OHIO SCIENTIFIC 560Z

THIS PACKAGE INCLUDES (IN ORDER):

- *560Z INTRODUCTION
- * THEORY OF OPERATION
- * SCHEMATICS
- * CONSTRUCTION GUIDELINES
- * SOFTWARE DOCUMENTATION
- * SOURCE LISTINGS
- * OPERATION EXAMPLES
- * PIA DOCUMENTATION
- * Z-80 DOCUMENTATION
- * 6100 DOCUMENTATION

Also includes 560Z Board!

INTRODUCTION

The 560Z is a totally new and revolutionary computer product. We are inclined to call it a CPU Expander, or a computer lab on a board. The 560Z has many applications, including the following: execution of standard PDP-8, Z-80, and 8080 programs; investigation of microcoding and multiprocessing; investigation of Z-80 and/or 6100 operation and architecture; a building block for large, experimental multiple processor arrays. Each of these applications lends itself to further discussion.

The 560Z Board utilizes an Intersil 6100 microprocessor, which is capable of executing the standard PDP-8E CPU instruction set. It is important that we state that it executes only the CPU's instruction set, that is, it does not execute exactly I/O instructions that may be found on various PDP-8 configurations. This is one of the main reasons that PDP-8-compatible microcomputers have not been available, even though the Intersil 6100 chip has been around for a long time. This is because the PDP-8 uses what is known as microcoded IOT instructions; that is, each peripheral interface must have the intelligence approaching that of a conventional CPU. For instance, I/O devices can force skips, rotates, and other operations in the accumulator, as well as providing data and other instructions to the processor during I/O operations. The 560Z overcomes these difficulties by having the executive 6502 processor provide the intelligence on all I/O instructions. It does this by actually microcoding the PDP-8 front panel, interrupt, and IOT instructions. We will discuss the concept of microcoding a little later, but what this actually boils down to is that the 560Z, in conjunction with a 6502 microprocessor, can totally emulate the functions of a PDP-8E computer, allowing it to run standard PDP-8 code without any modification to that code at all. The PDP-8 currently has the largest library of public domain software available in the world. Ohio Scientific does not provide any standard PDP-8 software. We suggest that the user become a member of DECUS, the Digital Equipment's Users Group. There are thousands of programs available to members of DECUS at simply the cost of duplication, to run on PDP-8 IS and E-based computers. Software provided by Ohio Scientific in this package should allow the user to run any standard PDP-8 4K Teletype-based program without any modification. By simply adding entries to the IOT microcode table in the software, the user should be able to emulate, or simulate, any standard PDP-8 configuration, including those using disk operations and extended memory.

The 560Z also utilizes a Z-80 microprocessor Through this microprocessor, it is, of course, possible to run standard Z-80 programs. The 560Z is configured to allow the user to microcode interrupt and I/O instructions on the Z-80 via the 6502, so that he can simulate or emulate I/O ports found on other computer systems with his standard OSI computer system.

The Z-80 on the 560Z Board is capable of running 8080 code. It does handle interrupts slightly differently from the 8080; however, since the 560Z allows microcoding of I/O and interrupt operations, it is possible to configure the 560Z system to exactly emulate standard 8080-based computers to allow the operation of 8080 programs without any modification to those programs.

The field of microcoding, or microprogramming, is totally new to microcomputer users. Microcoding has really nothing to do with microprocessors, per se. It has to do with coding, programming, or specifying the instruction set of a computer. That is, during the initial design phases of any computer, and in some large computer systems, it is possible or necessary to specify or dynamically specify the binary codes which perform certain operations. The question is, what does the machine specify, and what exactly does it do when it performs an ADD operation? The 560Z system allows an elementary form of microcoding to be used with the 6100 and Z-80 microprocessors. This is possible because the 6502 system can have complete executive control over the computer system. It can read any signal line on either of the processors, and can force certain conditions. The use of microcoding is necessary for I/O instructions and interrupts on both the Z-80 and 6100 to perform the desired emulations, but can be utilized in other areas. To clarify this application of microcoding, let us consider a couple of specific examples.

Take first the case of a Z-80 processor executing a program and encountering an INPUT statement. On the 560Z, normally configured, when the Z-80 sees an INPUT or OUTPUT statement, a signal line on the processor goes low, which stops its clock so that the Z-80 is stopped dead with the INPUT instruction on its bus and the INPUT address port on the low eight bits of its address bus. The 6502 system processor then observes this condition, goes to a table in its code, which specifies what action to take (based on this input port), obtains the proper information, and shoves it into the Z-80. It then single-steps the Z-80 out of the INPUT instruction, in which case, the machine takes off at full speed again. In this way, the 6502 system, by a process of microcoding the Z-80 instruction, has simulated or emulated some standard 8080 or Z-80 I/O port by utilizing the resources that it has. For instance, an 8080 program may specify a Teletype port at input port 20, and the user may have only an OSI video board. This can easily be accommodated by microcoding the input. In another example, a 6100 is performing an IOT instruction to Teletype. This is more sophisticated, because the Teletype port must specify conditions within the accumulator of the 6100. But the principle is the same. The 6100 sees the instruction and its processor, and its clock automatically stops, leaving the critical data and addresses on its buses. The 6502 then reads these buses, takes appropriate action, and single-steps the 6100 chip through the IOT instruction, providing it with information, and reading information from it as necessary. When the 6100 has been clocked all the way through the instruction, the machine takes off at full speed. Although the microcoded I/O instructions themselves take much longer than normal instructions do, their occurrence in programs is usually infrequent so that the actual program execution takes only a fraction of a percent longer than it would take with conventional I/O.

All signal lines and the address and data lines of the Z-80 and 6100 are available to be examined under program control by the 6502. The Z-80 and 6100 clocks are also single-cyclable by the 6502 and the machines are fully static. This allows a very elaborate and detailed study of the Z-80 and 6100 on a cycle-by-cycle basis with complete printout of all signal lines. This enables the student or engineer to completely understand the operation of these processors, including many subtleties which are not documented in any manuals for either processor.

The 560Z system can be utilized as a true multiprocessor. That is, the 6502 executive can set up a task on the 6100 or Z-80 and then detach itself from it, so that the 6100 or Z-80 is running completely by itself, independent of the 6502, which can then go on to another task. Thus, in the 560Z system, two processors can be running separately and independently. Multiple 560Zs can be placed on one 6502 system, and 560Zs can be placed on the system bus side of other 560Zs, so there is really no limit to the number of processors or amount of memory present on 560Z-based computers, allowing large, multiprocessor arrays.

It is apparent from the above discussion that the 560Z is a powerful research tool. Several 560Zs can be placed on an individual computer bus, and 560Zs can be daisy-chained. It is also possible to cross-couple units with or without processors. That is, two 560Z Boards can be populated to have portholes only with no processors present, and be utilized between two computer systems as high-speed portholes from one system to the other. Other exotic multiple processor arrays are of course possible by using the 560Z Board as a building block with the standard OSI system bus.

The 560Z output bus drivers are fully tristatable by an external device. This is done specifically so that other processors can be present on the system side, as well as on the executive (MOS) bus side. This is of particular interest to 510 Board users, because the 510 software switch has a fourth position which can be utilized to control the tristate outputs of the 560Z. It is therefore possible to have a 510 CPU Board on the executive (MOS) bus side of a 560Z and to have a 510 present on the system side. This means that it is a very straightforward operation to utilize a 560Z Board as a porthole between two 510 systems, or as an intelligent porthole with its own processor capability.

560Z CIRCUITRY

Since it has been stated earlier that the 560Z is an advanced circuit for the microcomputer expert and enthusiast, heavy emphasis will be placed on the theory of operation of the circuitry. Construction guidelines will be clearly set down since any person prepared to build the 560Z is expected to have ample previous experience in both the construction of kits and homebrew circuits.

DIAGRAM 1 ADRESS DECODING AND ENABLE CIRCUITRY

The 6502 executive side of the 560Z is referred to in all diagrams as the MOS bus. The other side of the board, which minimally must have 4K x 12 of RAM memory, is referred to as the system bus. There are several signals which appear on both ends of the bus, such as read/write, and these signals are labeled MOS read/write and system read/write, accordingly. The 560Z occupies a total of 4.25K of memory address space in the executive system. 4K of this is the system porthole; the other 256 words are used to access the four PIAs on the board. This 4.25K address space can be placed on any 8K boundary by selecting inverted or non-inverted A13, A14, and A15 via IC-FF and IC-R. IC-Z and IC-AA further decode A12 through A9 to provide four consecutive K for the porthole with the following 256 words for the PIA. It is strongly recommended that you jumper your board to conform to the OSI standard of placing the porthole at E000, which will then automatically place the PIAs at F000. This is a convention utilized with all 560Z soware. Address line A12 is provided inverted and non-inverted to combinational logic and mixed with read/write and MOS 02.VMA to form the board enables for both the porthole and the PIAs. The board enables are dialed or wired together to form the Data Direction line for the MOS data bus. MOS 02.VMA and the MOS Data Direction are gated via PIA signals for porthole operation which will be discussed later.

DIAGRAM 2 DATA AND ADDRESS BUS

It is extremely important that the 560Z user become intimately familiar with the subtleties and intricacies of the data and address bus on the 560Z system. First, let's consider the address which is the lower portion of Diagram 2. The MOS address bus is presented to the input of a set of four 74125s which can be enabled or disabled by TOM-bar. The 6502 system can optionally provide addresses to the 560Z Board and beyond. The address bus goes to both the Z-80 and the 6100 and is then passed out to 8T95 buffers to the system bus. The 8T95 buffers can be optionally tristated when used in conjunction with an additional 510 CPU on the system bus side. This is possible because there is a fourth position available on the software processor select switch on 510 computers. This feature allows a partially populated 560Z to be used as a unidirectional high-speed porthole between two 510 computer systems, a feature which is very valuable for experimenation in computer science. The upper four address bits are also passed through a four-line multiplexer which can select four address bits on the 6502 and the Z-80 systems on one side, or, from four PIA port lines on the other side. This feature allows memory

management to occur with the Z-80 and the 6502 systems and allows extended memory operations in 6100 or PDP-8 programs. The use of the memory multiplexer enables the 6502 system to access any of 65K of memory on the system bus side, while only addressing 4K of memory on the MOS bus side.

Now on to the data bus. There are four 8T26s on the MOS bus side of the 560Z data bus. Two 8T26s are simply used to feed the four PIAs on the board and utilize conventional OSI system circuit technology. The other two 8T26s feed the 560Z's data bus and beyond. This data bus is fed to the Z-80, the 6100, and optionally, onward to the system data bus. The data bus is also connected to the PIAs. It is very important that the user's system circuitry is properly functioning, and that he is fully familiar with the circuits before attempting any of his own software for use in controlling these circuits because there are no less than five tristate devices which can provide signals to this bus. It is not very difficult to attempt to get two tristate devices to output to the bus at the same time. If one of these devices is an 8T26 and the other is a micro, you will end up with a burned-up micro microprocessor. The output side of this has three 8T26s to service the twelve-bit wide 6100.

There are several subtle points about this bus. First of all, both sets of 8T26s on the 560Z's data bus have independently controlled IE and DE signals. This means that these 8T26s can be set to input, output, or totally tristate; that is, these 8T26s are set up as bi-directional switches with an off position so that sections of the bus can be totally isolated from one another. One more important feature is that the 7475 latch present on the system data bus allows the 6502 system to latch in the upper four data bits from twelve bit memory so that they can be read from a PIA port.

DIAGRAM 3 PIA IMPLEMENTATION

This simple diagram has extremely significant implications in the operation of the 560Z system. It is very important that the user become intimately familiar with the 6820 PIA since it is the heart of the 560Z system. There are four PIAs present on the PC Board which are selected by address A2 and A3 and decoded by IC-Z and IC-M. That means that there are a total of twenty-four PIA control registers on the board, which control sixty-four bi-directional I/O lines and sixteen interrupts. The standard syntax used for PIA lines used in the system is A0 throught A7 and B0 through B7 on PIA 0, 1, 2, & 3, so that 1A1 specifies pin 3 of the PIA at IC-S.

