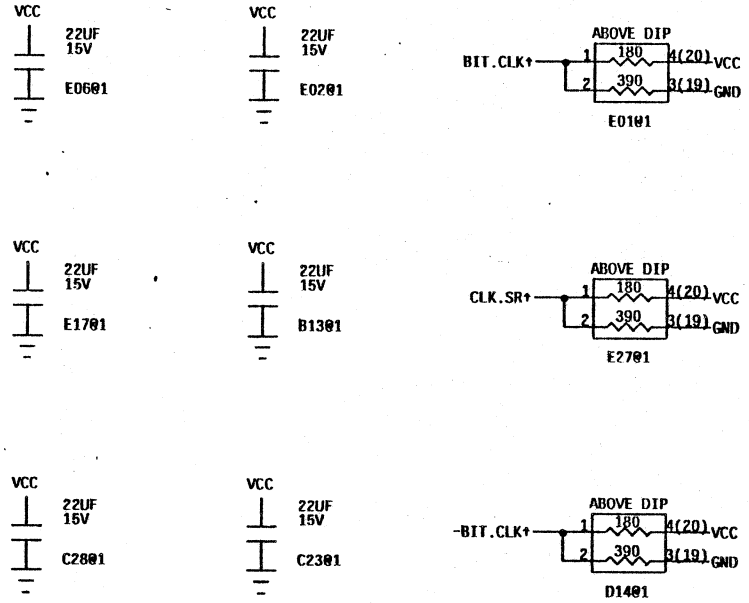


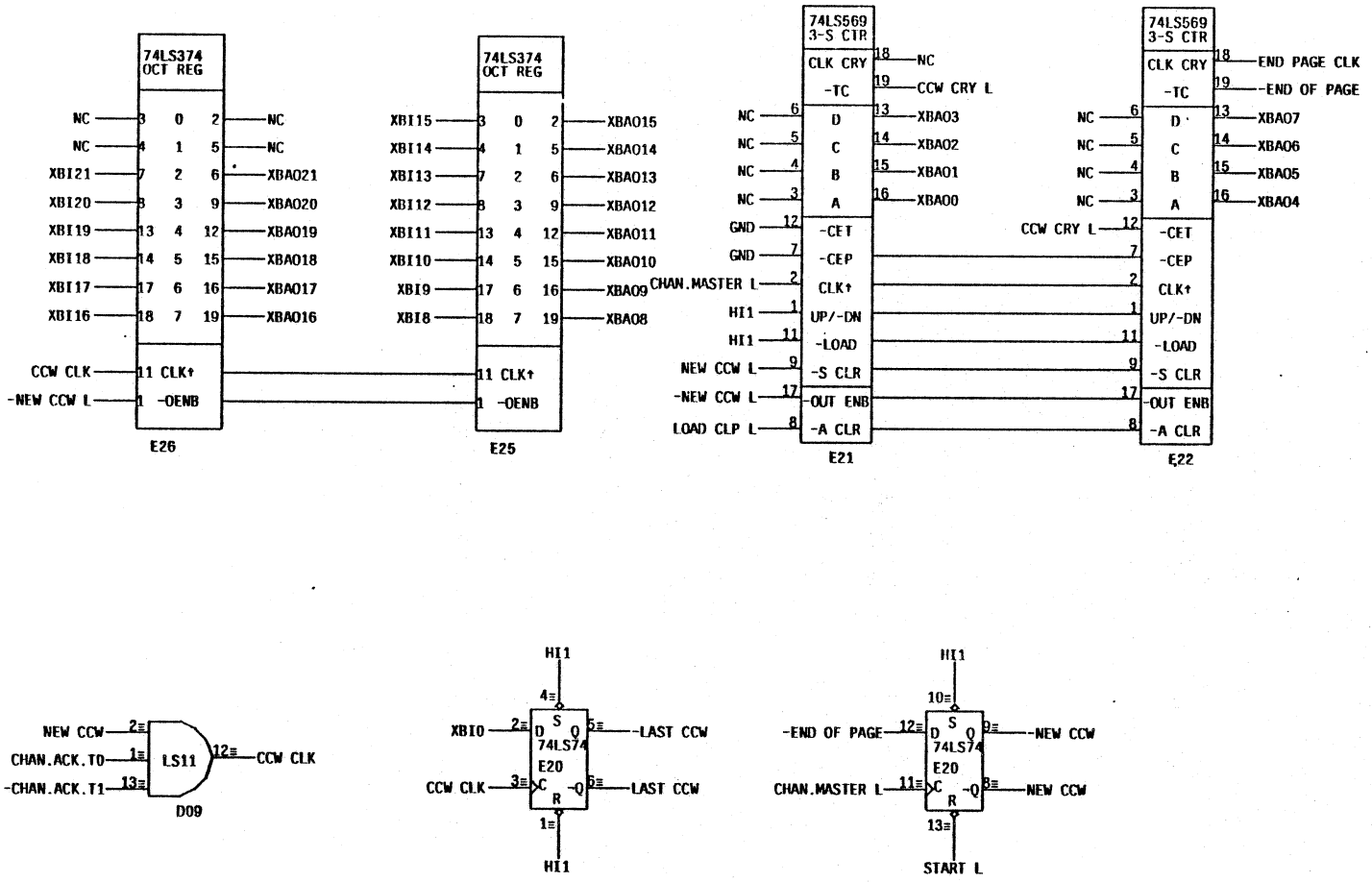




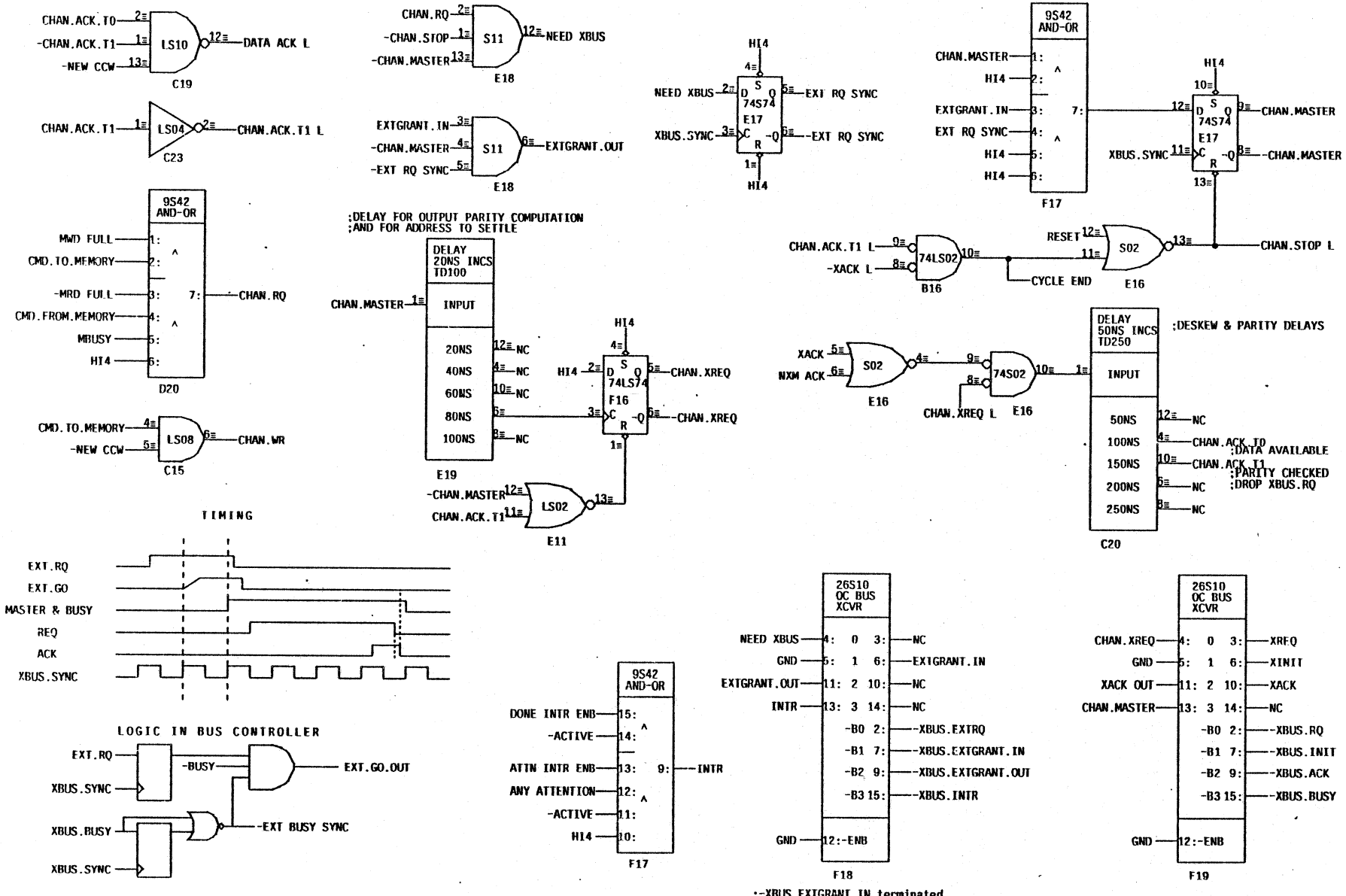


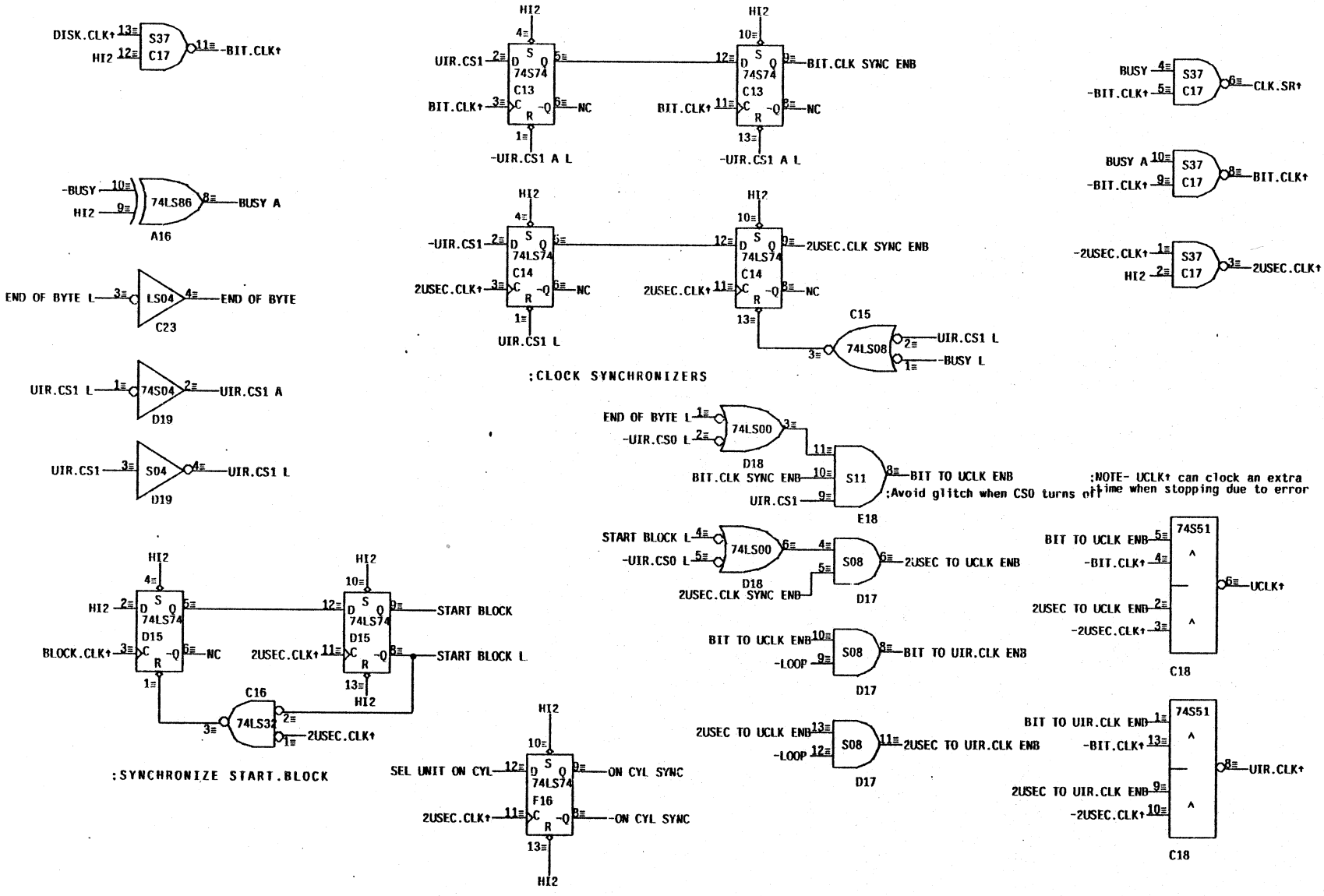
DISK CONTROL	BUSY, LOSSAGE	7-OCT-1980 21:32	AI: CADRDC; DCBUSY


VCC
22UF
15V
E0601
VCC
22UF
15V
E0201
ABOVE DIP
BIT.CLK↑
180
390
4(20)
3(19)
VCC
GND
E0101
VCC
22UF
15V
E1701
VCC
22UF
15V
B1301
ABOVE DIP
CLK.SR↑
180
390
4(20)
3(19)
VCC
GND
E2701
VCC
22UF
15V
C2801
VCC
22UF
15V
C2301
ABOVE DIP
-BIT.CLK↑
180
390
4(20)
3(19)
VCC
GND
D1401

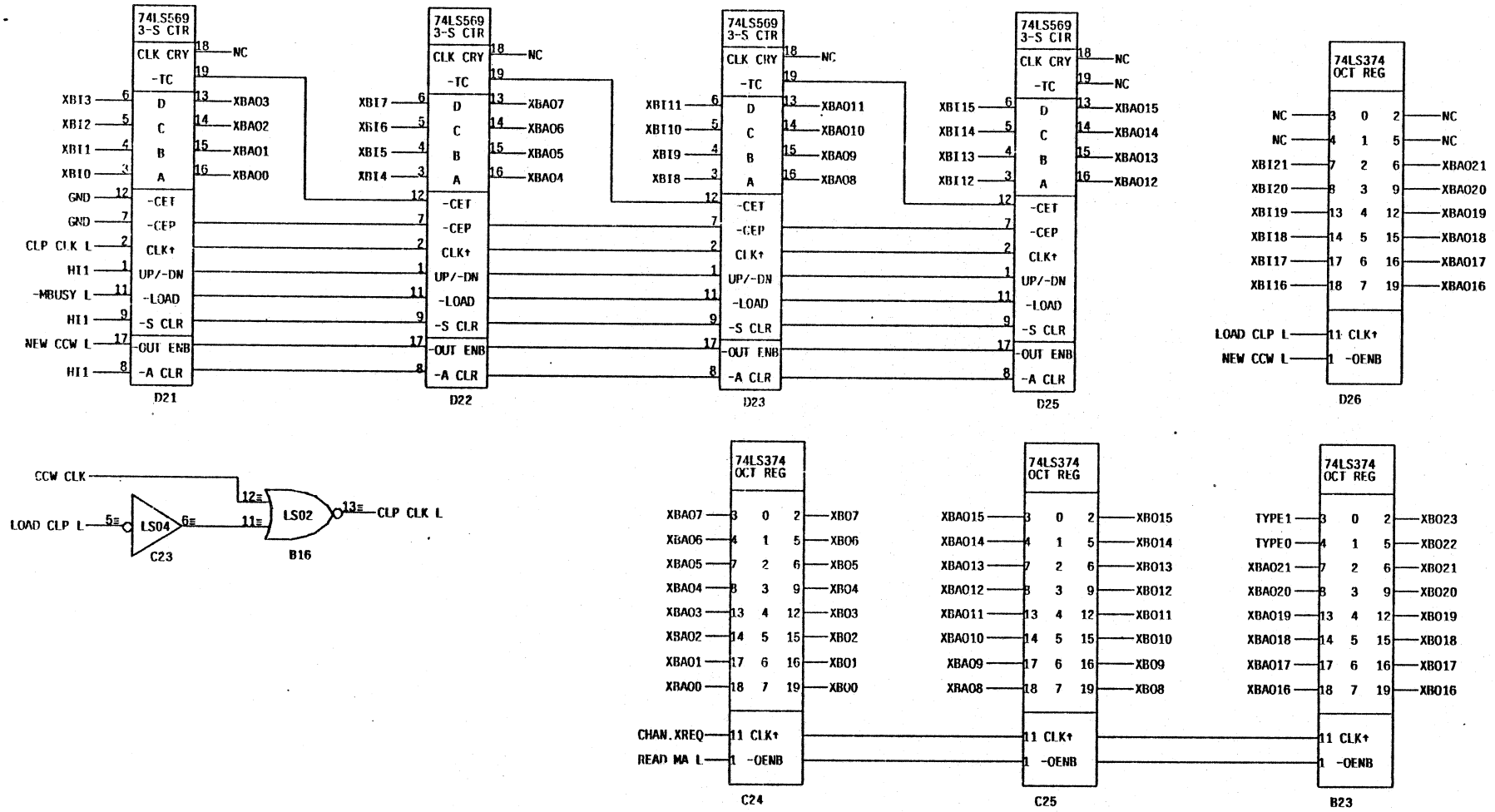


DISK CONTROL	CHANNEL CONTROL WORD	7-OCT-1980 21:32	AI: CADRDC; DCCCW


CHAN.ACK.T0
-CHAN.ACK.T1
-NEW CCW
LS10
DATA ACK L
C19
CHAN.ACK.T1
LS04
CHAN.ACK.T1 L
C23
9S42 AND-OR
MWD FULL
CMD.TO.MEMORY
-MRD FULL
CMD.FROM.MEMORY
MBUSY
HI4
CHAN.RQ
D20
CMD.TO.MEMORY
-NEW CCW
LS08
CHAN.WR
C15
CHAN.RQ
-CHAN.STOP
-CHAN.MASTER
S11
NEED XBUS
E18
EXTGRANT.IN
-CHAN.MASTER
-EXT RQ SYNC
S11
EXTGRANT.OUT
E18
;DELAY FOR OUTPUT PARITY COMPUTATION
;AND FOR ADDRESS TO SETTLE
DELAY 20NS INCS TD100
CHAN.MASTER
INPUT
20NS
40NS
60NS
80NS
100NS
NC
E19
HI4
NEED XBUS
XBUS.SYNC
74S74
E17
EXT RQ SYNC
-EXT RQ SYNC
HI4
HI4
74S74
F16
CHAN.XREQ
-CHAN.XREQ
-CHAN.MASTER
CHAN.ACK.T1
LS02
E11
9S42 AND-OR
CHAN.MASTER
HI4
EXTGRANT.IN
EXT RQ SYNC
HI4
HI4
F17
CHAN.ACK.T1 L
-XACK L
74LS02
B16
CYCLE END
RESET
S02
E16
HI4
74S74
E17
XBUS.SYNC
CHAN.MASTER
-CHAN.MASTER
CHAN.STOP L
XACK
NXM ACK
S02
E16
74S02
E16
CHAN.XREQ L
DELAY 50NS INCS TD250
;DESKEW & PARITY DELAYS
INPUT
50NS
100NS
150NS
200NS
250NS
NC
CHAN.ACK.T0
;DATA AVAILABLE
CHAN.ACK.T1
;PARITY CHECKED
NC
;DROP XBUS.RQ
NC
C20
TIMING
EXT.RQ
EXT.GO
MASTER & BUSY
REQ
ACK
XBUS.SYNC
LOGIC IN BUS CONTROLLER
EXT.RQ
XBUS.SYNC
-BUSY
EXT.GO.OUT
XBUS.BUSY
XBUS.SYNC
-EXT BUSY SYNC
9S42 AND-OR
DONE INTR ENB
-ACTIVE
ATTN INTR ENB
ANY ATTENTION
-ACTIVE
HI4
INTR
F17
26S10 OC BUS XCVR
NEED XBUS
GND
EXTGRANT.OUT
INTR
-B0
-B1
-B2
-B3
GND
NC
EXTGRANT.IN
NC
NC
-XBUS.EXTRQ
-XBUS.EXTGRANT.IN
-XBUS.EXTGRANT.OUT
-XBUS.INTR
-ENB
F18
;-XBUS.EXTGRANT.IN terminated
; on DCTMOT or MKPANL
26S10 OC BUS XCVR
CHAN.XREQ
GND
XACK OUT
CHAN.MASTER
-B0
-B1
-B2
-B3
GND
XREQ
XINIT
XACK
NC
-XBUS.RQ
-XBUS.INIT
-XBUS.ACK
-XBUS.BUSY
-ENB
F19

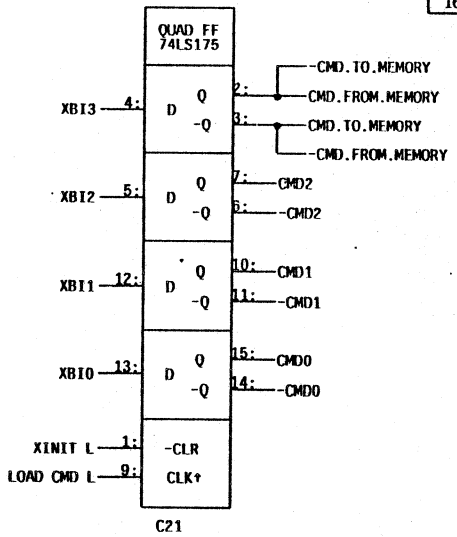


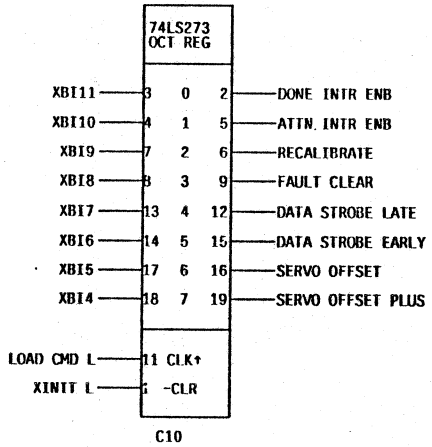
DISK CONTROL	CLOCKS	8-OCT-1978 16:48	AI: CADRDC; DCCLK

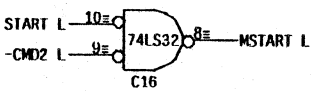


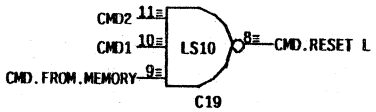
DISK CONTROL	COMMAND LIST POINTER	10-JUN-1979 21:52	AI: CADRDC; DCCLP

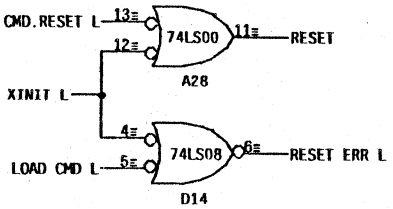
CMD	OPERATION
00	READ
10	READ COMPARE
11	WRITE
02	READ ALL
13	WRITE ALL
04	SEEK
05	AT EASE
1005	RECALIBRATE
405	FAULT CLEAR
06	OFFSET CLEAR
16	STOP, RESET

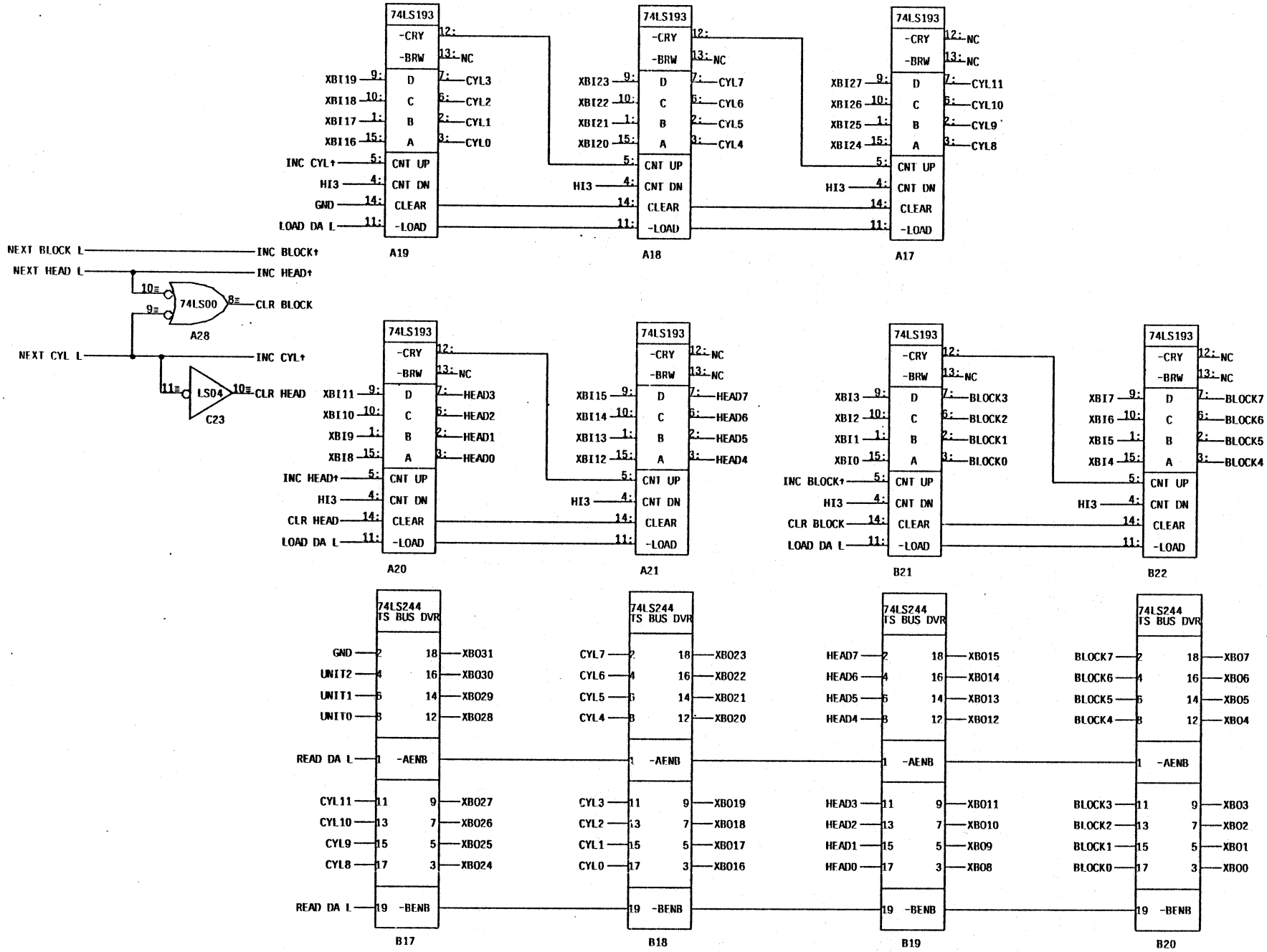




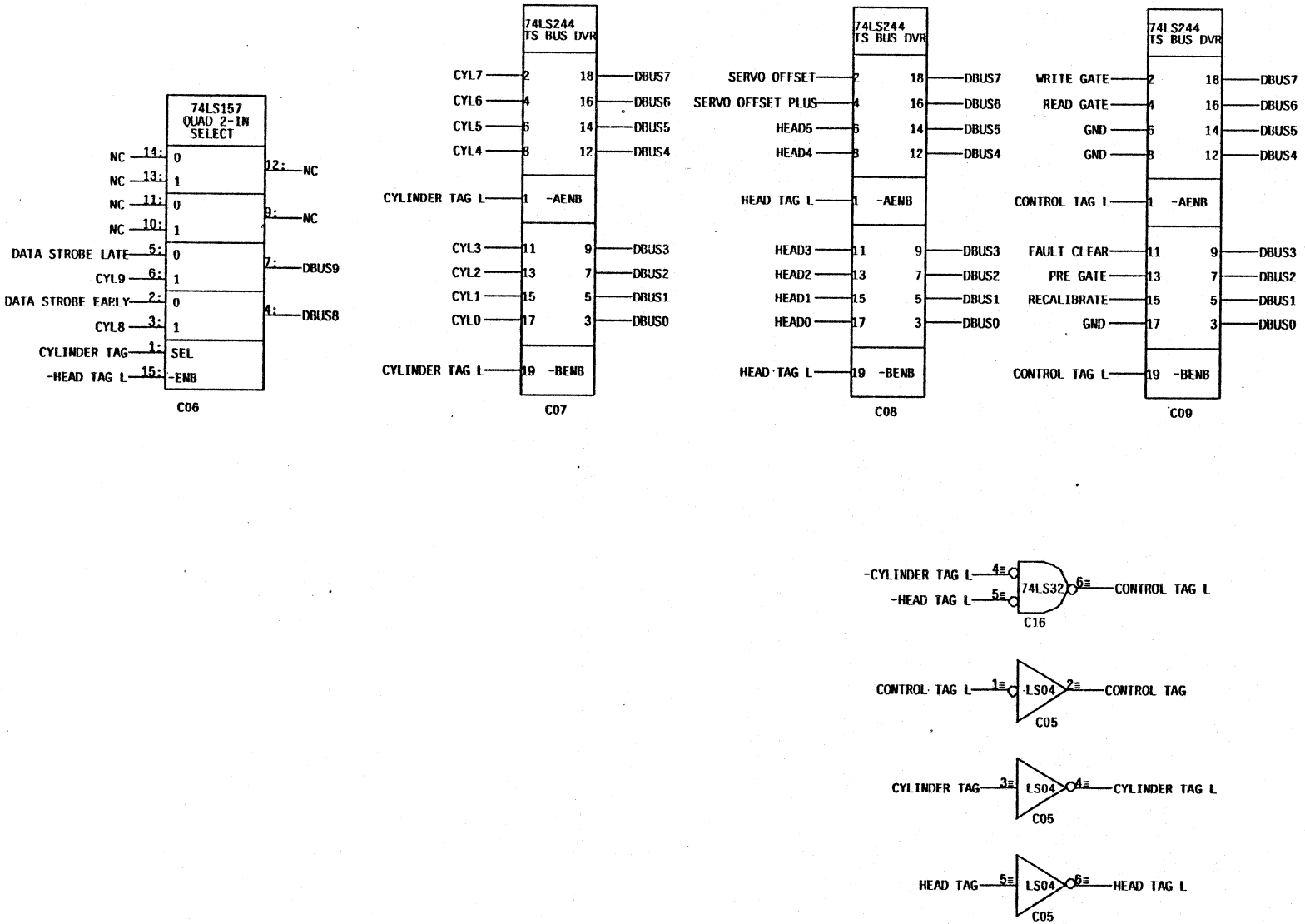


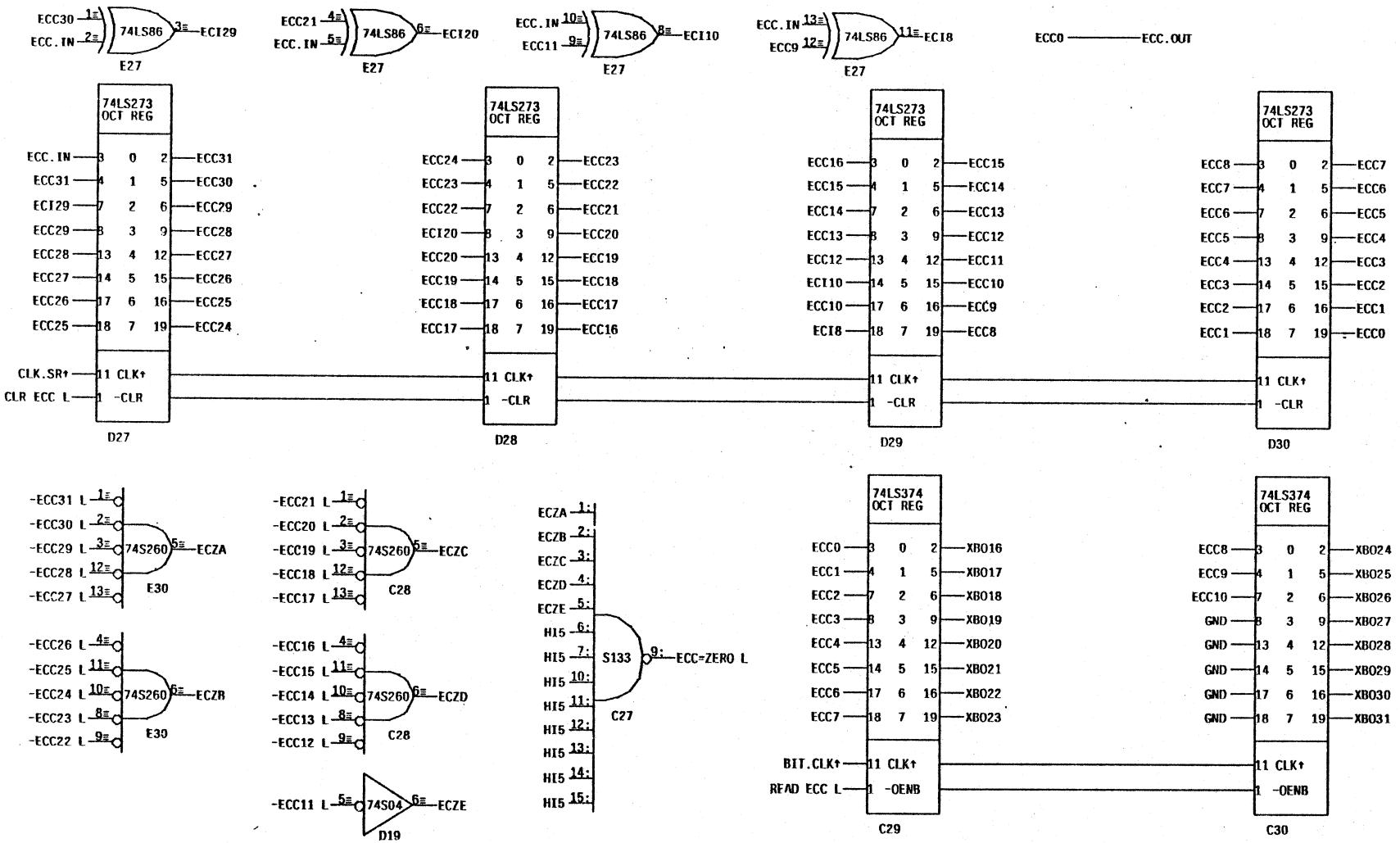




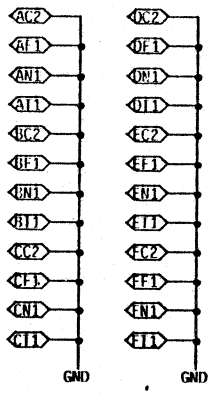


DISK CONTROL	DISK ADDRESS	30-SEP-1978 01:36	AI: CADRDC; DCDA


74LS157
QUAD 2-IN
SELECT
NC 14: 0
NC 13: 1
NC 11: 0
NC 10: 1
DATA STROBE LATE 5: 0
CYL9 6: 1
DATA STROBE EARLY 2: 0
CYL8 3: 1
CYLINDER TAG 1: SEL
-HEAD TAG L 15: -ENB
12: NC
9: NC
7: DBUS9
4: DBUS8
C06
74LS244
TS BUS DVR
CYL7 2 18 DBUS7
CYL6 4 16 DBUS6
CYL5 6 14 DBUS5
CYL4 8 12 DBUS4
CYLINDER TAG L 1 -AENB
CYL3 11 9 DBUS3
CYL2 13 7 DBUS2
CYL1 15 5 DBUS1
CYL0 17 3 DBUS0
CYLINDER TAG L 19 -BENB
C07
74LS244
TS BUS DVR
SERVO OFFSET 2 18 DBUS7
SERVO OFFSET PLUS 4 16 DBUS6
HEAD5 6 14 DBUS5
HEAD4 8 12 DBUS4
HEAD TAG L 1 -AENB
HEAD3 11 9 DBUS3
HEAD2 13 7 DBUS2
HEAD1 15 5 DBUS1
HEAD0 17 3 DBUS0
HEAD TAG L 19 -BENB
C08
74LS244
TS BUS DVR
WRITE GATE 2 18 DBUS7
READ GATE 4 16 DBUS6
GND 6 14 DBUS5
GND 8 12 DBUS4
CONTROL TAG L 1 -AENB
FAULT CLEAR 11 9 DBUS3
PRE GATE 13 7 DBUS2
RECALIBRATE 15 5 DBUS1
GND 17 3 DBUS0
CONTROL TAG L 19 -BENB
C09
-CYLINDER TAG L 4
-HEAD TAG L 5
74LS32
6 CONTROL TAG L
C16
CONTROL TAG L 1
LS04
2 CONTROL TAG
C05
CYLINDER TAG 3
LS04
4 CYLINDER TAG L
C05
HEAD TAG 5
LS04
6 HEAD TAG L
C05


ECC30
ECC.IN
74LS86
ECI29
E27
ECC21
ECC.IN
74LS86
ECI20
E27
ECC.IN
ECC11
74LS86
ECI10
E27
ECC.IN
ECC9
74LS86
ECI8
E27
ECC0
ECC.OUT
74LS273
OCT REG
ECC.IN
ECC31
ECI29
ECC29
ECC28
ECC27
ECC26
ECC25
ECC31
ECC30
ECC29
ECC28
ECC27
ECC26
ECC25
ECC24
CLK.SR↑
CLR ECC L
CLK↑
-CLR
D27
ECC24
ECC23
ECC22
ECI20
ECC20
ECC19
ECC18
ECC17
ECC23
ECC22
ECC21
ECC20
ECC19
ECC18
ECC17
ECC16
D28
ECC16
ECC15
ECC14
ECC13
ECC12
ECI10
ECC10
ECI8
ECC15
ECC14
ECC13
ECC12
ECC11
ECC10
ECC9
ECC8
D29
ECC8
ECC7
ECC6
ECC5
ECC4
ECC3
ECC2
ECC1
ECC7
ECC6
ECC5
ECC4
ECC3
ECC2
ECC1
ECC0
D30
-ECC31 L
-ECC30 L
-ECC29 L
-ECC28 L
-ECC27 L
74S260
ECZA
E30
-ECC26 L
-ECC25 L
-ECC24 L
-ECC23 L
-ECC22 L
74S260
ECZB
E30
-ECC21 L
-ECC20 L
-ECC19 L
-ECC18 L
-ECC17 L
74S260
ECZC
C28
-ECC16 L
-ECC15 L
-ECC14 L
-ECC13 L
-ECC12 L
74S260
ECZD
C28
-ECC11 L
74S04
ECZE
D19
ECZA
ECZB
ECZC
ECZD
ECZE
HI5
HI5
HI5
HI5
HI5
HI5
HI5
S133
ECC=ZERO L
C27
74LS374
OCT REG
ECC0
ECC1
ECC2
ECC3
ECC4
ECC5
ECC6
ECC7
XBO16
XBO17
XBO18
XBO19
XBO20
XBO21
XBO22
XBO23
BIT.CLK↑
READ ECC L
CLK↑
-OENB
C29
74LS374
OCT REG
ECC8
ECC9
ECC10
GND
GND
GND
GND
GND
XBO24
XBO25
XBO26
XBO27
XBO28
XBO29
XBO30
XBO31
CLK↑
-OENB
C30

GROUNDS



JUMPERS FOR 1-BOARD VERSION:

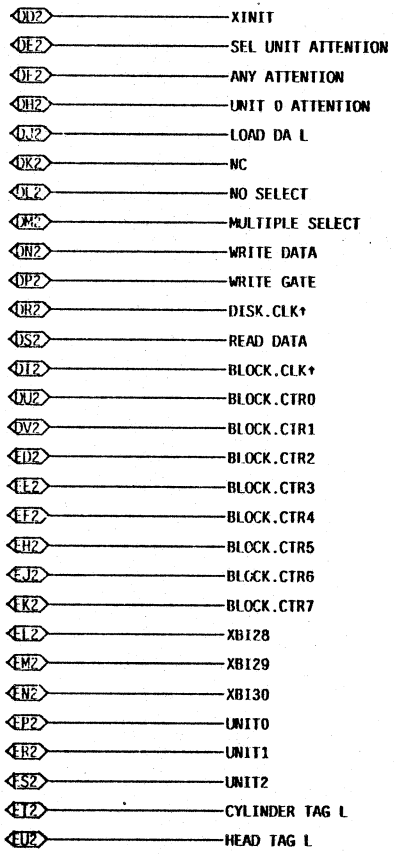
DE2 - DF2 - DH2
DN1 - DM2
ET1 - EP2 - ER2 - ES2

OMITTED DIPS IN 2-BOARD VERSION:

