SK8088 CPU BOARD

OWNER'S MANUAL

SKI ELECTRONICS

SKI ELECTRONICS, 3134 WOODS WAY, SAN JOSE, CA. 95148

T A B L E   O F   C O N T E N T S

PC8088 and SK8088 may be interchanged throughout the text.

GENERAL THEORY OF OPERATION

TEMPORARY MASTER MODE:

In the temporary master mode the PC8088 functions as a DMA device, although full DMA arbitration is not implemented. When power is applied all the flip flops and essential circuitry are initialized, which places the PC8088 in a hold state. Also, the extended address, A20 through A23, is reset to zero. While the permanent master CPU is in control of the bus following power-up or reset and undergoing normal activities, the PC8088 CPU board is completely invisible to the bus, except for the "onboard" I/O ports the master CPU needs access to. After an 8086/88 compatible program has been entered or loaded to RAM through the master CPU, i.e. from a terminal or disk system; then, control is transferred to the PC8088 board and it then proceeds to execute the program. This is accomplished by the master CPU outputting data to transfer port XD on the PC8088. Flip flop U16 changes state on the falling edge of the write pulse (PWR*) and Q (pin 8) brings the S-100 Hold* line (pin 74) active low causing the master CPU to enter a hold condition.

The HLDA line (pin 26) from the master CPU is connected to the input clock of U16 (pin 3). On receiving the active high signal (PHLDA), U16 changes state and Q (pin 6) brings the 8088 processor's Hold low, which also sinks the control, status, address and data out disable lines (CDSBL*, SDSBL*, ADSBL*, DODSBL*) active low. This disables the buffers of the master CPU and allows the PC8088 to control the bus. After the PC8088 has control of the bus, the LED (DS1) will light to show the transfer has been accomplished. Note, the LED only operates in the temporary master mode.

During power-up, U12 is reset so that control is transferred from the master CPU to the PC8088 board. The PC8088 is reset and begins execution at memory location FFFF0 (FFF0 in a 64K byte system). Here is where a jump instruction could be located in PROM or RAM to direct the PC8088 to the beginning of the program to be executed. Additional discussion on possible software schemes is given in the software section. If it is not desirable to have the PC8088 always reset and begin instruction at FFFF0 following transfer, the automatic reset function is disabled by outputting any data to the reset port XC. Port XC is connected to U12 (pin 11), a "D" flip flop, that changes state on the rising edge of PWR* and will either enable or disable reset to the processor. If the reset function is disabled, rewriting to port XC will reactivate the reset function.

When the PC8088 is ready to transfer control back to the bus master after completing the program, or if other boards need control (DMA disk controllers, I/O, etc.) it writes to the onboard transfer port XD. This input resets U16 and Q (pin 8) becomes high, reenabling the master CPU and simultaneously clearing U16 (pin 1); Q* also becomes high to place PC8088 in the hold condition (invisible mode) and release the bus to the master

CPU. After the bus master regains control, it continues executing from the point where the PC8088 took control. Meanwhile, the PC8088 CPU remains in a condition to accept another transfer. Switches S1a through S1f and S2e and S2h, and jumpers J3, J4, J5 and J6 are used to select the temporary master mode - refer to section IV where switch and jumper positions are shown for each mode and option.

MASTER MODE:

In the master mode the transfer circuitry is not used; howver, all other options are available, including a PROM socket for a 2716/32 cold boot PROM, eight vectored interrupts, I/O and PROM wait state generator, extended addressing, 5 Mhz operation, (upgradable to 8 Mhz) and other features. In the master mode the PC8088 is a complete and powerful S-100 16 bit CPU fully operational without any dependence on an 8-bit master.

EXTENDED ADDRESSING:

The PC8088 provides the capability for addressing a full 16 megabytes of memory through the extended addressing option. The Intel 8088 CPU chip addresses one megabyte directly using address lines A0 through A19. Additional address lines from U10 (74LS175) and port XF are provided for A20 through A23. The four upper data bits (D4 through D7) written to port XF provide the address bits A20 through A23. The four lower bits (D0 through D3) are not used.

After an address has been written to the port, it will remain unchanged until written to again, or the system is reset. On power-up the outputs of U10 are cleared to all zeroes and allows the board to initially cone up in the first (base) one megabyte memory segment. Therefore, 0FFFF0 is the reset starting address of the PC8088. In essence, the four upper address lines (A20 through A23) enable the PC8088 board to page select sixteen one megabyte sections. This option works in both the permanent and temporary master modes.

ON BOARD EPROM SOCKET:

This option provides a socket for either a 2716 (2K x 8) or 2732 (4K x8) EPROM. If a 2716 EPROM is used, jumper J2 a to b and J1 c to d. If a 2732 EPROM is used, jumper J1 b to c and J1 e to f. The EPROM socket is hardwired addressed to decode at FF800 for a 2716 and FF000 for a 2732. The extended address bits A20 through A23 are not decoded. Therefore, the EPROM will enable in all 16, one megabyte pages of memory. The EPROM should, through software, be disabled if other one megabyte sections are to be used. In an 8 bit S-100 system where all the boards see only address bits A0 throughA15 , the four uppermost address bits A16-23 won't be used. ~~The system will see the 2716 or 2732 EPROM ad- not be used~~ The system will see the 2716 and 2732 EPROM's addressed at F800 and F000, respectively. The EPROM is enabled by closing dip switch S1h and performing a read from the EPROM location.

If a write is performed while addressing the EPROM, the EPROM will be disabled so that data you are writing may be written to RAM at that address. This feature enables an overlap of RAM and PROM. On power-up, the PROM is always enabled by the clearing of flip flop U3. In addition, when enabled by S1h and properly addressed, the data in buffers U34 are tri-stated so that no conflict will exist on the bus. The PHANTOM is lowered to turn off any RAM at the PROM location. The PHANTOM is not always necessary since the data in buffer U34 is disabled, but it is provided as a safeguard. The PHANTOM can be disabled by cutting the trace between J8a and J8b. The PROM is visible only to PC8088 CPU and never to the master CPU.

In the following discussion, the PROM is assumed to be enabled by closing switch S1h. If not, then PROM is not used. When PC8088 CPU is given control in the temporary master mode, the PROM can be used as a permanent jump to a specific address to ease processor swapping. More discussion on processor swapping through the software is given in the Software Section. The PROM may be software disabled by writing data to port XE from either the PC8088 or the master CPU, and reenabled by writing again to port XE.

While in the permanent master mode, the PROM is enabled on power-up and the PC8088 processor begins execution at memory location 0FFFF0. Locating a jump instruction there can direct the CPU to begin at PROM location FF800 or FF000 for a "cold boot" of the operating system. The first command executed after the program transfers from the addressing space can be an output to port XE to turn off the PROM, or the PROM may remain on at the user's discretion.

I/O - MEMORY WAIT STATE GENERATION:

Closing switch S1g selects the option to insert one wait state every time an I/O and PROM read or write is performed. This allows slow I/O boards and devices to be used. When using the 8MHz version of PC8088 with this option enabled, a 450 ns onboard PROM and any standard device that can operate up to 5MHz may be used. Opening switch S1g disables this option. This option can be used in both the permanent and temporary master modes. When converting the PC8088 to an 8MHz unit, the 15 MHz crystal at Y12 must be replaced with a 24 MHz crystal. When the 450 ns onboard PROM is used, switch S1g must be closed.

I/O SELECT CIRCUITRY:

Eight I/O port addresses are used on the PC8088. Switches S2a through S2d are used to select the four upper address bits (A4 through A7). The four lower address bits (A0 through A3) are hardwired and not selectable. Ports X0 through X7 are not used here, but they can be used elsewhere in the system. The "X" represents your choice for A4 through A7.

U31 is a bidirectional octal transceiver (74LS245) and in the temporary master mode address bits A0 through A7 are inputs to the I/O ports. This is necessary for the permanent system master to be accessible to the transfer, reset and PROM enable ports.

Ports X8 and XA are read and write ports that control the registers in the 8259 Programmable Controller. These ports are only accessible to the PC8088 CPU. Ports XC, XD, XE and XF are write only ports. The data written to ports XC, XD, and XE are not used. Only the output to these ports, i.e., either a rising or falling edge of the write pulse changes the state of the flip flops connected to these ports.

Port XC controls the reset port. The I/O Select Circuitry option determines whether the PC8088 will reset every time it is transferred control or if it continues instructing where it stopped. This port is configured to cycle to the reset mode on power-up.

Port XD is the transfer port. Writing to this port controls transfers between the permanent master and the PC8088 CPU. This port is used only in the temporary master mode, and is configured so the PC8088 enters a hold state on power-up while in this mode.

Port XE controls the PROM enable. On power-up U3 is configured so that when switch S1h is closed, PROM is enabled. When writing to port XE the PROM is disabled through U3 and RAM can be read at memory location 0FF800 (2716) or 0FF000 (2732).

Port XF is the extended address port. The data in this port was discussed previously in the Extended Address Section. On power-up the output of the address bits A20 through A23 are reset to zero.

Ports XC, XD and XE control flip flops, i.e., each time one of these ports is written to the flip flop changes state. Example, on power up the PROM is enabled, then, by writing to port XE the PROM is disabled. When port XE is rewritten to the PROM is reenabled, etc., etc.

ADDITIONAL FEATURES:

In the permanent master mode, the PC8088 must provide a 2MHz clock signal to the S-100 bus, pin 49. This is achieved by a 4MHz crystal oscillator circuit consisting of U1, Y1, R1, R2, C3, and C4. The 4Mhz signal produced by this circuit is fed into the clock of U12, pin 3, where it is reduced to a 2MHz signal at pin 5. The clock has a 50/50 percent duty cycle and is enabled to the bus by closing dip switch S2g. When the PC8088 CPU board is used as a temporary master, switch S2g must be open so the permanent bus master will generate this signal.

An Mwrite signal is provided for users who's equipment does not have a front panel and have boards that use it. Closing dip switch S2f enables the Mwrite. Note, only one Mwrite signal is to be generated on the S-100 bus.

Switch S2h controls the selection of the option for the POC* signal to the bus. While in the temporary master mode this switch will normally be 1 eft open. Since some permanent master CPU boards generate a POC* signal directly from the reset signal, an infinite reset loop can be entered on power up. If dip switch If you are not familiar with the process that your CPU uses to generate these signals when first using the PC8088 as a temporary master in your system, leave dip switch S2h open. Normally, while in the master mode dip switch S2h will be closed. This may also vary depending on the other boards in your system.

The TEST input for the PC8088 CPU is connected through jumper J7 to ground. When the test feature of the PC8088 CPU board is used, cut the trace between J7a and J7c and connect J7a to J7b. This connects the TEST input to the RDY line from the output of U15 pin 3 or U37 pin 5 depending on the mode chosen. This feature can be used as a software debugging tool. The exact operation of the test input feature and how to utilize it is left up to the user. The TEST input is located on U24 (8088) at pin 23.

PROGRAMMABLE VECTORED INTERRUPTS:

The vectored interrupts on the PC8088 CPU board is accomplished by the use of Intel's 8259A programmable interrupt controller chip. A look at the schematic at U13 will give an idea on how thischip is interfaced to the bus and the 8088 CPU. Since an interrupt signal off the S-100 bus is active low and the 8259 (U13) needs the interrupt inputs active high, the signals must first go through U25 and U26, hex inverters, before entering the interrupt controller (U13). At first the operation of the 8259 may seem a bit complicated, but it really isn't. The 8259 applications manual and all that pertains to the 8259-8086/88 interface is reprinted in the following pages. This applications note is reprinted by permission of Intel Corporation, copyright 1979. All mnemonics which appear on these pages are copyright of the Intel Corporation, copyright 1982.

NOTE: Remember, only one Mwrite signal should be generated in a system. Systems with front panels generally generate the Mwrite through the front panel and therefore the Mwrite signal should be disabled on any boards in the system including the PC8088. Systems without front panels usually generate the Mwrite from the master CPU. Most 8-bit boards allow DMA boards to access the Mwrite generation circuitry from outside the CPU, but some boards do not. This causes the PC8088 CPU board in the temporary Computer master mode to have problems writing to some memories. California Computer Systems 2810 Z-80 CPU board is one such popular board. The best solution to this problem, should it arise, is to put a piece of tape over the gold finger of pin 68 (Mwrite) on the 8-bit CPU board (tape is less permanent than cutting a trace) or turn off the switch or jumper to the Mwrite signal (optional on some boards) and enable (turn on) the Mwrite option on the PC8088 board. The PC8088 allows any other boards in the system to access the Mwrite circuitry and thus, no matter what board is in control of the bus, a Mwrite signal will be generated.

1. CONCEPTS

In microcomputer systems there is usually a need for the processor to communicate with various Input/Output (I/O) devices such as keyboards, displays, sensors, and other peripherals. From the system viewpoint, the processor should spend as little time as possible servicing the peripherals since the time required for these I/O chores directly affects the amount of time available for other tasks. In other words, the system should be designed so that I/O servicing has little or no effect on the total system throughput. There are two basic methods of handling the I/O chores in a system: status polling and interrupt servicing.

The status poll method of I/O servicing essentially involves having the processor "ask" each peripheral if it needs servicing by testing the peripheral's status line. If the peripheral requires service, the processor branches to the appropriate service routine; if not, the processor continues with the main program. Clearly, there are several problems in implementing such an approach. First, how often a peripheral is polled is an important constraint. Some idea of the "frequency-of-service" required by each peripheral must be known and any software written for the system must accommodate this time dependence by "scheduling" when a device is polled. Second, there will obviously be times when a device is polled that is not ready for service, wasting the processor time that it took to do the poll. And other times, a ready device would have to wait until the processor "makes its rounds" before it could be serviced, slowing down the peripheral.

Other problems arise when certain peripherals are more important than others. The only way to implement the "priority" of devices is to poll the high priority devices more frequently than lower priority ones. It may even be necessary to poll the high priority devices while in a low priority device service routine. It is easy to see that the polled approach can be inefficient both time-wise and software-wise. Overall, the polled method of I/O servicing can have a detrimental effect on system throughput, thus limiting the tasks that can be performed by the processor.

A more desirable approach in most systems would allow the processor to be executing its main program and only stop to service the I/O when told to do so by the I/O itself. This is called the interrupt service method. In effect, the device would asynchronously signal the processor when it required service. The processor would finish its current instruction and then vector to the service routine for the device requesting service. Once the service routine is complete, the processor would resume exactly where it left off. Using the interrupt service method, no processor time is spent testing devices, scheduling is not needed, and priority schemes are readily implemented. It is easy to see that, using the interrupt service approach, system throughput would increase, allowing more tasks to be handled by the processor.

However, to implement the interrupt service method between processor and peripherals, additional hardware is usually required. This is because, after interrupting the processor, the device must supply information for vectoring program execution. Depending on the processor used, this can be accomplished by the device taking control of the data bus and "jamming" an instruction(s) onto it. The instruction(s) then vectors the program to the proper service routine. This of course requires additional control logic for each interrupt requesting device. Yet the implementation so far is only in the most basic form. What if certain peripherals are to be of higher priority than others? What if certain interrupts must be disabled while others are to be enabled? The possible variations go on, but they all add up to one theme: to provide greater flexibility using the interrupt service method, hardware requirements increase.

So, we're caught in the middle. The status poll method is a less desirable way of servicing I/O in terms of throughput, but its hardware requirements are minimal. On the other hand, the interrupt service method is most desirable in terms of flexibility and throughput, but additional hardware is required.

The perfect situation would be to have the flexibility and throughput of the interrupt method in an implementation with minimal hardware requirements. The 8259A Programmable Interrupt Controller (PIC) makes this all possible.

1.3 MCS-86/88™—8259A OVERVIEW

Operation of an MCS-86/88—8259A configuration has basic similarities of the MCS-80/85—8259A configurations. That is, a device can cause an interrupt by pulling one of the 8259A's interrupt request pins (IR0-IR7) high. If the 8259A honors the request, its INT pin will go high, driving the 8086/8088's INTR pin high. Like the 8080A and 8085A, the INTR pin of the 8086/8088 is asynchronous, thus it can receive an interrupt any time. The 8086/8088 can also accept or disregard requests on INTR under software control using the STI (Set Interrupt) or CLI (Clear Interrupt) instructions. These instructions set or clear the interrupt-enabled flag IF. Upon 8086/8088 reset the IF flag is cleared, disabling external interrupts on INTR. Beside the INTR pin, the 8086/8088 provides an NMI (Non-Maskable Interrupt) pin. The NMI functions similar to the 8085A's TRAP; it can't be disabled or masked. NMI has higher priority than INTR.

Although there are some basic similarities, the actual processing of interrupts with an 8086/8088 is different than an 8080A or 8085A. When an interrupt request is present and interrupts are enabled, the 8086/8088 enters its interrupt acknowledge machine cycle. The interrupt acknowledge machine cycle pushes the flag registers onto the stack (as in a PUSHF instruction). It then clears the IF flag which disables interrupts. The contents of both the code segment and the instruction pointer are then also pushed onto the stack. Thus, the stack retains the pre-interrupt flag status and pre-interrupt program location which are used to return from the service routine. The 8086/8088 then issues the first of two $\overline{\text{INTA}}$ pulses which signal the 8259A that the 8086/8088 has honored its interrupt request.

The 8259A is now ready to vector program execution to the corresponding service routine. This is done during the sequence of the two $\overline{\text{INTA}}$ pulses issued by the 8086/8088. Unlike operation with the 8080A or 8085A, the 8259A doesn't place a CALL instruction and the starting address of the service routine on the data bus. Instead, the first $\overline{\text{INTA}}$ pulse is used only to signal the 8259A of the honored request. The second $\overline{\text{INTA}}$ pulse causes the 8259A to place a single interrupt vector byte onto the data bus. Not used as a direct address, this interrupt-vector byte pertains to one of 256 interrupt "types" supported by the 8086/8088 memory. Program execution is vectored to the corresponding service routine by the contents of a specified interrupt type.

All 256 interrupt types are located in absolute memory locations 0 through 3FFH which make up the 8086/8088's interrupt-vector table. Each type in the interrupt-vector table requires 4 bytes of memory and stores a code segment address and an instruction pointer address. Figure 5 shows a block diagram of the interrupt-vector table. Locations 0 through 3FFH should be reserved for the interrupt-vector table alone. Furthermore, memory locations 00 through 7FH (types 0-31) are reserved for use by Intel Corporation for Intel hardware and software products. To maintain compatibility with present and future Intel products, these locations should not be used.

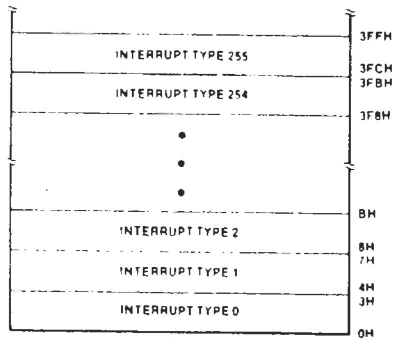


Figure 5. 8086/8088 Interrupt Vector Table

When the 8086/8088 receives an interrupt-vector byte from the 8259A, it multiplies its value by four to acquire the address of the interrupt type. For example, if the interrupt-vector byte specifies type 128 (80H), the vectored address in 8086/8088 memory is 4 × 80H, which equals 200H. Program execution is then vectored to the service routine whose address is specified by the code segment and instruction pointer values within type 128 located at 200H. To show how this is done, let's assume interrupt type 128 is to vector data to 8086/8088 memory location 2FF5FH. Figure 6 shows two possible ways to set values of the code segment and instruction pointer for vectoring to location 2FF5FH. Address generation by the code segment and instruction pointer is ac-

The IR1 routine has no such "protected" section since an "Enable Interrupts" instruction is the first one in its service routine. Note that in this example the IR1 request must stay high until it is acknowledged. This is covered in more depth in the "Interrupt Triggering" section.

