Wunderbuss Input/Output Controller

Technical Manual

MORROW DESIGNS

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

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

Appendices

LIST OF TABLES

1. INTRODUCTION

The Wunderbuss Input/Output Controller (WB I/O) is the heart of a general purpose S-100 system that combines all the features necessary for an efficient interrupt-driven, multi-user system. The WB I/O is built on a motherboard capable of holding up to 14 additional S-100 utility cards. Its features include:

1. A patented active termination system that reduces noise inherent to connection of S-100 signal lines.

2. An 8259-A Programmable Interrupt Controller (PIC) device designed to monitor up to eight peripheral devices and set priorities for their service.

3. Three 40-pin programmable Asynchronous Communication Elements (8250 ACE serial interfacing devices) capable of generating CPU interrupts in response to RS 232 signals and communication events.

4. A 50-pin connector for a daisy-wheel printer.

5. A bi-directional, undedicated, multi-purpose parallel port.

6. A CMOS crystal-controlled, multi-functional calendar/clock chip.

The serial, parallel, clock and PIC devices on the WB I/O are all I/O mapped. They are accessed through switch selectable I/O port addresses. These devices may be programmed to generate interrupts to the CPU via the PIC based on a rich selection of status conditions.

The design and versatility of the WB I/O ensures the user of a long useful life, even in a system subject to frequent upgrading. Like all Morrow Design products, it should give the user years of satisfaction.

2. WB I/O ACTIVE TERMINATION

The WB I/O features a 14 slot IEEE 696 standard S-100 motherboard with a patented active termination system referred to as Noise Guard. The structure and pinout of the S-100 bus normally lends itself to crosstalk and signal noise in an inproperly or unterminated bus. But the WB I/O eliminates this problem by the use of active termination. All IEEE 696 signal lines are actively pulled up through 180 Ohm resistors. One exception to this is PRESET, line 75. This line is pulled high on the MPZ80 CPU card. The table below depicts the power connections for the S-100 pins.

Table 2-1: S-100 Power Connections

Pins	Connection
1,51	+ 8 V unregulated
2	+ 16 V unregulated
52	- 16 V unregulated
20	ground
50	ground
70	ground
100	ground

The design of the WBI /O motherboard allows the bus to meet or exceed all the specifications for the S-100 bus at 6 Mhz making the board the heart of a powerful, reliable and expandable system. For a complete description of the S-100 bus, refer to the reference on specifications for the S-100 bus interface devices.

3. I/O ADDRESSING

All devices on the WB I/O are associated with some S-100 I/O port. In all, almost 30 distinct I/O registers are used to control the many device functions available on the board. Yet the WB I/O takes up only eight I/O port addresses. To understand how so many registers can be accessed through so few ports, it is useful to think of the port addressing scheme of the WB I/O as 'bank-select I/O'. As the name suggests, this is analogous to conventional bank-select memory schemes. Specifically, banks of registers are allowed to share the same block of consecutive I/O addresses while a dedicated I/O port is used to enable one bank, and at the same time, disable all other similarly addressed banks.

The WB I/O is divided into four I/O banks, (hereafter called groups) with each group occupying the same eight I/O port addresses - BASE to BASE+7. Port address BASE+7 is the GROUP SELECT port, and establishes which of the four I/O groups will be active at any given time. By outputting some number between Ø and 3 to the GROUP SELECT port the user enables operations directed to ports between BASE and BASE+6. To enable a different group, the user must output a different group number to GROUP SELECT port BASE+7. While this port selection technique is extremely efficient in conserving I/O space, it does impose the responsibility of keeping track of which I/O group is currently active.

3.1. I/O Port Addressing

DIP switch 7C is used to determine the BASE port address of the I/O groups on the WB I/O. Paddles 2 through 6 of switch 7C are compared with S-1ØØ address lines A3 through A7 allowing BASE to be located at any eight byte I/O boundary. The relationship between the the paddles and the address lines are as follows:

Table 3-1: DIP Switch 7C

Paddle Number	Address Line
2	A7
3	A6
4	A5
5	A4
6	A3

Setting a paddle to the ON position causes a match to occur when its associated address line is a low logic level. If all five switches are ON, the BASE address is at port Ø. The standard address in all Morrow Design systems is port 48 hex.

3.2. GROUP SELECT Port BASE+7

Once the base address has been established by setting DIP switch 7C, the addresses of all I/O functions on the WB I/O are determined (see I/O MAP on the following page). In order to gain access to a specific device function, however, the group number of that device function must first be output to I/O port BASE+7. The I/O group is selected by executing an output instruction to port BASE+7 with data bits Ø and 1 set as follows:

Table 3-2: Output to GROUP SELECT Port BASE+7

Data Bit-1	Data Bit-Ø	Group Number
Ø	Ø	Ø
Ø	1	1
1	Ø	2
1	1	3

Use of the group select port is best described by example. Suppose you want the I/O space taken by the WB I/O to extend from 48 hex to 4F hex and you want to access serial port and daisy-wheel printer port Ø. First set the I/O base by turning 7C, paddles 3 and 6 ON and paddles 2, 4 and 5 OFF. With this base address selected, the GROUP SELECT port is at BASE+7, or port 4F hex. In order to read serial device number two, the user first outputs a 2 to the GROUP SELECT port. Further outputting or inputting to ports 48 hex through 4F hex controls the registers for the number two ACE serial device. To access the parallel daisy-wheel printer port, the user would then output a Ø to the GROUP SELECT port. It is important to remember that the functions of ports at BASE to BASE+6 change from device to device depending upon the last value sent to the group select port. The following chart depicts the configuration of the GROUP SELECT port.

Table 3-3: I/O Map-out BASE+7

D 1	D Ø	Gp.#	Device
xØ	xØ	xØ	xDAISY ports, 199Ø clock, PIC, aux. par. port
Ø	1	1	Serial port 1 (IC 6D, cable connector P1)
1	Ø	2	Serial port 2 (IC 5D, cable connector P2)
1	1	3	Serial port 3 (IC 4D, cable connector P3)

The GROUP control register is I/O port BASE+7. To select an I/O group, output to port BASE+7 with data bits Ø and 1 set as indicated above. Once a group is selected, ports are assigned as follows:

Table 3-4: GROUP Assignments

GROUP Ø

I/O Address	Input	Output
BASE	DAISY Ø IN (STATUS)	DAISY Ø OUT
BASE+1	Switch/Parallel port flags	DAISY 1 OUT
BASE+2	R.T. Clock IN/RESET CLK. Int.	R.T. Clock OUT
BASE+3	Parallel data IN	Par. data OUT
BASE+4	8259 Ø register	8259 Ø register
BASE+5	8259 1 register	8259 1 register
BASE+6	not used	Par. port cntrl.

GROUPS 1, 2 & 3 - 825Ø ACE Serial I/O Ports

	Input	Output
BASE	Receive buffer	Transmit buffer/LSB baud
BASE+1	Interrupt Enable	Interrupt Enable/MSB baud
BASE+2	Interrupt Identify	not used
BASE+3	Line Control register	Line Control register
BASE+4	Modem Control register	Modem Control register
BASE+5	Line status register	not used
BASE+6	Modem status register	not used

Note that an output to BASE+7 always assigns an I/O group but has no function within any given I/O group.

4. THE INTERRUPT SYSTEM

Microcomputer systems in general are required to communicate with peripheral devices such as printers, CRT terminals and various types of parallel devices. There are classically two ways of approaching the way a CPU may service these devices - polled and interrupt.

In a polled mode, every device in the system is periodically querried about its service requirements. When a device requires servicing (for example, a person has just typed a character on a CRT terminal), the CPU stops polling all other devices until it has finished servicing the user's request. From a system viewpoint the CPU should handle these requests as quickly as possible. The total system throughput is a function of the number of devices on the system, the length of time to poll each device and service each device request. The operating system is never idle; it is always polling the devices searching for activity.

There is a direct analogy here to hardware design: This type of operation is said to be synchronous, meaning the CPU may branch to a service request subroutine only after it has determined from the device, through polling, that it is necessary to do so. There are certain problems with this approach, though. These lie in the amount of time needed to service each request. Another disadvantage lies in the lack of priority-setting for the peripheral devices. In a polled system, each device has equal status, which is unfortunate because in a real environment some devices require faster, more frequent service response than others. Polling high priority devices more frequently is one solution, but this burdens the system I/O subroutines with complex algorithms. Another disadvantage is that the processor is always occupied with the polling process and not able to perform other tasks.