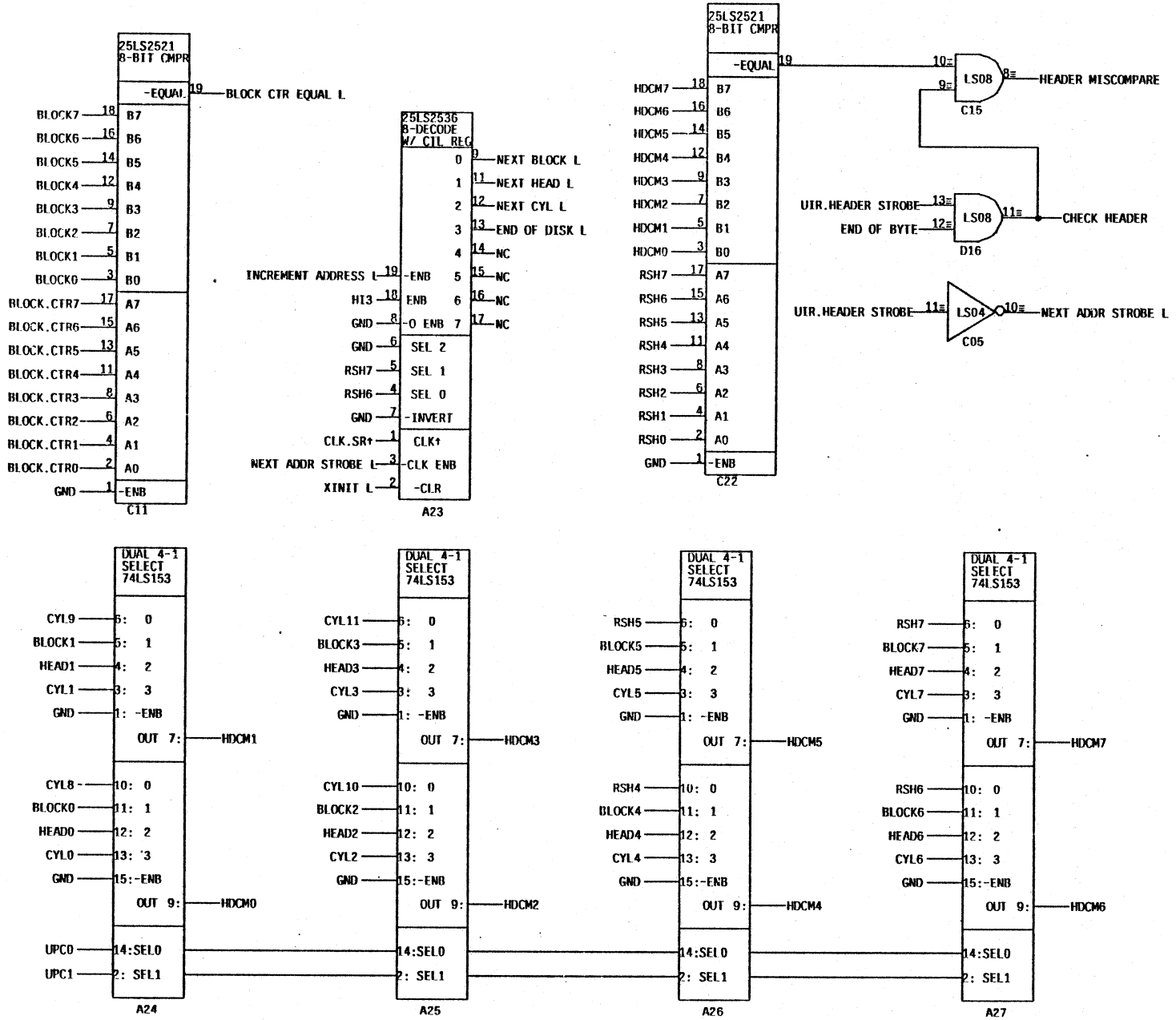
A7 A8 A9
A10 B7 B8

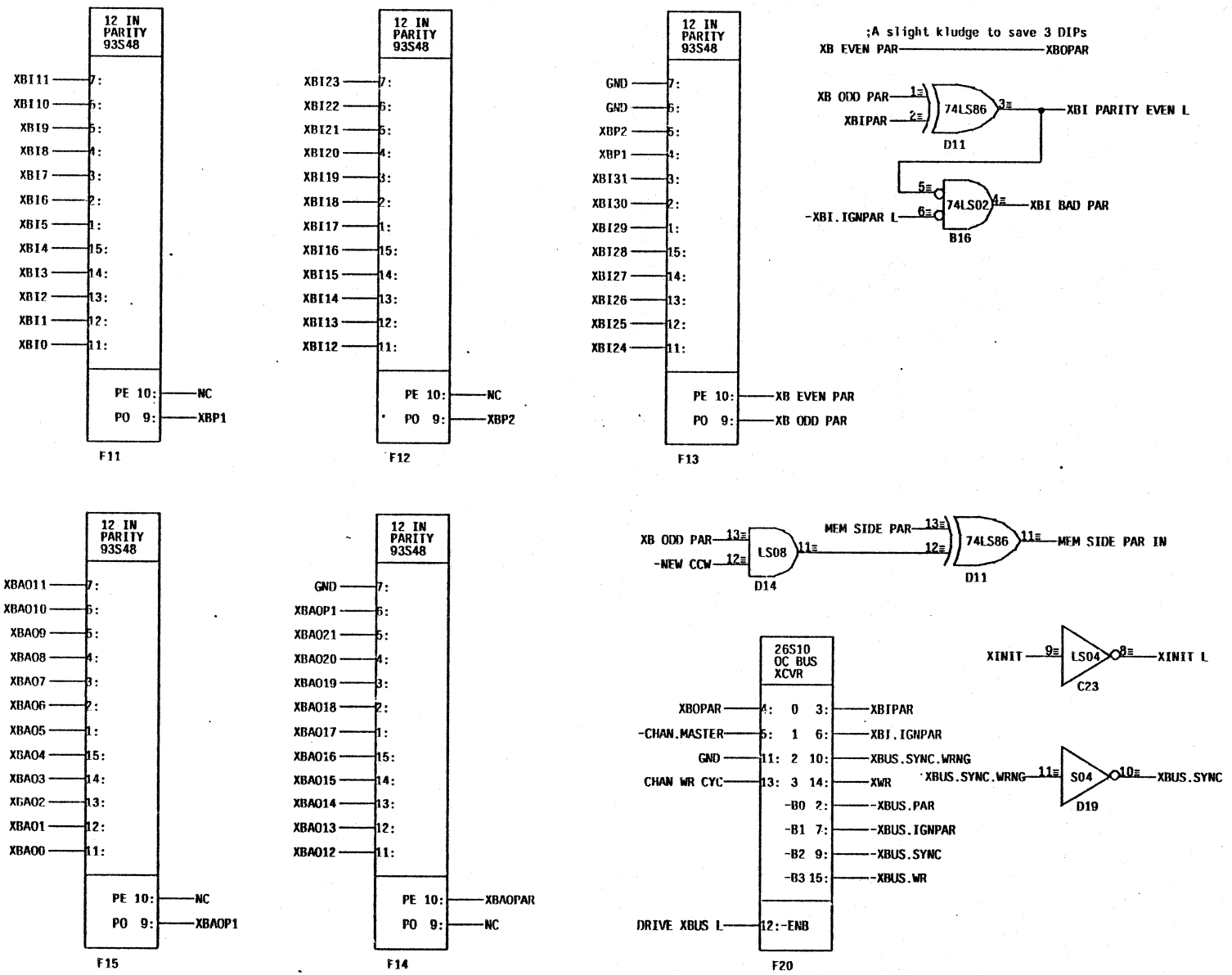
BE CERTAIN TO EDIT THE
RAY FILE FOR UNDEDICATED
GROUND PINS ON J1, J3

TO DISK MUX BOARD

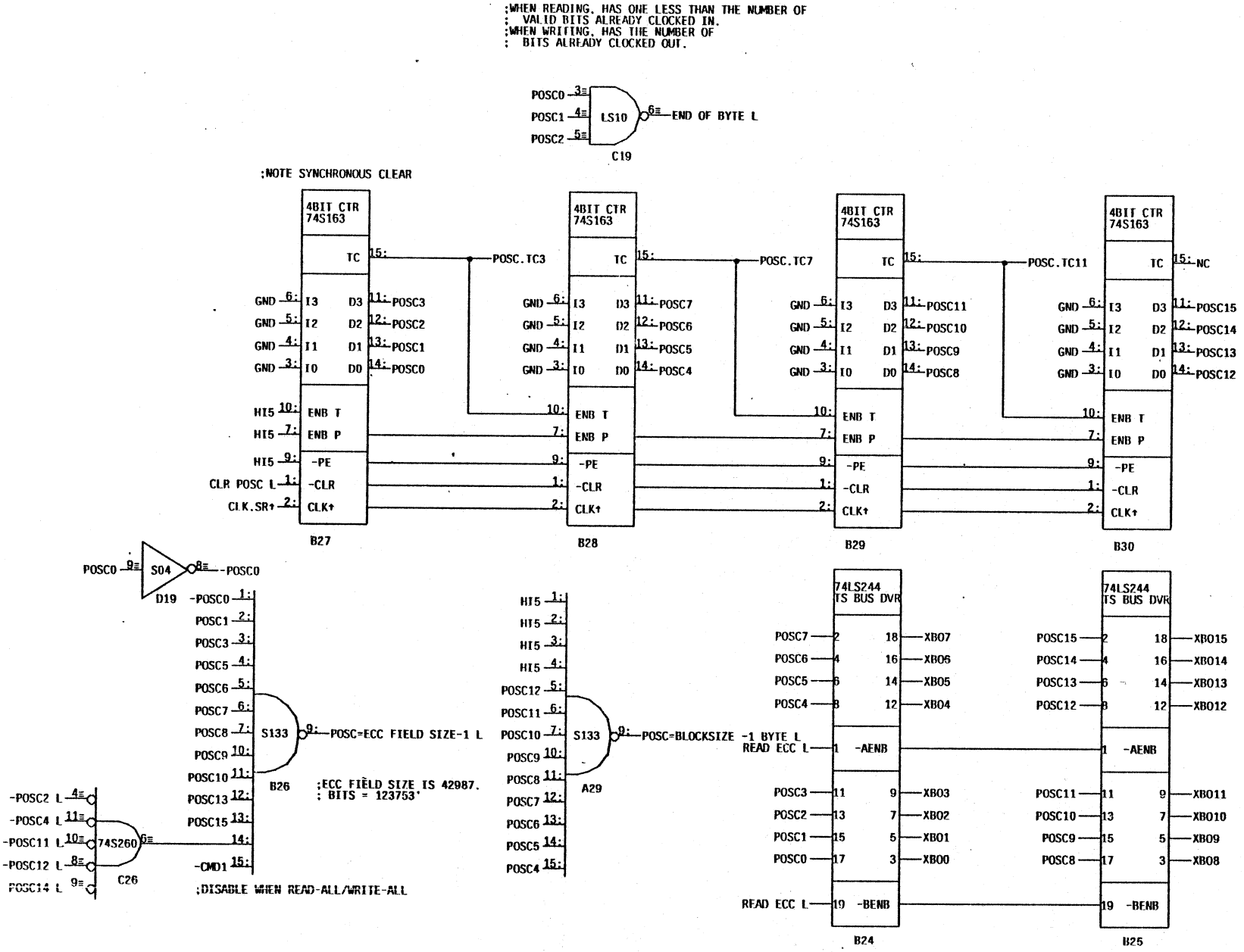


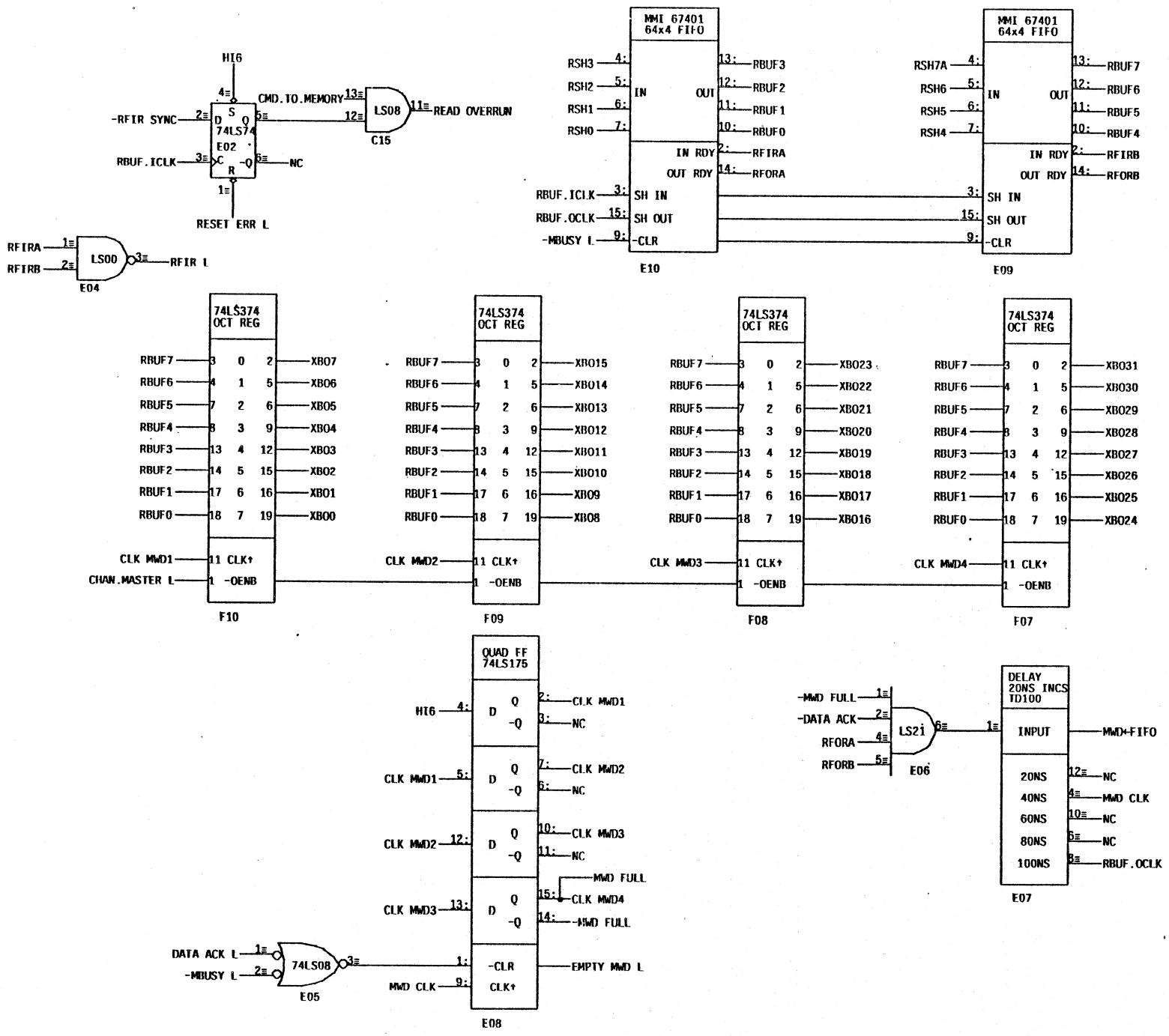
DISK CONTROL	EDGE CONNECTIONS	10-JUN-1979 20:33	AI: CADRDC; DCEDGE


25LS2521
8-BIT CMPR
-EQUAL
BLOCK CTR EQUAL L
BLOCK7
BLOCK6
BLOCK5
BLOCK4
BLOCK3
BLOCK2
BLOCK1
BLOCK0
BLOCK.CTR7
BLOCK.CTR6
BLOCK.CTR5
BLOCK.CTR4
BLOCK.CTR3
BLOCK.CTR2
BLOCK.CTR1
BLOCK.CTR0
GND
C11
25LS2536
8-DECODE
W/ CTL REG
NEXT BLOCK L
NEXT HEAD L
NEXT CYL L
END OF DISK L
NC
INCREMENT ADDRESS L
HI3
GND
GND
RSH7
RSH6
GND
CLK.SR↑
NEXT ADDR STROBE L
XINIT L
A23
HDCM7
HDCM6
HDCM5
HDCM4
HDCM3
HDCM2
HDCM1
HDCM0
RSH7
RSH6
RSH5
RSH4
RSH3
RSH2
RSH1
RSH0
GND
C22
LS08
HEADER MISCOMPARE
C15
UIR.HEADER STROBE
END OF BYTE
CHECK HEADER
D16
LS04
NEXT ADDR STROBE L
C05
DUAL 4-1
SELECT
74LS153
CYL9
BLOCK1
HEAD1
CYL1
GND
OUT 7:
HDCM1
CYL8
BLOCK0
HEAD0
CYL0
GND
OUT 9:
HDCM0
UPC0
UPC1
SEL0
SEL1
A24
CYL11
BLOCK3
HEAD3
CYL3
GND
HDCM3
CYL10
BLOCK2
HEAD2
CYL2
GND
HDCM2
A25
RSH5
BLOCK5
HEAD5
CYL5
GND
HDCM5
RSH4
BLOCK4
HEAD4
CYL4
GND
HDCM4
A26
RSH7
BLOCK7
HEAD7
CYL7
GND
HDCM7
RSH6
BLOCK6
HEAD6
CYL6
GND
HDCM6
A27

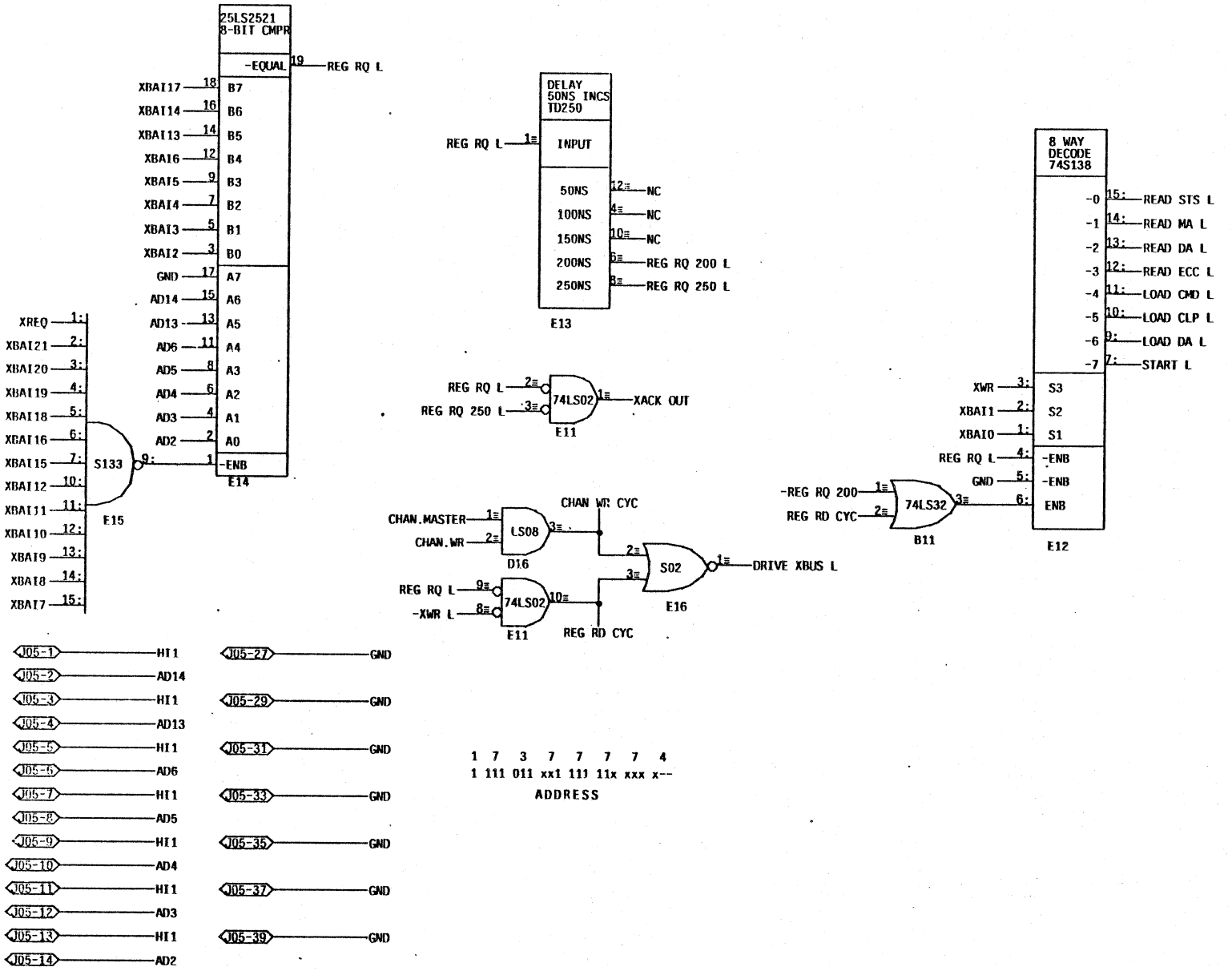


DISK CONTROL	PARITY	6-FEB-1980 00:54	AI: CADRDC; DCPAR

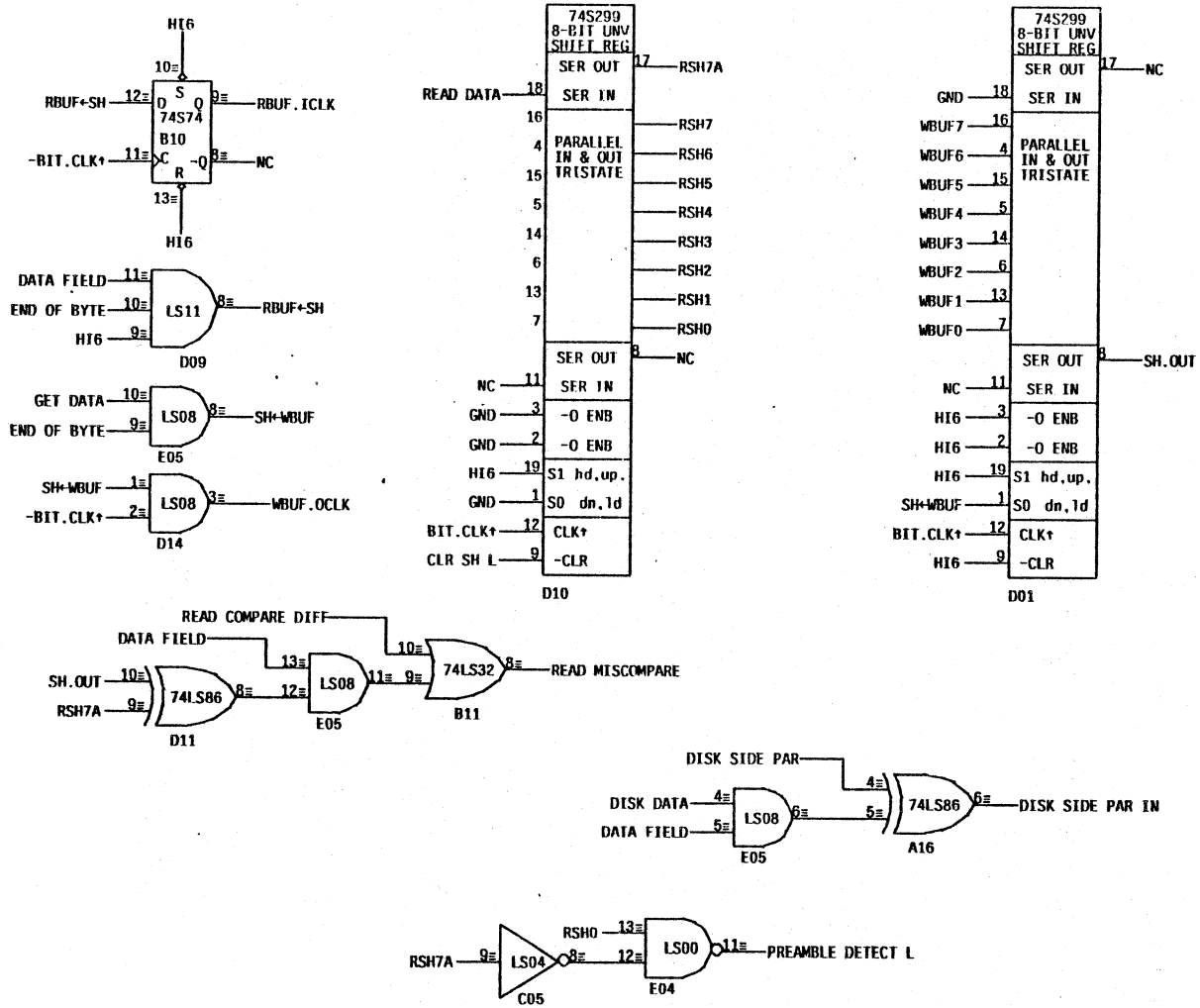

;WHEN READING, HAS ONE LESS THAN THE NUMBER OF
; VALID BITS ALREADY CLOCKED IN.
;WHEN WRITING, HAS THE NUMBER OF
; BITS ALREADY CLOCKED OUT.
POSC0
POSC1
POSC2
LS10
END OF BYTE L
C19
;NOTE SYNCHRONOUS CLEAR
4BIT CTR 74S163
TC
POSC.TC3
POSC.TC7
POSC.TC11
NC
GND
I3
I2
I1
I0
D3
D2
D1
D0
POSC3
POSC2
POSC1
POSC0
POSC7
POSC6
POSC5
POSC4
POSC11
POSC10
POSC9
POSC8
POSC15
POSC14
POSC13
POSC12
HI5
ENB T
ENB P
-PE
CLR POSC L
-CLR
CLK.SR↑
CLK↑
B27
B28
B29
B30
POSC0
S04
-POSC0
D19
-POSC0
POSC1
POSC3
POSC5
POSC6
POSC7
POSC8
POSC9
POSC10
POSC13
POSC15
-CMD1
S133
POSC=ECC FIELD SIZE-1 L
B26
;ECC FIELD SIZE IS 42987.
; BITS = 123753'
;DISABLE WHEN READ-ALL/WRITE-ALL
-POSC2 L
-POSC4 L
-POSC11 L
-POSC12 L
POSC14 L
74S260
C26
HI5
POSC12
POSC11
POSC10
POSC9
POSC8
POSC7
POSC6
POSC5
POSC4
S133
POSC=BLOCKSIZE -1 BYTE L
A29
74LS244 TS BUS DVR
POSC7
POSC6
POSC5
POSC4
POSC3
POSC2
POSC1
POSC0
XBO7
XBO6
XBO5
XBO4
XBO3
XBO2
XBO1
XBO0
READ ECC L
-AENB
-BENB
B24
POSC15
POSC14
POSC13
POSC12
POSC11
POSC10
POSC9
POSC8
XBO15
XBO14
XBO13
XBO12
XBO11
XBO10
XBO9
XBO8
B25



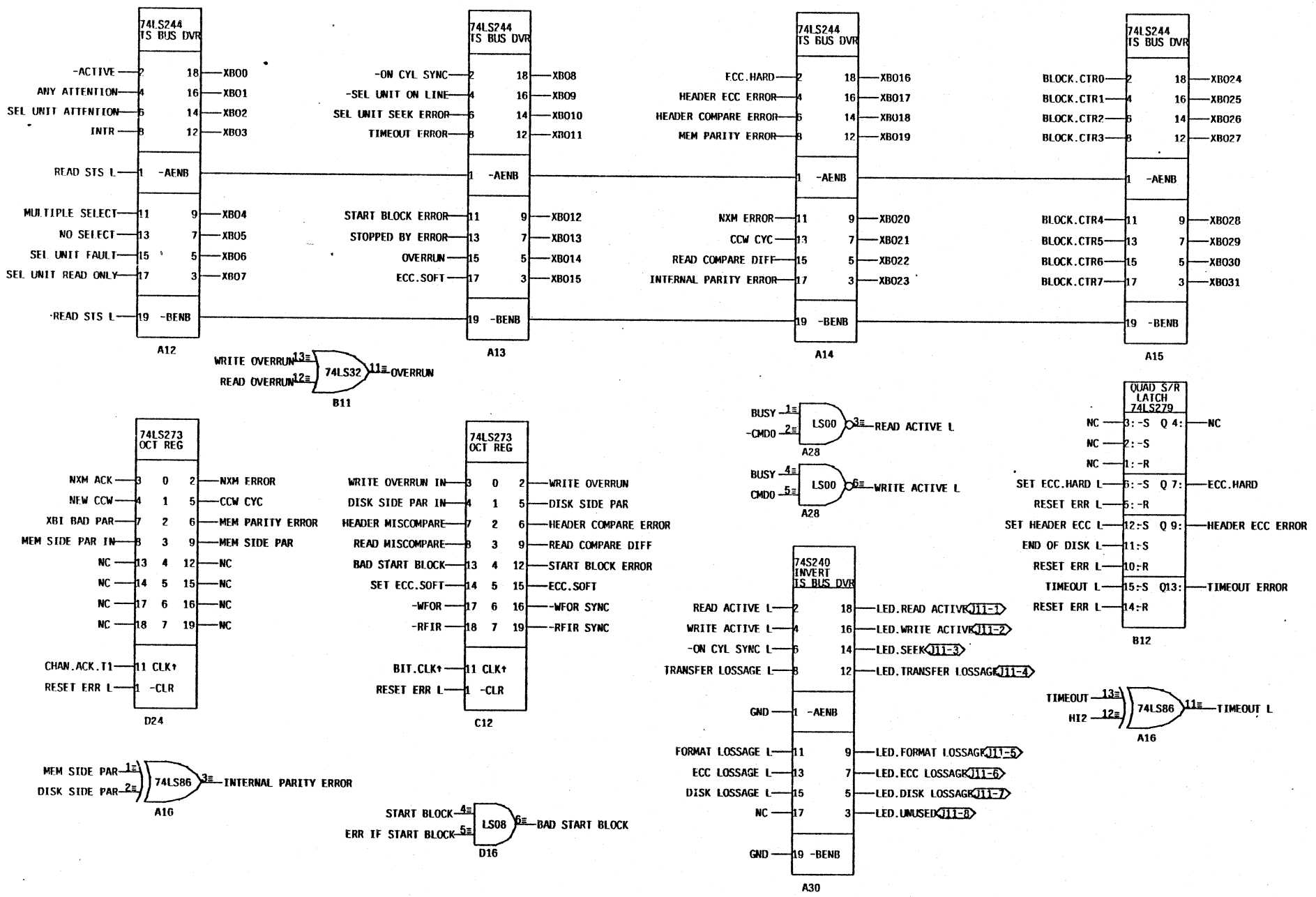
DISK CONTROL	READ BUFFER	30-SEP-1978 04:07	AI: CADRDC; DCRBUF



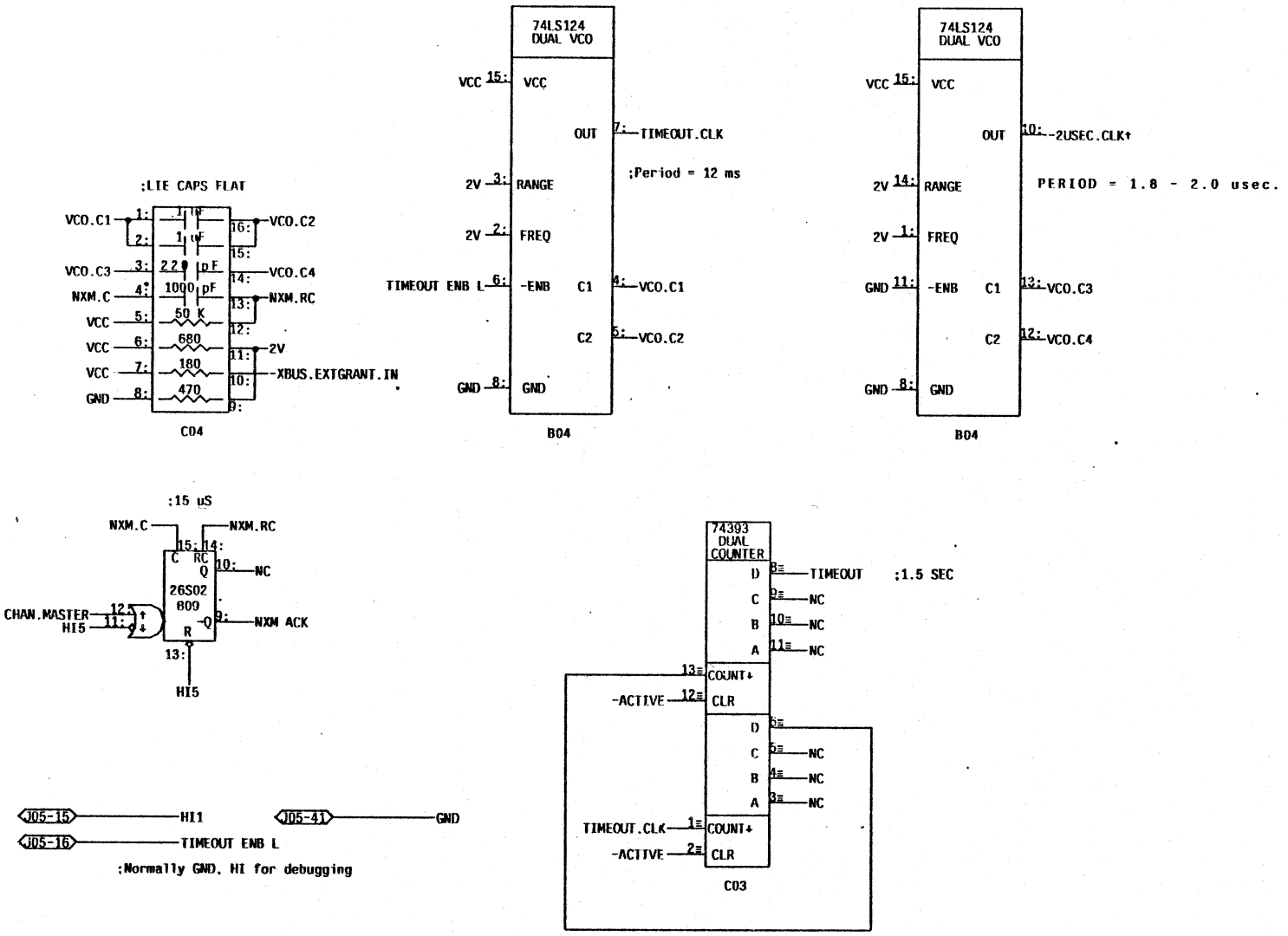
DISK CONTROL	REGISTER ADDRESSING	8-OCT-1978 16:56	AI: CADRDC; DCREG



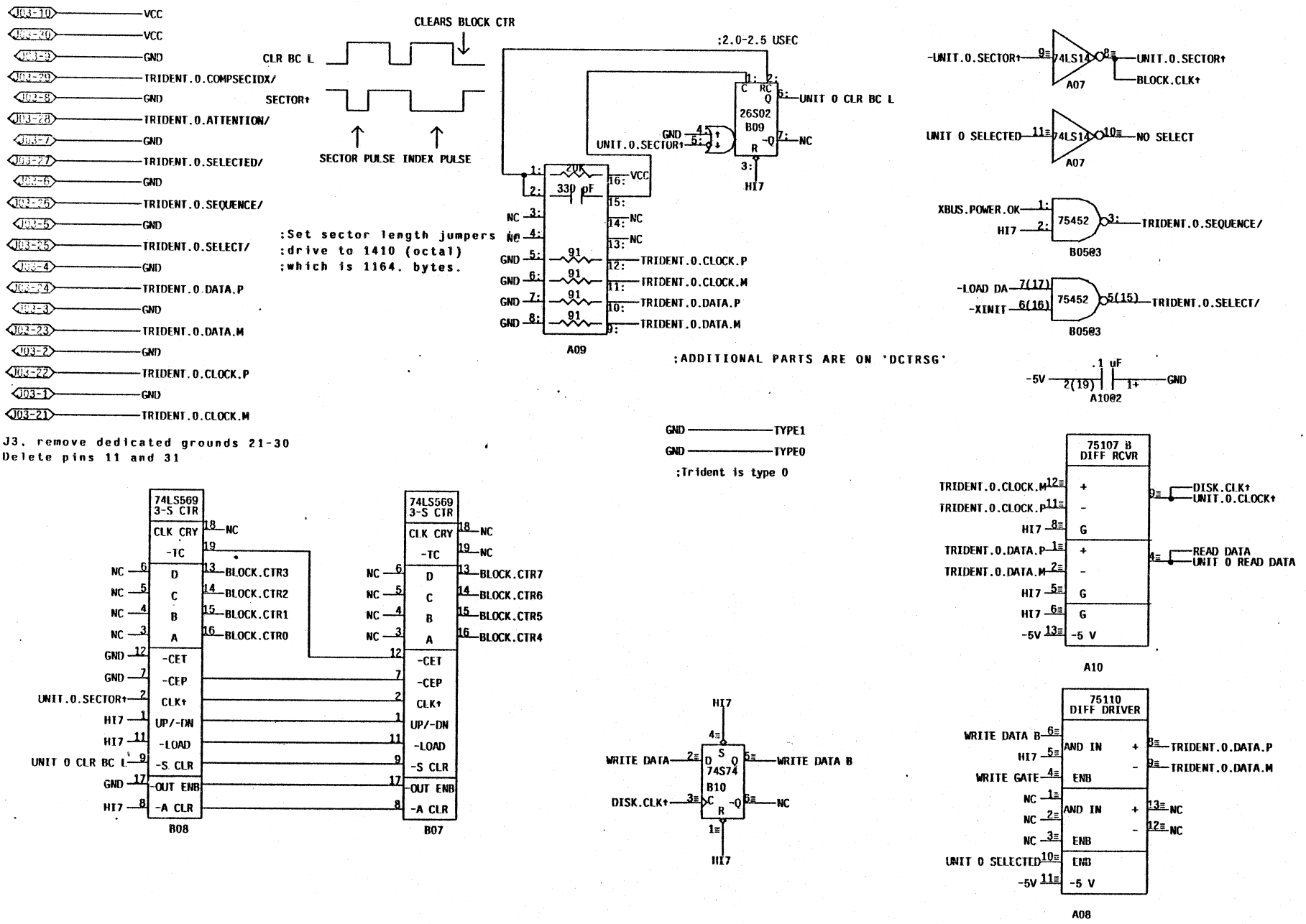
DISK CONTROL	SHIFT REGISTER	10-JUN-1979 20:37	AI: CADRDC; DCSH



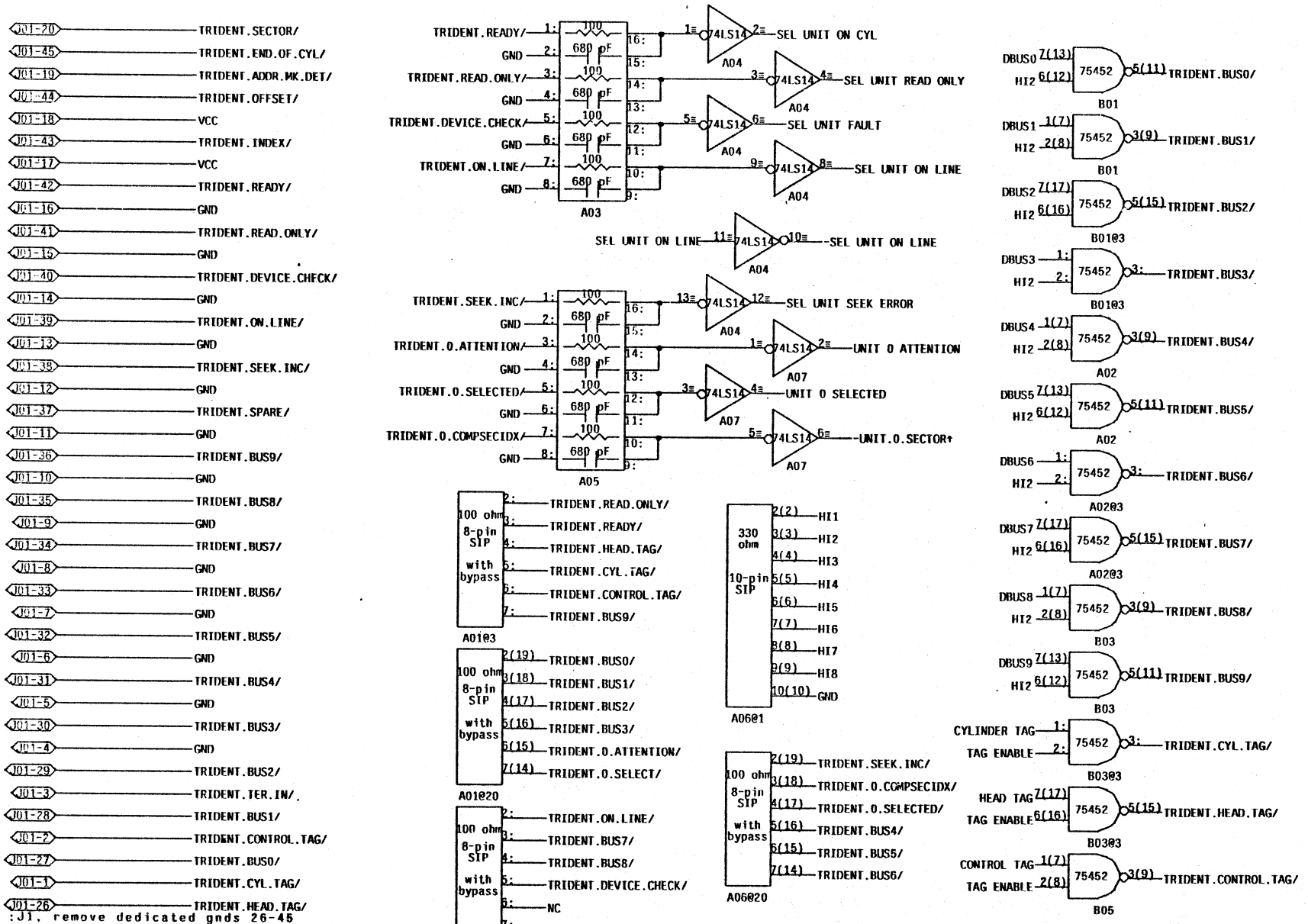
DISK CONTROL	STATUS	5-FEB-1980 01:50	AI: CADRDC; DCSTS