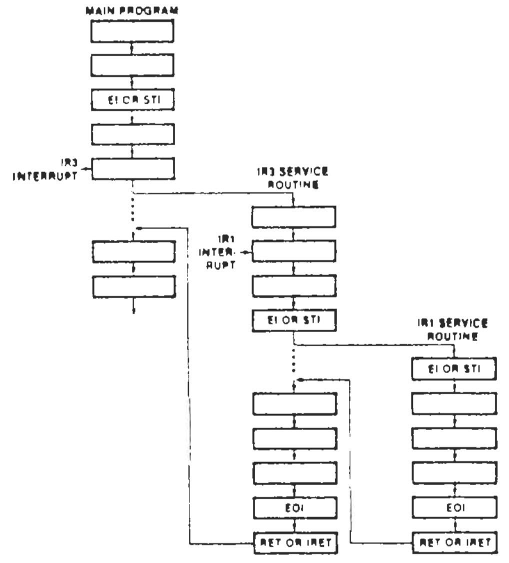


Figure 12. Fully Nested Mode Example (MCS 80/85™ or MCS 86/88™)

What is happening to the ISR register? While in the main program, no ISR bits are set since there aren't any interrupts in service. When the IR3 interrupt is acknowledged, the ISR3 bit is set. When the IR1 interrupt is acknowledged, both the ISR1 and the ISR3 bits are set, indicating that neither routine is complete. At this time, only IR0 could generate an interrupt since it is the only input with a higher priority than those previously in service. To terminate the IR1 routine, the routine must inform the 8259A that it is complete by resetting its ISR bit. **It does this by executing an EOI command. A "return" instruction then transfers execution back to the IR3 routine. This allows IR0-IR2 to interrupt the IR3** routine again, since ISR3 is the highest ISR bit set. No further interrupts occur in the example so the EOI command resets ISR3 and the "return" instruction causes the main program to resume at its pre-interrupt location, ending the example.

A single 8259A is essentially always in the fully nested mode unless certain programming conditions disturb it. The following programming conditions can cause the 8259A to go out of the high to low priority structure of the fully nested mode.

- The automatic EOI mode
- The special mask mode
- A slave with a master not in the special fully nested mode

These modes will be covered in more detail later, however, they are mentioned now so the user can be aware of them. As long as these program conditions aren't inacted, the fully nested mode remains undisturbed.

End of Interrupt

Upon completion of an interrupt service routine the 8259A needs to be notified so its ISR can be updated. This is done to keep track of which interrupt levels are in the process of being serviced and their relative priorities. Three different End-Of-Interrupt (EOI) formats are available for the user. These are: the non-specific EOI command, the specific EOI command, and the automatic EOI Mode. Selection of which EOI to use is dependent upon the interrupt operations the user wishes to perform.

Non-Specific EOI Command

A non-specific EOI command sent from the microprocessor lets the 8259A know when a service routine has been completed, without specification of its exact interrupt level. The 8259A automatically determines the interrupt level and resets the correct bit in the ISR.

To take advantage of the non-specific EOI the 8259A must be in a mode of operation in which it can predetermine in-service routine levels. For this reason the non-specific EOI command should only be used when the most recent level acknowledged and serviced is always the highest priority level. When the 8259A receives a non-specific EOI command, it simply resets the highest priority ISR bit, thus confirming to the 8259A that the highest priority routine of the routines in service is finished.

The main advantage of using the non-specific EOI command is that IR level specification isn't necessary as in the "Specific EOI Command", covered shortly. However, special consideration should be taken when deciding to use the non-specific EOI. Here are two program conditions in which it is best not used:

- Using the set priority command within an interrupt service routine.
- Using a special mask mode.

These conditions are covered in more detail in their own sections, but are listed here for the users reference.

Specific EOI Command

A specific EOI command sent from the microprocessor lets the 8259A know when a service routine of a particular interrupt level is completed. Unlike a non-specific EOI command, which automatically resets the highest priority ISR bit, a specific EOI command specifies an exact ISR bit to be reset. One of the eight IR levels of the 8259A can be specified in the command.

The reason the specific EOI command is needed, is to reset the ISR bit of a completed service routine whenever the 8259A isn't able to automatically determine it. An example of this type of situation might be if the priorities of the interrupt levels were changed during an interrupt routine ("Specific Rotation"). In this case, if any other routines were in service at the same time, a non-specific EOI might reset the wrong ISR bit. Thus the specific EOI command is the best bet in this case, or for that matter, any time in which confusion of interrupt priorities may exist. The specific EOI command can be used in all conditions of 8259A operation, including those that prohibit non-specific EOI command usage.

Automatic EOI Mode

When programmed in the automatic EOI mode, the microprocessor no longer needs to issue a command to notify the 8259A it has completed an interrupt routine. The 8259A accomplishes this by performing a non-specific EOI automatically at the trailing edge of the last $\overline{\text{INTA}}$ pulse (third pulse in MCS-80/85, second in MCS-86).

The obvious advantage of the automatic EOI mode over the other EOI command is no command has to be issued. In general, this simplifies programming and lowers code requirements within interrupt routines.

However, special consideration should be taken when deciding to use the automatic EOI mode because it disturbs the fully nested mode. In the automatic EOI mode the ISR bit of a routine in service is reset right after it's acknowledged, thus leaving no designation in the ISR that a sevice routine is being executed. If any interrupt request occurs during this time (and interrupts are enabled) it will get serviced regardless of its priority, low or high. The problem of "over nesting" may also happen in this situation. "Over nesting" is when an IR input keeps interrupting its own routine, resulting in unnecessary stack pushes which could fill the stack in a worst case condition. This is not usually a desired form of operation!

So what good is the automatic EOI mode with problems like those just covered? Well, again, like the other EOIs, selection is dependent upon the application. If interrupts are controlled at a predetermined rate, so as not to cause the problems mentioned above, the automatic EOI mode works perfect just the way it is. However, if interrupts happen sporadically at an indeterminate rate, the automatic EOI mode should only be used under the following guideline:

- When using the automatic EOI mode with an indeterminate interrupt rate, the microprocessor should keep its interrupt request input disabled during execution of service routines.

complished by an offset (they overlap). Of the total 20-bit address capability, the code segment can designate the upper 16 bits, the instruction pointer can designate the lower 16 bits.

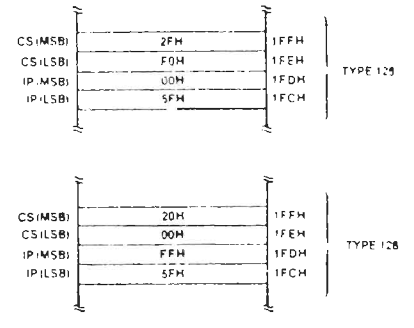


Figure 6. Two Examples of 8086/8088 Interrupt Type 128 Vectoring to Location 2FF5FH

When entering an interrupt service routine, those registers that are mutually used between the main program and service routine should be saved. The best way to do this is to "PUSH" each register used onto the stack immediately. The service routine can then "POP" each register off the stack in the same order when it is completed.

Once the service routine is completed the main program may be re-entered by using a IRET (Interrupt Return) instruction. The IRET instruction will pop the pre-interrupt instruction pointer, code segment and flags off the stack. Thus the main program will resume where it was interrupted with the same flag status regardless of changes in the service routine. Note especially that this includes the state of the IF flag, thus interrupts are re-enabled automatically when returning from the service routine.

Beside external interrupt generation from the INTR pin, the 8086/8088 is also able to invoke interrupts by software. Three interrupt instructions are provided: INT, INT (Type 3), and INTO. INT is a two byte instruction, the second byte selects the interrupt type. INT (Type 3) is a one byte instruction which selects interrupt Type 3. INTO is a conditional one byte interrupt instruction which selects Interrupt Type 4 if the OF flag (trap on overflow) is set. All the software interrupts vector program execution as the hardware interrupts do.

3.1 INTERRUPT VECTORING

Each IR input of the 8259A has an individual interrupt-vector address in memory associated with it. Designation of each address depends upon the initial programming of the 8259A. As stated earlier, the interrupt sequence and addressing of an MCS-80 and MCS-85 system differs from that of an MCS-86 and MCS-88 system. Thus, the 8259A must be initially programmed in either a MCS-80/85 or MCS-86/88 mode of operation to insure the correct interrupt vectoring.

MCS-86/88™ Mode

When programmed in the MCS-86/88 mode, the 8259A should only be used within an MCS-86 or MCS-88 system. In this mode, the 8086/8088 will handle interrupts in the format described earlier in the "8259A—8086/8088 Overview".

Upon interrupt in the MCS-86/88 mode, the 8259A will output a single interrupt-vector byte to the data bus. This is in response to only two $\overline{\text{INTA}}$ pulses issued by the 8086/8088 after the 8259A has raised INT high.

The first $\overline{\text{INTA}}$ pulse is used only for set-up purposes internal to the 8259A. As in the MCS-80/85 mode, this set-up includes priority resolution and cascade mode operations which will be covered later. Unlike the MCS-80/85 mode, no CALL opcode is placed on the data bus.

The second $\overline{\text{INTA}}$ pulse is used to enable the single interrupt-vector byte onto the data bus. The 8086/8088 uses this interrupt-vector byte to select one of 256 interrupt "types" in 8086/8088 memory. Interrupt type selection for all eight IR levels is made when initially programming the 8259A. However, reference to only one interrupt type is needed for programming. The upper 5 bits of the interrupt vector byte are user definable. The lower 3 bits are automatically inserted by the 8259A depending upon the IR level.

Contents of the interrupt-vector byte for 8086/8088 type selection is put on the data bus during the second $\overline{\text{INTA}}$ pulse and is shown in Figure 10.

IR	D7	D6	D5	D4	D3	D2	D1	D0
7	T7	T6	T5	T4	T3	1	1	1
6	T7	T6	T5	T4	T3	1	1	0
5	T7	T6	T5	T4	T3	1	0	1
4	T7	T6	T5	T4	T3	1	0	0
3	T7	T6	T5	T4	T3	0	1	1
2	T7	T6	T5	T4	T3	0	1	0
1	T7	T6	T5	T4	T3	0	0	1
0	T7	T6	T5	T4	T3	0	0	0

Figure 10. Interrupt Vector Byte, MCS 86/88™ Mode

3.2 INTERRUPT PRIORITIES

A variety of modes and commands are available for controlling interrupt priorities of the 8259A. All of them are programmable, that is, they may be changed dynamically under software control. With these modes and commands, many possibilities are conceivable, giving the user enough versatility for almost any interrupt controlled application.

Fully Nested Mode

The fully nested mode of operation is a general purpose priority mode. This mode supports a multilevel-interrupt structure in which priority order of all eight IR inputs are arranged from highest to lowest.

Unless otherwise programmed, the fully nested mode is entered by default upon initialization. At this time, IR0 is assigned the highest priority through IR7 the lowest. The fully nested mode, however, is not confined to this IR structure alone. Once past initialization, other IR inputs can be assigned highest priority also, keeping the multilevel-interrupt structure of the fully nested mode.

Figure 11A-C shows some variations of the priority structures in the fully nested mode.

IR LEVELS	IR7	IR6	IR5	IR4	IR3	IR2	IR1	IR0
PRIORITY	7	6	5	4	3	2	1	0

A

IR LEVELS	IR7	IR6	IR5	IR4	IR3	IR2	IR1	IR0
PRIORITY	4	3	2	1	0	7	6	5

B

IR LEVELS	IR7	IR6	IR5	IR4	IR3	IR2	IR1	IR0
PRIORITY	1	0	7	6	5	4	3	2

C

Figure 11. A-C. Some Variations of Priority Structure in the Fully Nested Mode

Further explanation of the fully nested mode, in this section, is linked with information of general 8259A interrupt operations. This is done to ease explanation to the user in both areas.

In general, when an interrupt is acknowledged, the highest priority request is determined from the IRR (Interrupt Request Register). The interrupt vector is then placed on the data bus. In addition, the corresponding bit in the ISR (In-Service Register) is set to designate the routine in service. This ISR bit remains set until an EOI (End-Of-Interrupt) command is issued to the 8259A. EOI's will be explained in greater detail shortly.

In the fully nested mode, while an ISR bit is set, all further requests of the same or lower priority are inhibited from generating an interrupt to the microprocessor. A higher priority request, though, can generate an interrupt, thus vectoring program execution to its service routine. Interrupts are only acknowledged, however, if the microprocessor has previously executed an "Enable Interrupts" instruction. This is because the interrupt request pin on the microprocessor gets disabled automatically after acknowledgement of any interrupt. The assembly language instructions used to enable interrupts are "EI" for 8080A/8085A and "STI" for 8086/8088. Interrupts can be disabled by using the instruction "DI" for 8080A/ 8085A and "CLI" for 8086/8088. When a routine is completed a "return" instruction is executed, "RET" for 8080A/8085A and "IRET" for 8086/8088.

Figure 12 illustrates the correct usage of interrupt related instructions and the interaction of interrupt levels in the fully nested mode.

Assuming the IR priority assignment for the example in Figure 12 is IR0 the highest through IR7 the lowest, the sequence is as follows. During the main program, IR3 makes a request. Since interrupts are enabled, the microprocessor is vectored to the IR3 service routine. During the IR3 routine, IR1 asserts a request. Since IR1 has higher priority than IR3, an interrupt is generated. However, it is not acknowledged because the microprocessor disabled interrupts in response to the IR3 interrupt. The IR1 interrupt is not acknowledged until the "Enable Interrupts" instruction is executed. Thus the IR3 routine has a "protected" section of code over which no interrupts (except non-maskable) are allowed.

By doing this, higher priority interrupt levels will be serviced only after the completion of a routine in service. This guideline restores the fully nested structure in regards to the IRR; however, a routine in-service can't be interrupted.

Automatic Rotation — Equal Priority

Automatic rotation of priorities serves in applications where the interrupting devices are of equal priority, such as communications channels. The concept is that once a peripheral is serviced, all other equal priority peripherals should be given a chance to be serviced before the original peripheral is serviced again. This is accomplished by automatically assigning a peripheral the lowest priority after being serviced. Thus, in worst case, the device would have to wait until all other devices are serviced before being serviced again.

There are two methods of accomplishing automatic rotation. One is used in conjunction with the non-specific EOI, "rotate on non-specific EOI command". The other is used with the automatic EOI mode, "rotate in automatic EOI mode".

Rotate on Non-Specific EOI Command

When the rotate on non-specific EOI command is issued, the highest ISR bit is reset as in a normal non-specific EOI command. After it's reset though, the corresponding IR level is assigned lowest priority. Other IR priorities rotate to conform to the fully nested mode based on the newly assigned low priority.

Figures 13A and B show how the rotate on non-specific EOI command effects the interrupt priorities. Let's assume the IR priorities were assigned with IR0 the highest and IR7 the lowest, as in 13A. IR6 and IR4 are already in service but neither is completed. Being the higher priority routine, IR4 is necessarily the routine being executed. During the IR4 routine a rotate on non-specific EOI command is executed. When this happens, bit 4 in the ISR is reset. IR4 then becomes the lowest priority and IR5 becomes the highest as in 13B.

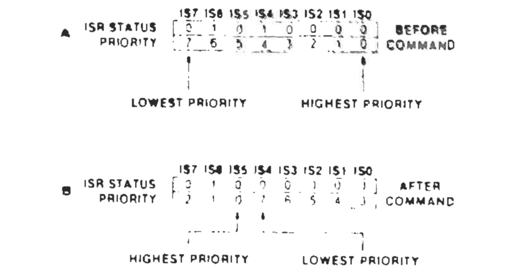


Figure 13. A-B Rotate on Non-specific EOI Command Example

Rotate in Automatic EOI Mode

The rotate in automatic EOI mode works much like the rotate on non-specific EOI command. The main difference is that priority rotation is done automatically after the last $\overline{\text{INTA}}$ pulse of an interrupt request. To enter or exit this mode a rotate-in-automatic-EOI set command and rotate-in-automatic-EOI clear command is provided. After that, no commands are needed as with the normal automatic EOI mode. However, it must be remembered, when using any form of the automatic EOI mode, special consideration should be taken. Thus, the guideline for the automatic EOI mode also stands for the rotate in automatic EOI mode.

Specific Rotation — Specific Priority

Specific rotation gives the user versatile capabilities in interrupt controlled operations. It serves in those applications in which a specific device's interrupt priority must be altered. As opposed to automatic rotation which automatically sets priorities, specific rotation is completely user controlled. That is, the user selects which interrupt level is to receive lowest or highest priority. This can be done during the main program or within interrupt routines. Two specific rotation commands are available to the user, the "set priority command" and the "rotate on specific EOI command."

Set Priority Command

The set priority command allows the programmer to assign an IR level the lowest priority. All other interrupt levels will conform to the fully nested mode based on the newly assigned low priority.

An example of how the set priority command works is shown in Figures 14A and 14B. These figures show the status of the ISR and the relative priorities of the interrupt levels before and after the set priority command. Two interrupt routines are shown to be in service in Figure 14A. Since IR2 is the highest priority, it is necessarily the routine being executed. During the IR2 routine, priorities are altered so that IR5 is the highest. This is done simply by issuing the set priority command to the 8259A. In this case, the command specifies IR4 as being the lowest priority. The result of this set priority command is shown in Figure 14B. Even though IR7 now has higher priority than IR2, it won't be acknowledged until the IR2 routine is finished (via EOI). This is because priorities are only resolved upon an interrupt request or an interrupt acknowledge sequence. If a higher priority request occurs during the IR2 routine, then priorities are resolved and the highest will be acknowledged.

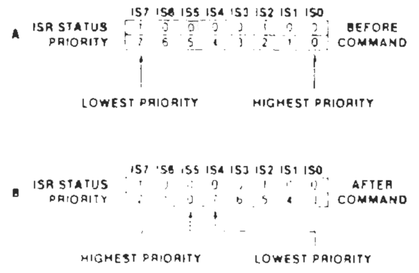


Figure 14. A-B. Set Priority Command Example

When completing a service routine in which the set priority command is used, the correct EOI must be issued. The non-specific EOI command shouldn't be used in the same routine as a set priority command. This is because the non-specific EOI command resets the highest ISR bit, which, when using the set priority command, is not always the most recent routine in service. The automatic EOI mode, on the other hand, can be used with the set priority command. This is because it automatically performs a non-specific EOI before the set priority command can be issued. The specific EOI command is the best bet in most cases when using the set priority command within a routine. By resetting the specific ISR bit of a routine being completed, confusion is eliminated.

Rotate on Specific EOI Command

The rotate on specific EOI command is literally a combination of the set priority command and the specific EOI command. Like the set priority command, a specified IR level is assigned lowest priority. Like the specific EOI command, a specified level will be reset in the ISR. Thus the rotate on specific EOI command accomplishes both tasks in only one command.

If it is not necessary to change IR priorities prior to the end of an interrupt routine, then this command is advantageous. For an EOI command must be executed anyway (unless in the automatic EOI mode), so why not do both at the same time?

Interrupt Masking

Disabling or enabling interrupts can be done by other means than just controlling the microprocessor's interrupt request pin. The 8259A has an IMR (Interrupt Mask Register) which enhances interrupt control capabilities. Rather than all interrupts being disabled or enabled at the same time, the IMR allows individual IR masking. The IMR is an 8-bit register, bits 0–7 directly correspond to IR0–IR7. Any IR input can be masked by writing to the IMR and setting the appropriate bit. Likewise, any IR input can be enabled by clearing the correct IMR bit.

There are various uses for masking off individual IR inputs. One example is when a portion of a main routine wishes only to be interrupted by specific interrupts. Another might be disabling higher priority interrupts for a portion of a lower priority service routine. The possibilities are many.