An interrupt-driven system is much different in its implementation. Although requiring more hardware and more complex software, the system has none of the problems associated with a polled system. With correct hardware, the devices are all prioritized according to their service requirements and the CPU is free to handle other tasks until a device requires service. The I/O devices themselves in this system interrupt whatever the CPU is presently doing only when they require something from the host processor. This type of system is more analogous to an asynchronous hardware design - one where events can occur at random intervals not related to the CPU's operations. Its randomness corresponds nicely with the relative randomness of device requirements tied into the system and allows maximum system response to these peripherals.

4.1. The Programmable Interrupt Controller (PIC)

This section describes the use of the PIC in the WB I/O, but before going any further, one assumption must be made: If using a Z8Ø CPU chip, an Enable Interrupt (EI) instruction must be executed and the Z8Ø set to Interrupt Mode Ø (8Ø8Ø mode). The PIC instructions and modes are described in further detail in the following pages.

The additional hardware design requirements in an interrupt system have been kept to a minimum in this system by using an 8259-A programmable interrupt controller integrated circuit chip. By using this chip in conjunction with standard integrated circuits a powerful interrupt driven system has been implemented. This section describes the software requirements necessary to utilize the PIC to its fullest.

The PIC can directly monitor the requirements of eight separate devices and prioritize them according to system requirements. The system has three serial channels (the hardware uses three Universal Asynchronous Receive Transmit integrated circuits called UARTs) which are normally connected to CRT terminals or a serial printer. These three devices are tied directly to the

PIC to provide a signal when they require servicing. The WB I/O also has a DAISY port which can generate a signal for the printer whenever it requires servicing. Besides the UARTs and the DAISY port, the on-board real-time clock may be programmed to generate interrupts at precise, software-selected intervals. Multi-user systems in general require a real-time clock to insure proper allocation of the CPU's time among various tasks.

So far we have described five of eight possible events the PIC may monitor. Besides these, the system provides the user with the option to monitor three of the S-100 vectored interrupt lines. These lines are jumper options on the WB I/O which allow the the on-board PIC to monitor and prioritize interrupts generated by boards plugged into the S-100 bus such as disk controllers or MultI/O boards.

4.2. PIC Interrupt Vectors

To signal the host CPU that one of the monitored devices requests service, the PIC must issue a signal called PINT (processor interrupt, line-73 of the S-100 bus) to the host CPU. The host CPU completes its current instruction and issues a signal called SINTA (interrupt acknowledged, line-96 of the S-100 bus) indicating it has recognized the requested interrupt and is willing to receive its next instruction from the interrupting device, in this case, the PIC.

At this point, a device may generate any instruction it wishes and the host CPU will execute it. Two logical instructions might be asked of the CPU in such a case - a Restart or a Call. These are logical choices because both of them predictably alter the current flow of instruction by changing the value of the Program Counter to a given address, then saves the location where the CPU is to return afterwards by pushing the current Program Counter onto the stack. The Restart instruction is limited to eight locations where the program may branch, making this instruction dependent on hardware and software environments and leaves us with the Call instruction.

The PIC has been designed to generate a Call instruction upon receiving the SINTA response from the host CPU. The CPU then fetches a 16-bit address of the location of the interrupt vector. Hardware on the WB I/O counts the next two CPU fetches (the address vector) and enables the PIC to output this address to the data-in bus. When programmed, the PIC has eight vector addresses associated with it that correspond to the eight interrupt devices it monitors. The vector contains a jump instruction to the address of the routine responsible for handling it.

The PIC generates interrupt vectors at either eight-byte or four-byte intervals in the 16-bit address space, limited by both the PIC and the CPU to a 64K address space. For compactness, most routines use the four-byte separation since a jump instruction is only three bytes long and few interrupt service routines fit in less than an eight byte address space. The eight-byte interval is provided for compatibility with the use of the 8080 and Z80 restart instructions which are spaced eight bytes apart. The following is a map of the hardware devices associated with the PIC input line.

Table 4-1: Map of the Hardware Devices Associated With PIC Input Lines

	IRQ Line	Device
Highest	0	S-100 vectored interrupt 0
	1	S-100 vectored interrupt 1
	2	S-100 vectored interrupt 2
	3	Serial Device #1
	4	Serial Device #2
	5	Serial Device #3
	6	DAISY print wheel ready
Lowest	7	Real-time clock TP line

4.3. PIC Modes

The PIC, being a software programmable device, can be set up in many different modes allowing itself to be tailored to any operating environment. The Decision 1 environment takes advantage of some of these features and the user is free to explore others. This section explores some of the more common PIC modes. For a rigorous description of the different modes please refer to the Intel Data Sheet and Application Note.

4.3.1. Triggered Modes

The PIC may be programmed to monitor the eight devices in either edge-triggered or level-triggered mode. In the edge-triggered mode, the PIC generates an interrupt when it senses a change on one of its eight input lines (IRQ0 - IRQ7). This is suitable for events that do not latch their interrupt requests to the PIC. However, this does cause a problem when the UARTs generate one edge only for one or more interrupts. The result is a possible loss of some interrupt requests. For this reason, all Morrow Designs software use only the level-triggered mode.

4.3.2. Master/Slave Mode

The PIC may be programmed to be either a single system PIC or part of a larger interrupt system involving up to four PICs. This would be the case in a system where more I/O is required and one or more Morrow Designs I/O controller boards has been installed. In a multiple configuration, one PIC is designated as the Master and is the only device which may control the PINT line on the S-100 bus. All other PICs drive the selected S-100 vectored interrupt lines monitored by the Master PIC. However, cascading of multiple PICs is not supported in the WB I/O hardware implementation.

4.3.3. Buffered Mode

The buffered mode option for the PIC is not implemented on the WB I/O board.

4.3.4. End of Interrupt (EOI) Mode

An in-service bit (IS) on the PIC indicates a pending interrupt. This may be reset manually by the interrupt service routine of the CPU, or automatically after the third byte of the Call instruction has been sent by the PIC. An automatic End of Interrupt (AEOI) instruction is programmable at the time of initialization only, so once set, the PIC must be re-initialized to change this mode. In AEOI mode, the full nesting capabilities of the PIC are lost. For this reason, and for maximum system flexibility, all Morrow Designs software has been written with the AEOI feature disabled.

4.3.5. Polled Mode

The PIC may be configured to resemble a polled I/O system by setting the Poll bit to a logic '1'. In this mode, the PIC does not generate an interrupt with a change in state on any of its IRQØ - IRQ7 lines. The CPU issues a Poll command to the PIC, the PIC then gates a byte onto the data-in lines to the CPU indicating the highest priority interrupt pending. The lower three bits of the byte are used to indicate which device requires service. The highest bit, if set, indicates a device is requesting service.

4.3.6. Nested Mode

The nested mode of the PIC allows service requests from I/O devices to be prioritized. When a device is in need of service, the PIC issues an interrupt to the host CPU only if there are no higher priority devices requesting service via the PIC. If a lower priority device requires service, it must wait until all higher priority devices are serviced and the interrupt-handling subroutine has issued an EOI command to that PIC. If a device of higher priority requires service, the lower priority device's service subroutine is interrupted until the higher priority device has been serviced. Although this requires intricate software routines to keep track of the signals, this mode allows maximum system response to devices which require immediate service. All Morrow Designs software take advantage of the PIC nesting.

4.3.7. Rotating Priority - Mode A

In the nested mode, devices are prioritized and the device with highest priority obtains service. The priorities are assigned according to which input line (IRQØ - IRQ7) a device is connected. This scheme works well for devices not inherently equal. In some instances all eight devices connected to the PIC have the same priority. The PIC may be programmed to rotate the priority through all devices. In this mode, each device gets rotated to the lowest priority after it has been serviced; all other devices are raised one level in the priority ladder. At present, Morrow designs software does not implement the rotating priority option.

4.3.8. Rotating Priority - Mode B

This mode is very similar to Mode A, the difference being rotation in Mode B can be controlled with software as opposed to a fixed rotation controlled by hardware internal to the PIC, as in Mode A. The software is only allowed, however, to set that device with the lowest priority. All other devices are ordered by priority via the PIC. The next lowest priority device is then shifted into the highest priority spot. For instance, if IRQ2 was set as the lowest priority, the PIC automatically sets IRQ3 as the highest.

4.4. PIC Status Registers

The PIC status registers may be read to determine the current state of the PIC. These registers place IRQØ - IRQ7 status on data-in bits, Ø - 7 respectively. IRQØ is assumed to be the highest priority and IRQ7 the lowest.