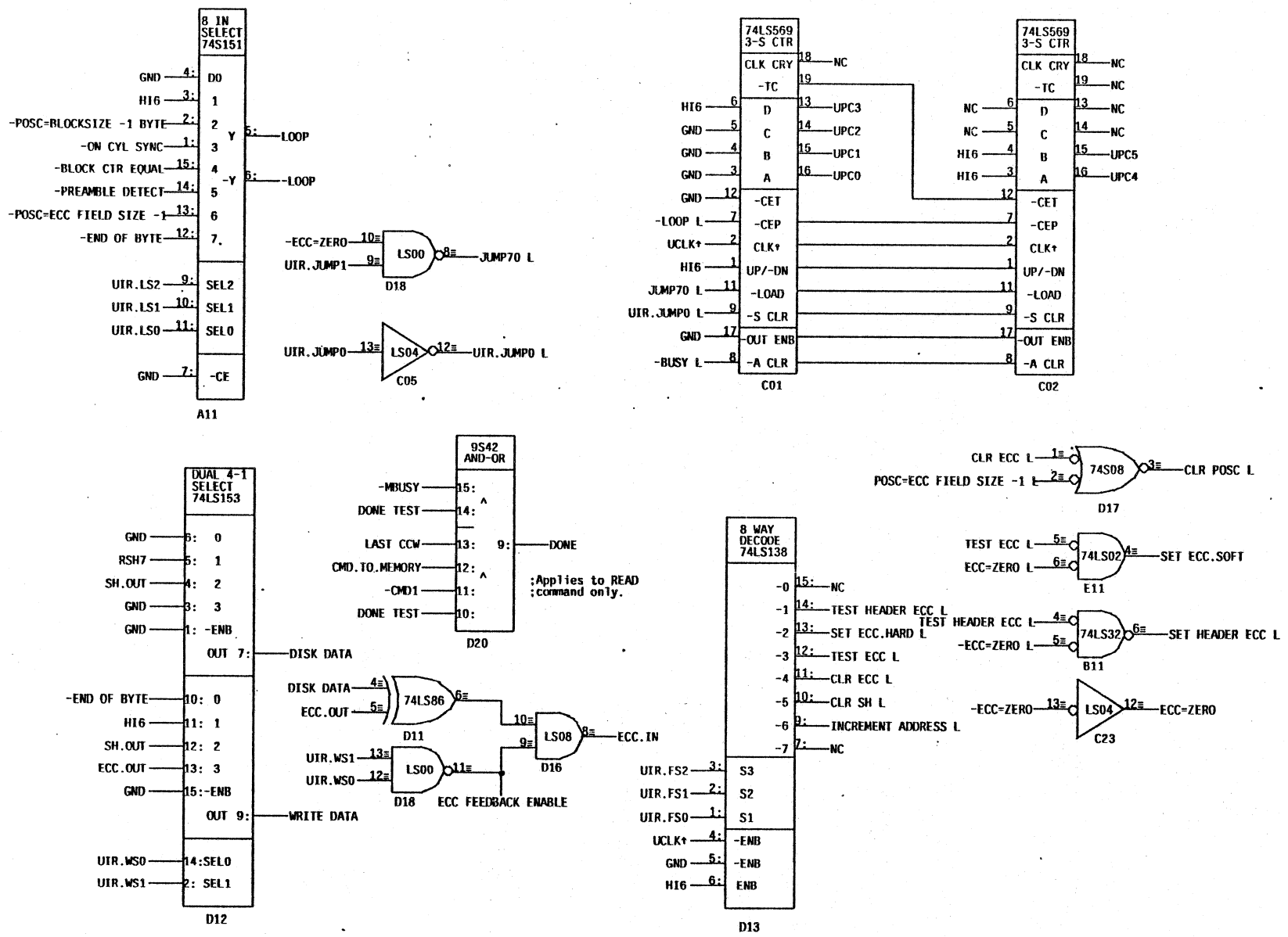
DISK CONTROL	TIMEOUT	7-OCT-1980 22:12	AI: CADRDC; DCTMOT

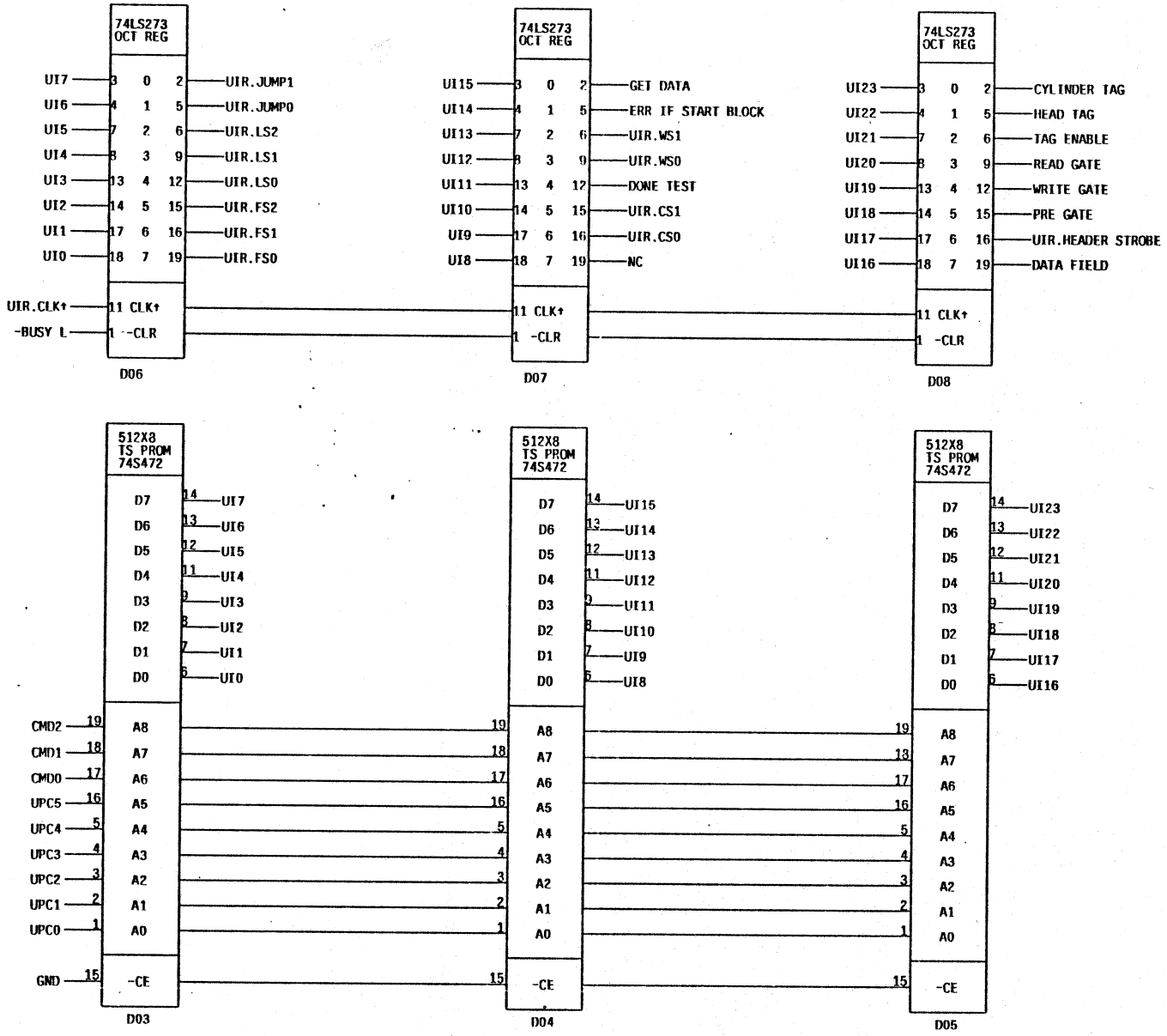


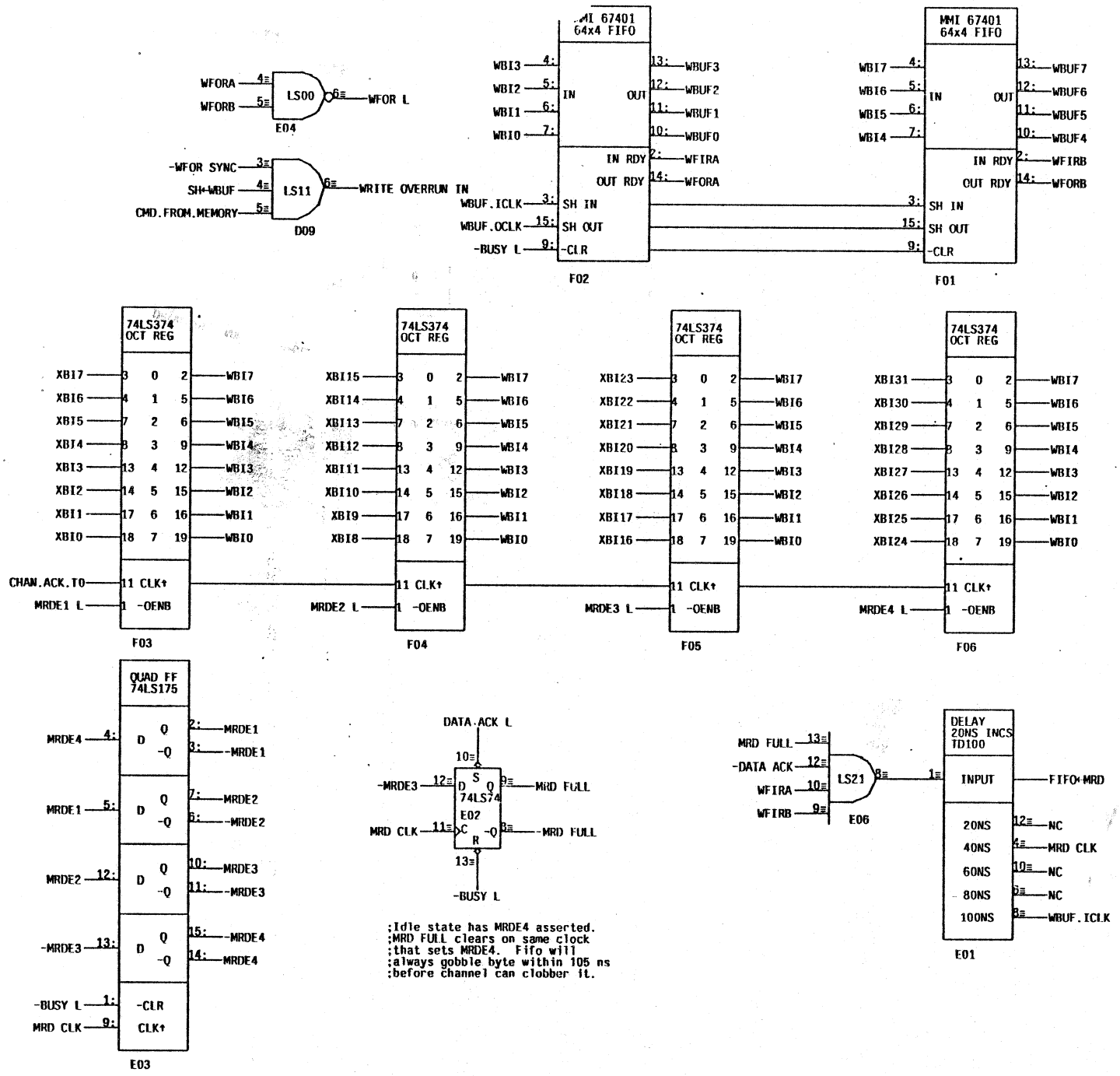
DISK CONTROL	SINGLE TRIDENT	10-JUN-1979 21:54	AI: CADRDC; DCTRID



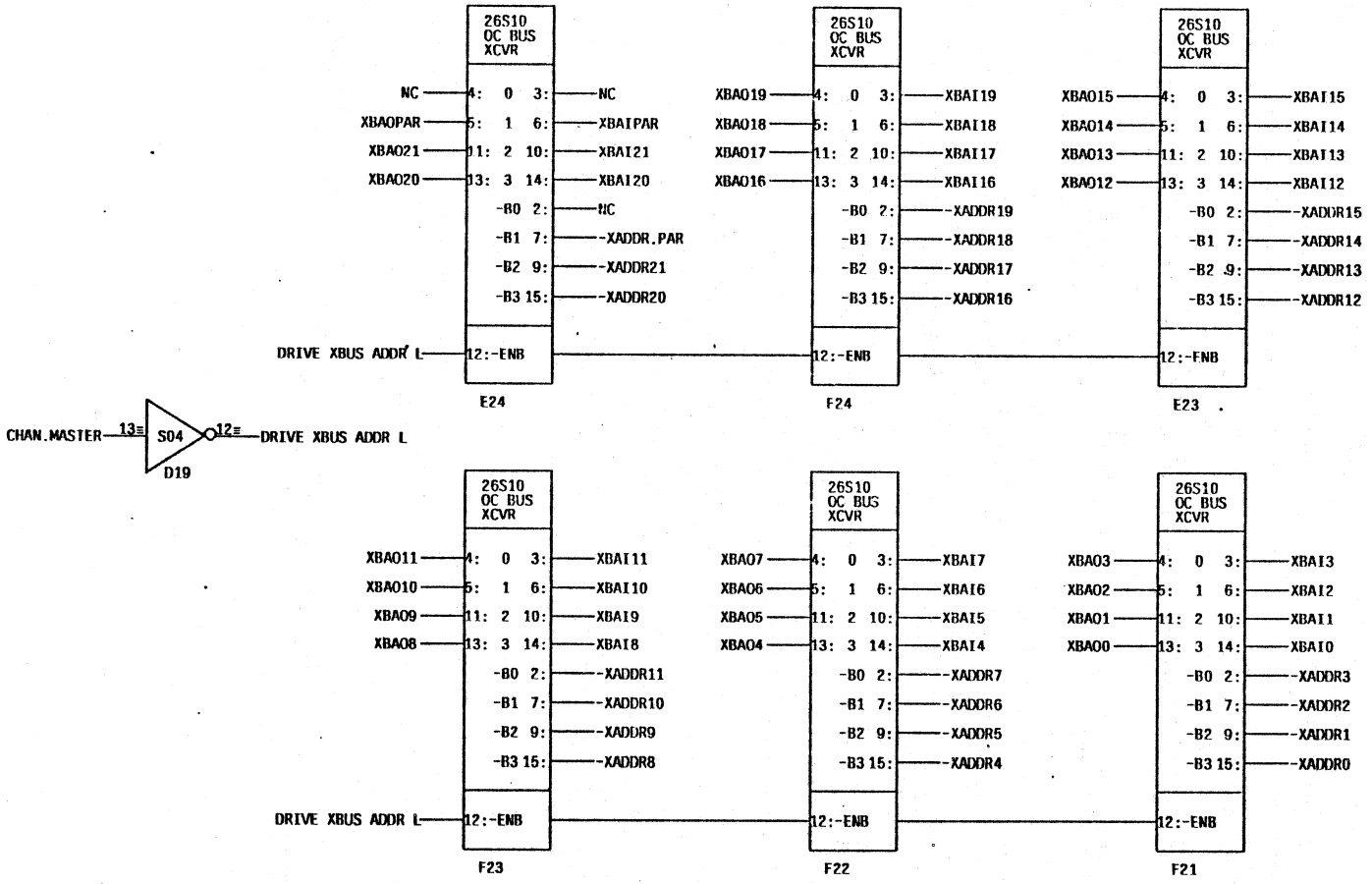
DISK CONTROL	TRIDENT SIGNAL CABLE	30-JUL-1979 02:11	AI: CADRDC; DCTRSG

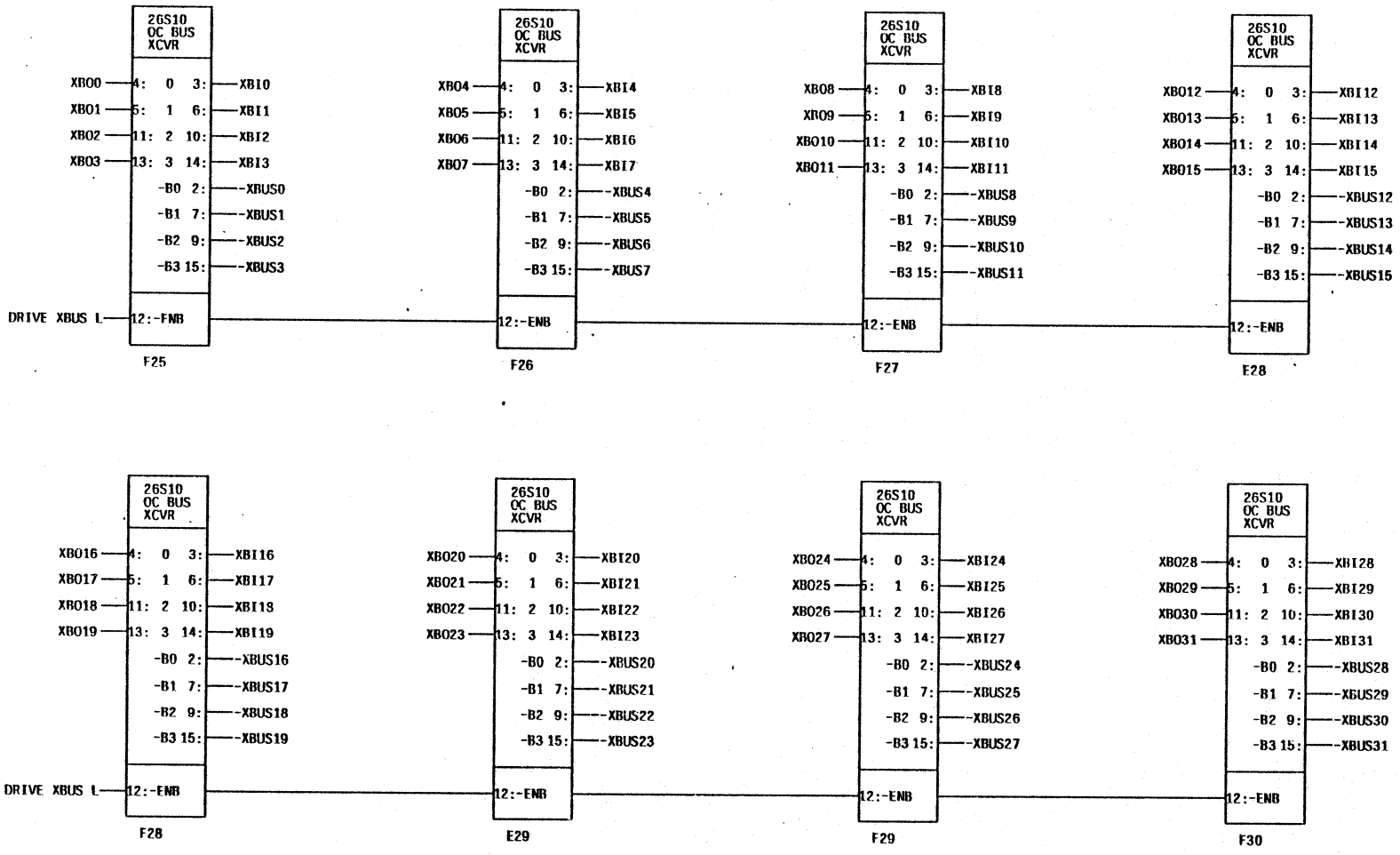

8 IN SELECT 74S151
GND 4:
HI6 3:
-POSC=BLOCKSIZE -1 BYTE 2:
-ON CYL SYNC 1:
-BLOCK CTR EQUAL 15:
-PREAMBLE DETECT 14:
-POSC=ECC FIELD SIZE -1 13:
-END OF BYTE 12:
D0
1
2
3
4
5
6
7.
Y
-Y
5: LOOP
6: -LOOP
UIR.LS2 9:
UIR.LS1 10:
UIR.LS0 11:
SEL2
SEL1
SEL0
GND 7:
-CE
A11
-ECC=ZERO 10≡
UIR.JUMP1 9≡
LS00
8≡ JUMP70 L
D18
UIR.JUMP0 13≡
LS04
12≡ UIR.JUMP0 L
C05
74LS569 3-S CTR
CLK CRY
-TC
18 NC
19
HI6 6
GND 5
GND 4
GND 3
GND 12
-LOOP L 7
UCLK↑ 2
HI6 1
JUMP70 L 11
UIR.JUMP0 L 9
GND 17
-BUSY L 8
D
C
B
A
-CET
-CEP
CLK↑
UP/-DN
-LOAD
-S CLR
-OUT ENB
-A CLR
13 UPC3
14 UPC2
15 UPC1
16 UPC0
C01
74LS569 3-S CTR
CLK CRY
-TC
18 NC
19 NC
NC 6
NC 5
HI6 4
HI6 3
12
7
2
1
11
9
17
8
D
C
B
A
-CET
-CEP
CLK↑
UP/-DN
-LOAD
-S CLR
-OUT ENB
-A CLR
13 NC
14 NC
15 UPC5
16 UPC4
C02
DUAL 4-1 SELECT 74LS153
GND 6: 0
RSH7 5: 1
SH.OUT 4: 2
GND 3: 3
GND 1: -ENB
OUT 7: DISK DATA
-END OF BYTE 10: 0
HI6 11: 1
SH.OUT 12: 2
ECC.OUT 13: 3
GND 15:-ENB
OUT 9: WRITE DATA
UIR.WS0 14:SEL0
UIR.WS1 2: SEL1
D12
9S42 AND-OR
-MBUSY 15:
DONE TEST 14:
LAST CCW 13:
CMD.TO.MEMORY 12:
-CMD1 11:
DONE TEST 10:
9: DONE
;Applies to READ
;command only.
D20
DISK DATA 4≡
ECC.OUT 5≡
74LS86
6≡
D11
UIR.WS1 13≡
UIR.WS0 12≡
LS00
11≡
D18
ECC FEEDBACK ENABLE
10≡
9≡
LS08
8≡ ECC.IN
D16
8 WAY DECODE 74LS138
-0 15: NC
-1 14: TEST HEADER ECC L
-2 13: SET ECC.HARD L
-3 12: TEST ECC L
-4 11: CLR ECC L
-5 10: CLR SH L
-6 9: INCREMENT ADDRESS L
-7 7: NC
UIR.FS2 3:
UIR.FS1 2:
UIR.FS0 1:
UCLK↑ 4:
GND 5:
HI6 6:
S3
S2
S1
-ENB
-ENB
ENB
D13
CLR ECC L 1≡
POSC=ECC FIELD SIZE -1 L 2≡
74S08
3≡ CLR POSC L
D17
TEST ECC L 5≡
ECC=ZERO L 6≡
74LS02
4≡ SET ECC.SOFT
E11
TEST HEADER ECC L 4≡
-ECC=ZERO L 5≡
74LS32
6≡ SET HEADER ECC L
B11
-ECC=ZERO 13≡
LS04
12≡ ECC=ZERO
C23


74LS273
OCT REG
UI7
UI6
UI5
UI4
UI3
UI2
UI1
UI0
UIR.JUMP1
UIR.JUMP0
UIR.LS2
UIR.LS1
UIR.LS0
UIR.FS2
UIR.FS1
UIR.FS0
UIR.CLK↑
-BUSY L
CLK↑
-CLR
D06
UI15
UI14
UI13
UI12
UI11
UI10
UI9
UI8
GET DATA
ERR IF START BLOCK
UIR.WS1
UIR.WS0
DONE TEST
UIR.CS1
UIR.CS0
NC
D07
UI23
UI22
UI21
UI20
UI19
UI18
UI17
UI16
CYLINDER TAG
HEAD TAG
TAG ENABLE
READ GATE
WRITE GATE
PRE GATE
UIR.HEADER STROBE
DATA FIELD
D08
512X8
TS PROM
74S472
D7
D6
D5
D4
D3
D2
D1
D0
A8
A7
A6
A5
A4
A3
A2
A1
A0
-CE
CMD2
CMD1
CMD0
UPC5
UPC4
UPC3
UPC2
UPC1
UPC0
GND
D03
D04
D05

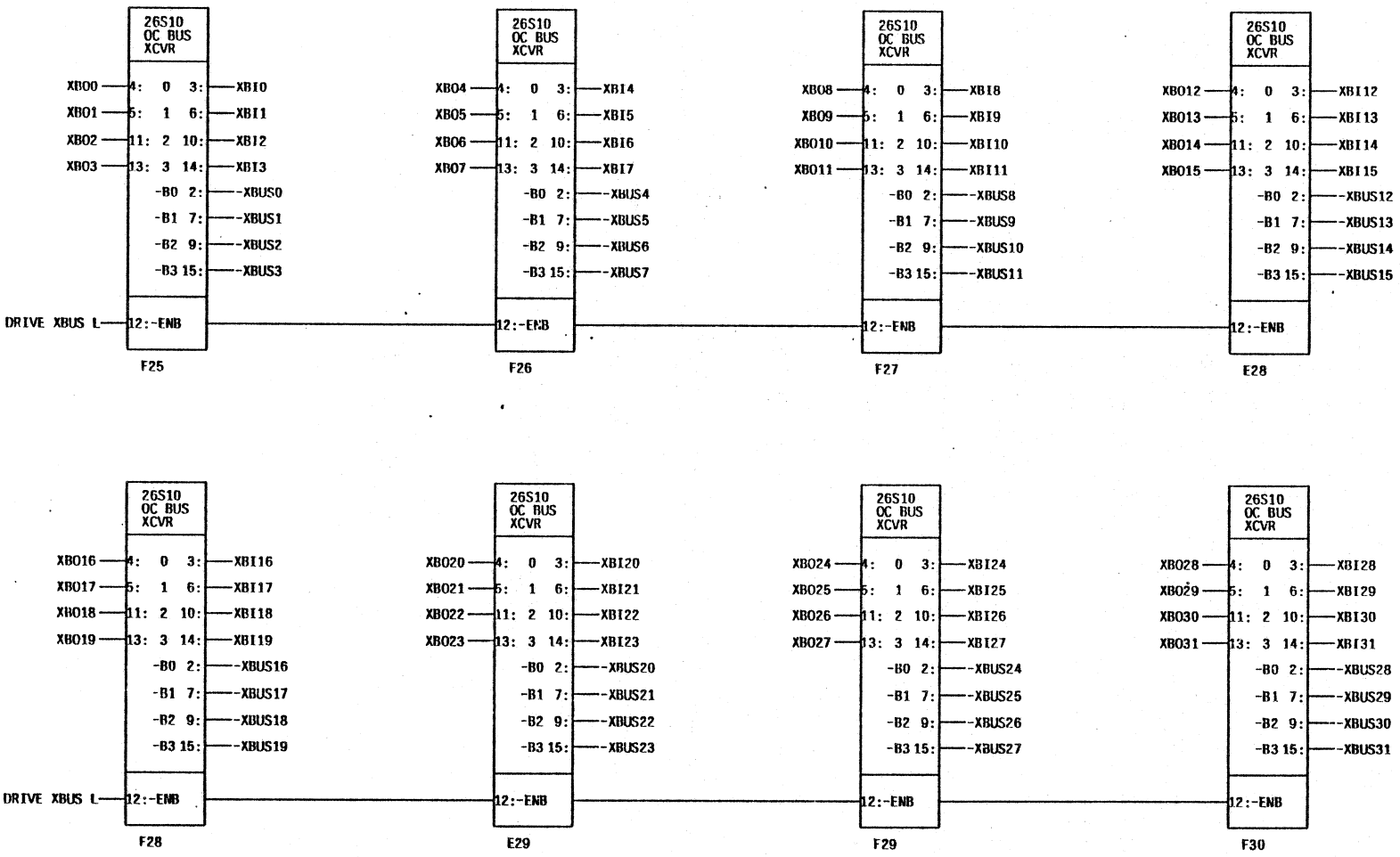


DISK CONTROL	WRITE BUFFER	30-SEP-1978 02:27	AI: CADRDC; DCWBUF

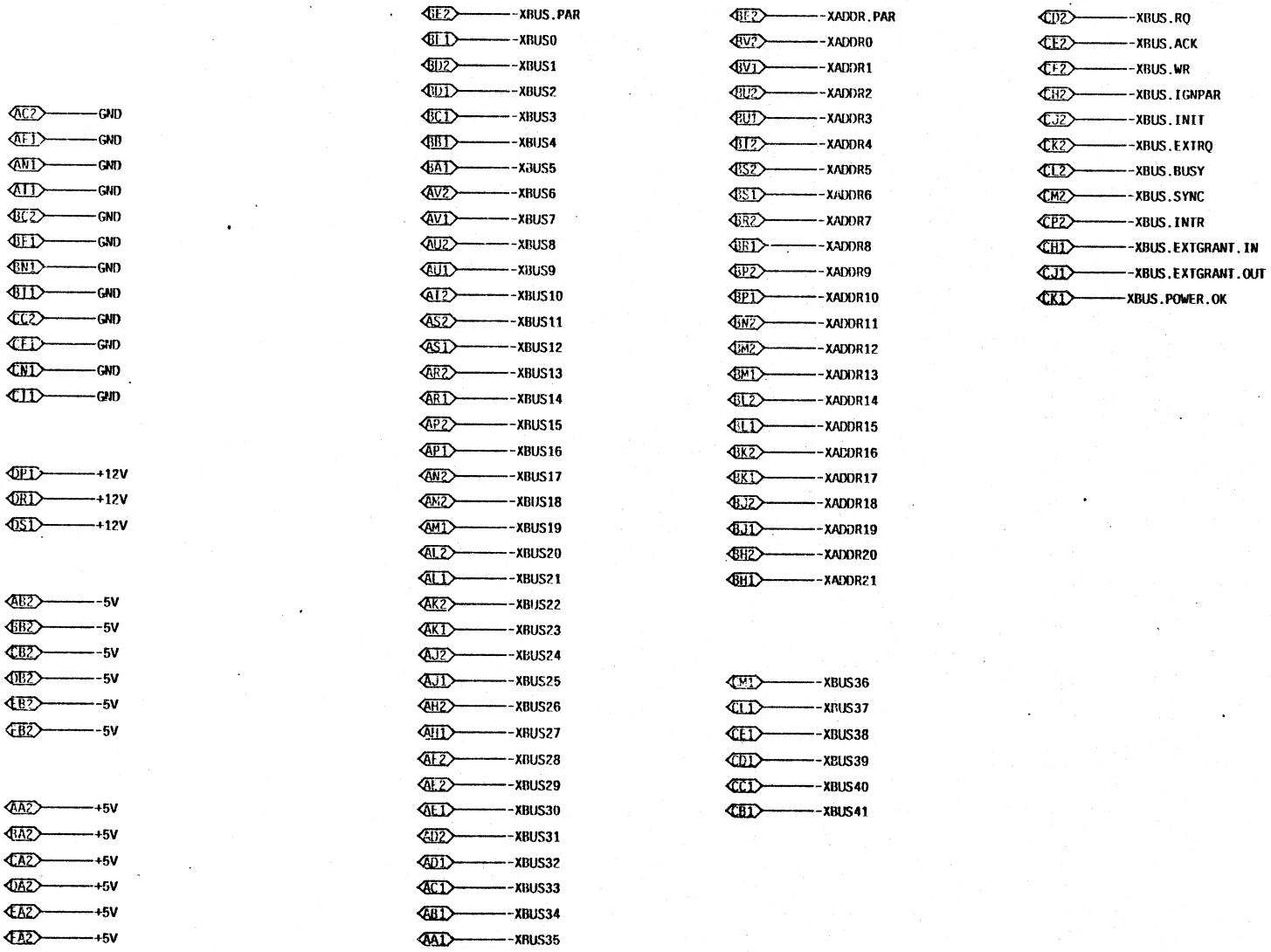

26S10
OC BUS
XCVR
NC
XBAOPAR
XBAO21
XBAO20
4: 0 3:
5: 1 6:
11: 2 10:
13: 3 14:
-B0 2:
-B1 7:
-B2 9:
-B3 15:
NC
XBAIPAR
XBAI21
XBAI20
NC
-XADDR.PAR
-XADDR21
-XADDR20
DRIVE XBUS ADDR L
12:-ENB
E24
XBAO19
XBAO18
XBAO17
XBAO16
XBAI19
XBAI18
XBAI17
XBAI16
-XADDR19
-XADDR18
-XADDR17
-XADDR16
F24
XBAO15
XBAO14
XBAO13
XBAO12
XBAI15
XBAI14
XBAI13
XBAI12
-XADDR15
-XADDR14
-XADDR13
-XADDR12
E23
CHAN.MASTER
13
S04
12
DRIVE XBUS ADDR L
D19
XBAO11
XBAO10
XBAO9
XBAO8
XBAI11
XBAI10
XBAI9
XBAI8
-XADDR11
-XADDR10
-XADDR9
-XADDR8
F23
XBAO7
XBAO6
XBAO5
XBAO4
XBAI7
XBAI6
XBAI5
XBAI4
-XADDR7
-XADDR6
-XADDR5
-XADDR4
F22
XBAO3
XBAO2
XBAO1
XBAO0
XBAI3
XBAI2
XBAI1
XBAI0
-XADDR3
-XADDR2
-XADDR1
-XADDR0
F21



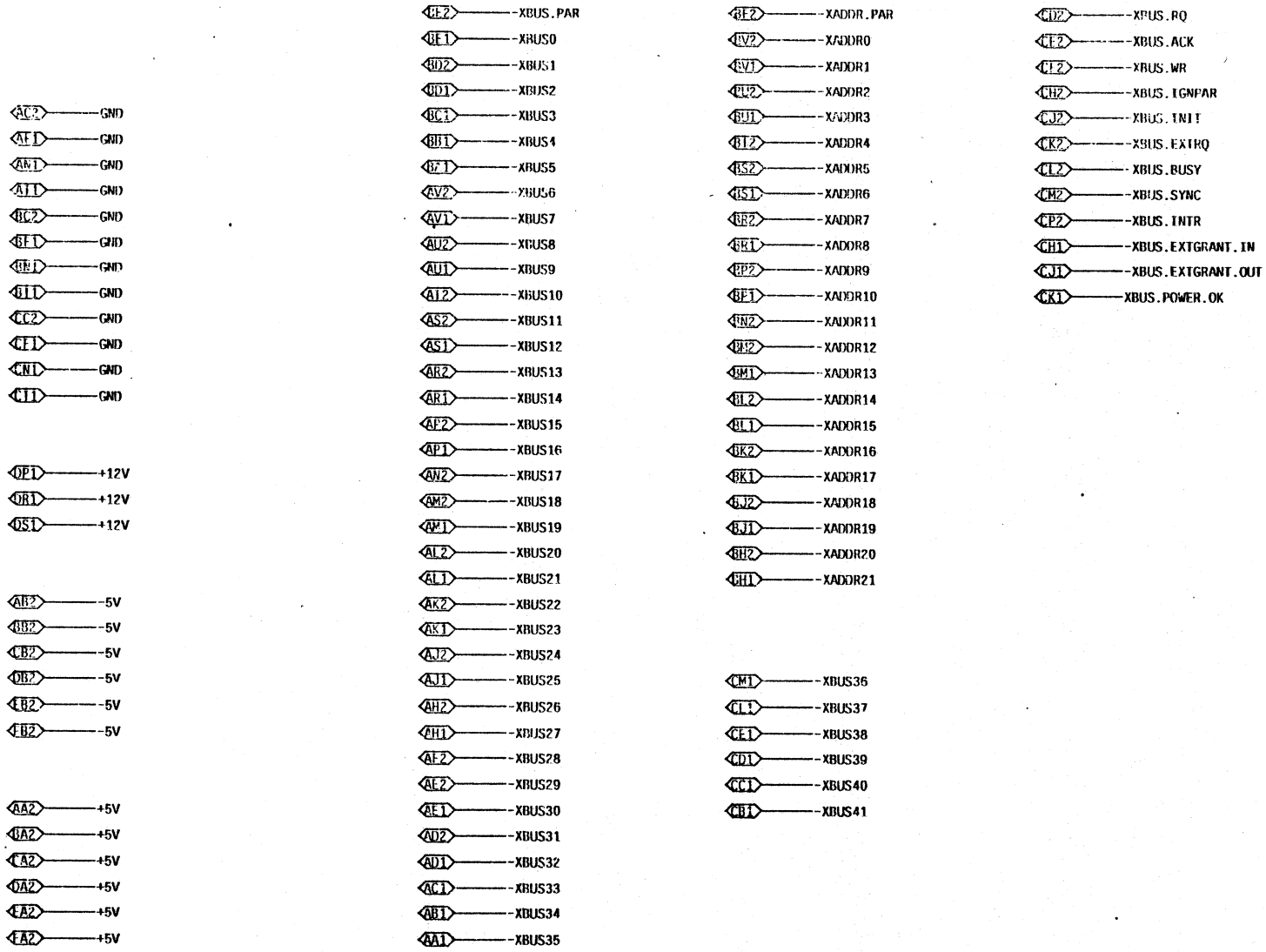
DISK CONTROL	XBUS DATA TRANSCEIVERS	30-SEP-1978 02:55	AI: CADRDC; DCXBUS



DISK CONTROL	XBUS DATA TRANSCEIVERS	30-SEP-1978 02:55	AI: CADRDC; DCXBUS



XBUS	BACKPLANE CONNECTIONS	19-DEC-1979 00:54	AI: CADRDC; XBUS



XBUS	BACKPLANE CONNECTIONS	19-DEC-1979 00:54	AI: CADRDC; XBUS

LISP Machine Disk Controller CADRDC;DC UML 10-DEC-80 1753
******* DIP MAP *******

F30	E30	D30	C30	B30	A30
26S10 DCXBUS x	74S260 DCECC xx	74LS273 DCECC x	74LS374 DCECC x	74S163 DCPOSC x	74S240 DCSTS x
F29	E29	D29	C29	B29	A29
26S10 DCXBUS x	26S10 DCXBUS x	74LS273 DCECC x	74LS374 DCECC x	74S163 DCPOSC x	74S133 DCPOSC x
F28	E28	D28	C28	B28	A28
26S10 DCXBUS x	26S10 DCXBUS x	74LS273 DCECC x	74S260 DCECC xx	74S163 DCPOSC x	74LS00 DCCMD xxxx
F27	E27	D27	C27	B27	A27
26S10 DCXBUS x	74LS86 DCECC xxxx	74LS273 DCECC x	74S133 DCECC x	74S163 DCPOSC x	74LS153 DCHDCM x
F26	E26	D26	C26	B26	A26
26S10 DCXBUS x	74LS374 DCCCW x	74LS374 DCCLP x	74S260 DCPOSC xx	74S133 DCPOSC x	74LS153 DCHDCM x
F25	E25	D25	C25	B25	A25
26S10 DCXBUS x	74LS374 DCCCW x	74LS569 DCCLP x	74LS374 DCCLP x	74LS244 DCPOSC x	74LS153 DCHDCM x
F24	E24	D24	C24	B24	A24
26S10 DCXBSA x	26S10 DCXBSA x	74LS273 DCSTS x	74LS374 DCCLP x	74LS244 DCPOSC x	74LS153 DCHDCM x
F23	E23	D23	C23	B23	A23
26S10 DCXBSA x	26S10 DCXBSA x	74LS569 DCCLP x	74LS04 DCCLP xxxxxx	74LS374 DCCLP x	25LS2536 DCHDCM x
F22	E22	D22	C22	B22	A22
26S10 DCXBSA x	74LS569 DCCCW x	74LS569 DCCLP x	25LS2521 DCHDCM x	74LS193 DCDA x	-------- --------
F21	E21	D21	C21	B21	A21
26S10 DCXBSA x	74LS569 DCCCW x	74LS569 DCCLP x	74LS175 DCCMD x	74LS193 DCDA x	74LS193 DCDA x
F20	E20	D20	C20	B20	A20
26S10 DCPAR x	74LS74 DCCCW xx	9S42 DCUC xx	TD250 DCCHAN x	74LS244 DCDA x	74LS193 DCDA x

LISP Machine Disk Controller CADRDC;DC UML 10-DEC-80 1754
******* DIP MAP *******

26S10 DCCHAN x F19	TD100 DCCHAN x E19	74S04 DCXBSA xxxxxx D19	74LS10 DCCMD xxx C19	74LS244 DCDA x B19	74LS193 DCDA x A19
26S10 DCCHAN x F18	74S11 DCCHAN xxx E18	74LS00 DCUC xxxx D18	74S51 DCCLK xx C18	74LS244 DCDA x B18	74LS193 DCDA x A18
9S42 DCCHAN xx F17	74S74 DCCHAN xx E17	74S08 DCUC xxxx D17	74S37 DCCLK xxxx C17	74LS244 DCDA x B17	74LS193 DCDA x A17
74LS74 DCCLK xx F16	74S02 DCREG xxxx E16	74LS08 DCUC xxxx D16	74LS32 DCCMD xxxx C16	74LS02 DCCLP xxxx B16	74LS86 DCSH xxxx A16
93S48 DCPAR x F15	74S133 DCREG x E15	74LS74 DCCLK xx D15	74LS08 DCRBUF xxxx C15	74LS21 DCBUSY xo B15	74LS244 DCSTS x A15
93S48 DCPAR x F14	25LS2521 DCREG x E14	74LS08 DCSH xxxx D14	74LS74 DCCLK xx C14	74S74 DCBUSY xx B14	74LS244 DCSTS x A14
93S48 DCPAR x F13	TD250 DCREG x E13	74LS138 DCUC x D13	74S74 DCCLK xx C13	74S260 DCBUSY xx B13	74LS244 DCSTS x A13
93S48 DCPAR x F12	74S138 DCREG x E12	74LS153 DCUC x D12	74LS273 DCSTS x C12	74LS279 DCSTS x B12	74LS244 DCSTS x A12
93S48 DCPAR x F11	74LS02 DCREG xxxx E11	74LS86 DCPAR xxxx D11	25LS2521 DCHDCM x C11	74LS32 DCSH xxxx B11	74S151 DCUC x A11
74LS374 DCRBUF x F10	67401 DCRBUF x E10	74S299 DCSH x D10	74LS273 DCCMD x C10	74S74 DCSH xx B10	75107 DCTRID xx A10
74LS374 DCRBUF x F09	67401 DCRBUF x E09	74LS11 DCCCW xxx D09	74LS244 DCDBUS x C09	26S02 DCTMOT xx B09	DUMMY DCTRID x A09

LISP Machine Disk Controller CADRDC;DC UML 10-DEC-80 1758
******* DIP MAP *******