When an interrupt occurs while its IMR bit is set, it isn't necessarily forgotten. For, as stated earlier, the IMR acts only on the output of the IRR. Even with an IR input masked it is still possible to set the IRR. Thus, when resetting an IMR, if its IRR bit is set it will then generate an interrupt. This is providing, of course, that other priority factors are taken into consideration and the IR request remains active. If the IR request is removed before the IMR is reset, no interrupt will be acknowledged.

Special Mask Mode

In various cases, it may be desirable to enable interrupts of a lower priority than the routine in service. Or, in other words, allow lower priority devices to generate interrupts. However, in the fully nested mode, all IR levels of priority below the routine in service are inhibited. So what can be done to enable them?

Well, one method could be using an EOI command before the actual completion of a routine in service. But beware, doing this may cause an "over nesting" problem, similar to in the automatic EOI mode. In addition, resetting an ISR bit is irreversible by software control, so lower priority IR levels could only be later disabled by setting the IMR.

A much better solution is the special mask mode. Working in conjunction with the IMR, the special mask mode enables interrupts from all levels except the level in service. This is done by masking the level that is in service and then issuing the special mask mode command. Once the special mask mode is set, it remains in effect until reset.

Figure 15 shows how to enable lower priority interrupts by using the Special Mask Mode (SMM). Assume that IR0 has highest priority when the main program is interrupted by IR4. In the IR4 service routine an enable interrupt instruction is executed. This only allows higher priority interrupt requests to interrupt IR4 in the normal fully nested mode. Further in the IR4 routine, bit 4 of the IMR is masked and the special mask mode is entered. Priority operation is no longer in the fully nested mode. All interrupt levels are enabled except for IR4. To leave the special mask mode, the sequence is executed in reverse.

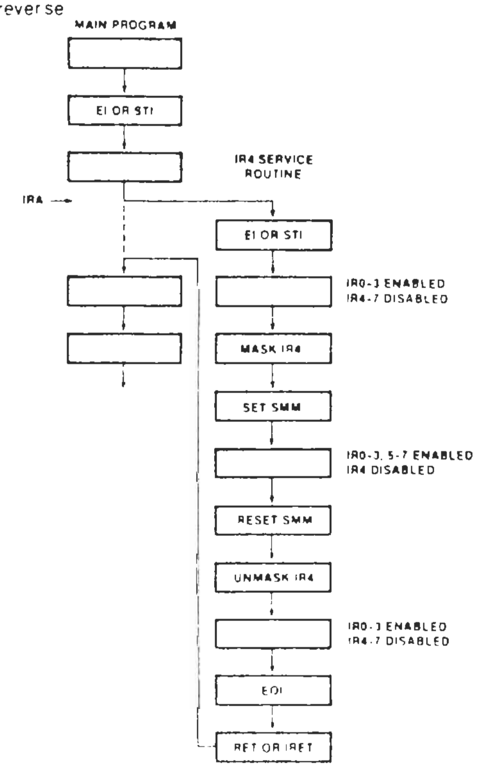


Figure 15. Special Mask Mode Example (MCS 80/85™ or MCS 86/88™)

Precautions must be taken when exiting an interrupt service routine which has used the special mask mode. A non-specific EOI command can't be used when in the special mask mode. This is because a non-specific won't clear an ISR bit of an interrupt which is masked when in the special mask mode. In fact, the bit will appear invisible. If the special mask mode is cleared before an EOI command is issued a non-specific EOI command can be used. This could be the case in the example shown in Figure 15, but, to avoid any confusion it's best to use the specific EOI whenever using the special mask mode.

It must be remembered that the special mask mode applies to all masked levels when set. Take, for instance, IR1 interrupting IR4 in the previous example. If this happened while in the special mask mode, and the IR1 routine masked itself, all interrupts would be enabled except IR1 and IR4 which are masked.

3.3 INTERRUPT TRIGGERING

There are two classical ways of sensing an active interrupt request: a level sensitive input or an edge sensitive input. The 8259A gives the user the capability for either method with the edge triggered mode and the level triggered mode. Selection of one of these interrupt triggering methods is done during the programmed initialization of the 8259A.

Level Triggered Mode

When in the level triggered mode the 8259A will recognize any active (high) level on an IR input as an interrupt request. If the IR input remains active after an EOI command has been issued (resetting its ISR bit), another interrupt will be generated. This is providing of course, the processor INT pin is enabled. Unless repetitious interrupt generation is desired, the IR input must be brought to an inactive state before an EOI command is issued in its service routine. However, it must not go inactive so soon that it disobeys the necessary timing requirements shown in Figure 16. Note that the request on the IR input must remain until after the falling edge of the first $\overline{\text{INTA}}$ pulse. If on any IR input, the request goes inactive before the first $\overline{\text{INTA}}$ pulse, the 8259A will respond as if IR7 was active. In any design in which there's a possibility of this happening, the IR7 default feature can be used as a safeguard. This can be accomplished by using the IR7 routine as a "clean-up routine" which might recheck the 8259A status or merely return program execution to its pre-interrupt location.

Depending upon the particular design and application, the level triggered mode has a number of uses. For one, it provides for repetitious interrupt generation. This is useful in cases when a service routine needs to be continually executed until the interrupt request goes inactive. Another possible advantage of the level triggered mode is it allows for "wire-OR'ed" interrupt requests. That is, a number of interrupt requests using the same IR input. This can't be done in the edge triggered mode, for if a device makes an interrupt request while the IR input is high (from another request), its transition will be "shadowed". Thus the 8259A won't recognize further interrupt requests because its IR input is already high. Note that when a "wire-OR'ed" scheme is used, the actual requesting device has to be determined by the software in the service routine.

Caution should be taken when using the automatic EOI mode and the level triggered mode together. Since in the automatic EOI mode an EOI is automatically performed at the end of the interrupt acknowledge sequence, if the processor enables interrupts while an IR input is still high, an interrupt will occur immediately. To avoid this situation interrupts should be kept disabled until the end of the service routine or until the IR input returns low.

Edge Triggered Mode

When in the edge triggered mode, the 8259A will only recognize interrupts if generated by an inactive (low) to active (high) transition on an IR input. The edge triggered mode incorporates an edge lockout method of operation. This means that after the rising edge of an interrupt request and the acknowledgement of the request, the positive level of the IR input won't generate further interrupts on this level. The user needn't worry about quickly removing the request after acknowledgement in fear of generating further interrupts as might be the case in the level triggered mode. Before another interrupt can be generated the IR input must return to the inactive state.

Referring back to Figure 16, the timing requirements for interrupt triggering is shown. Like the level triggered mode, in the edge triggered mode the request on the IR input must remain active until after the falling edge of the first $\overline{\text{INTA}}$ pulse for that particular interrupt. Unlike the level triggered mode, though, after the interrupt request is acknowledged its IRR latch is disarmed. Only after the IR input goes inactive will the IRR latch again become armed, making it ready to receive another interrupt request (in the level triggered mode, the IRR latch is always armed). Because of the way the edge triggered mode functions, it is best to use a positive level with a negative pulse to trigger the IR requests. With this type of input, the trailing edge of the pulse causes the interrupt and the maintained positive level meets the necessary timing requirements (remaining high until after the interrupt acknowledge occurs). Note that the IR7 default feature mentioned in the "level triggered mode" section also works for the edge triggered mode.

Depending upon the particular design and application, the edge triggered mode has various uses. Because of its edge lockout operation, it is best used in those applications where repetitious interrupt generation isn't desired. It is also very useful in systems where the interrupt request is a pulse (this should be in the form of a negative pulse to the 8259A). Another possible advantage is that it can be used with the automatic EOI mode without the cautions in the level triggered mode. Overall, in most cases, the edge triggered mode simplifies operation for the user, since the duration of the interrupt request at a positive level is not usually a factor.

3.4 INTERRUPT STATUS

By means of software control, the user can interrogate the status of the 8259A. This allows the reading of the internal interrupt registers, which may prove useful for interrupt control during service routines. It also provides for a modified status poll method of device monitoring, by using the poll command. This makes the status of the internal IR inputs available to the user via software control. The poll command offers an alternative to the interrupt vector method, especially for those cases when more than 64 interrupts are needed.

Reading Interrupt Registers

The contents of each 8-bit interrupt register, IRR, ISR, and IMR, can be read to update the user's program on the present status of the 8259A. This can be a versatile tool in the decision making process of a service routine, giving the user more control over interrupt operations. Before delving into the actual process of reading the registers, let's briefly review their general descriptions:

IRR (Interrupt Request Register)	Specifies all interrupt levels requesting service.
ISR (In-Service Register)	Specifies all interrupt levels which are being serviced.
IMR (Interrupt Mask Register)	Specifies all interrupt levels that are masked.

To read the contents of the IRR or ISR, the user must first issue the appropriate read register command (read IRR or read ISR) to the 8259A. Then by applying a $\overline{RD}$ pulse to the 8259A (an INput instruction), the contents of the desired register can be acquired. There is no need to issue a read register command every time the IRR or ISR is to be read. Once a read register command is received by the 8259A, it "remembers" which register has been selected. Thus, all that is necessary to read the contents of the same register more than once is the $\overline{RD}$ pulse and the correct addressing (A0 = 0, explained in "Programming the 8259A"). Upon initialization, the selection of registers defaults to the IRR. Some caution should be taken when using the read register command in a system that supports several levels of interrupts. If the higher priority routine causes an interrupt between the read register command and the actual input of the register contents, there's no guarantee that the same register will be selected when it returns. Thus it is best in such cases to disable interrupts during the operation.

Reading the contents of the IMR is different than reading the IRR or ISR. A read register command is not necessary when reading the IMR. This is because the IMR can be addressed directly for both reading and writing. Thus all that the 8259A requires for reading the IMR is a $\overline{RD}$ pulse and the correct addressing (A0 = 1, explained in "Programming the 8259A").

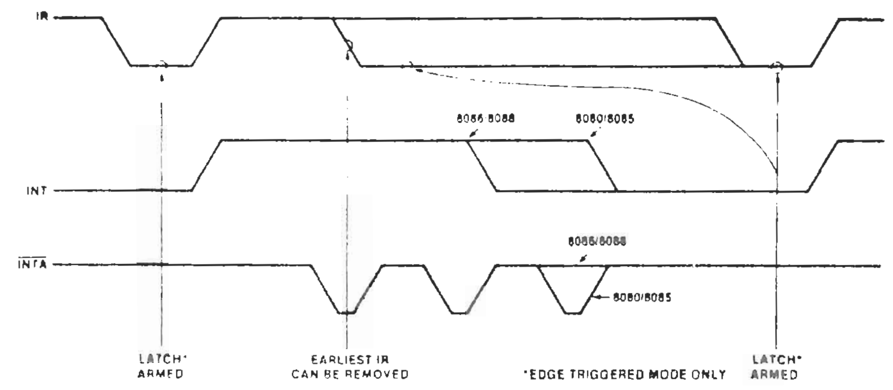


Figure 16. IR Triggering Timing Requirements

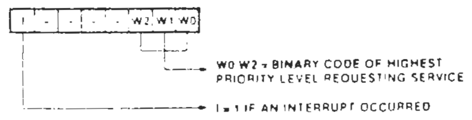
Poll Command

As mentioned towards the beginning of this application note, there are two methods of servicing peripherals: status polling and interrupt servicing. For most applications the interrupt service method is best. This is because it requires the least amount of CPU time, thus increasing system throughput. However, for certain applications, the status poll method may be desirable.

For this reason, the 8259A supports polling operations with the poll command. As opposed to the conventional method of polling, the poll command offers improved device servicing and increased throughput. Rather than having the processor poll each peripheral in order to find the actual device requiring service, the processor polls the 8259A. This allows the use of all the previously mentioned priority modes and commands. Additionally, both polled and interrupt methods can be used within the same program.

To use the poll command the processor must first have its interrupt request pin disabled. Once the poll command is issued, the 8259A will treat the next ($\overline{CS}$ qualified) $\overline{RD}$ pulse issued to it (an INput instruction) as an interrupt acknowledge. It will then set the appropriate bit in the ISR, if there was an interrupt request, and enable a special word onto the data bus. This word shows whether an interrupt request has occurred and the highest priority level requesting service. Figure 17 shows the contents of the "poll word" which is read by the processor. Bits W0-W2 convey the binary code of the highest priority level requesting service. Bit I designates whether or not an interrupt request is present. If an interrupt request is present, bit I will equal 1. If there isn't an interrupt request at all, bit I will equal 0 and bits W0-W2 will be set to ones. Service to the requesting device is achieved by software decoding the poll word and branching to the appropriate service routine. Each time the 8259A is to be polled, the poll command must be written before reading the poll word.

The poll command is useful in various situations. For instance, it's a good alternative when memory is very limited, because an interrupt-vector table isn't needed. Another use for the poll command is when more than 64 interrupt levels are needed (64 is the limit when cascading 8259's). The only limit of interrupts using the poll command is the number of 8259's that can be addressed in a particular system. Still another application of the poll command might be when the INT or $\overline{INTA}$ signals are not available. This might be the case in a large system where a processor on one card needs to use an 8259A on a different card. In this instance, the poll command is the only way to monitor the interrupt devices and still take advantage of the 8259A's prioritizing features. For those cases when the 8259A is using the poll command only and not the interrupt method, each 8259A must receive an initialization sequence (interrupt vector). This must be done even though the interrupt vector features of the 8259A are not used. In this case, the interrupt vector specified in the initialization sequence could be a "fake".



I
-
-
-
-
W2
W1
W0
W0-W2 = BINARY CODE OF HIGHEST PRIORITY LEVEL REQUESTING SERVICE
I = 1 IF AN INTERRUPT OCCURRED

Figure 17. Poll Word

4. PROGRAMMING THE 8259A

Programming the 8259A is accomplished by using two types of command words: Initialization Command Words (ICWs) and Operational Command Words (OCWs). All the modes and commands explained in the previous section, "Operation of the 8259A", are programmable using the ICWs and OCWs (see Appendix A for cross reference). The ICWs are issued from the processor in a sequential format and are used to set-up the 8259A in an initial state of operation. The OCWs are issued as needed to vary and control 8259A operation.

Both ICWs and OCWs are sent by the processor to the 8259A via the data bus (8259A $\overline{CS} = 0$, $\overline{WR} = 0$). The 8259A distinguishes between the different ICWs and OCWs by the state of its A0 pin (controlled by processor addressing), the sequence they're issued in (ICWs only), and some dedicated bits among the ICWs and OCWs. Those bits which are dedicated are indicated so by fixed values (0 or 1) in the corresponding ICW or OCW programming formats which are covered shortly. Note, when issuing either ICWs or OCWs, the interrupt request pin of the processor should be disabled.

4.1 INITIALIZATION COMMAND WORDS (ICWs)

Before normal operation can begin, each 8259A in a system must be initialized by a sequence of two to four programming bytes called ICWs (Initialization Command Words). **The ICWs are used to set-up the necessary conditions and modes for proper 8259A operation.**

Figure 20 shows the initialization flow of the 8259A. Both ICW1 and ICW2 must be issued for any form of 8259A operation. However, ICW3 and ICW4 are used only if designated so in ICW1. Determining the necessity and use of each ICW is covered shortly in individual groupings. Note that, once intialized, if any programming changes within the ICWs are to be made, the entire ICW sequence must be reprogrammed, not just an individual ICW.

Certain internal set-up conditions occur automatically within the 8259A after the first ICW has been issued. These are:

A. Sequencer logic is set to accept the remaining ICWs as designated in ICW1.

B. The ISR (In-Service Register) and IMR (Interrupt Mask Register) are both cleared.

C. The special mask mode is reset.

D. The rotate in automatic EOI mode flip-flop is cleared

E. The IRR (Interrupt Request Register) is selected for the read register command

F. If the IC4 bit equals 0 in ICW1, all functions in ICW4 are cleared. 8080/8085 mode is selected by default

G. The fully nested mode is entered with an initial priority assignment of IR0 highest through IR7 lowest

H. The edge sense latch of each IR priority cell is cleared, thus requiring a low to high transition to generate an interrupt (edge triggered mode effected only).

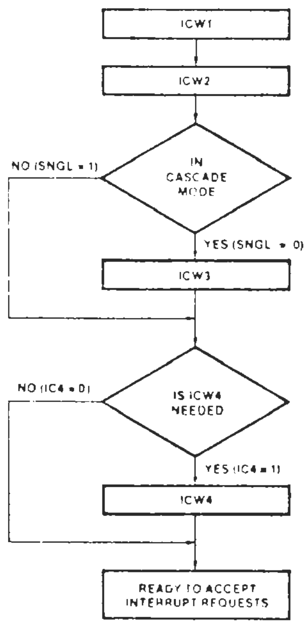


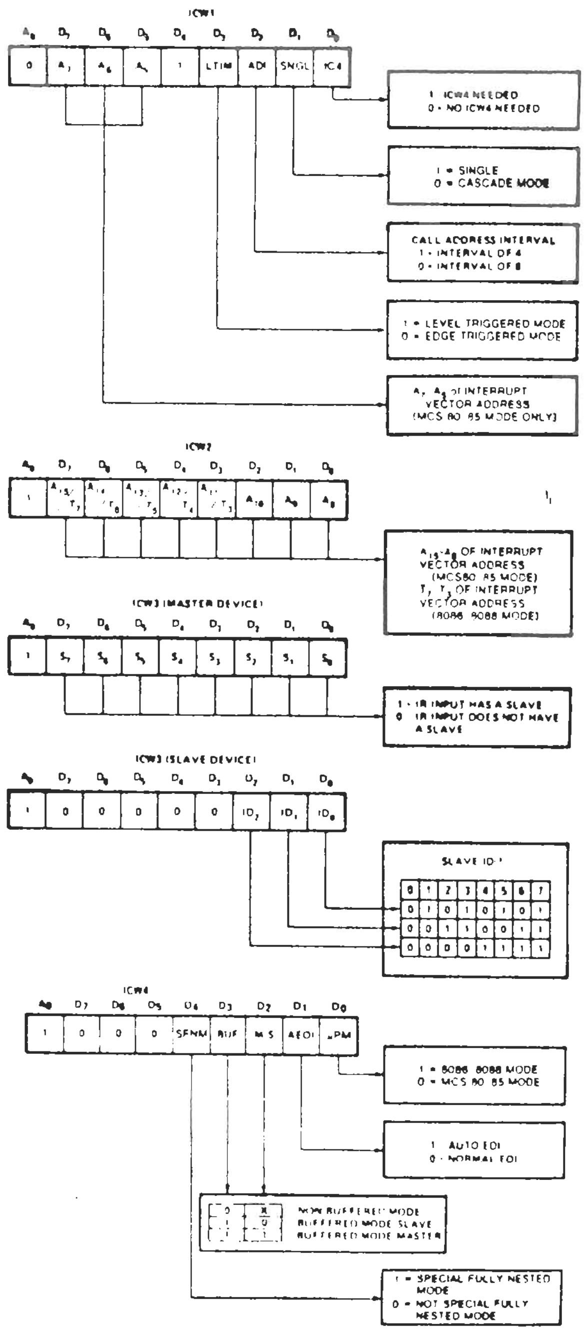
Figure 20. Initialization Flow

The ICW programming format, Figure 21, shows bit designation and a short definition of each ICW. With the ICW format as reference, the functions of each ICW will now be explained individually.

ICW1 and ICW2

Issuing ICW1 and ICW2 is the minimum amount of programming needed for any type of 8259A operation. The majority of bits within these two ICWs are used to designate the interrupt vector starting address. The remaining bits serve various purposes. Description of the ICW1 and ICW2 bits is as follows.