The 560Z software package includes a general purpose PIA register handler which is used commonly as a subroutine by other routines, and would be very valuable to exercise in any portion of the 560Z Board. Pin assignments are specified in the table immediately preceding Diagram 6. Each signal line will be discussed in detail later.

DIAGRAM 4 Z-80 PINOUTS

The Z-80 microprocessor has tristatable address data lines which

are forced into tristate condition during reset. Thus, it is possible to connect address lines A0 through A15 directly to the 560Z address bus and data lines D0 through D7 to the data bus. Additional control signals are connected through multiplexers to the system bus and PIAs. These multiplexers are discussed later.

DIAGRAM 5 INTERSIL 6100 IMPLEMENTATION

The Intersil 6100 has a far different architecture than other microprocessors such as the Z-80, 6800, and 6502. This is mainly because it has been designed to conform as nearly as possible to the PDP-8 minicomputer. It is extremely important that the user become quite familiar with the intricacies and timing diagrams of the Intersil 6100 in conjunction with attempting to understand the operation of the system. There are quite a few subtleties involved.

First of all, the Intersil 6100 has a twelve-bit data and address bus, which is time-multiplexed. During certain periods, this bus contains valid input or output data; at other times, it contains valid address. These combinational data lines, DX0 through DX11, are tied directly to the twelve-bit data bus of the 560Z. The upper four data bits are also connected to a four-bit data latch, allowing eight-bit processors to read the upper four data bits. Signal LXMAR is used to toggle the valid address into two hex D-flops, IC-HH and IC-GG, which are then outputed to tristate buffers, IC-PP, IC-OO, and IC-NN, which are controlled by one of the system control multiplexers. The other control lines are routed through multiplexers to both the system bus and PIAs. Many lines required for the operation of the PDP-8 switch register interrupts and IOT instructions are routed through the PIAs so that the 560Z can totally emulate the operation of the PDP-8E by use of software.

DIAGRAM 6 BUS CONTROL

The major states of the 560Z Board are controlled by two 74153 dual four-to-one multiplexers. These multiplexers are controlled by two PIA lines, referred to as CONPIA1 and CONPIA2. The table near the lower left corner of Diagram 6 specifies the states of the boards. When both CONPIA1 and CONPIA2 are high, the MOS bus is routed through to the system. In this mode, the bus acts as a porthole. With CONPIA2 high and CONPIA1 low, the Z-80 is selected; with CONPIA1 high and CONPIA2 low, the 6100 is selected; with both low, the third processor is selected.

Let us study the operation of the two 74153s and the 74155 in detail. First of all, consider the top half of the 74155. This provides a low signal to one of four lines, leaving the other lines high. These lines are TOM-bar, Z-80 Bus Request-bar, and 6100 Data Request. These lines are then inverted or non-inverted as necessary to the proper signals. This multiplexer insures that one and only one device is operable and the other devices are tristated at any given time. That is, the board can act as a porthole with the two processors tristated, or, with the porthole disconnected and either one, but not both, of the processors enabled. The other half of the 74155 properly routes interrupts to the active device.

The two 74153s primarily provide signals to the system. That is, the processor selected control signals are routed to the system bus in a form compatible with all Ohio Scientific system boards. First consider the left side of the 74153. This multiplexer provides a synthetic 02. VMA. When the board is in the porthole mode, it is a gated 02. VMA, that is, 02. VMA is only present on the system bus side in a high or clock state when the 6502 is addressing the porthole. Under normal circumstances, this is at E000 to EFFF. When the Z-80 is selected, a pseudo-02. VMA is produced by gating Z-80 MREQ-bar with Z-80 RFSH-bar. When the 6100 is selected, we use an inverted MEMSEL-bar. The right side of this is used to control one-half of the output 8T26s on the 560Z Board. Remember that the 560Z can be the source or the receiver of data in either direction. Therefore particular caution must be taken with this hardware to prevent disastrous conflicts. One-half of the 8T26s are always controlled by the system cards, just as in any OSI bus system, so that when the 6502 system is selected, system data direction is provided to both IE and DE on the output 8T26s. When the Z-80 is selected, a common gated IRQ and read-data is provided from the Z-80 because sometimes the Z-80 does not want data presented to it. The 6100 does not normally require gating of this signal line, but, it is optionally provided for anyone who might require it by a jumper.

The 74153 ICV is utilized to provide both system 02 and system read/write. System 02 is provided from the appropriate processor clock signal, which may vary, depending on how the user has jumpered the clocks. The right side of this is used to select read/write which comes straight from the read/write line or the Z-80 or 6100 read/write line. The top half of Diagram 6 is the all-important clock generation circuitry for the 560Z. The MOS 02 single cycle is then put out to a 7493 which will provide clock signals divided by 1, 2, 4, 8, or 16. Both the 6100 and the Z-80 can be run at 1MHz or 2 MHz, of course. They can be run at higher speeds asynchronously, if desired. The signals and clock frequencies desired are jumpered into 74157 for each processor. The other side of the multiplexer is connected to the PIA. The select line is OR-wired to a large number of control devices. In the simplest form of operation, the two control lines (CONPIA3 and CONPIA5) are high so that the standard full-speed clocks are routed to the processors. By bringing the select line CONPIA5 low, CONPIA3 can be used to single cycle the processors including the system bus. This allows the 6502 to slowly read and write information into twelve-bit 6100 memory. It also allows complete cycle-by-cycle analysis of the operatin of both the Z-80 and the 6100.

The real power of the board, however, is realized with the OR-wired control lines. These allow full microcoding of the 6100 and Z-80 instructions. Consider the case of an Intersil 6100 processor running at full speed which encounters a switch register instruction. In this case, it will bring line SWSEL low. This will pop the 6100 clock signal over to single step which causes the clock to now come from CONPIA3. Since both the 6100 and the Z-80 are fully static chips, the Intersil 6100 will be locked in a SWSEL instruction. The 6502 microprocessor can casually come in and read the condition of the 6100 single step . The 6502 can also inject data, if so desired, onto the 6100's bus, which it will then read and interpret as a switch register output. The 6502 will then single-step the 6100 chip

throughout its SWSEL operation, until SWSEL goes high, in which case the 6100 will take off at full speed again until it encounters an interrupt, a switch register, or an IOT instruction which will allow full microcoding. This will cause the processor to revert automatically to single-stepping, and thus permit full microcoding. Since in any normal program, such instructions are only executed a small percentage of the time, the total execution speed of a program with microcoding is only negligibly longer than it is on a conventional machine without microcoding. It is in this way that the 560Z system can emulate other machines such as the PDP-8E and the Altair 8800 completely--without any modification at all to user software.

Please refer to the table at the end of Diagram 6 in conjunction with this discussion. PIA-0, A0 through A7, PIA-1, A0 through A7, are sixteen address lines which are directly across the address bus of the system. These lines can read the address currently present on the 560Z Board. Refer to the software discussion for further details. These PIAs can force addresses on to the address bus. When forcing addresses, extreme caution should be taken to make sure that all other devices are in a tristate or read mode (rather than write mode) to avoid conflicts on the tristate bus and possible damage to the PIA port or some other part. PIA-1, PB0 through PB7, are the data bus on the system and are directly across the 560Z data bus. They of course can read data at any given time, and can write data, but again, care should be taken to insure that other devices are tristated when a PIA is used to force data on the data bus. PIA-3, A0 through A-3, are the upper four data bits in twelve-bit systems. By presetting these four data bits and executing a normal write operation to the 560Z's porthole, twelve bits will be loaded into system memory. PIA-3, PA4 through PA7, are the corresponding data bits coming back from the 7475 address latch. By simply reading any system memory location with the board set in a porthole status, the 7475 will automatically latch the upper four data bits which can be read via these PIA ports. Any attempt to output data on these four PIA ports will cause damage to that PIA since it will be in direct conflict with the output of the 7475. PIA-0, PB0 through PB2, are Z-80 control signals which are useful primarily for educational use of the Z-80 and 8080. These are available for classroom and industrial research on the Z-80 and 8080 processor. They can also be used to microcode certain instructions such as interrupts. PIA-0, PB3 through PB7, are the all-important memory management lines. PIA-0, B3, is the strobe, or enable, for the memory management. When this line is brought low, the four address bits specified by PIA-0, B4 through B7, are brought out on address lines 12, 13, 14, &15. Some address must always be specified with 6100 systems, and it may be desirable to specify the upper address with Z-80 systems. It is always necesary to specify the memory management address when utilizing the porthole. PIA-2, PA0 and PA1, are additional signal lines on the Z-80, which are provided primarily for research and educational applications for the Z-80. PIA-2, A2 through B7, are all control lines on the Intersil 6100. Some are available for only educational and experimental purposes while others are quite necessry for normal operation of the system. Refer to the software discussion and source code for applications of these lines. PIA-3, B0 and B1, are the two processor-select, or CONPIA1 and CONPIA2 lines, which specify the mode or state of a board. PIA-3, B2 and B4, operate the single-step run multiplexer and can force any processor in the single-step mode and single-step machine. PIA-3, B3, is the 6100 run halt, which specifies run or stop operations on the 6100. PIA-3, B5

through B7, are important Z-80 control lines.

SUGGESTED METHOD FOR BUILDING YOUR 560Z SYSTEM

The 560Z is an extremely complex subsystem. It is extremely important that you have a good understanding of what is going on before you attempt to construct and use the board since it is quite probable that you could have fully operational hardware and not know it because of operator error. The 560Z has been called a computer lab on a board, and rightly so. It is extremely powerful and has broad applications and uses which haven't even been thought of yet. But because of this, it will demand your best effort. No one can simply stuff up the PC board from the parts overlay without a fundamental understanding of the principles. It is very important to work in an orderly, step-by-step process. This is particularly important because of the large number of tristatble, bi-directional devices on both the address and data buses. Anybody who attempts to simply stuff up the board and apply power to it will most likely burn out several parts. It is, therefore, imperative that you build the unit and test it in stages to preclude the possibility of destroying expensive components.

Here are the recommended steps. First of all, learn how your Ohio Scientific 6502-based system works. Look at some schematics, follow the descriptions in the 400 or 500 manuals; discover how the data direction line, 02. VMA, and other critical control signals interact with the boards. Secondly, learn how to read and understand the 6502 Assembler code. The 560Z operating system is specifically written in simple-to-follow linear code without many of the tricks of the trade involved, so that with a cursory knowlege of the 6502 assembler, you should be able to follow and successfully modify the 560Z operating system to suit your needs. Third, Carefully study the 560Z circuits and try to understand what each IC is present for. Look up the PIA in a Motorola M6800 Manual and be sure you understand how it works. Carefully study the 6100 and Z-80 descriptions and review some of the larger medium-scale pieces on the board, particularly the operation of 8T26s, 74153s, and 74157s.

You should now be ready to start on the board. First build up the four PIAs just as an accessory board to your existing OSI system. Then test these PIAs in conjunction with the procedures outlined in the 560Z software description. Become somewhat familiar with the 560Z monitor so that you can set any PIA lines that you desire to input or output high or low. Next set up a 4K porthole . Perform both manual and automatic operations via the monitor so that you can understand how the porthole works.

Now you should be ready to attack the microprocessors. First install the Z-80 following the instructions on both assembly and testing and finally bring up the Intersil 6100. At this point, you should have understanding of the 560Z system capabilities and should be able to proceed with the emulation of the PDP-8 and common 8080 systems as well as devising and executing original experimentation in computer architecture. The Ohio Scientific's Small Systems Journal will carry regular articles on the 560Z subsystem.

