# THE Z80 DATAWIDGET MICROPROCESSOR DEVELOPMENT SYSTEM

Erik DeBenedictis

December 1977

## 1.0 INTRODUCTION TO THE DATAWIDGET SYSTEM

The Datawidget system is an integrated group of hardware and software designed to aid the development of microprocessor systems. The system consists of a small device to be used in the laboratory, and a remote interactive computer. The in-laboratory development aid, usually called the Datawidget, forms a complete, but limited microprocessor development system in itself. The interactive computer, and its software, extend the software development capabilities of the Datawidget by allowing interactive editing of software and the immediate testing of that software in its actual hardware environment.

The in-laboratory part of the Datawidget system consists of the actual development aid, and a terminal. The Datawidget interfaces to the CPU of the system under test, as well as to the terminal and the interactive computer.

The Datawidget is itself an especially designed microprocessor system without the microprocessor chip. When attached to the microprocessor chip of the system under test it forms a complete system. Since the Datawidget shares the CPU with the target system some interesting properties are possible. The CPU sharing is so designed that the Datawidget system is, in a sense, the higher priority system, i.e. the Datawidget can exercise complete control over the combined system. The Datawidget can, at operator command, exercise, or transfer control to, the target system.

The Datawidget also has features to aid development in stages before the Datawidget software can run on the target system's CPU.

A typical development cycle consists of using the interactive computer, through the Datawidget, to modify target system software, loading this software into the combined Datawidget-target system, and transfering control to the CPU of the target system. The performance of the target system can be monitored by observing its output, or the operator can interactively step through the software and observe the intermediate results.

The interactive computer allows the user, once he has determined a source of difficulty, to edit his software using the powerful editors available on interactive computers, and then load it back into the Datawidget with maximum efficiency.

## 2.0 THE DATAWIDGET SOFTWARE

The description of the Datawidget software presented here reflects version 4 of the Z80 Datawidget monitor. Supplements should be issued to describe differences in newer versions as these versions are produced.

### 2.1 Introduction To RAM Mapping

The Z80 Datawidget has 4K of random access memory and a virtual mapping feature which allows the RAM to be mapped to almost anywhere in the 65K memory space. The 4K of RAM is divided into sixteen blocks of 256 bytes each. The virtual mapping feature associates each of the 256 blocks in the full virtual memory space the number of one of the sixteen blocks of real RAM. When any normal location in memory space is accessed the real RAM block corresponding to this number is enabled. Since it is often desired to have no RAM mapped to a particular group of locations block fifteen has been especially modified to not enable any RAM. When the system is initialized the monitor associates block fifteen with every virtual address in the RAM mapping table.

The memory mapping allows the user to run programs from any location in the memory space even though the system has less than 4K of mappable RAM.

There are some minor restrictions, the Datawidget software runs in a PROM which is mapped between $AC00 and $AFFF, because of this any RAM which is mapped to these addresses will be overlayed by the system PROM. Also the software overlays the following locations in block zero: $00, $01, $02, $66, $67, $68. The purpose of these overlays is to allow RESET or NMI to enter the Datawidget software, these locations contain jumps to locations in the Datawidget's PROM. The Datawidget software also requires a small amount of RAM to operate, the system hardware therefore maps block fifteen of RAM to the locations between $AB00 and $ABFF. This RAM is special, however because mapped into the upper locations of this RAM are memory mapped peripherals.

The Datawidget will initialize the user's stack to the lower part of this RAM, but the user is advised not to access any of this memory above the Datawidget initialized stack.

## 2.2 RAM Mapping Commands

The Datawidget software has two keyboard commands and an automatic function to facilitate memory mapping. The most basic keyboard command is the rubout command. This command clears all of the mapping hardware. The rubout command should always be used after power-up of the system, and can be used when the existing map configuration is no longer valid. To use enter a rubout. The effect of the rubout command is to specify block fifteen as the real block associated with every virtual block. The mapping hardware, of course, interprets block fifteen as no RAM.

The most common way for the memory mapping to occur is by automatic mapping during program loading. Before any particular byte is stored during program loading, that memory address is checked to determine if it is a RAM location. If the location is determined not to be RAM (indicating that the location is overlayed, or there is no RAM responding to that address in either the target system or the Datawidget) then the RAM mapper will be called to map a block of RAM to that location. If the location cannot be mapped over then the RAM mapper will detect its failure to map to that location and will respond with an error message.

Since the assembler does not attempt to store data in the VSECT part of the program there must be a way to map RAM manually, this is accomplished by the exclam command. To use this determine which addresses are required for the VSECT, then enter the address followed by an exclam. This will map a block to include operand's address. For VSECTs requiring more than 256 bytes the command must be used once for each address with a different virtual block address.

## 2.3 Memory Manipulation

The Datawidget has four basic memory manipulation commands: the slash, return, line feed and up-arrow. To display the contents of a memory location enter the address followed by </>, the Datawidget will type its contents in hexadecimal. To change the contents first 'open' the location with a slash command, then enter the new value and type <CR>. The line feed and up-arrow commands are like the return command, except after they alter the location they open a location. The line feed command opens the next location, the up-arrow opens the preceding location.

To speed the storing of data in memory the dot command can be used. When used with an operand the dot command is the same as the line feed command, except that the operand is saved. When used without an operand the command is the same as entering the saved operand followed by a line feed.

To examine large amounts of memory the memory command is used. The memory command displays the contents of the sixteen locations following the supplied operand. To use, enter the address of the first location to be examined followed by <M>.

## 2.4 Register Manipulation

When the Datawidget monitor is running it maintains in RAM a set of pseudo registers. These registers become the real registers when execution is transfered to the users program. To display or change these registers the <R> command is used. The character <R> is entered followed by another character. If the character is not a register name then the registers are displayed. If an operand is entered and a register name is entered after the <R> then the register named is changed to the value of the operand, then registers are displayed. The registers are all changed as sixteen bit registers. The sixteen bit registers and their shortened names are:

Accumulator and Flags-A
B and C registers-B
D and E registers-D
H and L registers-H
Index register X-X
Index register Y-Y
Stack pointer-S
Program counter-P

The register names must be in upper case.

## 2.5 Execution Transfer

The commands <G>,<T> and <S> transfer execution to the users program. The <G> command simply loads the pseudo registers, including the PC. The <T> command loads the pseudo registers and also sets the single step logic in the Datawidget. The result of this command is that one instruction will be executed then the Z80 will print the registers and display the next op code to be executed. To trace the next instruction enter <T> again. The <S> command executes a subroutine at the pseudo PC, upon return the Datawidget prints the registers.

## 2.6 Loading Programs From The Interactive Computer.

To enter load mode enter a <L> from the terminal. This puts the terminal in communication with the interactive computer, all characters entered from the terminal are sent to the interactive computer, and all characters received from the interactive computer are sent to terminal. To get back to the normal Datawidget mode enter <↑Y> from the terminal. To load a file into RAM have the interactive computer send a <↑B> or <↑Y> in ASCII followed by a standard Intel format load file. Upon completion of the loading the Datawidget will return to normal mode. If an error occurs an error message will be printed on the terminal.

Many of the commands described above will have a different effect if an operand is or is not given, refer to the following detailed summary.

## 2.7 Datawidget Commands

### 2.7.1 Operands -

Operands (denoted hereafter by a ? symbol) are a string of hexadecimal characters which may precede a command.

### 2.7.2 Memory Manipulation Commands -

?</> Open memory location specified by ?.

</> Open last specified memory location.

?<CR> Store ? in open memory location.

?<LF> Store ? in open memory location, then open next memory location.

<LF> Open memory location following last specified location.

?<↑> Store ? in open memory location, then open preceding memory location.

<↑> Open memory location preceding last specified memory location.

?<.> Store ? in dot then do up arrow command.

<.> Use dot as an operand in up arrow command.

Note: A memory location may not be changed unless it was opened by the last command.

### 2.7.3 Execution Control Commands -

?<G> Execute a program at the location specified by ?.

<G> Execute a program at the same address specified by the last <G>,<S> or <T> command.

?<S> Call a subroutine at the location specified by ?.

<S> Call a subroutine at the same address specified by the last <G>,<S> or <T> command.

Note: Upon return from the <S> command a <R><R> command will be executed.

?<T> Execute one instruction at location ?.

<T> Execute one instruction at the location specified by the PC.

Note: Upon return from the <T> command a <R><R> command will be executed, followed by the contents of the memory location specified by the PC.

### 2.7.4 Register Modification Commands -

<R><R> Prints a line specifying the contents of the registers.

?<R><A> Stores ? in the registers AF, then prints the registers.

?<R><B> Stores ? in the registers BC, then prints the registers.

?<R><D> Stores ? in the registers DE, then prints the registers.

?<R><H> Stores ? in the registers HL, then prints the registers.

?<R><X> Stores ? in index register X, then prints the registers.

?<R><Y> Stores ? in index register Y, then prints the registers.

?<R><P> Stores ? in the PC, then prints the registers.

Note: If the operand is left out in a register change command, the last value used for a register change command is used.

### 2.7.5 RAM Mapping Commands -

<RUBOUT> Causes all of the mappable RAM to be deallocated.

?<!> Causes RAM to be mapped to location ?. that is it causes a block of RAM to be mapped at all locations with the high order address the same as in the operand. if no RAM is left a message is printed that says so.

### 2.7.6 Other Commands -

?<M> Prints the contents of sixteen memory locations starting with ?, the location ?+16 is then specified.

<M> Prints the contents of sixteen memory locations starting with the last specified location, the location specified is increased by 16.

<L> Datawidget enters load mode, all characters entered from either input port are echoed to the other. To exit from the terminal enter <↑Y>. the interactive computer port is watched for a <↑B> or <↑Y>, at which point a load file is expected. Upon completion of the loading normal Datawidget is reentered. If an error occurs then an error message is printed and normal Datawidget mode is reentered.

<I> Types the monitor name and version number.

### 2.7.7 Other Features -

Firm Reset Upon a reset Datawidget will reinitialize the stack, zero the PC and do a <I> command.

Gentle Reset This is a switch that asserts NMI to the Z80. It has much the same effect as firm reset, but it does not destroy the contents of any of the registers. Gentle reset is useful for locating infinite loops. It can happen that the NMI signal will be asserted for an extended period, in this case the gentle reset will be ineffective.

### 2.7.8 Literal Mode -

Literal input and output modes may not be available on all versions.

<'>? Any single character received from the terminal after a single quote will have the same effect as if two hexadecimal digits representing the ASCII received had been

entered.

<"> The double quote command toggles a switch between literal and hexadecimal typeout modes. When the system is initialized the toggle is in hexadecimal mode. In literal mode byte output is in ASCII if the character is a normal character, otherwise a blank is printed. The quote effects the </>, <LF>, <↑>, <M>, and register commands.

System subroutines. If the literal output option is used in a monitor then the names of some of the system subroutines of general interest are written into the PROM. The name is in ASCII preceding the entry point. These subroutines can be located easily by scanning through the PROM with the <M> command looking for its name.

## 3.0 SUMMARY OF ERROR MESSAGES

The Datawidget software presently has two error messages.

1. RAM MAPPING PROBLEM This error is issued by the RAM mapper. The error is issued when a maping request is made and all of the available RAM has been mapped. This error can occur either because of an exclam command, or during program loading. If during an exclam command it simply indicates that there is no RAM left to map. If during program loading it indicated either that the RAM has been exhausted, or that a request to map over an overlayed location was made.

2. BAD LOAD FILE This error occures during program loading only and indicated that either an illegal character was encountered in a load file, of that a parity error occured. Normally this error is caused by transmission errors in the link to the interactive computer, and its remedy is to repeat the load. In some versions of the monitor the error message has been reduced in length due to space limitations in the program PROM, the abbreviated error message is YECH! pointing at the offending character.

## 4.0 THE USE OF THE INTERACTIVE COMPUTER

A discussion of the use of a interactive computer in conjunction with the Datawidget will not be presented here, the reader is refered to general documentation about interactive systems or specific documentation about the

assemblers and other microprocessor related software items.

## 5.0 HARDWARE DEVELOPMENT USING THE Z80 DATAWIDGET

In order that the Datawidget have complete control over the microprocessor, a key timing signal generated by the CPU is interrupted by the Datawidget. This signal is processed by the Datawidget and sent back to the target system. Through the use of this timing signal the Datawidget can force the target system to deselect some or all of its devices in memory space. In this way the Datawidget can force the CPU to execute its software, or it can allow execution to occur in the target system.

### 5.1 Target System Interface

In the case of the Z80 Datawidget, the interface to the target system is by means of a 40 conductor clamp to the CPU and a one conductor timing signal. The clamp to the CPU interacts with some of the CPU signals, others it simply monitors, and some are ignored altogether. Specifically the data bus is both read from and actively driven, the address bus and the timing signals -RD, -WR, -MREQ, CLK, and -RFSH are monitored only. The Datawidget also drives the -RESET and -NMI lines. In the standard configuration -RESET and -NMI must be driveable by the Datawidget, i.e. not tied to 5V with a wire. The Datawidget also generates a signal called -MRQ, which is available at the front panel, the purpose of this signal is to replace -MREQ from the processor. It is suggested that, for the standard interface arrangement, the wire attached to -MREQ be removed from the CPU pin and the -MRQ line from the Datawidget be attached to that wire.

An alternate arrangement is possible. The CPU is removed from the board and a special socket with a modified Z80 CPU in it is placed into the board, the 40 conductor clamp can be attached to the Z80 CPU and the -MRQ line to a conductor provided on the socket. This socket and Z80 CPU device serves two purposes, first the -RESET, -NMI, and -MREQ lines are interrupted in the socket, allowing the Datawidget to monitor or drive these lines independently of the wiring in the target system. The socket also provides a direct connection to the target board's -MREQ pin in the CPU socket. This arrangement allows much greater convenience, but at a slight sacrifice in generality.

## 5.2 The Front Panel

The Datawidget has a front panel allowing the user to interact with the target system. As previously described the front panel contains a connector for the -MRQ line. Also present are three reset switches. One of these is a simple reset, when activated the reset line to the CPU is asserted. Another of these is called infinite reset, when this switch is turned on the reset to the CPU will be asserted for 16 out of 256 clock cycles. This is a hardware development aid which will be described later. The third reset is called gentle reset, this switch asserts the NMI signal to the CPU, causing Datawidget software to be executed. The use of this switch is more fully described in the software manual.

Also present on the front panel are two serial interface connectors with baud rate select switches. One of these gets connected to a terminal and the other to the interactive computer. The voltage levels are RS-232 and most common baud rates are provided.

An additional hardware function provided by the Datawidget is a cable with some common signals necessary for the operation of a very small microcomputer system. The signals provided include power supplies of -5V, 0V, 5V, and 12V. A clock signal appropriate for a Z80 CPU is provided; the frequency is 1.84320 MHz, a derivative of the serial interface timing. A copy of the -MRQ signal is also provided. The user is urged to use this feature with care since the power supplies of the Datawidget are not designed to handle large additional loads.

## 5.3 Initial Hardware Development

The infinite reset feature described previously is used in the very early stages of the system development. When a system is initially constructed it is possible that shorts will exist between key signals or to ground. It is also possible that the -MRQ line will be ineffective in actually deselecting the memory space, because of improper gating of the tri-state buffers on the various device. When the infinite reset is on the user can use a scope or logic analyzer to locate this type of problem. The infinite reset causes the waveforms generated to be periodic, and the reset line can be used as a sync to the test instrument.

All of the interface signals to the CPU are buffered by five chips. The data bus is buffered by two type 8833 bi-directional bus drivers. The other signals, address bus and control are buffered by three 74S373 octal latches. If the connectors are subjected to unusual voltages or loads, such as would occur if the clamp were put on backwards, then

there is a possibility that the buffers will require replacement.

## 5.4 Later Hardware Development

Once the target system is developed to the point that the Datawidget itself can function, then the user can use the Datawidget, through the terminal, to aid his development. By using the memory manipulation commands an operator can usually get a good idea of what his system is doing in a very short time. An example of using the Datawidget to find why a device doesn't work is writing a loop in Datawidget RAM which accesses a device which is believed not to function. A scope can then be used to examine the various chip selects and other signals.

## 5.5 Virtual Memory

The Datawidget is basicly a memory space with no processor. Some of the devices in the memory space are alterable or movable, others are not. There are several basic different devices, each with a different priority, i.e. if several devices exist at a certain memory address then only the highest priority device will be selected. At the highest priority level is the main program memory ROM and locations called overlays. The ROM is located at $AC00-$AFFF and the overlays at locations $0-$2 and $66-$68. These locations are read only and contain software. The next priority level has the devices and the triggers, the devices are the eight registers of the two communications interfaces. The triggers are locations which are monitored by the hardware, and a flag is set when they are accessed. The triggers are transparent, i.e. the RAM underneath them can be seen. Both the devices and the triggers are located in the upper part of the stack RAM. Next in priority is the mappable RAM. The Datawidget contains hardware which can map one of sixteen 256 byte blocks of RAM to any block address in the memory space. If RAM has not been mapped to a particular location in memory space then the next lower priority device will be seen below it. There is a 256 byte block in memory space which is special, this space is reserved for the stack RAM. The Datawidget monitor must have a small amount of permanently mapped RAM in order to function. Its location is $AB00-$ABFF. The next lower, and lowest, priority is the devices in the target system. The -MRQ line from the Datawidget will be asserted in this case, and whatever the target system responds with will be accepted. Also all refresh cycles are transmitted to the target system, regardless of the address.

## 5.6 Speed Considerations

The Datawidget was designed to perform the above described functions at the greatest simplicity. To achieve this, performance at the highest speeds may have been sacrificed. The most severe speed related problem is the delay from address to -MRQ, this delay is guaranteed only to be less than 120 ns. In some systems, at high speeds, there may be glitching of this line if the CPU issues -MREQ less than 120 ns after the address becomes stable. If a system would be sensitive to this glitching then it should be operated at a lower speed. Future upgrades in parts may reduce the 120 ns figure.

## APPENDIX 1 SETTING UP THE Z80 DATAWIDGET

1. Connect a terminal to the upper "D" type connector. Select the baud rate with the rotary switch. The terminal baud rate must not be less than the interactive computer baud rate.

2. Connect the line to the interactive computer to the lower "D" type connector. Select the baud rate with the rotary switch.

3. Connect the datawidget to the dummy Z80 system.

4. Make sure the infinite reset switch is off.

5. Plug in the power cord.

6. Push the reset switch on the front panel.

7. The datawidget software should type a herald when the reset switch is released. Pressing an <I> on the terminal will also type the herald.

# DIFFERENCES BETWEEN THE Z80 AND 8080 DATAWIDGETS

Erik DeBenedictis
December 1977

## 1.0 PRIMARY DIFFERENCES

The biggest difference between the 8080 and Z80 Datawidgets, other than the obvious difference in the CPUs, is in the area of memory space utilization. The 8080 Datawidget requires a 4K kernel, located starting at location zero, in order to run. This kernel contains the systems PROM, RAM and IO devices. The user mappable RAM is mappable only in 1K blocks, and it is mappable by switches on the front panel, rather than by software commands. The interface with the interactive computer is by hardware switching of signals instead of by software.

### 1.1 Memory Utilization And Mapping

The Datawidget hardware maps the system kernel into the low addresses, at locations $0000 to $03FF the system's 1K PROM is mapped. In the next K of memory space there is 256 bytes of RAM wrapped around four times, addresses $0800 to $0BFF contain the special single step hardware. The fourth K of memory space contains the systems IO port.

The 8080 Datawidget contains 3K of mappable RAM. This RAM is mapped by the actual switching of the chip selects. The switching occurs in two stages, in each stage three bits of the address are compared to the setting of the front panel switches. If a match is made the signal is passed down to the next lower stage or the actual RAM. The first stage compares the highest three bits of the address to the setting on the single HI RAM SELECT switch. A match at this stage is passed to the three second stages, the second stage compares the next highest three bits. Since there is only one high order select switch the user is constrained to have all of his mappable RAM in one 8K block. The user must enter the high order six bits of his RAM addresses into the HI RAM SELECT and LOW RAM SELECT switches in octal.

### 1.2 Interface To The Interactive Computer

The interface to the interactive computer is by the switching of the serial data signals between the terminal, Datawidget and interactive computer. There are two modes that the user is interested in, in one mode he is in communication with the interactive computer, but the Datawidget is listening to the output of the interactive

computer for a load file. The other mode is when the user is interacting only with the Datawidget and the interactive computer is inactive. To get into the first mode the Datawidget must be put into load mode and a switch on the front panel must be put into the PDP-10 position. In this mode the output of the terminal is sent to the Datawidget. The Datawidget listens to this, but does not transmit anything. The terminal output is sent to the interactive computer, and the line from the interactive computer is sent to the terminal. When the switch is in the other, or TERMINAL, position the terminal output goes to the Datawidget only.

Unlike the Z80 Datawidget, there can be no baud rate conversion, so the terminal baud rate must be the same as the line rate to the interactive computer.

## 1.3 Differences In The Monitors

The description of the software differences presented here reflects version 3.1 of the 8080 Datawidget monitor and version 4 of the Z80 Datawidget monitor. No further modifications of the 8080 software are planned.

The Z80 and 8080 Datawidget monitors are essentially the same, however the extended instruction set of the Z80 has allowed a more powerful monitor to be placed in the same size PROM. The 8080 monitor can be viewed as a subset of the Z80 monitor. The major differences are summerized in the following table:

1. The 8080 Datawidget does not support the dot command.

2. The register display and modification commands are different. To display the registers enter <R>. To change a register enter the new value followed by control-<register name>.

3. The RAM mapping commands are absent.

4. The load command simply puts the Datawidget into a state where it listens for a control-B followed by a load file. No echoing occurs in this state. The state is terminated when the end of the load file is received, or a control-C is received before the initial control-B.

5. The literal mode is never implemented.

6. The 8080 Datawidget accepts the characters control-C and space, and does nothing important with them.

7. The 8080 Datawidget will accept a RST 5 instruction when executing as an entry into the monitor. Upon execution of this instruction the Datawidget will print a dot and wait.

8. When the RST instructions are executed, except RST 0 and RSR 5, the effect will be to call the same subroutine except at an address $8000 greater than normal.

## 1.4 Differences In Hardware Development Techniques

The internal structures of the Z80 and 8080 Datawidgets vary significantly due to the differences in the microprocessors. The interface concepts are identical in the two systems, although the details are very different in some cases. The only variance in the designs is with the infinite resets, the 8080 Datawidget asserts the infinite reset with a period independent of the processor clock rate.

## APPENDIX 1 SETTING UP THE 8080 DATAWIDGET

1. Connect a terminal to the male "D" type connector. Select the baud rate to be used for all of the devices on both the Datawidget and the terminal.