IC4: The IC4 bit is used to designate to the 8259A whether or not ICW4 will be issued. If any of the ICW4 operations are to be used, ICW4 must equal 1. If they aren't used, then ICW4 needn't be issued and IC4 can equal 0. Note that if IC4 = 0, the 8259A will assume operation in the MCS-80/85 mode



ICW1
A0 D7 D6 D5 D4 D3 D2 D1 D0
0 A7 A6 A5 1 LTIM ADI SNGL IC4
1 = ICW4 NEEDED
0 = NO ICW4 NEEDED
1 = SINGLE
0 = CASCADE MODE
CALL ADDRESS INTERVAL
1 = INTERVAL OF 4
0 = INTERVAL OF 8
1 = LEVEL TRIGGERED MODE
0 = EDGE TRIGGERED MODE
A7 - A5 of INTERRUPT VECTOR ADDRESS (MCS 80/85 MODE ONLY)
ICW2
1 A15/T7 A14/T6 A13/T5 A12/T4 A11/T3 A10 A9 A8
A15-A8 OF INTERRUPT VECTOR ADDRESS (MCS80/85 MODE)
T7-T3 OF INTERRUPT VECTOR ADDRESS (8086/8088 MODE)
ICW3 (MASTER DEVICE)
1 S7 S6 S5 S4 S3 S2 S1 S0
1 = IR INPUT HAS A SLAVE
0 = IR INPUT DOES NOT HAVE A SLAVE
ICW3 (SLAVE DEVICE)
1 0 0 0 0 0 ID2 ID1 ID0
SLAVE ID
0 1 2 3 4 5 6 7
0 1 0 1 0 1 0 1
0 0 1 1 0 0 1 1
0 0 0 0 1 1 1 1
ICW4
A0 D7 D6 D5 D4 D3 D2 D1 D0
1 0 0 0 SFNM BUF M/S AEOI μPM
1 = 8086/8088 MODE
0 = MCS-80/85 MODE
1 = AUTO EOI
0 = NORMAL EOI
0 X - NON BUFFERED MODE
1 0 - BUFFERED MODE/SLAVE
1 1 - BUFFERED MODE/MASTER
1 = SPECIAL FULLY NESTED MODE
0 = NOT SPECIAL FULLY NESTED MODE

Figure 21. Initialization Command Words (ICWS) Programming Format

SNGL: The SNGL bit is used to designate whether or not the 8259A is to be used alone or in the cascade mode. If the cascade mode is desired, SNGL must equal 0. In doing this, the 8259A will accept ICW3 for further cascade mode programming. If the 8259A is to be used as the single 8259A within a system, the SNGL bit must equal 1. ICW3 won't be accepted.

ADI: The ADI bit is used to specify the address interval for the MCS-80/85 mode. If a 4 byte address interval is to be used, ADI must equal 1. For an 8-byte address interval, ADI must equal 0. The state of ADI is ignored when the 8259A is in the MCS 86/88 mode.

LTIM: The LTIM bit is used to select between the two IR input triggering modes. If LTIM = 1, the level triggered mode is selected. If LTIM = 0, the edge triggered mode is selected.

A5-A15: The A5-A15 bits are used to select the interrupt vector address when in the MCS-80/85 mode. There are two programming formats that can be used to do this. Which one is implemented depends upon the selected address interval (ADI). If ADI is set for the 4-byte interval, then the 8259A will automatically insert A0-A4 (A0, A1 = 0 and A2, A3, A4 = IR0-7). Thus A5-A15 must be user selected by programming the A5-A15 bits with the desired address. If ADI is set for the 8 byte interval, then A0-A5 are automatically inserted (A0, A1, A2 = 0 and A3, A4, A5 = IR0-7). This leaves A6-A15 to be selected by programming the A6-A15 bits with the desired address. The state of bit 5 is ignored in the latter format.

T3-T7: The T3-T7 bits are used to select the interrupt type when the MCS-86/88 mode is used. The programming of T3-T7 selects the upper 5 bits. The lower 3 bits are automatically inserted, corresponding to the IR level causing the interrupt. The state of bits A5-A10 will be ignored when in the MCS-86/88 mode. Establishing the actual memory address of the interrupt is shown in Figure 22.

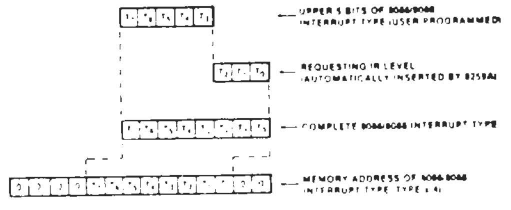

UPPER 5 BITS OF 8086/8088 INTERRUPT TYPE (USER PROGRAMMED)
REQUESTING IR LEVEL (AUTOMATICALLY INSERTED BY 8259A)
COMPLETE 8086/8088 INTERRUPT TYPE
MEMORY ADDRESS OF 8086/8088 INTERRUPT TYPE (TYPE × 4)

Figure 22. Establishing Memory Address of 8086/8088 Interrupt Type

ICW3

The 8259A will only accept ICW3 if programmed in the cascade mode (ICW1, SNGL = 0). ICW3 is used for specific programming within the cascade mode. Bit definition of ICW3 differs depending on whether the 8259A is a master or a slave. Definition of the ICW3 bits is as follows

S0-7 (Master): If the 8259A is a master (either when the $\overline{SP}/\overline{EN}$ pin is tied high or in the buffered mode when M/S = 1 in ICW4), ICW3 bit definition is S0-7, corresponding to "slave 0-7". These bits are used to establish which IR inputs have slaves connected to them. A 1 designates a slave, a 0 no slave. For example, if a slave was connected to IR3, the S3 bit should be set to a 1. (S0) should be last choice for slave designation.

ID0-ID2 (Slave): If the 8259A is a slave (either when the $\overline{SP}/\overline{EN}$ pin is low or in the buffered mode when M/S = 0 in ICW4), ICW3 bit definition is used to establish its individual identity. The ID code of a particular slave must correspond to the number of the masters IR input it is connected to. For example, if a slave was connected to IR6 of the master, the slaves ID0-2 bits should be set to ID0 = 0, ID1 = 1, and ID2 = 1.

ICW4

The 8259A will only accept ICW4 if it was selected in ICW1 (bit IC4 = 1). Various modes are offered by using ICW4. Bit definition of ICW4 is as follows:

μPM: The μPM bit allows for selection of either the MCS-80/85 or MCS-86/88 mode. If set as a 1 the MCS-86/88 mode is selected, if a 0, the MCS 80/85 mode is selected.

AEOI: The AEOI bit is used to select the automatic end of interrupt mode. If AEOI = 1, the automatic end of interrupt mode is selected. If AEOI = 0, it isn't selected; thus an EOI command must be used during a service routine.

M/S: The M/S bit is used in conjunction with the buffered mode. If in the buffered mode, M/S defines whether the 8259A is a master or a slave. When M/S is set to a 1, the 8259A operates as the master; when M/S is 0, it operates as a slave. If not programmed in the buffered mode, the state of the M/S bit is ignored.

BUF: The BUF bit is used to designate operation in the buffered mode, thus controlling the use of the $\overline{SP}/\overline{EN}$ pin. If BUF is set to a 1, the buffered mode is programmed and $\overline{SP}/\overline{EN}$ is used as a transceiver enable output. If BUF is 0, the buffered mode isn't programmed and $\overline{SP}/\overline{EN}$ is used for master/slave selection. Note if ICW4 isn't programmed, $\overline{SP}/\overline{EN}$ is used for master/slave selection.

SFNM: **The SFNM bit designates selection of the** special fully nested mode which is used in conjunction with the cascade mode. Only the master should be programmed in the special fully nested mode to assure a truly fully nested structure among the slave IR inputs. If SFNM is set to a 1, the special fully nested mode is selected; if SFNM is 0, it is not selected.

4.2 OPERATIONAL COMMAND WORD (OCWs)

Once initialized by the ICWs, the 8259A will most likely be operating in the fully nested mode. At this point, operation can be further controlled or modified by the use of OCWs (Operation Command Words). Three OCWs are available for programming various modes and commands. Unlike the ICWs, the OCWs needn't be in any type of sequential order. Rather, they are issued by the processor as needed within a program.

Figure 23, the OCW programming format, shows the bit designation and short definition of each OCW. With the OCW format as reference, the functions of each OCW will be explained individually.

OCW1

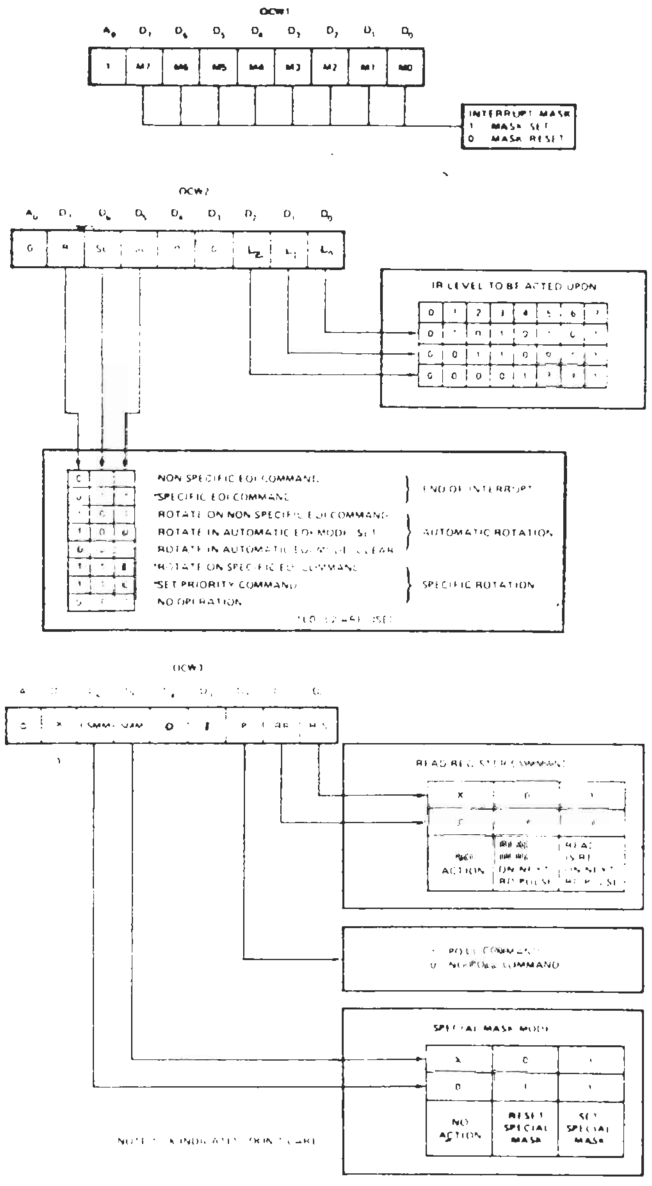
OCW1 is used solely for 8259A masking operations. It provides a direct link to the IMR (Interrupt Mask Register). The processor can write to or read from the IMR via OCW1. The OCW1 bit definition is as follows:

M0-M7: The M0-M7 bits are used to control the masking of IR inputs. If an M bit is set to a 1, it will mask the corresponding IR input. A 0 clears the mask, thus enabling the IR input. These bits convey the same meaning when being read by the processor for status update.

OCW2

OCW2 is used for end of interrupt, automatic rotation, and specific rotation operations. Associated commands and modes of these operations (with the exception of AEOI initialization), are selected using the bits of OCW2 in a combined fashion. Selection of a command or mode should be made with the corresponding table for OCW2 in the OCW programming format (Figure 20), rather than on a bit by bit basis. However, for completeness of explanation, bit definition of OCW2 is as follows:

L0-L2: The L0-L2 bits are used to designate an interrupt level (0-7) to be acted upon for the operation selected by the EOI, SL, and R bits of OCW2. The level designated will either be used to reset a specific ISR bit or to set a specific priority. The L0-L2 bits are enabled or disabled by the SL bit.



SOME OF THE TERMINOLOGY USED MAY DIFFER SLIGHTLY FROM EXISTING 8259A DATA SHEETS. THIS IS DONE TO BETTER CLARIFY AND EXPLAIN THE PROGRAMMING OF THE 8259A. THE OPERATIONAL RESULTS REMAIN THE SAME.

Figure 23. Operational Command Words (OCWs) Programming Format

EOI:	The EOI bit is used for all end of interrupt commands (not automatic end of interrupt mode). If set to a 1, a form of an end of interrupt command will be executed depending on the state of the SL and R bits. If EOI is 0, an end of interrupt command won't be executed.
SL:	The SL bit is used to select a specific level for a given operation. If SL is set to a 1, the L0-L2 bits are enabled. The operation selected by the EOI and R bits will be executed on the specified interrupt level. If SL is 0, the L0-L2 bits are disabled.
R:	The R bit is used to control all 8259A rotation operations. If the R bit is set to a 1, a form of priority rotation will be executed depending on the state of SL and EOI bits. If R is 0, rotation won't be executed.

OCW3

OCW3 is used to issue various modes and commands to the 8259A. There are two main categories of operation associated with OCW3: interrupt status and interrupt masking. Bit definition of OCW3 is as follows:

RIS:	The RIS bit is used to select the ISR or IRR for the read register command. If RIS is set to 1, ISR is selected. If RIS is 0, IRR is selected. The state of the RIS is only honored if the RR bit is a 1.
RR:	The RR bit is used to execute the read register command. If RR is set to a 1, the read register command is issued and the state of RIS determines the register to be read. If RR is 0, the read register command isn't issued.
P:	The P bit is used to issue the poll command. If P is set to a 1, the poll command is issued. If it is 0, the poll command isn't issued. The poll command will override a read register command if set simultaneously.
SMM:	The SMM bit is used to set the special mask mode. If SMM is set to a 1, the special mask mode is selected. If it is 0, it is not selected. The state of the SMM bit is only honored if it is enabled by the ESMM bit.
ESMM:	The ESMM bit is used to enable or disable the effect of the SMM bit. If ESMM is set to a 1, SMM is enabled. If ESMM is 0, SMM is disabled. This bit is useful to prevent interference of mode and command selections in OCW3.

THEORY OF OPERATION

U1-(74LS14 Hex Schmitt triggered inverter) used to generate a clean power on clear pulse and to generate the 4MHz clock signal which appears as the 2MHz CLOCK signal at pin 49.
U2-(74LS125 Quad tristate buffers) are used where individually controlled tristate buffered outputs were necessary.

U3-(74LS74 Dual D-flip flops) of which one is used to generate the I/O-PROM wait state and the other controls the software enable-disable of the "on board" PROM socket.

U4-(74LS221 Dual one shot retriggerable multivibrator) one shapes the incoming clock (0) pulses in the temporary master mode and makes them suitable to drive the 8088. The other generates the reset pulse to the 8088 upon temporary master transfer to the PC8088 board.

U5, U7, U40- (74LS00 Quad NAND gates) are general purpose, U5 is mainly used to gate the reset signals to the 8088, U7 is mainly used as the final PROM enabling gate and to enable the I/O port address. U40 has two spare gates.

U6- (74LS02 Quad NOR gate) are general purpose, mainly used in the I/O-PROM wait state generater and final stage to control the Data-in disable (DIDSBL) buffers.

U8, U12, U16- (74LS74 Dual D-flip flops) U8 is used to generate proper signal timing of PSYNC, PSTVAL and I/O-PROM wait state. U12 is the software control for the processor reset upon temporary master mode transfer and the other divides the 4MHz CLOCK signal from U1 down to the required 2MHz CLOCK output signal at S-100 pin 49. U16 handles the hand shaking transfer operation from the permanent mastser CPU to the temporary master (PC8088) CPU while in the temporaray master mode.

U9- (74LS85 4-bit comparator) is used to decode I/O port addressing by comparing bits A4 through A7 and DIP switch selection of S2 a-d.

U10- (74LSL175 Quad D-Latch) this is the extended addressing latch used to direct data bits D4 through D7 to address bits A20 through A23 when properly written to the correct extended addressing ports.

U11, U23- (74LS373 Octal tristate latches) these latch the multiplexed address bits A0 through A7 and A16 through A23 from the 8088 processor to the rest of the system.

U13- (8259 Programable vector interrupt controller) this is the heart of the 8 levels of vectored interrupts from the S-100 bus.

U14, U30- (74LS367 Hex tristate buffers) U14 is used mainly for general signal buffering. Two buffers are used together to implement the RST,POC functions. U30 drives the control signals onto the S-100 bus, i.e., PSYNC,PWR,PDBIN etc.

U15- (74LS08 Quad AND gate) two gates are used to coordinate the processor RDY signal, one is used to generate sMEMR and one operates DS1, the LED which indicates the PC8088 has control of the bus while in the temporary master mode.

U17, U25, U26- (74LS04 Hex inverters) used to invert and buffer various signals throughout the board though U25 and 2 gates of U26 are specifically used to invert the vector interrupt lines from off the S-100 bus to the 8259 (U13) vector interrupt chip. U17 has one spare gate.

U18- (74LS32 Quad OR gate) general use of which two are used in the I/O decoding circuitry.

U19- (74LS27 Triple 3 input NOR gate) used specifically in decoding the "on-board" PROM address location of FFF00 or FF000 for 2716/32.

U20- (74LS20 Dual 4 input NAND gate) same usage as U19.

U21- (74LS319 Dual 2-to-4 decoder) used to generate the individual I/O port select signal to the various "onboard" I/O port circuits.

U22- (2716/32 EPROM socket) selectable EPROM type via jumper J1.

U24- (8088 Central Processing Unit) the heart and brains of the PC8088 CPU board.

U27, U32, U33, U34, U29- (74LS244 Octal non-inverting tristate drivers) are used, except for U29, to buffer and drive the eight Data in and Data out lines and address lines A8 through A23. U29 comprises half of the special bidirectional disable (DSBL) circuitry.

U28, U35- (74LS240 Octal inverting tristate drivers) U28 comprises the other half of the disable (DSBL) circuitry. U35 drives the S-100 status signals to the bus, i.e., sM1, SINTA etc.

U31- (74LS245 Octal bidirectional tristate bus transceivers) this allows address bits A0 through A7 to be generated either on or off the PC8088 board. This lets the permanent master access to the PC8088 I/O ports.

U36- (74LS138 3-to-8 decoder) decodes the status signals from the 8088 status lines (IO/M*, DT/R*, SSO*) and buffers them through U35.

U37- (8284A Clock generator for the 8086/88) this generates the 8088 CPU clock signal in the permanent master mode. It uses Y2, a 15 or 24 Mhz crystal depending on the speed of the 8088 CPU.

U38, U39- (7805 Positive 5 Volt regulators @ 1.0 Amp) provides the the regulated 5 Volt supply to all the "on-board" circuitry.

SWITCHES AND JUMPERS

SWITCH PACK ONE (S1):

S1a, b, c, d- Connects ADSBL*, DODSB*L, CDSBL*, SDSBL* respectively to HOLD of the 8088 pin 31 of U24. All four are closed for the temporary master mode.

S1e- Connects S-100 signal PHLDA (pin 26) to the control signal buffer U30 pin 11. Closed for permanent master mode.

S1g- Enables the I/O, PROM read wait state generator. When this switch is closed, both an I/O and "on-board" prom wait state is generated. This is especially useful for slow I/O and slow EPROMS. A must for 8Mhz upgrade.

S1h- "On-board" EPROM enable. Open this switch and the PROM is completely disabled.

SWITCH PACK TWO (S2):

S2a, b, c, d- Selects upper four I/O port bits A4, A5, A6, A7 respectively. S2d=A7, S2c=Ag, S2b=A5, S2a=A4 O=open C=closed.

S2	d/A7	c/A6	b/A5	a/A4
0X	C	C	C	C
1X	C	C	C	O
2X	C	C	O	C
3X	C	C	O	O
4X	C	O	C	C
5X	C	O	C	O
6X	C	O	O	C
7X	C	O	O	O
8X	O	C	C	C
9X	O	C	C	O
AX	O	C	O	C
BX	O	C	O	O
CX	O	O	C	C
DX	O	O	C	O
EX	O	O	O	C
FX	O	O	O	O

S2e- This is the master mode control switch. If this switch is:
OPEN: Board buffers are in temporary master mode.
CLOSED: Board buffers are in permanent master mode.