CONFIGURING YOUR 560Z SYSTEM

The 560Z system minimally requires a standard Ohio Scientific system utilizing a 400, 500, or 510 CPU Board with a 6502 microprocessor and at least 4K of main memory. The 560Z Board is then plugged into the MOS bus. Then a minimum of 4K of memory must be added to the system side of the bus, which should also include pull-up resistors so that a full or partial backplane board should be utilized. This minimal system can be made to work with either video or serial port on the MOS side. However, the system we recommend is the system that we use in development work with the 560Z system, that is, a Challenger II disk system with a minimum of 16K RAM on the MOS bus side, and a minimum of 4K x 12 on the system bus side. This Challenger II system should be equipped with disk, and can have any kind of terminal as its console terminal. However, we we recommend that the the console terminal operate at a high baud rate, such as our video display, or 2400 baud or higher CRT terminal; and that the system should also have a teletype port on a 430A or 430B, located at FBXX. The use of teletype here may seem strange, but makes sense, because almost all PDP-8 software is written for teletype systems and comes from the Decus Library on conventional paper tape. Thus the system that we primarily support is a 16K Challenger disk system with an additional teletype port on a 430A or 430B Board, a 560Z, and 4K x 12 RAM on the system bus side of the 560Z. The software provided here is designed to operate on this system under the OS-65D Disk Operating System. The software listed in this manual is provided in object code form on all OS-65D diskettes, Version 2.0 and higher, for your convenience. Therefore if you have a disk system with a 2.0 disk, or newer, the software is already on the disk, eliminating your need to enter it. If you do not have or wish to use a teletype, we strongly urge you to use an additional port such as a video terminal as your control terminal for the 560Z. It can be very confusing and difficult to use the same terminal for system control functions as well as the 560 Board, particularly since you have two independent machines running when the 560Z is up. The next-best thing to a teletype would be a video board or RS-232 port and a high-speed paper tape reader since you would need some way to get standard Decus Library PDP-8 programs into your machine, if you desire to use the 6100. This will require that you modify the input and output routines in the 560Z

MOUNTING

The 560Z Board can be mounted in slot No. 1 of a Challenger system where the floppy disk normally goes. You then have room for an additional small backplane board to be plugged into the system side of the 560Z and extending backward into the case. Be careful not to overload the system as there is only enough power to handle eight slots. The 560Z would account for one slot, and any board plugged into its system bus would account for an additional slot. The best way to obtain the backplane for this is to buy a conventional backplane board and shear, nibble, or saw off a portion of the backplane containing two or more slots and pullup resistor area. You would then install normal pullup resistors on this system bus as you would for any OSI system.

There is no real limitation to the number of boards you place on the system bus side except for power and space requirements. You may have any number of boards up to the standard bus limitations including dedicated I/O ports on the system bus side, and up to 65K of RAM which would allow you a world's first (since the normal PDP-8 can have a maximum of 32K). If more than 4K x 12 of memory is anticipated in a normal Challenger configuration, it is recommended that you place the 560Z in the main Challenger case and run its system bus side out via ribbon cables to an additional backplane board in another Challenger case so that you can power as many boards as desired in a 560Z subsystem.

The 560Z, as stated elsewhere, has a fairly open-ended architecture so that many exciting experimental configurations are feasible. For example, it is possible to place a 510 CPU Board on the system bus side of the 560Z as well as have CPU boards on the MOS bus side. This allows interesting configurations; you can, for instance, use two 560Z Boards to tie two 510 systems together with full band-width data channels. 560Z Boards can also be daisy-chained or cascaded and addresses can be strapped so that signals can propagate through levels of 560Z Boards. All of this is fairly esoteric, but it is directly possible with virtually no foil cutting or circuit modification, allowing totally new and unexplored computer configurations.

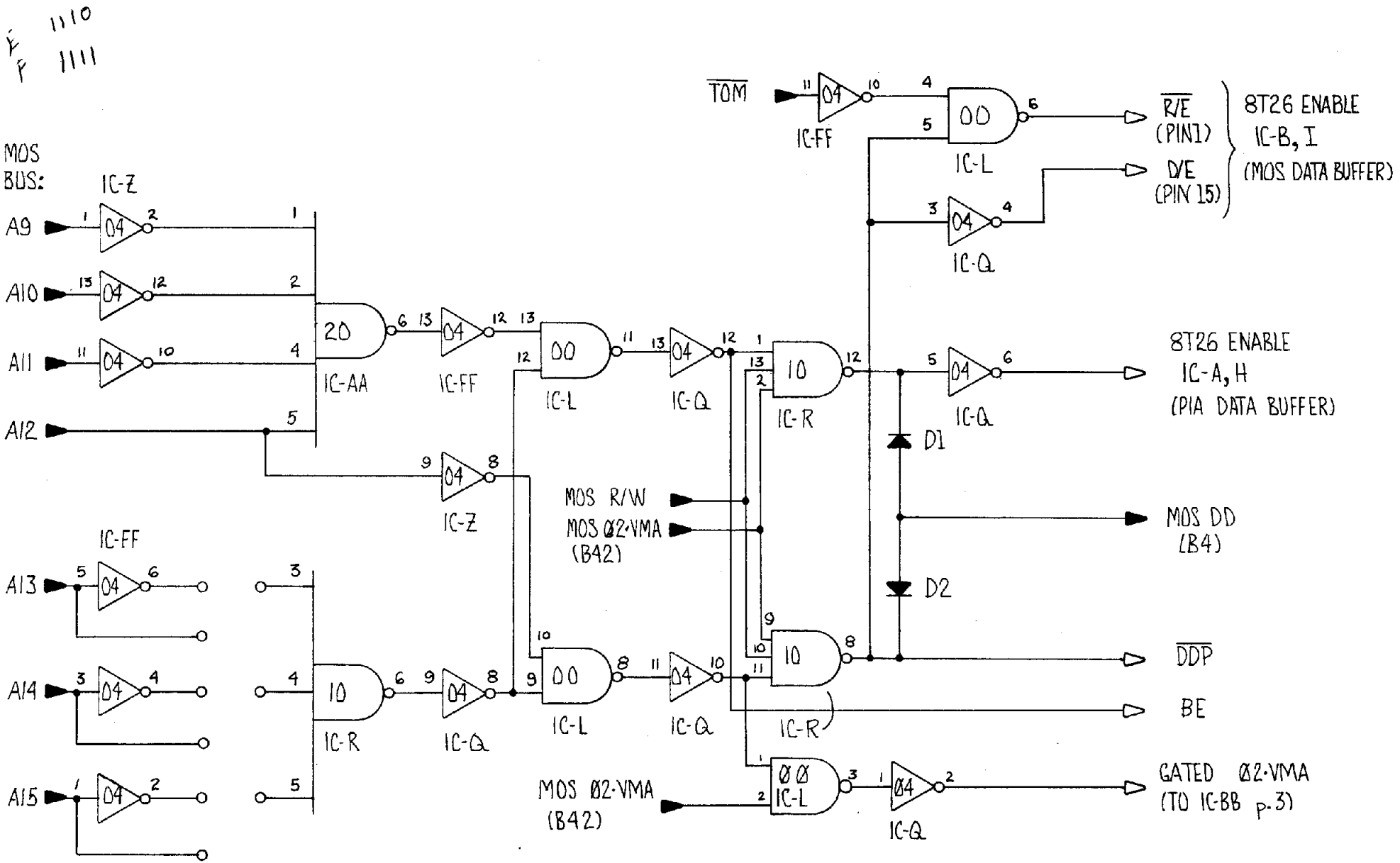
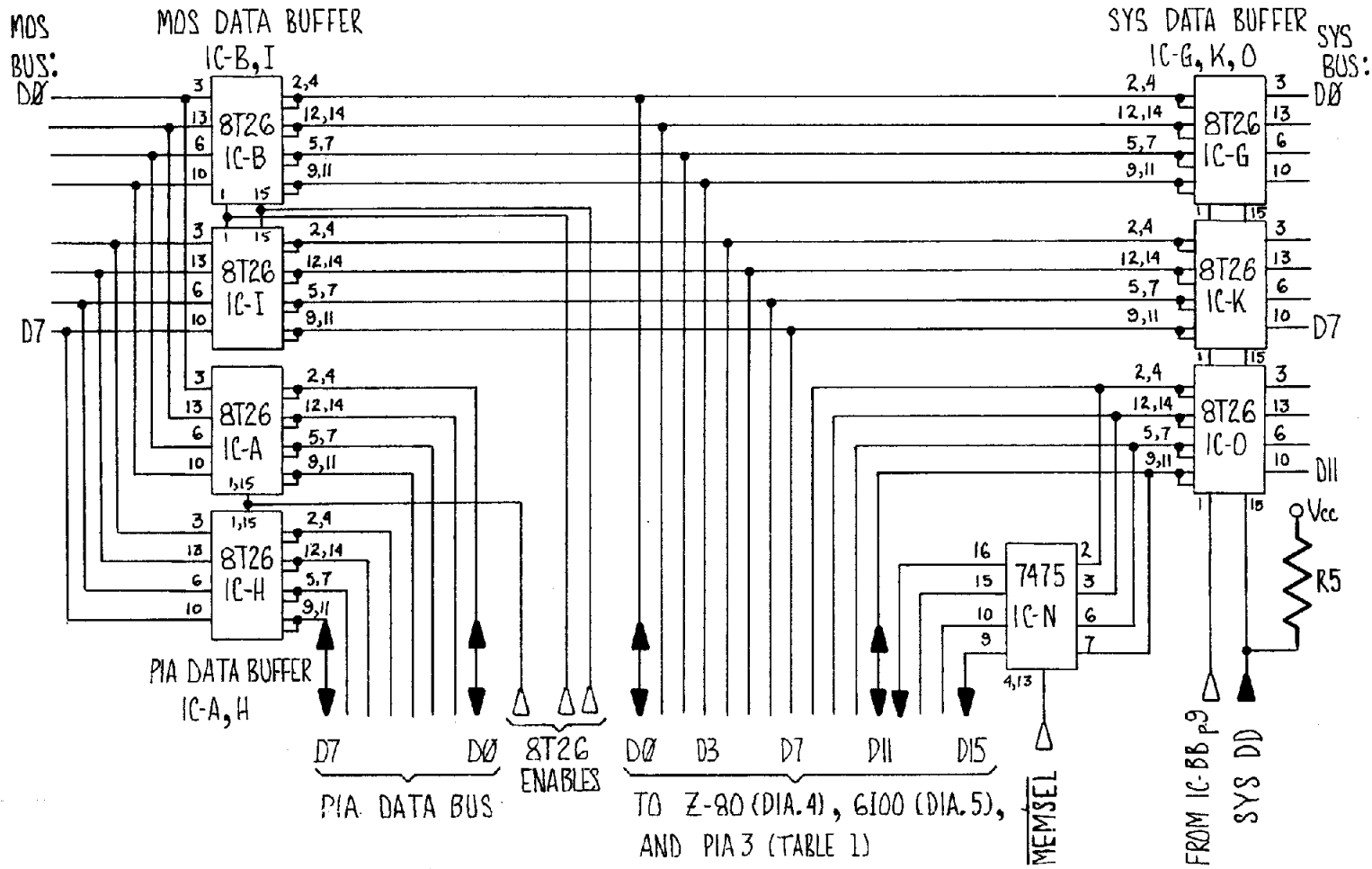