F08	E08	D08	C08	B08	A08
74LS374 DCRBUF x	74LS175 DCRBUF x	74LS273 DCUI x	74LS244 DCDBUS x	74LS569 DCTRID x	75110 DCTRID x
F07	E07	D07	C07	B07	A07
74LS374 DCRBUF x	TD100 DCRBUF x	74LS273 DCUI x	74LS244 DCDBUS x	74LS569 DCTRID x	74LS14 DCTRID xxxxxo
F06	E06	D06	C06	B06	A06
74LS374 DCWBUF x	74LS21 DCRBUF xx	74LS273 DCUI x	74LS157 DCDBUS x	74LS74 DCBUSY xo	SIP330-1 DCTRSG x
F05	E05	D05	C05	B05	A05
74LS374 DCWBUF x	74LS08 DCSH xxxx	74S472 DCUI x	74LS04 DCDBUS xxxxxx	75452 DCTRSG xx	DUMMY DCTRSG x
F04	E04	D04	C04	B04	A04
74LS374 DCWBUF x	74LS00 DCSH xxox	74S472 DCUI x	DUMMY DCTMOT x	74LS124 DCTMOT xx	74LS14 DCTRSG xxxxxx
F03	E03	D03	C03	B03	A03
74LS374 DCWBUF x	74LS175 DCWBUF x	74S472 DCUI x	74 393 DCTMOT xx	75452 DCTRSG xx	DUMMY DCTRSG x
F02	E02	D02	C02	B02	A02
67401 DCWBUF x	74LS74 DCRBUF xx	-------- --------	74LS569 DCUC x	SIP100-8 DCTRSG x	75452 DCTRSG xx
F01	E01	D01	C01	B01	A01
67401 DCWBUF x	TD100 DCWBUF x	74S299 DCSH x	74LS569 DCUC x	75452 DCTRSG xx	SIP100-8 DCTRSG x

LISP Machine Disk Controller CADRDC;DC UML 10-DEC-80 1759
******* EDGE CONNECTIONS Flags: (# Output, @ Terminator, ---- Dedicated ground, ++++ Dedicated power) *******

-A-		-B-		-C-		-D-	
A1 -XBUS35		A1 -XBUS5	#	A1		A1	
A2 +5.0V++++++++++++++++++		A2 +5.0V++++++++++++++++++		A2 +5.0V++++++++++++++++++		A2 +5.0V++++++++++++++++++	
B1 -XBUS34		B1 -XBUS4	#	B1 -XBUS41		B1	
B2 -5.0V		B2 -5.0V		B2 -5.0V		B2 -5.0V	
C1 -XBUS33		C1 -XBUS3	#	C1 -XBUS40		C1	
C2 GND----------	@	C2 GND----------	@	C2 GND----------	@	C2 GND----------	@
D1 -XBUS32		D1 -XBUS2	#	D1 -XBUS39		D1	
D2 -XBUS31	#	D2 -XBUS1	#	D2 -XBUS.RQ	#	D2 XINIT	#
E1 -XBUS30	#	E1 -XBUS0	#	E1 -XBUS38		E1	
E2 -XBUS29	#	E2 -XBUS.PAR	#	E2 -XBUS.ACK	#	E2 SEL UNIT ATTENTION	
F1 GND	@	F1 GND	@	F1 GND	@	F1 GND	@
F2 -XBUS28	#	F2 -XADDR.PAR	#	F2 -XBUS.WR	#	F2 ANY ATTENTION	
H1 -XBUS27	#	H1 -XADDR21	#	H1 -XBUS.EXTGRANT.IN	#	H1	
H2 -XBUS26	#	H2 -XADDR20	#	H2 -XBUS.IGNPAR	#	H2 UNIT 0 ATTENTION	#
J1 -XBUS25	#	J1 -XADDR19	#	J1 -XBUS.EXTGRANT.OUT	#	J1	
J2 -XBUS24	#	J2 -XADDR18	#	J2 -XBUS.INIT	#	J2 -LOAD DA H	#
K1 -XBUS23	#	K1 -XADDR17	#	K1 XBUS.POWER.OK		K1	
K2 -XBUS22	#	K2 -XADDR16	#	K2 -XBUS.EXTRQ	#	K2 NC	#
L1 -XBUS21	#	L1 -XADDR15	#	L1 -XBUS37		L1	
L2 -XBUS20	#	L2 -XADDR14	#	L2 -XBUS.BUSY	#	L2 NO SELECT	#
M1 -XBUS19	#	M1 -XADDR13	#	M1 -XBUS36		M1	
M2 -XBUS18	#	M2 -XADDR12	#	M2 -XBUS.SYNC	#	M2 MULTIPLE SELECT	
N1 GND	@	N1 GND	@	N1 GND	@	N1 GND	@
N2 -XBUS17	#	N2 -XADDR11	#	N2		N2 WRITE DATA	#
P1 -XBUS16	#	P1 -XADDR10	#	P1		P1 +12.0V	
P2 -XBUS15	#	P2 -XADDR9	#	P2 -XBUS.INTR	#	P2 WRITE GATE	#
R1 -XBUS14	#	R1 -XADDR8	#	R1		R1 +12.0V	
R2 -XBUS13	#	R2 -XADDR7	#	R2		R2 DISK.CLK^	#
S1 -XBUS12	#	S1 -XADDR6	#	S1		S1 +12.0V	
S2 -XBUS11	#	S2 -XADDR5	#	S2		S2 READ DATA	#
T1 GND----------	@	T1 GND----------	@	T1 GND----------	@	T1 GND----------	@
T2 -XBUS10	#	T2 -XADDR4	#	T2		T2 BLOCK.CLK^	#
U1 -XBUS9	#	U1 -XADDR3	#	U1		U1	
U2 -XBUS8	#	U2 -XADDR2	#	U2		U2 BLOCK.CTR0	#
V1 -XBUS7	#	V1 -XADDR1	#	V1		V1	
V2 -XBUS6	#	V2 -XADDR0	#	V2		V2 BLOCK.CTR1	#

LISP Machine Disk Controller CADRDC;DC UML 10-DEC-80 1759
******* EDGE CONNECTIONS Flags: (# Output, @ Terminator, ---- Dedicated ground, ++++ Dedicated power) *******

-E-		-F-		-J01-		-J02-
A1		A1		01 TRIDENT.CYL.TAG/	#	01
A2 +5.0V++++++++++++++++++		A2 +5.0V++++++++++++++++++		02 TRIDENT.CONTROL.TAG/	#	02
B1		B1		03 TRIDENT.TER.IN/		03
B2 -5.0V		B2 -5.0V		04 GND	@	04
C1		C1		05 GND	@	05
C2 GND----------	@	C2 GND----------	@	06 GND	@	06
D1		D1		07 GND	@	07
D2 BLOCK.CTR2	#	D2		08 GND	@	08
E1		E1		09 GND	@	09
E2 BLOCK.CTR3	#	E2		10 GND	@	10
F1 GND	@	F1 GND	@	11 GND	@	11
F2 BLOCK.CTR4	#	F2		12 GND	@	12
H1		H1		13 GND	@	13
H2 BLOCK.CTR5	#	H2		14 GND	@	14
J1		J1		15 GND	@	15
J2 BLOCK.CTR6	#	J2		16 GND	@	16
K1		K1		17 +5.0V		17
K2 BLOCK.CTR7	#	K2		18 +5.0V		18
L1		L1		19 TRIDENT.ADDR.MK.DET/		19
L2 XBI28	#	L2		20 TRIDENT.SECTOR/		20
M1		M1		21		21
M2 XBI29	#	M2		22		22
N1 GND	@	N1 GND	@	23		23
N2 XBI30	#	N2		24		24
P1		P1		25		25
P2 UNIT0		P2		26 TRIDENT.HEAD.TAG/------	#	26 ----------
R1		R1		27 TRIDENT.BUS0/----------	#	27 ----------
R2 UNIT1		R2		28 TRIDENT.BUS1/----------	#	28 ----------
S1		S1		29 TRIDENT.BUS2/----------	#	29 ----------
S2 UNIT2		S2		30 TRIDENT.BUS3/----------	#	30 ----------
T1 GND----------	@	T1 GND----------	@	31 TRIDENT.BUS4/----------	#	31 ----------
T2 -CYLINDER TAG H	#	T2		32 TRIDENT.BUS5/----------	#	32 ----------
U1		U1		33 TRIDENT.BUS6/----------	#	33 ----------
U2 -HEAD TAG H	#	U2		34 TRIDENT.BUS7/----------	#	34 ----------
V1		V1		35 TRIDENT.BUS8/----------	#	35 ----------
V2		V2		36 TRIDENT.BUS9/----------	#	36 ----------
				37 TRIDENT.SPARE/----------		37 ----------
				38 TRIDENT.SEEK.INC/------	@	38 ----------
				39 TRIDENT.ON.LINE/-------	@	39 ----------
				40 TRIDENT.DEVICE.CHECK/--	@	40 ----------
				41 TRIDENT.READ.ONLY/-----	@	41 ----------
				42 TRIDENT.READY/---------	@	42 ----------
				43 TRIDENT.INDEX/---------		43 ----------
				44 TRIDENT.OFFSET/--------		44 ----------
				45 TRIDENT.END.OF.CYL/----		45 ----------
				46 ----------		46 ----------
				47 ----------		47 ----------
				48 ----------		48 ----------
				49 ----------		49 ----------
				50 ----------		50 ----------

LISP Machine Disk Controller CADRDC;DC UML 10-DEC-80 1759
******* EDGE CONNECTIONS Flags: (# Output, @ Terminator, ---- Dedicated ground, ++++ Dedicated power) *******

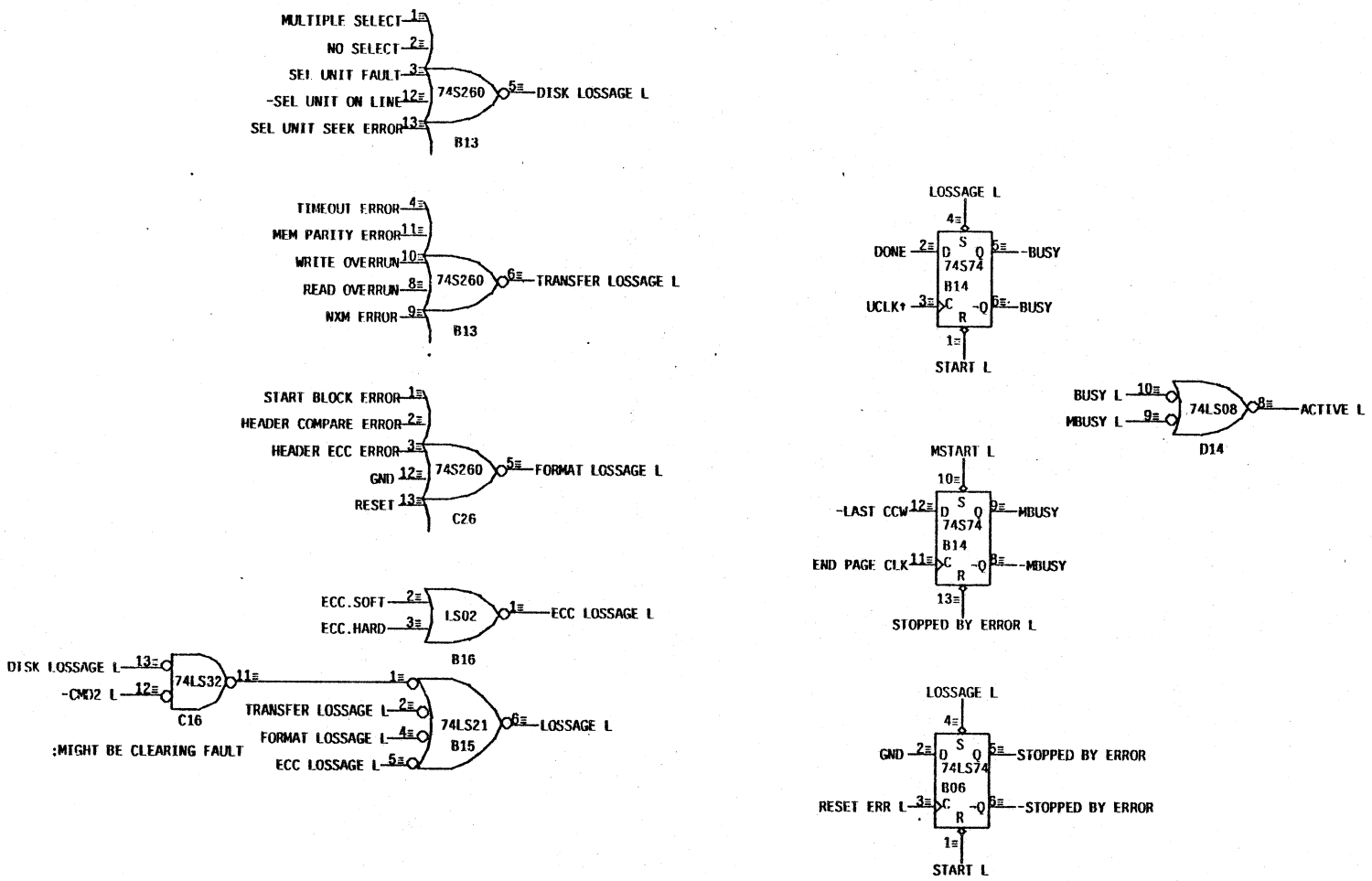
-J03-		-J04-		-J05-		-J06-	
01 GND	@	01		01 HI1	@	01	
02 GND	@	02		02 AD14		02	
03 GND	@	03		03 HI1	@	03	
04 GND	@	04		04 AD13		04	
05 GND	@	05		05 HI1	@	05	
06 GND	@	06		06 AD6		06	
07 GND	@	07		07 HI1	@	07	
08 GND	@	08		08 AD5		08	
09 GND	@	09		09 HI1	@	09	
10 +5.0V		10		10 AD4		10	
11		11		11 HI1	@	11	
12		12		12 AD3		12	
13		13		13 HI1	@	13	
14		14		14 AD2		14	
15		15		15 HI1	@	15	
16		16		16 -TIMEOUT ENB H		16	
17		17		17		17	
18		18		18		18	
19		19		19		19	
20		20		20		20	
21 TRIDENT.0.CLOCK.M------		21 ----------		21		21	
22 TRIDENT.0.CLOCK.P------		22 ----------		22		22	
23 TRIDENT.0.DATA.M-------	#	23 ----------		23		23	
24 TRIDENT.0.DATA.P-------	#	24 ----------		24		24	
25 TRIDENT.0.SELECT/------	#	25 ----------		25		25	
26 TRIDENT.0.SEQUENCE/----	#	26 ----------		26 ----------		26 ----------	
27 TRIDENT.0.SELECTED/----	@	27 ----------		27 GND----------	@	27 ----------	
28 TRIDENT.0.ATTENTION/---	@	28 ----------		28 ----------		28 ----------	
29 TRIDENT.0.COMPSECIDX/--	@	29 ----------		29 GND----------	@	29 ----------	
30 +5.0V----------		30 ----------		30 ----------		30 ----------	
31 ----------		31 ----------		31 GND----------	@	31 ----------	
32 ----------		32 ----------		32 ----------		32 ----------	
33 ----------		33 ----------		33 GND----------	@	33 ----------	
34 ----------		34 ----------		34 ----------		34 ----------	
35 ----------		35 ----------		35 GND----------	@	35 ----------	
36 ----------		36 ----------		36 ----------		36 ----------	
37 ----------		37 ----------		37 GND----------	@	37 ----------	
38 ----------		38 ----------		38 ----------		38 ----------	
39 ----------		39 ----------		39 GND----------	@	39 ----------	
40 ----------		40 ----------		40 ----------		40 ----------	
				41 GND----------	@	41 ----------	
				42 ----------		42 ----------	
				43 ----------		43 ----------	
				44 ----------		44 ----------	
				45 ----------		45 ----------	
				46 ----------		46 ----------	
				47 ----------		47 ----------	
				48 ----------		48 ----------	
				49 ----------		49 ----------	
				50 ----------		50 ----------	

LISP Machine Disk Controller CADRDC;DC UML 10-DEC-80 1800
******* EDGE CONNECTIONS Flags: (# Output, @ Terminator, ---- Dedicated ground, ++++ Dedicated power) *******

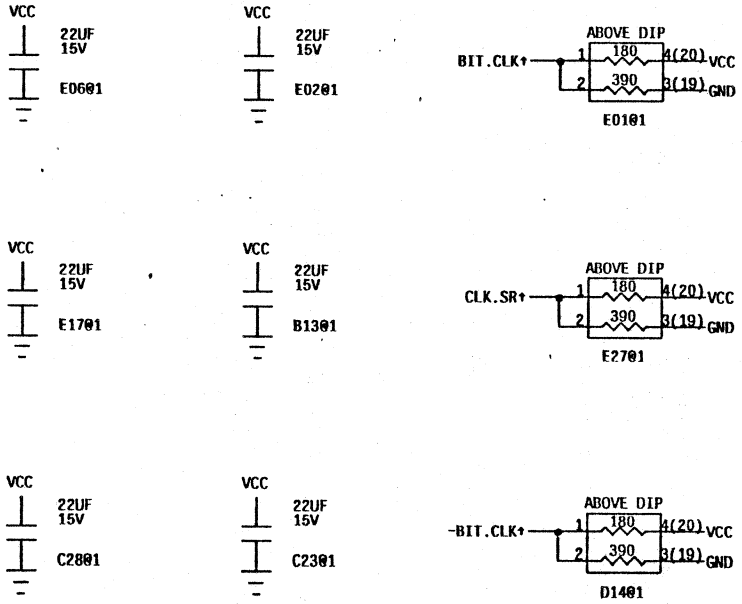
-J07-	-J08-	-J09-	-J10-
01 02	01 02	01 02	01 02
03 04	03 04	03 04	03 04
05 06	05 06	05 06	05 06
07 08	07 08	07 08	07 08
09 10	09 10	09 10	09 10
11 12	11 12	11 12	11 12
13 14	13 14	13 14	13 14
15 16	15 16	15 16	15 16
17 18	17 18	17 18	17 18
19 20	19 20	19 20	19 20
21 ---------- 22 ----------	21 ---------- 22 ----------	21 22	21 22
23 ---------- 24 ----------	23 ---------- 24 ----------	23 24	23 24
25 ---------- 26 ----------	25 ---------- 26 ----------	25 26 ----------	25 26 ----------
27 ---------- 28 ----------	27 ---------- 28 ----------	27 ---------- 28 ----------	27 ---------- 28 ----------
29 ---------- 30 ----------	29 ---------- 30 ----------	29 ---------- 30 ----------	29 ---------- 30 ----------
31 ---------- 32 ----------	31 ---------- 32 ----------	31 ---------- 32 ----------	31 ---------- 32 ----------
33 ---------- 34 ----------	33 ---------- 34 ----------	33 ---------- 34 ----------	33 ---------- 34 ----------
35 ---------- 36 ----------	35 ---------- 36 ----------	35 ---------- 36 ----------	35 ---------- 36 ----------
37 ---------- 38 ----------	37 ---------- 38 ----------	37 ---------- 38 ----------	37 ---------- 38 ----------
39 ---------- 40 ----------	39 ---------- 40 ----------	39 ---------- 40 ----------	39 ---------- 40 ----------
		41 ---------- 42 ----------	41 ---------- 42 ----------
		43 ---------- 44 ----------	43 ---------- 44 ----------
		45 ---------- 46 ----------	45 ---------- 46 ----------
		47 ---------- 48 ----------	47 ---------- 48 ----------
		49 ---------- 50 ----------	49 ---------- 50 ----------

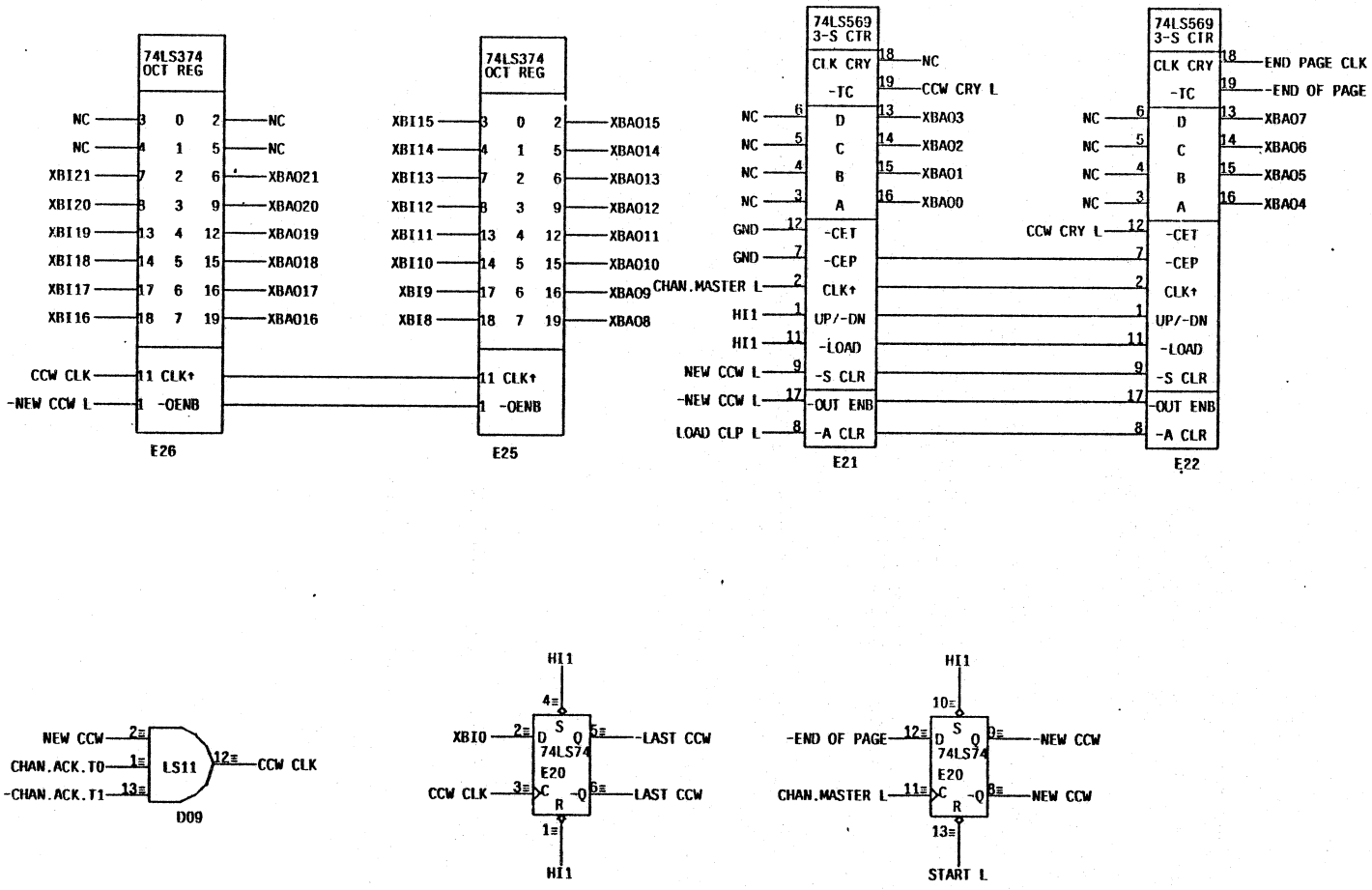
LISP Machine Disk Controller CADRDC;DC UML 10-DEC-80 1800
******* EDGE CONNECTIONS Flags: (# Output, @ Terminator, ---- Dedicated ground, ++++ Dedicated power) *******

-J11-		-J12-
01 LED.READ ACTIVE	#	01
02 LED.WRITE ACTIVE	#	02
03 LED.SEEK	#	03
04 LED.TRANSFER LOSSAGE	#	04
05 LED.FORMAT LOSSAGE	#	05
06 LED.ECC LOSSAGE	#	06
07 LED.DISK LOSSAGE	#	07
08 LED.UNUSED	#	08
09		09
10		10
11		11
12		12
13		13
14		14
15		15
16		16
17		17
18		18
19		19
20		20
21 ----------		21 ----------
22 ----------		22 ----------
23 ----------		23 ----------
24 ----------		24 ----------
25 ----------		25 ----------
26 ----------		26 ----------
27 ----------		27 ----------
28 ----------		28 ----------
29 ----------		29 ----------
30 ----------		30 ----------
31 ----------		31 ----------
32 ----------		32 ----------
33 ----------		33 ----------
34 ----------		34 ----------
35 ----------		35 ----------
36 ----------		36 ----------
37 ----------		37 ----------
38 ----------		38 ----------
39 ----------		39 ----------
40 ----------		40 ----------

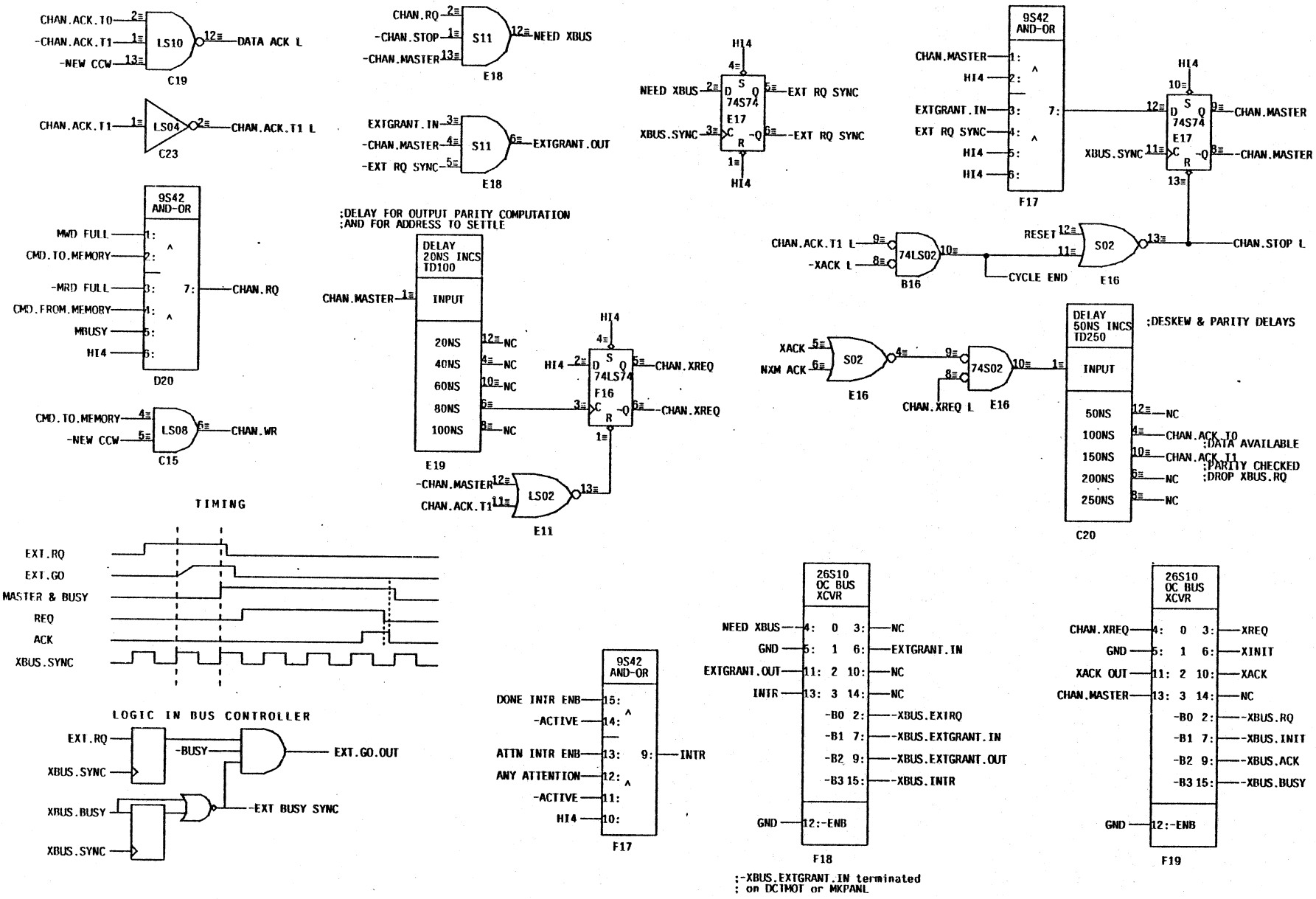


DISK CONTROL	BUSY, LOSSAGE	7-OCT-1980 21:32	AI: CADRDC; DCBUSY

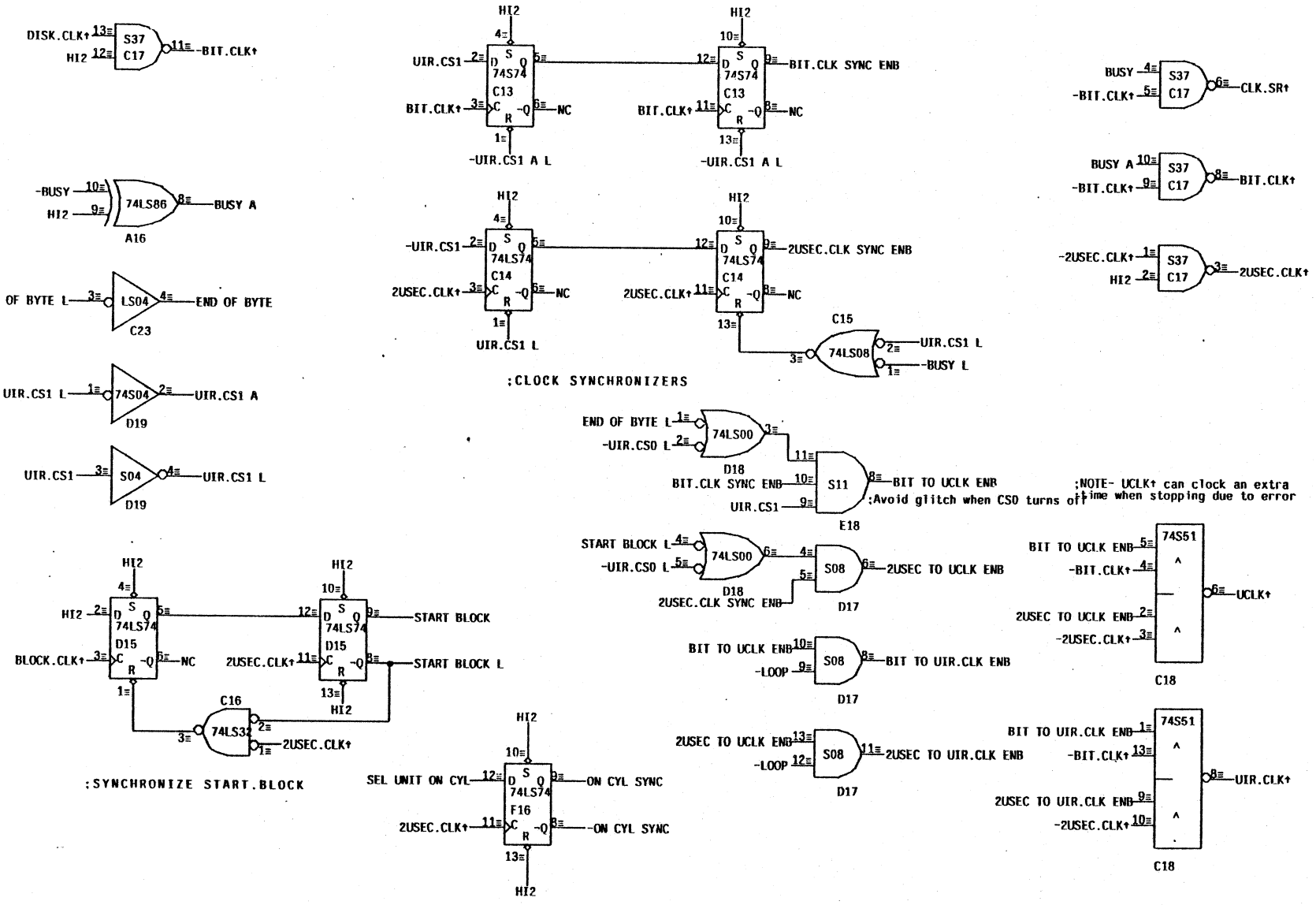

VCC
22UF
15V
E0601
VCC
22UF
15V
E0201
ABOVE DIP
BIT.CLK↑
180
390
4(20) VCC
3(19) GND
E0101
VCC
22UF
15V
E1701
VCC
22UF
15V
B1301
ABOVE DIP
CLK.SR↑
180
390
4(20) VCC
3(19) GND
E2701
VCC
22UF
15V
C2801
VCC
22UF
15V
C2301
ABOVE DIP
-BIT.CLK↑
180
390
4(20) VCC
3(19) GND
D1401



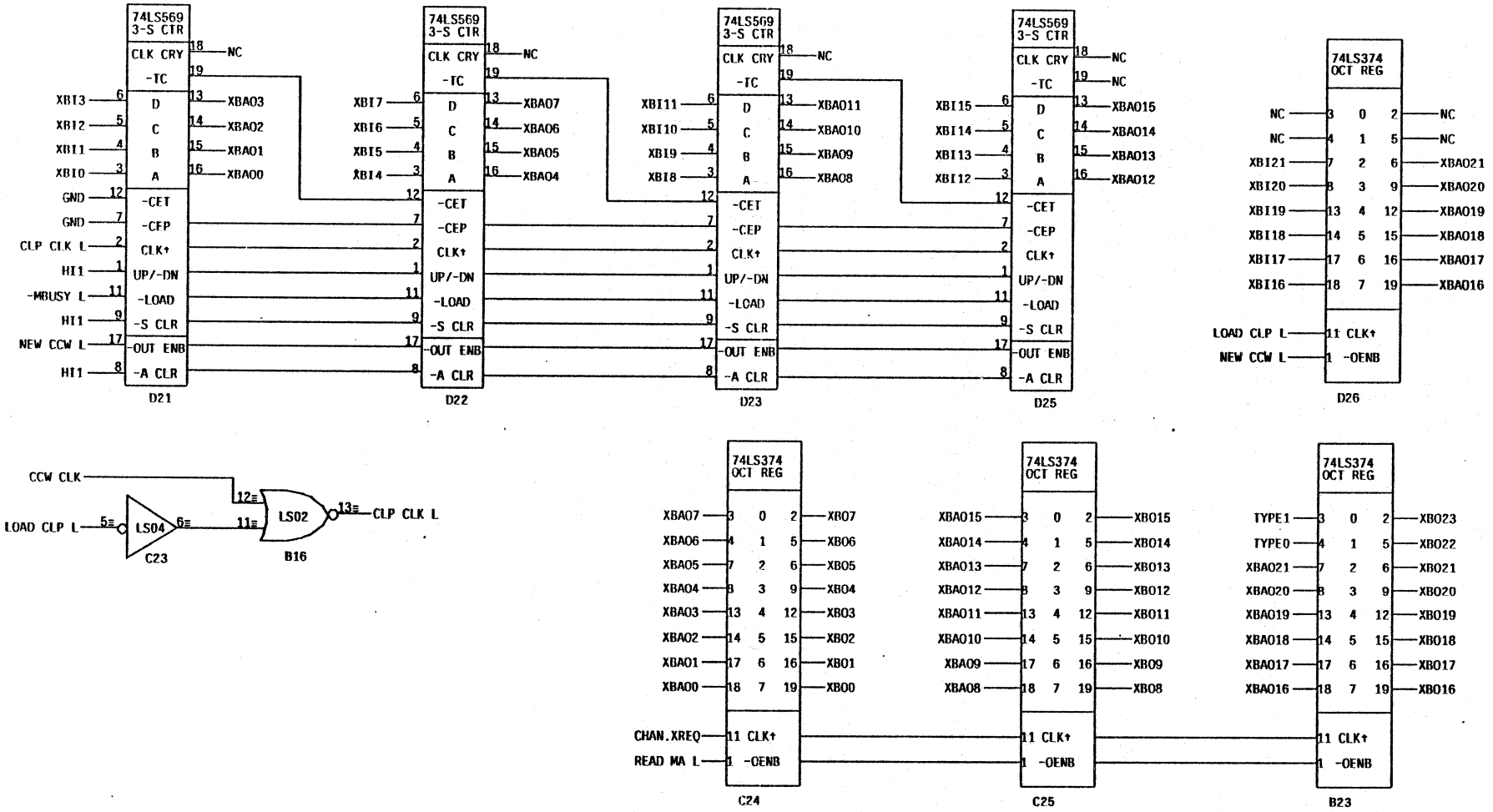
DISK CONTROL	CHANNEL CONTROL WORD	7-OCT-1980 21:32	AI: CADRDC; DCCCW



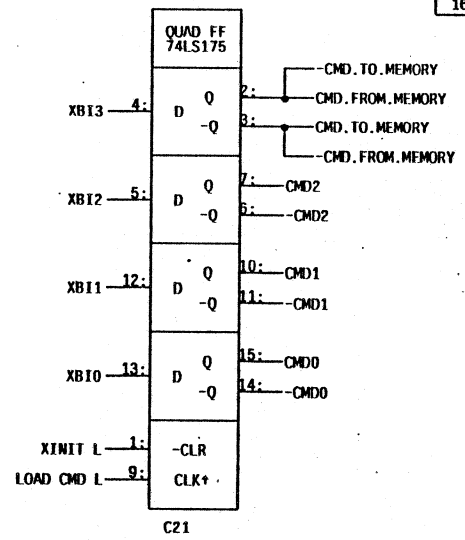
DISK CONTROL	DMA CHANNEL CONTROL	6-FEB-1980 00:48	AI: CADRDC; DCCHAN

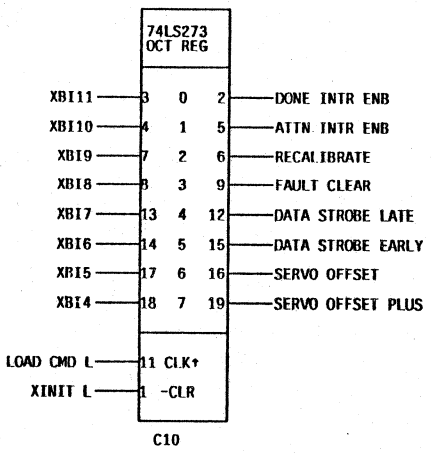


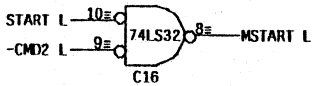
DISK CONTROL	CLOCKS	8-OCT-1978 16:48	AI: CADRDC; DCCLK

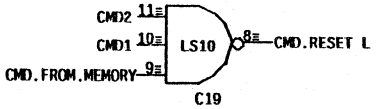

74LS569
3-S CTR
CLK CRY
-TC
D
C
B
A
-CET
-CEP
CLK↑
UP/-DN
-LOAD
-S CLR
-OUT ENB
-A CLR
XBI3
XBI2
XBI1
XBI0
GND
GND
CLP CLK L
HI1
-MBUSY L
HI1
NEW CCW L
HI1
NC
XBA03
XBA02
XBA01
XBA00
D21
XBI7
XBI6
XBI5
XBI4
XBA07
XBA06
XBA05
XBA04
D22
XBI11
XBI10
XBI9
XBI8
XBA011
XBA010
XBA09
XBA08
D23
XBI15
XBI14
XBI13
XBI12
XBA015
XBA014
XBA013
XBA012
D25
74LS374
OCT REG
NC
NC
XBI21
XBI20
XBI19
XBI18
XBI17
XBI16
LOAD CLP L
NEW CCW L
CLK↑
-OENB
XBA021
XBA020
XBA019
XBA018
XBA017
XBA016
D26
CCW CLK
LOAD CLP L
LS04
LS02
CLP CLK L
C23
B16
XBA07
XBA06
XBA05
XBA04
XBA03
XBA02
XBA01
XBA00
CHAN.XREQ
READ MA L
XB07
XB06
XB05
XB04
XB03
XB02
XB01
XB00
C24
XBA015
XBA014
XBA013
XBA012
XBA011
XBA010
XBA09
XBA08
XB015
XB014
XB013
XB012
XB011
XB010
XB09
XB08
C25
TYPE1
TYPE0
XBA021
XBA020
XBA019
XBA018
XBA017
XBA016
XB023
XB022
XB021
XB020
XB019
XB018
XB017
XB016
B23

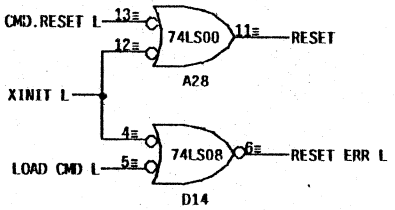
CMD	OPERATION
00	READ
10	READ COMPARE
11	WRITE
02	READ ALL
13	WRITE ALL
04	SEEK
05	AT EASE
1005	RECALIBRATE
405	FAULT CLEAR
06	OFFSET CLEAR
16	STOP, RESET