S2f- Mwrite clock signal enable. Close to enable, open to disable.

S2h- POC* enable. Close to enable, open to disable. Leave open if a power on clear is generated anywhere else on the bus.

JUMPERS:

J1- PROM type selection.
For a; 2716- J1 a to b & J1 c to d
2732- J1 e to f & J1 b to c

J2- Non-maskable interrupt selection.
To select S-100 signal NMI*pin 12 to U24 pin 17, jumper J2 b to c.
To select S-100 signal Error*pin 98 to U24 pin 17, jumper J2 a to d.

J3- 8088 operating clock fequency. In the temporary master mode for master CPU clock (0) frequencies of up to 3 MHz, jumper J3 d to b. For master CPU clock (0) frequencies of 3MHz or higher, jumper J3 a to b. In the permanent master mode, jumper J3 b to c.

NOTE: In some instances in the temporary master mode it may be necessary to try both J3 d to b and J3 a to b for master frequencies of 3 to 4 MHz in some systems for continuous operation.

J4- Jumper J4 a to b for permanent master mode. Leave J4 a to b open for temporary master mode.

J5- Ready select. Jumper J5d c to d for temporary master mode and J6 a to b for permanent master mode.

J7- TEST* option. J7 is trace selected between J7 a to c. This deselects this option. If the test feature is to be used, trace jumper a to c must be cut and jumper J7 a to b must be connected. The actual operation of the test feature is left to the user.

J8- Phantom enable. J8 is preenabled by a trace between a and b. This enables the phantom option. If no other board on the bus uses the phantom option, then this may be left intact. If the phantom needs to be disabled from the bus, cut the trace between J8 a and b.

J9- Interrupt select. To use the S-100 INT* signal (pin 73), jumper J9 a to c. This connects the INT* signal to the 8259 vector interrupt chip. It becomes the highest priority of the eight levels. In this configuration, VI0 pin 4 of the S-100 bus is not used. To connect the VI0 on the S-100 bus to the 8259 vector interrupt chip, jumper J9 b to c. This configuration disconnects the INT* line, pin 73, from use and connects VI0 through VI7 to the vector interrupt chip.

J53- lEEE 696 requires S-100 pin 53 to be ground while many older CPU boards use 53 as SSW-DISABLE. If your bus grounds pin 53 then you may jumper pin 53 otherwise you should leave it unjumpered.

MODE SELECT-SWITCH AND JUMPER GUIDE

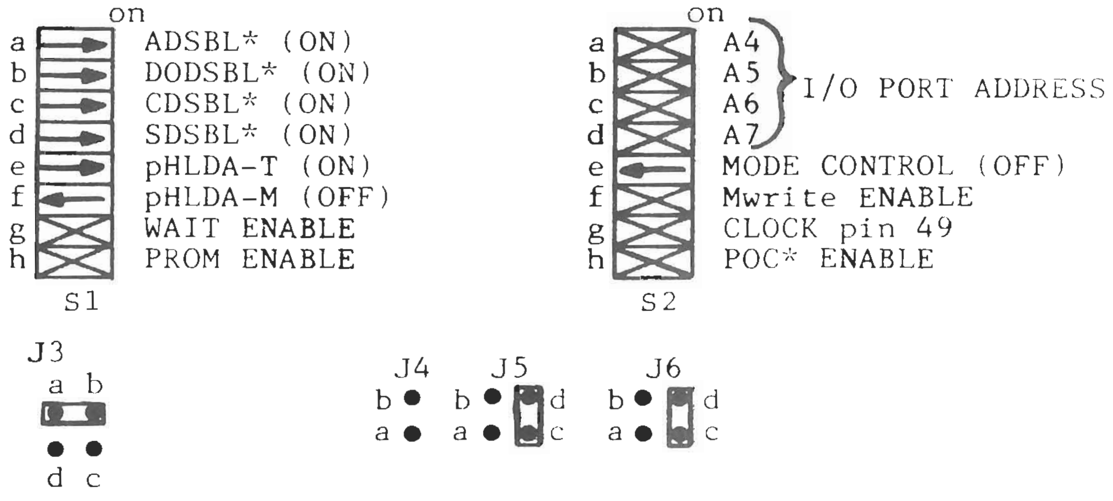
on

☒ Don't care, user defined

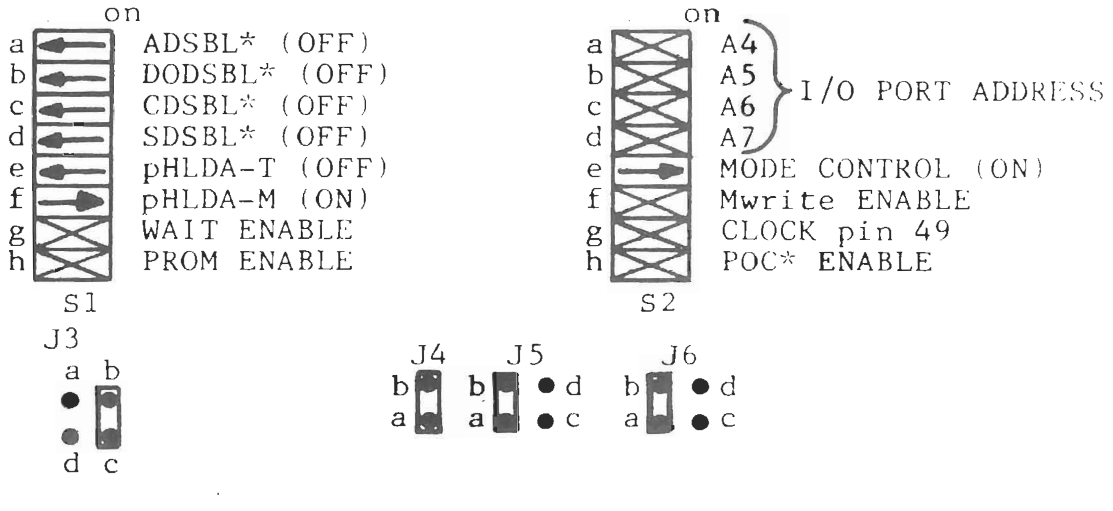
← Push OFF (left)

→ Push ON (right)

Temporaty Master Mode:



Permanent Master Mode:



SOFTWARE HELP

As was mentioned earlier, control is passed to and from the PC8088 board (in the temporary master mode only) by either the master CPU or the PC8088 CPU doing an output instruction to the transfer port 'XD'. There are a few important points to remember when writing code that uses this transfer between CPU's.

1) If the PC8088 is to always begin code execution at its reset address of (F)FFF0H then do nothing to port 'XC'. But if the 8088 is to pick up execution from where it left off (after the output instruction to port 'XD'), then there must be an output to port 'XC' during the code. This usually takes place during initialization of the progr am. This turns off the reset upon transfer option and is how the PC8088 will work during normal program execution. Either a system reset or another output to port 'XC' will return the PC8088 to the reset on transfer mode.

2) The Intel 8086 and 8088 processor chips have a small block of RAM internally called the instruction queue. This queue stores several instructions that are ahead of the present instruction being executed. Most always, this advanced instruction fetching is a very large asset to the speed of the processor. There are a very few instances, though, where this queue can be a hindrance. During the transfer from the PC8088 to the 8-bit CPU, the queue has a distinct habit of stopping internal program execution half way into the next instruction after the transfer to the 8-bit CPU. When control is passed back to the PC8088, this half executed instruction may cause a misalignment of the program counter and the 8088 may go into oblivion. The way to cure this is by a jump to the output transfer instruction. Any time program control is passed to a new program location within the 8088 code (such as a jump) the queue is reset to zero and execution picks up proper-ly when control is passed back to the PC8088. This is how the transfer can look:

```
    _________________________       _____________________________
   !PC8088 is executing code!       !8 bit transfers to 8088    !<--
   !_________________________!      !by a OUT XDH               !  !
              !              -<-    !---------------------------!  !
              v                !    !8 bit now in a HOLD        !  !
                               !    !---------------------------!  !
    _________________________  !                                   
 <-!Jump to TRANSFER routine!  ! ->!8 bit CPU picks up program!   !
 ! !_________________________! ! !  !___________________________!  !
 !                             ! !              v                  !
-!>!PC8088 continues program!  ! !   ___________________________   !
!! !_________________________! ! !  !executes code and com-     !  !
!!           !                 ! !  !pletes its task            !  !
!!           v                 ! !  !___________________________!  !
!!  _________________________  ! !              !                  !
!! !PC8088 transfers control!  ! !              v                  !
!->!to the 8-bit by a        ! ! !   ___________________________   !
!  !   OUT XDH               ! ! !  !jumps to its transfer      !  !
!  !-------------------------!-!>-  !routine back to the PC8088!->-
!  !Queue clear, CPU holding! !     !___________________________!
!  !-------------------------!<-
!  !8088 picks up execution,!
-<-!may jump back to code    !
   !_________________________!
```

PARTS LIST

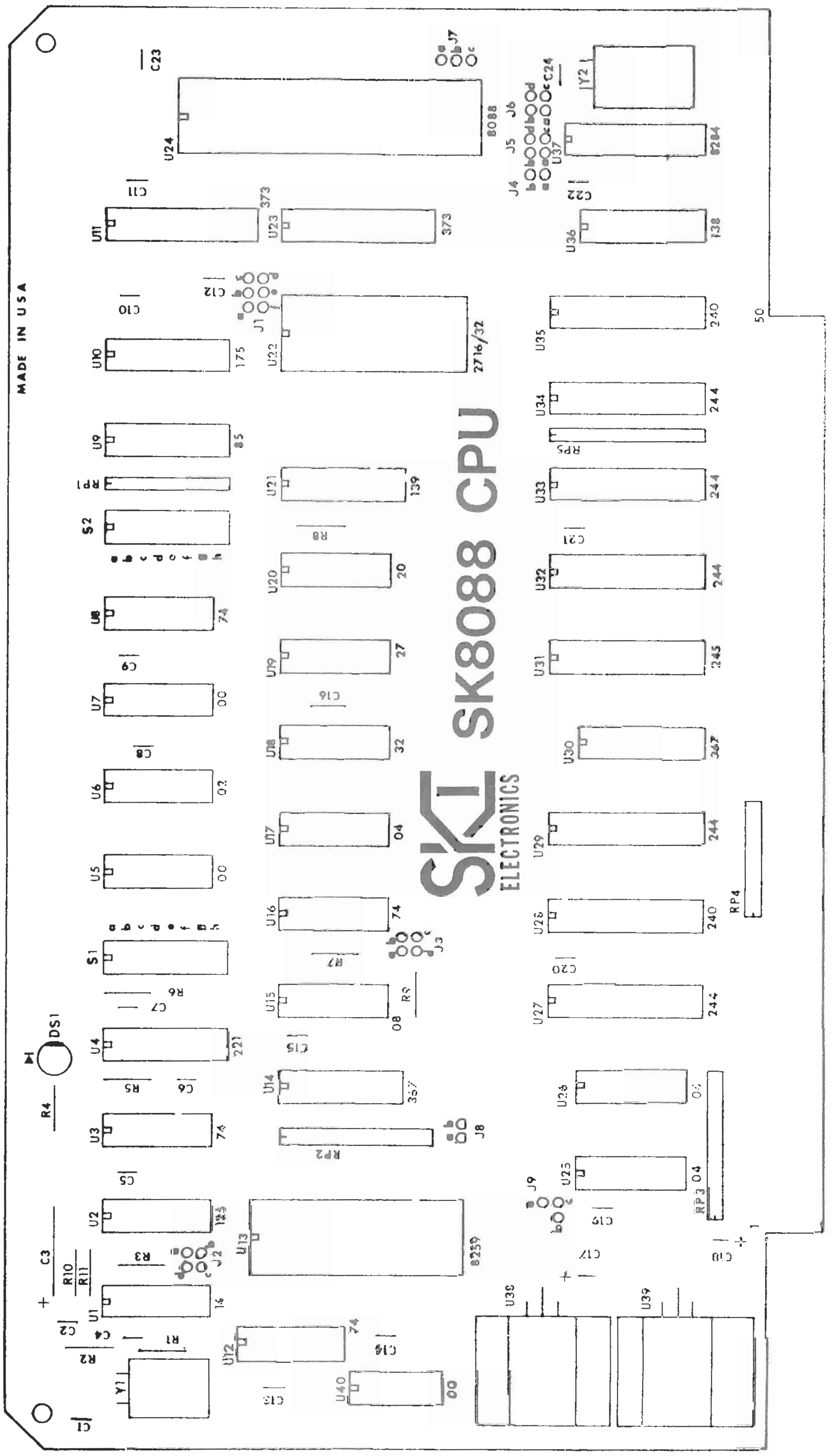
Quantity	Location	Description
1	U1	74LS14 Hex schmitt inverters
1	U2	74LS125 Quad tristate buffers
4	U3,8,12,16	74LS74 Dual D-flip flops
1	U4	74LS221 Retrig. mono. multivibrator
2	U5,7,40	74LS00 Quad NAND gate
1	U6	74LS02 Quad NOR gate
1	U9	74LS85 4 bit comparator
1	U10	74LS175 4 bit latch
2	U11,23	74LS373 Octal tristate latch
1	U13	8259A Programmable vector int. controller
2	U14,30	74LS367 Hex tristate buffers
1	U15	74LS08 Quad AND gate
3	U17,25,26	74LS04 Hex inverters
1	U18	74LS32 Quad OR gate
1	U19	74LS27 Triple three input NOR gate
1	U20	74LS20 Dual four input NAND gate
1	U21	74LS139 Dual 2-4 decoder
1	U22	Socket for 2716/32 EPROM
1	U24	8088 16 bit internal CPU
5	U27,29,32,33,34	74LS244 Octal tristate buffers
2	U28,35	74LS240 Octal inv. tristate buffers
1	U31	74LS245 Octal tranceivers tristate
1	U36	74LS138 3-8 decoder
1	U37	8284A Clock generator for 8086/88
2	U38,39	7805 Positive 5 volt regulators TO-220
16	C1,5,8-16,19-23	.1uf ceramic disc cap. (bypass)
2	C2,24	10pf ceramic disc cap.
1	C3	25-40uf electrolytic cap. 15V
1	C4	.01uf ceramic disc cap.
1	C6	20pf ceramic disc cap.
1	C7	330pf ceramic disc cap.
2	C17,18	15-25uf dipped tantalum cap. 15V
3	RP2,3,5	2.7K 10 pin resistor SIP pack
2	RP1,4	2.7K 8 pin resistor SIP pack
2	R1,2	820 ohm 1/4W
2	R3,9	1K ohm 1/4W
1	R10	560 ohm 1/4W
1	R11	270 ohm 1/4W
1	R6	10K ohm 1/4W
1	R5	3.9K ohm 1/4W
1	R4	390 ohm 1/4W
2	R7,8	2.7K ohm 1/4W
1	RP6	10K 10 pin resistor SIP pack

P A R T S   L I S T   C O N T .

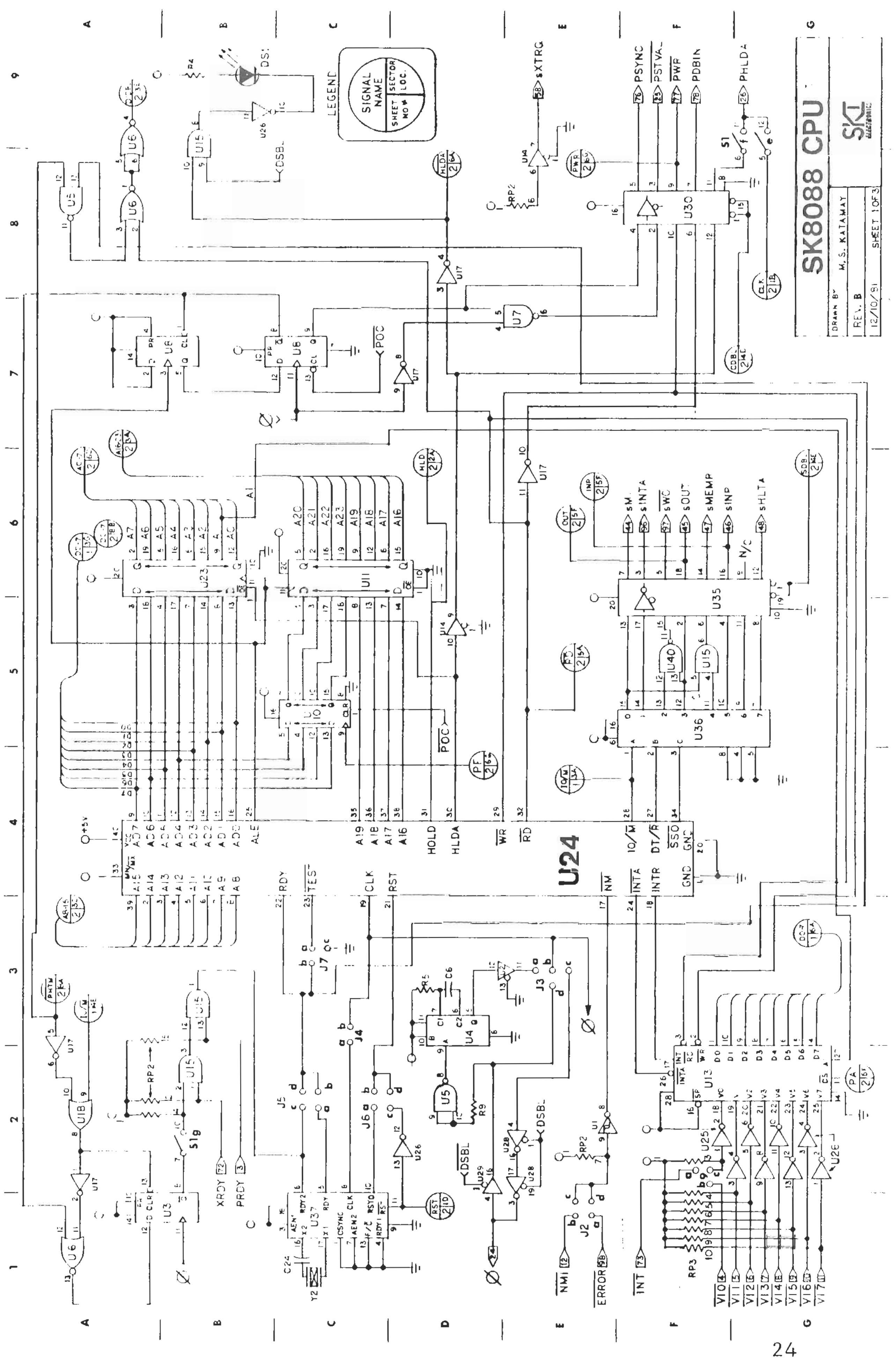
Quantity	Location	Description
2	S1,2	8PST DIP switch
1	Y1	4Mhz crystal
1	Y2	15Mhz crystal
1		40 pin socket
1		28 pin socket
1		24 pin socket
10		20 pin sockets
1		18 pin socket
7		16 pin sockets
16		14 pin sockets