DIAGRAM 1 - ADDRESS DECODE AND ENABLE CIRCUITRY



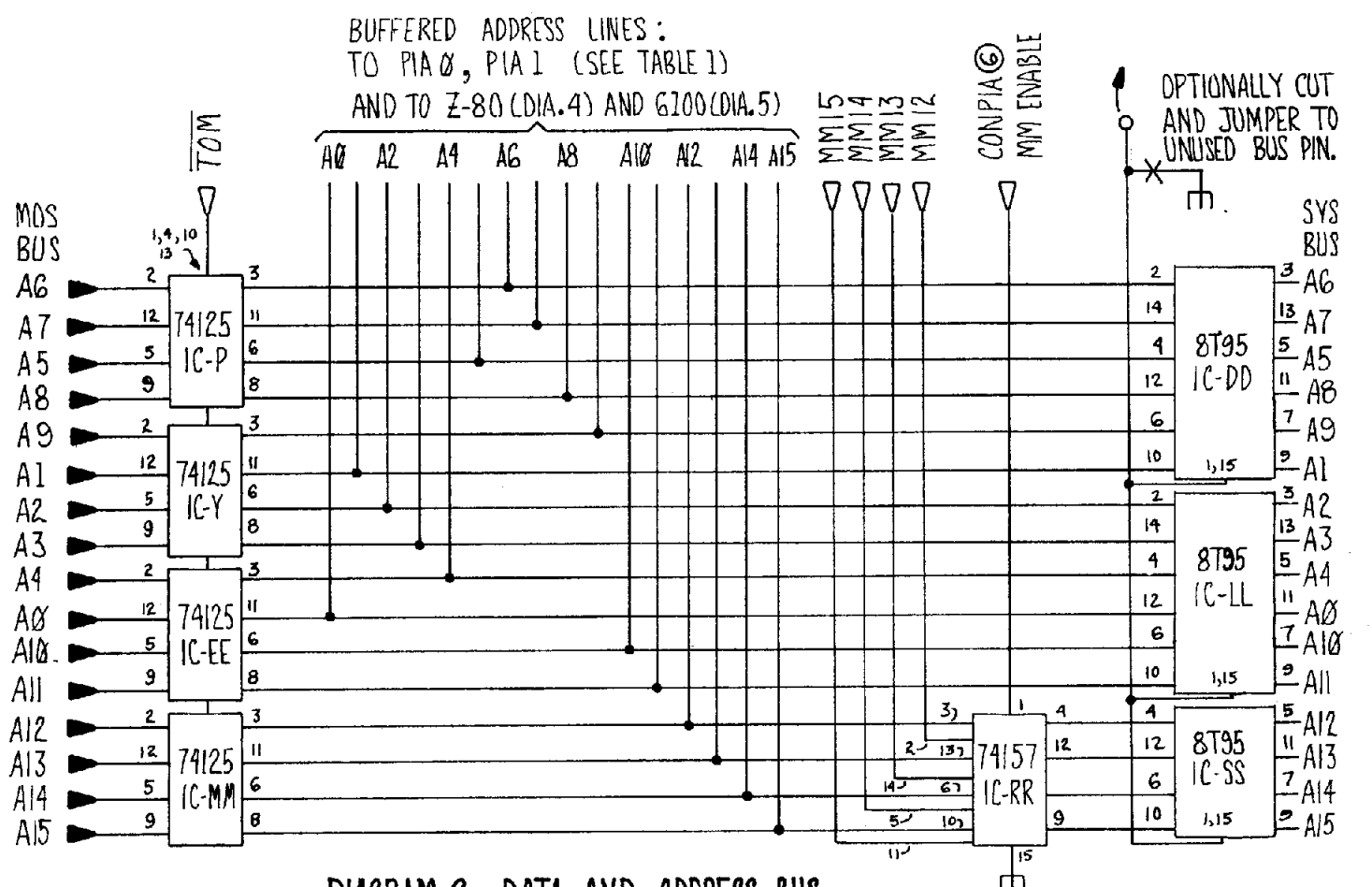
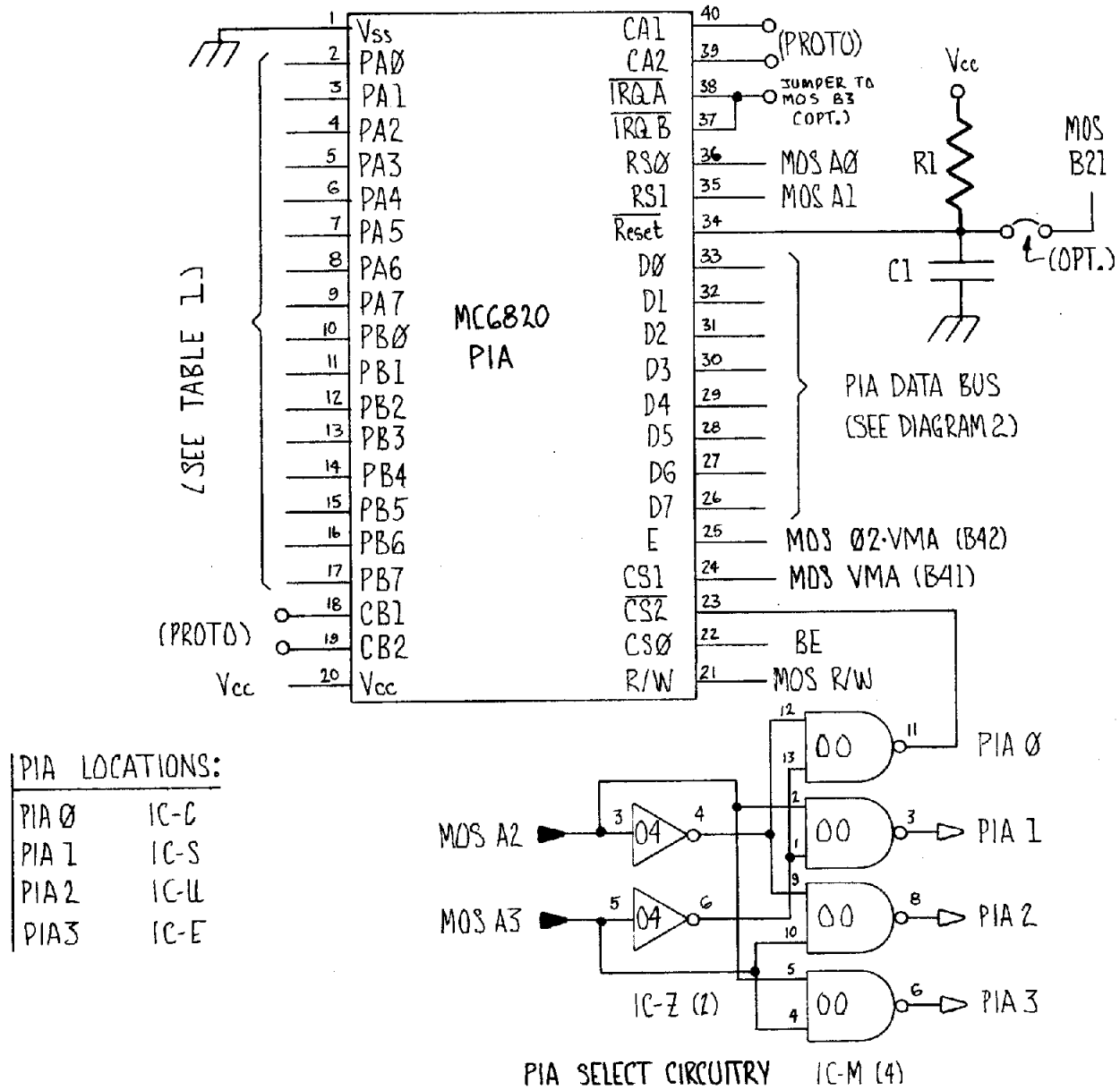


DIAGRAM 2 - DATA AND ADDRESS BUS



PIA LOCATIONS:	
PIA Ø	IC-C
PIA 1	IC-S
PIA 2	IC-U
PIA 3	IC-E

DIAGRAM 3 - PIA IMPLEMENTATION

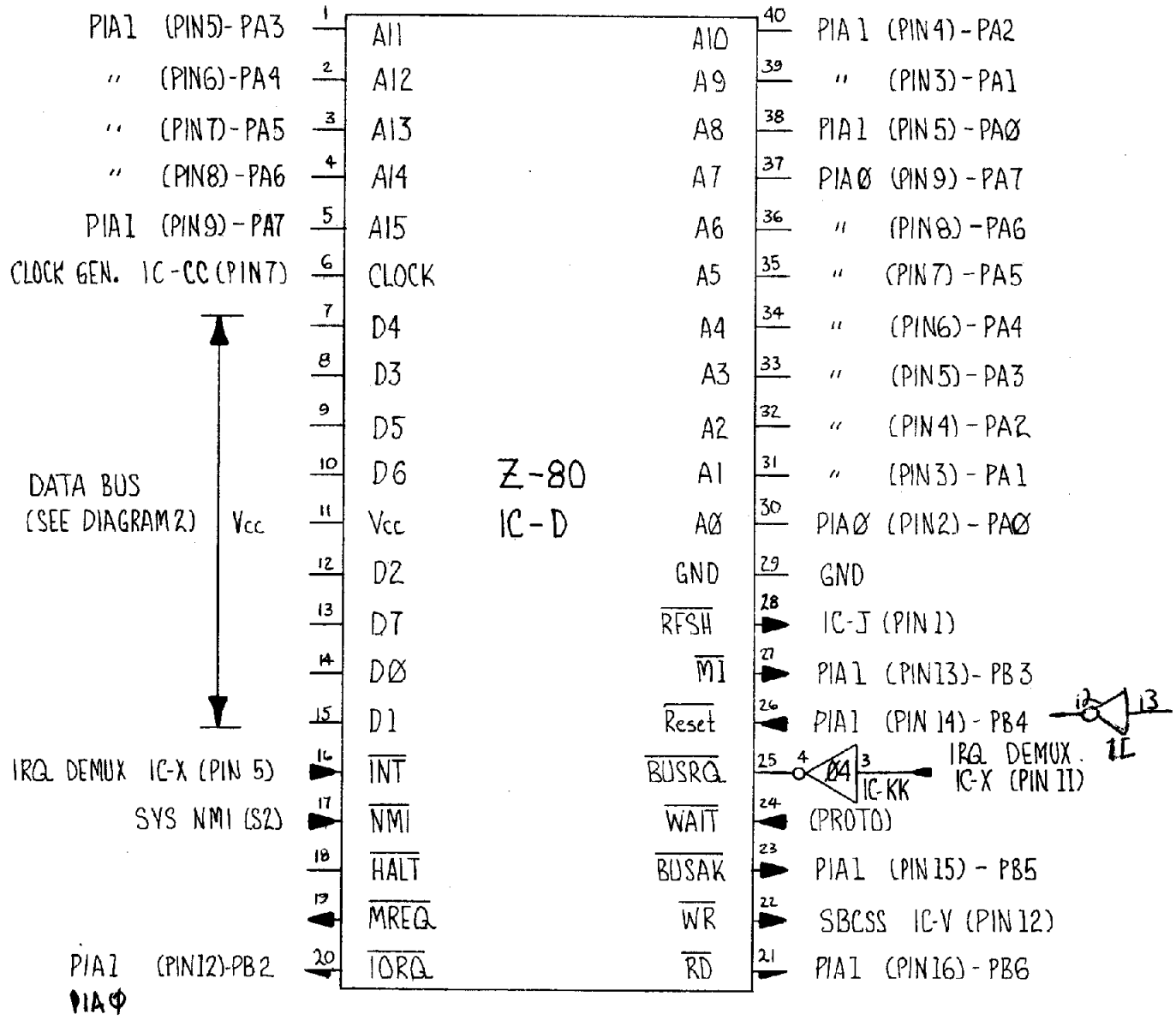
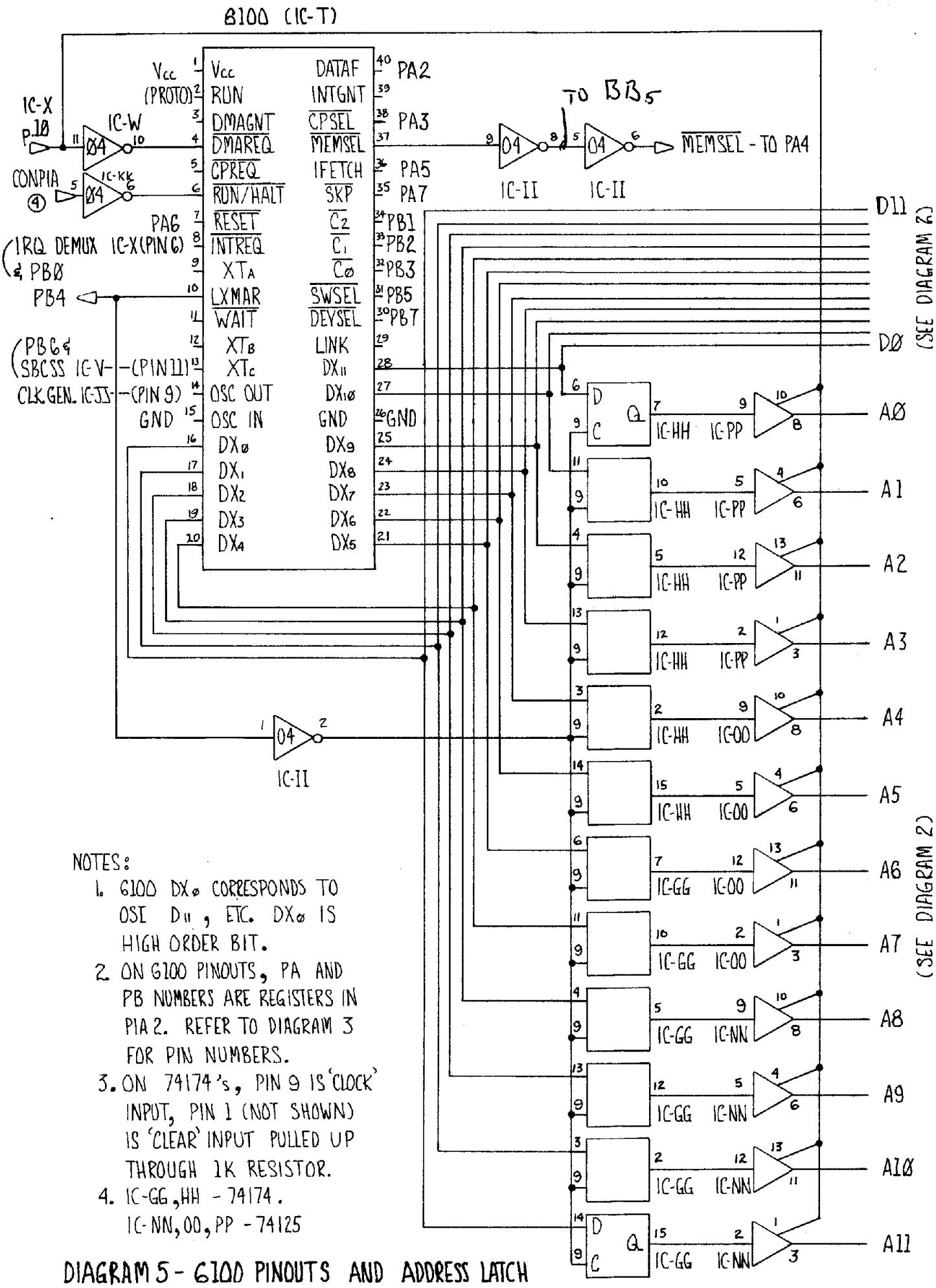