2. Connect the line to the interactive computer to the female "D" type connector.

3. Connect the cable to the dummy 8080 system found under the front panel.

4. Make sure the infinite reset switch is turned off.

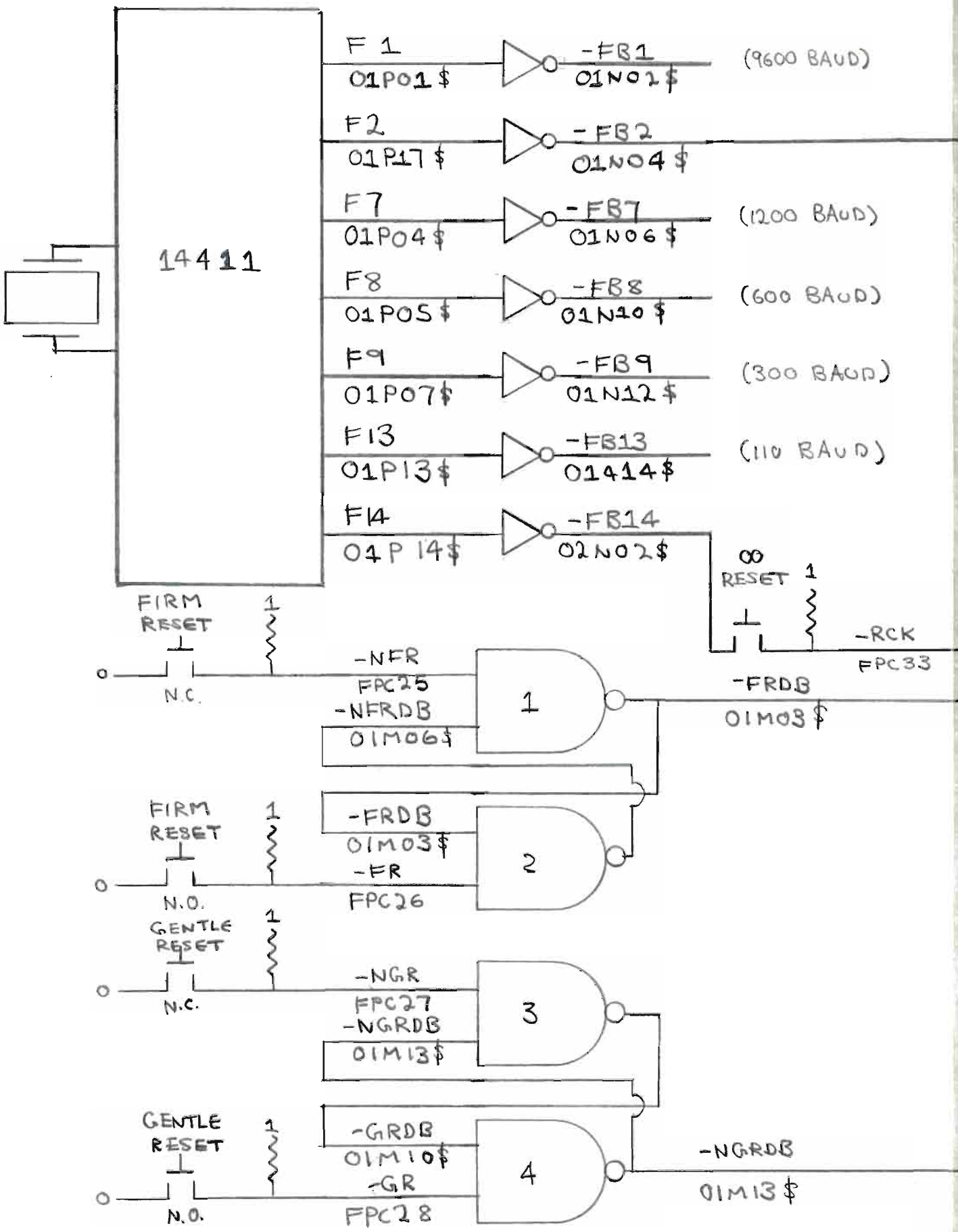
5. Move the switch labeled PDP-10 and TERMINAL to the TERMINAL position.

6. Plug in the power cord.

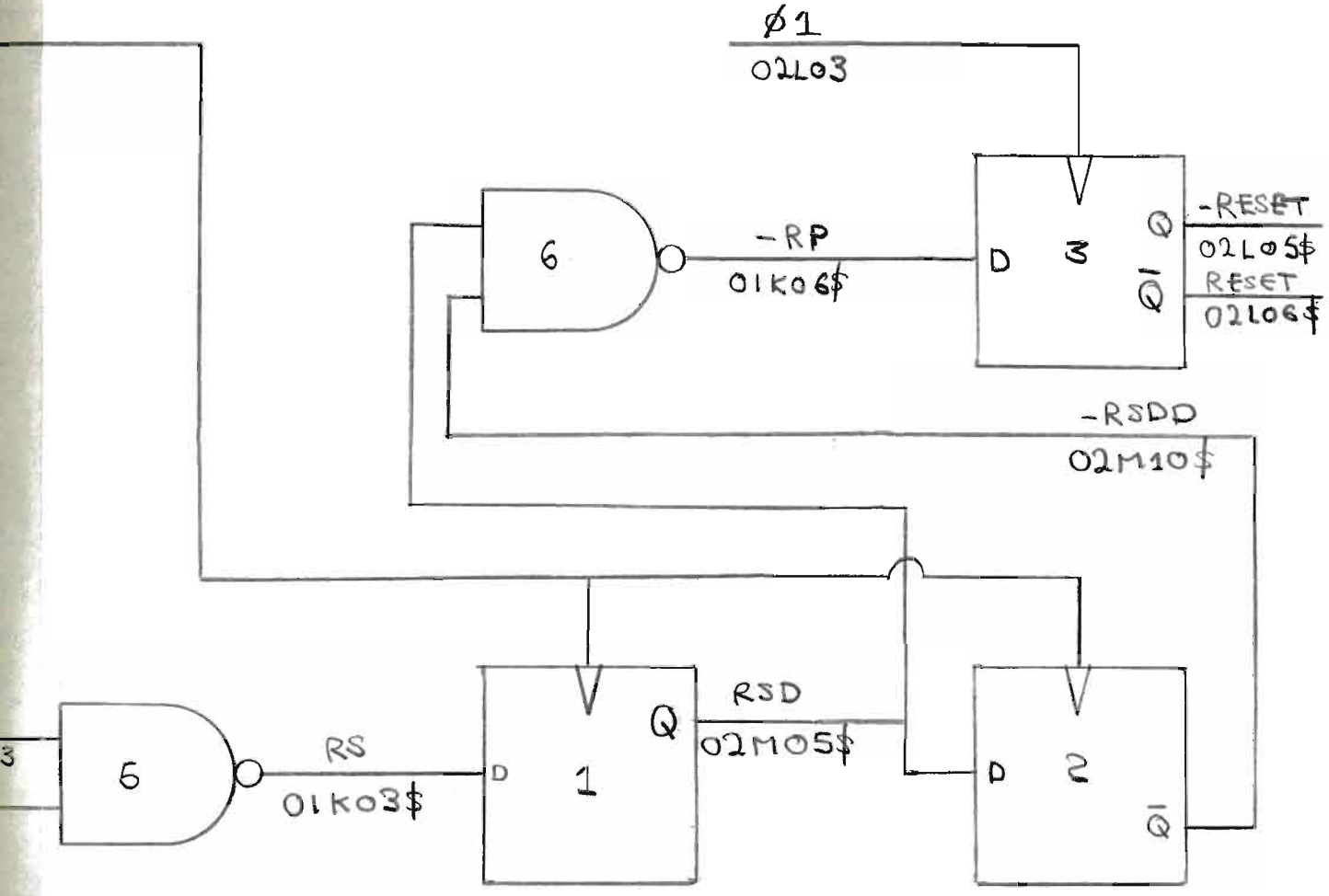
7. Push the reset switch on the front panel.

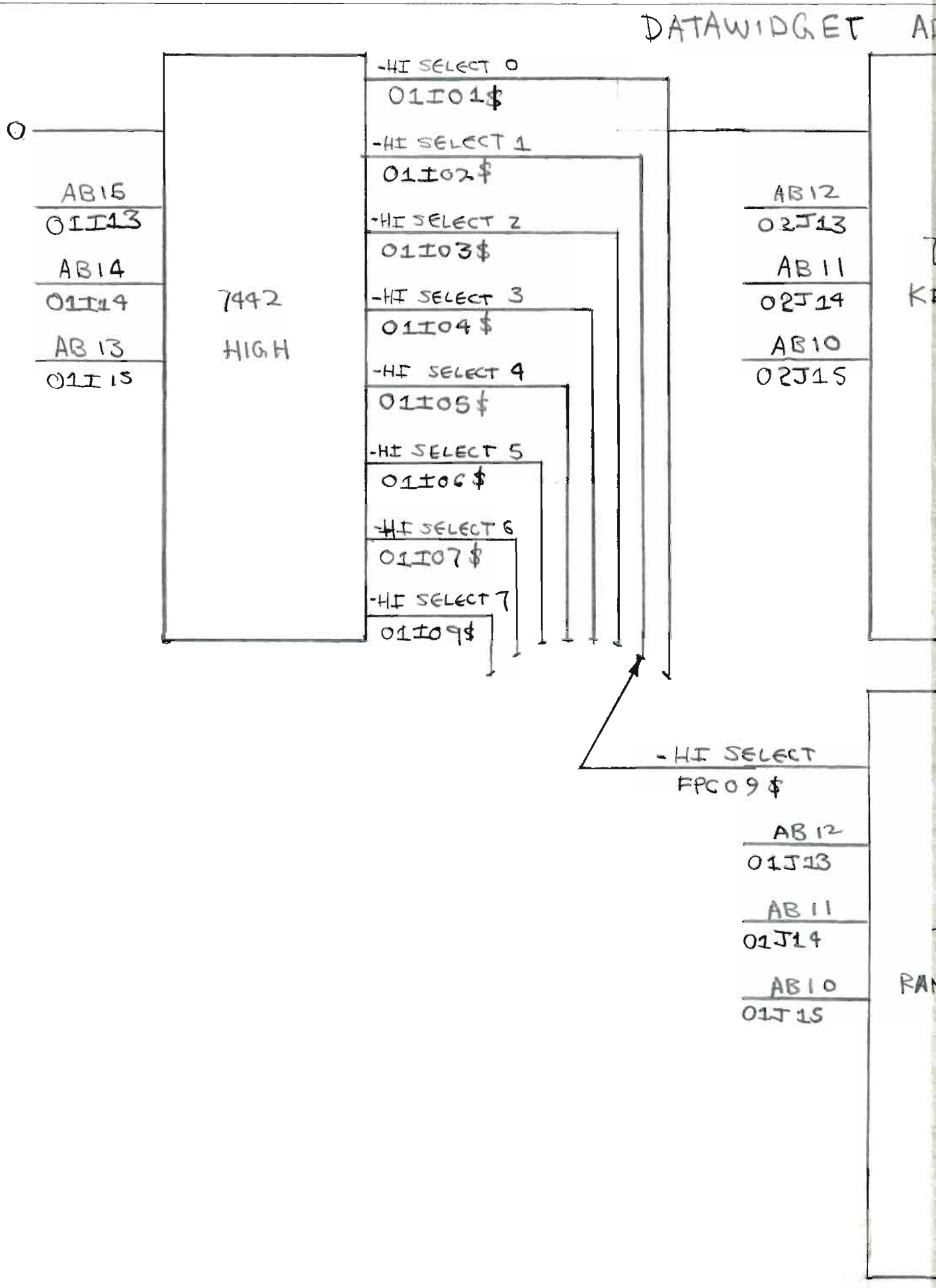
8. The Datawidget should type a herald when the reset switch is released. Pressing an <I> on the terminal will also type the herald.

DATAWIDGET

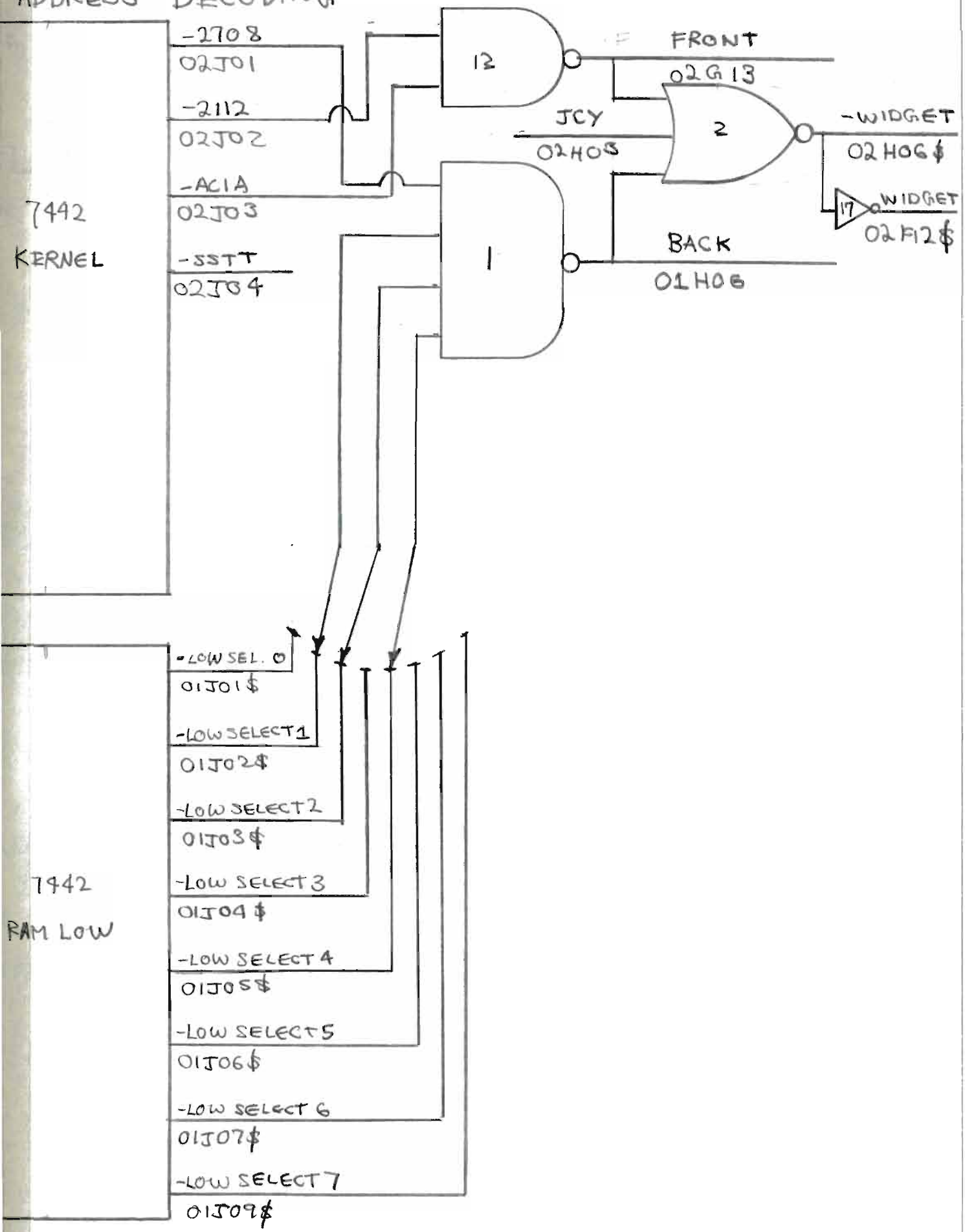

14411
F 1
01P01$
-FB1
01N02$
(9600 BAUD)
F2
01P17$
-FB2
01N04$
F7
01P04$
-FB7
01N06$
(1200 BAUD)
F8
01P05$
-FB8
01N10$
(600 BAUD)
F9
01P07$
-FB9
01N12$
(300 BAUD)
F13
01P13$
-FB13
01414$
(110 BAUD)
F14
01P14$
-FB14
02N02$
∞
RESET
1
-RCK
FPC33
FIRM
RESET
1
0
N.C.
-NFR
FPC25
-NFRDB
01M06$
1
-FRDB
01M03$
FIRM
RESET
1
0
N.O.
-FRDB
01M03$
-FR
FPC26
2
GENTLE
RESET
1
0
N.C.
-NGR
FPC27
-NGRDB
01M13$
3
GENTLE
RESET
1
0
N.O.
-GRDB
01M10$
-GR
FPC28
4
-NGRDB
01M13$

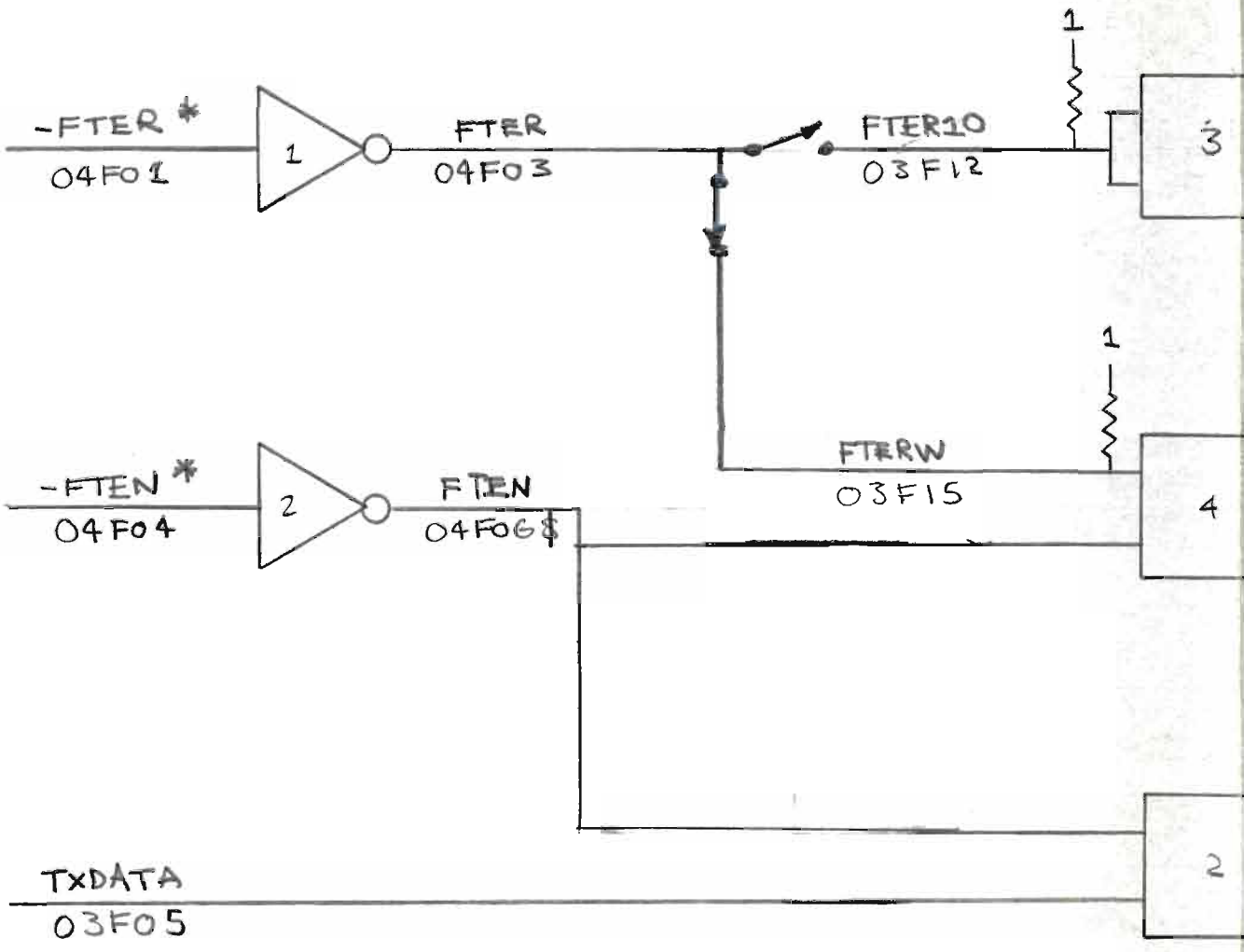
T ACCESSORIES




DATAWIDGET
0
AB15
01I13
AB14
01I14
AB13
01I15
7442
HIGH
-HI SELECT 0
01I01$
-HI SELECT 1
01I02$
-HI SELECT 2
01I03$
-HI SELECT 3
01I04$
-HI SELECT 4
01I05$
-HI SELECT 5
01I06$
-HI SELECT 6
01I07$
-HI SELECT 7
01I09$
AB12
02J13
AB11
02J14
AB10
02J15
-HI SELECT
FPC09$
AB12
01J13
AB11
01J14
AB10
01J15

# ADDRESS DECODING



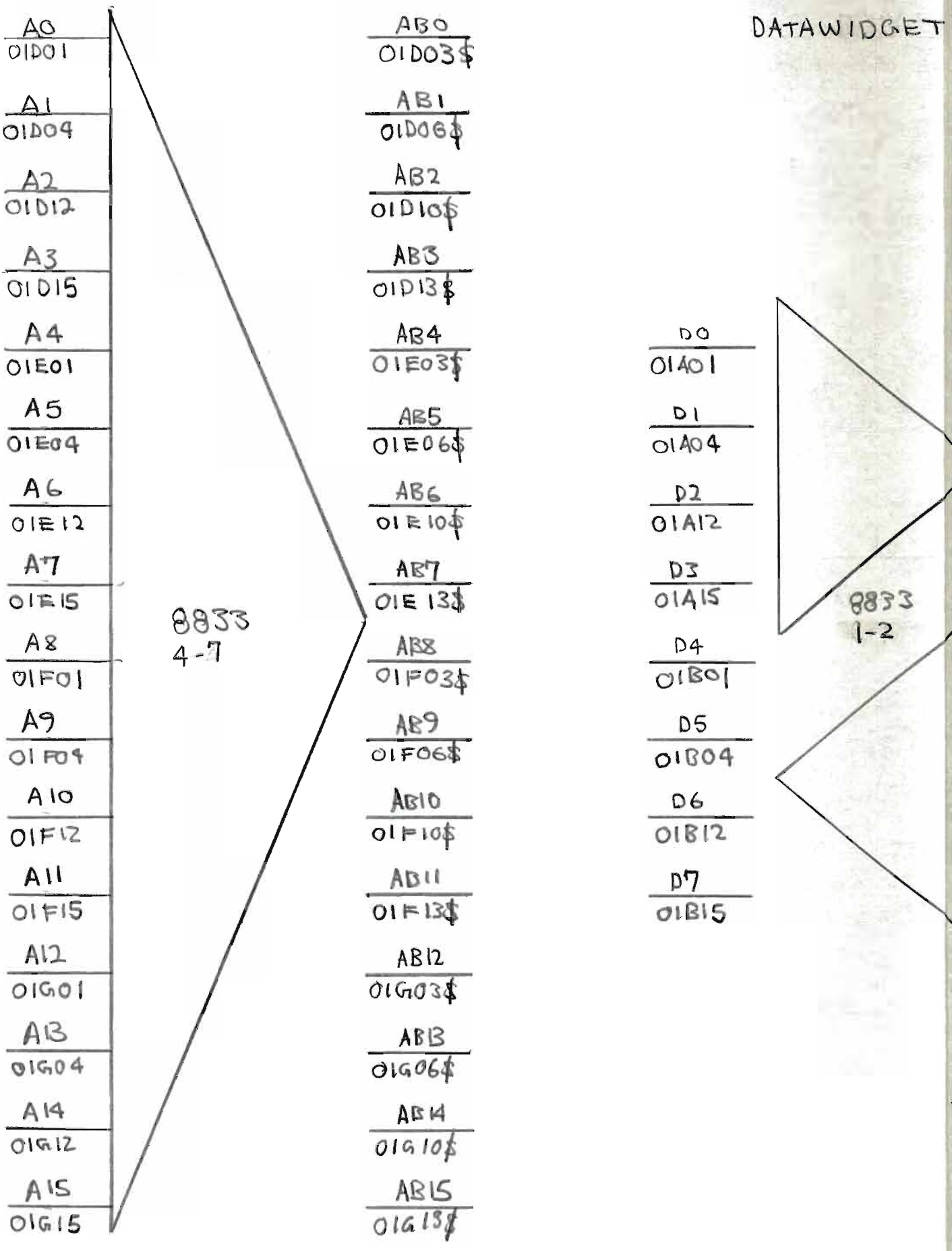



* INDICATES EIA VOLTAGE LEVELS (-5,12 V.)