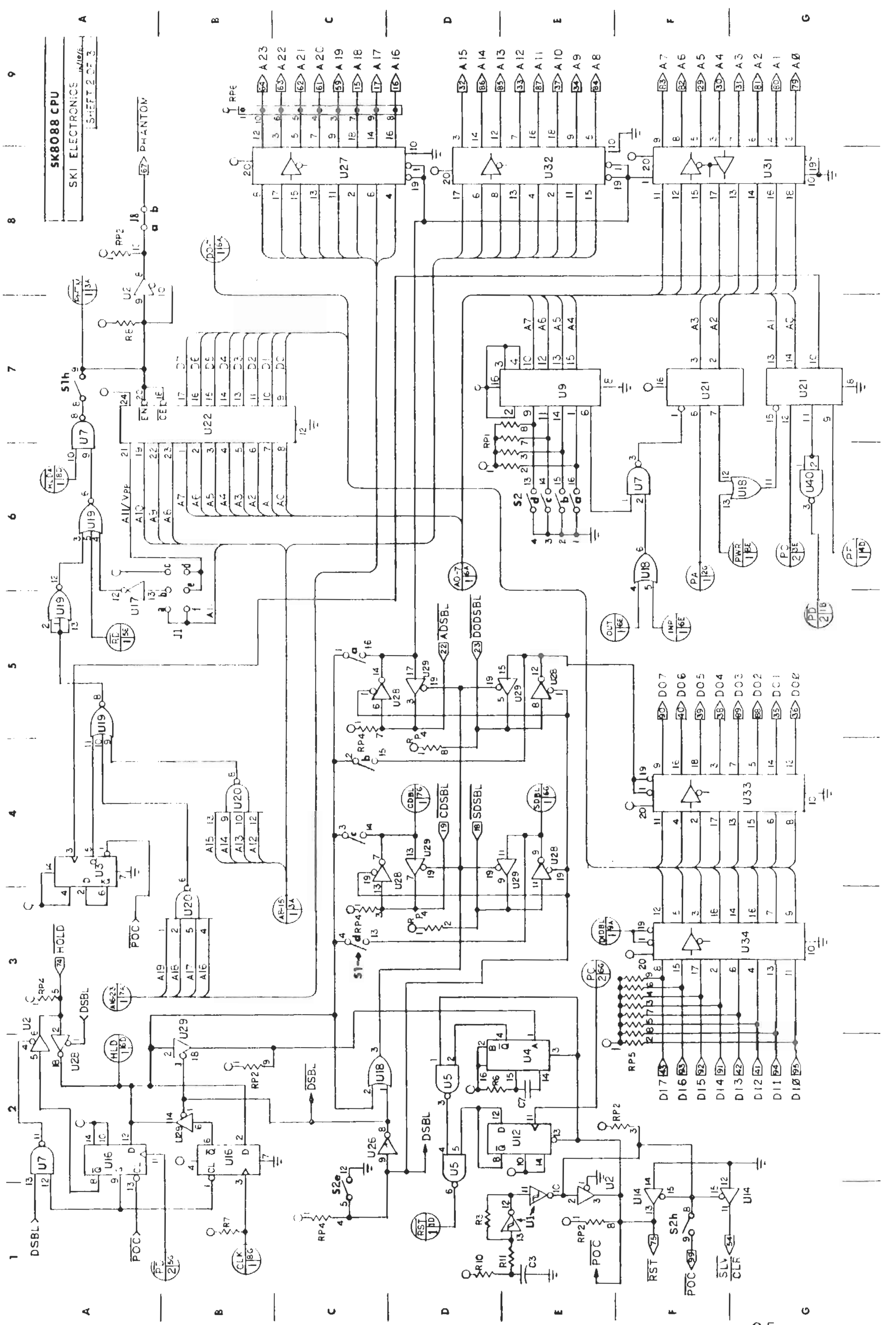
Misc. parts

2		TO-220 heat sinks
1	DS1	Rectangular L.E.D.
32		Jumper or W/W pins
8		Jumper of shorting bars
2		Card ejectors with pins
1		SK8088 PC Board
2		4-40 x 5/16" screws
2		4-40 hex nuts

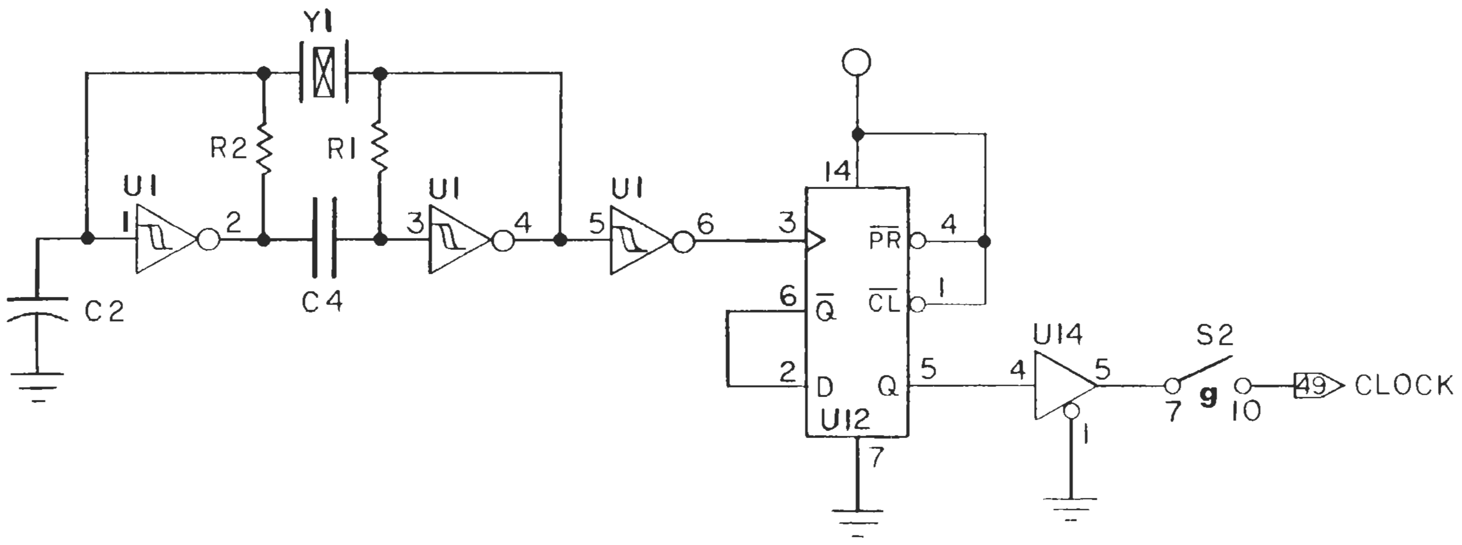


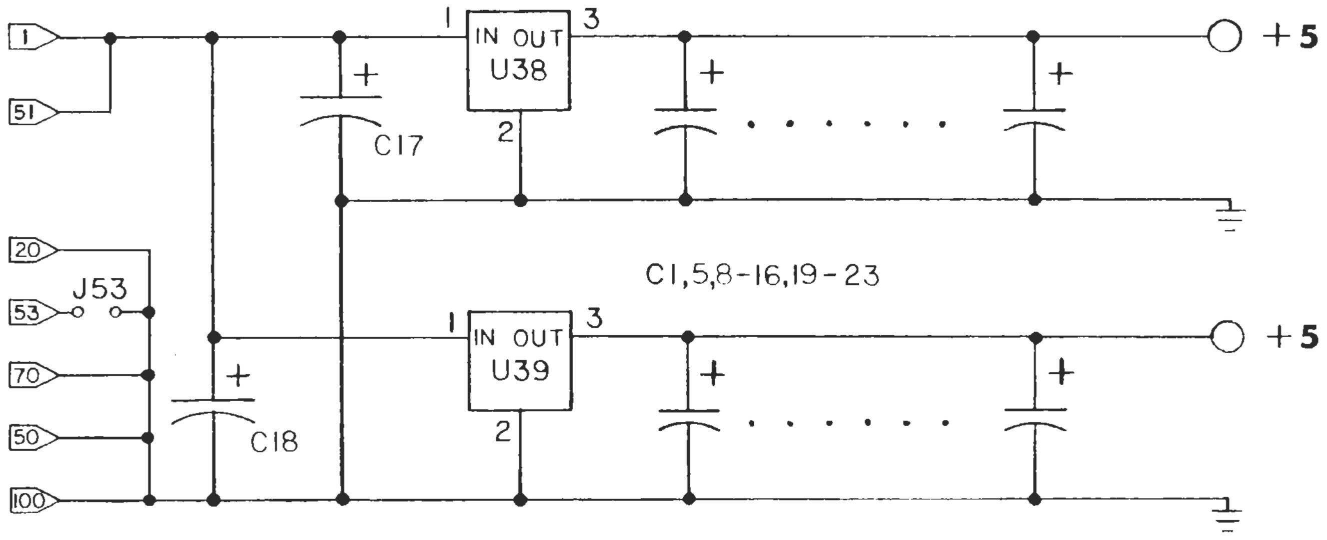
ASSEMBLY DRAWING

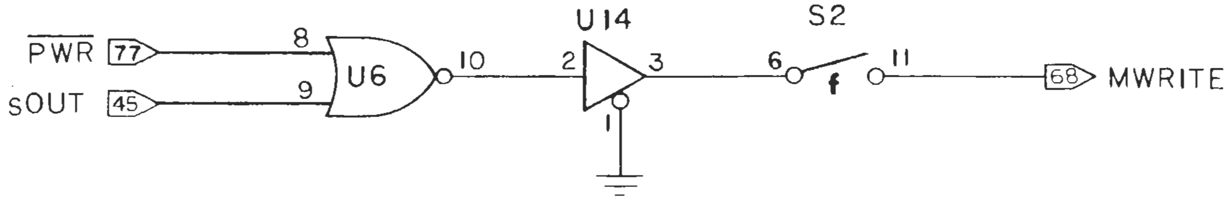

SK8088 CPU
SKI ELECTRONICS
DRAWN BY M. S. KATAMAY
REV. B
12/10/91
SHEET 1 OF 3
LEGEND
SIGNAL NAME
SHEET NO.
SECTOR LOC.
U24
U23
U11
U13
U30
U35
U36
U37
PSYNC
PSTVAL
PWR
PDBIN
PHLDA
sXTRG
sM1
sINTA
sWO
sOUT
sMEMR
sINP
sHLTA
XRDY
PRDY
NMI
ERROR
INT
VI0
VI1
VI2
VI3
VI4
VI5
VI6
VI7
DSBL
POC



SK8088 CPU
SKI ELECTRONICS
SHEET 2 OF 3
PHANTOM
HOLD
DSBL
POC
CLK
ADSBL
DODSBL
CDSBL
SDSBL
RST
SLV CLR
U27
U32
U31
U22
U9
U21
U33
U34
U4
U12
U16
U19
U20
U3
A23 A22 A21 A20 A19 A18 A17 A16
A15 A14 A13 A12 A11 A10 A9 A8
A7 A6 A5 A4 A3 A2 A1 AØ
DO7 DO6 DO5 DO4 DO3 DO2 DO1 DOØ
DI7 DI6 DI5 DI4 DI3 DI2 DI1 DIØ







SK8088 CPU
SKI ELECTRONICS
12/10/81 M.S. Katamay SHEET 3 OF 3

```
  MONINSTR.TXT          PC8088 MONITOR  VERS. 1.0
*******************************************************************
NOTE:THIS VERS. RUNS WITH THE PC8088 IN THE TEMPORARY MASTER MODE ONLY!
*******************************************************************
LAST MODIFIED 8/9/82 MSK
```

THE PC88MON IS A HANDY AID IN GETTING TO KNOW HOW THE PC8088 WORKS AND ALLOWS YOU TO GET MORE FAMILIAR WITH 8086/88 CODE AND OPERATION WITHOUT THE NEED FOR EXPENSIVE SOFTWARE. IT IS PROVIDED WITH THE PC8088 CPU BOARD TO GIVE THE USER A MEANS TO TEST THE PC8088 BOARD IN HIS/HER SYSTEM. THOUGH THIS MONITOR IS NOT AN EXTREMELY POWERFUL MONITOR DEBUGGER, IT DOES ALLOW THE INSERTION OF ONE BREAK POINT IN A PROGRAM. THIS LETS THE USER KNOW IF HE/SHE HAS REACHED A CERTAIN POINT IN A PROGRAM. THE MONITOR ALSO ALLOWS ACCESS TO ALL 16 SEGMENTS IN WHICH THE 8088 PROCESSOR CAN DIRECTLY ADDRESS.

TO ALLOW THIS MONITOR TO BE AS UNIVERSAL AS POSSIBLE WITH ALL THE THOUSANDS OF I/O CONFIGURATIONS OUT THERE, THE USER HAS TO FIRST CREATE A SMALL PERSONALIZED I/O INTENSIVE PROGRAM THAT CAN EITHER BE LOADED IN OFF OF A DISK DRIVE (UNDER CP/M-80) OR A CASSETTE OR IT CAN BE STORED IN A PROM ELSEWHERE IN THE SYSTEM. THE ONLY REQUIREMENT IS THERE BE FREE MEMORY OF 40 TO 80 HEX BYTES OF RAM AT THE BASE SEGMENT AND STARTING AT 0000H TO 0040H OR 0080H. THERE IS A OUTLINE GIVEN IN THIS TEXT OF THE PERSONALIZED I/O ROUTINE. THIS IS TO BE FOLLOWED AS CLOSELY AS POSSIBLE WITH ONLY THE BASIC CONSOLE PARAMETERS MODIFIED TO FIT YOUR SYSTEM. VIDEO DRIVERS CAN BE USED AS LONG AS THEY ARE ENTERED WITH THE SAME FLOW AS GIVEN IN THE EXAMPLE. THIS OUTLINE GIVEN IS AN ACTUAL PROGRAM USED WHICH IS LOADED IN FROM A CP/M-80 DISK TO THE STANDARD LOCATION OF 0100H. ALL THE STACKS ARE LOCATED AT 00080H AND LOWER. THIS IS THE OPTIMAL FORM SINCE IT ALLOWS BEING STORED IN RAM OR IN A PROM. ONCE THIS I/O PROGRAM WAS WRITTEN, IT WAS STORED ON DISK UNDER THE TITLE OF 'MON88.COM'. THEN ANY TIME IT NEEDED TO BE ENTERED, BRINGING IN THE FILE 'MON88' WOULD GET US INTO THE MONITOR. IT WAS ALSO, IN THIS SAME FORM AS THE OUTLINE, BURNED INTO PROM AT LOCATION 0F000H, AND PUT INTO A POWER ON JUMP LOCATED PROM SOCKET. THIS ALLOWED US TO ENTER THE MONITOR UPON POWER UP. THOUGH IN THE PROM VERSION SOME EXTRA INITIALIZATIONS WERE INCLUDED BEFORE THE START LOOP FOR CONSOLE INITIALIZATIONS, SO BE AWARE OF THIS BEFORE YOU PUT IT INTO A PROM.

THERE ARE A FEW POSSIBLY CONFUSING EQUATES THAT WE WOULD LIKE TO TRY TO CLEAR UP. ONE IS THE PORT EQUATE, IN THE OUTLINE, PORT IS EQUAL TO 00H. THIS MEANS THE PORTS ON THE PC8088 HAVE BEEN SET TO 0XH. WHERE X IS THE EIGHT POSSIBLE PORTS ON THE PC8088, IE. CLEAR ON TRANSFER IS PORT 0CH, TRANSFER PORT IS 0DH,ETC.. PLEASE READ THAT SECTION OF THE MANUAL IF THIS IS NEW TO YOU. SAY ONE WANTED THE PC8088 PORTS TO BE ADDRESSED AT 4XH, THEN THE EQUATE WOULD READ LIKE THIS

```
THIS            PORT    EQU     40H .
```

IT WILL ALWAYS BE ONE OF THESE NUMBERS-00H,10H,20H,30H,40H,50H,60H, 70H,80H,90H,0A0H,0B0H,0C0H,0D0H,0E0H,OR 0F0H, SINCE THE LEAST SIGN-IFICANT 4 BITS ARE ALREADY HARDWARE SELECTED. THE STACKL EQUATE WHICH IS LOCATED AT ADDRESS 0002H CONTAINS THE ADDRESS OF THE TOP OF STACK FOR THE 8088 PROCESSOR. IN THE OUTLINE, IT IS 0070H, SINCE IT NEEDS MUCH MORE STACK AREA THAN DOES THE 8 BIT I/O PROGRAM YOU WROTE (THE OUTLINE) WHICH ITS STACK IS LOCATED AT 0080H OR 16 BYTES LONG.

*********************** IMPORTANT FEATURE *****************************
THE PC8088 HAS A BUILT IN FEATURE THAT ALLOWS THE MONITOR TO ACT AS A JUMP INTO CP/M 86 IF IT WAS LOADED BEFORE ENTERING THE PROM. IN OTHER WORDS, IF YOU HAVE CP/M 86 WITH THE PC8088 RUNNING IN THE TEMPORARY MASTER MODE AND THE BIOS IS OF THE FORM GIVEN IN THE PC8088 CPU MANUAL, THEN WHEN THE 8 BIT CPU PASSES INITIAL CONTROL TO THE PC8088, THE MONITOR WILL DIRECT THE 8088 TO LOCATION 2900H. THIS IS WHERE THE BIOS JUMP TABLE IS LOCATED FOR THE STANDARD VERSION OF CP/M 86. YOU WILL HAVE TO BECOME FAMILIAR WITH CP/M 86 AND THE PC8088 CPU IN TEMPORARY MASTER MODE TO APPRECIATE THIS FEATURE. REMEMBER, THIS FEATURE ONLY WORKS IF CP/M 86 IS CONFIGURED WITH THE PC8088 CPU IN THE TEMPORARY MASTER MODE, THE 8 BIT IS CONTROLLING THE 86 BIOS AND THAT CP/M 86 IS LOCATED AT THE STANDARD OFFSET OF 0400H.
****

COMMANDS IN THE MONITOR ARE ENTERED AS A LETTER PROCEEDED BY SOME APPROPRIATE PARAMETERS. SINCE THIS MONITOR ALLOWS OPERATIONS IN ALL SEGMENTS, THE CURRENT WORKING SEGMENT FOR THAT COMMAND IS THE FIRST PARAMETER ENTERED, FOLLOWED BY A COMMA, THEN ANY OTHER NEEDED PARAMETERS ARE ENTERED. THE DEFAULT SEGMENT IS SEGMENT 0000 OR BASE SEGMENT AND IS DEFAULTED TO BY ENTERING A SPACE AFTER A COMMAND LETTER PROCEEDED BY OTHER PARAMETERS. EXAMPLES ARE GIVEN IN THE EXPLANATIONS OF THE FIRST COUPLE OF COMMANDS. PRESENTLY, ONE CAN ONLY DO OPERATIONS WITHIN ONE SEGMENT. AS AN EXAMPLE YOU CAN'T MOVE DATA FROM ONE SEGMENT TO ANOTHER, ONLY MOVE BLOCKS OF DATA WITHIN ONE SEGMENT.
****** NOTE: YOU MUST ALWAYS DEFINE A SEGMENT FOR EACH COMMAND ******
WHETHER IT BE EXPLICIT 'IE. 2000,' OR DEFAULTED 'IE. >(SPACE)X,'

ANYWHERE YOU SEE A <CR> THAT MEANS ENTER A CARRIAGE RETURN.
WHEN YOU SEE A CNTL- YOU MUST PRESS THE CONTROL KEY WHILE PRESSING THE LETTER NEXT TO IT, IE. CNTL-C PRESS CONTROL KEY AND C KEY.
ALSO WHEN ENTERING PARAMETERS, THE LAST FOUR KEYS ENTERED WILL BE VALID, UNLESS ITS A SINGLE BYTE DATA PARAMETER, THEN IT WOULD BE THE LAST TWO ENTRIES THAT WOULD BE VALID. SOME EXAMPLES ARE:

12345678	WOULD ENTER 5678 AS THE PARAMETER
0	WOULD ENTER 0000 AS THE PARAMETER
23	WOULD ENTER 0023 AS THE PARAMETER
'SPACE'	WOULD DEFAULT TO 0000 AS THE PARAMETER.