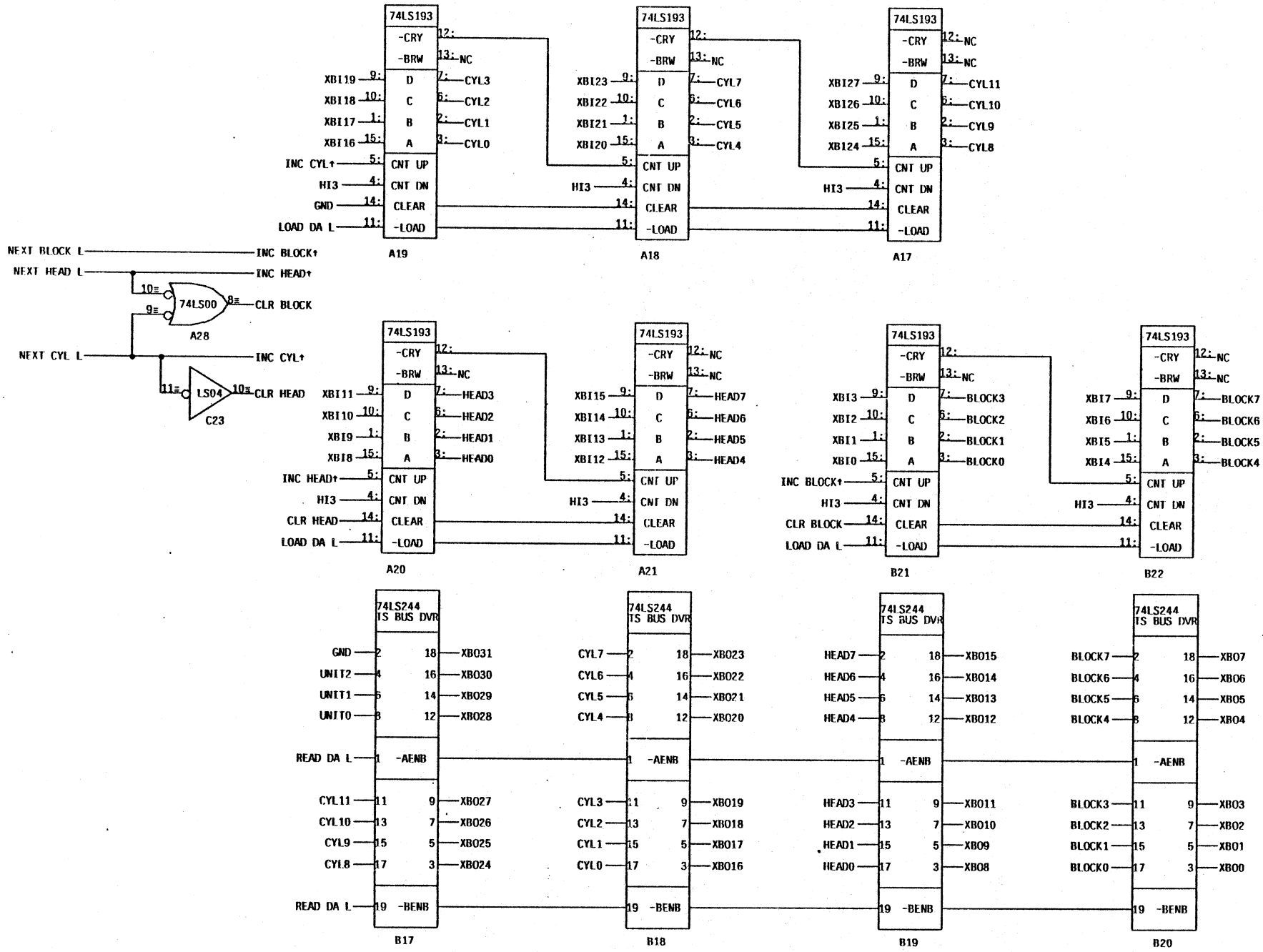


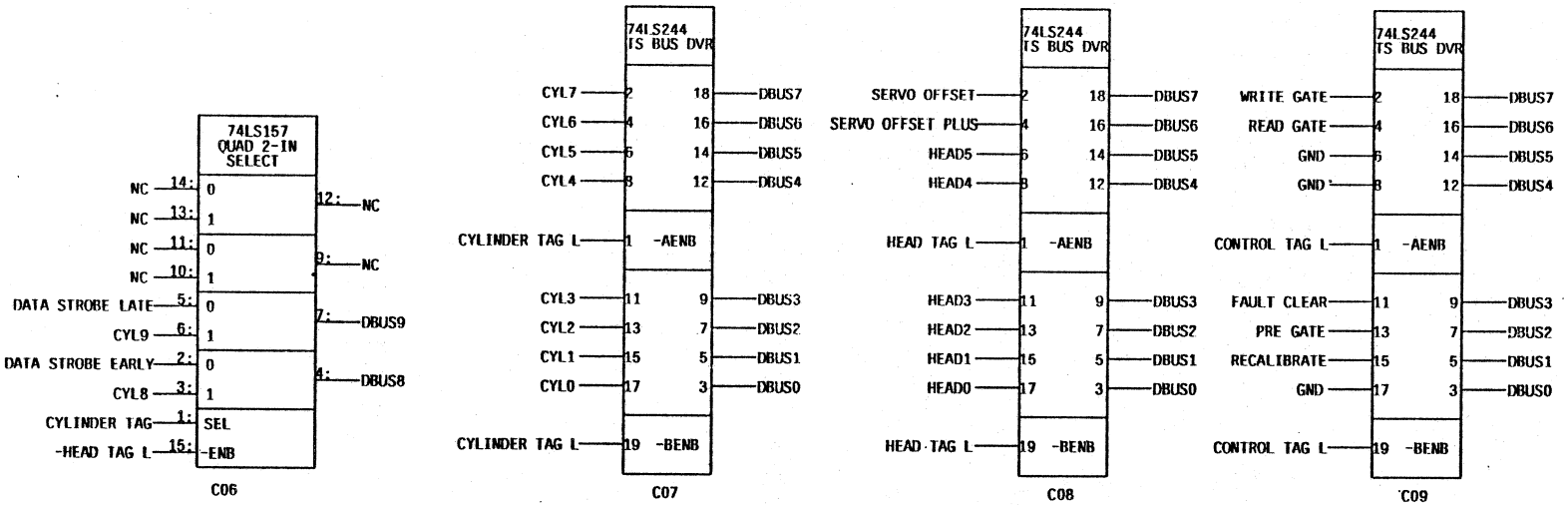


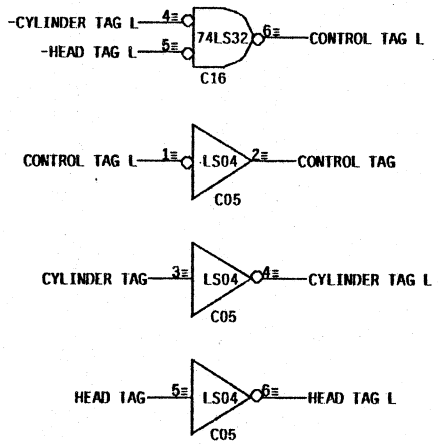




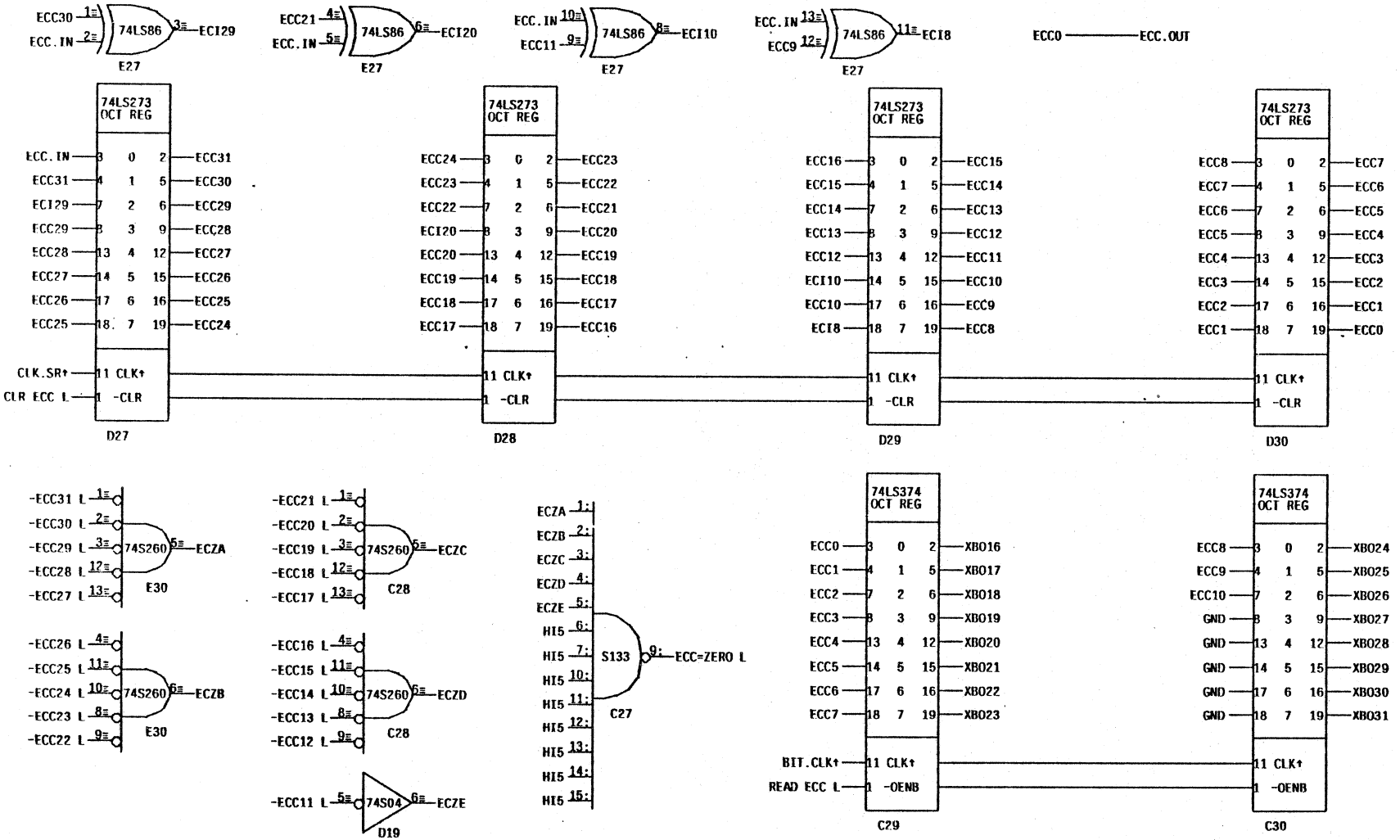



74LS193
-CRY
-BRW
D
C
B
A
CNT UP
CNT DN
CLEAR
-LOAD
A19
A18
A17
A20
A21
B21
B22
74LS00
A28
LS04
C23
NEXT BLOCK L
NEXT HEAD L
NEXT CYL L
INC BLOCK↑
INC HEAD↑
INC CYL↑
CLR BLOCK
CLR HEAD
74LS244
TS BUS DVR
-AENB
-BENB
B17
B18
B19
B20

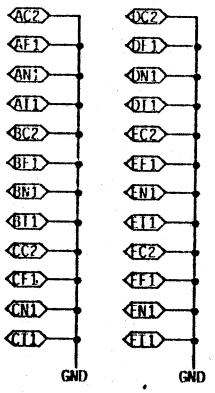




DISK CONTROL	DISK BUS	6-JUN-1979 08:00	AI: CADRDC; DCDBUS


ECC30
ECC.IN
74LS86
ECI29
E27
ECC21
ECC.IN
74LS86
ECI20
E27
ECC.IN
ECC11
74LS86
ECI10
E27
ECC.IN
ECC9
74LS86
ECI8
E27
ECC0
ECC.OUT
74LS273
OCT REG
ECC.IN
ECC31
ECI29
ECC29
ECC28
ECC27
ECC26
ECC25
CLK.SR↑
CLR ECC L
ECC31
ECC30
ECC29
ECC28
ECC27
ECC26
ECC25
ECC24
D27
ECC24
ECC23
ECC22
ECI20
ECC20
ECC19
ECC18
ECC17
ECC23
ECC22
ECC21
ECC20
ECC19
ECC18
ECC17
ECC16
D28
ECC16
ECC15
ECC14
ECC13
ECC12
ECI10
ECC10
ECI8
ECC15
ECC14
ECC13
ECC12
ECC11
ECC10
ECC9
ECC8
D29
ECC8
ECC7
ECC6
ECC5
ECC4
ECC3
ECC2
ECC1
ECC7
ECC6
ECC5
ECC4
ECC3
ECC2
ECC1
ECC0
D30
-ECC31 L
-ECC30 L
-ECC29 L
-ECC28 L
-ECC27 L
74S260
ECZA
E30
-ECC26 L
-ECC25 L
-ECC24 L
-ECC23 L
-ECC22 L
74S260
ECZB
E30
-ECC21 L
-ECC20 L
-ECC19 L
-ECC18 L
-ECC17 L
74S260
ECZC
C28
-ECC16 L
-ECC15 L
-ECC14 L
-ECC13 L
-ECC12 L
74S260
ECZD
C28
-ECC11 L
74S04
ECZE
D19
ECZA
ECZB
ECZC
ECZD
ECZE
HI5
HI5
HI5
HI5
HI5
HI5
HI5
HI5
S133
ECC=ZERO L
C27
74LS374
OCT REG
ECC0
ECC1
ECC2
ECC3
ECC4
ECC5
ECC6
ECC7
BIT.CLK↑
READ ECC L
XBO16
XBO17
XBO18
XBO19
XBO20
XBO21
XBO22
XBO23
C29
74LS374
OCT REG
ECC8
ECC9
ECC10
GND
GND
GND
GND
GND
XBO24
XBO25
XBO26
XBO27
XBO28
XBO29
XBO30
XBO31
C30

GROUNDS



JUMPERS FOR 1-BOARD VERSION:

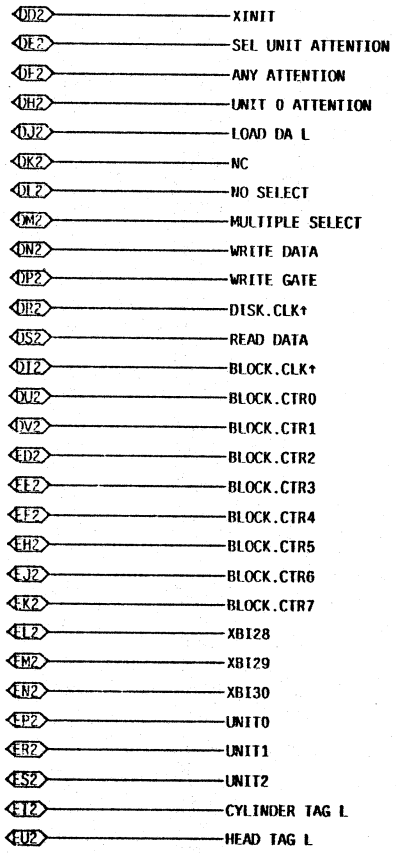
DE2 - DF2 - DH2
DN1 - DM2
ET1 - EP2 - ER2 - ES2

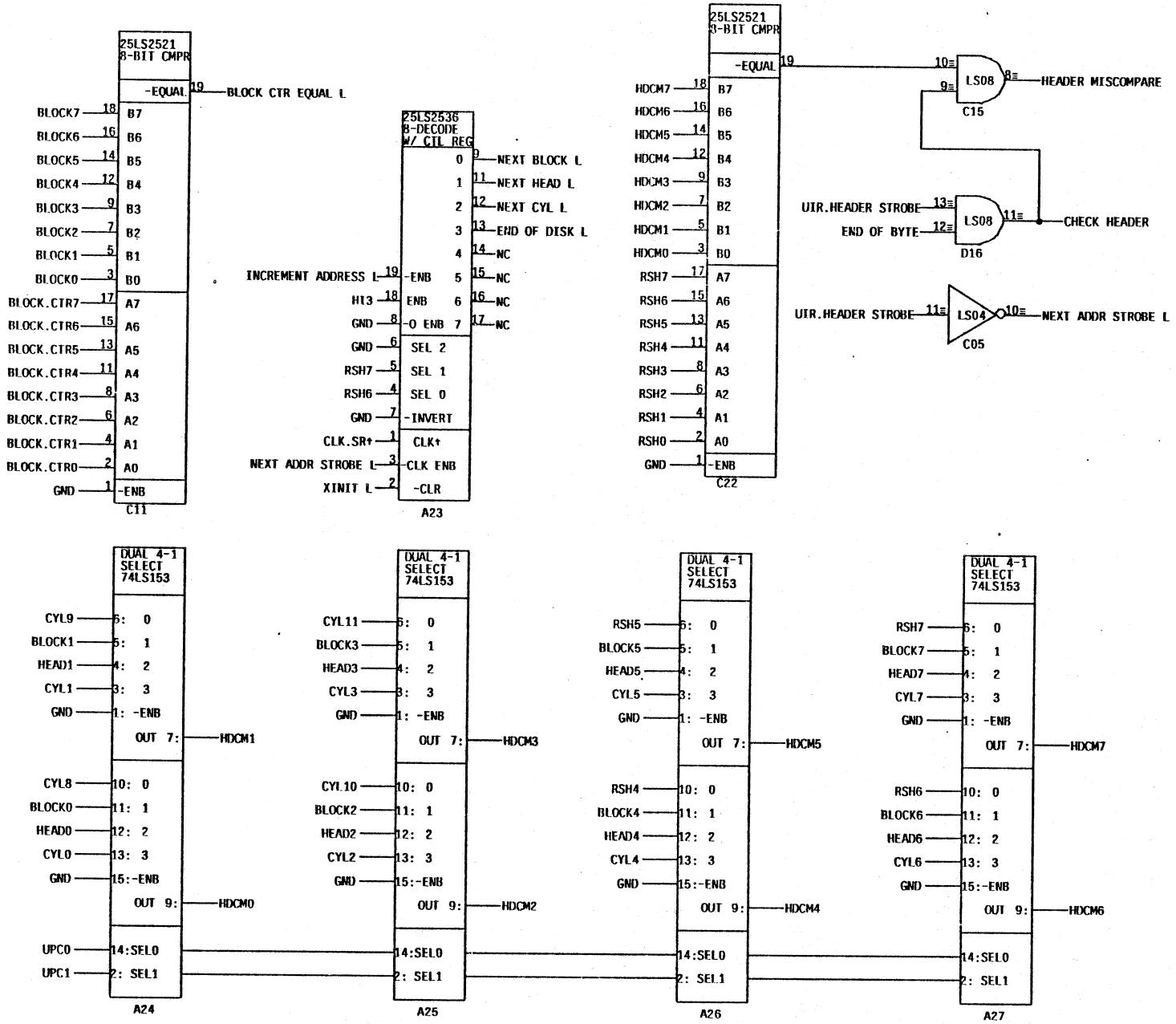
OMITTED DIPS IN 2-BOARD VERSION:

A7 A8 A9
A10 B7 B8

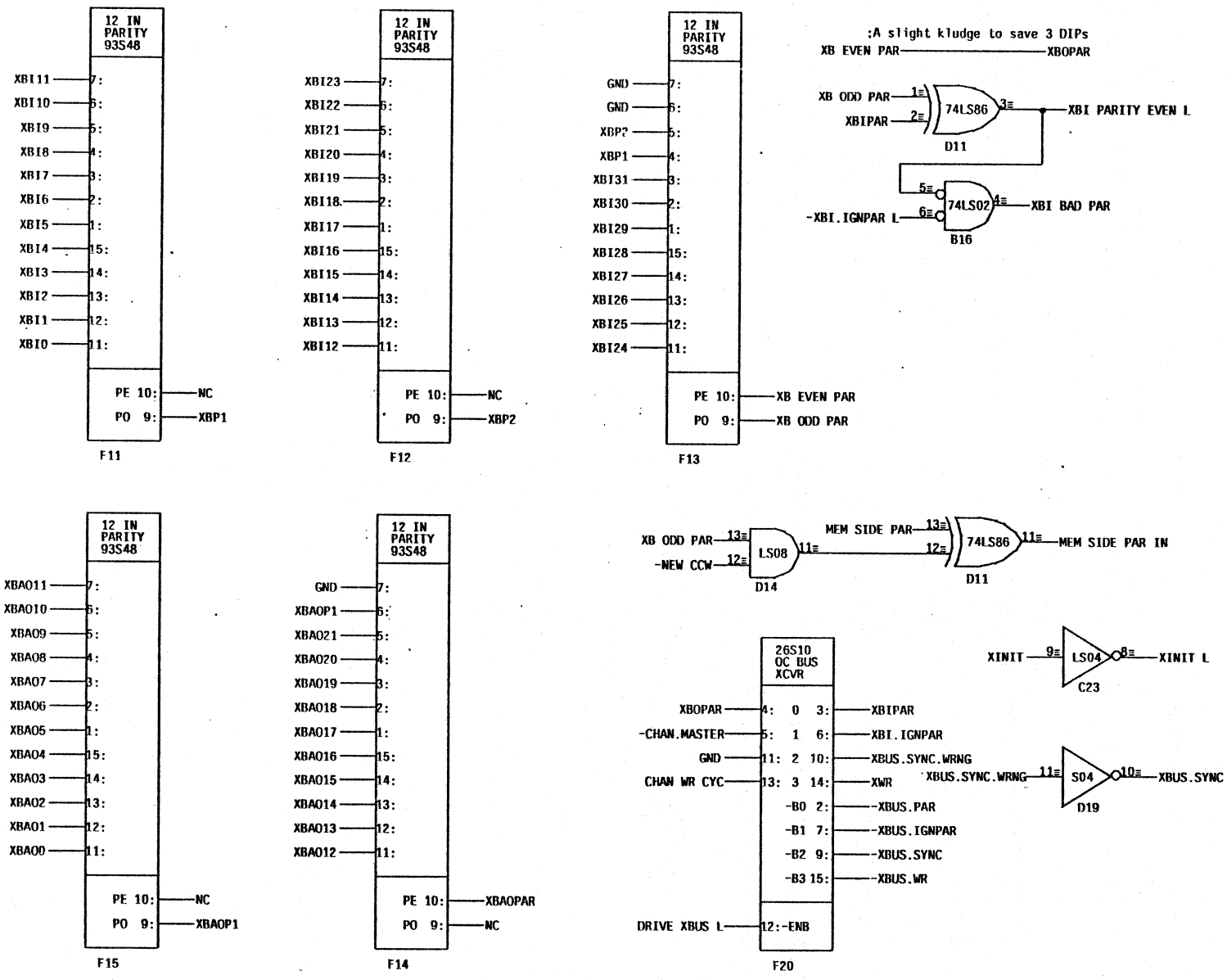
BE CERTAIN TO EDIT THE
RAY FILE FOR UNDEDICATED
GROUND PINS ON J1, J3

TO DISK MUX BOARD

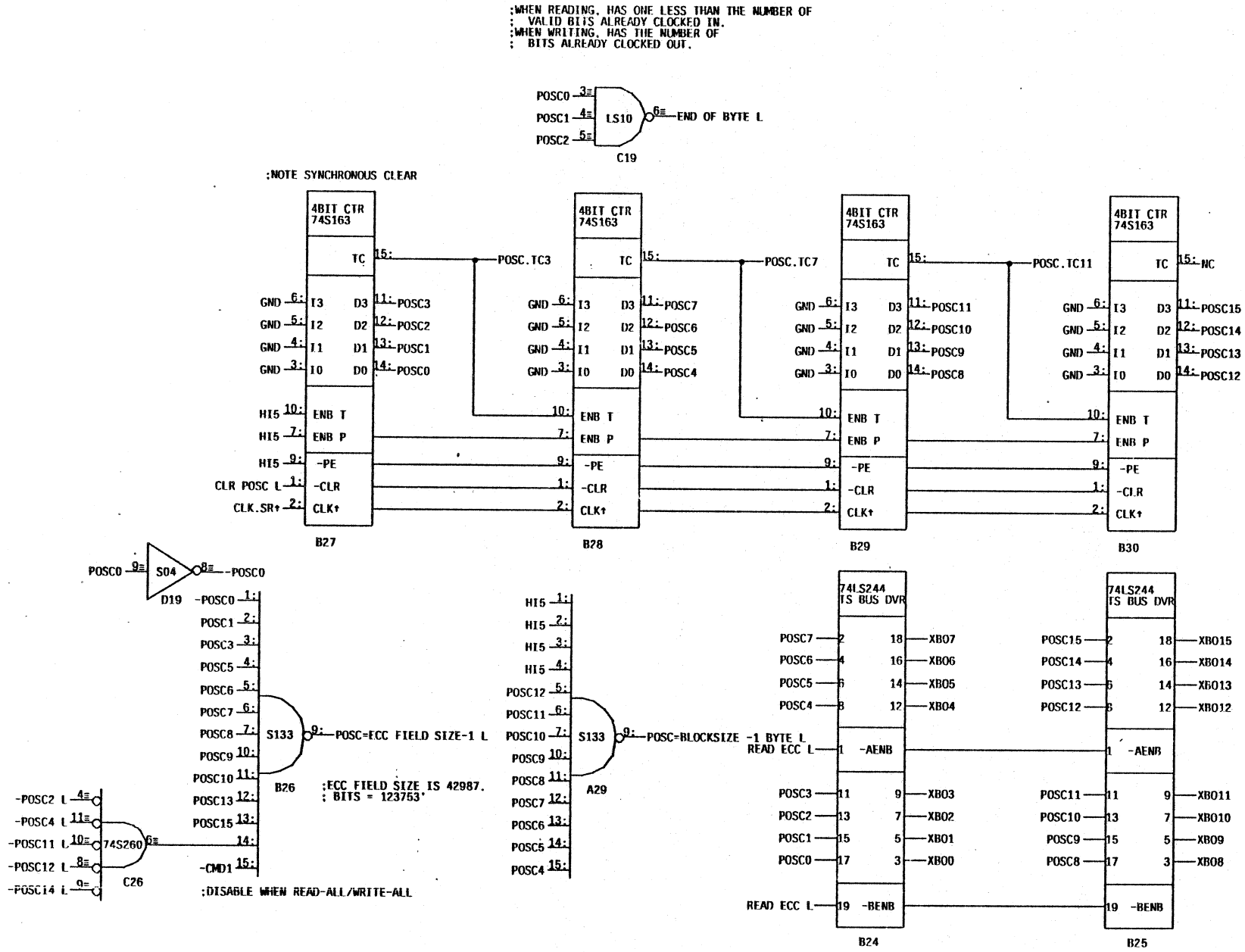


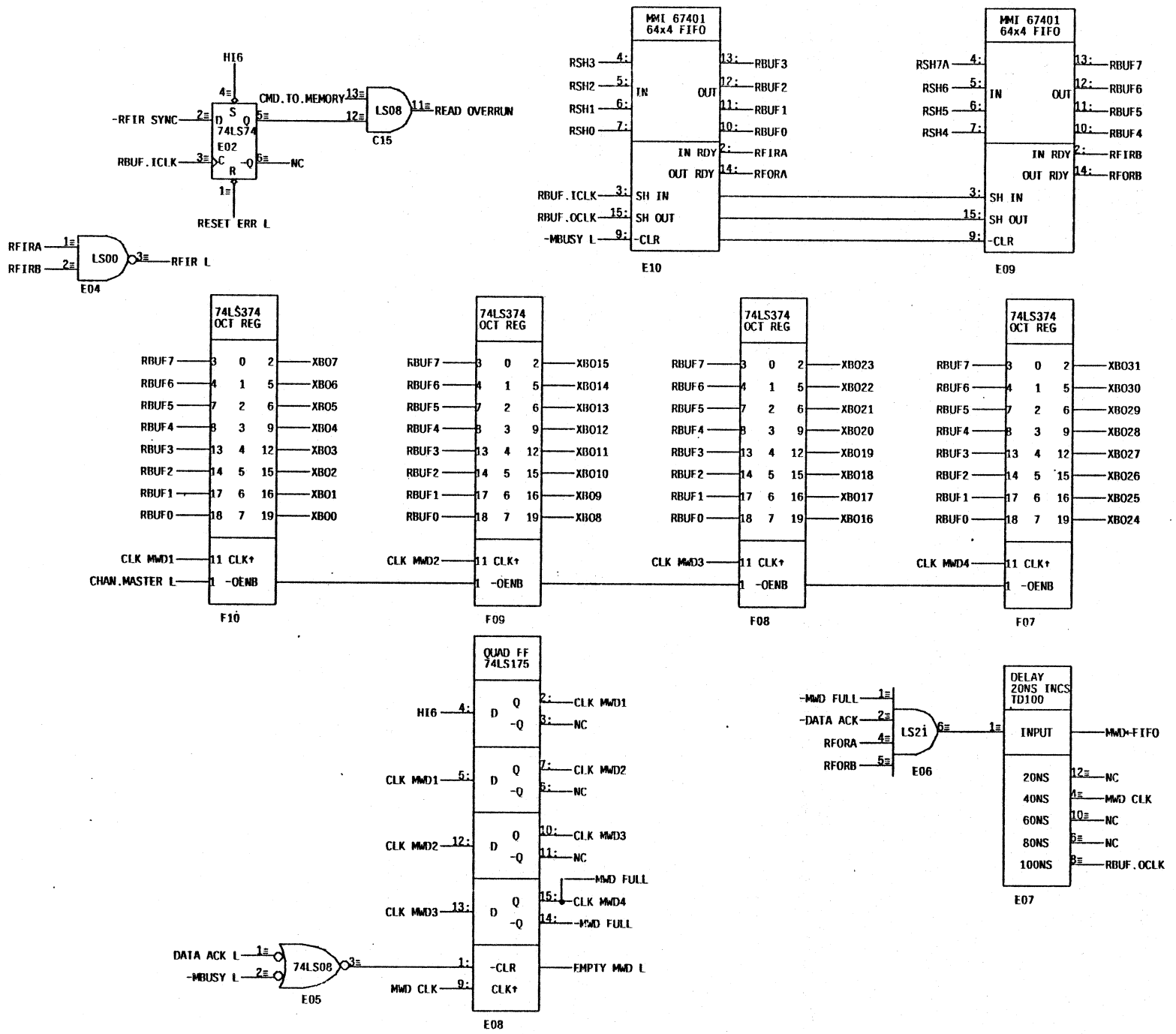


DISK CONTROL	HEADER COMPARE	10-JUN-1979 20:34	AI: CADRDC; DCHDCM

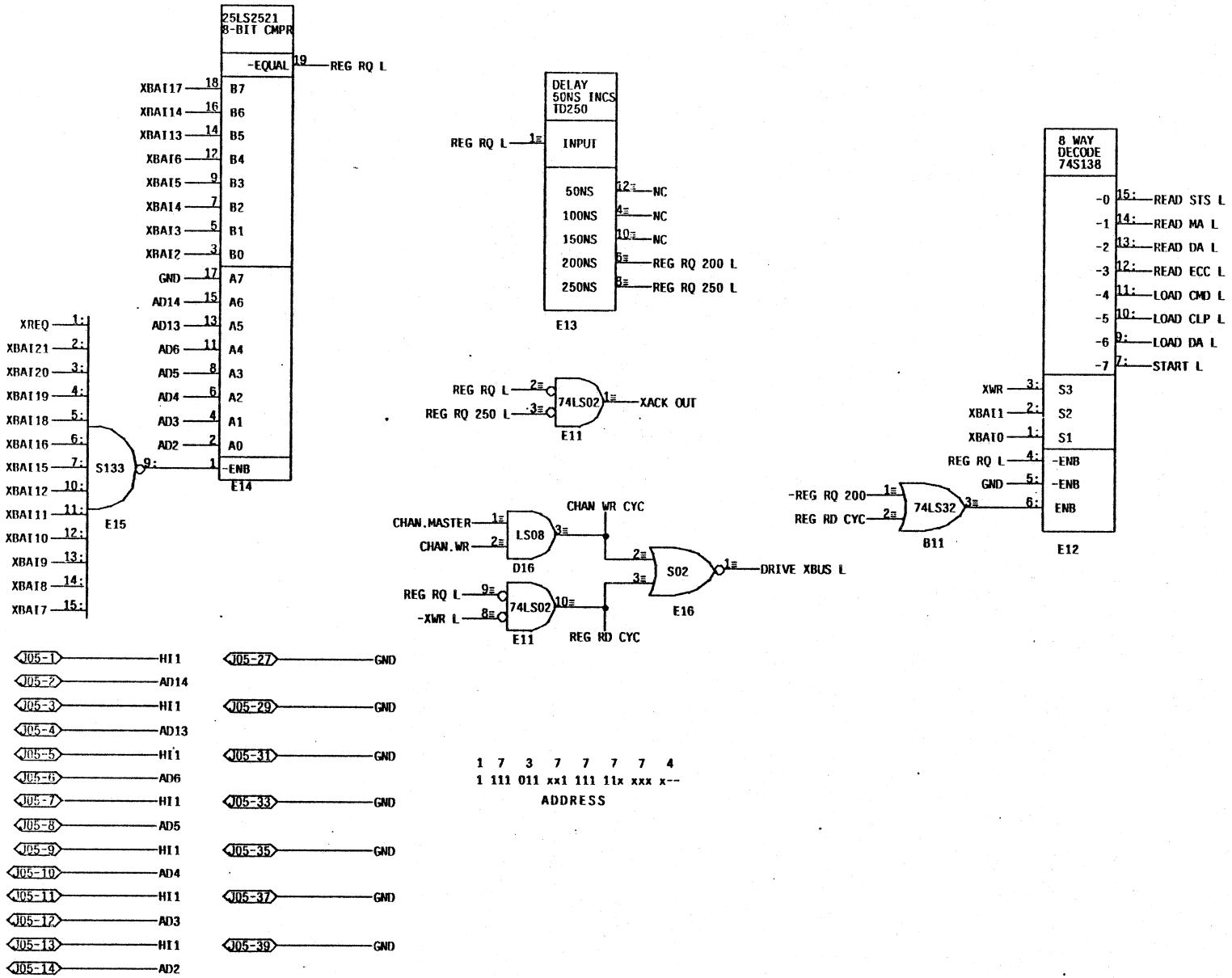


DISK CONTROL	PARITY	6-FEB-1980 00:54	AI: CADRDC; DCPAR

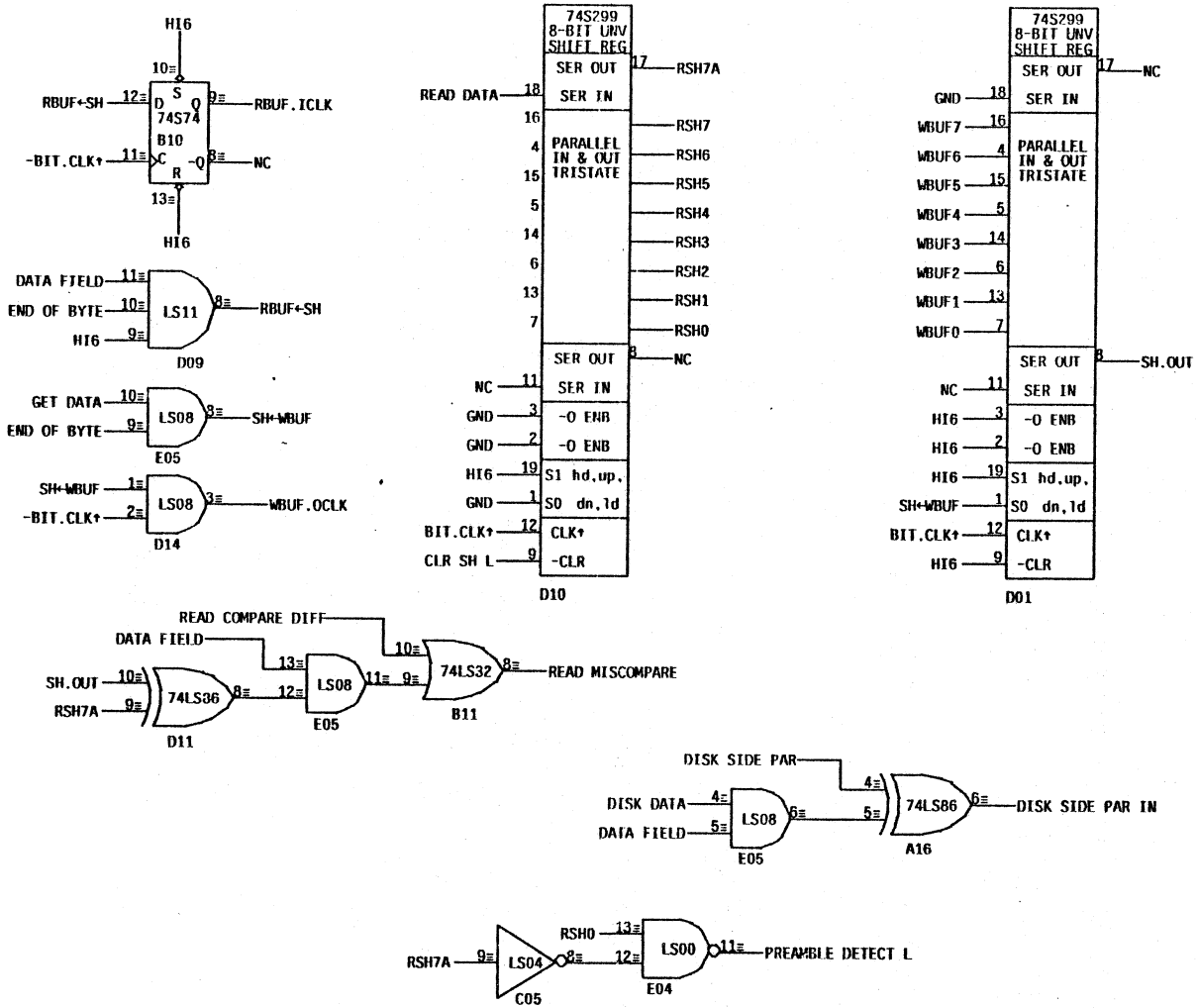

;WHEN READING, HAS ONE LESS THAN THE NUMBER OF
; VALID BITS ALREADY CLOCKED IN.
;WHEN WRITING, HAS THE NUMBER OF
; BITS ALREADY CLOCKED OUT.
POSC0
POSC1
POSC2
LS10
END OF BYTE L
C19
;NOTE SYNCHRONOUS CLEAR
4BIT CTR 74S163
TC
POSC.TC3
POSC.TC7
POSC.TC11
NC
GND
I3
I2
I1
I0
D3
D2
D1
D0
POSC3
POSC2
POSC1
POSC0
POSC7
POSC6
POSC5
POSC4
POSC11
POSC10
POSC9
POSC8
POSC15
POSC14
POSC13
POSC12
HI5
ENB T
ENB P
-PE
-CLR
CLK↑
CLR POSC L
CLK.SR↑
B27
B28
B29
B30
POSC0
S04
-POSC0
D19
S133
POSC=ECC FIELD SIZE-1 L
B26
;ECC FIELD SIZE IS 42987.
; BITS = 123753'
-POSC2 L
-POSC4 L
-POSC11 L
-POSC12 L
-POSC14 L
74S260
C26
-CMD1
;DISABLE WHEN READ-ALL/WRITE-ALL
POSC=BLOCKSIZE -1 BYTE L
A29
74LS244 TS BUS DVR
-AENB
-BENB
READ ECC L
XB07
XB06
XB05
XB04
XB03
XB02
XB01
XB00
XB015
XB014
XB013
XB012
XB011
XB010
XB09
XB08
B24
B25



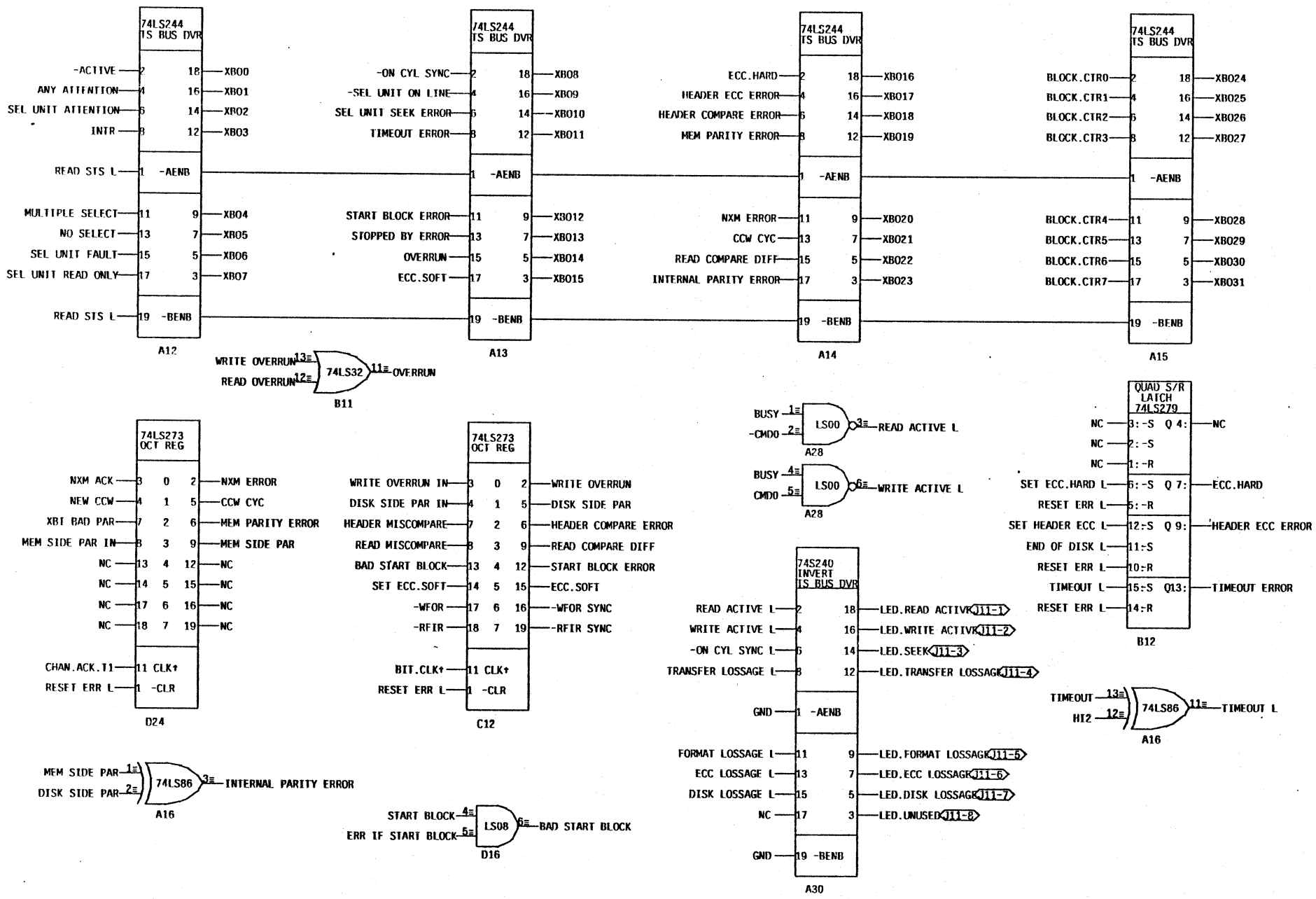
DISK CONTROL	READ BUFFER	30-SEP-1978 04:07	AI: CADRDC; DCRBUF