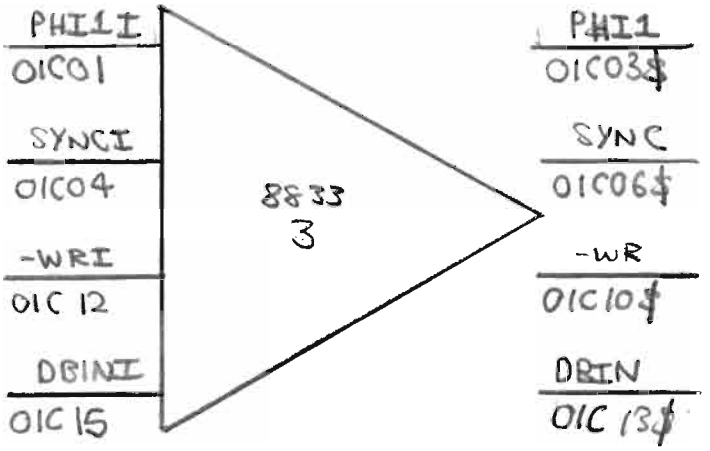
GET EIA DRIVERS

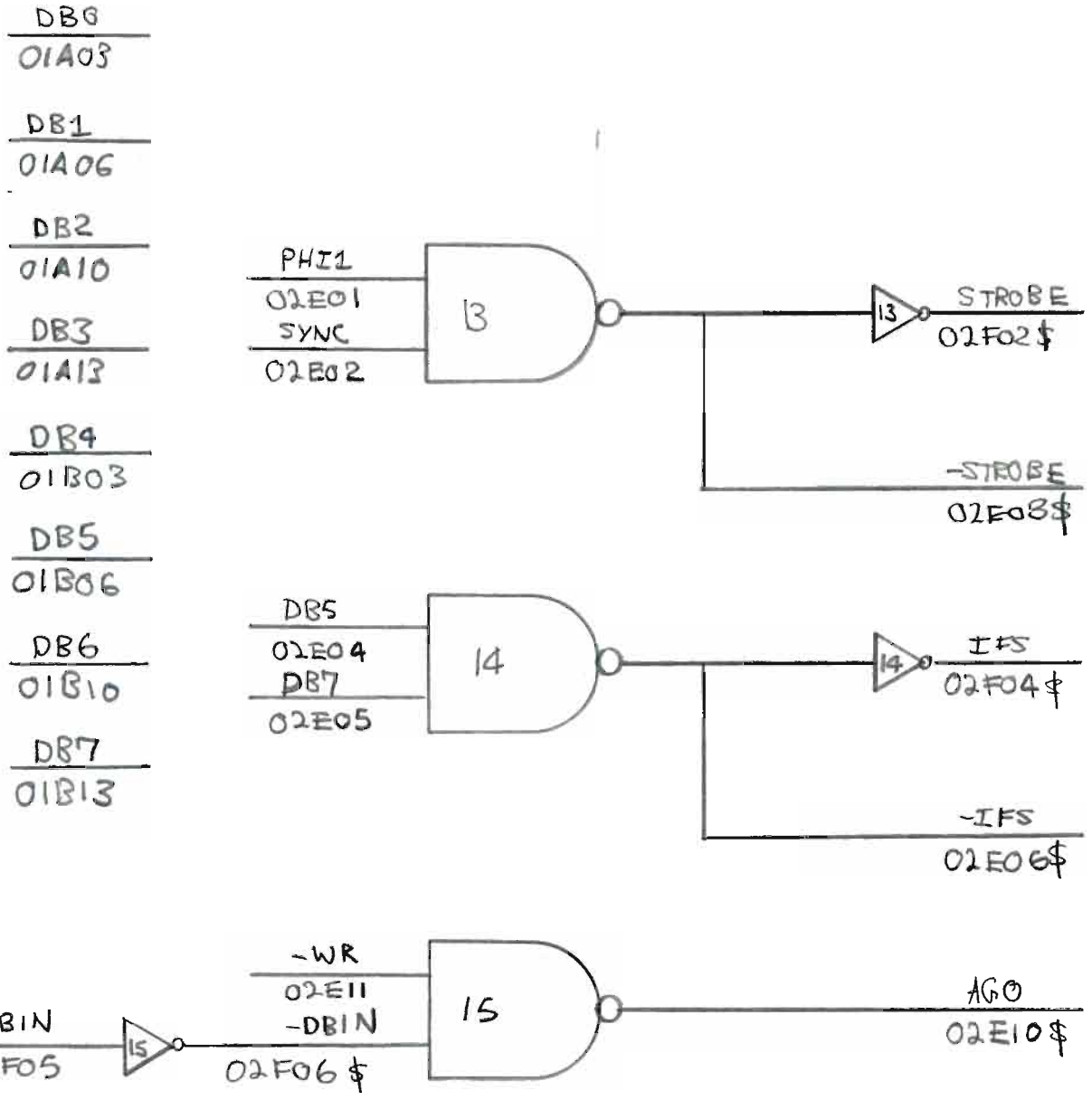


DATAWIDGET

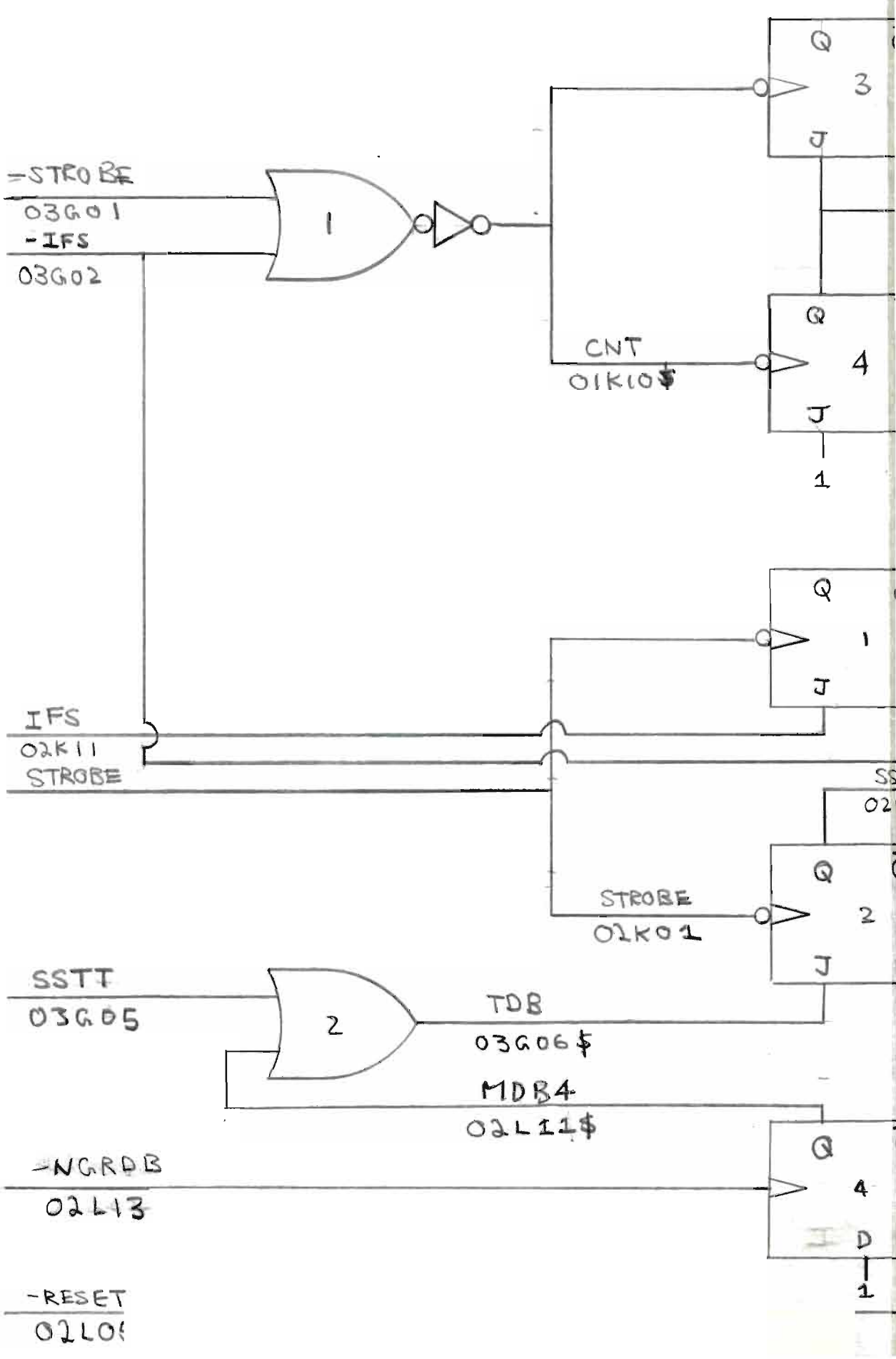

A0
01D01
A1
01D04
A2
01D12
A3
01D15
A4
01E01
A5
01E04
A6
01E12
A7
01E15
A8
01F01
A9
01F04
A10
01F12
A11
01F15
A12
01G01
A13
01G04
A14
01G12
A15
01G15
8833
4-7
AB0
01D03$
AB1
01D06$
AB2
01D10$
AB3
01D13$
AB4
01E03$
AB5
01E06$
AB6
01E10$
AB7
01E13$
AB8
01F03$
AB9
01F06$
AB10
01F10$
AB11
01F13$
AB12
01G03$
AB13
01G06$
AB14
01G10$
AB15
01G13$
D0
01A01
D1
01A04
D2
01A12
D3
01A15
D4
01B01
D5
01B04
D6
01B12
D7
01B15
8833
1-2

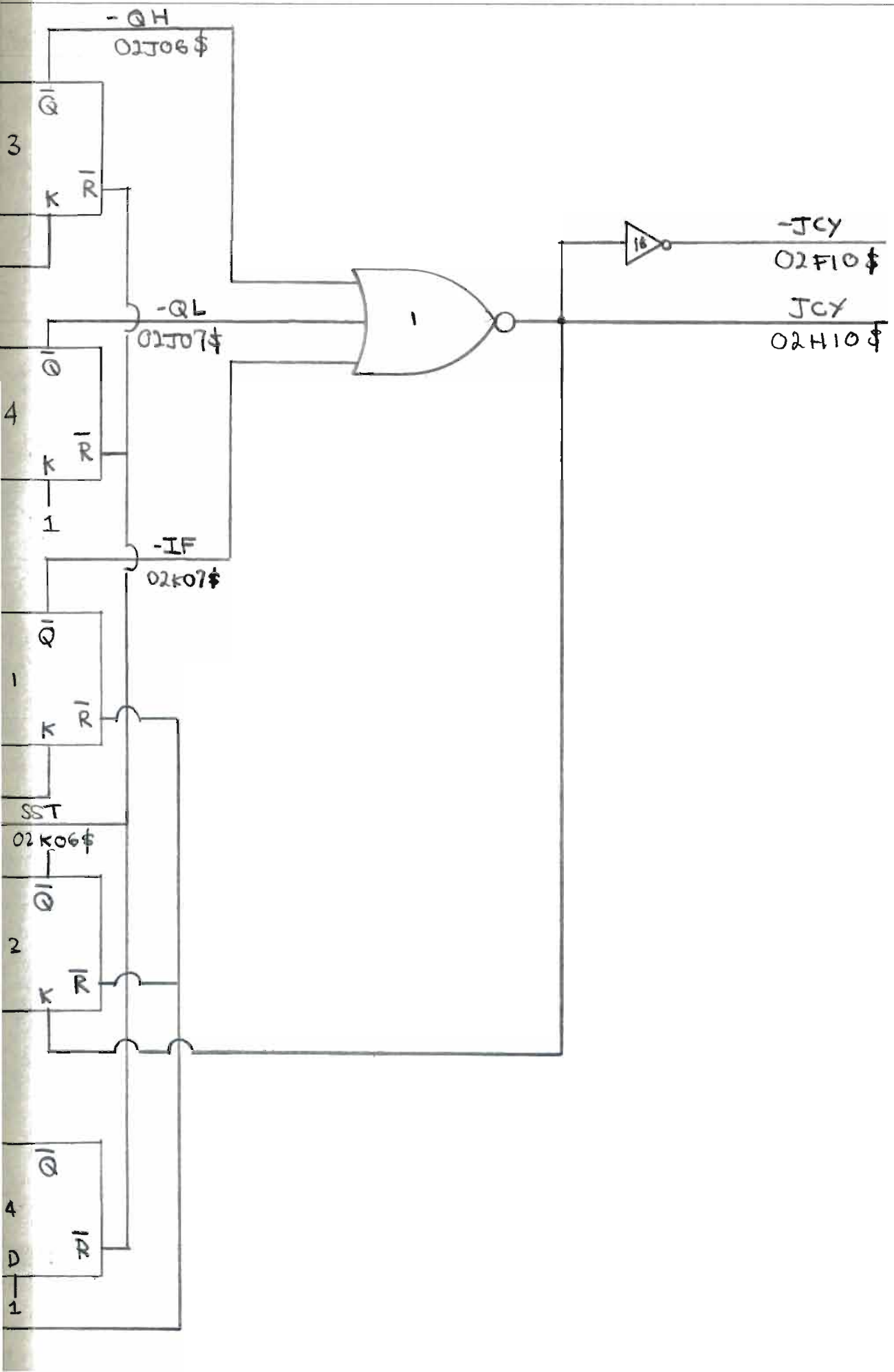
FRONT END

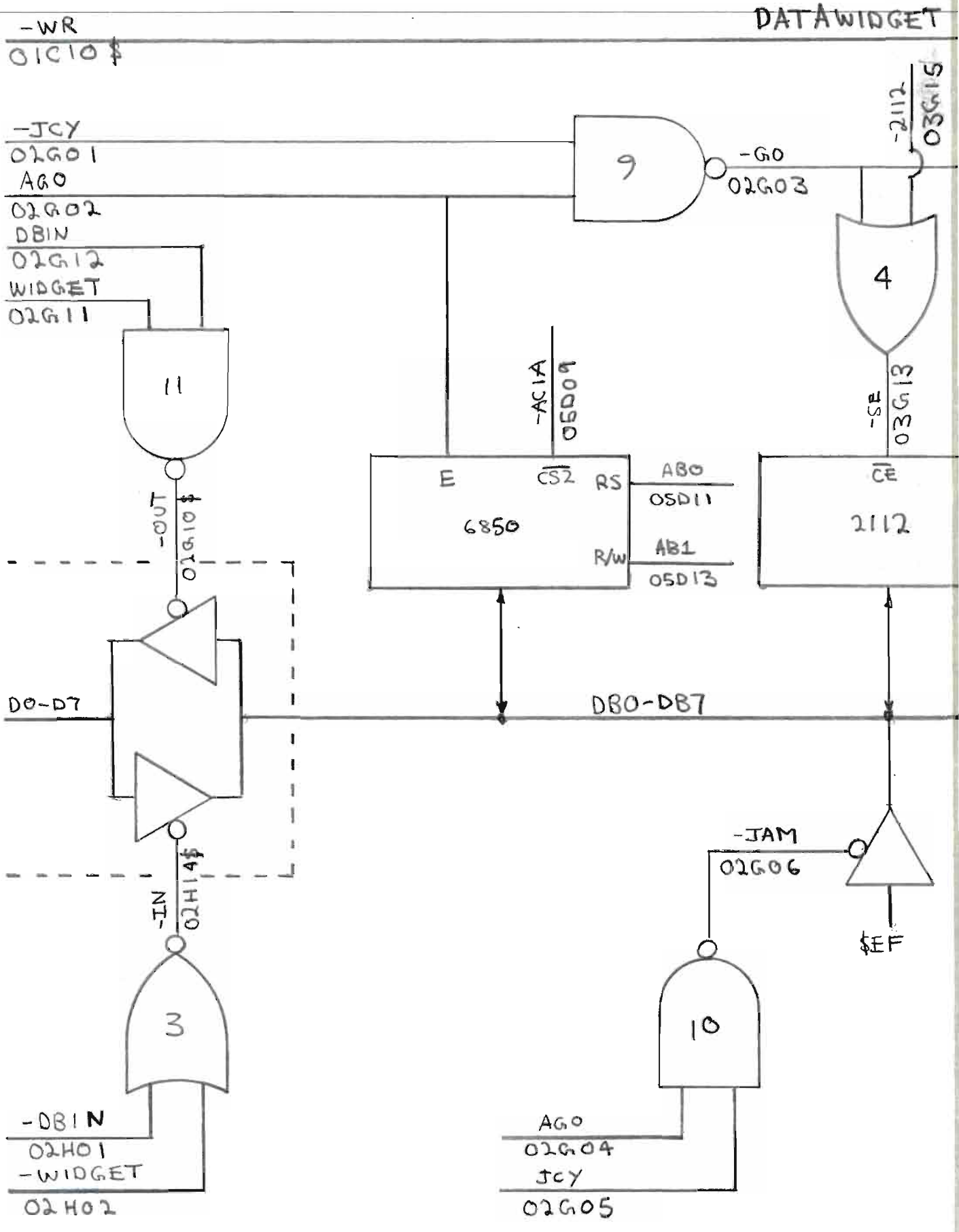

PHI1I
01C01
SYNCI
01C04
-WRI
01C12
DBINI
01C15
8833
3
PHI1
01C03$
SYNC
01C06$
-WR
01C10$
DBIN
01C13$


DB0
01A03
DB1
01A06
DB2
01A10
DB3
01A13
DB4
01B03
DB5
01B06
DB6
01B10
DB7
01B13
PHI1
02E01
SYNC
02E02
13
13
STROBE
02F02$
-STROBE
02E08$
DB5
02E04
DB7
02E05
14
14
IFS
02F04$
-IFS
02E06$
DBIN
02F05
15
-WR
02E11
-DBIN
02F06$
15
AGO
02E10$