EITHER UPPER OR LOWER CASE MAY BE ENTERED AS THE MONITOR CONVERTS EVERYTHING TO UPPER CASE. ANY ERRONEOUS ENTRIES ARE FLAGGED WITH A * AND YOU ARE GIVEN BACK A PROMPT. THIS DOES EXCLUDE WHEN A CNTL-D OR OTHER TERMINATOR IS USED, YOU WILL GET THE * THEN A NEW PROMPT.
A CNTL-C WILL MOST ALWAYS TERMINATE EXECUTION AND GET YOU BACK INTO THE MONITOR. WHEN SCROLLING THROUGH LENGTHY DISPLAYS, HITTING ANY KEY, THOUGH A 'SPACE' KEY IS RECOMMENDED, WILL STOP THE DISPLAY. HITTING A KEY AGAIN WILL RESTART THE DESPLAY. AS ALWAYS A CNTL-C WILL TERMINATE THE EXECUTION.
THROUGHOUT THIS TEXT:

(SEGMENT)	MEANS FOUR HEX NUMBERS TO DESCRIBE THE WORKING SEGMENT IE. 2000,'SPACE',4300,0000,98,ETC..
(ADDRESS)	MEANS FOUR HEX NUMBERS TO DESCRIBE AN ADDRESS.
(ADDRESS 1)	MEANS FOUR HEX NUMBERS DESCRIBING THE BEGINING OF A BLOCK OF MEMORY, IE.3401,0021,21,3888,456,ETC..
(ADDRESS 2)	MEANS FOUR HEX NUMBERS DESCRIBING THE END OF THE BLOCK.
(ADDRESS 3)	MEANS FOUR HEX NUMBERS DESCRIBING THE BEGINNING OF THE SECOND BLOCK OF MEMORY AND HAVING A LENGTH EQUAL TO THE BLOCK BOUNDED BY (ADDRESS 1) AND (ADDRESS 2).
(DATA)	MEANS TWO HEX NUMBERS DESCRIBING ENTERED DATA.

WE WILL NOW DESCRIBE EACH COMMAND:

-A- ENTER ASCII TEXT INTO MEMORY. ALLOWS THE USER THE ABILITY TO ENTER FROM THE KEYBOARD ASCII CHARACTERS INTO A DEFINED MEMORY BLOCK. YOU CAN BACKSPACE IN THIS COMMAND IF YOU ENTER A BACK SPACE KEY OR A LEFT ARROW. ANY ENTERED TEXT THAT IS BACK SPACED OVER IS LOST AND MUST BE RETYPED.
THE FORM USED IS AS FOLLOWS:

```
>A0000,2200<CR>    ENTERS ASCII TEXT STARTING IN SEGMENT 0000 AND AT
                   LOCATION 2200H. TO TERMINATE THE COMMAND YOU MUST
                   ENTER A CNTL-D, THE MONITOR WILL THEN COME BACK AND
                   TELL YOU WHAT SEGMENT AND THE LAST ADDRESS WRITTEN.
THIS EXAMPLE COULD ALSO HAVE BEEN DONE USING THE DEFAULT LIKE THIS:
> 2200<CR>         NOTE THE SPACE AFTER THE PROMPT DEFAULTS THE SEGMENT.
                   THIS IS THE CASE FOR ALL SEGMENT RELATED COMMANDS.
```

-B- SET A BREAK POINT IN A PROGRAM. ALLOWS THE USER THE ABILITY TO SET ONE BREAK POINT. IF THAT BREAK POINT IS REACHED (PROBABLY AFTER A GOTO TO A PROGRAM) THE MONITOR WILL TELL YOU WHAT SEGMENT AND AT WHAT LOCATION THE BREAK POINT WAS ENCOUNTERED LEAVING THE PROGRAM INTACT AND READY TO BE RUN AGAIN. FORM USED IS AS FOLLOWS:

```
>B(SEGMENT),(ADDRESS)<CR>  YOU WILL THEN GET ANOTHER PROMPT FOR THE
                           NEXT COMMAND. AGAIN, THE SEGMENT COULD
                           HAVE BEEN DEFAULTED TO SEGMENT 0000 BY
                           JUST ENTERING A SPACE IN PLACE OF THE
                           (SEGMENT), EXAMPLE;
>B (ADDRESS)<CR>
```

-C- COMPARE TWO BLOCKS OF MEMORY FOR EQUAL VALUE IN A GIVEN SEGMENT

```
>C(SEGMENT),(ADDRESS 1),(ADDRESS 2),(ADDRESS 3)<CR>
```

WHERE ADDRESS 1 IS THE BEGINNING FOR THE FIRST BLOCK, ADDRESS 2 IS THE END OF THE FIRST BLOCK AND ADDRESS 3 IS THE BEGINNING OF THE SECOND BLOCK OF EQUAL LENGTH TO THE FIRST. ANY DISCREPANCIES BETWEEN THE TWO BLOCKS WILL BE FLAGGED. AS IS ALWAYS THE CASE, THE SEGMENT CAN BE DEFAULTED BY A SPACE.

-D- DISPLAY A BLOCK OF MEMORY IN HEX. THE USUAL FORM IS AS FOLLOWS:

```
>D(SEGMENT),(ADDRESS 1),(ADDRESS 2)<CR>
```

THIS COMMAND DISPLAYS A BLOCK OF MEMORY BOUNDED BY ADDRESS 1 AND ADDRESS 2 IN THE DEFINED SEGMENT IN ITS HEX FORMAT. THE DISPLAY WILL GIVE YOU SEGMENT:ADDRESS THEN THE 16 BYTES OF DATA.

-E- ENTER HEX DATA INTO MEMORY. THE GENERAL FORM IS:

```
>E(SEGMENT),(ADDRESS)<CR>  THE MONITOR THEN DISPLAYS THE HEX DATA
                        ALREADY IN THAT LOCATION FOLLOWED BY A -.
                        YOU MAY ENTER TWO HEX DIGIT OR ENTER A
                        'SPACE' WHICH WILL GET YOU TO THE NEXT BYTE
                        OF DATA AND LEAVE THE PREVIOUS BYTE UNCHANGED.
                        ENTER A <CR> TO LEAVE THE COMMAND.
```

-F- FILL A BLOCK OF MEMORY WITH A BYTE OF DATA. THE FORM IS:

```
>F(SEGMENT),(ADDRESS 1),(ADDRESS 2),(DATA)      AN EXAMPLE IS:

>F 2000,2200,55<CR>   WILL FILL A BLOCK OF MEMORY IN SEGMENT 0000
                      STARTING FROM 2000 UNTIL AND INCLUDING 2200 WITH
                      THE DATA BYTE  55H.
```

-G- GO TO AN ADDRESS IN MEMORY AND BEGIN EXECUTION. THE FORM IS:

```
>G(SEGMENT),(ADDRESS)
```

NOTE: IF A BREAK POINT HAS PREVIOUSLY BEEN SET AND EXECUTION GETS TO THE BREAK POINT, EXECUTION WILL STOP, AND A MESSAGE WILL TELL YOU WHERE THE BREAK POINT WAS REACHED. THE PROGRAM WILL BE LEFT INTACT READY TO GO AGAIN.
YOU WILL END UP IN THE MONITOR IF A BREAK POINT IS REACHED.

-H- HEX MATH   THE SUM AND DIFFERENCE OF TWO FOUR DIGIT HEX NUMBERS WILL BE GENERATED AND PRINTED ON THE SCREEN. THE FIRST NUMBER IS THE DESTINATION NUMBER, THE SECOND IS THE SOURCE. THE FORM IS:

```
>H6000,2000<CR>   NOTICE NO SEGMENT IS NEEDED FOR -H- WILL NOT SUM
               ACROSS SEGMENT BOUNDARIES, ONLY WITHIN A SEGMENT.
```

THE ANSWER GIVEN BACK WILL LOOK LIKE THIS:

```
SUM   DIFF
8000  4000
```

-I- INPUT A BYTE FROM A PORT. EITHER AN 8 BIT OR A 16 BIT PORT ADDRESS CAN BE SPECIFIED. THE FORM IS:

```
>I(ADDRESS)<CR>   THIS TIME THE ADDRESS CAN EITHER BE A FOUR HEX DIGIT
                  PORT NUMBER OR A TWO DIGIT HEX PORT NUMBER.
```

-J- NOT IMPLEMENTED IN THIS VERSION

-K- NOT IMPLEMENTED IN THIS VERSION

-L- LIST A BLOCK OF ASCII MEMORY. THIS COMMAND WILL DISPLAY A BLOCK OF MEMORY IN THEIR ASCII EQUIVALENTS. ANY NON ASCII CHARACTERS WILL BE DISPLAYED AS A '.' . THE FORM IS:

```
>L(SEGMENT),(ADDRESS 1),(ADDRESS 2)<CR>
```

-M- MOVE A BLOCK OF MEMORY TO ANOTHER BLOCK WITHIN THE SAME SEGMENT. THE FORM IS:

```
>M(SEGMENT),(ADDRESS 1),(ADDRESS 2),(ADDRESS 3)<CR>
```

-N- NON-DESTRUCTIVE MEMORY TEST. WILL TEST A BLOCK OF MEMORY OVER AND OVER UNTIL A CNTL-C IS GIVEN. ANY ERRORS WILL BE FLAGGED AS LOCATION, HEX BYTE AND BINARY BITS. THE FORM IS:

```
>N(SEGMENT),(ADDRESS 1),(ADDRESS 2)<CR>
```

-O- OUTPUT A BYTE TO A PORT. THIS COMMAND LETS YOU OUTPUT A GIVEN DATA BYTE TO A PORT. THE FORM IS:

```
>O(ADDRESS),(DATA)<CR>
```

THE (ADDRESS) CAN EITHER BE A FOUR OR A TWO HEX DIGIT NUMBER FOR 16 OR 8 BIT PORTS. TO OUTPUT TO A PORT THAT DECODES THE UPPER 8 ADDRESS LINES (A8-A15) YOU CAN USE THE FOUR HEX DIGIT FORM.

-P- NOT USED IN THIS VERSION

-Q- NOT USED IN THIS VERSION

-R- NOT USED IN THIS VERSION

-S- NOT USED IN THIS VERSION

-T- NOT USED IN THIS VERSION

-U- NOT USED IN THIS VERSION

-V- NOT USED IN THIS VERSION

-W- NOT USED IN THIS VERSION

-X- NOT USED IN THIS VERSION

-Y- NOT USED IN THIS VERSION

-Z- PUT SYSTEM ON HOLD, IN REMEMBRANCE OF THE ZAPPLE DAYS.
ONCE ENTERED, THE KEY BOARD BECOMES TRANSPARENT TO ALL BUT THE % KEY. THIS WILL BRING THE MONITOR BACK AND A MESSAGE IS PRINTED TO TELL YOU SO.

THIS IS THE END OF THE MONITOR COMMANDS AND FEATURES. HOPEFULLY MANY MORE COMMANDS WILL BE ADDED IN UPCOMING VERSIONS.

END

THIS IS THE USER CONFIGURED PORTION OF PC88MON.
IT IS A WORKING 'OUTLINE' AND SHOULD BE FOLLOWED
AS CLOSELY AS POSSIBLE.

```
;
;THIS IS THE 8 BIT PORTION OF PC88MON AND ALLOWS THE
;MONITOR TO BE UNIVERSAL TO ALL MACHINES SINCE IT IS USER WRITTEN
;FOLLOWING THIS OUTLINE EXACTLY.
;
;
PORT    EQU     00H     ;ADDRESS BASE OF PC8088 BOARD PORTS.  YOU WILL
;   HAVE TO CHANGE THIS TO MATCH THE PORT BASE OF THE PC8088 CPU BOARD.
TRPORT  EQU     PORT+0DH  ;PORT ADDRESS FOR TRANSFER
CLPORT  EQU     PORT+0CH  ;PORT TO CLEAR JUMP ON TRANSFER OPTION
START   EQU     0100H  ;LOCATION OF THIS INTERFACE CODE (USER DEFINED),
;   THIS IS TOTALLY ARBITRARY, BUT IF THIS PROGRAM IS TO RESIDE ON A
;   CP/M 80 FLOPPY THEN 100H IS MORE STANDARD.
;
;THE MONITOR NEEDS THE FIRST 40H BYTES STARTING AT 00000H (IN A ONE
; MEGABYTE ADDRESS SPACE) FOR IT'S OWN USE.  IF THIS PROGRAM IS TO BE
; PUT INTO A PROM ELSE WHERE, THEN YOU MAY WANT TO DEFINE THE STACK
; AREAS TO BE AT 00080H AND 00070H FOR THE 8 BIT AND 16 BIT STACK AREAS
; RESPECTIVILY.  IN THIS CASE YOU WILL HAVE TO HAVE THE FIRST 80H BYTES
; ALWAYS AVAILABLE TO THE MONITOR.  AS AN IMPORTANT NOTE THE STACK,
; WHERE EVER LOCATED, MUST RESIDE IN THE BASE (0000H) SEGMENT !
;       COMMAND AREA STARTING AT 0000H IN THE BASE SEGMENT
CMDBYT  EQU     0000H  ;ADDRESS OF COMMAND BYTE STORAGE
CMDATA  EQU     CMDBYT+1 ;ADDRESS OF COMMAND DATA STORAGE
STACKL  EQU     CMDATA+1  ;ADDRESS OF STACK LOCATION FOR PC88MON
PORTL   EQU     0010H  ;ADDRESS OF PORT BASE LOCATION
;
; THESE NEXT EQUATES ARE USER DEFINED TO FIT YOUR CONSOLE ROUTINES
; STATUS AND DATA PORTS AND BITS IF NEEDED.
TXMTY   EQU     80H  ; TRANSMITTER STATUS BITS FOR CONSOLE
RXMTY   EQU     01H  ; RECIEVER       "       "     "     "
DATA    EQU     01H  ; DATA PORT FOR CONSOLE DEVICE
STATUS  EQU     00H  ;PORT FOR STATUS BYTE
;
        ORG     START
;
        LXI     B,0
        LXI     D,0                 ;CLEAR REGISTERS
        LXI     SP,SSTACK           ;SET UP STACK POINTER FOR 8 BIT CPU
        LXI     H,STACK88           ;MOVE THE STACK ADDRESS FOR THE 16
        SHLD    STACKL              ;BIT CPU TO THE COMMAND AREA.
        MVI     A,00H               ;MAKE SURE MONITOR DOES NOT THINK
        STA     2900H               ;THAT CP/M-86 IS LOADED.
        MVI     A,PORT              ;TELL THE MONITOR WHAT THE PORTS ARE
        STA     PORTL               ;ON THE PC8088 CPU BOARD.
        OUT     CLPORT              ;MAKE SURE THAT FROM NOW ON THE
        ;8088 PICKS UP WHERE IT LEFT OFF ON SUBSEQUENT TRANSFERS.
;  IF THIS IS PUT INTO PROM, THEN YOU MAY NEED TO PUT SOME CONSOLE
;  INITIALIZATIONS HERE.  HERE ARE SOME EXAMPLES:
;       MVI     A,05                ;INITIALIZE CRT SERIAL TO 4800 BAUD
;       OUT     STATUS          ;SEND IT TO CONTROL REG. ON SERIAL BOARD
;
BEGIN   OUT     TRPORT              ;HERE IS WHERE WE GO TO THE 8088
        LDA     CMDBYT              ;NOW THAT WE ARE BACK TO THE 8 BIT
                       ;LETS SEE WHAT THE MONITOR WANTS US TO DO?
        CPI     50H                 ;DOES IT WANT US TO DO A CONSOLE STATUS
        JZ      CONST               ;
        CPI     80H                 ;OR MAYBE WAIT AND GET A CHARACTER
        JZ      CONIN               ;
        CPI     0F0H                ;OR JUST GET THE CHARACTER NOW
        JZ      KEYIN               ;
        CPI     10H                 ;OR OUTPUT A CHARACTER TO THE CONSOLE
        JZ      CONOUT              ;
        XRA     A                   ;IF NON OF THESE THAN WE DON'T KNOW
        STA     CMDBYT              ;WHAT IT WANTS SO RETURN A 0 TO SAY SO
        JMP     BEGIN               ;
;
CONST   CALL    CONT1               ;GO GET CONSOLE STATUS BYTE
MOVE    STA     CMDBYT              ;AND RETURN INFORMATION TO THE MONITOR
        JMP     BEGIN               ;AND TRANSFER BACK TO THE 8088
;
CONT1   IN      STATUS              ;THIS IS THE GENERAL STATUS ROUTINE
        ANI     RXMTY               ;AND SHOULD BE MODIFIED TO FIT YOUR
        RZ                          ;STATUS ROUTINE BUT FOLLOW THIS
        MVI     A,0FFH              ;SAME GENERAL FORMAT !
        RET                         ;
;
KEYIN
CONIN   CALL    CONT1               ;CALL GENERAL STATUS ROUTINE AND
        JZ      CONIN               ;KEEP LOOKING AT STATUS UNTIL A
                                    ;CHARACTER IS ENTERED.
        IN      DATA                ;NOW GO GET THE DATA
        JMP     MOVE                ;AND RETURN IT TO THE 8088
;
CONOUT  CALL    OUTST               ;WAIT UNTIL WE CAN OUTPUT A CHARACTER
        JZ      CONOUT              ;TO THE CONSOLE
        LDA     CMDATA              ;IF WE CAN, GO GET THE CHARACTER
        OUT     DATA                ;AND OUTPUT IT TO THE CONSOLE
        JMP     BEGIN               ;NOW RETURN TO MONITOR AND 8088
;
OUTST   IN      STATUS              ;THIS IS THE GENERAL CONSOLE OUTPUT
        ANI     TXMTY               ;STATUS ROUTINE
        RZ                          ;
        MVI     A,0FFH              ;
        RET                         ;
;
; THIS IS THE DATA AND STACK AREA ***********************************
;
;THIS SET UP PUTS THE STACKS IN LOW MEMORY, @0070H AND 0080H AND
; IS SUGESTED WAY SO THAT THIS PROGRAM CAN BE PUT ANY WHERE IN MEMORY,
; IN ANY SEGMENT OR IN RAM,ROM OR PROM.  JUST MAKE SURE THAT THE FIRST
; 80H BYTES AT 00000H ARE ALWAYS AVAILABLE FOR THE COMMAND AREA.
SSTACK  EQU     0080H
STACK88 EQU     0070H
;
        END
```

WARRANTY

SKI Electronics warrants to the original purchaser of its products that its assembled and tested products will be free from workmanship and material defects for a period of one (1) year.

The responsibility of SKI Electronics under this warranty to the original purchaser, without breach to this warranty, is limited to the correction or replacement by and at SKI Electronics option, at SKI Electronics' service facility, of any product or part which has been returned to SKI Electronics and in which there is a defect covered by this warranty. In the case of SKI Electronics assembled and tested products, SKI Electronics will correct any defect in materials and workmanship free of charge if the product is returned to SKI Electronics within one year to date of the original purchase from SKI Electronics. All such returned products shall be shipped prepaid and insured by original purchaser with a note containing problems encountered to:

SKI Electronics
3134 Woods Way
San Jose, CA 95148

SKI Electronics reserves the right to determine the existence and cause of a defect, with the sole right to decide whether the product should be repaired or replaced.

This warranty is void to any product or any part of a product which has been subject to:

(1) accident, negligence, misuse, or any unauthorized maintenance; or
(2) any alteration, modification, or repair by anyone other than SKI Electronics or its authorized representative.

THIS WARRANTY IS EXPRESSLY IN LIEU OF ALL OTHER WARRANTIES EXPRESSED OR IMPLIED OR STATUTORY INCLUDING THE WARRANTIES OF DESIGN, MERCHANTABILITY, OR FITNESS OR SUITABILITY FOR USE OR INTENDED PURPOSE AND OF ALL OTHER OBLIGATIONS OR LIABILITIES OF SKI ELECTRONICS.