4.4.1. Interrupt Mask Register (IMR)

The PIC has the capability of masking any one of eight interrupt inputs - i.e. not allowing that particular device to generate an interrupt. The mask register contains eight bits, any of which, when high, shut off the appropriate IRQ input to the PIC. If all the bits are set high, no interrupts are generated. If all are set low, all devices are recognized in their normal prioritized sequence. This allows the software complete control over each individual device's service requests. The register can be written and read by the system software.

4.4.2. In-Service Register (ISR)

The in-service register allows the software to query the PIC for those devices currently being serviced. Each of the eight lines are associated with eight bits. A high level indicates that device being serviced. Bits in this register are reset by the software issuing an EOI (either specific or non-specific) at the end of the associated interrupt service routine.

4.4.3. Interrupt Request Register (IRR)

This eight-bit register is read to determine which of the eight devices is requesting service. The highest pending priority is reset whenever an interrupt from the PIC has been acknowledged by the CPU. (This register is not affected by the IMR - a device may request an interrupt and be masked out.)

5. PROGRAMMING THE PIC

The PIC is a programmable device and must be initialized for correct operation.

NOTE: If the PIC is not initialized, it is still possible for it to generate spurious interrupt requests to the host CPU. Programs such as DDT - the Dynamic Debugging Tool by Digital Research - only aggravate this problem by issuing Enable Interrupt instructions whenever the 'GO' command is invoked. This caution should be followed in systems where interrupts are not implemented as well.

The PIC is accessed through system ports BASE+4 and BASE+5. Since context plays an important role in determining what each of these ports control, remember this rule: outputting to BASE+4 sets PIC address bit-AØ to a 'Ø' or low logic level; outputting to BASE+5 sets PIC address line AØ to a '1' or high logic level. There are two types of registers internal to the PIC. Registers referred to as ICW are initialization registers and are typically accessed only when the PIC has been first powered up. Registers referred to as OCW are operation control registers and are read from and written to during regular PIC operation (subsequent to initialization).

5.1. Initialization Registers

The PIC is ready to accept commands for initialization on power-up. There are a minimum of two registers in the PIC which must be initialized for the PIC to begin servicing interrupt requests. Depending on the mode the user operates in, as many as four registers must be initialized prior to operation. These registers are detailed below.

5.1.1. Initialization Control Word 1 (ICW1)

The first word written to initialize the PIC is ICW1. It is specified by outputting to port BASE+4 a value with data bit-4 set logic high. This informs the PIC that the initialization sequence is beginning. In addition to bit-4 being set, the other bits are assigned the following function:

Table 5-1: ICW1 Bit Assignments

Bit	Function
7	Part of the high byte of the beginning address of the interrupt vectors; bit-A7 of the call address.
6	Part of the low byte of the beginning address of the interrupt vectors; bit-A6 of the call address.
5	Part of the low byte of the beginning address of the interrupt vectors; bit-A5 of the call address.
4	Set high to begin initialization sequence.
3	LTIM - set to 1 for level-triggered mode (normally high for all Morrow Designs software).
2	ADI - Call address interval. Low for call address at eight-byte intervals, high for four-byte intervals (normally high for all Morrow Designs software).
1	SNGL - Single or multiple PICs in the system to be used in cascade mode. Since WB I/O does not support cascading, this bit set to a 1.
Ø	ICW4 - This bit set high allows access to the Initialization Control Word 4 for selection of operation modes. If this bit is set low, the PIC initialized as master, non-buffered mode, no AEOI and in the normal nested mode (normally low for all Morrow Designs software; set this bit low when initializing).

5.1.2. Initialization Control Word 2 (ICW2)

Initialization Control Word 2 is available at BASE+5 after ICW1 has been selected and initialized. The ICW2 register contains the high byte of the call address vector starting address. The bits are configured as follows:

Table 5-2: ICW2 Bit Assignment

Bit	Function
7	Part of the high byte of the beginning address of the interrupt vectors. This is bit-A15 of the call address.
6	Part of the high byte of the beginning address of the interrupt vectors. This is bit-A14 of the call address.
5	Part of the high byte of the beginning address of the interrupt vectors. This is bit-A13 of the call address.
4	Part of the high byte of the beginning address of the interrupt vectors. This is bit-A12 of the call address.
3	Part of the high byte of the beginning address of the interrupt vectors. This is bit-A11 of the call address.
2	Part of the high byte of the beginning address of the interrupt vectors. This is bit-A1Ø bit of the call address.
1	Part of the high byte of the beginning address of the interrupt vectors. This is bit-A9 of the call address.
Ø	Part of the high byte of the beginning address of the interrupt vectors. This is bit-A8 of the call address.

5.1.3. Initialization Control Word 3 (ICW3)

Morrow Designs implementation does not require the initialization of ICW3. If the cascade feature is absolutely required within a system configuration, a Morrow Designs Mult I/O board should be installed to become the master PIC for the system. The user is free to explore this option and is referred to the Mult I/O manual for details on both that board and on cascading PICs.

5.1.4. Initialization Control Word 4 (ICW4)

This register is available at BASE+5 if the ICW4 access bit of register ICW1 (bit-Ø) was not set when beginning the PIC initialization routine. Normally, this register need not be accessed as all bits are automatically cleared to the mode that Morrow Design's software uses. If the user wishes to change to AEOI, buffered, slave or fully nested mode, he is free to program this register appropriately.

5.2. Operation Control Registers

One the PIC is initialized, it is ready to function as the system interrupt controller. Further changes in the PIC operating parameters are accomplished by programming a set of registers referred to as the Operation Control Registers.

5.2.1. Operation Control Word 1 (OCW1)

This register contains a software mask that allows the operating system to mask out any of the eight interrupt inputs and is available any time after initialization sequence through port BASE+5. Setting any of the bits high forces the PIC to ignore the interrupt request line associated with that bit. The bits are arranged with data bit-7 corresponding to IRQ7 and data bit-Ø corresponding with IRQØ. As indicated, a bit set high masks the interrupt request; a bit set low unmasks it. The PIC clears this register to Ø (all enabled) on power up.

5.2.2. Operation Control Word 2 (OCW2)

Operation Control Word 2 (OCW2) is selected by outputting to BASE+4 with bits 3 and 4 reset (logic Ø) any time after the initialization sequence. On power up, these bits are all reset (logic Ø). This registers allows control over the following functions:

Table 5-3: OCW2 Bit Assignments

Bit	Function
4	Must be low to access OCW2.
3	Must be low to access OCW2.
2	L2 - Specific end of interrupt bit-2 (MSB)
1	L1 - Specific end of interrupt bit-1
Ø	LØ - Specific end of interrupt bit-Ø (LSB)

Bits 5, 6 and 7 are multiplexed and have the following functions:

Function	Bit-5	Bit-6	Bit-7
Clears rotate priority - Mode A flip-flop	Ø	Ø	Ø
End of Interrupt	1	Ø	Ø
Specific Interrupt	1	1	Ø
Sets rotate priority - Mode A flip-flop	Ø	Ø	1
EOI causes rotate - priority Mode A	1	Ø	1
Sets rotate priority Mode B	Ø	1	1
EOI causes rotate priority Mode B	1	1	1

5.2.3. Operation Control Word 3 (OCW3)

Operation Control Word 3 (OCW3) is selected by outputting to BASE+4 with data bit-3 set and bit-4 reset (logic Ø) any time after the initialization sequence. On power up, these bits are all reset (logic Ø). Morrow Designs software does not use this register and leaves all bits reset. This register allows control over the following functions:

Table 5-4: OCW3 Bit Assignments

Bit	Function
7	Not used
6	ESMM - Enable Special Mask Mode when high.
5	SMM - Special Mask Mode when high.
4	Must be Ø to access OCW3
3	Must be 1 to access OCW3
2	Enter poll mode when high, interrupt mode when low. A high on this line allows the next read BASE+5 to read the BCD code of the highest interrupt request pending (in non-interrupt environments).
1	SRIS - allows access to the Interrupt Request register (IRR) and the In-service register (ISR).
Ø	RIS - when low, allows access to the IRR by reading port BASE+5. When high, allows access to the ISR by reading port BASE+5.