DIAGRAM 4- Z-80 PINOUTS



NOTES:

1. 6100 DXØ CORRESPONDS TO OSI D11, ETC. DXØ IS HIGH ORDER BIT.
2. ON 6100 PINOUTS, PA AND PB NUMBERS ARE REGISTERS IN PIA 2. REFER TO DIAGRAM 3 FOR PIN NUMBERS.
3. ON 74174's, PIN 9 IS 'CLOCK' INPUT, PIN 1 (NOT SHOWN) IS 'CLEAR' INPUT PULLED UP THROUGH 1K RESISTOR.
4. IC-GG, HH - 74174. IC-NN, 00, PP - 74125

DIAGRAM 5 - 6100 PINOUTS AND ADDRESS LATCH

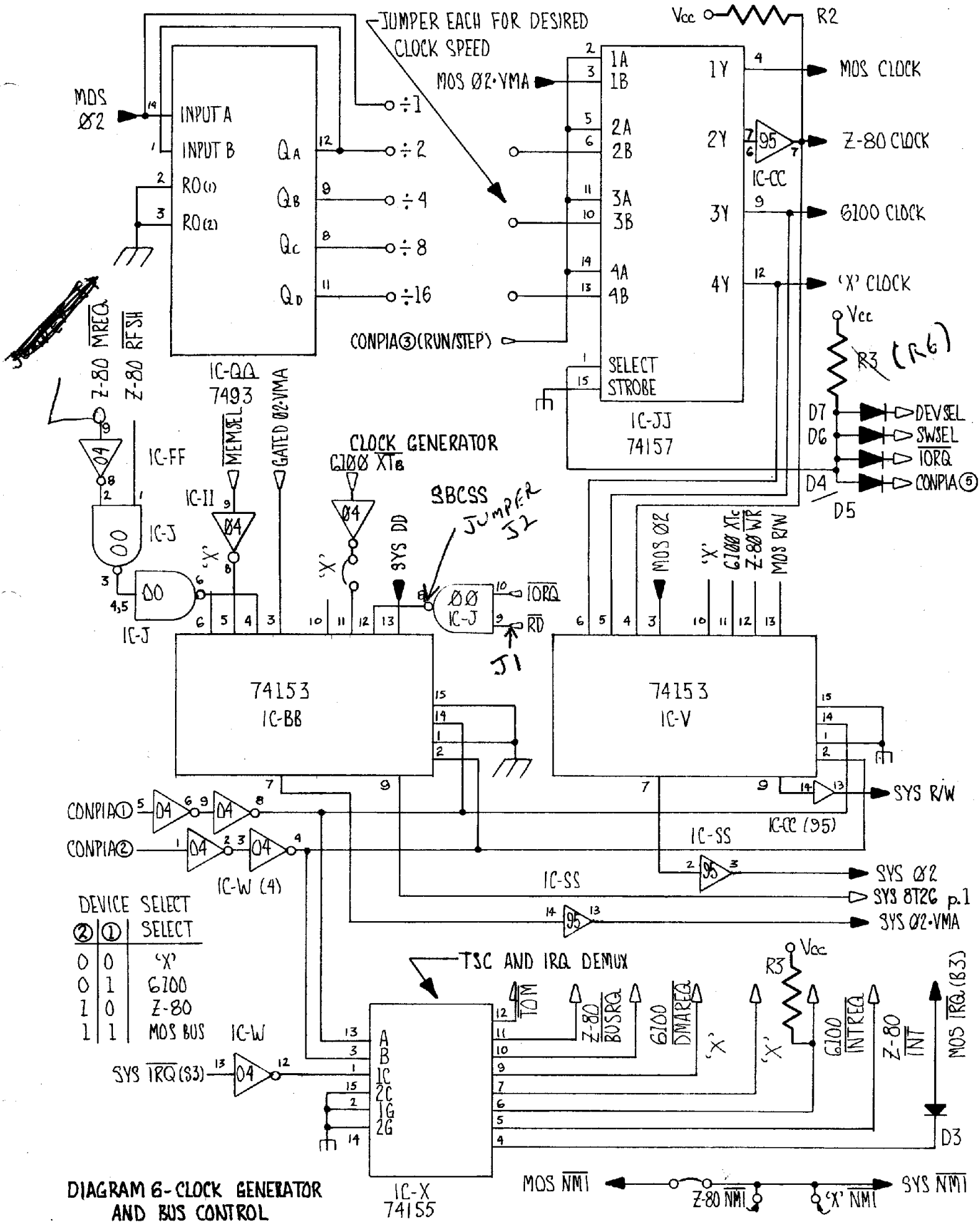
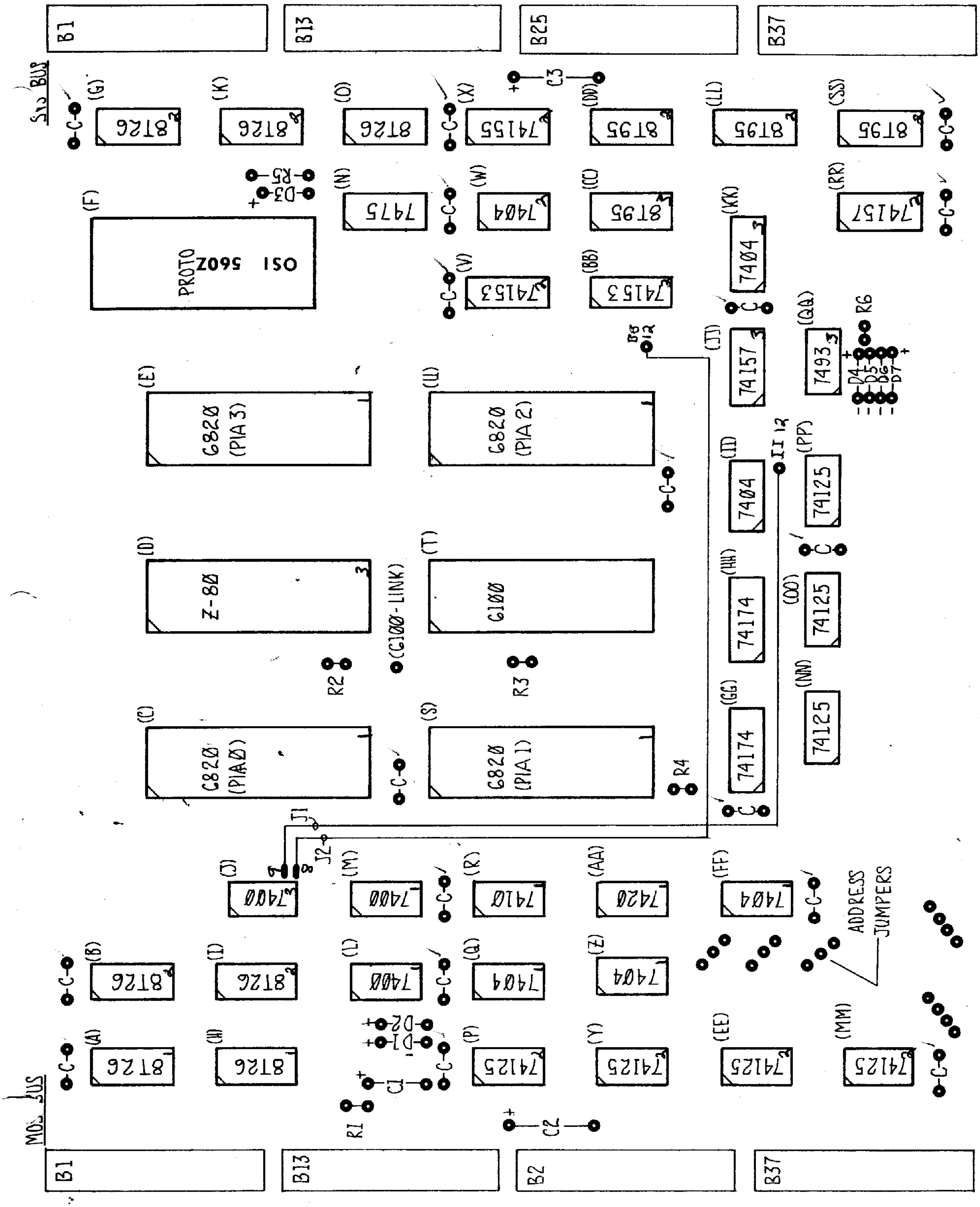


DIAGRAM 6-CLOCK GENERATOR AND BUS CONTROL

PIA Pin #		PIAØ IC-C	PIA1 IC-S	PIA2 IC-U	PIA3 IC-E
2	PAØ	AØ	A8	$\overline{WR}$ p.22 (Z-80)	DA8
3	PA1	A1	A9	$\overline{MREQ}$ p.19 (Z-80)	DA9
4	PA2	A2	A10	DATAF p.40	DAIØ
5	PA3	A3	A11	$\overline{CPSEL}$ p.38	DA11
6	PA4	A4	A12	$\overline{MEMSEL}$ p.37	DA12
7	PA5	A5	A13	IFETCH p.36	DA13
8	PA6	A6	A14	$\overline{RESET}$ p.7	DA14
9	PA7	A7	A15	$\overline{SKP}$ p.35	DA15
10	PBØ	Z-80 $\overline{RFSH}$ p.10	DØ	*$\overline{INTREQ}$	*PROCESSOR SELECT (CONPIA1)
11	PB1	Z-80 $\overline{INT}$ p. 11	D1	$\overline{C}_2$	*PROCESSOR SELECT (CONPIA2)
12	PB2	Z-80 $\overline{IORQ}$ p. 12	D2	$\overline{C}_1$	CLOCK MODE SELECT (RUN/STEP) (CONPIA3)
13	PB3	MM STROBE (CONPIA6)	D3	$\overline{C}_Ø$	6100 $\overline{RUN/HALT}$ (CONPIA4)
14	PB4	MM12	D4	*LXMAR	STEP (CONPIA5)
15	PB5	MM13	D5	$\overline{SWSEL}$	$\overline{M}_1$ p.27 (Z-80)
16	PB6	MM14	D6	XTC	$\overline{RESET}$ p.26 (Z-80)
17	PB7	MM15	D7	$\overline{DEVSEL}$	$\overline{BUSAK}$ p.23 (Z-80)

*See Selection Table, Diagram 6.