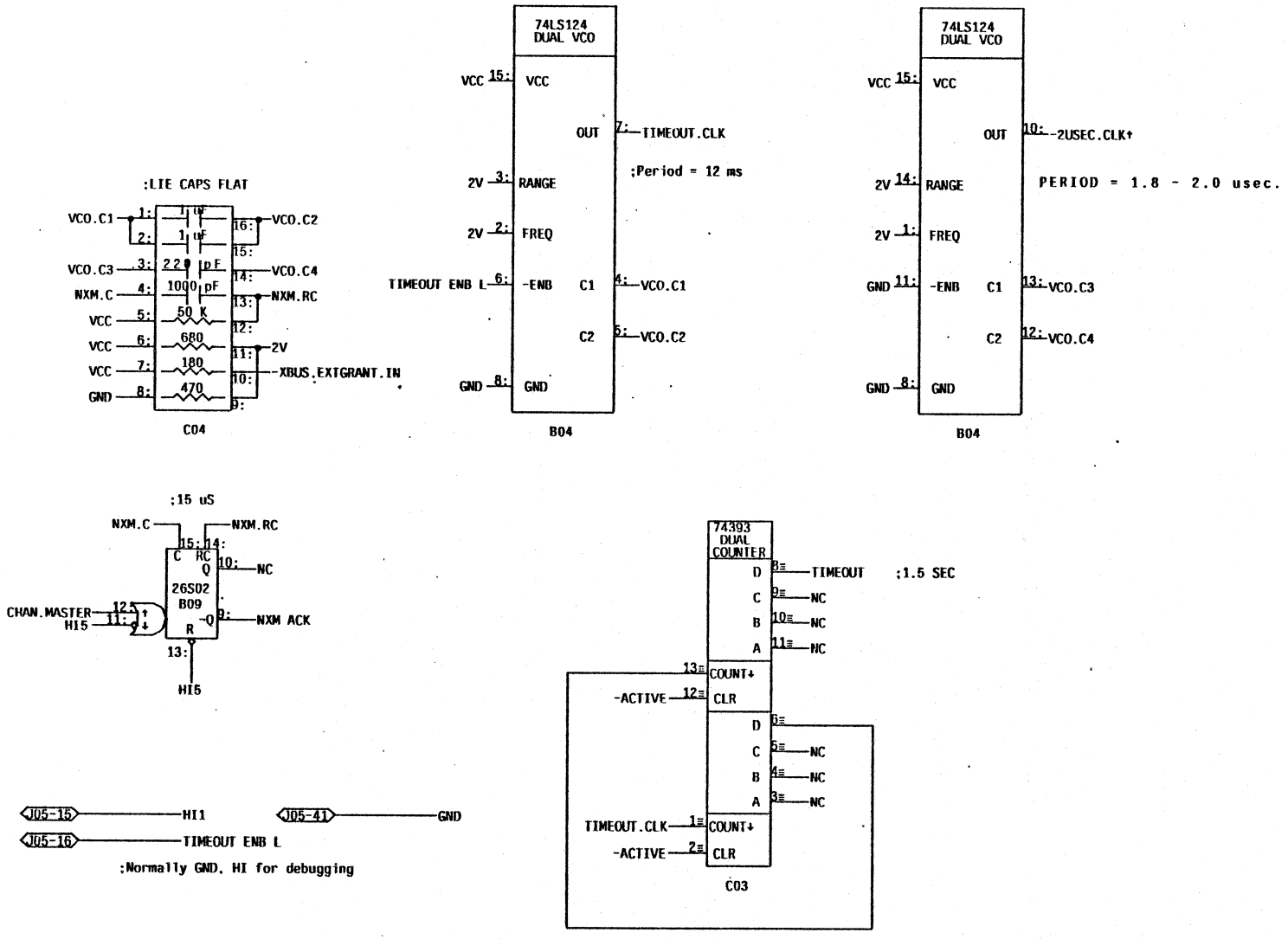
DISK CONTROL	REGISTER ADDRESSING	8-OCT-1978 16:56	AI: CADRDC; DCREG

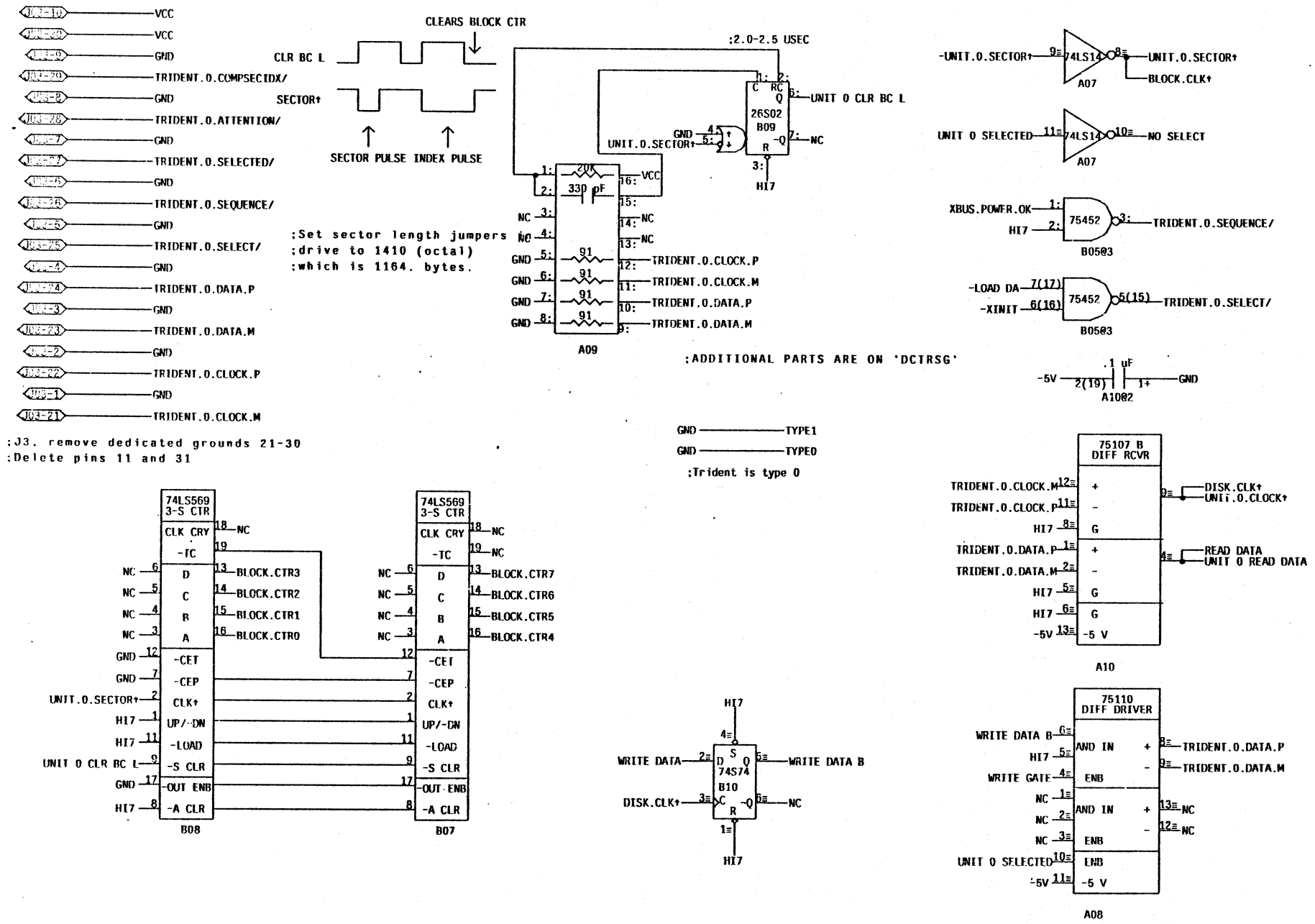


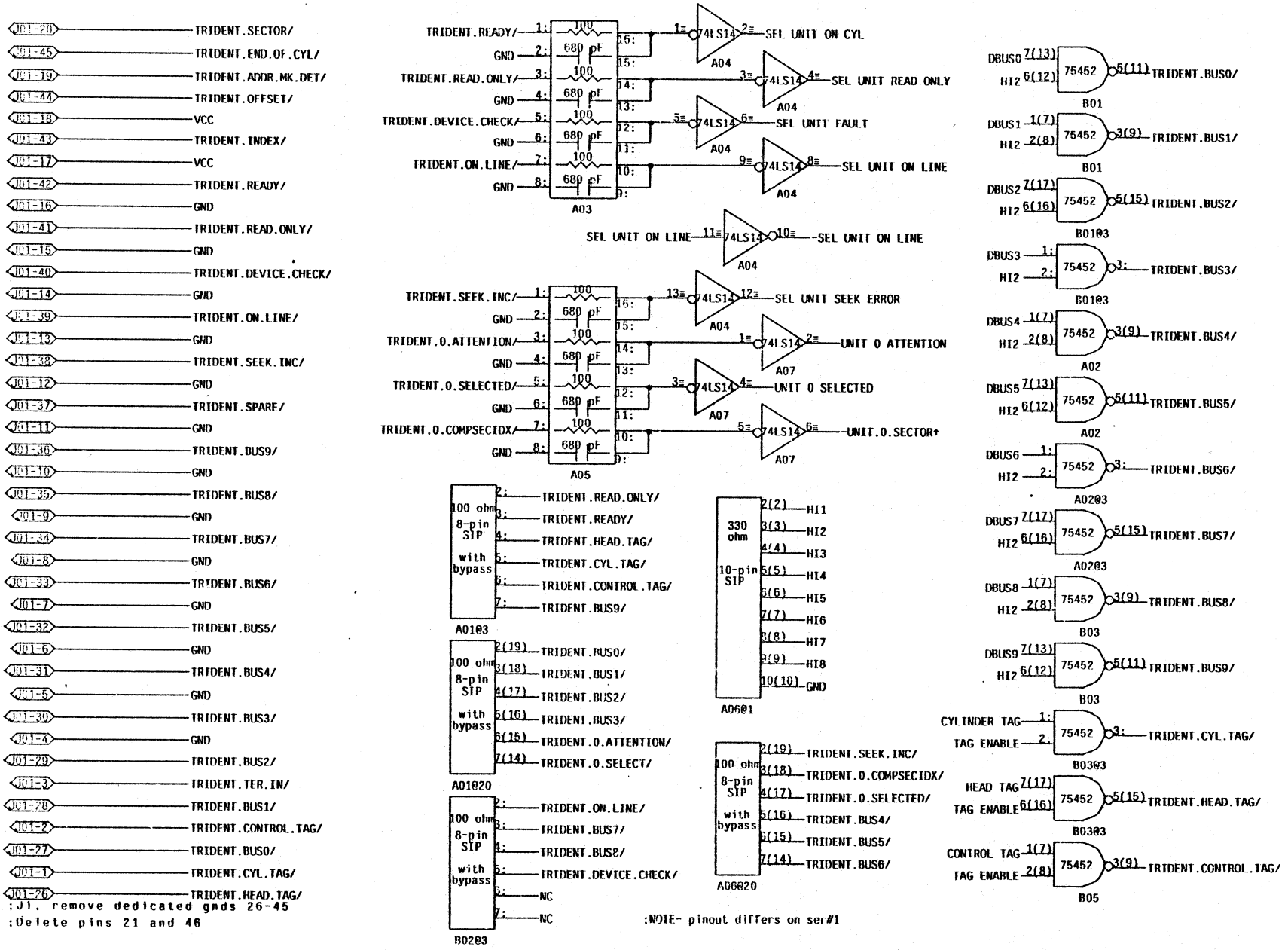
DISK CONTROL	SHIFT REGISTER	10-JUN-1979 20:37	AI: CADRDC; DCSH



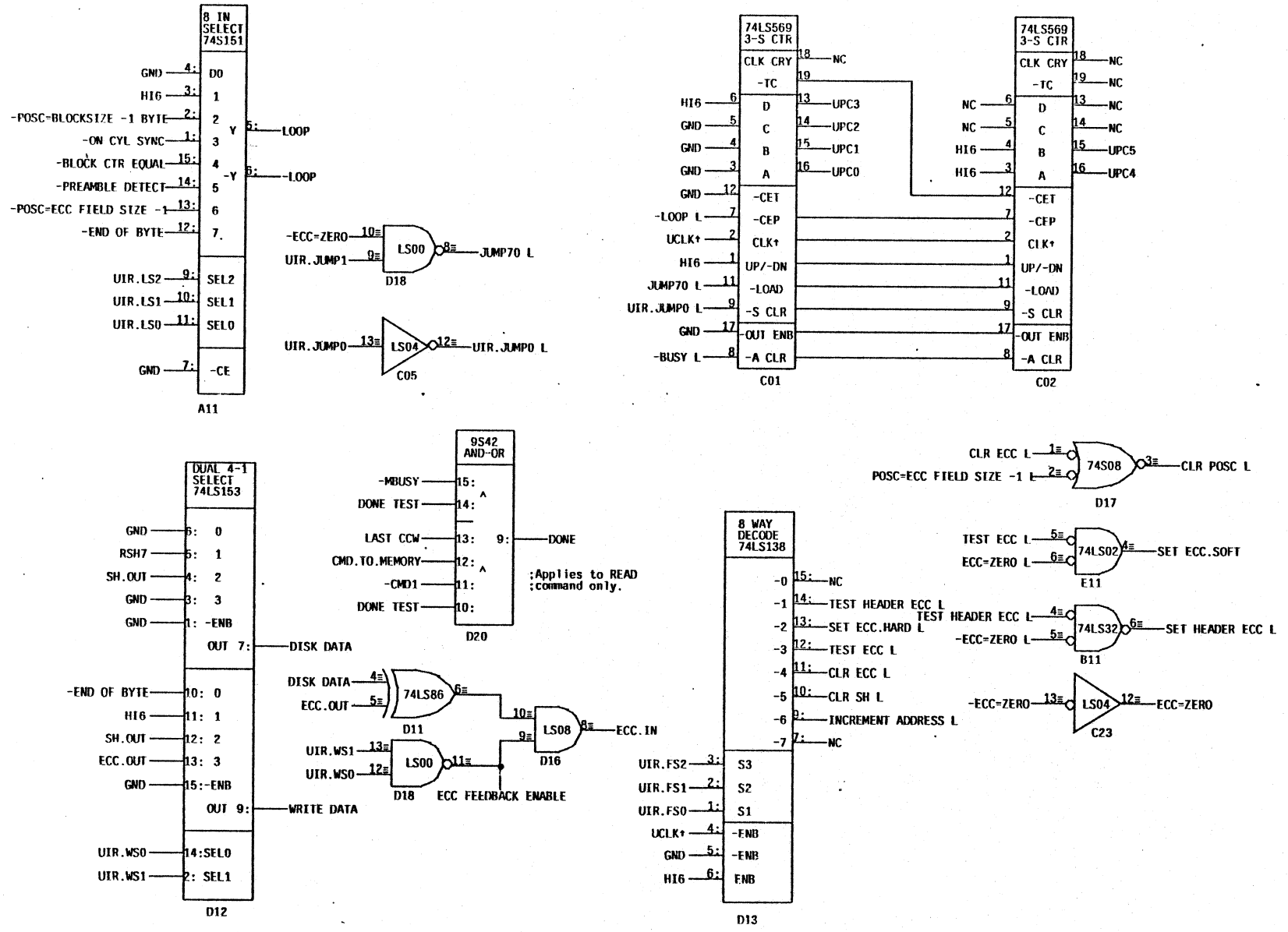
DISK CONTROL	STATUS	5-FEB-1980 01:50	AI: CADRDC; DCSTS

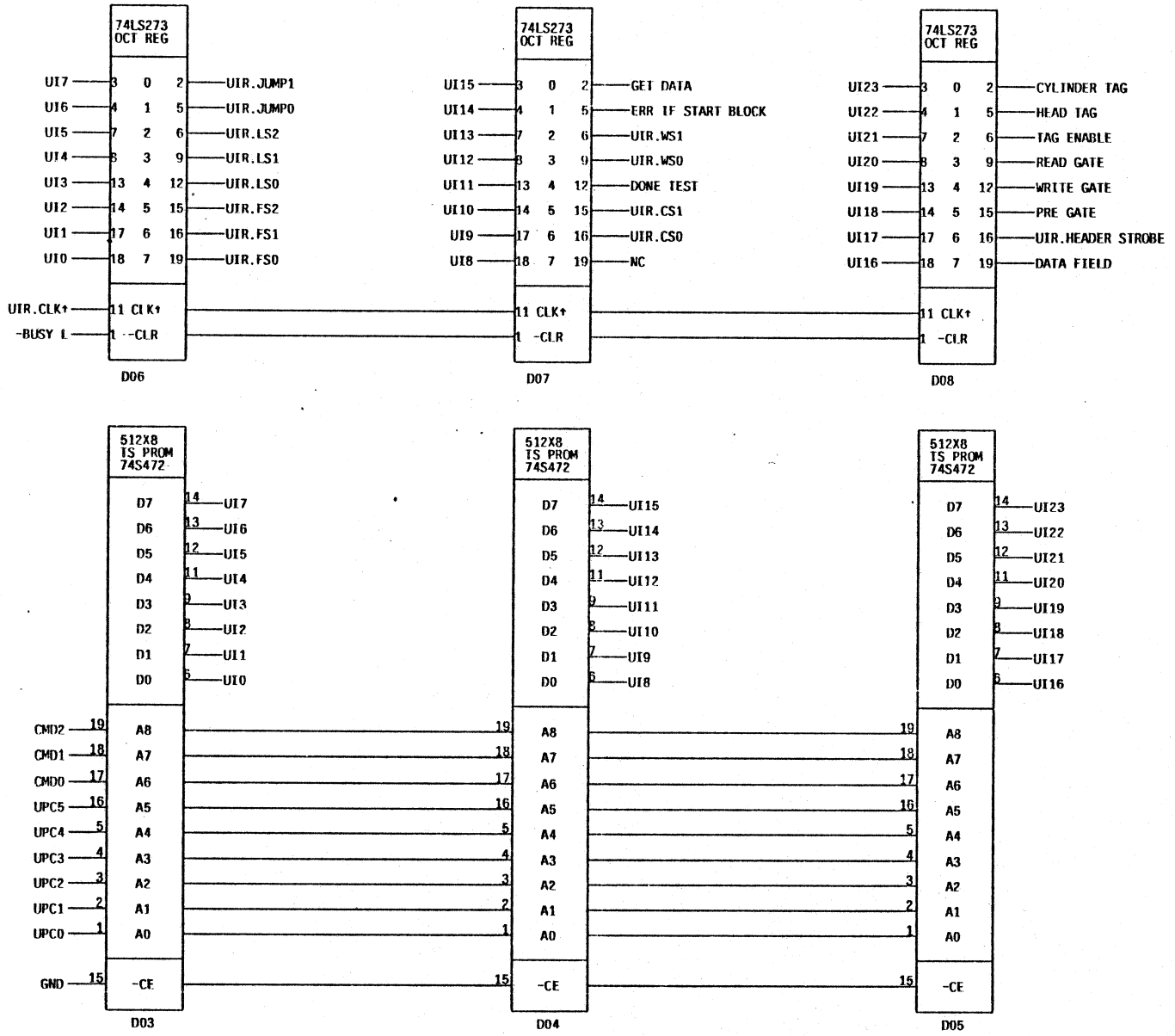

:LIE CAPS FLAT
VCO.C1
1 uF
VCO.C2
1 uF
VCO.C3
220 pF
VCO.C4
NXM.C
1000 pF
NXM.RC
VCC
50 K
VCC
680
2V
VCC
180
-XBUS.EXTGRANT.IN
GND
470
C04
74LS124
DUAL VCO
VCC
OUT
TIMEOUT.CLK
;Period = 12 ms
2V
RANGE
2V
FREQ
TIMEOUT ENB L
-ENB
C1
VCO.C1
C2
VCO.C2
GND
GND
B04
74LS124
DUAL VCO
VCC
OUT
-2USEC.CLK↑
2V
RANGE
PERIOD = 1.8 - 2.0 usec.
2V
FREQ
GND
-ENB
C1
VCO.C3
C2
VCO.C4
GND
GND
B04
;15 uS
NXM.C
NXM.RC
C
RC
Q
NC
26S02
B09
CHAN.MASTER
HI5
-Q
NXM ACK
R
HI5
74393
DUAL
COUNTER
D
TIMEOUT
;1.5 SEC
C
NC
B
NC
A
NC
COUNT↓
-ACTIVE
CLR
D
C
NC
B
NC
A
NC
TIMEOUT.CLK
COUNT↓
-ACTIVE
CLR
C03
J05-15
HI1
J05-41
GND
J05-16
TIMEOUT ENB L
;Normally GND, HI for debugging

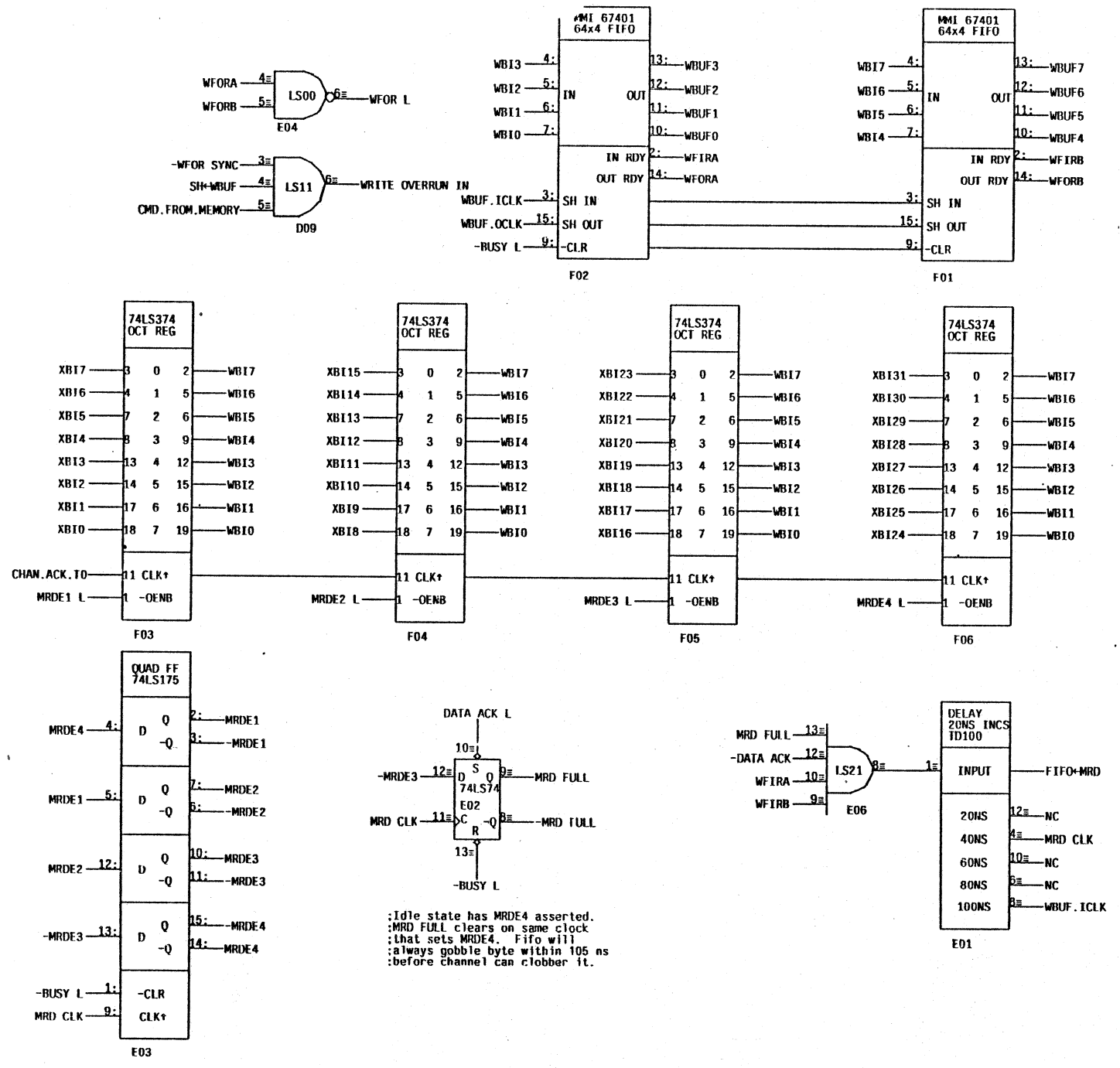

J03-10 VCC
J03-20 VCC
J03-9 GND
J03-19 TRIDENT.0.COMPSECIDX/
J03-8 GND
J03-18 TRIDENT.0.ATTENTION/
J03-7 GND
J03-17 TRIDENT.0.SELECTED/
J03-6 GND
J03-16 TRIDENT.0.SEQUENCE/
J03-5 GND
J03-15 TRIDENT.0.SELECT/
J03-4 GND
J03-14 TRIDENT.0.DATA.P
J03-3 GND
J03-13 TRIDENT.0.DATA.M
J03-2 GND
J03-12 TRIDENT.0.CLOCK.P
J03-1 GND
J03-11 TRIDENT.0.CLOCK.M
:J3. remove dedicated grounds 21-30
:Delete pins 11 and 31
CLEARS BLOCK CTR
CLR BC L
SECTOR↑
SECTOR PULSE INDEX PULSE
:Set sector length jumpers
:drive to 1410 (octal)
:which is 1164. bytes.
:2.0-2.5 USEC
20K
330 pF
91
91
91
91
VCC
NC
NC
NC
TRIDENT.0.CLOCK.P
TRIDENT.0.CLOCK.M
TRIDENT.0.DATA.P
TRIDENT.0.DATA.M
A09
26S02
B09
UNIT.0.SECTOR↑
UNIT 0 CLR BC L
NC
HI7
-UNIT.0.SECTOR↑
74LS14
A07
UNIT.0.SECTOR↑
BLOCK.CLK↑
UNIT 0 SELECTED
NO SELECT
XBUS.POWER.OK
HI7
75452
B05@3
TRIDENT.0.SEQUENCE/
-LOAD DA
-XINIT
TRIDENT.0.SELECT/
:ADDITIONAL PARTS ARE ON 'DCTRSG'
.1 uF
-5V
GND
A10@2
GND TYPE1
GND TYPE0
:Trident is type 0
74LS569
3-S CTR
B08
B07
CLK CRY
-TC
D
C
B
A
-CET
-CEP
CLK↑
UP/-DN
-LOAD
-S CLR
-OUT ENB
-A CLR
BLOCK.CTR3
BLOCK.CTR2
BLOCK.CTR1
BLOCK.CTR0
BLOCK.CTR7
BLOCK.CTR6
BLOCK.CTR5
BLOCK.CTR4
UNIT.0.SECTOR↑
HI7
UNIT 0 CLR BC L
GND
75107 B
DIFF RCVR
TRIDENT.0.CLOCK.M
TRIDENT.0.CLOCK.P
TRIDENT.0.DATA.P
TRIDENT.0.DATA.M
-5 V
DISK.CLK↑
UNIT.0.CLOCK↑
READ DATA
UNIT 0 READ DATA
A10
WRITE DATA
DISK.CLK↑
74S74
B10
WRITE DATA B
75110
DIFF DRIVER
WRITE DATA B
WRITE GATE
AND IN
ENB
UNIT 0 SELECTED
A08



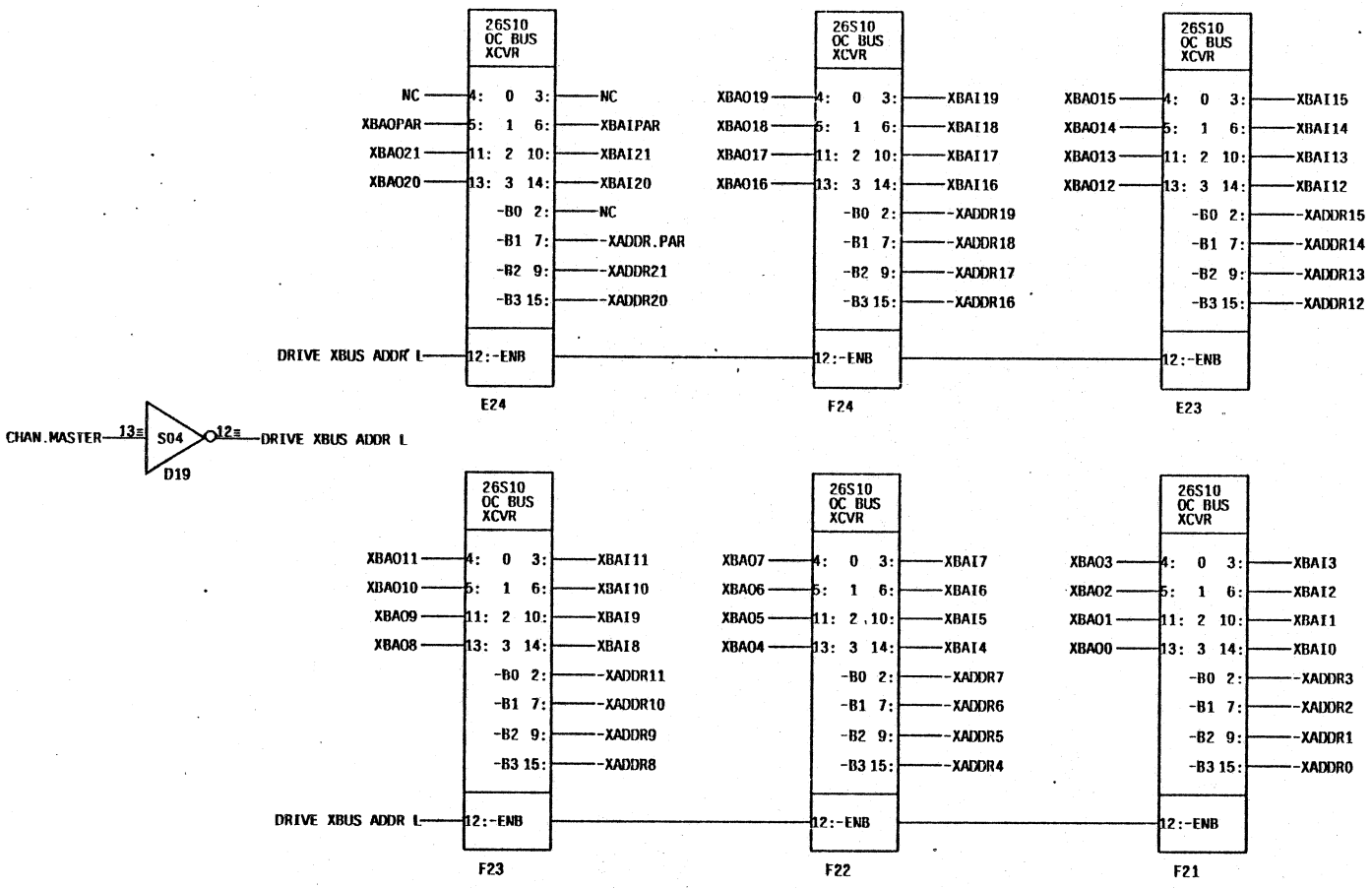
DISK CONTROL	TRIDENT SIGNAL CABLE	30-JUL-1979 02:11	AI: CADRDC; DCTRSG


8 IN SELECT 74S151
GND
HI6
-POSC=BLOCKSIZE -1 BYTE
-ON CYL SYNC
-BLOCK CTR EQUAL
-PREAMBLE DETECT
-POSC=ECC FIELD SIZE -1
-END OF BYTE
UIR.LS2
UIR.LS1
UIR.LS0
GND
D0
1
2
3
4
5
6
7.
Y
-Y
SEL2
SEL1
SEL0
-CE
LOOP
-LOOP
A11
-ECC=ZERO
UIR.JUMP1
LS00
JUMP70 L
D18
UIR.JUMP0
LS04
UIR.JUMP0 L
C05
74LS569 3-S CTR
CLK CRY
-TC
HI6
GND
GND
GND
GND
-LOOP L
UCLK↑
HI6
JUMP70 L
UIR.JUMP0 L
GND
-BUSY L
D
C
B
A
-CET
-CEP
CLK↑
UP/-DN
-LOAD
-S CLR
-OUT ENB
-A CLR
NC
UPC3
UPC2
UPC1
UPC0
C01
74LS569 3-S CTR
NC
NC
HI6
HI6
UPC5
UPC4
C02
DUAL 4-1 SELECT 74LS153
GND
RSH7
SH.OUT
GND
GND
-END OF BYTE
HI6
SH.OUT
ECC.OUT
GND
UIR.WS0
UIR.WS1
0
1
2
3
-ENB
OUT 7:
DISK DATA
-ENB
OUT 9:
WRITE DATA
SEL0
SEL1
D12
9S42 AND-OR
-MBUSY
DONE TEST
LAST CCW
CMD.TO.MEMORY
-CMD1
DONE TEST
DONE
;Applies to READ
;command only.
D20
DISK DATA
ECC.OUT
74LS86
D11
LS08
ECC.IN
D16
UIR.WS1
UIR.WS0
LS00
D18
ECC FEEDBACK ENABLE
8 WAY DECODE 74LS138
-0
-1
-2
-3
-4
-5
-6
-7
NC
TEST HEADER ECC L
SET ECC.HARD L
TEST ECC L
CLR ECC L
CLR SH L
INCREMENT ADDRESS L
NC
UIR.FS2
UIR.FS1
UIR.FS0
UCLK↑
GND
HI6
S3
S2
S1
-ENB
-ENB
ENB
D13
CLR ECC L
POSC=ECC FIELD SIZE -1 L
74S08
CLR POSC L
D17
TEST ECC L
ECC=ZERO L
74LS02
SET ECC.SOFT
E11
TEST HEADER ECC L
-ECC=ZERO L
74LS32
SET HEADER ECC L
B11
-ECC=ZERO
LS04
ECC=ZERO
C23


74LS273
OCT REG
UI7
UI6
UI5
UI4
UI3
UI2
UI1
UI0
UIR.JUMP1
UIR.JUMP0
UIR.LS2
UIR.LS1
UIR.LS0
UIR.FS2
UIR.FS1
UIR.FS0
UIR.CLK↑
-BUSY L
CLK↑
-CLR
D06
UI15
UI14
UI13
UI12
UI11
UI10
UI9
UI8
GET DATA
ERR IF START BLOCK
UIR.WS1
UIR.WS0
DONE TEST
UIR.CS1
UIR.CS0
NC
D07
UI23
UI22
UI21
UI20
UI19
UI18
UI17
UI16
CYLINDER TAG
HEAD TAG
TAG ENABLE
READ GATE
WRITE GATE
PRE GATE
UIR.HEADER STROBE
DATA FIELD
D08
512X8
TS PROM
74S472
D7
D6
D5
D4
D3
D2
D1
D0
A8
A7
A6
A5
A4
A3
A2
A1
A0
-CE
CMD2
CMD1
CMD0
UPC5
UPC4
UPC3
UPC2
UPC1
UPC0
GND
D03
D04
D05



DISK CONTROL	WRITE BUFFER	30-SEP-1978 02:27	AI: CADRDC; DCWBUF


26S10
OC BUS
XCVR
NC
XBAOPAR
XBAO21
XBAO20
4: 0 3:
5: 1 6:
11: 2 10:
13: 3 14:
-B0 2:
-B1 7:
-B2 9:
-B3 15:
NC
XBAIPAR
XBAI21
XBAI20
NC
-XADDR.PAR
-XADDR21
-XADDR20
DRIVE XBUS ADDR L
12:-ENB
E24
XBAO19
XBAO18
XBAO17
XBAO16
XBAI19
XBAI18
XBAI17
XBAI16
-XADDR19
-XADDR18
-XADDR17
-XADDR16
F24
XBAO15
XBAO14
XBAO13
XBAO12
XBAI15
XBAI14
XBAI13
XBAI12
-XADDR15
-XADDR14
-XADDR13
-XADDR12
E23
CHAN.MASTER
13≡
S04
12≡
DRIVE XBUS ADDR L
D19
XBAO11
XBAO10
XBAO9
XBAO8
XBAI11
XBAI10
XBAI9
XBAI8
-XADDR11
-XADDR10
-XADDR9
-XADDR8
F23
XBAO7
XBAO6
XBAO5
XBAO4
XBAI7
XBAI6
XBAI5
XBAI4
-XADDR7
-XADDR6
-XADDR5
-XADDR4
F22
XBAO3
XBAO2
XBAO1
XBAO0
XBAI3
XBAI2
XBAI1
XBAI0
-XADDR3
-XADDR2
-XADDR1
-XADDR0
F21

PDP-11 Unibus Chaosnet Interface CHAOS;UBCHNI UML 17-DEC-80 1800

******* DIP MAP *******

DM8838 QUBXCV x F20	74LS74 QADTIM xx E20	74LS193 QADTIM x D20	74LS193 QADTIM x C20	74LS174 LMUCON x B20	74LS193 QADTIM x A20
DM8838 QUBXCV x F19	74LS193 QADTIM x E19	74LS244 LMDATP x D19	74S133 LMUCON x C19	74LS164 LMTURN x B19	74LS193 LMTURN x A19
DM8838 QUBXCV x F18	74LS244 LMDATP x E18	74LS244 LMDATP x D18	74LS138 LMUCON x C18	74LS164 LMTURN x B18	74LS193 LMTURN x A18
DM8838 QUBXCV x F17	74LS244 LMDATP x E17	74LS244 LMDATP x D17	74LS244 LMDATP x C17	74S04 QUBINT xoooxx B17	74LS193 LMTURN x A17
74S38 QUBINT xxxx F16	25LS2521 QADADR x E16	74LS244 LMDATP x D16	74LS244 LMDATP x C16	74LS02 QUBINT xxxx B16	74LS161 LMTURN x A16
74S38 QUBINT xoxx F15	25LS2521 QADADR x E15	74S32 LMRCTL xxxx D15	74S00 LMTBFC xxxx C15	74S74 LMMODU xx B15	TD100 QUBINT x A15
74S175 QUBINT x F14	DUMMY QUBINT x E14	74S112 LMTURN xx D14	74S00 LMTBFC xxxx C14	74S04 LMTBFC xxxxxx B14	74LS163 QADTIM x A14
DUMMY QUBINT x F13	74S04 QADADR xxxxxx E13	74S112 LMTBFC xx D13	25LS193 LMTBUF x C13	74 165 LMTBUF x B13	74S37 QADADR xxox A13
74S00 QUBINT xxxx F12	74LS163 QADTIM x E12	DUMMY LMMYNM x D12	25LS193 LMTBUF x C12	74 165 LMTBUF x B12	TD250 QADADR x A12
74LS10 QUBINT xxx F11	74LS02 QADADR xxox E11	P SIP100 LMMYNM x D11	25LS193 LMTBUF x C11	TD100 LMDETC x B11	TD25 LMDETC x A11
DM8838 QUBINT x F10	74S08 LMRCTL xxxx E10	DUMMY LMMYNM x D10	2147 LMTBUF x C10	74S08 LMTBFC xxxx B10	74S288 LMMODU x A10

PDP-11 Unibus Chaosnet Interface CHAOS;UBCHNI UML 17-DEC-80 1801

******* DIP MAP *******

DM8837A QUBXCV x	DUMMY QADADR x	74 279 LMRCTL oxox	9401 LMTBUF x	74S04 LMDETC xxxxxx	74S112 LMMODU xx
F09	E09	D09	C09	B09	A09
DM8837A QUBXCV x	74S74 QADADR xx	74S74 LMRCTL xx	74S51 LMTBUF xx	74S02 LMTBFC xxxx	74S374 LMMODU x
F08	E08	D08	C08	B08	A08
DM8837A QUBXCV x	PULLUP-D QADADR x	74S00 LMRBUF xxxx	9401 LMRBUF x	74LS164 LMRBUF x	74LS164 LMRBUF x
F07	E07	D07	C07	B07	A07
DM8837A QUBXCV x	74S251 LMMYNM x	25LS193 LMRBUF x	74S74 LMRCLK xx	74LS161 LMRCLK x	74S112 LMTCLK xx
F06	E06	D06	C06	B06	A06
74S251 LMMYNM x	74S51 LMRBUF xx	25LS193 LMRBUF x	74S37 LMRBUF xxxx	74LS161 LMRCLK x	EXAR-CL LMTCLK x
F05	E05	D05	C05	B05	A05
74LS161 LMMYNM x	74S00 LMUCON xxxx	25LS193 LMRBUF x	2147 LMRBUF x	TD100 LMDETC x	26S02 LMRCTL xx
F04	E04	D04	C04	B04	A04
74LS161 LMMYNM x	74S00 LMRCTL xxxx	74S04 LMTCLK xxxxxx	TD250 LMRCLK x	74S163 LMTCLK x	DUMMY LMLNDR x
F03	E03	D03	C03	B03	A03
74LS161 LMMYNM x	74S158 LMLNDR x	74S10 LMMODU xxx	74S08 LMRBUF xxxx	74S74 LMDETC xx	26LS31 LMLNDR xxxx
F02	E02	D02	C02	B02	A02
74S11 LMMODU xxx	74S174 LMMYNM x	74S287 LMMYNM x	74S74 LMDETC xx	74S04 LMRCLK xxxxxx	26LS33 LMLNDR xxxx
F01	E01	D01	C01	B01	A01

PDP-11 Unibus Chaosnet Interface CHAOS;UBCHNI UML 17-DEC-80 1803
******* EDGE CONNECTIONS Flags: (# Output, @ Terminator, ---- Dedicated ground, ++++ Dedicated power) *******