5.3. Interrupt Status Registers

During normal PIC operation it may be desirable to examine the status and operating parameters of the device. There are three readable registers on the PIC which contain status information. They are accessed by inputting from the appropriate port and are defined as follows:

5.3.1. Interrupt Mask Register (IMR)

The interrupt mask register may be read at any time by inputting from WB I/O port BASE+5. This eight-bit port contains a map of the IRQ lines which have been previously masked by outputting to BASE+5, the OCW1. If no IRQ lines are masked, all bits are low (logic Ø) which is the normal condition on power-up. Any IRQ line that is masked has its appropriate bit set. IRQ7 is data bit-7 and IRQØ is data bit-Ø.

> NOTE: The following two status registers are selected by setting the appropriate bits with OCW3. The registers is then available through BASE+4. The state of OCW3 bits Ø and 1, (SRIS and RIS) once set, will allow continuous access to the selected register until the bits are changed (bits are internally latched by the PIC).

5.3.2. Interrupt Request Register (IRR)

The IRR is an eight-bit register which, when read by inputting from WB I/O BASE+4, tells which of the IRQ lines are currently asserted at a high logic level and are awaiting acknowledgement. By reading this register, it is possible to determine which interrupt requests have been recognized and which have yet to be acknowledged. Bit-7 maps to IRQ7 and bit-Ø maps to IRQØ. After initialization, this register may be read from BASE+5 as long as OCW3 is not changed (i.e. OCW3 bits ERIS = 1 and RIS = Ø). The register is updated each time an interrupt request is acknowledged by the CPU.

5.3.3. In-service Register (ISR)

The in-service register (ISR) is an eight-bit register containing information on which priority levels are currently being serviced. By reading this register (inputting BASE+4 with the OCW3 bits ERIS = 1 and RIS = 1), the user determines the number of the IRQ lines being serviced. IRQ7 maps to data bit-7 and IRQØ maps to data bit-Ø. A logic high level on any bit indicates that the associated IRQ line is in service. The register is updated each time an EOI is issued.

Table 5-5: Typical Initialization Sequence

```
***************************************************************

        This  routine will initialize the PIC as a  single,
        master PIC,  non-buffered mode, level-triggered, no
        automatic End of Interrupt (AEOI disabled), regular
        nested  mode with the call vectors at 4 byte inter-
        vals. Although ICW4 and OCW1 are cleared to zero on
        power-up,the routine initializes them for complete-
        ness.

***************************************************************

base    equ     Ø48h            ;base port address
grpsel  equ     base + 7        ;group select port
groupØ  equ     Ø
init    equ     Ø1Øh            ;bit high to initialize the PIC
icw1    equ     base + 4        ;initialization control word 1
icw2    equ     base + 5        ;initialization control word 2
icw3    equ     base + 5        ;initialization control word 3
icw4    equ     base + 5        ;initialization control word 4
ltim    equ     Ø8              ;Level-triggered mode
adi     equ     Ø4              ;Call address interval = 4 bytes
sngl    equ     Ø2              ;one PIC in the system
IC4     equ     Ø1              ;ICW4 access bit
lovect  equ     ØEØh            ;low byte of interrupt vector address
hivect  equ     Øffh            ;high byte of interrupt vector address
normal  equ     Ø               ;master/reg. nest/non-buffered/no
                                ;AEOI/8Ø85
                                ; -normal mode for Morrow software
ocw1    equ     base + 5        ;operation control word 1 - MASK


begin:  mvi     a,groupØ
        out     grpsel
        mvi     a,lovect + init + ltim + adi + sngl + IC4
        out     icw1
        mvi     a,hivect
        out     icw2            ;vectors begin at address FFEØh
        mvi     a,normal
        out     icw4
        out     ocw1
        ret
```

This code initializes the PIC to generate the call instructions to addresses at four byte intervals beginning at FFEØh. Jump vectors to the interrupt service routines must be placed in these locations by the system software. The interrupt service vectors are as follows:

Table 5-6: Interrupt Service Vectors

IRQ Line	Device	Call Vector (hex address)
Ø	S-1ØØ VØ	FFEØ
1	S-1ØØ V1	FFE4
2	S-1ØØ V2	FFE8
3	Serial Device 1	FFEC
4	Serial Device 2	FFFØ
5	Serial Device 3	FFF4
6	Daisy PWR line	FFF8
7	RT Clock TP line	FFFC

5.4. System Software Requirements

A typical system interrupt service routine (ISR) to service the PIC on the WBI/O must perform the following functions:

1. Enable interrupt instructions to the CPU.

2. When the interrupt occurs, the ISR saves the registers to be restored when the interrupt routine returns to the routine it interrupted.

3. Service the device which generated the interrupt.

4. Send an Enable Interrupt (EI) instruction to the CPU. This is necessary because interrupts are automatically disabled by the CPU whenever an interrupt has been received. Failure to do so prevents further interrupts to be acknowledged by the CPU. Once enabled, higher priority interrupts than the one being serviced are honored by CPU.

5. Send and EOI (end of interrupt) to the PIC. This would mean sending a 2Øh to WB I/O port BASE+4 of GROUP Ø. This allows the current ISR to be interrupted by a device of same or lower priority.

6. Restore all the registers of the interrupted routine and return to that routine. Since the ISR was invoked through use of a Call instruction, a Return instruction must be executed to restore the Stack Pointer to its original position.

6. ACE SERIAL PORTS

The WB I/O has three 8250 programmable Asynchronous Communications Elements (ACE's) which can be connected to RS-232 devices via three 25-pin D-type connectors. Each ACE has an I/O group dedicated to it - GROUPS 1, 2 and 3. The ACE's are programmable and must be initialized before they can be used. Initialization includes setting the baud rate, word length, parity, number of stop bits, and interrupt conditions. Each ACE can be programmed to generate an interrupt in response to up to ten conditions (e.g., data available, transmitter buffer empty, etc.). The interrupt is sent directly to the WB I/O PIC which can in turn pass it on to the host CPU. The interrupt handling routine then interrogates the interrupt status register of the ACE responsible for generating the interrupt, and is thus able to determine the precise cause of the interrupt.

The following chart describes the ACE devices on the WB I/O, including the location of the 8250 on the circuit board, the location of the 26-pin ribbon cable connector associated with each ACE, the I/O GROUP controlling each ACE, and the interrupt level assigned to each device by the 8259-A PIC.

Table 6-1: ACE I/O GROUP Description

	I/O GROUP #	25-pin Connector	Board Location	Interrupt Level
ACE # 1	1	P1	6D	3
ACE # 2	2	P2	5D	4
ACE # 3	3	P3	4D	5

P1 is the right-most connector with the board-oriented connectors facing you. P2 is the connector immediately left of P1 and P3 is to the left of P2.

The pins on the DB25-S type connectors P1-P3 are configured as follows (as viewed from the rear of the computer):

13	12	11	10	9	8	7	6	5	4	3	2	1
25	24	23	22	21	20	19	18	17	16	15	14	

The pins have been arranged to conform as closely as possible to the IEEE RS-232 communications equipment standards for data terminal equipment. The following is a pinout guide for the DB-25 connector.

Table 6-2: ACE Serial Connectors

	Connector Pin	Definition	ACE Mnemonic
Output	3	Transmit data	SOUT
From	4	Request to Send	RTS
WB I/O	2Ø	Data Terminal Ready	DTR
Input	2	Receive data	SIN
To	5	Received Signal Detect	RCSD
WB I/O	6	Data Set Ready	DSR
	8	Clear to Send	CTS
	1	(chassis ground)	
	7	(signal ground)	

7. PROGRAMMING THE 825Ø

Any 825Ø device on the WB I/O can be accessed if its I/O group is currently selected. Once a 1, 2 or 3 has been output to GROUP SELECT port BASE+7, ACE device number 1, 2 or 3 can be accessed. Each ACE contains internal 8-bit registers which occupy the first seven I/O ports of the WB I/O space, or ports BASE to BASE+6. The ACE registers accessed after the correct group has been selected are dependent on the status of the Most Significant Bit (MSB) of the line control register (BASE+3). If this bit is high, BASE and BASE+1 access the divisor latch low byte and high byte, respectively. Since the ACE has programmed baud rates, these registers must be programmed for the desired baud rate (refer to the data sheet on the 825Ø for the common divisor latch values). If the MSB of the line control register is low, the register at BASE becomes the RECEIVE buffer or TRANSMIT buffer, depending on whether it is a read or write operation. The register at BASE+1 becomes the Interrupt Enable register. The following is a summary of the 825Ø registers:

Table 7-1: Registers for the 825Ø

I/O Port	Operation	Condition of DLAB	Register
BASE	Write	Ø	Transmitter buffer
BASE	Read	Ø	Receiver buffer
BASE	Write	1	Divisor latch - low byte
BASE+1	Read/Write	Ø	Interrupt Enable register
BASE+1	Write	1	Divisor latch - high byte
BASE+2	Read	X	Interrupt ID register
BASE+3	Read/Write	X	Line Control register
BASE+4	Read/Write	X	Modem Control register
BASE+5	Read/Write	X	Line Status register
BASE+6	Read/Write	X	Modem Status register

X= Not important

7.1. Baud Rate

The 825Øs on the WB I/O have been hard wired so the baud rate for data coming in is the same as for data going out. The crystal used to provide the reference frequency for the three ACE devices on the WB I/O is 1.8432 Mhz. The data sheets give a broad sample of the divisors which must go into the divisor latch in order to generate the most common baud rates, and generally any baud rate may be generated from DC to 56,ØØØ baud (a zero in the divisor latch inhibits all data transmission). The formula for determining the divisor constant to produce a given baud rate is:

DIVISOR = 1.8432 M/ (BAUD RATE X 16)

Although in most applications the user will simply look up the baud rate divisor in the data sheet table, there are instances when odd ball baud rates may be useful. For example, an ACE is being used to generate interrupts at timed intervals based on the Transmitter Holding Register Empty Interrupt (see Serial Device Interrupts).

The following is a list of the divisor latch constants for the standard baud rates (values are in decimal):

Table 7-2: Divisor Latch Constants for Standard Baud Rates

Contents	Baud rate
2304	50
1536	75
1047	110
857	134.5
768	150
384	300
192	600
96	1200
64	1800
58	2000
48	2400
32	3600
24	4800
16	7200
12	9600
6	19200
3	38400
2	56000

7.2. Initialization

Though the reset pin (MR) of each 8250 is asserted during power ON or RESET, no assumptions should be made about the contents of any 8250 register unless that register has been initialized. Keep in mind that an on-board ACE cannot be accessed, much less initialized, unless its I/O group is selected. Furthermore, the Line Control, Modem Control, Interrupt Enable and Divisor Registers are normally initialized before any data can be transferred to or from an 8250.

The following three software routines are brief samples of how a WB I/O ACE device could be driven in a CP/M* type environment. All these routines adhere to CP/M* I/O protocol. The INIT routine sets up ACE # 1 to run at 9600 baud with an eight bit word, no parity and two stop bits. The Interrupt Enable Register is set to generate no interrupts, and the Modem Control Register is ignored. This initialization would be appropriate for most RS-232 CRT terminals in a non-interrupt driven environment. Assume that the WB I/O I/O has been set to begin at 48H. The cluster of assembler directives (equ's) at the beginning of these routines establish constants which hold for all three specimen routines. The comments included with these routines may be used as a general flow analysis of ACE programming.

*CP/M is a trademark of the Digital Research Corporation.

Table 7-3: Sample I/O Routines

```
group1   equ     1        ;code for first ACE (attached to J1)
base     equ     48h      ;base I/O address set by SW-7C
dll      equ     base     ;ACE baud rate divisor (lsb)
dlm      equ     base+1   ;ACE baud rate divisor (msb)
ier      equ     base+1   ;ACE interrupt enable register
lcr      equ     base+3   ;ACE line control register
lsr      equ     base+5   ;ACE line status register
rbr      equ     base     ;ACE receiver buffer register
thr      equ     base     ;ACE transmitter holding register
dlab     equ     8Øh      ;divisor latch access bit
thre     equ     2Øh      ;line status register THRE bit
dr       equ     1        ;line status register DR bit
baudl    equ     12       ;divisor latch low byte-- 96ØØ baud
baudh    equ     Ø        ;divisor latch high byte-- 96ØØ baud
wlsØ     equ     1        ;word length select bit Ø-- 8 bit word
wlsl     equ     2        ;word length select bit 1-- 8 bit word
stb      equ     4        ;stop bit count-- 2 stop bits
imask    equ     Ø        ;interrupt mask-- disable all
;


;The following routine initializes the ACE as described above
;
init:    mvi     a,group1 ;set up desired I/O group
         out     grpctl   ;select first serial device
                          ;next set up format and set dlab
         mvi     a,dlab+wlsØ+wlsl+stb
         out     lcr      ;base reg is now lsb baud rate reg
         mvi     a,baudl  ;low byte of baud rate constant
         out     dll      ;into low baud rate register
         mvi     a,baudh  ;high byte of baud rate constant
         out     dlm      ;into high baud rate register
                          ;set up format and clear dlab
         mvi     a,wlØ+wll+stb
         out     lcr      ;into line control register
         xra     a        ;zero register a
         out     lsr      ;clear data available flag in line status
         mvi     a,imask  ;interrupt mask set up
         out     ier      ;base+1 now interrupt mask- not baud
         ret              ;end of initialization routine
```

Table 7-3 Cont.

```
;
;The  following  routine will return in the accumulator any new
;character typed to ACE # 1
;
conin:  mvi     a,groupl
        out     grpctl  ;put a 1 into WB I/O group select port
                        ;make sure dlab is cleared
        mvi     a,wlsØ+wlsl+stb
        out     lcr     ;make base port the ACE data register
coninl: in      lsr     ;get line status register
        ani     dr      ;any new data from terminal?
        jz      coninl  ;if no then keep waiting
        in      rbr     ;get data
        ani     7fh     ;strip off bit 7 of input character
        ret             ;return with data in accumulator
;

;The following routine will output the character in Register C
;to ACE # 1
;
conout:  mvi    a,groupl
         out    grpctl  ;put a 1 into WB I/O GROUP SELECT port
                        ;make sure dlab is low
         mvi    a,wlsØ+wlsl+stb
         out    lcr     ;make base port the ACE data register
conoutl: in     lsr     ;get line status
         ani    thre    ;is ACE ready to transmit?
         jz     conoutl ;if not then keep waiting
         mov    a,c     ;transfer data from reg c to reg a
         out    thr     ;output character typed from terminal
         ret            ;return to calling program
;

;The following routine will return an FF in the Register A if ACE
;device  # 1 has received a new character (i.e.,  DR is set in the
;ACE line status register).  Otherwise, return a Ø.
;
status:  mvi    a,groupl
         out    grpctl  ;put a 1 into WB I/O GROUP SELECT port
         in     lsr     ;get line status
         ani    dr      ;check DR bit
         rz             ;return if reg a is zero-- no character
         mvi    a,Øffh  ;ff into reg a since character is ready
         ret
```

In the above examples, it should be noted that the GROUP SELECT port is re-initialized at the beginning of every routine. This is done to insure against inadvertently sending serial I/O instructions to the clock, parallel ports or interrupt controller of the WB I/O. Further note that before accessing the ACE data register, the format word is sent again to the Line Control Register. This is done so that port BASE of GROUP 1 will be interpreted as a data port rather than as a divisor port. This guards against a situation such as losing access to the console device due to a careless reading of the divisor latch (from a monitor or front panel, for example) without subsequently clearing DLAB.

7.3. Serial Device Interrupts

The three 825Ø ACE devices on the WB I/O each have a dedicated interrupt request line on the 8259 PIC. The chart below describes the PIC interrupt level assigned to each ACE:

Table 7-4: ACE Interrupt Assignments - 8259 PIC

Serial Device	PIC Interrupt Request Line
ACE # 1	IR3
ACE # 2	IR4
ACE # 3	IR5

7.4. ACE Interrupt Programming

As explained in the data sheet on the 825Ø, each ACE device can be programmed to generate an interrupt on any of four general conditions. These conditions are, in order of descending priority: Receiver Line Status, Received Data Available, Transmitter Holding Register Empty, and Modem Status. The Received Data Available and the Transmitter Holding Register Empty interrupts can be identified directly from the Interrupt ID Register of the source ACE.

The remaining two interrupts must use the Interrupt ID Register to point to either the Receiver Line Status Register or the Modem Status Register. These two registers each have four interrupt flags which can be read to identify the source of an ACE-generated interrupt. (The third interrupt of the Modem Status Register - The Trailing Edge of Ring Indicator, or TERI - is not usefully supported by the WB I/O, since the Ring Indicator line of each ACE is tied to +5V.) Because the 825Ø prioritizes its interrupts, the Interrupt ID Register will 'freeze' the highest priority interrupt pending by ignoring all further interrupts until the previous interrupt has been serviced. See the data sheets for further information on the 825Ø.