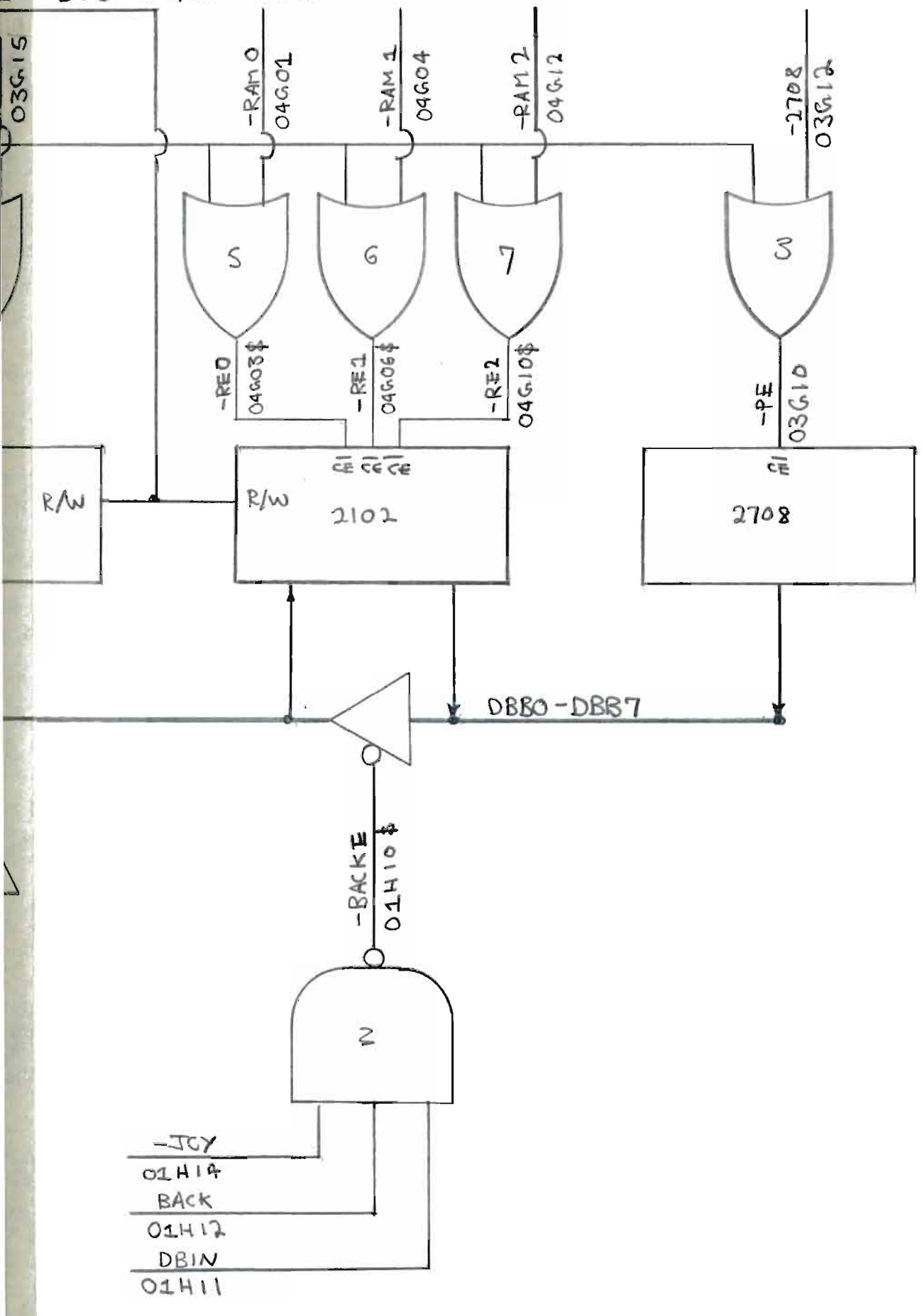
# DATAWIDGET SINGLE STEP



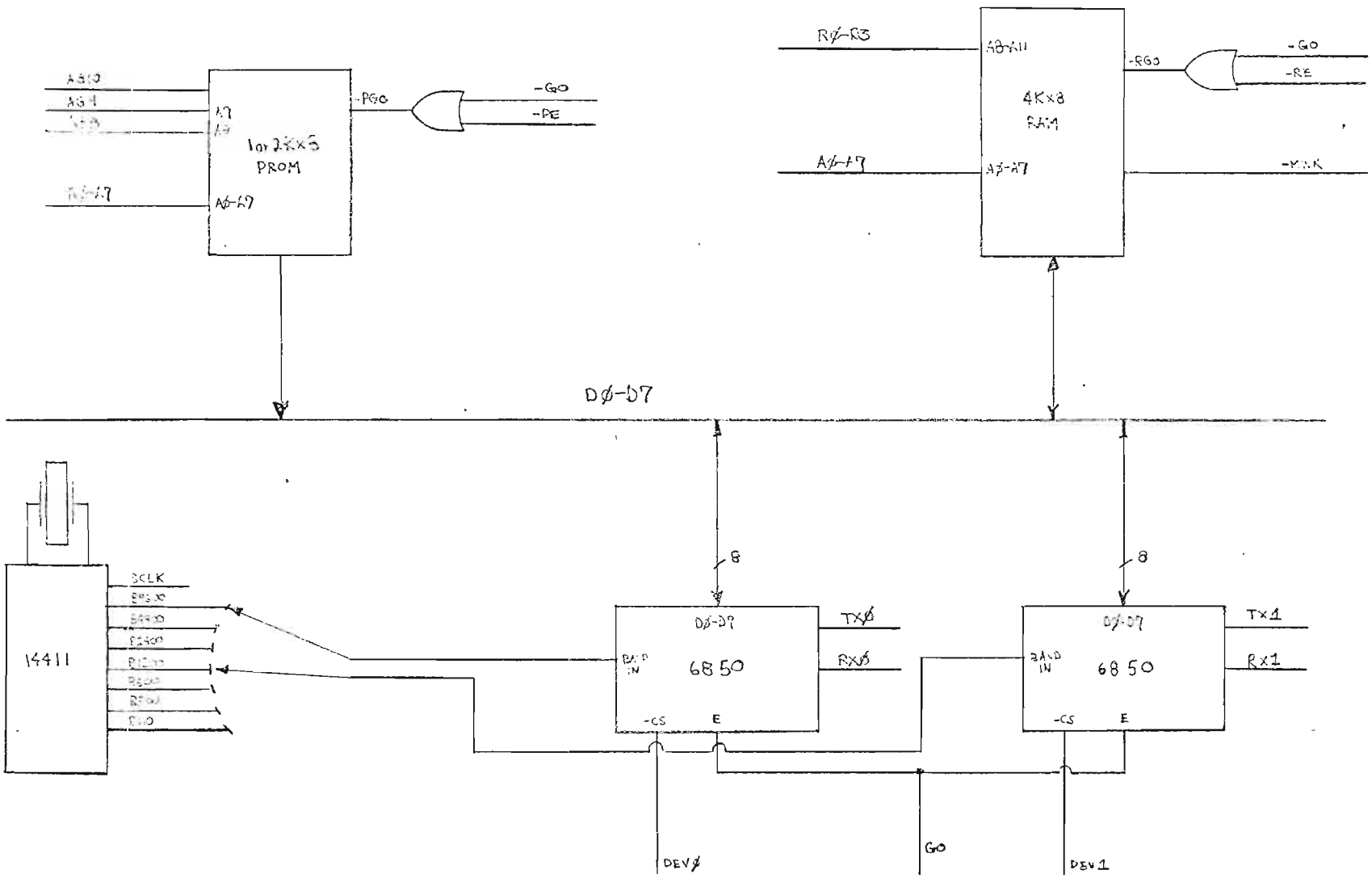

-QH
02J06$
-QL
02J07$
-IF
02K07$
SST
02K06$
-JCY
02F10$
JCY
02H10$
16
1
3
4
1
2
4
D
1


-WR
01C10$
-JCY
02G01
AG0
02G02
DBIN
02G12
WIDGET
02G11
9
-GO
02G03
-2112
03G15
4
11
-ACIA
05D09
-SE
03G13
E
CS2
RS
AB0
05D11
6850
R/W
AB1
05D13
CE
2112
-OUT
02G10$
D0-D7
DB0-DB7
-JAM
02G06
$EF
-IN
02H14$
3
10
-DBIN
02H01
-WIDGET
02H02
AG0
02G04
JCY
02G05