SKI Electronics expressly disclaims any and all liability for products sold that are used in any application not specifically recommended, tested or certified to in writing from SKI Electronics. SKI ELECTRONICS DISCLAIMS ANY LIABILITY ARISING FROM THE USE AND/OR OPERATION OF ITS PRODUCTS, AND SHALL NOT BE LIABLE FOR ANY CONSEQUENTIAL OR INCIDENTAL OR COLLATERAL DAMAGES OR INJURY TO PERSONS OR PROPERTY.

This warranty applies to the original purchaser and only with a warranty card on file at SKI Electronics. SKI Electronics holds the right to disclaim any warranty without proof of the original purchase. SKI Electronics makes no warranty whatsoever with anyone other than the original purchaser.

Unless otherwise agreed, in writing, and except as may be necessary to comply with this warranty, SKI Electronics reserves the right to make changes in its products without any obligation to incorporate such changes in any product manufactured thereto-fore.

This warranty is limited to the terms stated herein. SKI Electronics disclaims all liability for incidental or consequential damages. Some states do not allow limitations on how long an implied warranty lasts and some do not allow the exclusion or limitation of incidental or consequential damages so the above limitations and exclusions may not apply to you. This warranty gives you specific legal rights, and you may also have other rights which vary from state to state.

1.0 THE SK8088 EXTENDED ADDRESS DAUGHTER BOARD IS DESIGNED TO GIVE THE MASTER 8 BIT CPU THE ABILITY TO CONTROL THE UPPER 8 ADDRESS LINES (A16 THROUGH A23), IF IT DOES NOT HAVE PROVISIONS TO DO SO ALREADY. THIS CONTROL IS NECESSARY FOR ANY SYSTEM THAT USES MORE THAN 64K OF MEMORY OR USES ANY MEMORY BOARDS THAT DECODE THE 8 HIGHEST ADDRESS LINES. IN THE SIMPLIST OPERATION OF THIS BOARD, UPON POWER UP OR A HARD RESET, THE ADDRESS REGISTERS ARE CLEARED, SETTING ADDRESS A16 THROUGH A23 TO ZERO. THIS ALLOWS THE 8 BIT TO FIX ITS 64K WORKING SEGMENT TO THE BASE OR ZERO PAGE. THE SK8088 PREBIOS SOFTWARE USES THIS FUNCTION.
ADDRESS A16 THROUGH A23 CAN BE CHANGED BY OUTPUTTING THE DESIRED ADDRESS TO THE EXTENDED ADDRESS PORT. ONE SHOULD BE CAREFUL IN DOING THIS FOR WHEN YOU CHANGE THESE ADDRESSES, YOU ARE CHANGING THE MEMORY LOCATION YOU WERE WORKING IN. IN OTHER WORDS THE CODE YOU WERE PREVI-OUSLY WORKING IN WILL NOT BE IN THE NEW MEMORY PAGE YOU JUST CHANGED TO. YOUR PROCESSOR WILL GO INTO OBLIVION UNLESS SOME STEPS WERE TAKEN TO PRE-SERVE THE PROCESSORS STATE. AS LONG AS THE SK8088 BOARD WITH THIS MODULE IS PLUGGED INTO YOUR SYSTEM, WHETHER THE SK8088 IS USED OR NOT, THE EXTENDED ADDRESS BOARD IS STILL OPERATIONAL.

2.0 CONIGURING THE I/O PORT:
AS AN EXAMPLE:

AFTER POWER UP, A16 THROUGH A23 ARE SET TO ZERO. BUT WE WISH TO CHANGE THEM TO 'E5H'. TO DO THIS WE SIMPLY MOVE 'E5H' INTO THE ACCUMULATOR AND OUTPUT IT TO THE EXTENDED ADDRESS PORT. THE UPPER ADDRESSES ARE NOW SET TO 'E5H' UNTIL ALTERED AGAIN OR A RESET IS IS DONE.

THE EXTENDED ADDRESS PORT ON THE BOARD CAN BE USER DEFINED TO ANY ONE PORT ADDRESS IN THE 256 BYTE PORT ADDRESS SPACE. ON THIS DAUGHTER BOARD, THERE ARE TWO JUMPER STRIPS LOCATED AT THE TOP OF THE BOARD. EACH COLUMN OF TWO JUMPERS REPRESENTS ONE BIT IN THE PORT ADDRESS FEILD, A0 THROUGH A7. BY LEAVING EACH COLUMN UNJUMPERED, IT SETS THAT PORT ADDRESS BIT HIGH (1). BY INSERTING A PEICE OF WIRE ACROSS THE COLUMN, YOU WILL SET THAT BIT TO A ZERO. USING THIS SCHEME FOR ALL EIGHT COLUMNS, YOU CAN SET THE EXTENDED ADDRESS TO ANY DESIRED PORT ADDRESS. (REFER TO FIGURE 1)

3.0 CONNECTION TO SK8088:
FIRST, FAMILIARIZE YOUR SELF WITH THE ORIENTATION OF THE DAUGHTER BOARD TO THE SK8088. THE THREE LEGS WILL POINT DOWN TOWARD THE GOLD FINGERS OF THE SK8088 BOARD AND WILL PLUG INTO THE SOCKETS OF U27,31 AND U33. IF YOU INTEND TO USE THE DAUGHTER BOARD TO CHANGE THE UPPER ADDRESS LINES BY OUT PUTTING THE ADDRESS DATA TO THE ON BOARD PORT, THEN THE PORT ADDRESS MUST BE SET NOW. REFER TO THE PREVIOUS DISCUSSION FOR DETAILS. IF YOU DO NOT USE THIS OPTION AND DO NOT JUMPER THE PORT ADDRESS, THE PORT DEFAULTS TO PORT FFH.
NOW, CAREFULLY PULL OUT THE I.C. CHIPS IN THE SOCKETS LOCATED AT U27, U31 AND U33. DO NOT PLUG THE DAUGHTER BOARD INTO THESE SOCKETS YET. NEXT, YOU MUST SOLDER THE FOUR WIRES FROM THE DAUGHTER BOARD TO THE SK8088. MATCH THE NUMBERS FROM THE DAUGHTER BOARD TO THE NUMBERED HOLES ON THE SK8088 AND SOLDER THE WIRE TO THAT HOLE. THE LONG WIRE TO THE TOP OF THE DAUGHTER BOARD IS NOT NUMBERED BUT CONNECTS TO THE HOLE MARKED NUMBER 75. THE DAUGHTER BOARD SHOULD NOW BE POSITIONED OVER U27,31 AND U33 WITH JUST A LITTLE PLAY FROM THE WIRES TO THE SK8088 BOARD.
CAREFULLY PLUG THE DAUGHTER BOARD INTO THE APPROPRIATE THREE SOCKETS PUSHING THEM IN FIRMLY. NOW PLUG THE THREE I.C. CHIPS YOU REMOVED

EARLIER INTO THE APPROPRIATE SOCKETS ON THE TOP OF THE LEGS OF THE DAUGHTER BOARD. U27 AND U33 ARE 74LS244'S AND U31 IS A 74LS245. REFER TO THE ASSEMBLY DRAWING FOR GRAPHIC DETAIL.

4.0 THEORY OF OPERATION:

THE ACTUAL OPERATION OF THE DAUGHTER BOARD IS RELATIVELY SIMPLE. U3 AND U4 ALONG WITH THE SOUT SIGNAL COMBINE TO FORM AN 8 BIT PORT ADDRESS COMPARATOR. WHEN AN OUTPUT IS DONE TO THE PORT SELECTED ON THE DAUGHTER BOARD, PIN 6 OF U3 GOES HIGH. ANDED WITH THIS SIGNAL IS THE GENERALIZED WRITE PULSE PWR* WHICH PROVIDES U1 AND U2 WITH A SELECT CLOCK PULSE. U1 AND U2 COMBINE TO FORM AN EIGHT BIT LATCH. THE SELECT PULSE FROM U6 PIN 6 CLOCKS THE DATA FROM THE DATA BUS INTO THE LATCHES. THIS LATCHED DATA THEN APPEARS AT THE INPUT TO U5, AN OCTAL TRI-STATE BUFFER. WHEN THE 8 CPU IS IN CONTROL OF THE BUS, WHICH MEANS THE ADDRESS DISABLE LINE IS HIGH, THE TRI-STATE BUFFER (U5) IS ACTIVE ALLOWING THE DATA LATCHED INTO U1 AND U2 TO APPEAR ON ADDRESS LINES A16 THROUGH A23. WHENEVER ANOTHER BOARD ON THE BUS WISHES TO TAKE CONTROL, SUCH AS THE SK8088 OR A DMA DEVICE, THEY WILL ASERT THE ADDRESS DISABLE LINE (ADSBL*) AND THE TRI-STATE BUFFER (U5) WILL FLOAT ITS OUTPUTS AND FREE THE ADDRESS LINES A16 THROUGH A23 FOR THE REQUESTING DEVICE TO USE.

FIG. 1 EXAMPLE: PORT SET TO 9EH. PLACING A WIRE BETWEEN JUMPERS SETS THAT ADDRESS TO DECODE A ZERO, NO WIRE DECODES A ONE.

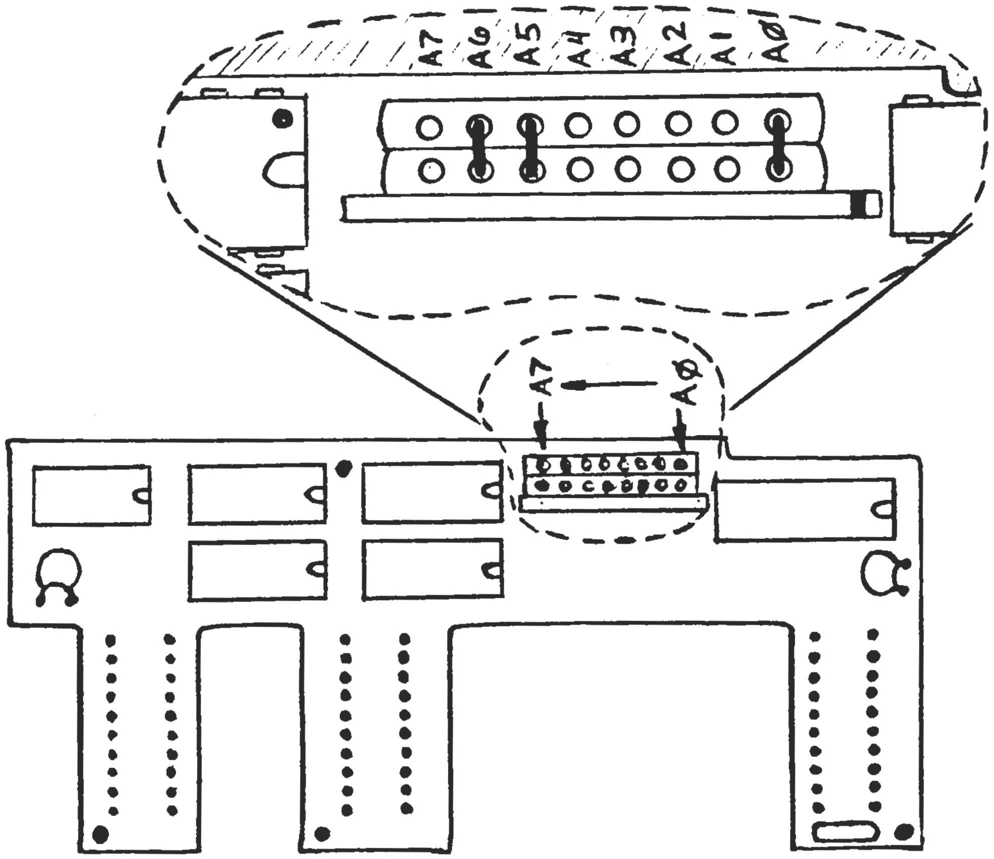
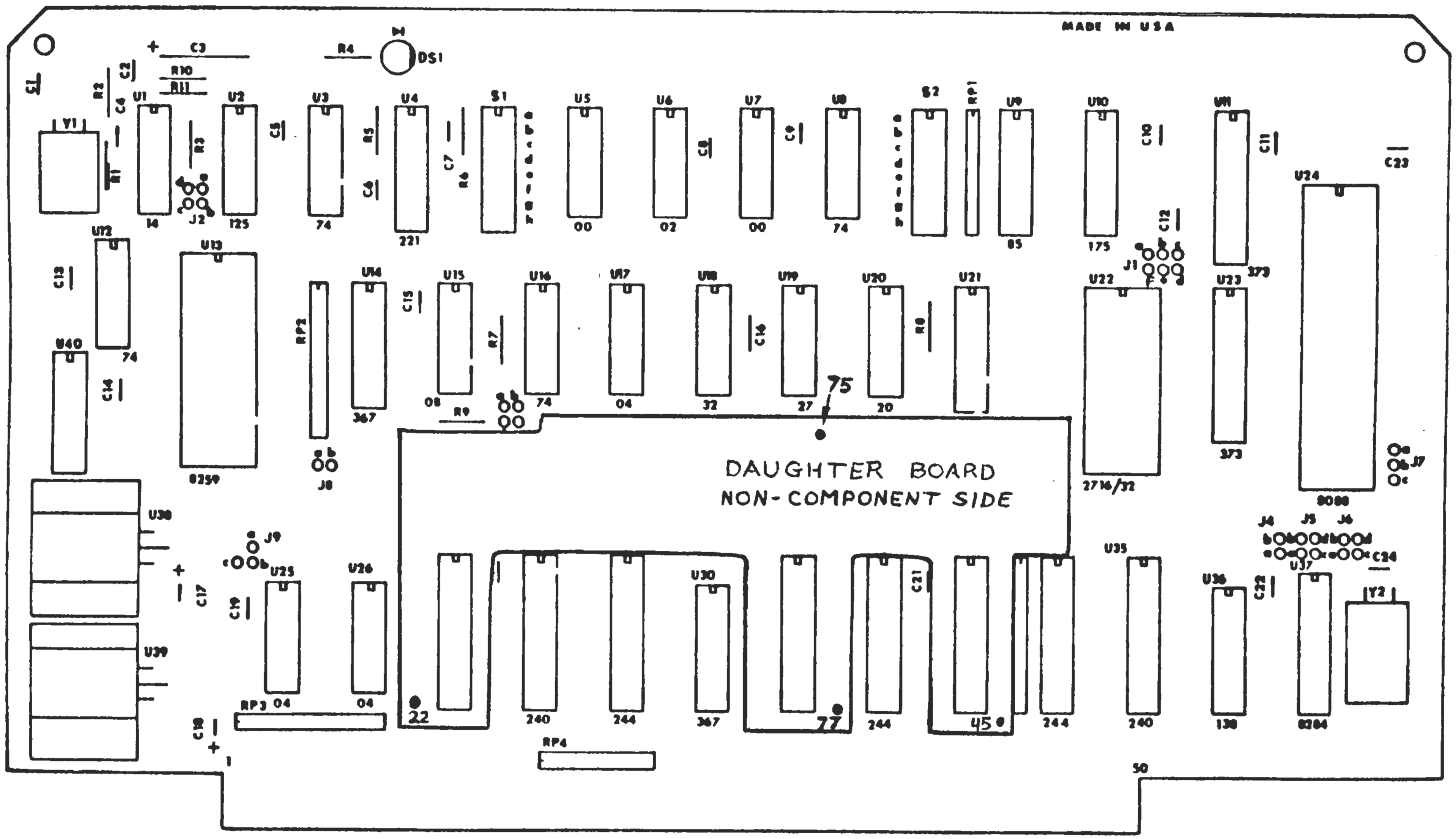
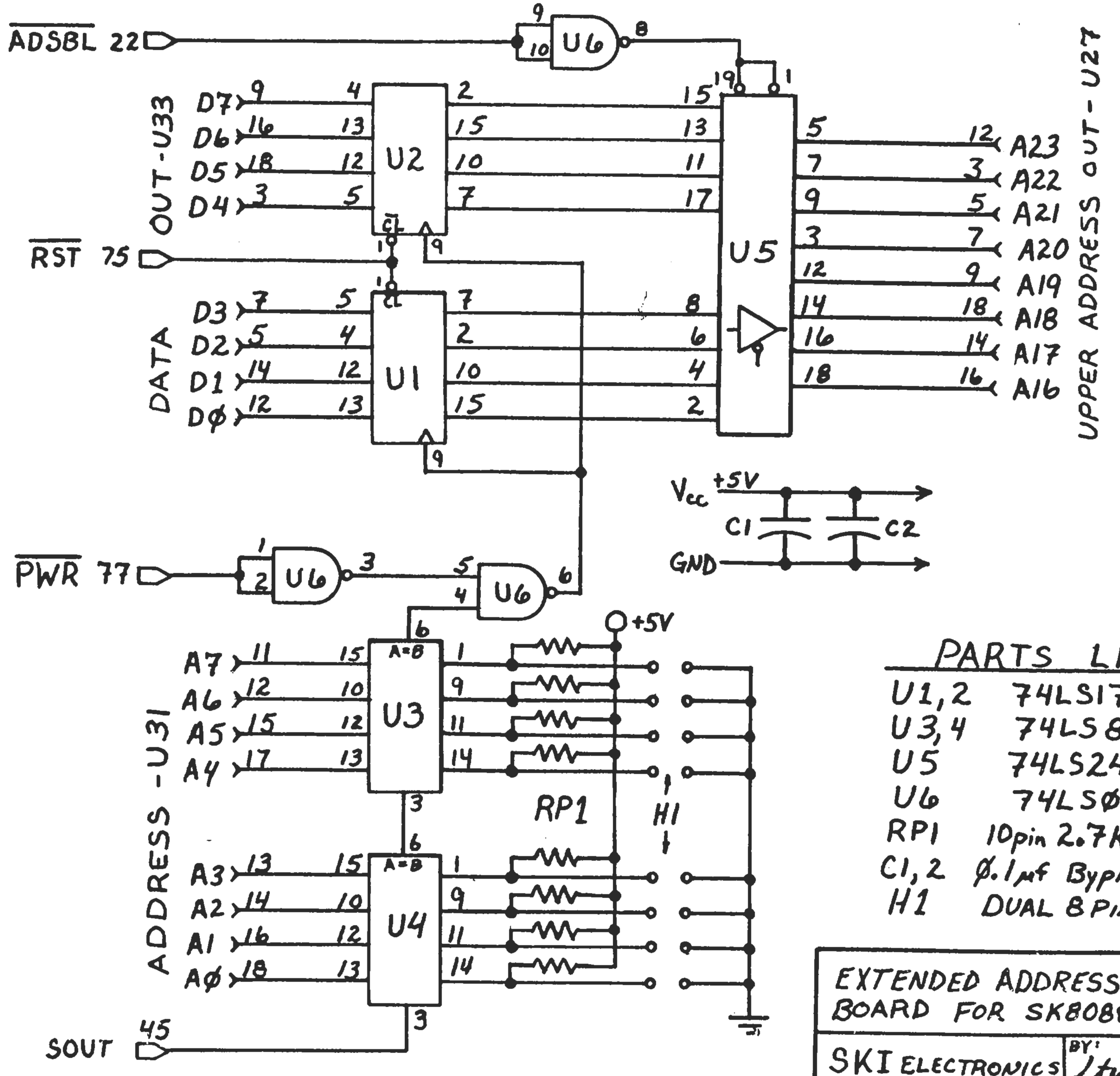


FIGURE 1



ASSEMBLY DRAWING



PARTS LIST

U1,2	74LS175	Vcc=16, G=8
U3,4	74LS85	Vcc=16, G=8
U5	74LS244	Vcc=20, G=10
U6	74LSφφ	Vcc=14, G=7
RP1	10pin 2.7K SIP RESISTOR	
C1,2	φ.1µf BYPASS CAPACITORS	
H1	DUAL 8 PIN JUMPER STRIPS	

EXTENDED ADDRESS DAUGHTER BOARD FOR SK8088 CPU

SKI ELECTRONICS BY: [illegible] 8/26/82

© 1982