When using the 8250's ACE devices on the WB I/O to generate interrupts, it is advisable to set the 8259-A PIC to operate in level-mode, rather than edge-mode. In edge-mode, it is possible under certain circumstances for an ACE-generated interrupt to be 'lost'- that is, to go unrecognized. The 8250 generates one edge for an interrupt and all interrupts which occur during the time when the first interrupt is active will not generate additional edges. In this situation, the interrupt line of the 8250 remains low until all interrupts have been acknowledged, but the 8259 PIC in edge-triggered mode has seen no additional edges to indicate the presence of further interrupts.

8. THE PARALLEL DAISY-WHEEL PRINTER PORT

The WB I/O contains parallel I/O ports configured to accommodate a standard Diablo-type daisy-wheel R/O printer. These ports are brought out to the 50-pin ribbon cable connector at P5 (board location 8E - 11E) for easy attachment. The pin assignments of P5 correspond exactly to those of an internal Diablo 50 conductor flat cable connector, so simply tying the Diablo to the WB I/O via a ribbon cable with female sockets at either end is the only hardware requirement for interfacing the two devices.

The daisy-wheel interface standard requires 12 bits of data information and four strobe lines which determine the meaning of the data lines. These four strobes are:

Table 8-1: Printer Strobe Lines

$\overline{\text{RESTORE}}$	-	Send the print head to the 'home' position (position assumed when the printer is powered up).
$\overline{\text{PRINT WHEEL}}$ STROBE	-	Indicates 12 bits of data on data lines contain characters to be printed and the strike intensity of the hammer.
$\overline{\text{CARRIAGE}}$ STROBE	-	Indicates that data lines contain the appropriate number of steps and direction the print head is to be moved.
$\overline{\text{PAPER FEED}}$	-	Indicates that data lines contain valid number representing amount of paper to advance or retract.
$\overline{\text{RIBBON}}$	-	Lifts the ribbon cartridge in preparation to print a character.
$\overline{\text{SELECT}}$	-	Low to select the printer.

The last two lines are additional daisy-wheel printer control lines. They are accessed through GROUP Ø BASE+2 output port. Bit-6 generates the ribbon lift signal and bit-7 is an inverted version of the select signal. All software must account for this inversion for correct selection. (For more information on printer standards for Diablo-type systems, see referenced manual.)

Two latched output ports (plus an extra latched output bit) and one transparent input port are used to communicate with the daisy-wheel printer. These ports can be used with almost any parallel device (e.g., a Centronics-style printer or a keyboard) provided that the I/O lines are properly routed from the WB I/O connector at P5 to the target device. This additional cabling burden is standard in parallel I/O interfacing, and so should not be considered as a major disadvantage by those using the DAISY port with a non-Diablo parallel device.

The WB I/O daisy-wheel printer port occupies I/O ports BASE and BASE+1 plus a part of BASE+2 - all within I/O GROUP Ø. A single input line (BASE+1 bit-5, or the Print Wheel Ready line when interfacing with a daisy-wheel printer) is, after going to the DAISY port, inverted, then brought to IRQ 6 of the 8259-A interrupt controller to generate an interrupt whenever it goes to a low logic state. The eight input lines brought to daisy-wheel printer port BASE are also pulled up to +5V through 18Ø Ohms (nominal), and may be used with open-collector devices.

These eight input lines are inverted by an input buffer; if left unconnected, appear to software as a high.

The signal returning from the daisy-wheel printer indicates whether it can accept a new command from the WB I/O. The lines are defined as:

Table 8-2: Printer Line Commands

$\overline{\text{PRINTER READY}}$	-	Power is ON and printer is ready to accept commands.
$\overline{\text{CHECK}}$	-	Fault condition indicating either a software error (e.g. sending the print head too far in one direction) or hardware failure in the printer.
$\overline{\text{P.W. READY}}$	-	Print wheel can accept a new character address.

Table 8-2 Cont.

$\overline{\text{CARRIAGE READY}}$	-	Carriage is ready to be repositioned.
$\overline{\text{P.F. READY}}$	-	Platen motor ready to advance or retract the paper.
$\overline{\text{COVER OPEN}}$	-	Case cover was removed.
$\overline{\text{OUT OF PAPER}}$	-	Printer has run out of paper.
$\overline{\text{RIBBON OUT}}$	-	A print ribbon cartridge has not been inserted or has run out.

Connector P5, line 48, enables all daisy-wheel printer port output drivers. If this line is not tied to nominal +5 volts (if it is grounded or allowed to float) the DAISY port output lines controlled by I/O ports BASE, BASE+1 and BASE+2, remain at a high impedance state regardless of any software commands. (Note that some printers such as C. Itoh do not supply this level and are non-standard Diablo interfaces.) In the event you have chosen such a printer and are not able to jumper pin-48 of the daisy-wheel printer connector to +5 volts, you may lift 4 of chip 1ØC and tie it to pin 7 of 1ØC using a short piece of 3Ø gauge insulated wire.

> WARNING: In no way does Morrow Designs support this modification or take responsibility for products which have been modified. This solution is provided here in the unlikely event you have purchased a non-standard daisy-wheel printer and have no way in which to modify the printer itself. It should be considered a temporary solution.

The parallel ports have no special facility for generating a strobe on output or latching a strobe on input. All data lines operate as levels, so strobes must be generated in software.

The following page depicts the parallel lines available on the WB I/O, including the I/O port and bit number controlling each line and the function assigned to each line on a standard parallel Diablo-type interface. Remember, these functions have no inherent meaning to the WB I/O; it only sees so many latches. Do not preclude interfacing the WB I/O with parallel devices other than daisy-wheel printers.

Table 8-3: Daisy-Wheel Printer Signals and I/O Map

I/O Group Ø

	I/O Port	Data Bit	WB I/O and Diablo Pin #	Diablo Function
Input	BASE*	Ø	4	End of Ribbon (-)
		1	3	Paper Out (-)
		2	5	Cover Open (-)
		3	34	Paper Feed Ready (-)
		4	26	Carriage Ready (-)
		5	27 **	Print Wheel Ready (-)
		6	12	Check (-)
		7	28	Printer Ready (-)
Output	BASE	Ø	46	Data Bit 9 (256) (-)
		1	1	Data Bit 1Ø (512) (-)
		2	9	Data Bit 11 (1Ø24) (-)
		3	1Ø	Data Bit 12 (2Ø48) (-)
		4	15	Paper Feed Strobe (-)
		5	17	Carriage Strobe (-)
		6	21	Print Wheel Strobe (-)
		7	13	Restore (-)
Output	BASE+1	Ø	37	Data Bit 1 (1) (-)
		1	36	Data Bit 2 (2) (-)
		2	39	Data Bit 3 (4) (-)
		3	33	Data Bit 4 (8) (-)
		4	4Ø	Data Bit 5 (16) (-)
		5	42	Data Bit 6 (32) (-)
		6	43	Data Bit 7 (64) (-)
		7	45	Data Bit 8 (128) (-)
Output	BASE+2	6	23	Ribbon Lift (-)
		7	24	Select (-)

*These eight input lines are pulled up to +5 volts by 18Ø Ohms and inverted.

**In addition to being associated with bit-6 of the input port BASE, the Diablo Print Wheel Ready line (pin-27 of P5) is connected through an inverter to Interrupt Request line 6 (pin-24) of the 8259-A PIC. Thus, this line may be used to generate an interrupt whenever any external device brings it low (e.g., when the print wheel is ready).

The following lines on WB I/O connector P5 are tied to ground as described by Diablo interface standards:

2, 8, 11, 14, 18, 2Ø, 22, 25, 3Ø, 31, 32, 35, 38, 41, 44, 47.

(Line 24, defined by Diablo as Select (-), is also grounded.)

Unimplemented (left floating) are lines 6, 7, 29, and 5Ø.

Table 8-4: Printer Port P5 - Connector Pinouts

Top View

Back

Right	49	47	45	43	41	...	9	7	5	3	1	Left
	50	48	46	44	42	...	10	8	6	4	2	

Front

8.1. Programming the Daisy-Wheel Printer Port

As with all I/O devices on the WB I/O, the user must be careful when accessing the daisy-wheel printer port to initialize the correct I/O group - in this case, GROUP 0. Once the proper I/O group has been selected, all data output from the CPU to the parallel ports will be latched (if P5, pin-48 is at a high level) or ignored (if P5, pin-48 is grounded or allowed to float). Latched means the data output to a parallel port appears on the appropriate pins on the P5 connector, and remains there until either different data is output to the port in question or until pin-48 is floated or grounded. When pin-48 is grounded or allowed to float, all 17 parallel output pins of connector P5 enter a high impedance state.