B1
B13
B25
B37
SYS BUS
8T26
74155
8T95
74157
7475
7404
74153
OSI 560Z
PROTO
6820 (PIA 3)
6820 (PIA 2)
Z-80
6100
6100-LINK
6820 (PIA 0)
6820 (PIA 1)
7493
74125
74174
7400
7410
7420
74125
ADDRESS JUMPERS
R1
R2
R3
R4
R5
R6
C1
C2
C3
D1
D2
D3
D4
D5
D6
D7
J1
J2
MOD BUS
B1
B13
B2
B37

560Z Construction Notes

The Model 560Z should be constructed in four complete sections. These are:

I. PIAs
II. 4K Porthole
III. Z-80
IV. 6100

All steps must be done in order, except that either III or IV may be omitted. Complete each step and thoroughly test it before proceeding to the next step.

We strongly recommend socketing all parts. This will aid immensely in troubleshooting and bringing up the board. As a minimum, the four PIAs and the microprocessors should be socketed.

The first and most important construction step is to inspect the board thoroughly! The Model 560Z is a dense board; narrow traces and close spacings sometimes result in broken runs or foil bridges between runs. These can be seen most easily by backlighting the board with a strong source such as a desk lamp, and inspecting both sides of the board carefully. Foil bridges can be scratched out with a sharp razor; open runs can be touched up with a soldering iron. Be thorough! A few extra minutes now may save an hour of troubleshooting and a blown part later.

Prototyping area is available in the upper right hand corner of the board. It will support one 40-pin microprocessor, or two 16-pin packages for custom modifications to the existing circuitry. A number of unused gates are located in various places; consult the table on page 23.

Connect together the power bus lines on both backplanes. Using 18-gauge wire, jumper +5V, -9V, and ground across the back of the board.

I. PIA

1. Install eight Molex connectors, four each along the right and left board edges.
2. (DATA BUFFERS) Install (diagram 1 + 2)

IC-A	8T26
H	8T26
L	7400
Q	7404
R	7410
Z	7404
AA	7420
FF	7404

Install diode D1.

3. (PIA) Install (diagram 3)

IC-M	7400
IC-C	PIA
E	PIA
S	PIA
U	PIA

R1/C1 form an automatic power-on reset circuit. This may be omitted in lieu of connection to a system reset line on the backplane. One of the bus lines must be dedicated as a master reset line (preferably B17). Then install jumper J3. Install capacitors C2 and C3.

4. Install seven bypass capacitors near the circuits just installed.

II. Porthole (diagrams 1 + 2)

1. (Buffers) Install

IC-B	8T26
I	8T26
G	8T26
K	8T26
P	74125
Y	74125
EE	74125
MM	74125
DD	8T95
LL	8T95
SS	8T95
RR	74157

Install diode D2 and R5.

2. (Control Logic) Install

IC-V	74153
BB	74153
W	7404
X	74155

3. Install the remainder of the bypass capacitors (11 capacitors).

III. Z-80 (diagram 4)

1. (Control Logic) Install

IC-J	7400
CC	8T95
JJ	74157
KK	7404
QQ	7493
II	J7404 PIN 12,13

Install diodes D3, D4, D5, and R2 and R6. Install jumper J1 from IC-J pin 9 to the pad near IC-BB which runs to IC-BB pin 12. Install jumper J2 from IC-J pin 8 to the pad below IC-II which runs to IC-II pin 12. JUMPER DATA DOES NOT AGREE WITH DIAGRAM SEE DIAG #6

2. Install clock jumper (see diagram 6) from IC-QQ to IC-JJ, pin 6. Consult diagram 6, and select the appropriate pin of IC-QQ for the desired clock speed. TRACE EXISTS FOR ÷R1 SPEED

3. (Microprocessor) Install the Z-80 at IC-D

IV. 6100 (diagram 5)

1. (Control Logic) Install

IC-II	7404
IC-KK	7404

Install R3, R4, and diodes D6, D7.

2. (Buffers) Install

IC-GG	74174
HH	74174
NN	74125
OO	74125
PP	74125
O	8T26
N	7475

3. (Clock) (see diagram 6) Install a jumper from IC-QQ to IC-JJ, pin 10. Consult diagram 6, and select the appropriate pin of IC-QQ for the desired clock speed. CIRCUIT TRACE EXIST FOR ÷1 CLOCK
4. (Microprocessor) Install the 6100 at IC-T.

Table of Unused Gates

Gate	Location	Pins
7400	IC-J	11, 12, 13
7404	IC-II	3, 4
	IC-KK	1, 2
		8, 9
		10, 11
		12, 13
7420	IC-AA	8, 9, 10, 12, 13
8T95	IC-CC	2, 3
		4, 5
		9, 10
		11, 12

OHIO SCIENTIFIC
11679 HAYDEN STREET HIRAM, OHIO 44234

ALL CAPS ARE .1

ALL DIODES ARE 1N914

ALL R'S ARE 4.7K

560Z SOFTWARE PACKAGE

Software for the 560Z Consists of Three Parts

3200 - 33D3 Z-80 I/O Handler
3400 - 37F2 6100 IOT, Interrupt, & Switch Handler
3800 - 3FE9 Main Driver and Subroutines

Initializing Start address is 3800
Non-Initializing Restart Address is 382F

Initialization Code Does the Following:

- Clears trace flag
- Sets high four bits (H command)
- Initializes all PIAs to inputs and low sets predefined states*
- Pulses CKLN 100 times to clear out processors

Available Commands

The Commands Below Employ the Following Notation:

<nnnn> is a 4-digit hex number (address)
<n> is a 1-digit hex number (high four bits field)
<L> is a 1-digit number from 0 to 3 (PIA number)
<oooooo> is a 6-digit octal number (address)
<oooo> is a 4-digit octal number (data)
<o> is a 1-digit octal number (PIA line)
<side> is an A or B (PIA side)
<function> is an H, I, or L, (high, input, or low)

- Commands are not followed by a <return>.
- If you make an error a ?? will be typed.
- All the move instructions are inclusive in their move limits.

B<nnnn> Bin load a paper tape from the UART at FB0X. <nnnn> is the start of an 8K buffer for the uncompressed twelve-bit format data. A bin tape loaded in this manner can be moved into the I6100 address space with the MI command. The uncompressed format :
at <load address #2>+<nnnn> is <high four bits of data>,
at <load address #2>+<nnnn>+1 is <low eight bits of data>, right justified.
This bin loader is applied only on 4K loads without field changes. If a field change is attempted, FLD is typed out and the loader stops.

C Clear all PIAs. They are all initialized as inputs. CKLN is pulsed 100 times, as in initialization. Also, several predefined states are set.*

E Exit to OS-65D (location 2500)

F<nnnn> Fetch an indirect memory resident command file at <nnnn>. Indirect files should end with an X command.

GI<oooo> Start I6100 running at location <oooo>. High four bits definning the field (4K Block) must be set with an H command. Reset is done by placing <oooo> at location 7776 and a 5776 at location 7777, COMPIA2 is brought low, 6100 reset is brought low, INTREQ is brought high, and the 6100 run/halt toggle is brought low and then high.

GZ<nnnn> Start the Z-80 running at location <nnnn>. Reset is done by placing a C3 at location ØØØØ, the low address byte of <nnnn> at ØØØ1, and the high address byte of <nnnn> at ØØØ2, setting CONPIA1 low, and Z-80 reset low.

H <n> Set high four bits of memory management multiplexer to <n> when a GI command is done. This sets the 4K field which the I6100 will use when set running .This command does not affect the PIA until a GI command is executed.

I Go to the I6100 IOT, Interrupt, and Switch Register handler located at 34ØØ. The only way to exit the IOT handler is to reset the computer and restart this program. If the I6100 is running and not executing IOTs or interrupts when you reset, reentry of this driver package at 382F will not interrupt it. You may halt a free-running I6100 or Z-80 whith the C Command. I/O is passed to the UART at FBØX.

KH Set CKLN as an output and output a high. This effectively sets Ø2 high on the 560Z Board. This and the KL Command are used for manually clocking a processor.

KL Same as KH, but outputs a low.

L<nnnn> Set to learn mode. Everyting you type will be placed in memory at location <nnnn> upwards. This is used to generate an indirect command file accessible by an F Command. <Return>s may be typed now to format commands, but are not stored. A $ typed at any time will exit you to Command Mode again. The number typed is the location where the next character would have gone. It is effectively the end of the file. Command files should have an X Command as their last command so that when executed, one returns to Command Mode.

MF$\langle nnnn\rangle_1=\langle nnnn\rangle_2,\langle nnnn\rangle_3$ Moves eight bit memory from location $\langle nnnn\rangle_2$ on the 560Z's side of the bus, ending at $\langle nnnn\rangle_3$ to location $\langle nnnn\rangle_1$ on the 6502's bus and upwards. Move from the 560Z side to the 6502 side.

MI$\langle nnnn\rangle_1=\langle nnnn\rangle_2,\langle nnnn\rangle_3$ Moves the uncompressed twelve bit format from the 6502 bus starting at $\langle nnnn\rangle_2$ and ending at $\langle nnnn\rangle_3$ to location $\langle nnnn\rangle_1$ upwards on the 560Z side.

MT$\langle nnnn\rangle_1=\langle nnnn\rangle_2,\langle nnnn\rangle_3$ Opposite of the MF command. Moves from the 6502 side from $\langle nnnn\rangle_2$ to $\langle nnnn\rangle_3$ to the 560Z side starting at $\langle nnnn\rangle_1$.

P Print PIA status table. Prints a 16 line table, two lines for each of the four PIA ports. Every other line is labeled as to which PIA it is referencing (0, 1, 2, or 3) and which side (A or B). The first of each of the two lines is a dump of the Data Direction Register (Øs correspond to lines set as inputs, 1s correspond to lines set as outputs). The second of each eight pairs of lines is the contents of the Data Register. If a PIA bit was set as an output, the corresponding data bit is being outputted. If a PIA bit was set as an input, the corresponding data bit is the status of that input.

R<nnnn> Pulse the clock line (CKLN) with a delay constant of <nnnn>. ØØØØ means run as fast as possible. FFFF is the slowest run. (Ø2 period of 1 sec.)

S<L><Side><o><Function> Set PIA number <L>, Side <Side> bit <o> according the the Command Function . Bits are numbered with the least significant bit as Ø. The following are the possible commands for <F>:

H - Set as an output and set that output high
I - Set as an input
L - Set as an output and set that output low.

Set the I6100 IOT handler's trace flag. Each opcode is pritned as it is handled.

W<oooo> Set the switch register contents to be <oooo>. Switch register value is contained in the IOT handler package. When an OSR is executed, the contents of that location are fed to the I6100.

X Exit the indirect file command mode and return control to the console.

Z Go to the Z-80 I/O Handler at location 32ØØ. This must be exited with a reset much like the I Command, I/O is passed to the UART at FBØX.