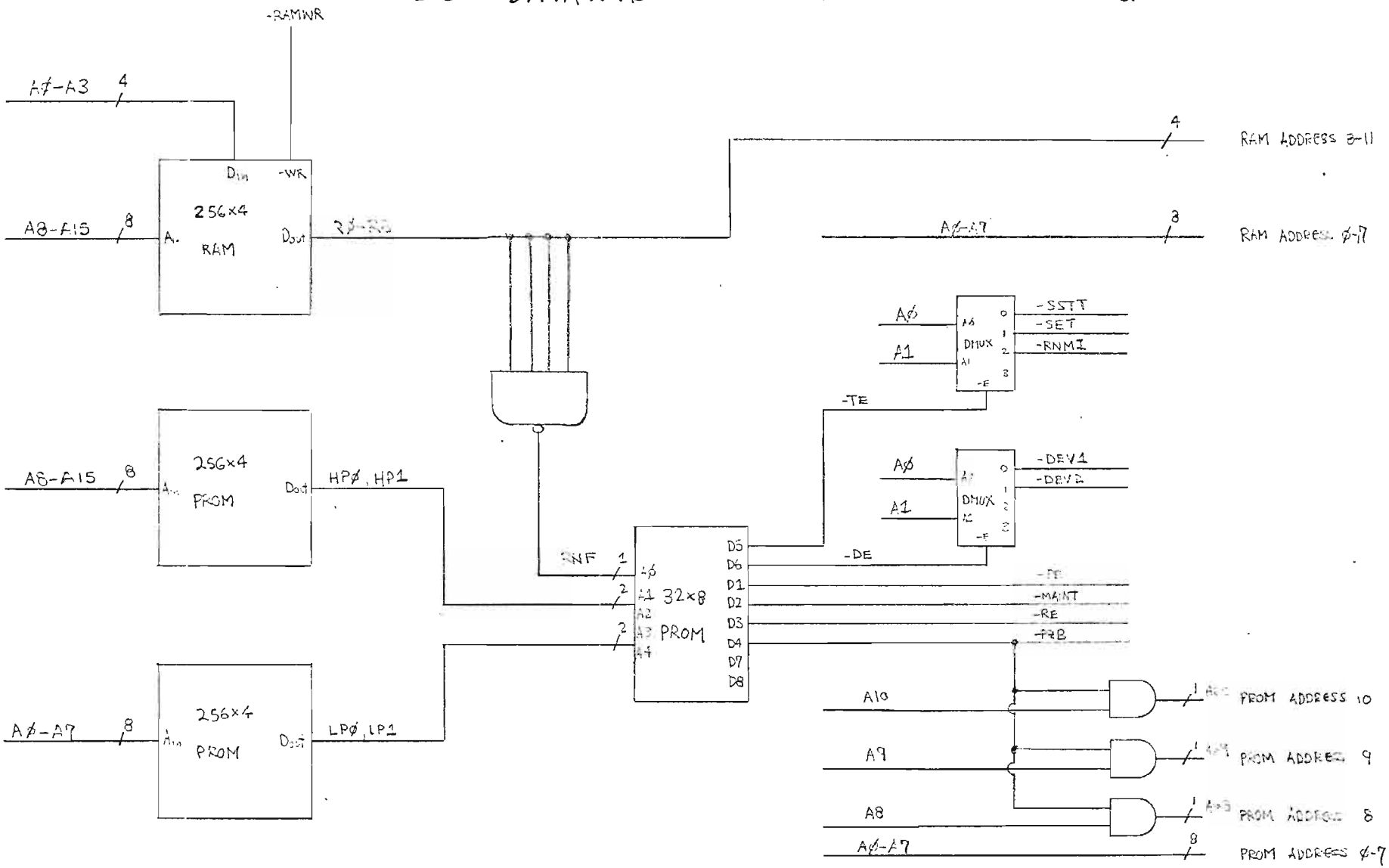
ET BUS STRUCTURE



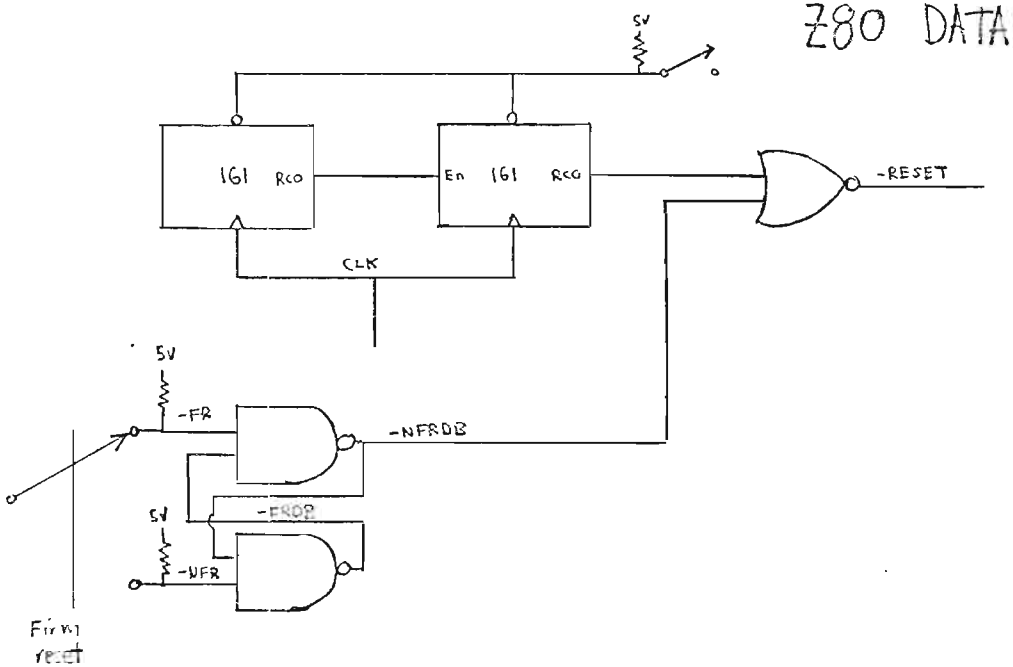
# Z80 DATAWIDGET INTERNAL BUS

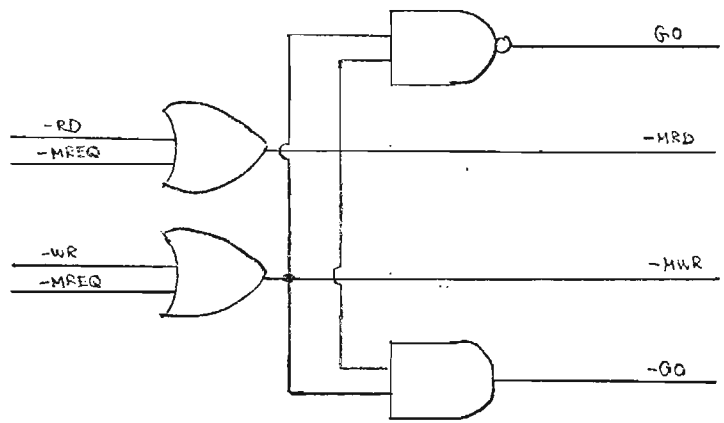


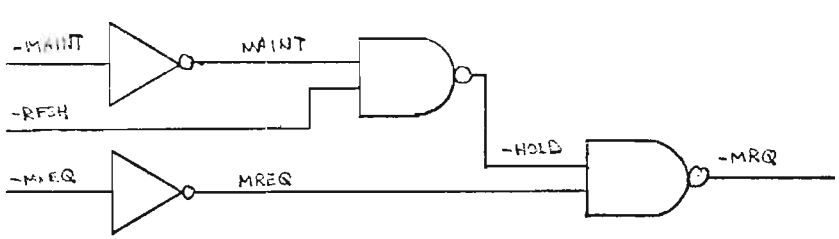
# Z80 DATAWIDGET ADDRESS DECODING



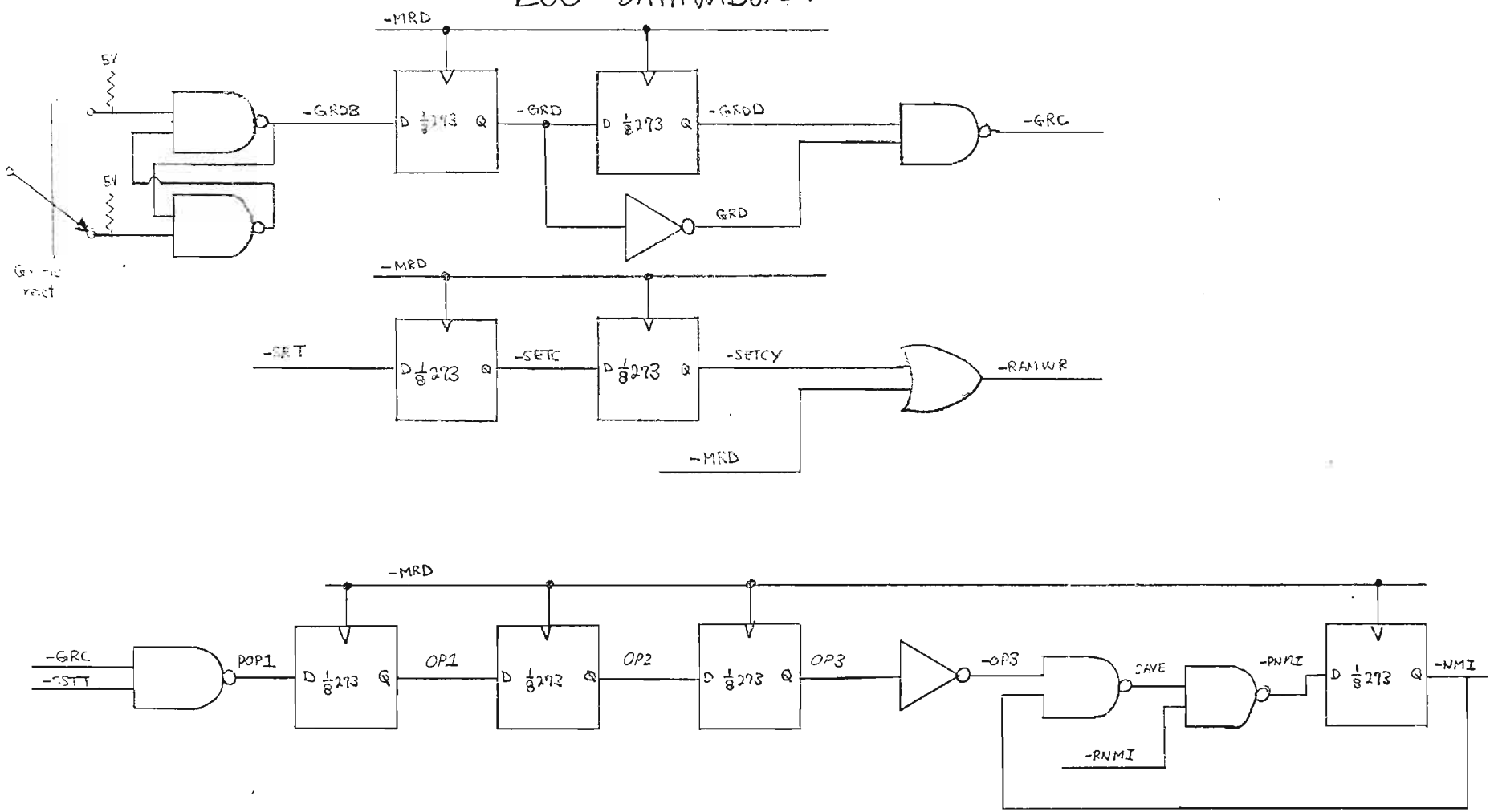
# Z80 DATAWIDGET



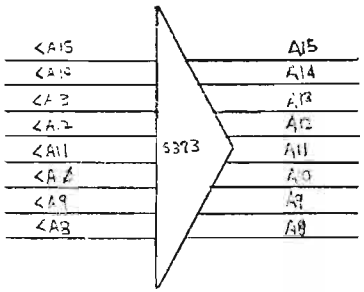



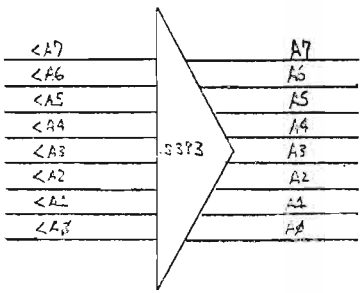



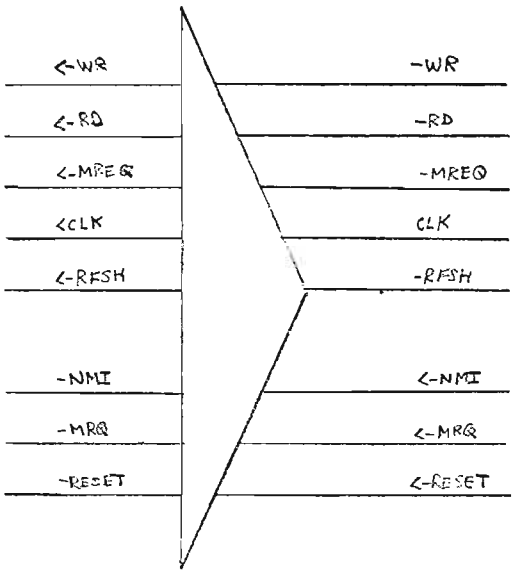
# Z80 DATAWIDGET

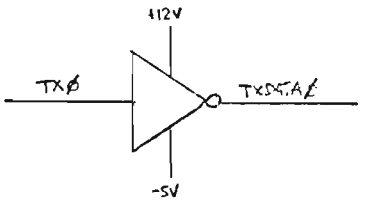


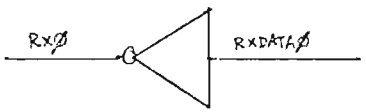
# Z80 DATAWIDGET BUFFERS

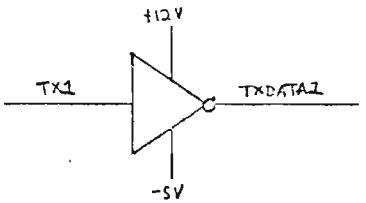


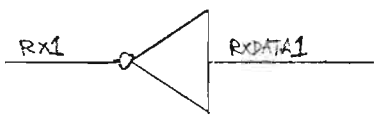



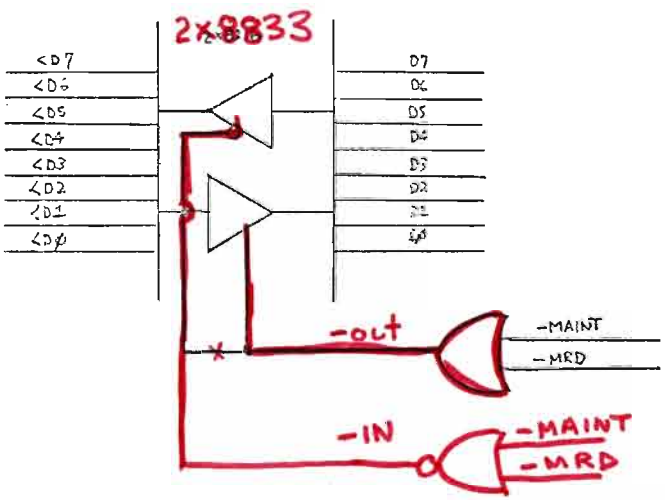



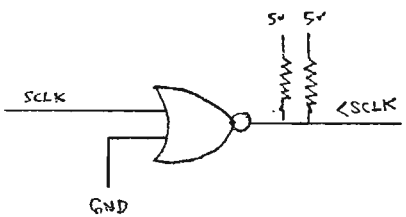



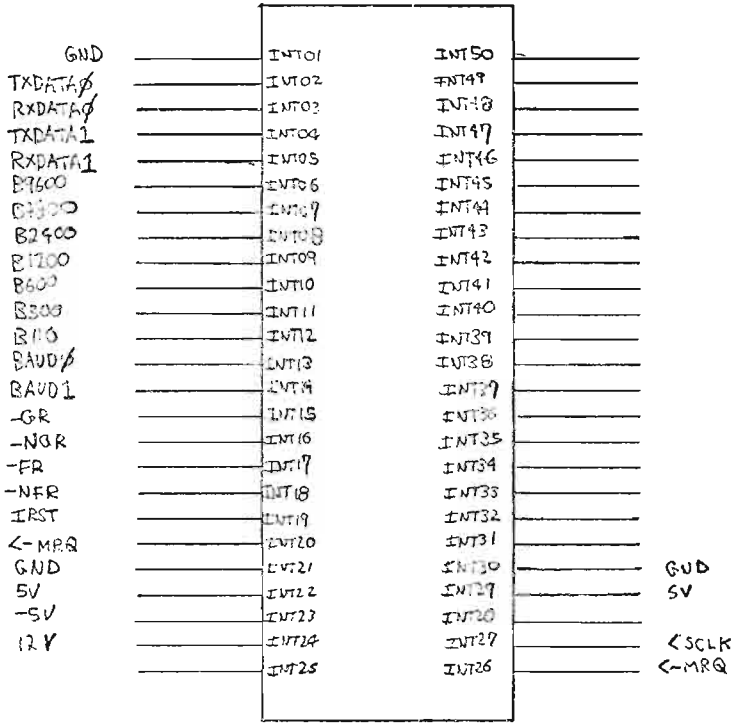



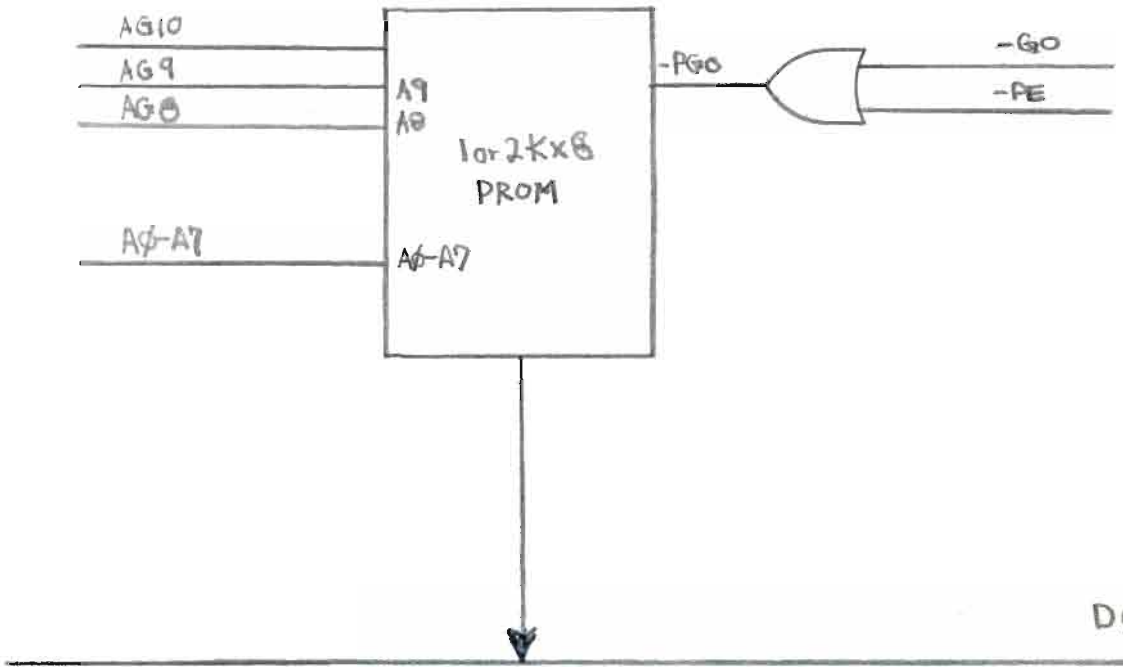



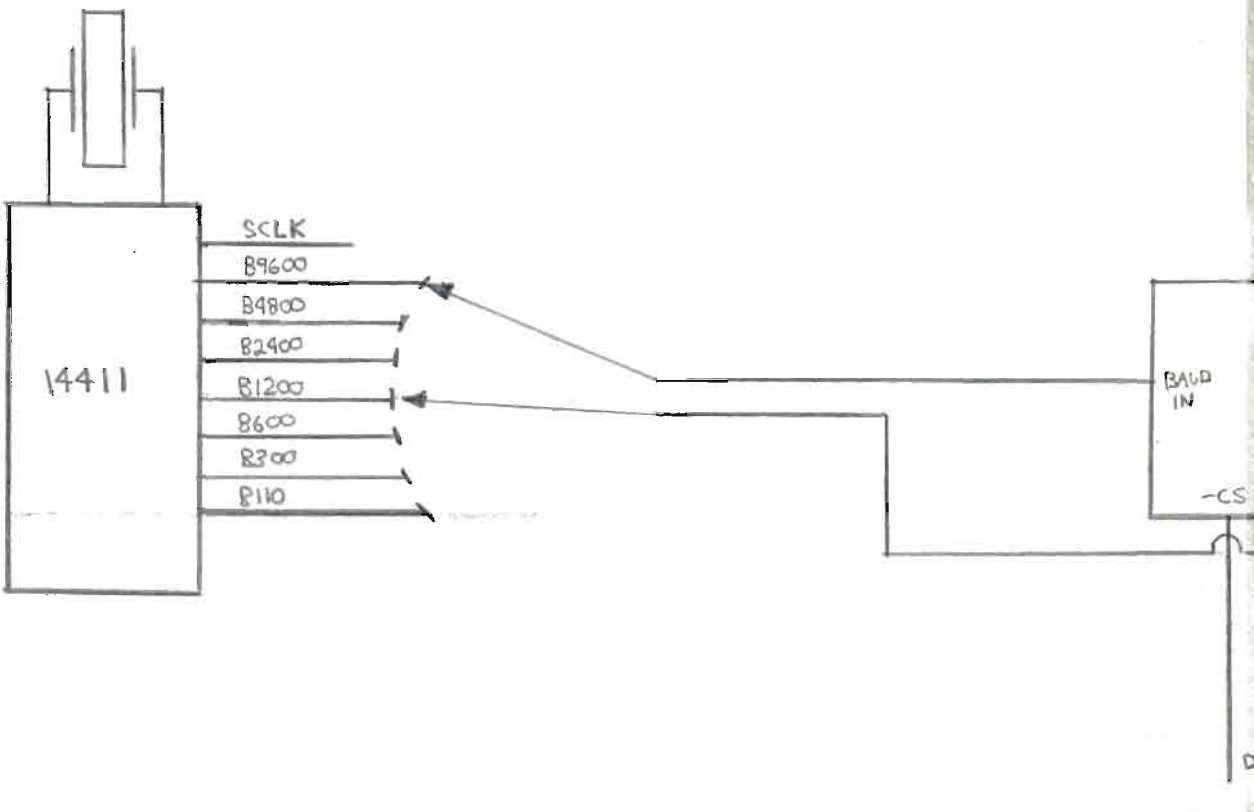



# Z80 DATAWIDGET INTERFACE CONNECTOR

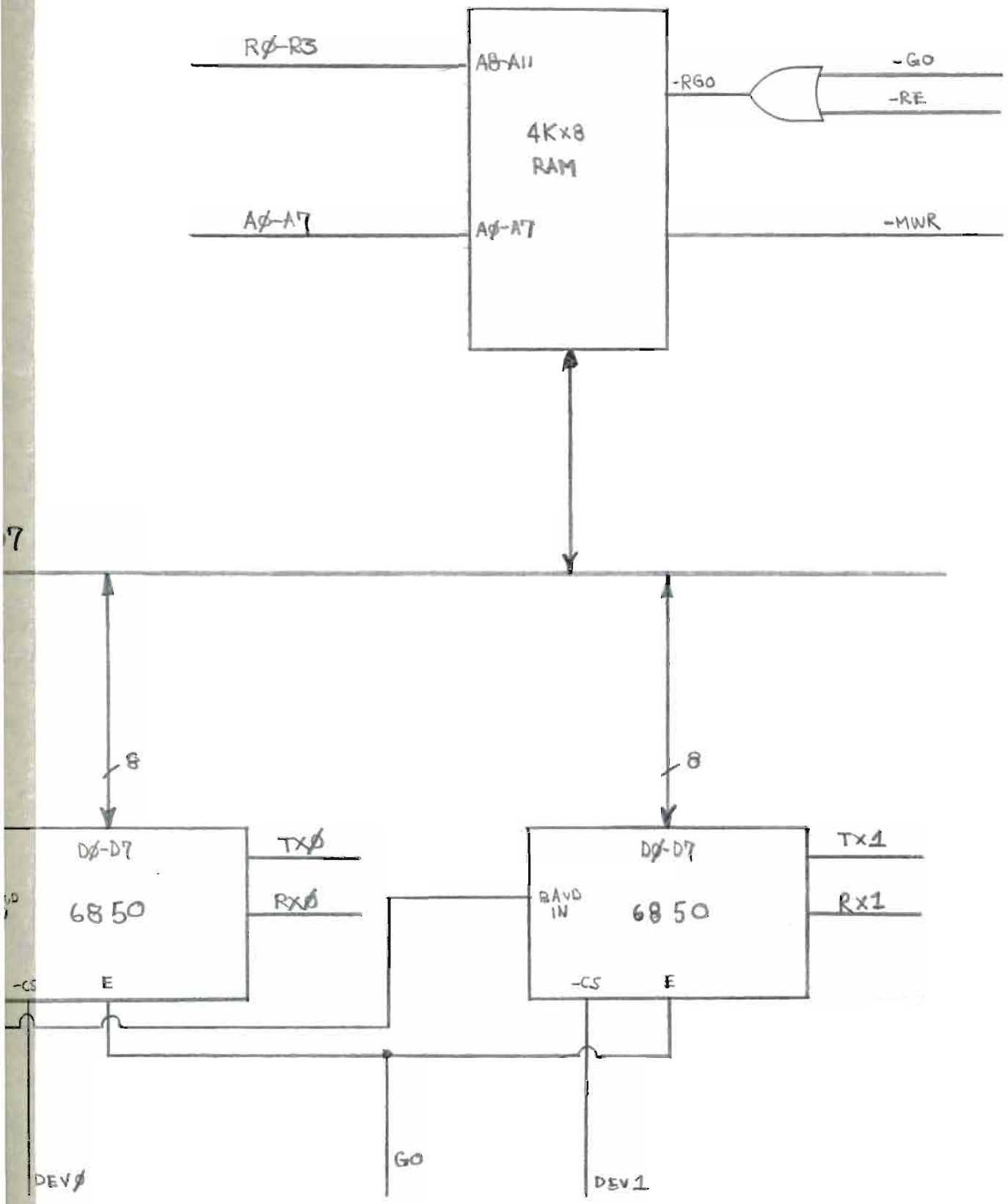


# Z80 DATAWI

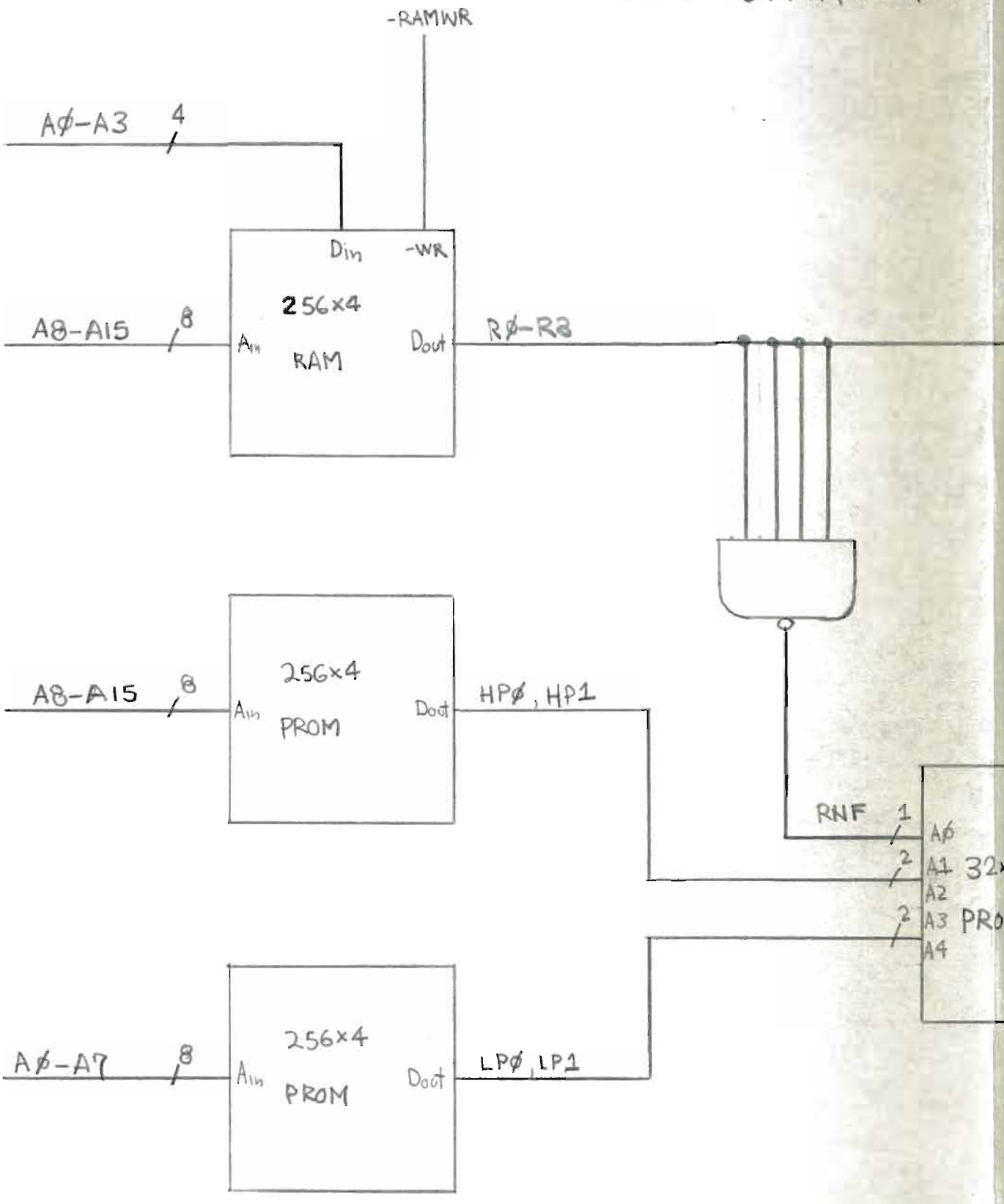

AG10
AG9
AG8
A9
A8
1or2Kx8
PROM
A0-A7
A0-A7
-PGO
-GO
-PE
D0-D7


SCLK
B9600
B4800
B2400
B1200
B600
B300
B110
14411
BAUD IN
-CS

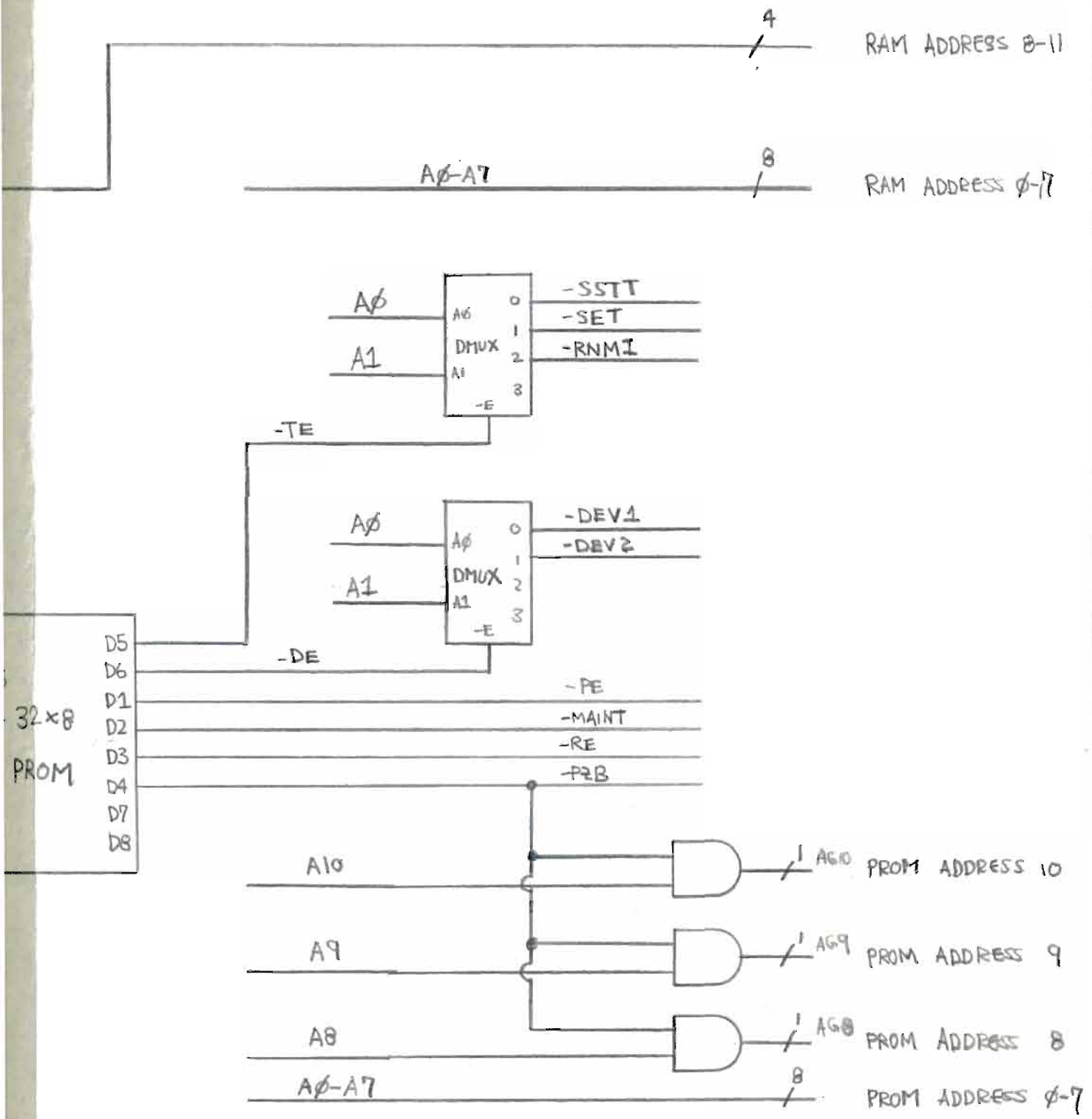
# WIDGET INTERNAL BUS



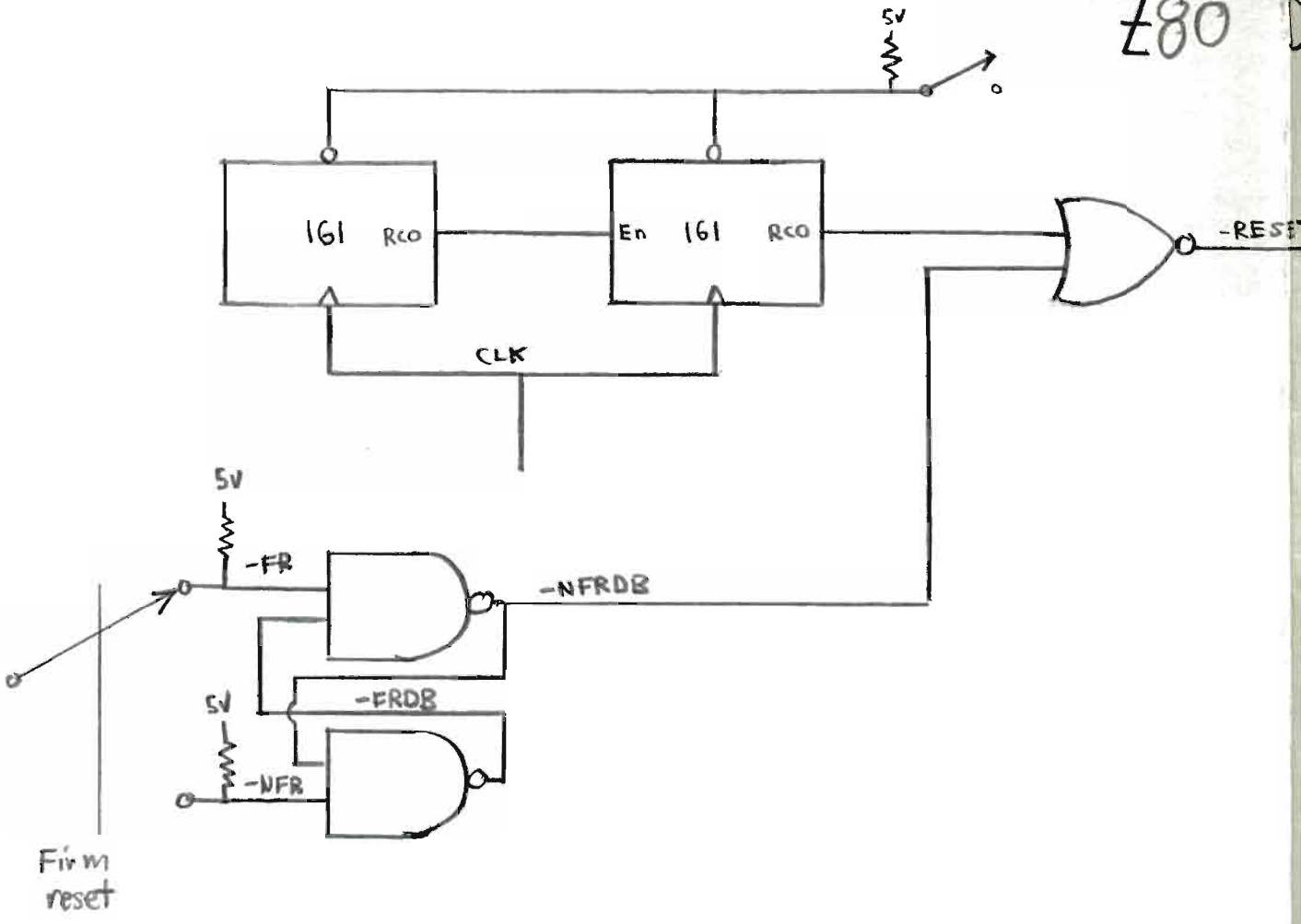
# Z80 DATAWIDG

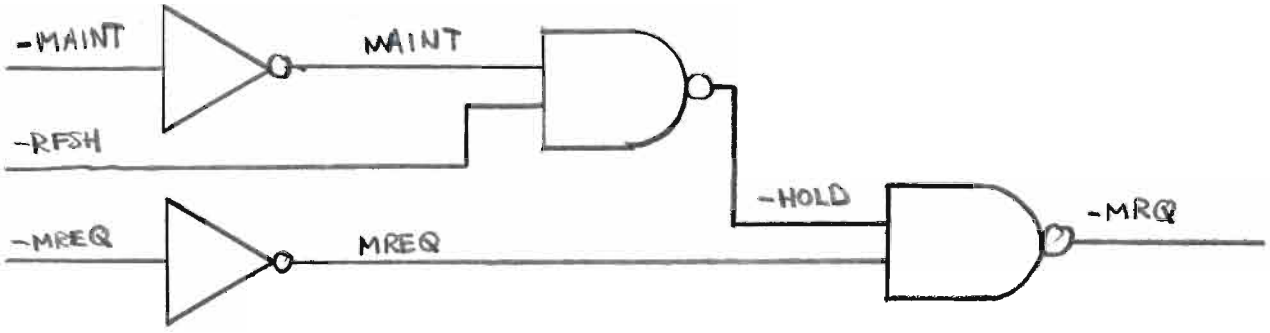


# GET ADDRESS DECODING



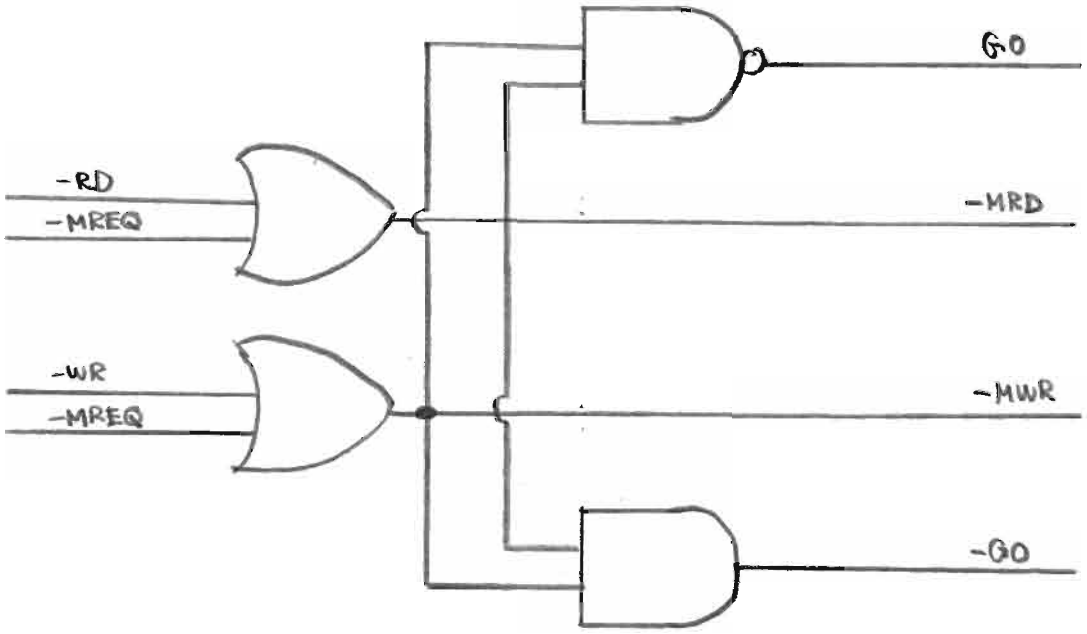
Z80


5V