The eight input lines from the daisy-wheel printer port are available to the CPU by reading BASE+0 (48h in standard configuration) with GROUP 0 selected. When an input instruction is directed at daisy-wheel printer port 0, the CPU reads whatever data is on the appropriate lines of connector P5 at the time the instruction is executed. There is no provision for latching the daisy-wheel printer port input data because this data is buffered only. The input daisy-wheel printer port/pin assignments are listed in the tables beginning on page 27.

The WB I/O daisy-wheel printer port inverts its input lines but does *not* invert its output lines. Daisy-wheel printers use negative logic: a low signal is taken as active. To activate any output line when talking to a daisy-wheel printer, the *software* must put the line low. Input lines from a daisy-wheel printer, on the other hand, are inverted in hardware, and so will appear to software to be active high.

8.2. Generating an Output Strobe

Generating an output strobe off any of the parallel output ports on the WB I/O requires the use of a software mask. This means the line to be strobed must be output (at most) three times in succession, changing state each time, while the data lines associated with the same port be allowed to remain unchanged. For example, to output a strobe going high-low-high on bit-6 of port BASE without changing the other seven bits being output from that port, the following routine could be used:

```
mvi  a,data    ;original data into register A
ori  40h       ;preserve data but bring bit-6 high
out  base      ;output data with bit-6 high
ani  0bfh      ;preserve data but bring bit 6 low
out  base      ;output data with bit-6 low
ori  40h       ;preserve data but bring bit 6 high
out  base      ;output data with bit-6 high
```

NOTE: GROUP 0 port BASE+2 is shared with another device on the WB I/O-- the real time clock. Be careful when outputting to this port.

8.3. The Daisy-Wheel Printer Port and Interrupts

The Print Wheel Ready status line of the daisy-wheel printer port (P5 connector, pin-27, BASE input port bit-5) is brought through an inverter to Interrupt Request line 6 of the 8259-A PIC. The PIC can generate an interrupt whenever this line goes to an active (i.e. logic low) state. To take full advantage of this interrupt option, the printer driver software should be written so that the Print Wheel Strobe (P5, pin-21, BASE output port bit-6) is not activated until all carriage positioning commands have first been sent to the printer. Print-after-space will execute significantly faster than space-after-print. When the Print Wheel Ready line goes active the printer should be able to accept another motion-then-print sequence.

A sample Diablo printer driver, including source code for the WB I/O, can be obtained from Morrow Designs.

9. THE AUXILIARY PARALLEL PORT

Besides the daisy-wheel printer port, the WB I/O contains an eight-bit, bi-directional parallel port with handshaking. The port is available at the DB15-S type connector P4 (location 12 and 13E) on the PC board.

Since the port has only a 15-pin connector, the data lines are bi-directional. The WB I/O and the external device time share the eight-bit bus. This means software must keep track of when the external device is trying to drive the eight lines to prevent both the WB I/O and the external parallel device from driving the lines simultaneously.

The port is available by accessing (read or write) port BASE+3 of GROUP Ø. There are two bits of status available from the external parallel device, FLAG1 and FLAG2. These two latched status lines, when high, indicate the external parallel device is ready to receive a character. Switch 7C determines which polarity the handshaking lines acknowledge. Switches are configured as follows:

Table 9-1: Parallel Port Switch Configuration

S7 paddle 8	-	ON if handshaking from the external parallel device is a positive-going strobe, OFF if it is a negative-going strobe. The output of this latch is referred to as FLAG1 and is high active.
S7 paddle 7	-	ON if handshaking from the external parallel device is a positive-going strobe, OFF if it is a negative-going strobe. The output of this latch is referred to as FLAG2 and is high active.

The bits may be read from GROUP Ø port BASE+1 as bits Ø and 1 respectively. Most parallel devices require the use of only one of these handshaking lines. These status lines are latched and cleared by software (output to BASE+6 with bit-1 low for FLAG1, bit-2 low for FLAG2). In addition to the two status flags, there are five port control lines available at BASE+6 of GROUP Ø. These lines are configured as follows:

Table 9-2: GROUP Ø BASE+6 Output Port Assignment

Bit	Active	Signal name	Description
Ø	high	POE	Enable data from the WB I/O auxiliary parallel output port latch onto the bi-directional data bus on P4.
1	low	RST1	Resets the handshaking latch (FLAG1) from the external device.
2	low	RST2	Resets the handshaking latch (FLAG2) from the external device.
3	low	ATTN1*	This bit gets inverted when sent out to P4 to become a positive-going edge. This informs the external parallel device that the WB I/O has a character it wishes to send out to the external device.
4	low	ATTN2*	This bit gets inverted when sent out to P4 to become a positive going edge. This informs the external parallel device that the WB I/O has a character it wishes to send out to the external device.

*Most parallel devices require only one attention line.

The pinout of the 15-pin DB15-S connector is as follows:

Pin	Polarity	Name
3	Positive	Data 7
7	Positive	Data 6
2	Positive	Data 5
6	Positive	Data 4
4	Positive	Data 3
8	Positive	Data 2
1	Positive	Data 1
5	Positive	Data Ø
12	Positive	ATTN1
13	Positive	ATTN2
14	Switch selectable	FLAG1
15	Switch selectable	FLAG2

10. THE 1990 CALENDAR/CLOCK CHIP

The 1990 CMOS crystal-controlled calendar/clock chip at location 12A supports a real-time environment by providing two functions: 1) a calendar clock accessible from software able to run off a battery, and 2) a timed interrupt generator able to provide real-time interval interrupts with three possible software programmable interval lengths. The clock uses six bits of port BASE+2, Select Line and Ribbon Lift Line of the daisy-wheel printer port. The chart below shows the WB I/O I/O ports and data bits used by the 1990, and indicates the correspondence between data bit and 1990 pin number/function.

Table 10-1: 1990 Calendar/Clock I/O Map

I/O Port BASE+2	BASE+2 Bit #	1990 Pin # & Mnemonic	1990 Function
Input	0	9 - Data Out	Output of 40-bit shift register
to CPU:	1	10 - TP	Timed pulse output
Output	0	6 - Data In	Input of 40-bit shift register
from CPU:	1	8 - Clk	Shift clock for 40-bit register
	2	3 - C0	Command input bit-0
	3	2 - C1	Command input bit-1
	4	1 - C2	Command input bit-2
	5	4 - STB	Strobe input

Table 1Ø-2: uPD199ØC Pinout Definitions:

Name	Pin #	Definition
C2	1	Mode select pin. When high, this pin selects the time pulse output register. When low, this pin selects the calendar clock mode. This pin is set low to read or set the time and high to set the time pulse interrupt frequency.
C1	2	This pin is used to select the time pulse interrupt if C2 is high or enable the shift register if C2 is low.
CØ	3	This pin is used to select the time pulse interrupt frequency if C2 is high. If C2 is low, and CØ is low, the contents of the shift register is written into the clock. If CØ is high and C2 is low, the clock contents are written into the shift registers for reading.
STB	4	This line is used to strobe the contents of the CØ - C2 lines into the clock chip, for selecting the various command modes.
CS	5	When high allows the CLK, STB and OE lines to reach the internal circuitry of the clock chip. Morrow Designs hardware ties this line high unless there is a system power failure.
Data In	6	The serial data input to the chip allowing the clock's shift register to be altered for setting the clock.
GND	7	Ground pin (Ø volts)
CLK	8	This pin is used to clock data into or out of the clock shift register. Data is clocked into the shift register on the rising edge of the clock. Data is clocked out of the shift register on the falling edge of the clock.
Data Out	9	The serial data output line of the clock allowing contents of the shift register to be clocked into the system CPU. This data is available by reading bit-Ø of WB I/O port BASE+2.
TP	1Ø	Time pulse output provides interrupts at preset intervals. This output is available by reading bit-1 of WB I/O port BASE+2.

Table 1Ø-2, Cont.

OE	11	Output enable pin, when high, allows the TP and data out pins to be read. Morrow Designs hardware ties this pin high unless there is a system power failure.
XTAL1	12	Crystal clock input (32.768 Khz).
XTAL2	13	Crystal clock input (32.768 Khz).
VDD	14	Power supply input (3.6 V max.).