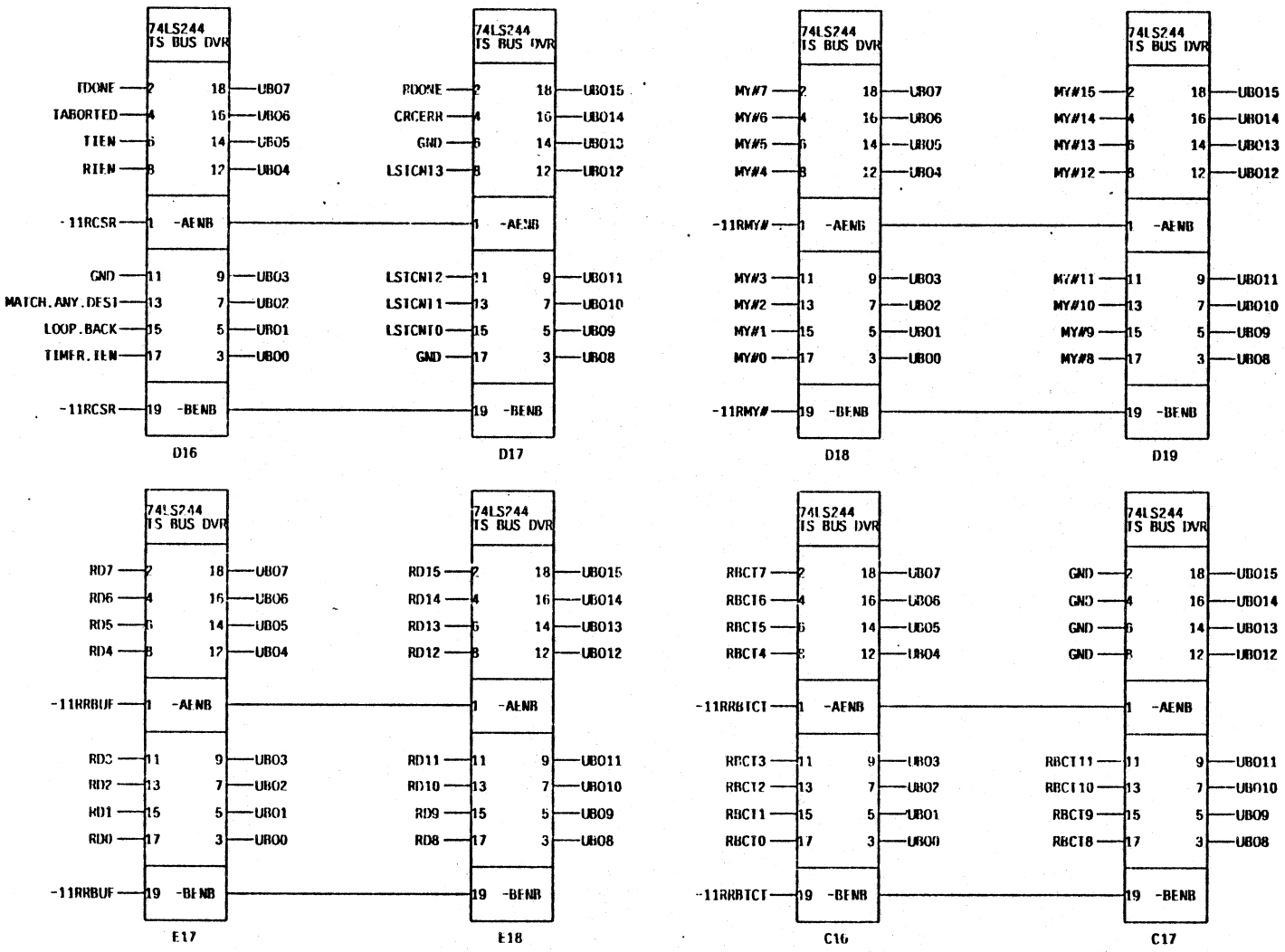
	-A-	-B-	-C-	-D-
1	A1 NPG.IN* A2 +5.0V++++++++++++++++++	A1 A2 +5.0V++++++++++++++++++	A1 GND # A2 +5.0V++++++++++++++++++	A1 A2 +5.0V++++++++++++++++++
2	B1 NPG.OUT* B2 -5.0V	B1 B2 -5.0V	B1 B2 -5.0V	B1 B2 -5.0V
3	C1 C2 GND---------- #	C1 C2 GND---------- #	C1 -A12* C2 GND---------- #	C1 C2 GND---------- #
4	D1 D2 -D15* #	D1 D2 -BR7*	D1 -A17* D2 -A15*	D1 -BBSY* # D2
5	E1 E2 -D14* #	E1 E2 -BR6*	E1 -MSYN* E2 -A16*	E1 E2
6	F1 NC # F2 -D13* #	F1 NC # F2 -BR5*	F1 -A2* F2 -C1*	F1 NC # F2
7	H1 -D11* # H2 -D12* #	H1 H2 -BR4*	H1 -A1* H2 -A0*	H1 H2
8	J1 J2 -D10* #	J1 J2	J1 -SSYN* # J2 -C0*	J1 -NPR* J2
9	K1 K2 -D9* #	K1 K2 BG7.IN*	K1 -A14* K2 -A13*	K1 K2
10	L1 L2 -D8* #	L1 -INIT* L2 BG7.OUT*	L1 -A11* L2	L1 L2
11	M1 M2 -D7* #	M1 M2 BG6.IN*	M1 M2 GND #	M1 -INTR* # M2
12	N1 NC # N2 -D4* #	N1 NC # N2 BG6.OUT*	N1 GND # N2 -A8*	N1 NC # N2
13	P1 -BOOT* P2 -D5* #	P1 P2 BG5.IN*	P1 -A10* P2 -A7*	P1 P2
14	R1 R2 -D1* #	R1 R2 BG5.OUT*	R1 -A9* R2	R1 R2
15	S1 S2 -D0* #	S1 S2 BG4.IN*	S1 S2 GND #	S1 S2
16	T1 GND---------- # T2 -D3* #	T1 GND---------- # T2 BG4.OUT*	T1 GND---------- # T2	T1 GND---------- # T2 -SACK* #
17	U1 U2 -D2* #	U1 U2	U1 -A6* U2 -A4*	U1 U2
18	V1 V2 -D6* #	V1 V2	V1 -A5* V2 -A3*	V1 V2
19				
20				
21				
22				
23				
24				
25				

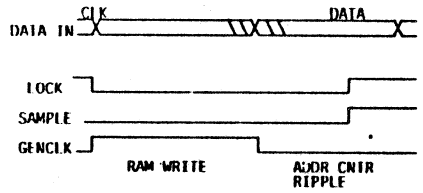
PDP-11 Unibus Chaosnet Interface CHAOS:UBCHNI UML 17-DEC-80 1803
******* EDGE CONNECTIONS Flags: (# Output, @ Terminator, ---- Dedicated ground, ++++ Dedicated power) *******

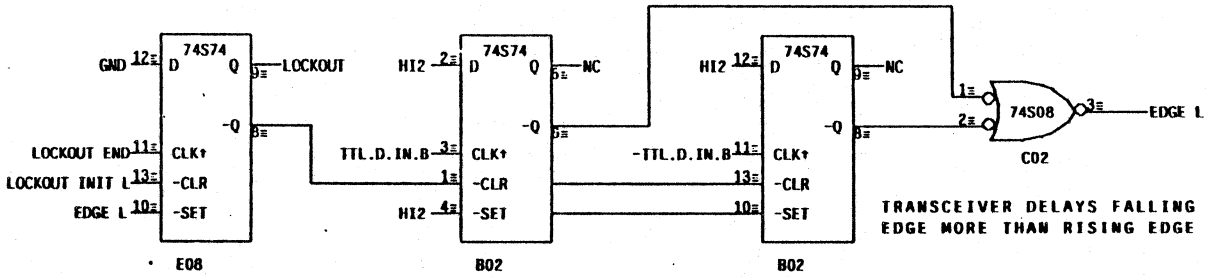
-J01-	-J02-	-J03-	-J04-
01	01 GND #	01	01
02	02 INTERFERE+	02	02
03	03 INTERFERE-	03	03
04	04 GND #	04	04
05	05 RCVR.DATA+	05	05
06	06 RCVR.DATA-	06	06
07	07 GND #	07	07
08	08 TRANS.DATA+ #	08	08
09	09 TRANS.DATA- #	09	09
10	10 GND #	10	10
11	11	11	11
12	12	12	12
13	13	13	13
14	14	14	14
15	15	15	15
16	16	16	16
17	17	17	17
18	18	18	18
19	19	19	19
20	20	20	20
21	21	21 ----------	21 ----------
22	22	22 ----------	22 ----------
23	23	23 ----------	23 ----------
24	24	24 ----------	24 ----------
25	25	25 ----------	25 ----------
26 ----------	26 GND---------- #	26 ----------	26 ----------
27 ----------	27 GND---------- #	27 ----------	27 ----------
28 ----------	28 GND---------- #	28 ----------	28 ----------
29 ----------	29 GND---------- #	29 ----------	29 ----------
30 ----------	30 GND---------- #	30 ----------	30 ----------
31 ----------	31 GND---------- #	31 ----------	31 ----------
32 ----------	32 GND---------- #	32 ----------	32 ----------
33 ----------	33 GND---------- #	33 ----------	33 ----------
34 ----------	34 GND---------- #	34 ----------	34 ----------
35 ----------	35 GND---------- #	35 ----------	35 ----------
36 ----------	36 ----------	36 ----------	36 ----------
37 ----------	37 ----------	37 ----------	37 ----------
38 ----------	38 ----------	38 ----------	38 ----------
39 ----------	39 ----------	39 ----------	39 ----------
40 ----------	40 ----------	40 ----------	40 ----------
41 ----------	41 ----------		
42 ----------	42 ----------		
43 ----------	43 ----------		
44 ----------	44 ----------		
45 ----------	45 ----------		
46 ----------	46 ----------		
47 ----------	47 ----------		
48 ----------	48 ----------		
49 ----------	49 ----------		
50 ----------	50 ----------		

PDP-11 Unibus Chaosnet Interface CHAOS;UBCHNI UML 17-DEC-80 1803
******* EDGE CONNECTIONS Flags: (# Output, @ Terminator, ---- Dedicated ground, ++++ Dedicated power) *******

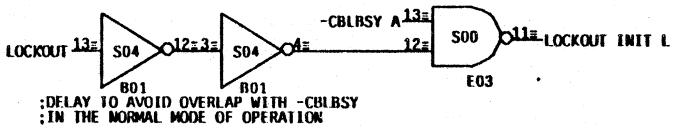
-J05-	-J06-	-J07-	-J08-
01 02	01 02	01 02	01 02
03 04	03 04	03 04	03 04
05 06	05 06	05 06	05 06
07 08	07 08	07 08	07 08
09 10	09 10	09 10	09 10
11 12	11 12	11 12	11 12
13 14	13 14	13 14	13 14
15 16	15 16	15 16	15 16
17 18	17 18	17 18	17 18
19 20	19 20	19 20	19 20
21 22	21 22	21 ---------- 22 ----------	21 ---------- 22 ----------
23 24	23 24	23 ---------- 24 ----------	23 ---------- 24 ----------
25 26 ----------	25 26 ----------	25 ---------- 26 ----------	25 ---------- 26 ----------
27 ---------- 28 ----------	27 ---------- 28 ----------	27 ---------- 28 ----------	27 ---------- 28 ----------
29 ---------- 30 ----------	29 ---------- 30 ----------	29 ---------- 30 ----------	29 ---------- 30 ----------
31 ---------- 32 ----------	31 ---------- 32 ----------	31 ---------- 32 ----------	31 ---------- 32 ----------
33 ---------- 34 ----------	33 ---------- 34 ----------	33 ---------- 34 ----------	33 ---------- 34 ----------
35 ---------- 36 ----------	35 ---------- 36 ----------	35 ---------- 36 ----------	35 ---------- 36 ----------
37 ---------- 38 ----------	37 ---------- 38 ----------	37 ---------- 38 ----------	37 ---------- 38 ----------
39 ---------- 40 ----------	39 ---------- 40 ----------	39 ---------- 40 ----------	39 ---------- 40 ----------
41 ---------- 42 ----------	41 ---------- 42 ----------		
43 ---------- 44 ----------	43 ---------- 44 ----------		
45 ---------- 46 ----------	45 ---------- 46 ----------		
47 ---------- 48 ----------	47 ---------- 48 ----------		
49 ---------- 50 ----------	49 ---------- 50 ----------		

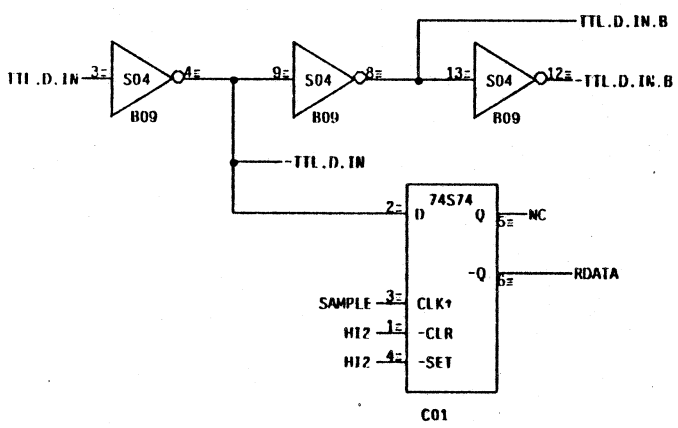

74LS244
TS BUS DVR
TDONE
TABORTED
TIEN
RIEN
-11RCSR
-AENB
GND
MATCH.ANY.DEST
LOOP.BACK
TIMER.IEN
-11RCSR
-BENB
UB07
UB06
UB05
UB04
UB03
UB02
UB01
UB00
D16
RDONE
CRCERR
GND
LSTCNT3
LSTCNT2
LSTCNT1
LSTCNT0
GND
UB015
UB014
UB013
UB012
UB011
UB010
UB09
UB08
D17
MY#7
MY#6
MY#5
MY#4
-11RMY#
MY#3
MY#2
MY#1
MY#0
-11RMY#
D18
MY#15
MY#14
MY#13
MY#12
MY#11
MY#10
MY#9
MY#8
D19
RD7
RD6
RD5
RD4
-11RRBUF
RD3
RD2
RD1
RD0
-11RRBUF
E17
RD15
RD14
RD13
RD12
RD11
RD10
RD9
RD8
E18
RBCT7
RBCT6
RBCT5
RBCT4
-11RRBTCT
RBCT3
RBCT2
RBCT1
RBCT0
-11RRBTCT
C16
GND
GND
GND
GND
RBCT11
RBCT10
RBCT9
RBCT8
C17

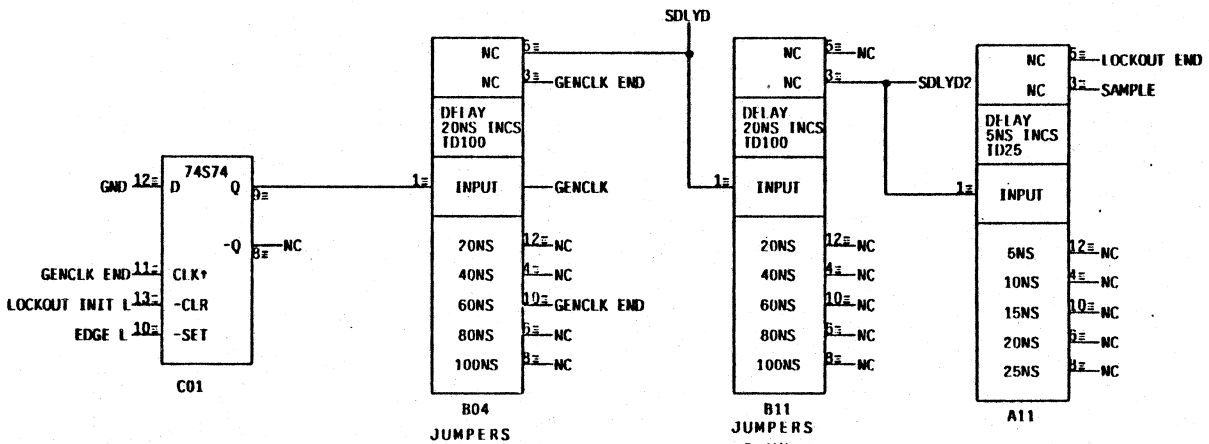




SAMPLE should be timed to occur in the middle of the data part of the cycle. Normally, this will be .75 of the cycle time less 15-25 ns. LOCKOUT END should be 5 ns earlier.







JUMPERS
8 MHz

5(8) -> 20 ns 12(15)
3(6) -> 40 ns 4(7)

4 MHz

5(8) -> 100 ns 8(11)
3(6) -> 60 ns 10(13)

JUMPERS
8 MHz

3(6) -> 20 ns 12(15)

4 MHz

3(6) -> 60 ns 10(13)

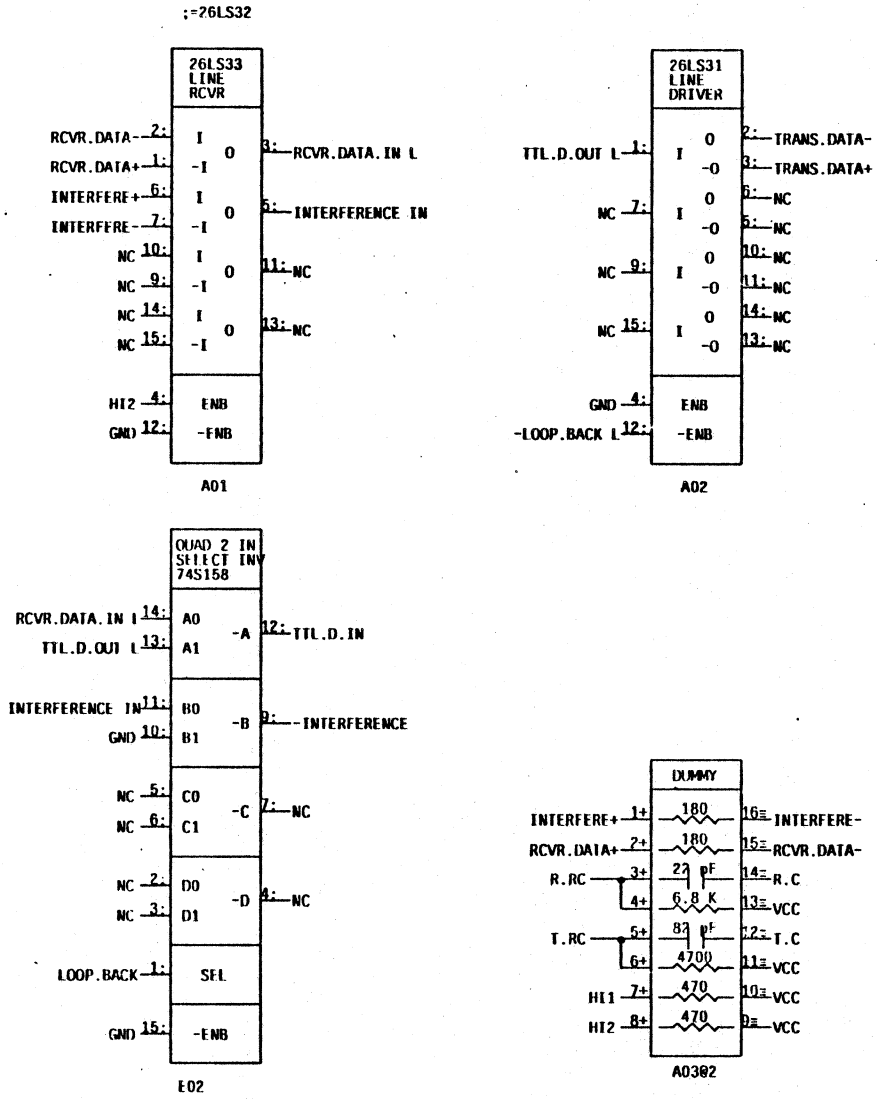
JUMPERS
8 MHz

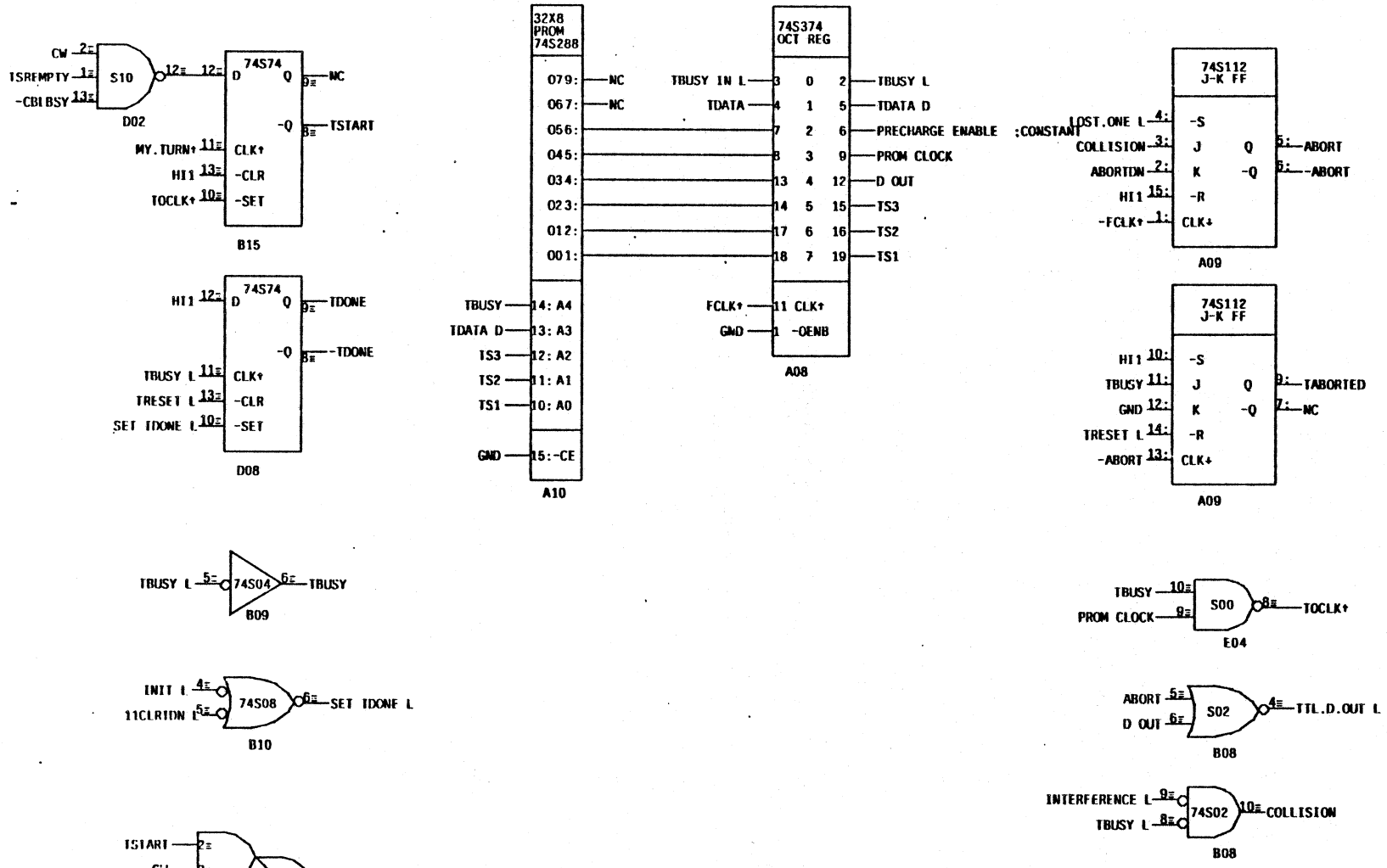
3(6) -> 25 ns 8(11)
5(8) -> 20 ns 6(9)

4 MHz

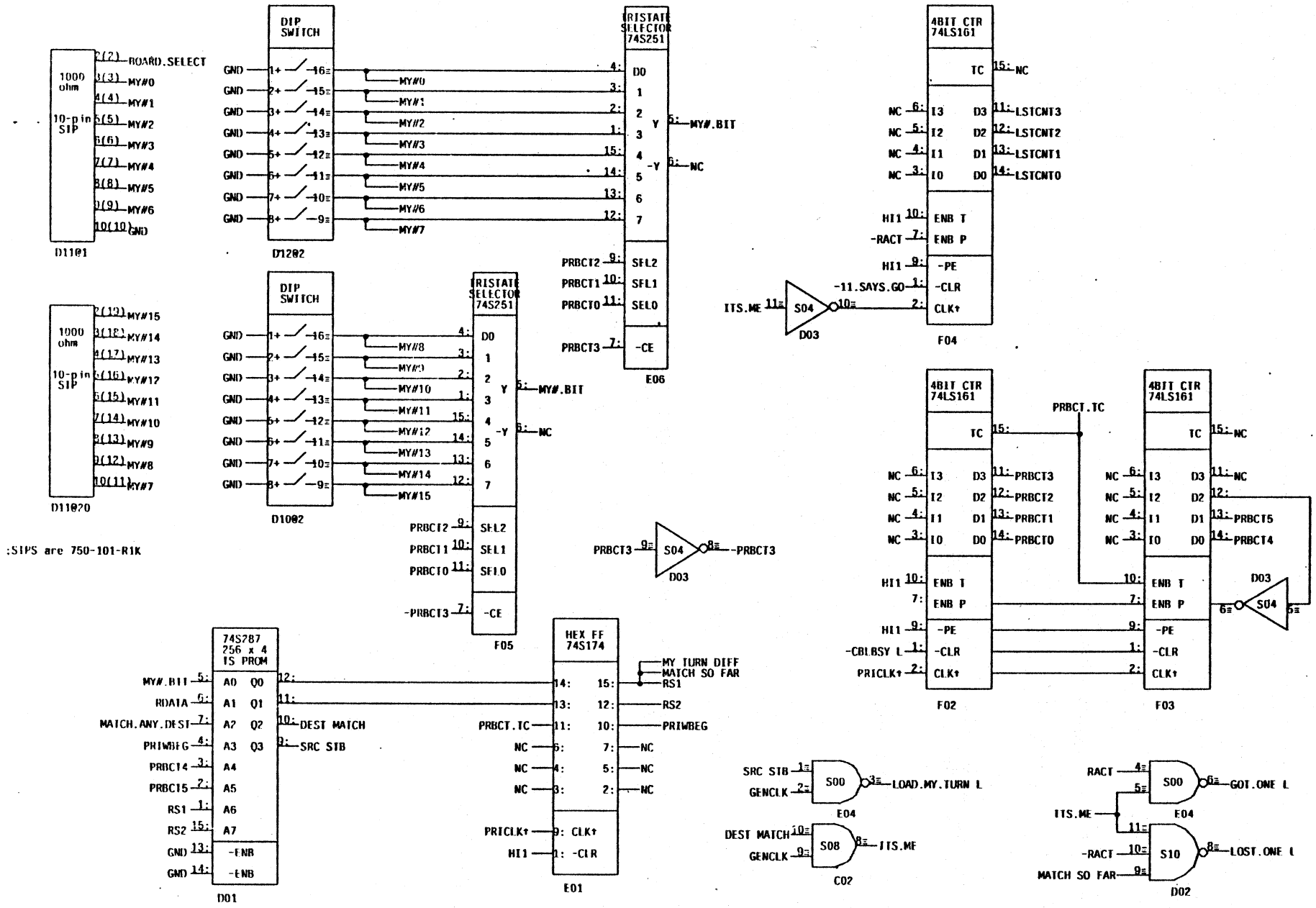
3(6) -> 15 ns 10(13)
5(8) -> 10 ns 4(7)

LISP MACHINE CHAOS NET DETECTOR	13-AUG-1980 09:46	AI: CHAOS; LMDETC

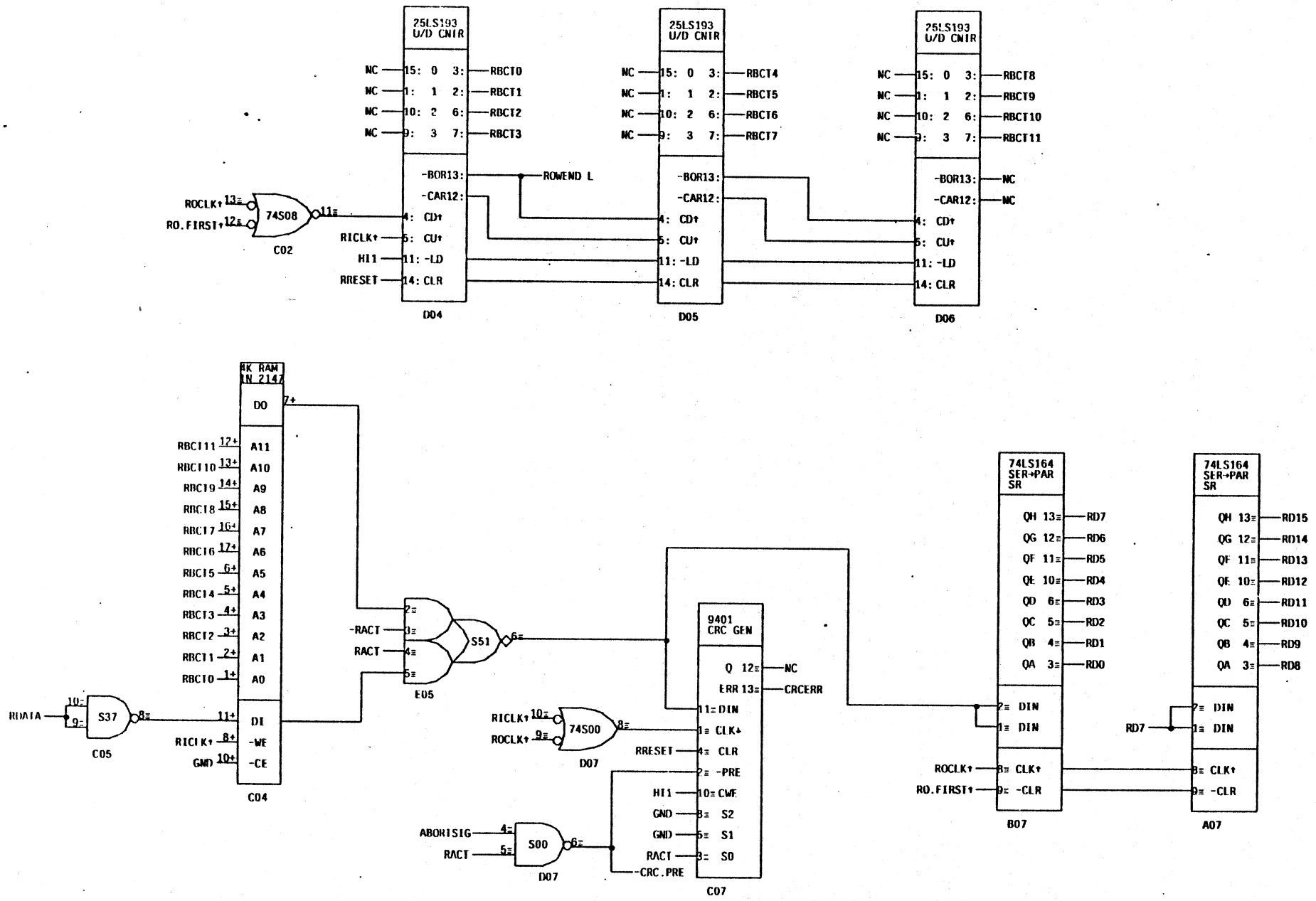

:=26LS32
26LS33
LINE
RCVR
RCVR.DATA- 2:
RCVR.DATA+ 1:
INTERFERE+ 6:
INTERFERE- 7:
NC 10:
NC 9:
NC 14:
NC 15:
I
-I
O
3: RCVR.DATA.IN L
5: INTERFERENCE IN
11: NC
13: NC
HI2 4:
GND 12:
ENB
-ENB
A01
26LS31
LINE
DRIVER
TTL.D.OUT L 1:
NC 7:
NC 9:
NC 15:
-O
2: TRANS.DATA-
3: TRANS.DATA+
6: NC
5: NC
10: NC
11: NC
14: NC
13: NC
GND 4:
-LOOP.BACK L 12:
A02
QUAD 2 IN
SELECT INV
74S158
RCVR.DATA.IN L 14:
TTL.D.OUT L 13:
A0
A1
-A
12: TTL.D.IN
INTERFERENCE IN 11:
GND 10:
B0
B1
-B
9: -INTERFERENCE
NC 5:
NC 6:
C0
C1
-C
7: NC
NC 2:
NC 3:
D0
D1
-D
4: NC
LOOP.BACK 1:
SEL
GND 15:
-ENB
E02
DUMMY
INTERFERE+ 1+
RCVR.DATA+ 2+
R.RC
3+
4+
T.RC
5+
6+
HI1 7+
HI2 8+
180
180
22 pF
6.8 K
82 pF
4700
470
470
16= INTERFERE-
15= RCVR.DATA-
14= R.C
13= VCC
12= T.C
11= VCC
10= VCC
9= VCC
A0302



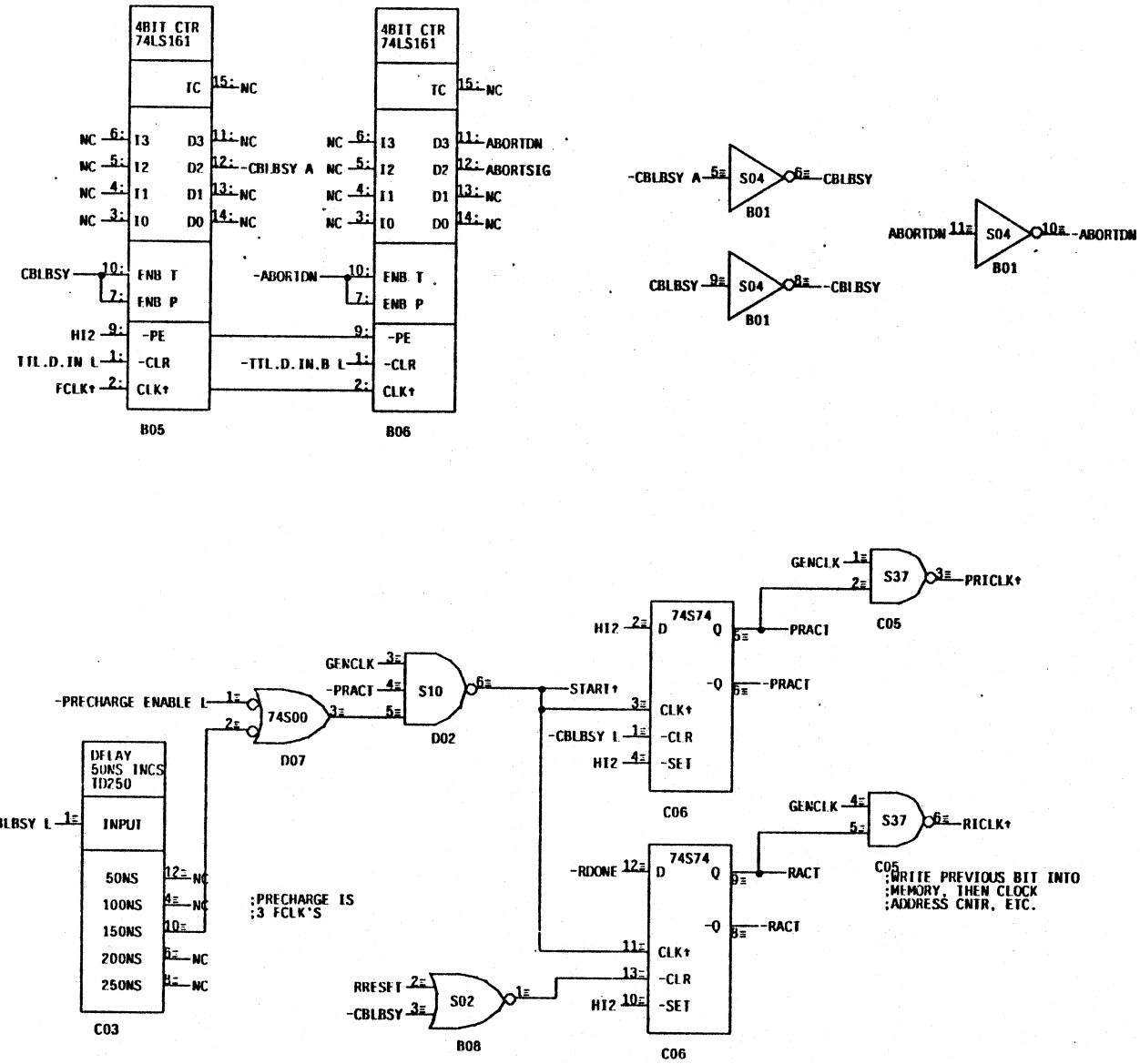
LISP MACHINE CHAOS NET SERIAL MODULATOR	13-AUG-1980 09:46	AI: CHAOS; LMMODU



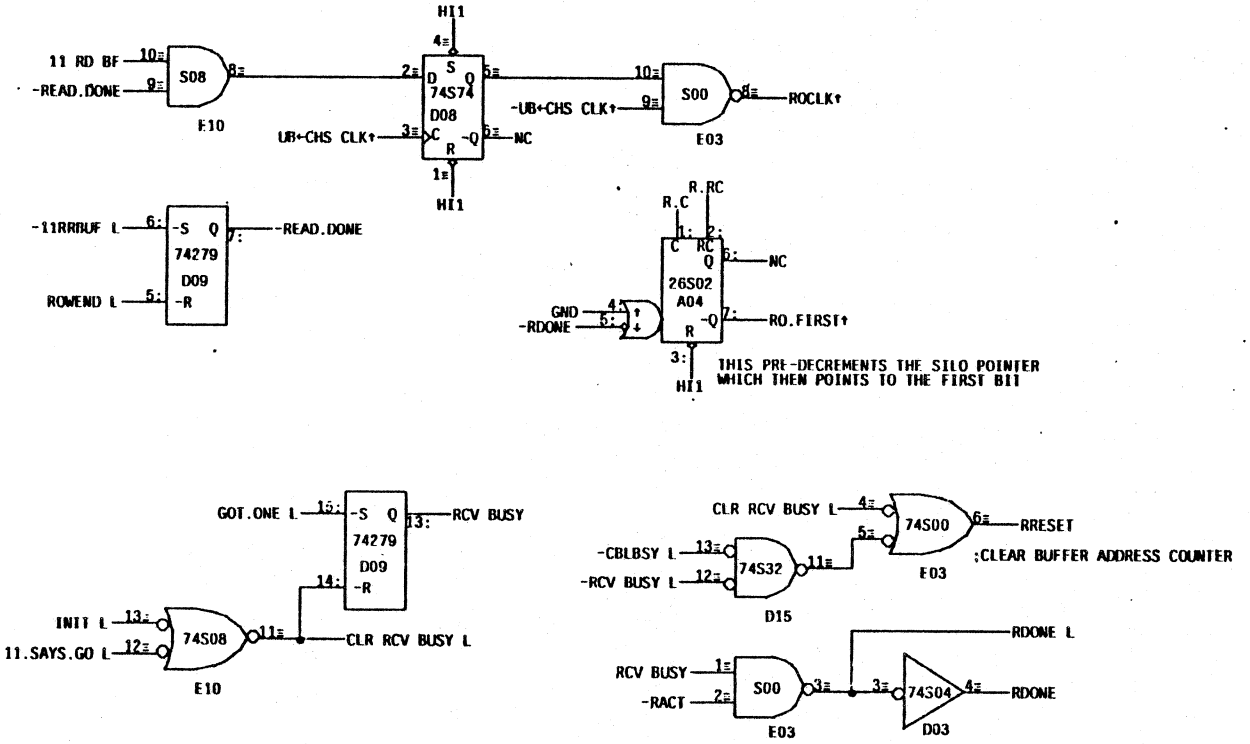
LISP MACHINE CHAOS NET FIND MSGS DESTINED FOR ME	13-AUG-1980 09:54	AI: CHAOS; LMMYNM