<oooooo>/<oooo> Open the twleve bit data location <oooooo> on the 560Z side. The driver responds with the /<oooo> which is the contents of location <oooooo>. A line feed closes this location and opens the next. An open location is subjected to being reopened by changing the contents of the location. This is much like the # command for the Extended Monitor.

* Predefined States on Start or C Command. The following line are set outputs high:

(Z80)INT	(ØB1)	CONPIA1	(3BØ)
MMENAB	(ØB3)	CONPIA2	(3B1)
IGIRST	(3B3)	RUNSTP	(3B4)
INIREQ	(2BØ)	ICIPHT	(2A6)

560Z BOARD TESTING WITH SOFTWARE PACKAGE

1. Testing PIA Ports. Once the board is populated enough for the PIAs to be functional, it is a good idea to test a few PIA lines on each side of each of the four PIAs.

Start up the utility driver and set MMENAB low (SØB3L). Then, set up the high address on MM_{12}, MM_{13}, MM_{14}, and MM_{15}. Most likely, one would set them all low to access the first 4K on the 560Z Bus. Exit the driver package with the E Command. Enter a Monitor (The Extended Monitor or a PROM Monitor). The 4K block which you have selected with MM_{12}, MM_{13}, MM_{14}, and MM_{15} should be visible through the 4K window at EXXX. The first byte of the 4K block is at location EØØØ, the second at EØØ1, and so on. Try writing and reading what you wrote to verify that the memory is really there. Be sure that the memory has been pre-tested before you use it on the 560Z Bus.

Once the program has been loaded and started, a P command should yield all PIAs set as inputs and various different bit patterns being inputted. Try setting some bits as outputs low and outputs high and see if those settings were made properly by examining the status of the PIAs with the P command.

If the board passes these initial tests, then use a voltmeter or a scope to examine several PIA lines as they are toggled low and high. Check at least a couple bits on each side of each PIA.

2. Accessing Memory through the 4K portal using the memory management multiplexer. MMENAB line to PIA Ø,Side B , Bit 3 controls the memory management multiplexer.

MMENAB- low - the high four bits of an address on the 560Z's bus come from MM12, MM13, MM14, and MM15.

MMENAB - high - The high four bits of an address on the 560Z's Bus simply pass through from the 6502's Bus.

3. Running a Z-80 Program Manually

a. Clear all PIAs with a C Command.
b. Set MMENAB low (SØB3L) so we can set up a 4K block of memory to view through the 4K window.
c. Set MM12, MM13, MM14, and MM15 low. (SØB4L, SØB5L, SØB6L, SØB7L)
d. Exit the driver with an E Command and enter the Extended Monitor on your PROM Monitor.
e. Place a C3 (Hex) at location EØØØ, the low-byte starting address at EØØ1, and the high bytes starting address at EØØ2. This sets up a jump instruction to your program since the Z-80 on reset starts executing at location ØØØØ. If your program resides also in the first 4K block of memory, enter it at this time. Otherwise, restart the driver package and set up the high four bits, exit, and use a Monitor to enter the program. Understand that you must write data into EØØØ to EFFF to access the proper 4K block.
f. Restart the driver at 38ØØ. (This clears all PIAs)
g. Set CONPIA1 low (S3BØL)
h. Set Z9ØRST low (S3B6L)
i. The Z-80 should now be running.
j. To halt it, set Z8ORST high (S3B6H) and then clear the PIAs with a C Command.

A Z-80 Test Program

Location	Contents						
					*=2ØØ		
2ØØ	3E	45		START	LA	#45	
2Ø2	32	ØD	Ø2		L	R1	R1←45
2Ø5	E6	ØF			AND	#F	
2Ø7	32	ØE	Ø2		L	R2	R2←5
2ØA	C3	ØA	Ø2	LOOP	JP	LOOP	
2ØD	ØØ			R1	.BYTE	Ø	
2ØE	ØØ			R2	.BYTE	Ø	
					.END		

If this program runs successfully, a 45 will be placed at location 2ØD and a 5 will be placed at location 2ØE.

TEST Z8Ø I/O

```
2ØØ - DB  Ø1       START  IN  1
  2   CB  1F              RR  A
  4   D2  ØØ Ø2           JNC START
  7   DB  ØØ              IN Ø
  9   F5                  PUSH AF
  A   DB  Ø1       WAIT   IN 1
  C   CB  1F              RR   A
  E   CB  1F              RR   A
 1Ø   D2  ØA Ø2           JNC  WAIT
 13   F1                  POP  AF
 14   D3  1Ø              OUT  1Ø
216   C3  ØØ Ø2           JP START
 19
```

Port Ø - Input Data
Port 1 - Input/Output Flags

Bit Ø high keyboard data ready
Bit 1 high printer ready for data

Port 1Ø Output Data

THE Z8Ø I/O HANDLER

The program which is supplied handles the following port assignments:

Port Ø - Teletype input. Read this location for data when bit Ø of port 1 goes high.

Port 1 - Bit Ø high means input data is ready and available at port Ø. Bit 1 high means port 1Ø is ready to to receive new output data.

Port 1Ø - Teletype output. Write data to be printed to this port when bit 1 of port 1 goes high.

Additional ports may be decoded (not all unassigned ports pass valid data). Simply replace the zero in the location 32C8 + Port number with an offset which, when multiplied by 3 and added to 33C8 will point to a jump instruction to a subroutine to handle it. Since ports may be used both as inputs and outputs a single port handler will have to look at the value of location 26 on page Ø. A zero means an OUT instruction, non-zero implies an IN instruction. The example shows I/O routed through the UART at FBØX. This could easily be patched to use an I/O device other than the UART for Teletype I/O.

Note: Patch area is located from 317E ——> 31FF, 33D4 ——>33FF, 37F3——>37FF, and 3FEA——>(end of memory).

4. Running an I6100 Program Manually

a. Load the program into memory via the # Command
b. Load the starting location into location 7776 in the 4K block where the program executes, and a 5776 into location 7777 in the same 4K block. This sets up the reset start.
c. Type the C Command to clear all PIAs.
d. Set MMENAB low. (SØB3L)
e. Set up the high four bits of the memory management (MM12, MM13, MM14, and MM15). For example:

SØB4L
SØB5H
SØB6L
SØB7L

Sets the high four bits to ØØ1Ø which selects the third 4K block.

f. Set CONPIA2 low (S3B1L)
g. Set I61RST low (S3B3L)
h. Set I61RHT(CONPIA4) low, then high (S2A6L S2A6H)
i. The I6100 should now be running. You can halt if by repeating step h which sets the run/halt toggle in the I6100 to halt. To continue, repeat step h.
j. To stop the I6100 halt it as shown in step i. then clear all PIAs with the C Command.

A Test I61ØØ Program

This test requires only a 4K by twelve-bit 420 Memory Board on the 560Z bus. If run successfully, it will place an octal 5777 in location Ø211 and a 1 in location Ø21Ø. On reset the I6100 starts executing at location 7777. Thus, our general practice will be to place an indirect jump at that location. So, to

Location	Contents			
			*=2ØØ	
2ØØ	72Ø1	START,	CLA IAC	/ Set AC=1
2Ø1	321Ø		DCA R1	/ RL←1
2Ø2	7Ø4Ø		CMA	/ Set AC=7777
2Ø3	7112		CLL RTR	/ Shift right twice
2Ø4	3211		DCA R2	/ R2←5777
2Ø5	751Ø		SPA	
2Ø6	52Ø6		JMP .	/ Infinite loop 1
2Ø7	52Ø7		JMP .	/ Infinite loop 2
21Ø	ØØØØ	R1,	Ø	
211	ØØØØ	R2,	Ø	
7776	Ø2ØØ	INDIR,	START	
7777	5776		JMP I INDIR	/ Restart jump

start any program, place the starting address at location 7776 and the instruction 5776 at location 7777.

Test I/O Program for I6100

```
Location    Data      / Read Character Add 1 Echo it
                      *200
 000200     6046     START    TLS              Clear print buffer
    201     6032              KCC              Clear flag
    202     6031              KSF              Wait
    203     5202              JMP     .-1
    204     6036              KRB              Read
    205     1212              TAD     ONE      Add 1

    206     6041              TSF              Wait
    207     5206              JMP     .-1
    210     6046              TLS              Print it
    211     5201              JMP     START+1  Loop Back
    212     0001     ONE      1                Constant
```

Restart Jump

7776 - 0200
7777 - 5776

/ Interrupt Test for 6100

```
                      *=0
  0  0000     INT,    0
  1  5240             JMP SERV
```

/ Start + Wait Loop

```
                      *=30
 30  6001     START,  ION
 31  5231             JMP START+1
                      *=40
 40  6031     SERV,   KSF
 41  7410             SKP
 42  5250             JMP KB
 43  6041             TSF
 44  7410             SKP
 45  5260             JMP TP
 46  5246     INFINT, JMP INFINT
```

/ Device Service

```
                      *=50
 50  6036     KB,     KRB
 51  1271             TAD ONE
 52  3270             DCA BUFFER
 53  6001     EXIT,   ION
 54  5600             JMP I INT
```

```
                    *=6Ø
6Ø  72ØØ     TP,     CLA
61  127Ø             TAD BUFFER
62  745Ø             SNA
63  5253             JMP EXIT
64  6Ø46             TLS
65  72ØØ             CLA
66  327Ø             PCA BUFFER
67  5253             JMP EXIT
                    *=7Ø
7Ø  ØØØØ     BUFFER, Ø
71  ØØØ1     ONE,    1
```

7776 - ØØ3Ø
7777 - 5776

560Z IOT, Interrupt, and Switch Register Handler for I6100

Additions to the IOT Tables

I6100 IOT instructions are of the form:

6 ⟨device number⟩ ⟨device op⟩

The ⟨device number⟩ is a two-digit octal number specifying a unique device. For example, Ø4 is the Teletype printer. The IOT handler obtains this number and the ⟨device op⟩ during the execution of an IOT instruction by the I6100. The first digit of the device number times two is used as an index into the table of device tables. This yields an address of a table, which, when indexed by the second digit of the ⟨device number⟩ times two yields a table of addresses of opcode handlers. This opcode table is indexed by the ⟨device op⟩ times two to yield an address of a subroutine to handle that particular device operation. For example the IOT instruction 6Ø34 (KRS) has a ⟨device code⟩ of Ø3 and a ⟨device op⟩ of 4. Let us follow through the index tables:

> Ø indexes to location 374B in the table of device tables. This gives us the address 375B. The 3 indexes to location 3761 where we find the address 378B. Finally the 4 indexes to location 3793 where we find the address (36BD) of the KRS instruction handler.

Each instruction handler ends with an RTS which returns us to location 3484. Since there are two parts of an I6100's IOT instruction execution, we again index into tables to see what to do next. The first digit of the ⟨device number⟩ times two indexes into the table at 37C3, yielding an address of the table which is indexed by the second digit. We then go to that location to handle the second part of an IOT. Normally most instructions will wait for XTC to go low, then the I6100 is set to run free.

Loading and Running FOCAL Off Paper Tape

1. Load and start up the driver package.
2. Type a BØ2ØØ. This calls up the bin loader to load a tape from the UART. The loader makes a modification to OS-65D to enable it to get full eight-bit data (i.e., not masked to 7 bits). If one were to reset while bin

loading, a 7F would have to be placed in location 23ØC before OS-65D, or the driver package would work.

Start reading the tape in, the loader will return when it senses the trailer tape (channel 8 punches). FOCAL consists, usually, of 2 tapes together; the first part is FOCAL and the second part is its initialization program. Thus, continue the bin loader with BØ2ØØ again to load the second part in over the first.