161
RCO
En
161
RCO
-RESE
CLK
5V
-FR
-NFRDB
-FRDB
5V
-NFR
Firm
reset

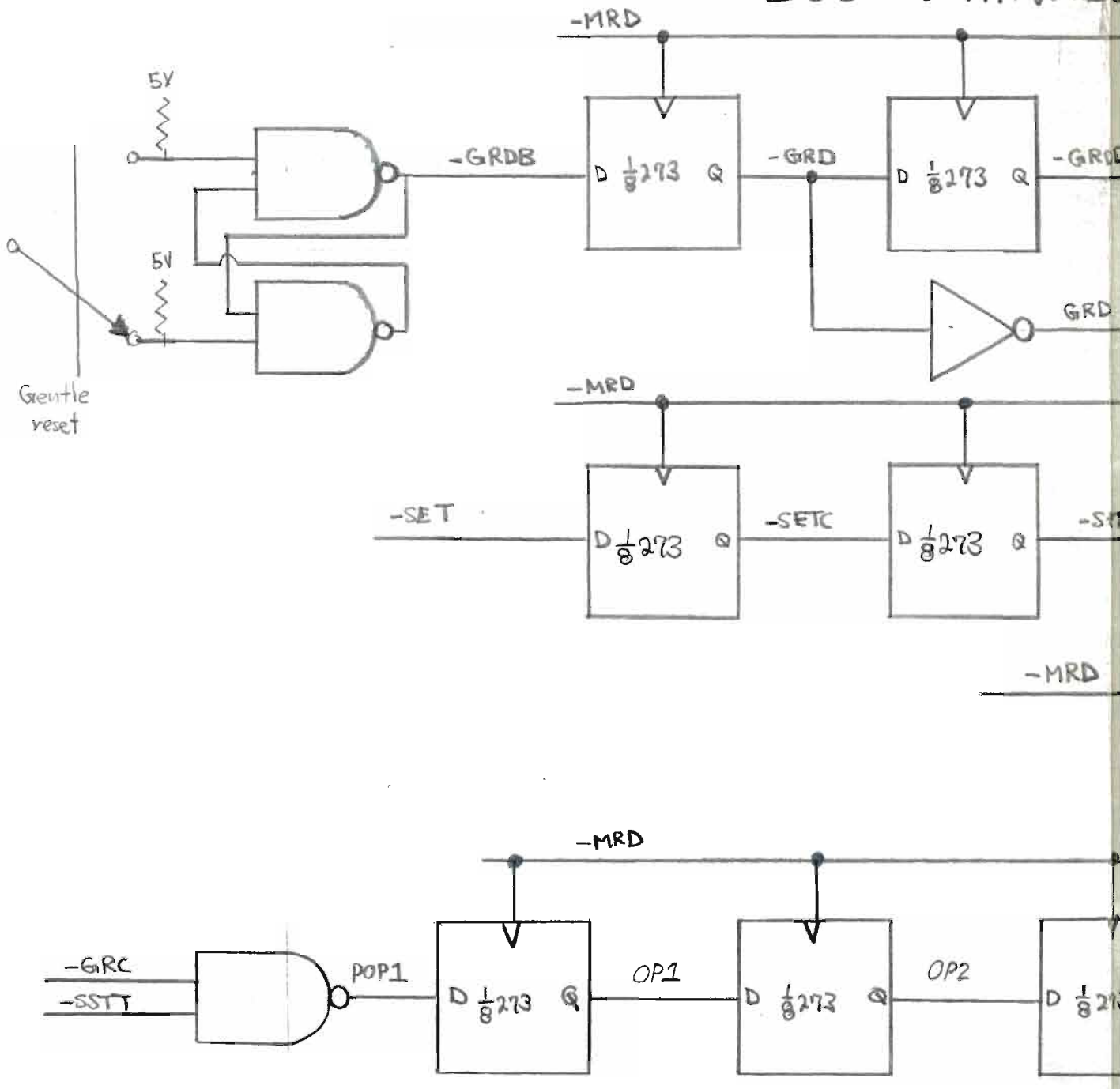

-MAINT
MAINT
-RFSH
-HOLD
-MREQ
MREQ
-MRQ

# DATAWIDGET

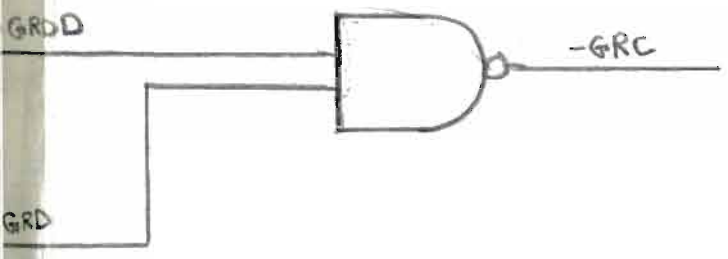
RESET

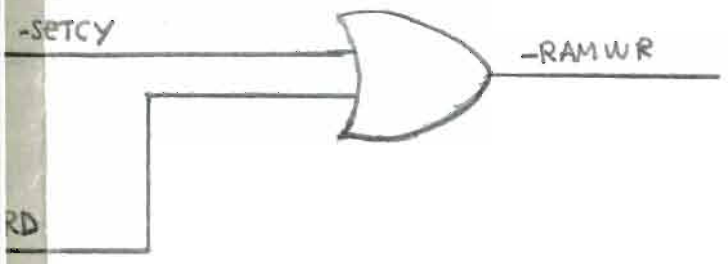


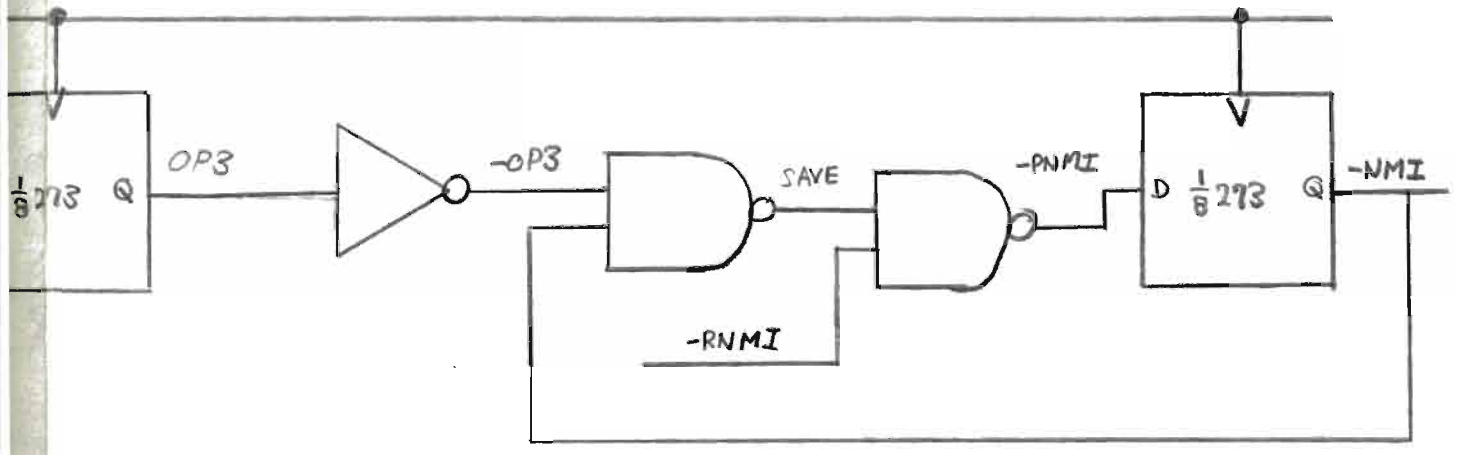
# Z80 DATA WID


-MRD
5V
5V
-GRDB
D 1/8 273 Q
-GRD
D 1/8 273 Q
-GRD
GRD
Gentle reset
-MRD
-SET
D 1/8 273 Q
-SETC
D 1/8 273 Q
-MRD
-MRD
-GRC
-SSTT
POP1
D 1/8 273 Q
OP1
D 1/8 273 Q
OP2
D 1/8 273 Q

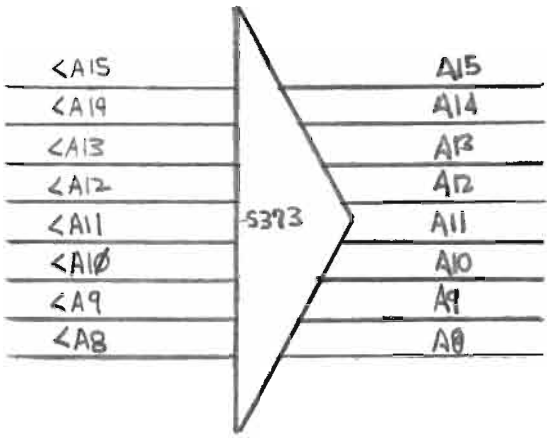
IDGET

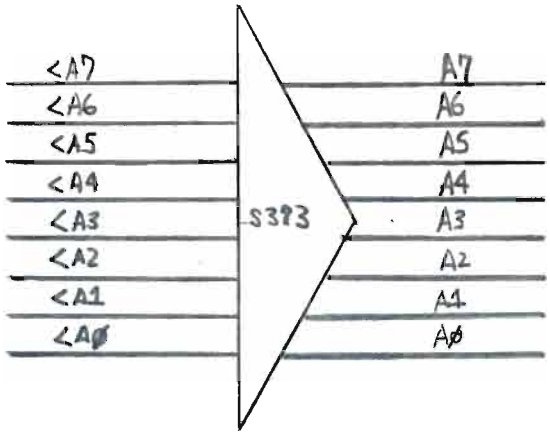


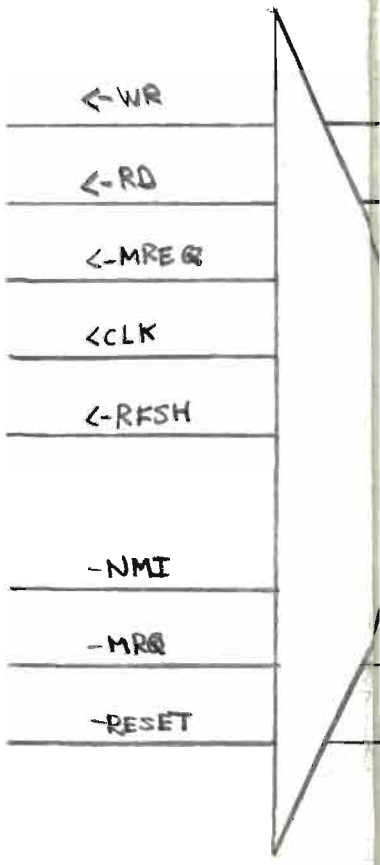



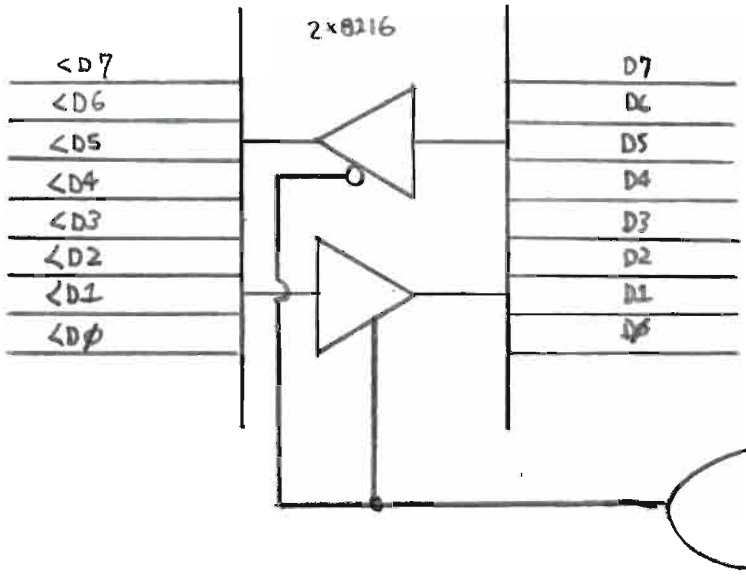



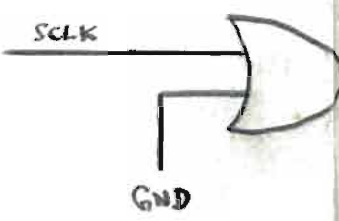
# Z80 DATAWIDGET



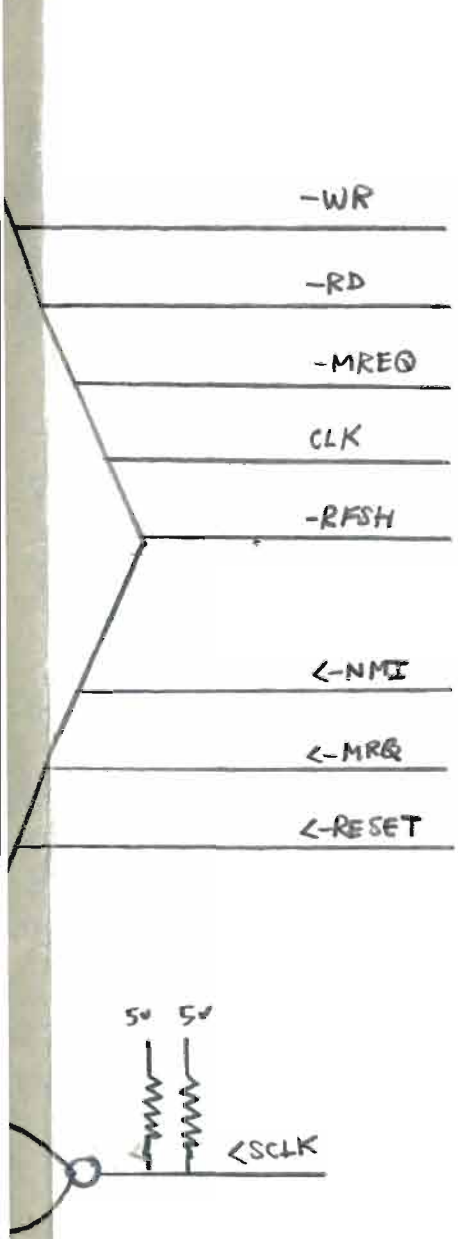



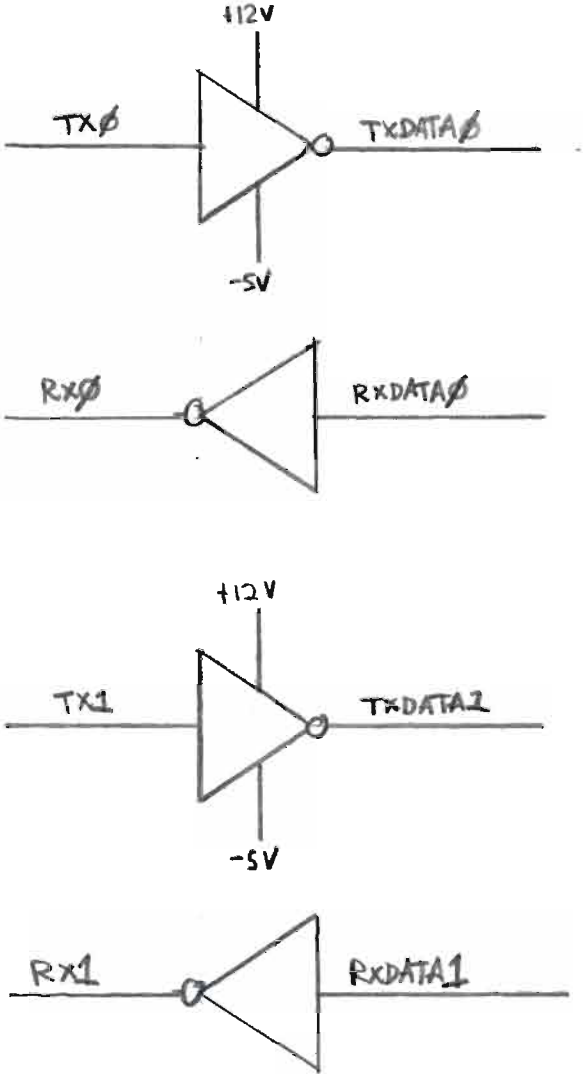



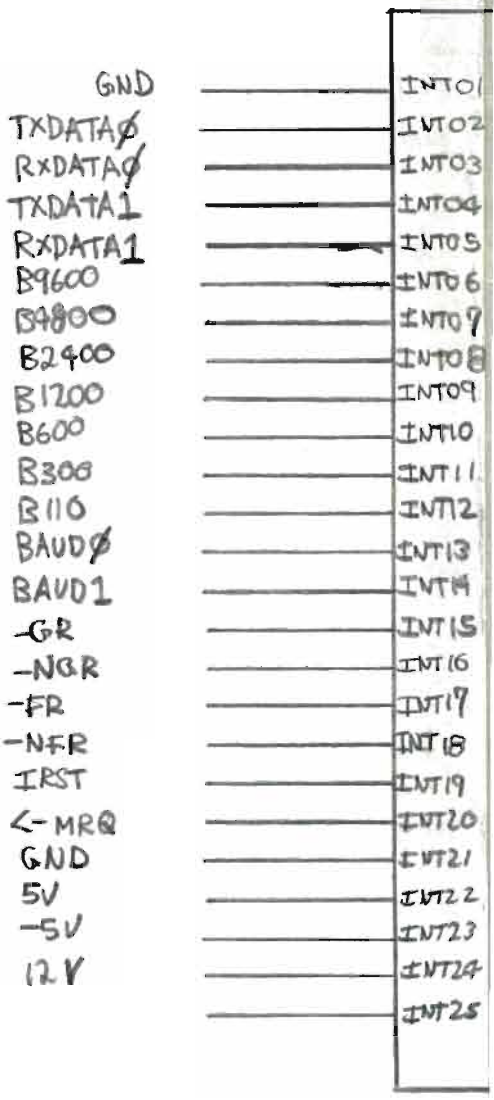
# BUFFERS


-WR
-RD
-MREQ
CLK
-RFSH
<-NMI
<-MRQ
<-RESET

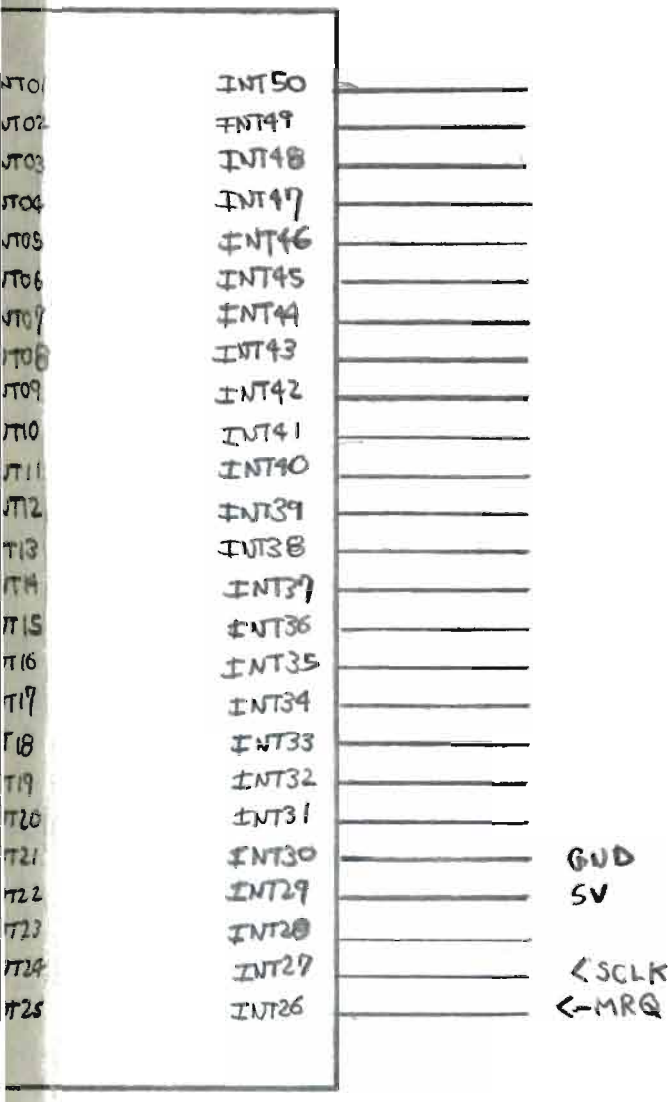

5v
5v
<SCLK


+12V
TXØ
TXDATAØ
-5V
RXØ
RXDATAØ
+12V
TX1
TXDATA1
-5V
RX1
RXDATA1

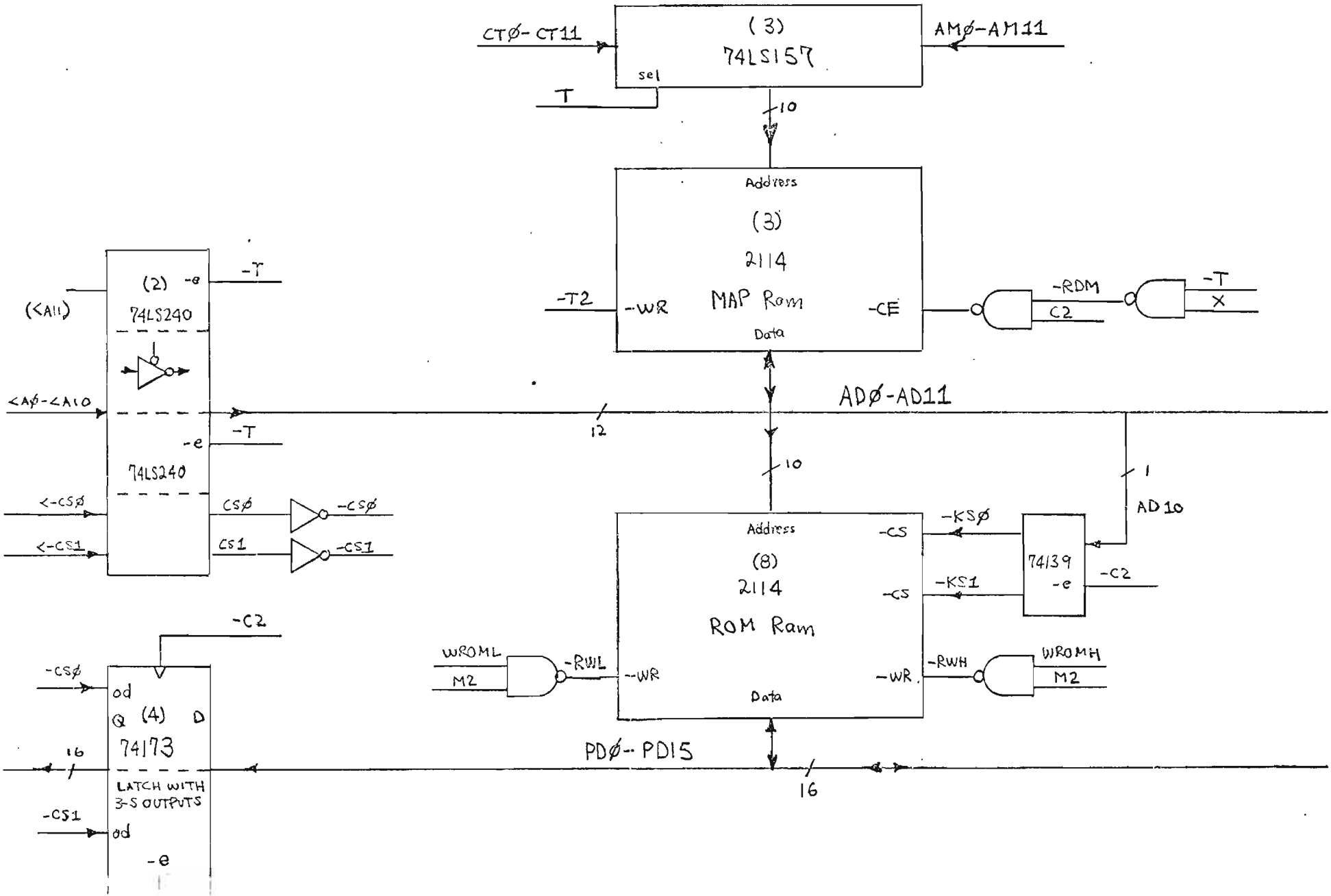
# Z80 DATAWID


GND
TXDATA0
RxDATA0
TXDATA1
RXDATA1
B9600
B4800
B2400
B1200
B600
B300
B110
BAUD0
BAUD1
-GR
-NGR
-FR
-NFR
IRST
<- MRQ
GND
5V
-5V
12V
INT01
INT02
INT03
INT04
INT05
INT06
INT07
INT08
INT09
INT10
INT11
INT12
INT13
INT14
INT15
INT16
INT17
INT18
INT19
INT20
INT21
INT22
INT23
INT24
INT25