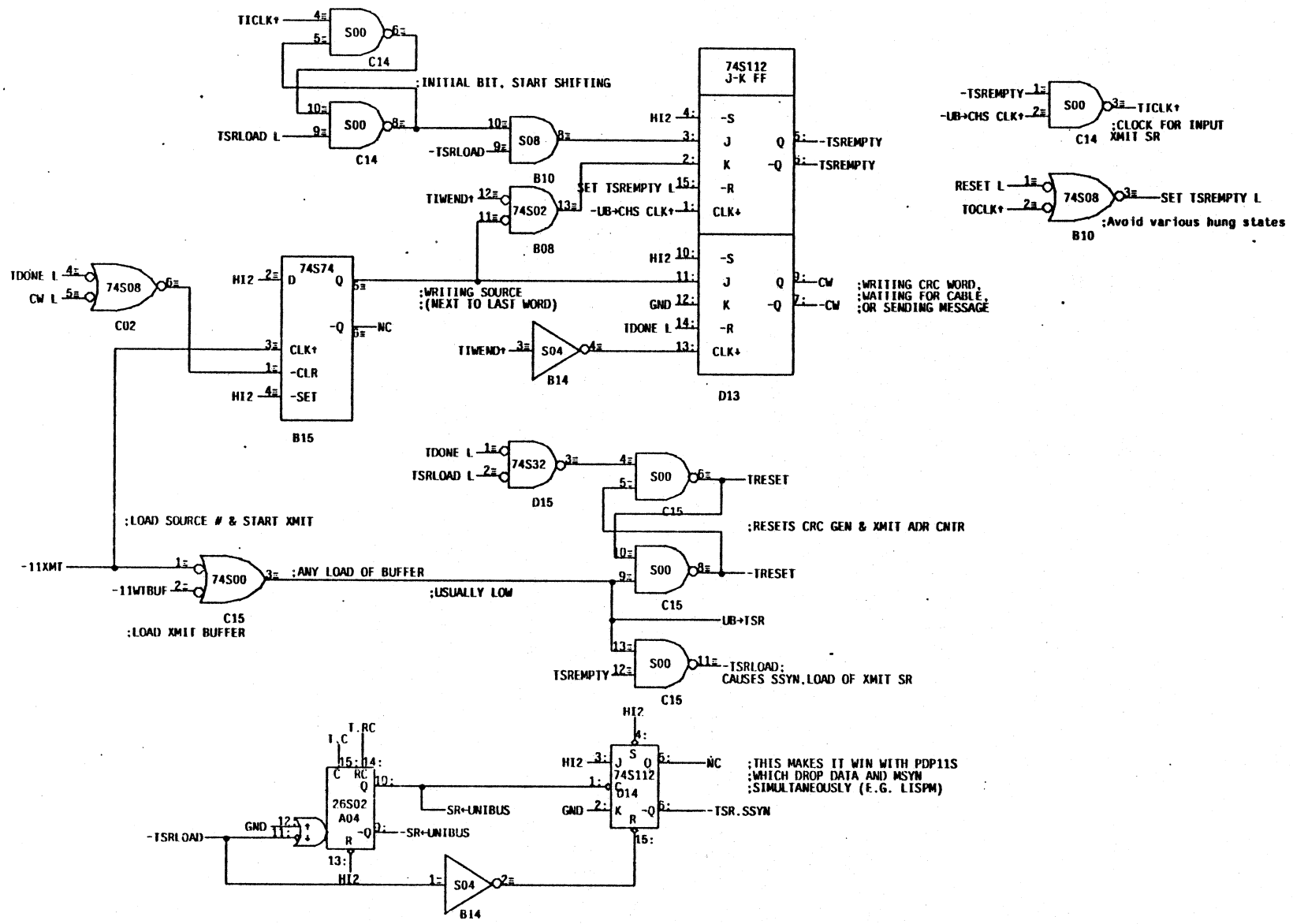
LISP MACHINE CHAOS NET RECEIVER BUFFER | 13-AUG-1980 09:47 | AI: CHAOS; LMRBUF



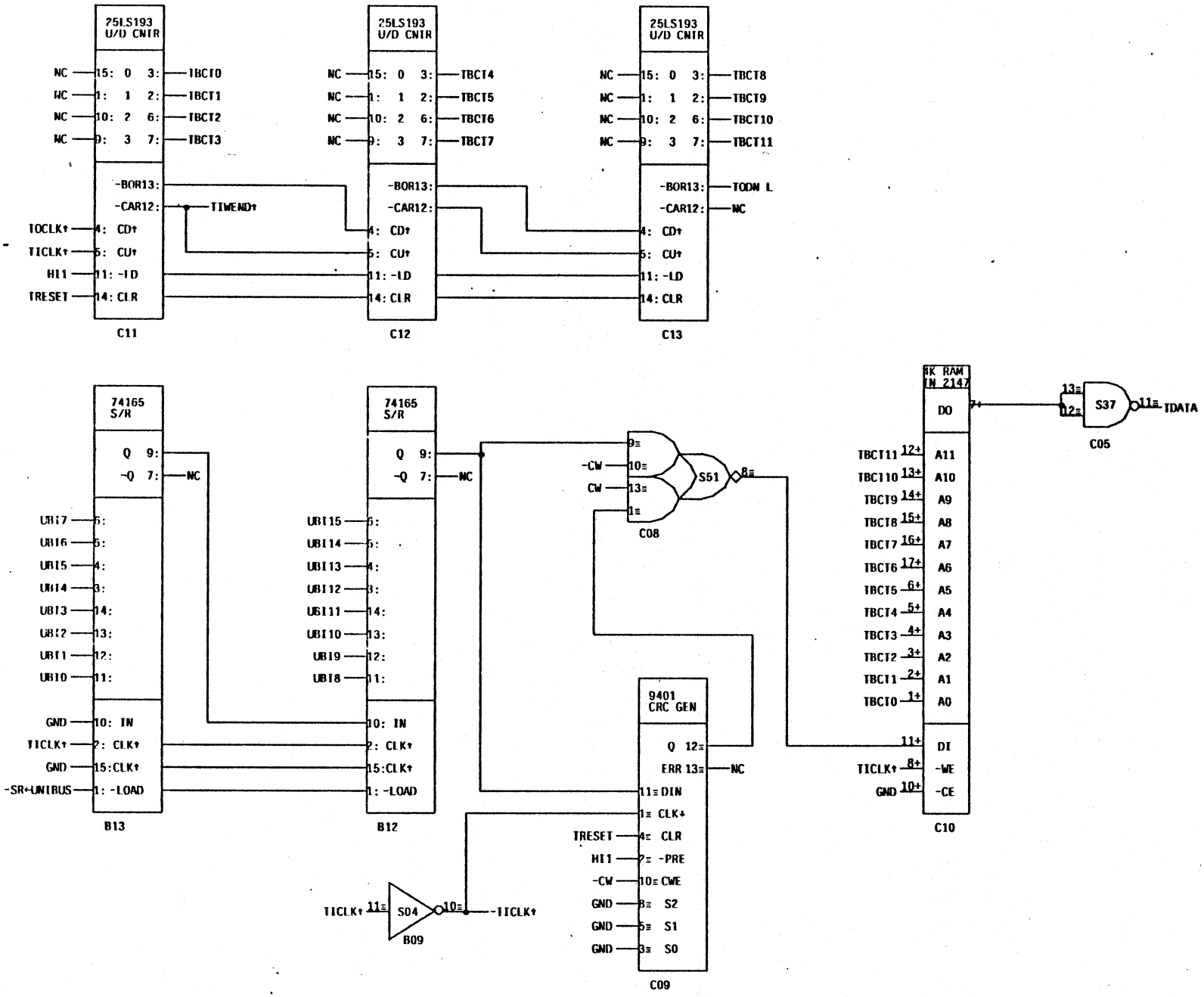
LISP MACHINE CHAOS NET RECEIVE CLOCK	13-AUG-1980 09:48	AI: CHAOS; LMRCLK



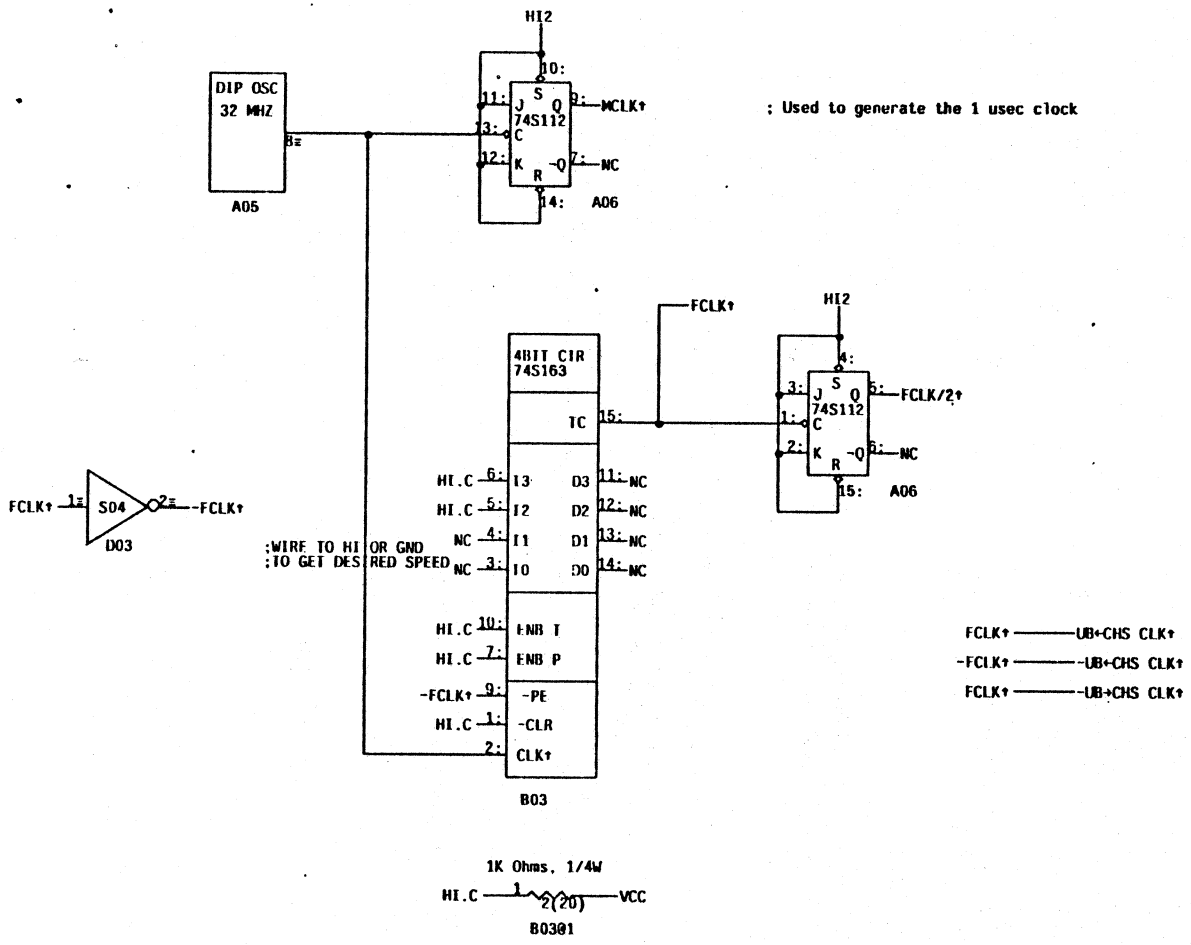
LISP MACHINE CHAOS NET RECEIVE CONTROL	13-AUG-1980 09:48	AI: CHAOS; LMRCTL



LISP MACHINE CHAOS NET TRANSMIT CONTROL	13-AUG-1980 09:49	AI: CHAOS; LMTBFC

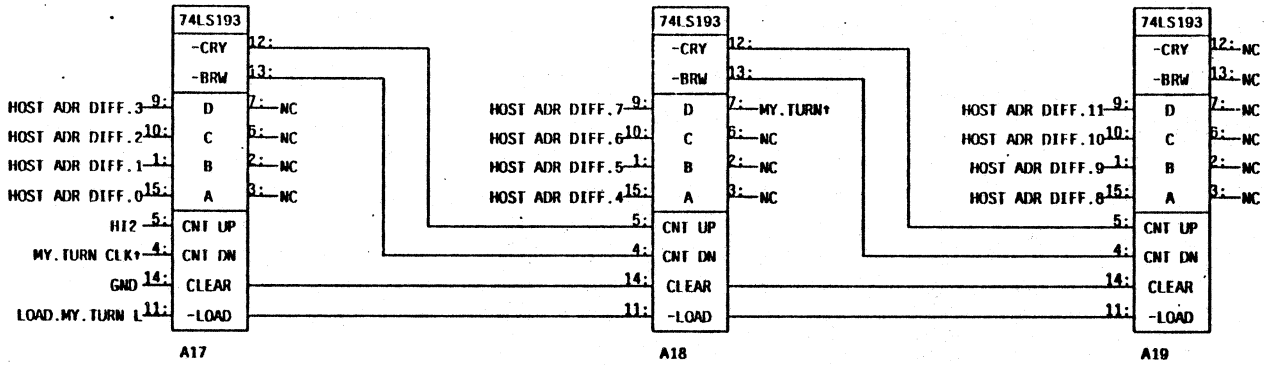


LISP MACHINE CHAOS NET TRANSMIT BUFFER	13-AUG-1980 09:49	AI: CHAOS; LMTBUF

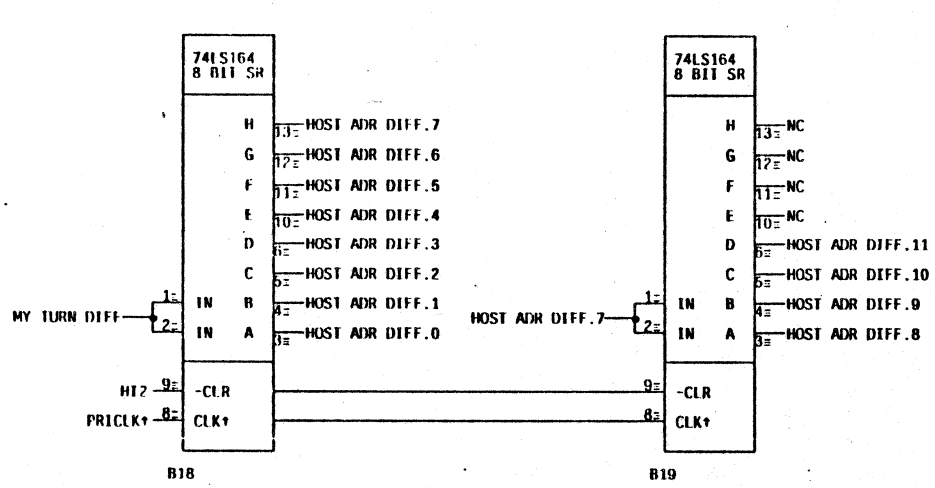


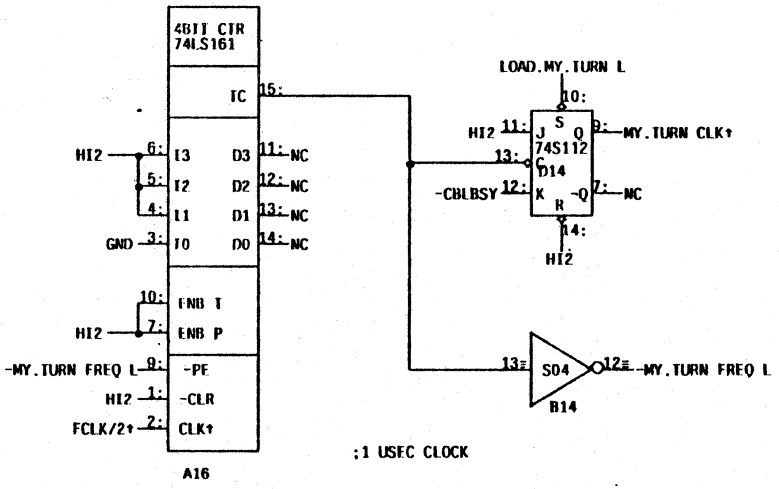
LISP MACHINE CHAOS NET TRANSMIT CLOCK	17-DEC-1980 11:19	AI: CHAOS; LMTCLK

"My Turn" Counter

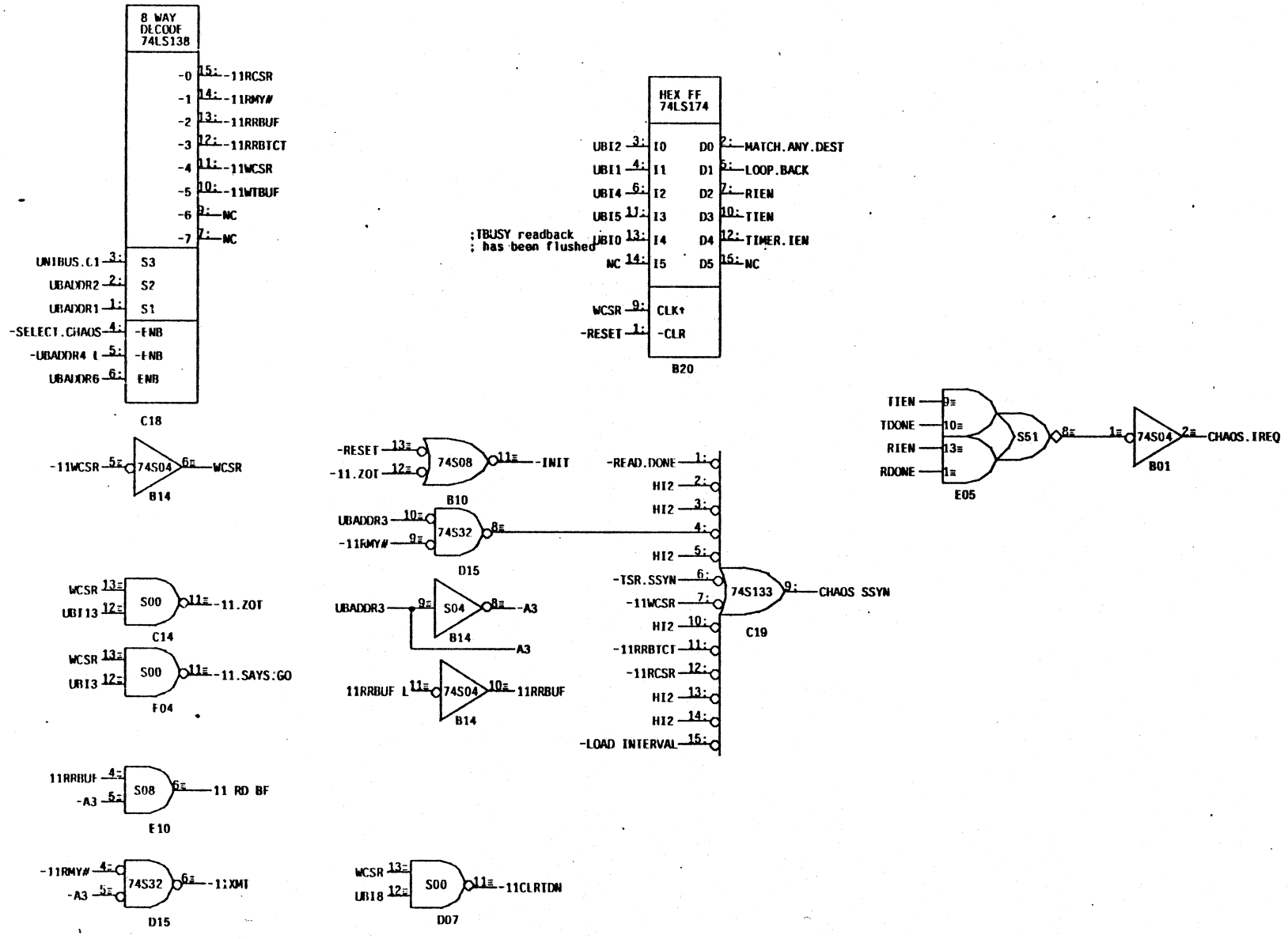


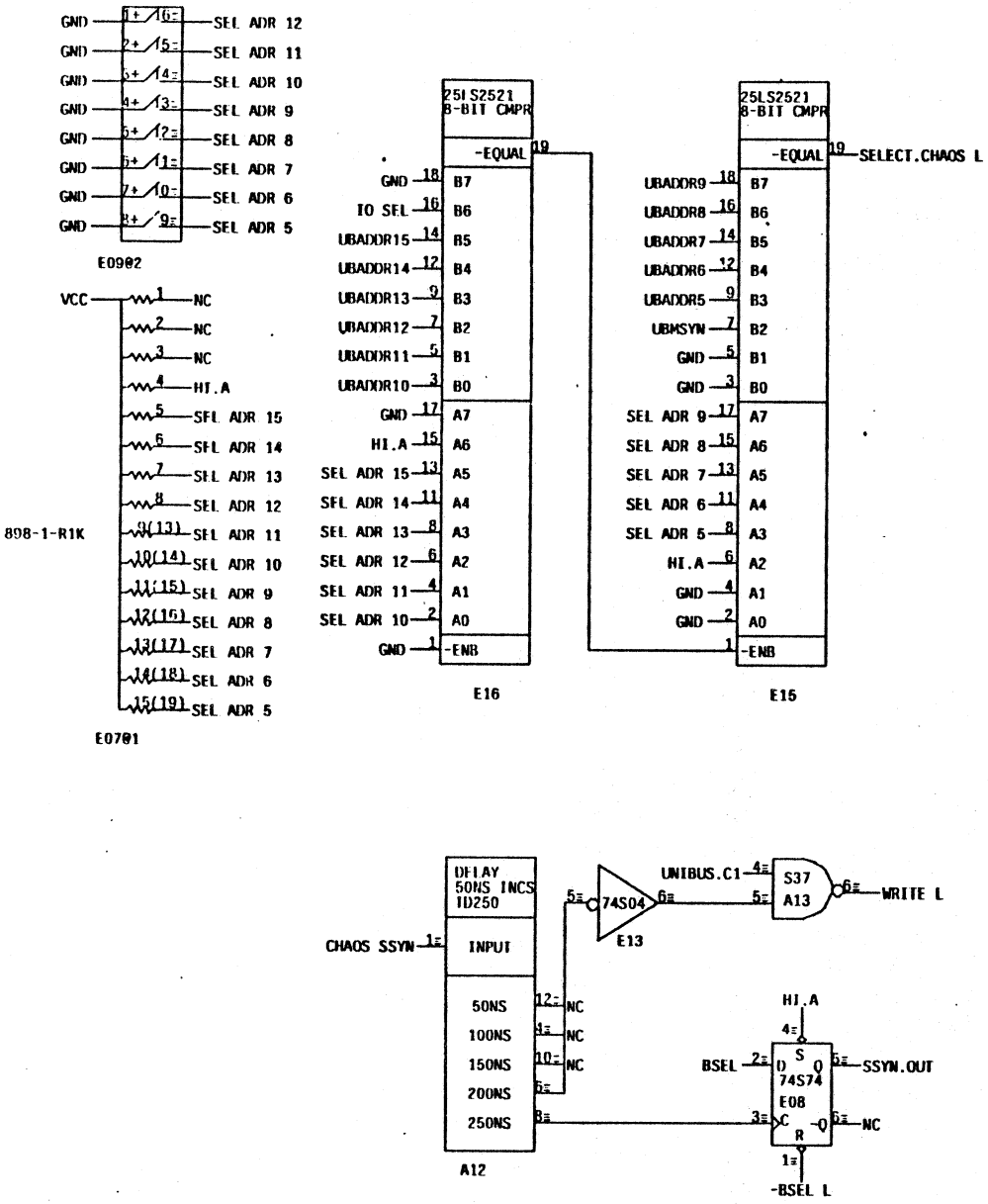
;DEFAULT IS MYTURN.MOD 2+7 OR 64 USECS

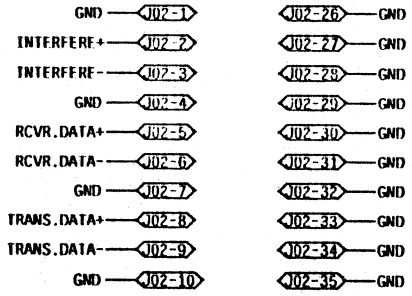


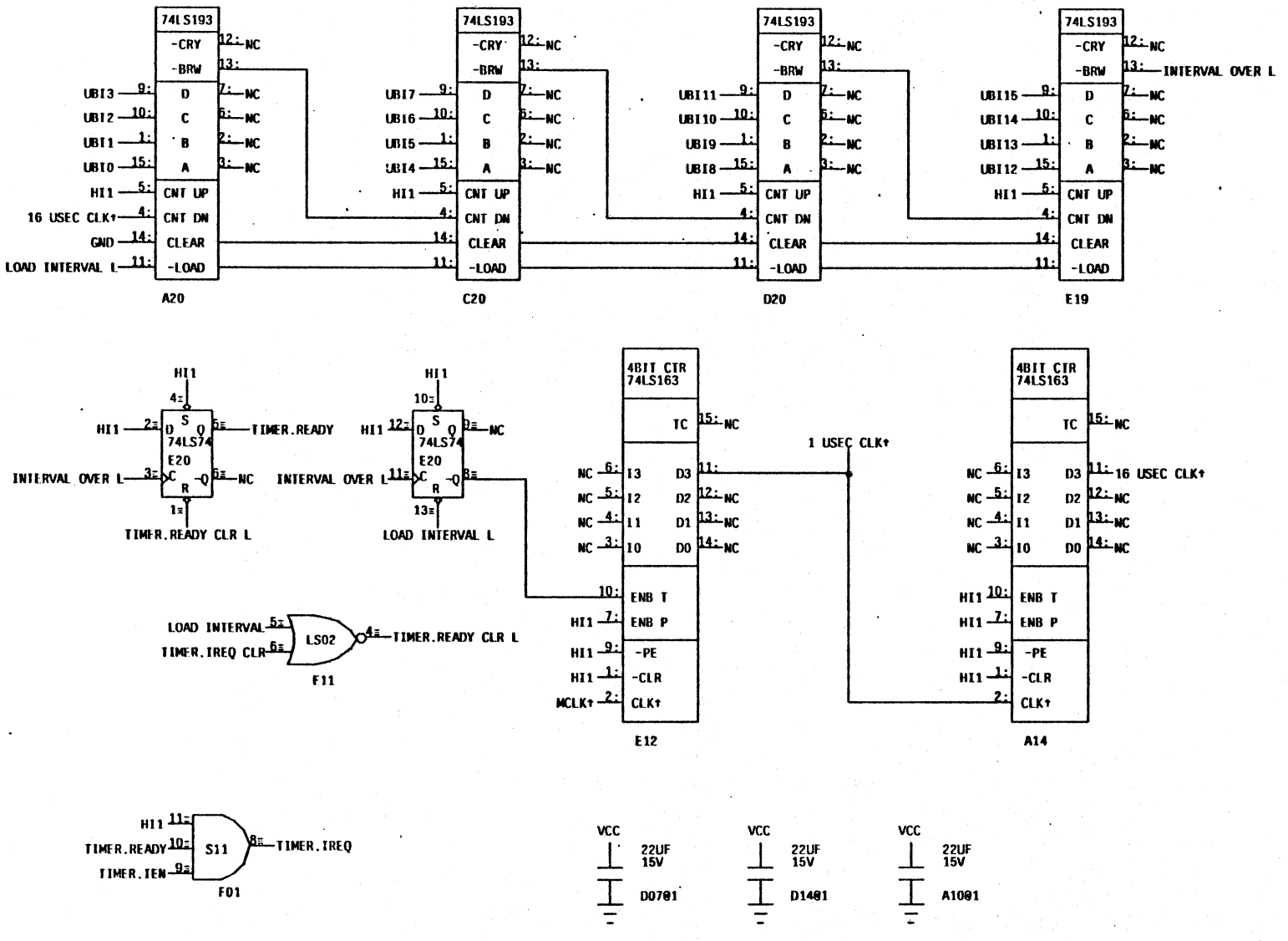


;1 USEC CLOCK

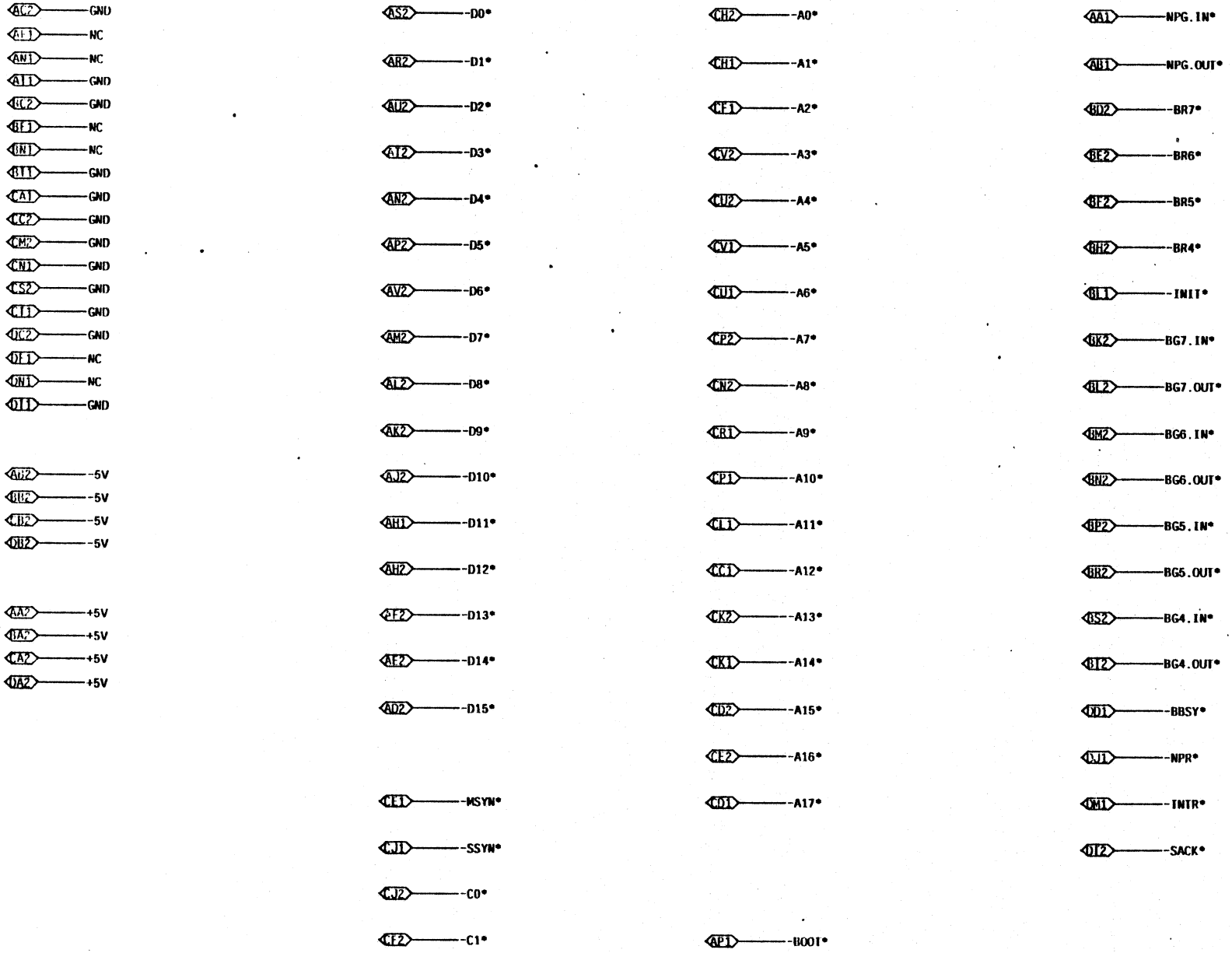

8 WAY DECODE 74LS138
-0 15: -11RCSR
-1 14: -11RMY#
-2 13: -11RRBUF
-3 12: -11RRBTCT
-4 11: -11WCSR
-5 10: -11WTBUF
-6 9: NC
-7 7: NC
UNIBUS.C1 3: S3
UBADDR2 2: S2
UBADDR1 1: S1
-SELECT.CHAOS 4: -ENB
-UBADDR4 L 5: -ENB
UBADDR6 6: ENB
C18
HEX FF 74LS174
UBI2 3: I0 D0 2: MATCH.ANY.DEST
UBI1 4: I1 D1 5: LOOP.BACK
UBI4 6: I2 D2 7: RIEN
UBI5 11: I3 D3 10: TIEN
;TBUSY readback ; has been flushed
UBI0 13: I4 D4 12: TIMER.IEN
NC 14: I5 D5 15: NC
WCSR 9: CLK↑
-RESET 1: -CLR
B20
-11WCSR 5: 74S04 6: WCSR
B14
-RESET 13: 74S08 11: -INIT
-11.ZOT 12:
B10
UBADDR3 10: 74S32 8:
-11RMY# 9:
D15
WCSR 13: S00 11: -11.ZOT
UBI13 12:
C14
WCSR 13: S00 11: -11.SAYS:GO
UBI3 12:
F04
UBADDR3 9: S04 8: -A3
B14
A3
11RRBUF L 11: 74S04 10: 11RRBUF
B14
11RRBUF 4: S08 6: 11 RD BF
-A3 5:
E10
-11RMY# 4: 74S32 6: -11XMT
-A3 5:
D15
WCSR 13: S00 11: -11CLRTDN
UBI8 12:
D07
-READ.DONE 1:
HI2 2:
HI2 3:
4:
HI2 5:
-TSR.SSYN 6:
-11WCSR 7:
HI2 10:
-11RRBTCT 11:
-11RCSR 12:
HI2 13:
HI2 14:
-LOAD INTERVAL 15:
74S133 9: CHAOS SSYN
C19
TIEN 9:
TDONE 10:
RIEN 13:
RDONE 1:
S51 8: 1: 74S04 2: CHAOS.IREQ
E05
B01


GND
SEL ADR 12
SEL ADR 11
SEL ADR 10
SEL ADR 9
SEL ADR 8
SEL ADR 7
SEL ADR 6
SEL ADR 5
E0902
VCC
NC
HI.A
898-1-R1K
E0701
25LS2521
8-BIT CMPR
-EQUAL
-ENB
IO SEL
UBADDR15
UBADDR14
UBADDR13
UBADDR12
UBADDR11
UBADDR10
HI.A
B7
B6
B5
B4
B3
B2
B1
B0
A7
A6
A5
A4
A3
A2
A1
A0
E16
UBADDR9
UBADDR8
UBADDR7
UBADDR6
UBADDR5
UBMSYN
SELECT.CHAOS L
E15
RESET
RESET L
S04
E13
WRITE L
WRITE
74S04
LOAD INTERVAL
LOAD INTERVAL L
BSEL
S11
F01
SELECT.CHAOS L
DELAY
50NS INCS
ID250
INPUT
50NS
100NS
150NS
200NS
250NS
CHAOS SSYN
UNIBUS.C1
S37
A13
A12
S
Q
74S74
E08
SSYN.OUT
-BSEL L
UNIBUS.C1
-READ
READ L
READ
DRIVE.UNIBUS L
;This looks useless but
;causes the initiate-transmit
;read-my-# kludge to win
CHAOS SSYN
UB+TSR
LS02
E11
B09
BOARD.SELECT FOR REAL

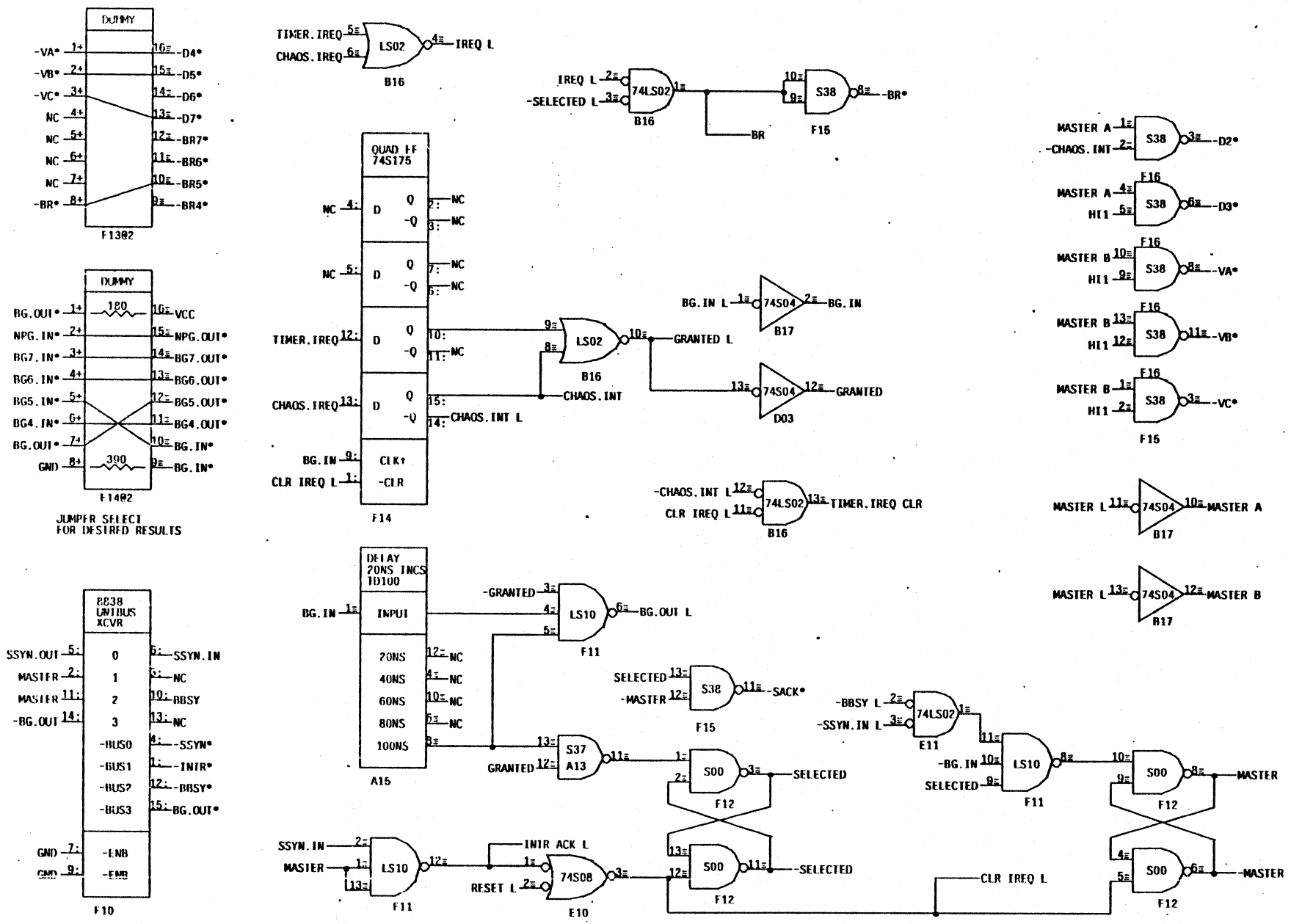

GND
INTERFERE+
INTERFERE-
GND
RCVR.DATA+
RCVR.DATA-
GND
TRANS.DATA+
TRANS.DATA-
GND
J02-1
J02-2
J02-3
J02-4
J02-5
J02-6
J02-7
J02-8
J02-9
J02-10
J02-26
J02-27
J02-28
J02-29
J02-30
J02-31
J02-32
J02-33
J02-34
J02-35
GND



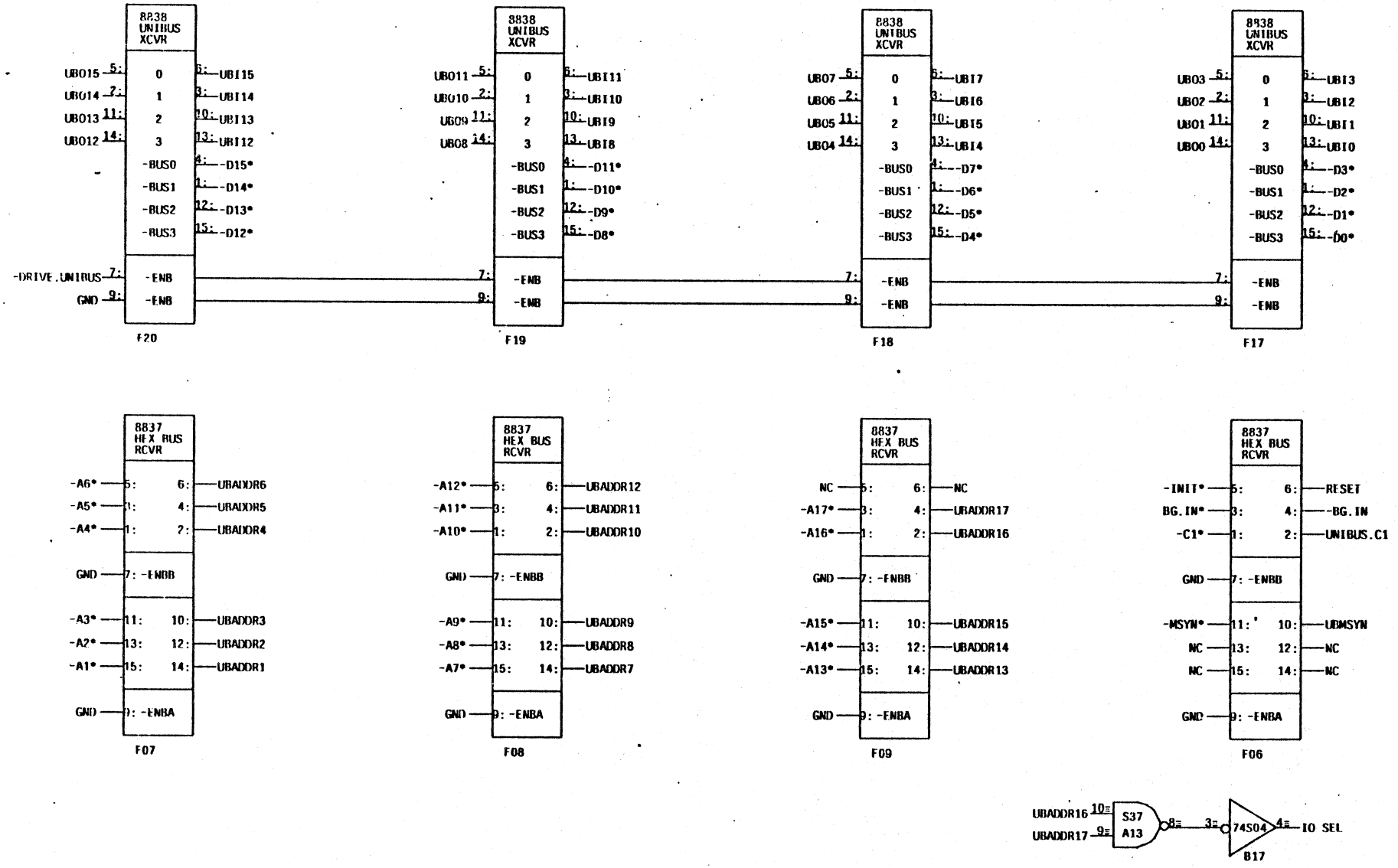
QUAD SPC BOARD	INTERVAL TIMER	17-DEC-1980 15:21	AI: CHAOS; QADTIM



QUAD SPC SLOT	BACKPLANE CONNECTIONS	4-MAR-1980 12:45	AI: CHAOS; QUBBPL



QUAD SPC BOARD	UNIBUS INTERRUPT	17-DEC-1980 15:50	AI: CHAOS; QUBINT



QUAD BOARD CHAOS NET UNIBUS TRANSCEIVER	9-FEB-1981 10:13	AI: CHAOS; QUBXCV