3. Exit with an E Command. Save the block of memory from Ø2ØØ to 21FF on disk. For example:

```
S40,1=Ø2ØØ/C    Ø2ØØ/C
S41,1=ØEØØ/A    ØEØØ/A
S42,1-18ØØ/A    18ØØ/A
```

4. Return to the 560Z driver package and do a MIØØØØ=Ø2ØØ,21FF Command. This moves FOCAL into the 560Z's memory (twelve-bit memory).
5. Clear all the PIAs with a C Command.
6. Set the switch registers to 6ØØØ with the W Command. (Woooo)
7. Set the high four bits with the H Command (HØ)
8. Then start FOCAL at Ø2ØØ with GIØ2ØØ.
9. Execute an I Command to handle the IOTs and interrupts. Be sure the IOT handler package is loaded at 34ØØ. All I/O is passed to the UART at FBØX by the IOT handler.

Hand Clocking a Processor (I6100) / (Z-80)

Follow the same procedure you use when running a I6100 manually up to step 8. Just after step 8, use the following steps:

9. Set RUNSTP low. (S3B4L). This sets the clock multiplexer into single step mode. Ø2 comes from CKLN rather than the 6502's Ø2.
10. Set I61RHT(CONPIA4) low, then high (S2A6L S2A6H). This starts the I6100.
11. To single-step the I6100 you must provide the Ø2 clock with the KL and KH Commands. Ø2 is generated by typing alternating commands of KL and KH or the R Command. At any time you may view the address or data lines with the P Command, or examine and change any other lines. Then continue. This permits you a detailed look at a running micro.

The Z-80 can be single-stepped by using the instructions to manually run a Z-80 up to step 6. Just after step 6, use the following steps:

7. Set RUNSTP low. (S3B4L)
8. Set Z8ØRST low. (S3B6L)
9. Clock the Z-80 in the same way as you clocked the I6100 in step 11 above.

PIA Bit Assignment Table in the Utilities Package

On the 560Z board there are 4 PIAs as listed below:

```
PIA Ø at FØØØ
PIA 1 at FØØ4
PIA 2 at FØØ8
PIA 3 at FØØC
```

Note that each PIA has two sides, A and B, each of which has eight bits. Referring to the spec sheets on the PIAs you are using, you will see that each side has two registers, a control register and a data register. The control registers are at odd locations (FØØ1, FØØ3, FØØ5, FØØ7, FØØ9, FØØB, FØØD, and FØØF). The data registers are at the even locations (FØØØ, FØØ2, FØØ4, FØØ6, FØØ8, FØØA, FØØC, and FØØE). In the utilities package each PIA bit (out of the 64 possible) has two bytes which completely identify it. The first is an offset from FØØØ to the data register. We know, of course, that the corresponding control register is 1 location past the data register. For example DEVSEL has an offset of A which means it is on PIA 2, side B.

PIA No.	Side	Offset
Ø	A	Ø
Ø	B	2
1	A	4
1	B	6
2	A	8
2	B	A
3	A	C
3	B	E

The second number is an eight-bit number with the bit set which corresponds to the bit line in that data register. For example CKLN which is in PIA 3, side B has a bit mask byte of 4Ø which is bit 6. Thus the CKLN is on PIA number 3's PB6 line.

Bit Mask	Bit Number
1	Ø
2	1
4	2
8	3
1Ø	4
2Ø	5
4Ø	6
8Ø	7

All 64 lines are uniquely identified on the table at the end of the utilities package. (locations 3F2F to 3FAE). The positional order of the table is important. Changes in the order of the table without proper changes in the software could render the driver useless.

This format for line definition was adopted so that PIA bit assignments could easily be changed. This is useful if one is redefining bit assignments. Note that since some lines can be tristated on some processors, single lines may have more than one function. This feature was not used in implementing just the I6100 and Z-80, for the sake of simplicity.

Example:
Testing PIA Ports

```
A*G3800
Z:P
0A 00000000
   00000000
0B 00001010
   00001111
1A 00000000
   11110000
1B 00000000
   11111111
2A 01000000
   11011011
2B 00000001
   11100001
3A 00000000
   11111111
3B 00011011
   11111011
Z:S0A0H
Z:S0A7L
Z:S1A6L
Z:S1A0H
Z:S1B3H
Z:S1B4L
Z:S1B5H
Z:S3A1L
Z:S3A0H
Z:S3A2L
Z:S3A3H
Z:P
0A 10000001
   00000000
0B 00001010
   00001111
1A 01000001
   11110000
1B 00011000
   11101111
2A 01000000
   11011011
2B 00000001
   10100001
3A 00001001
   11111001
3B 00011011
   11111011
Z:C
Z:E
A*
```

0 = INPUT

Example: Using
4K Data Window

```
A*G3800
Z:S0B3L
Z:S0B4L
Z:S0B5L
Z:S0B6L
Z:S0B7L
Z:E
A*RE
:#E000/45 23 /23
:#E348/00 FF /FF
:#EE01/80 A7 /A7
:#EFFF/24 67 /67
:D
A*
```

Example: Running
Z-80 Manually

```
A*G3800
Z:C
Z:S0B3L
Z:S0B4L
Z:S0B5L
Z:S0B6L
Z:S0B7L
Z:E
A*RE
:#E000/23 C3
:#E001/00 00
:#E002/3E 02
:#E200/DB 3E
:#E201/01 45
:#E202/CB 32
:#E203/1F 0D
:#E204/D2 02
:#E205/00 E6
:#E206/02 0F
:#E207/DB 32
:#E208/00 0E
:#E209/F5 02
:#E20A/DB C3
:#E20B/01 0A
:#E20C/CB 02
:#E20D/1F 00
:#E20E/CB 00
:D
A*G3800
Z:S3B0L
Z:S3B6L
Z:S3B6H
Z:C
Z:S0B3L
Z:S0B4L
Z:S0B5L
Z:S0B6L
Z:S0B7L
Z:E
A*RE
:#E20D/45
:#E20E/05
:D
A*
```

Example: Z-80
In/Out Program

```
A*RE
:#0200/31 DB
:#0201/80 01
:#0202/01 CB
:#0203/0E 1F
:#0204/04 D2
:#0205/21 00
:#0206/00 02
:#0207/D0 DB
:#0208/06 00
:#0209/00 F5
:#020A/36 DB
:#020B/20 01
:#020C/23 CB
:#020D/10 1F
:#020E/FB CB
:#020F/0D 1F
:#0210/20 D2
:#0211/F8 0A
:#0212/21 02
:#0213/00 F1
:#0214/00 D3
:#020R?
:#0215/22 10
:#0216/FE C3
:#0217/00 00
:#0218/18 02
:G3800
Z:MT0200=0200,0218
Z:GZ0200
Z:Z
```

THIS IS INPUT ECHOED BY THE Z80 PROGRAM...

Example: Manually
Running I6100

```
A*G3800
Z:#000200/5674 7201
Z:#000201/1137 3210
Z:#000202/3022 7040
Z:#000203/7001 7112
Z:#000204/3100 3211
Z:#000205/3026 7510
Z:#000206/1226 5206
Z:#000207/3013 5207
Z:#000210/1225 0000
Z:#000211/4551 0000
Z:#007776/7444 0200
Z:#007777/7547 5776
Z:C
Z:S0B3L
Z:S0B4L
Z:S0B5L
Z:S0B6L
Z:S0B7L
Z:S3B1L
Z:S3B3L
Z:S2A6L
Z:S2A6H
Z:S2A6L
Z:S2A6H
Z:C
Z:#000210/0001
Z:#000211/5777
Z:E
A*
```

Example: Using BIN Loader and MI Command

```
A*G3800
Z:#000200/7201
Z:#000201/3203
Z:B0200
Z:MI0000=0200,0R??
Z:MI0000=0200,21FF
Z:#000200/0000
Z:#000201/7300
Z:E
A*
```

Example: Single Stepping the I6100

```
A*G3800
Z:#000200/5567 7201
Z:#000201/3420 3203
Z:#000202/3400 5202
Z:#000203/7777 0000
Z:#007776/0200
Z:#007777/5776
Z:C
Z:S0B3L
Z:S0B4L
Z:S0B5L
Z:S0B6L
Z:S0B7L
Z:S3B1L
Z:S3B3L
Z:S3B4L
Z:S2A6L
Z:S2A6H
Z:KL
Z:KH
Z:KL
Z:KH
Z:KL
Z:KH
Z:KL
Z:KH
Z:KL
Z:KH
Z:KL
Z:KH
Z:KL
Z:KH
Z:KL
Z:KH
Z:R3000
```

Example: I6100 IOT Program

```
A*G3800
Z:#000200/0000 6046
Z:#000201/7300 6032
Z:#000202/1224 6031
Z:#000203/1225 5202
Z:#000204/1226 6036
Z:#000205/1227 1212
Z:#000206/3230 6041
Z:#000207/1225 5206
Z:#000208??
Z:#000210/3224 6046
Z:#000211/1226 5201
Z:#000212/3225 0000
Z:C
Z:W0000
Z:H0

Z:GI0200
Z:I
```

THIS IS BEING ECHOED BY THE I6100 IOT INSTRUCTIONS...

Example: Hand Clocking a Z-80 Program

```
A*G3800
Z:S0B3L
Z:S0B4L
Z:S0B5L
Z:S0B6L
Z:S0B7L
Z:E
A*RE
:#E000/C3
:#E001/00
:#E002/00 07
:#E700/00 3E
:#E701/52 2C
:#E702/00 32
:#E703/3E 08
:#E704/00 07
:#E705/52 C3
:#E706/00 05
:#E707/3E 07
:#E708/00
:G3800
Z:S3B0L
Z:S3B4L
Z:S3B6L
Z:KL
Z:KH
Z:KL
Z:KH
Z:KL
Z:KH
Z:KL
Z:KH
Z:KL
Z:P
0A 00000000
   00000000
0B 00001010
   00001111
1A 00000000
   00000000
1B 00000000
   11111111
2A 01000000
   11011000
2B 00000001
   11100001
3A 00000000
   11111111
3B 01011111
   10001010
Z:KL
Z:KH
Z:
```

Example:
Using FOCAL

```
*WRITE
C-FOCAL,1969

01.10 COMMENT  DEMO PROGRAM IN FOCAL
01.20 SET B=10
01.30 TYPE "POWERS OF"
01.35 ASK A
01.40 FOR I=1,B;TYPE A↑I;TYPE !
01.50 TYPE $
01.60 TYPE "DONE"!
01.70 QUIT
*GO
POWERS OF;2
=     2.0000
=     4.0000
=     8.0000
=    16.0000
=    32.0000
=    64.0000
=   128.0000
=   256.0000
=   512.0000
=  1024.0000
B0(00)=    10.0000
A0(00)=     2.0000
I0(00)=    11.0000
DONE
*1.15 ?
*1.5 ?
*WRITE
C-FOCAL,1969

01.10 COMMENT  DEMO PROGRAM IN FOCAL
01.15 ?
01.20 SET B=10
01.30 TYPE "POWERS OF"
01.35 ASK A
01.40 FOR I=1,B;TYPE A↑I;TYPE !
01.50 ?
01.60 TYPE "DONE"!
01.70 QUIT
*GO

SET B=10
TYPE "POWERS OFASK A
;2
FOR I=1,B;TYPE A↑I;=     2.0000TYPE !

TYPE A↑I;=     4.0000TYPE !

TYPE A↑I;=     8.0000TYPE !

TYPE A↑I;=    16.0000TYPE !
?01.00 @ 01.40
*ERASE 1.0
*WRITE
C-FOCAL,1969
*
```

Example: Using
ODT Symbolic Editor

```
3000/4551 7201
3001 /1125 1235
3002 /3152 7040
3003 /5177 7402

3001/1235 !
3035 /1471

3002B
3000G
3002 (1472
3003B
C
3003 (6305
B
A6305




K

A
*200
START, CLA CLL
       TAD A
       CIA
       DCA TALLY
MULT,  TAD B
       ISZ TALLLY
       JMP MULT
       HLT
A,     22
B,     44
TALLY, 0
$

6,8L
MULT,  TAD B
       ISZ TALLLY
       JMP MULT

7C
       ISZ TALLY

L
*200
START, CLA CLL
       TAD A
       CIA
       DCA TALLY
MULT,  TAD B
       ISZ TALLY
       JMP MULT
       HLT
A,     22
B,     44
TALLY, 0
$
```