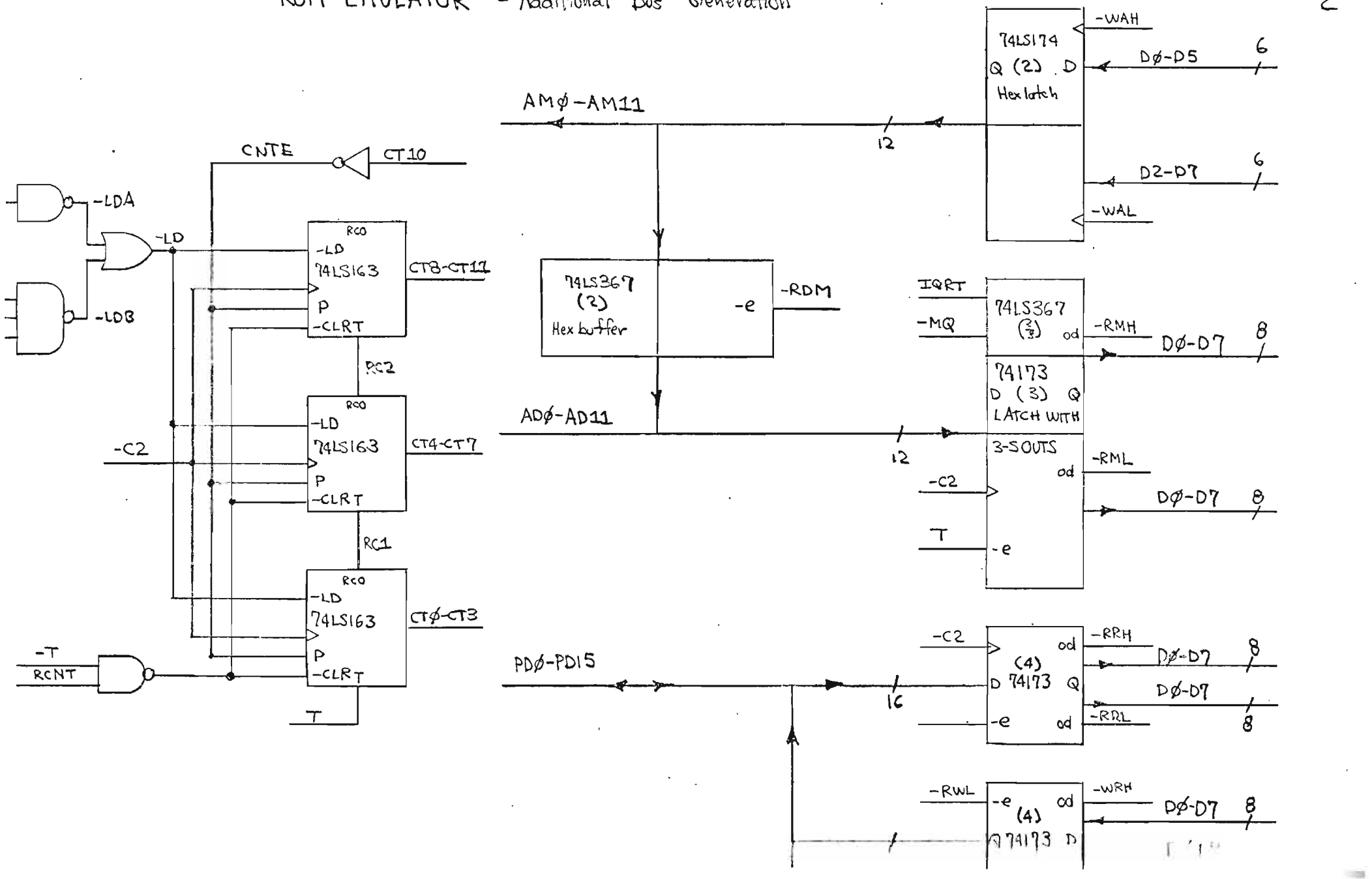
IDGET INTERFACE CONNECTOR



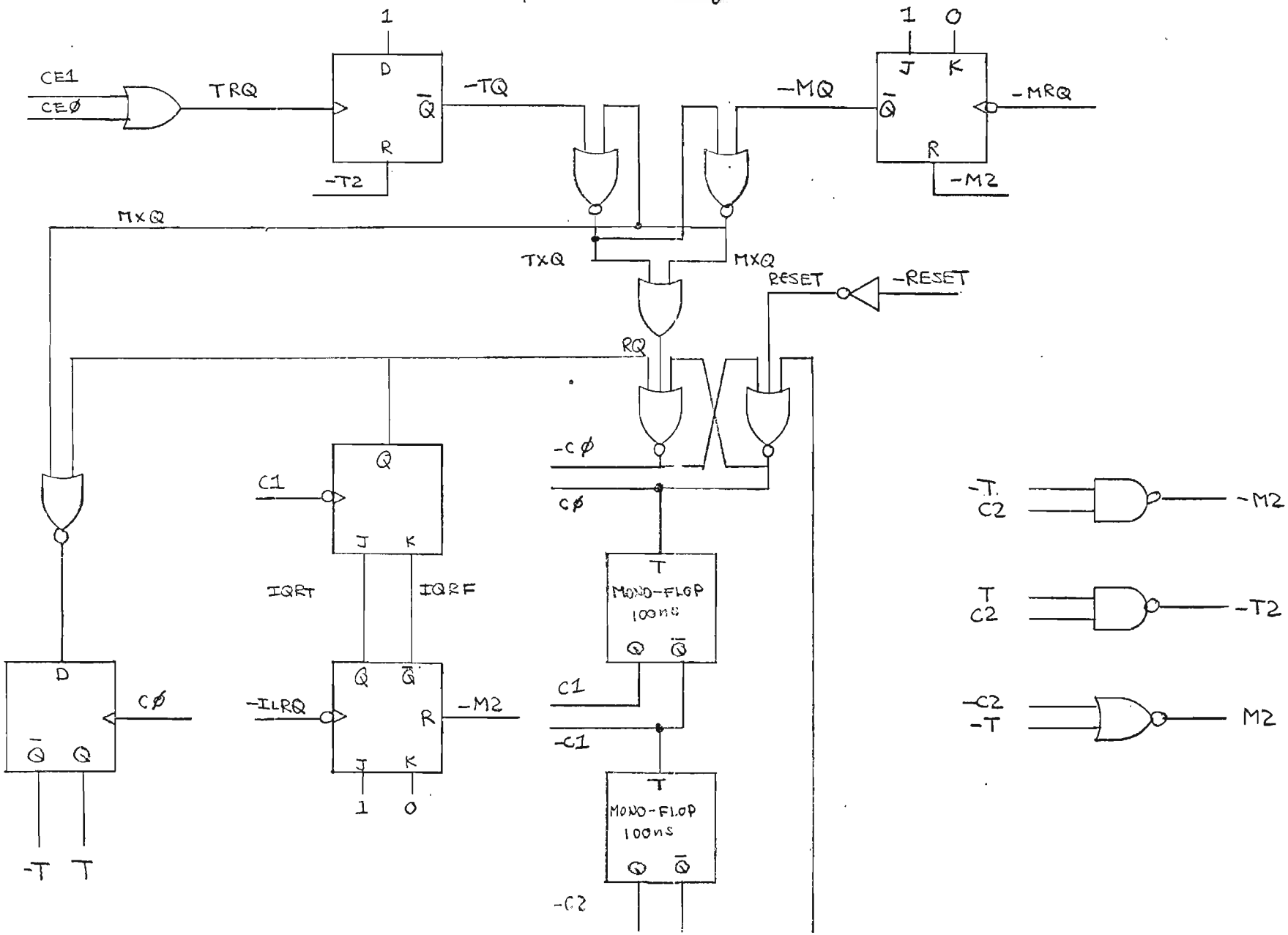
# ROM EMULATOR - Interface and Rams



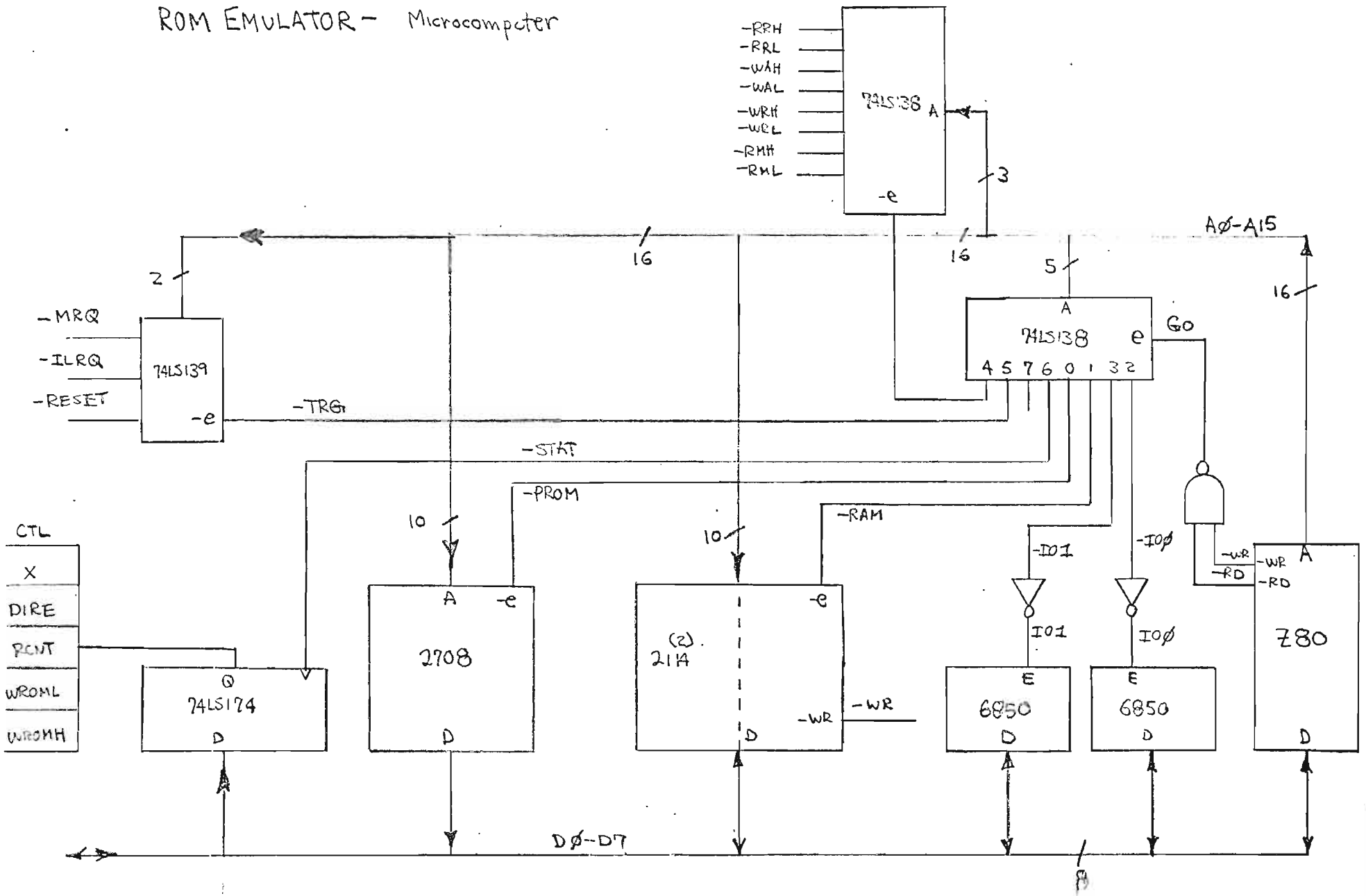
# ROM EMULATOR - Additional Bus Generation



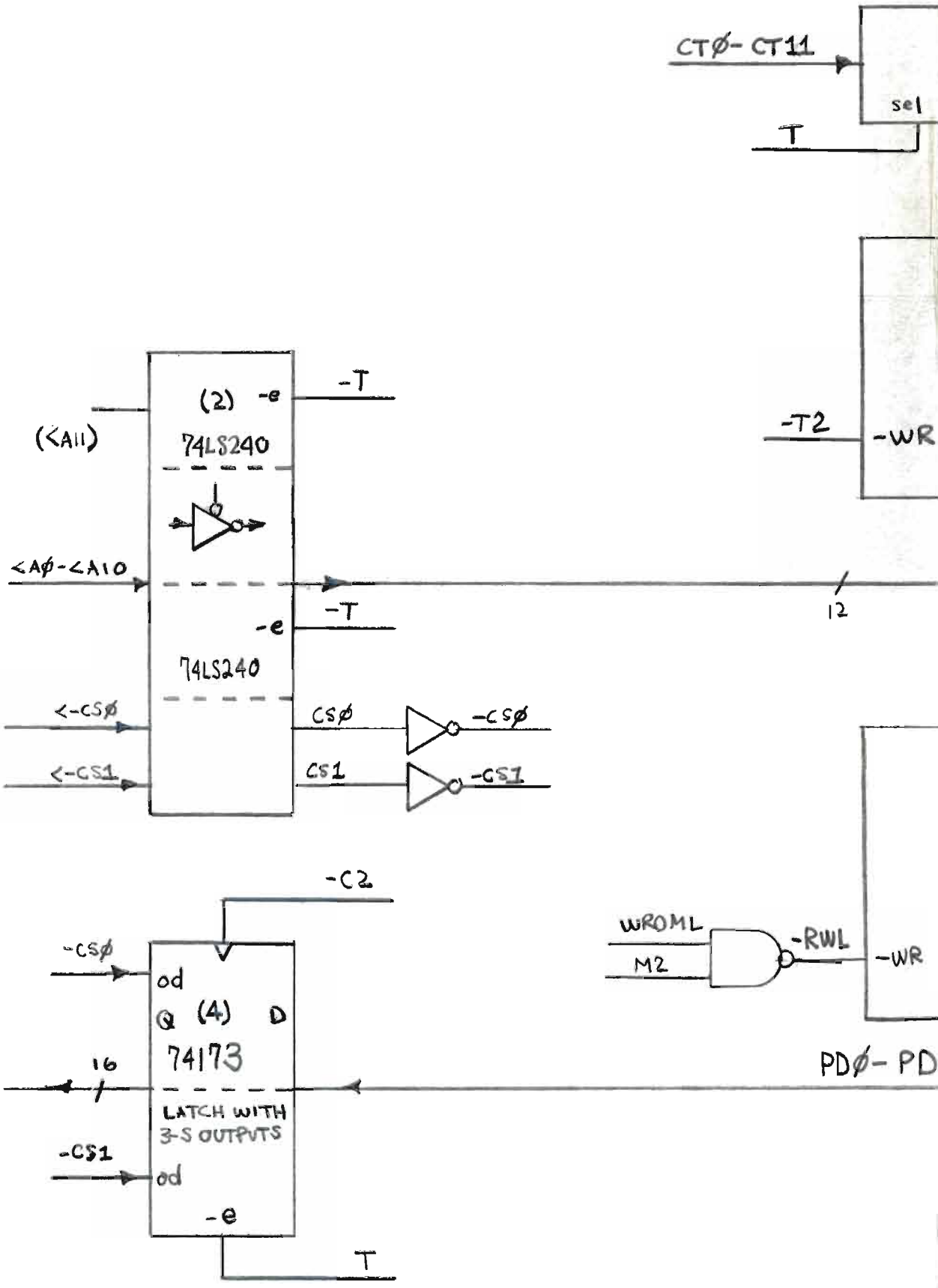
# ROM EMULATOR – Asynchronous Timing



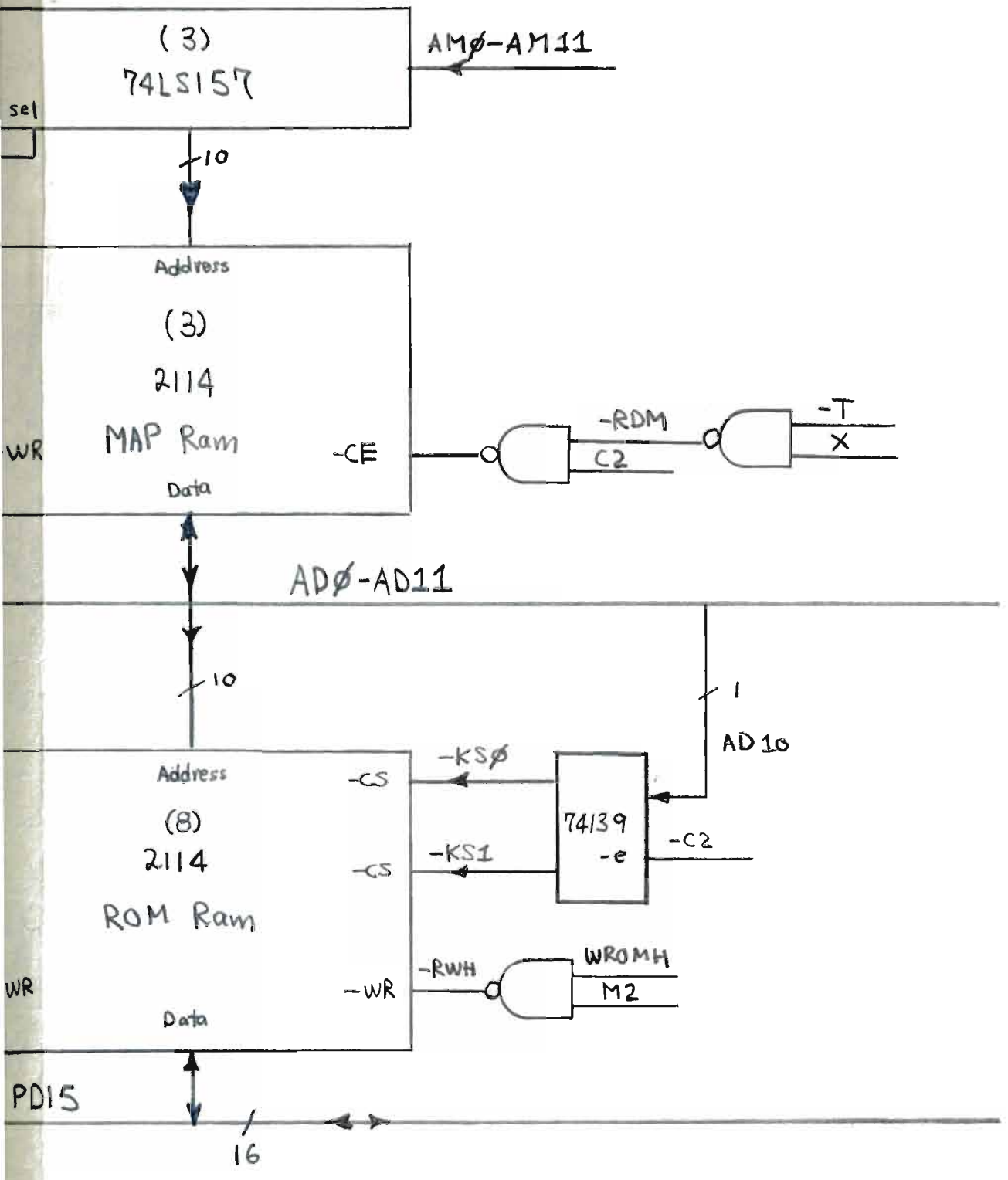
# ROM EMULATOR - Microcomputer



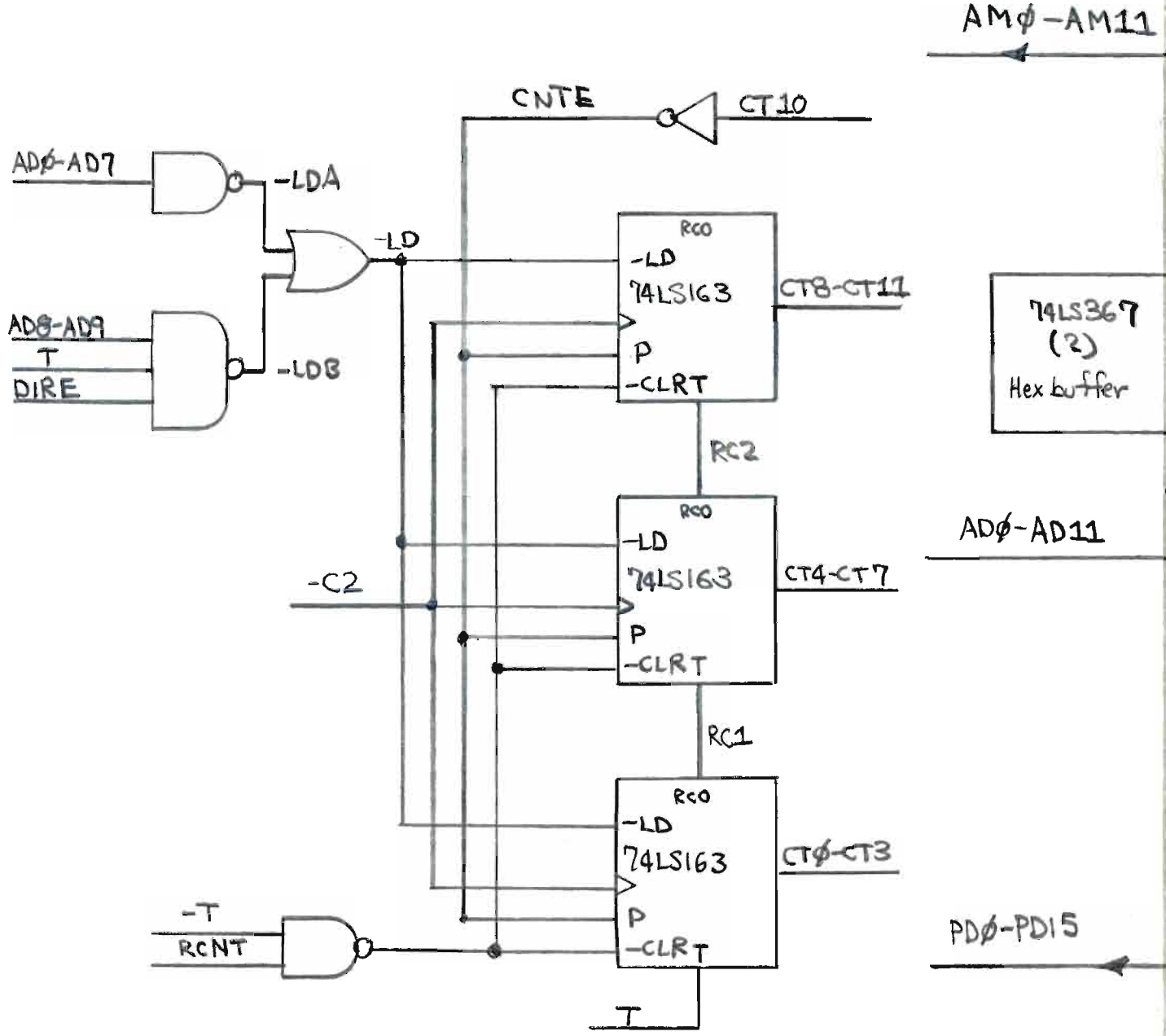
# ROM EMULATOR - Interface and Ra

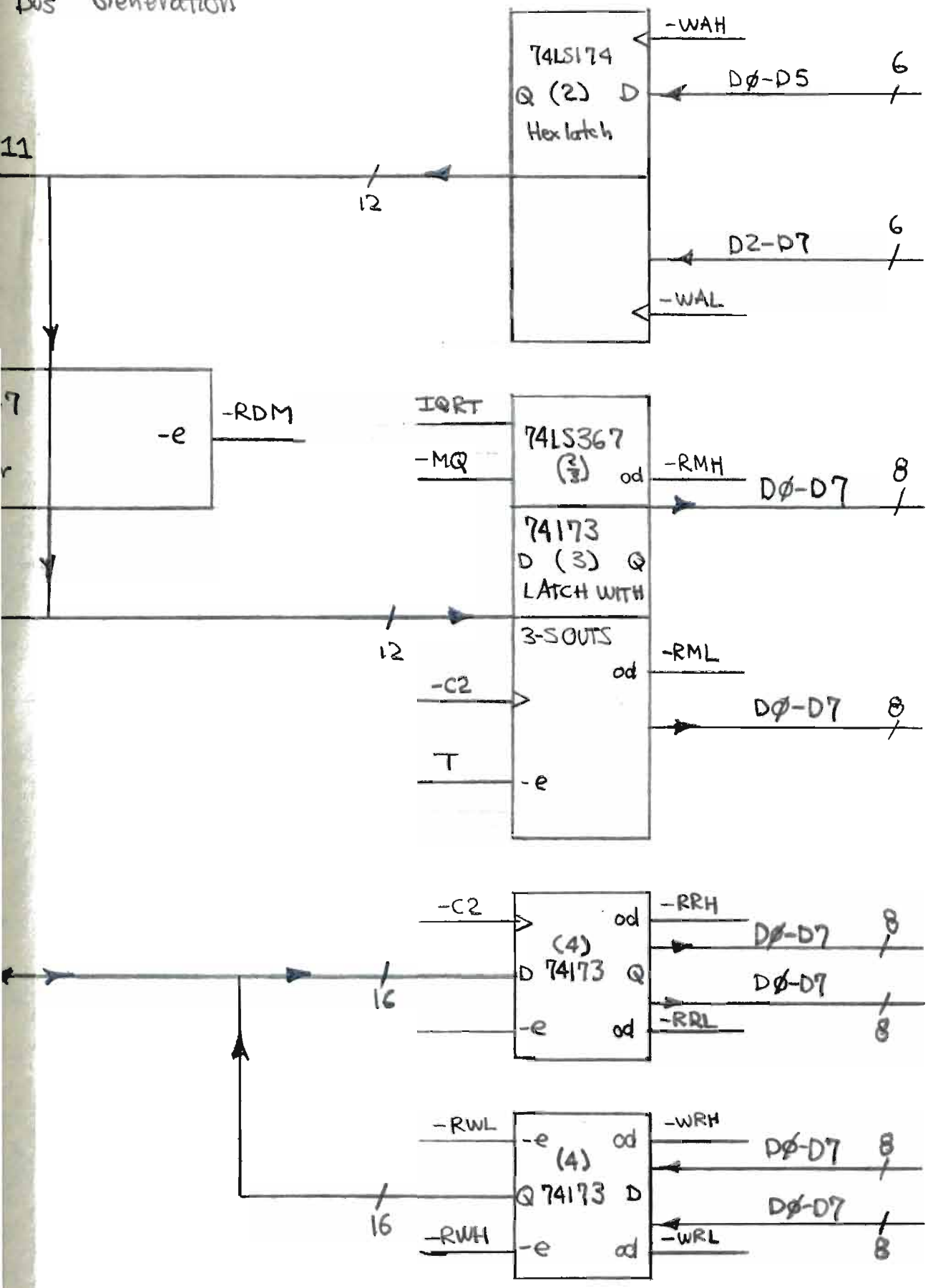


Rams

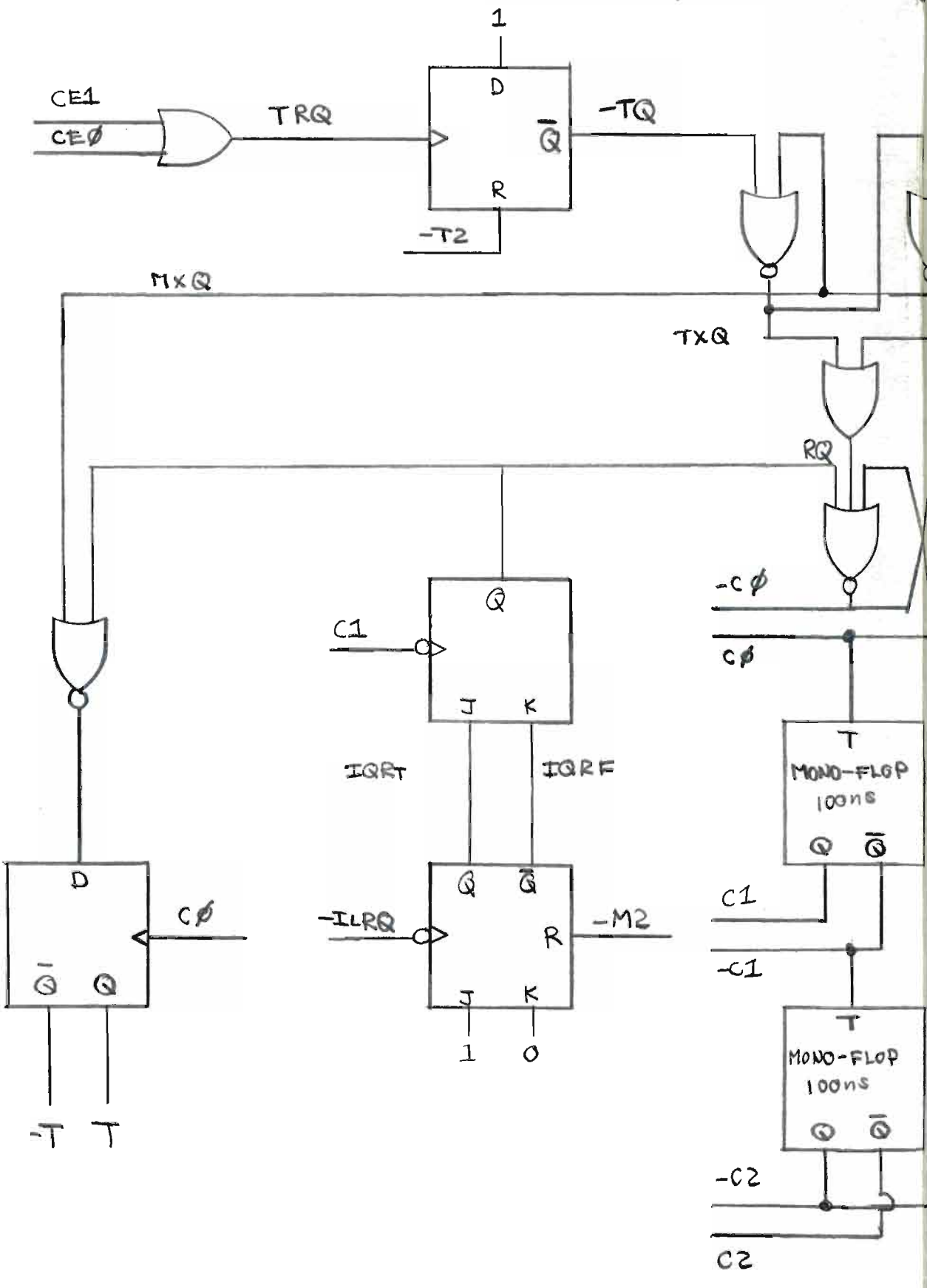

(3)
74LS157
sel
AMØ-AM11
10
Address
(3)
2114
MAP Ram
WR
-CE
Data
-RDM
C2
-T
X
ADØ-AD11
10
1
AD 10
Address
(8)
2114
ROM Ram
WR
-CS
-CS
-WR
-KSØ
-KS1
-RWH
74139
-e
-C2
WROMH
M2
Data
PD15
16

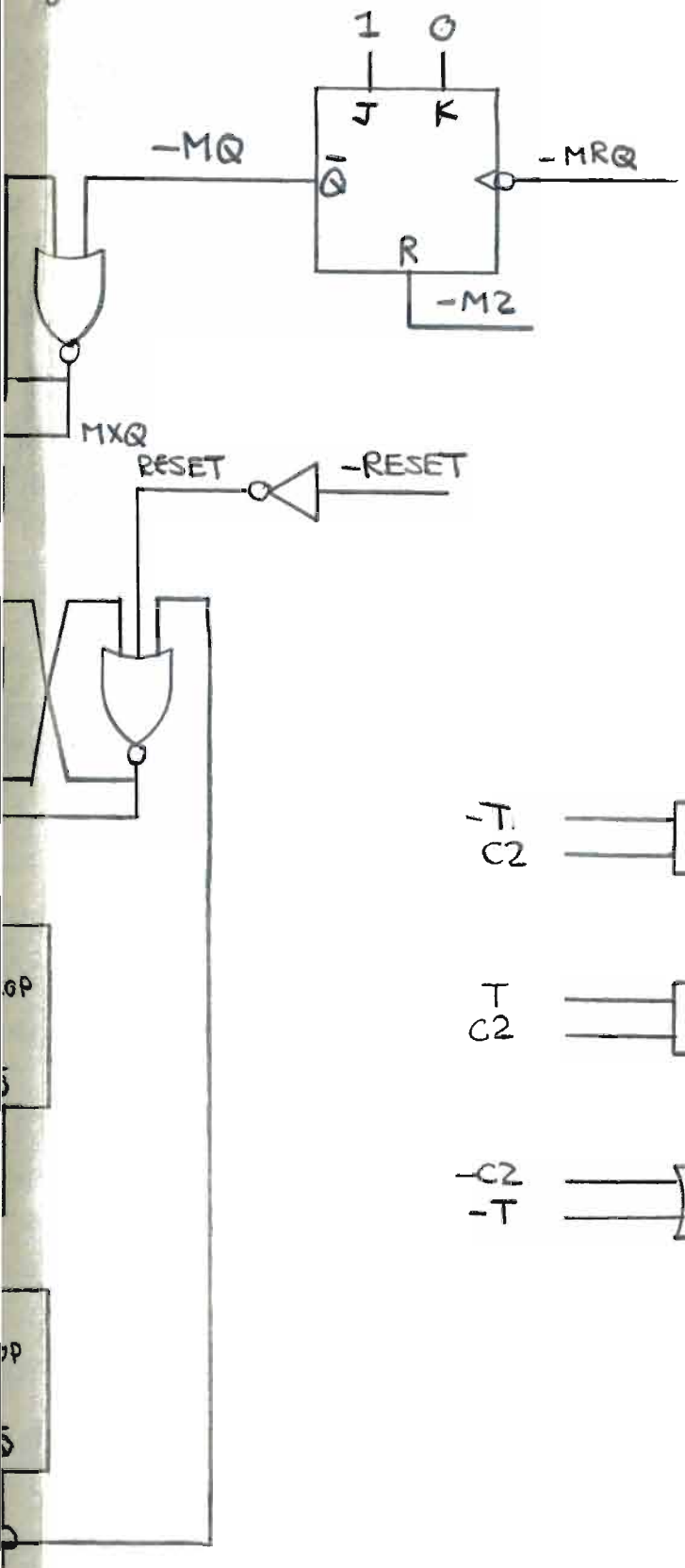
# ROM EMULATOR - Additional B

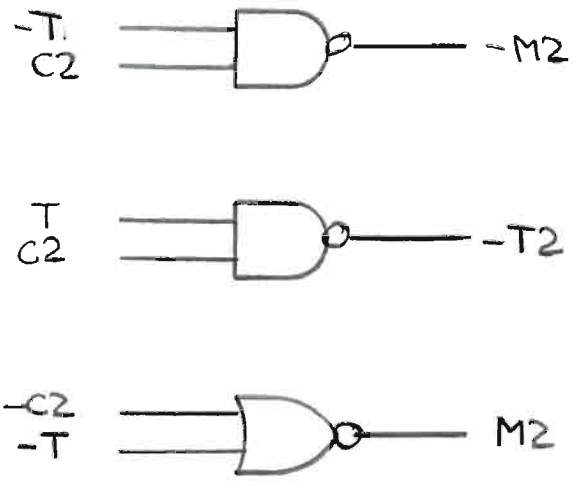



74LS174
Q (2) D
Hex latch
-WAH
DØ-D5
6
-WAL
D2-D7
6
12
11
-e
-RDM
IQRT
-MQ
74LS367
(2/3)
od
-RMH
DØ-D7
8
74173
D (3) Q
LATCH WITH
3-SOUTS
od
-RML
DØ-D7
8
-C2
T
-e
12
(4)
74173
-C2
od
-RRH
DØ-D7
8
D
Q
-e
od
-RRL
DØ-D7
8
16
(4)
74173
-RWL
-e
od
-WRH
DØ-D7
8
Q
D
-RWH
-e
od
-WRL
DØ-D7
8
16

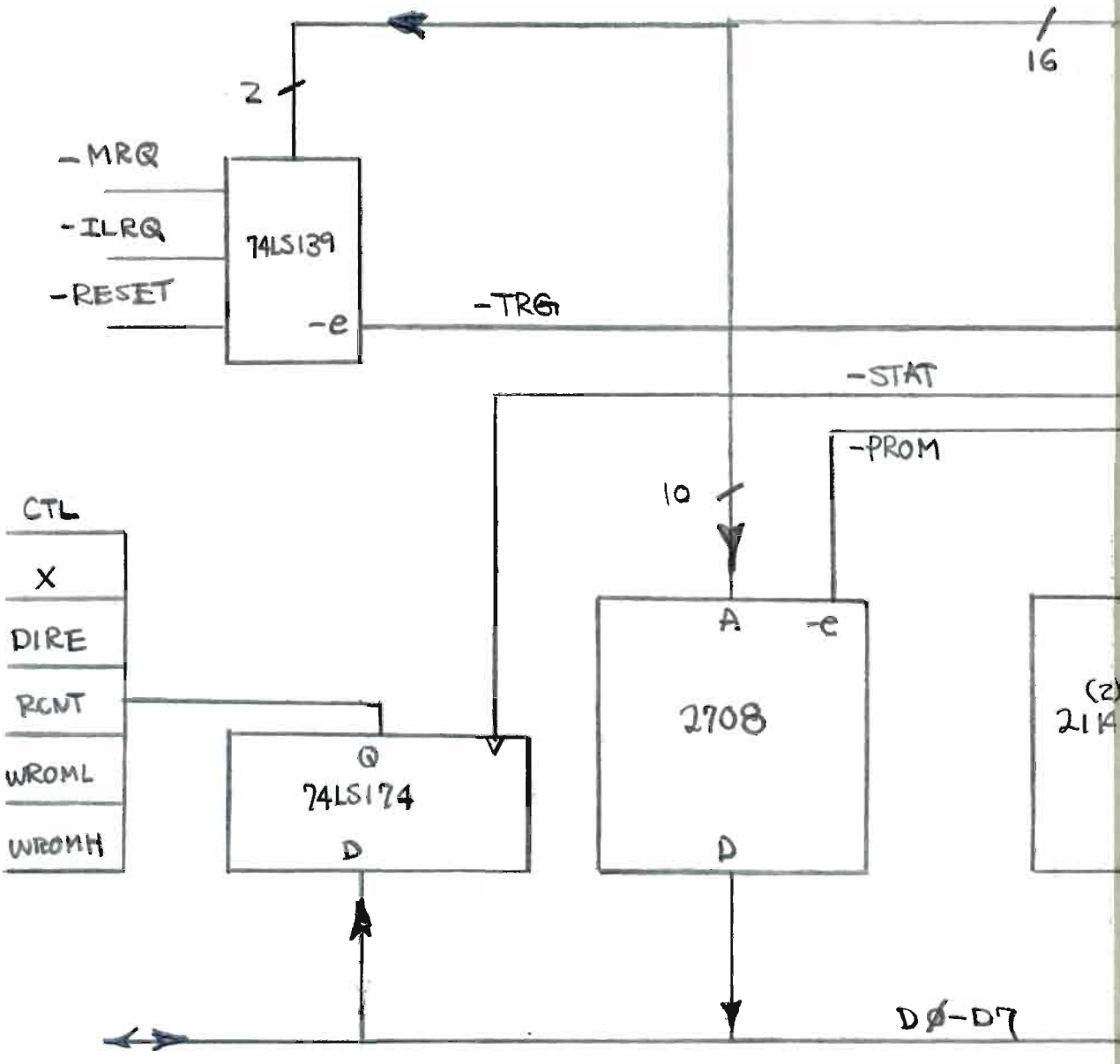
# ROM EMULATOR – Asynchronous Timing

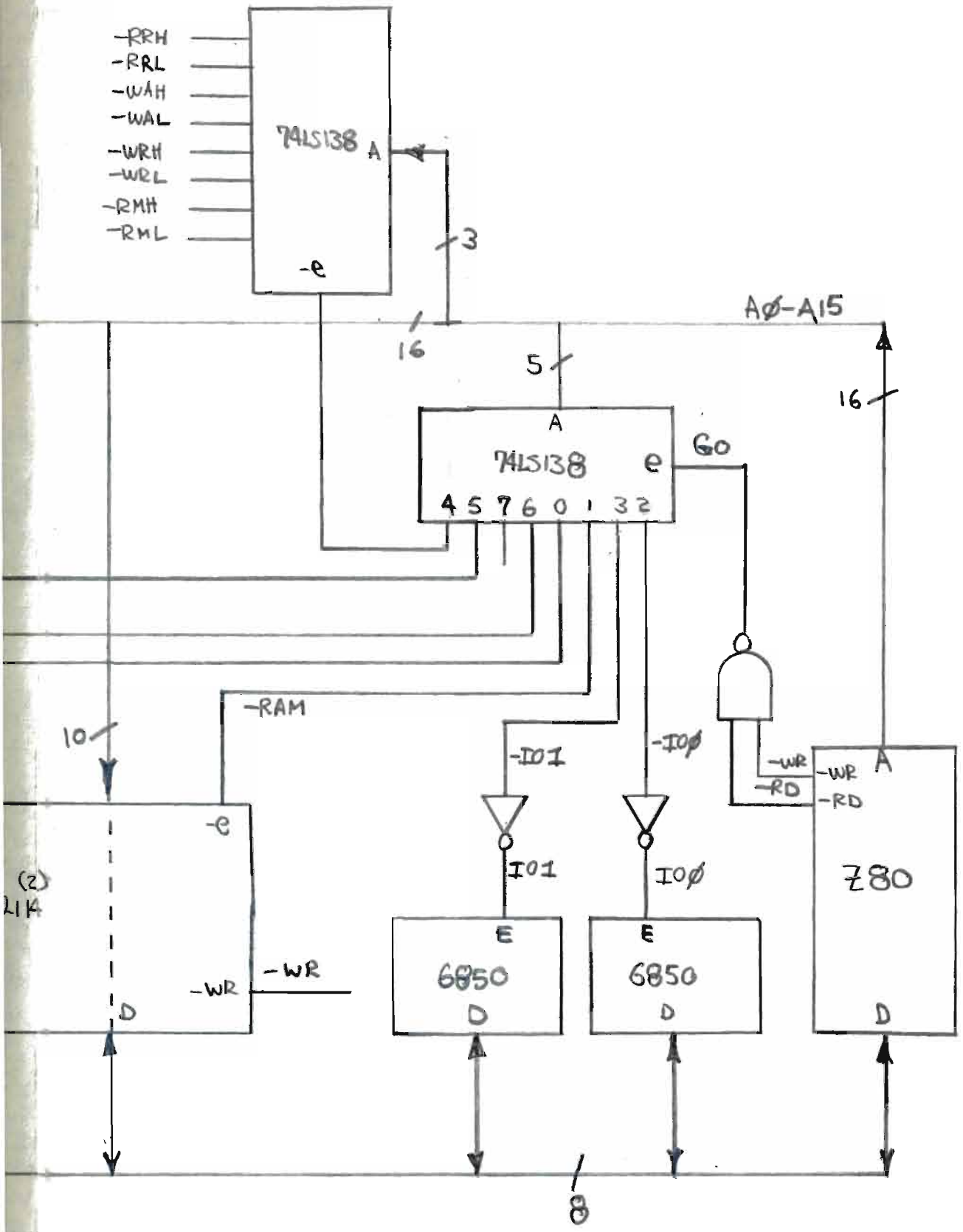



1
0
J
K
-MQ
Q̄
-MRQ
R
-M2
MXQ
RESET
-RESET


-T
C2
-M2
T
C2
-T2
-C2
-T
M2

# ROM EMULATOR - Microcomputer




-RRH
-RRL
-WAH
-WAL
-WRH
-WRL
-RMH
-RML
74LS138
A
-e
3
16
A0-A15
5
A
74LS138
e
G0
4 5 7 6 0 1 3 2
16
-RAM
10
-IO1
-IO0
-WR
-RD
A
-WR
-RD
-e
(2)
2K
I01
IO0
E
6850
D
E
6850
D
Z80
D
-WR
-WR
D
8