The CØ - C2 inputs can be summarized as follows:

Function	C2	C1	CØ	
Register hold	Ø	Ø	Ø	GROUP Ø
Register shift	Ø	Ø	1	GROUP Ø
Write shift register into the clock	Ø	1	Ø	GROUP Ø
Read the clock time into shift register	Ø	1	1	GROUP Ø
TP = 64 Hz	1	Ø	Ø	GROUP 1
TP = 256 Hz	1	Ø	1	GROUP 1
TP = 2Ø48 Hz	1	1	Ø	GROUP 1
Test mode (32 Hz)	1	1	1	GROUP 1

1Ø.1. Clock Initialization

The clock powers up in the test mode. The TP output is clocking at 32 Hz. The clock TP pulse must be set to one of the three TP values before any clock Group Ø (any command with C2 set low) command will execute. If at any time during operation the user sets the clock to 'Test Mode', he must again select one of the other TP values before attempting any clock Group Ø commands. The test mode should NOT be considered as one of the possible timed interrupt values unless these peculiarities are acknowledged through software.

For a 64 Hz TP the power up sequence would look like:

> Set STB bit, CØ and C1 bits low and C2 bit high (1Øh) and output to WB I/O port BASE+2. Then, with the CØ - C2 bits unchanged, set the STB bit high (3Øh) and output to WB I/O port BASE+2. Then, again with the CØ - C2 bits unchanged, set the STB bit low and output to WB I/O port BASE+2. From this point on, any one of the clock commands may be executed.

Any command issued to the clock requires the STB bit to be low initially, then brought high and then low again with the data unchanged. This is all accomplished by manipulating bit-5 of port BASE+2. In order to write data into the shift register, the user first uses the Register Shift mode to enable the shift register (strobe-in with CØ high, C1 and C2 low). Now data may be clocked into the shift register. After all the bits have been clocked into the shift register, the user then enters the Time Set mode (strobe-in with C1 high, CØ and C2 low). This writes the contents just shifted into the shift register into the clock itself. Conversely, when reading the clock, the Time Read mode must be entered first (CØ and C1 high, C2 low). This takes the clock's internal time and places it in the shift register. The data may then be clocked out from the shift register.

1Ø.2. Clock Programming

The data sheets on the 199Ø chip should be studied before attempting to program this device. The 199Ø stores the time of day, day of week, and month of year in an internal 4Ø-bit shift register which is accessible to the WB I/O user through bit-Ø of I/O port BASE+2 of GROUP Ø. Commands to set or read time must be strobed into this port using bit-4 as the strobe bit. The 4Ø bits of time data must be clocked in or out using bit-1 as the clock bit. The format of this internal 4Ø-bit shift register is seven four-bit binary coded decimal nibbles and, for the month of the year, one hex nibble. The 4Ø-bit shift register is a FIFO - first in, first out - the first being the Least Significant Bit (LSB). Thus, the first bit in or out is always the LSB of the single seconds nibble, and the last bit out is always the Most Significant Bit (MSB) of the month of the year nibble.

Note in the following table how each individual nibble seems to coded backwards.

Table 10-3: Time Format of the 1990 40-Bit FIFO

Bits 1 to 8 -- Seconds (0 to 59)

	Seconds Units				Tens of Seconds			
1990 bits	1 LSB	2	3	4 MSB	5 LSB	6	7	8 MSB

Example: 38 seconds would be stored as follows:

1990 bits	1	2	3	4	5	6	7	8
Logic Level	0	0	0	1	1	1	0	0
Interpretation:	8				3			

Bits 9 to 16 -- Minutes (0 to 59)

	Minutes Units				Tens of Minutes			
1990 bits	9 LSB	10	11	12 MSB	13 LSB	14	15	16 MSB

Example: 41 minutes would be stored as follows:

1990 bits	9	10	11	12	13	14	15	16
Logic Level	1	0	0	0	0	0	1	0
Interpretation:	1				4			

Bits 17 to 24 -- Hours (0 to 23)

	Hours Units				Tens of Hours			
1990 bits	17 LSB	18	19	20 MSB	21 LSB	22	23	24 MSB

Example: 11 o'clock p.m. (2300 hours) would be stored as follows:

1990 bits	17	18	19	20	21	22	23	24
Logic Level	1	1	0	0	0	1	0	0
Interpretation:	3				2			

Table 1Ø-3 Cont.

Bits 25 to 32 -- Day of Month (1 to 31)

	Day Units				Tens of Days			
199Ø bits	25 LSB	26	27	28 MSB	29 LSB	3Ø	31	32 MSB

Example: the 14th of the month

199Ø bits	25	26	27	28	29	3Ø	31	32
logic level	Ø	Ø	1	Ø	1	Ø	Ø	Ø
Interpretation:	4				1			

Bits 33 to 36 -- Day of the Week (Ø to 6)

199Ø bits	33 LSB	34	35 MSB	36 Garbage Bit

Sunday = Ø
Monday = 1
Tuesday = 2
. . .
Saturday = 6

Example: Thursday

199Ø bits	33	34	35	36
Logic Level	Ø	Ø	1	Ø
Interpretation:	4			

Bits 37 to 4Ø -- Month of the Year (Ø to B Hex)

199Ø bits	37 LSB	38	39 MSB	4Ø Garbage Bit

January = Ø
February = 1
March = 2
. . .
November = A Hex
December = B Hex

Example: July

199Ø bits	37	38	39	4Ø
Logic Level	1	1	1	Ø
Interpretation:	7			

10.3. Calendar Clock Idiosyncracies

Once the 40-bit shift register of the 1990 has been set with the desired time and date, it automatically increments the time and date for later reference. Note, however, that the 1990 considers all months to have 31 days, so September, April, June and November - and certainly February - require a special update at the end of the month to keep the calendar current.

10.4. Strobe and Clock Timing

The 1990 is not capable of reading or writing serial data fast enough to keep up with the CPU unless the clock and strobe bits are prolonged for about 700 micro-seconds. This can be easily accomplished in software.

10.5. Time/Date Software

Writing the time to the 1990 requires a four step procedure:

1: Select I/O GROUP 0 of the WB I/O.

2: Strobe the Register Shift Command to I/O port BASE+2. This is done outputting first a 04H, then a 24, then a 04H to port BASE+2 (but see note below).

3: Clock forty consecutive bits to the data-in pin of the 1990. Each bit is sent via three output instructions to I/O port BASE+2 with suitable delays in between. The the data-bit (bit-0) stays the same, the Strobe Bit (bit-5) stays low, and the Clock Bit (bit-1) is first low, then high, then low again (see note below).

4: Strobe the Set Time Command to I/O port BASE+2. This is done by outputting first an 8H, then a 28H, then an 8H to port BASE+2 (see note below).

NOTE: Bits 6 and 7 of WB I/O port BASE+2 of GROUP 0 control the Ribbon Lift Line of the daisy-wheel printer port and the Printer Select Line. These bits should not be carelessly altered when outputting to the clock.

1Ø.6. Software Flow for Reading the Time/Date

Reading the time from the 199Ø requires a four step procedure:

1: Select I/O GROUP Ø of the WB I/O.

2: Strobe the Read Time Command to I/O port BASE+2. This is done by outputting first a CH, then a 2CH, then a CH to port BASE+2 (see note on previous page).

3: Strobe the Register Shift Command to I/O port BASE+2. This is done outputting first a 24H, then a 4, then a 24H to port BASE+2 (see note on previous page.)

4: Clock forty consecutive bits from the data-out pin of the 199Ø. Each bit is read via two output and one input instructions from I/O port BASE+2, with suitable delays in between, in which the Strobe Bit (bit-5) stays low, and the Clock Bit (bit-1) is first low, then high, then low again (see note on previous page).

The appendix contains a source listing of a CP/M compatible program which can write the time to the 199Ø clock or read it back.

It is probably a good idea to have interrupts disabled when writing to or reading from the clock, since a lengthy interrupt service routine could cause the data read or written to be inaccurate.

1Ø.7. The Timed Interrupt Generator

In addition to being a calendar/clock, the 199Ø is capable of generating interrupts at timed intervals. The interrupts generated by the 199Ø are routed to Interrupt Request number 7 of the 8259-A PIC. In order for these interrupts to be received properly, the PIC must be set to operate in level, rather than edge, mode. Three interval times are available and are selected under software control. The intervals are:

1) Once every .488 milliseconds, or 2Ø48 interrupts per second

2) Once every 3.9 milliseconds, or 256 interrupts per second

3) Once every 15.Ø milliseconds, or 64 interrupts per second

1Ø.8. Generating a Timed Interrupt

As indicated in the data sheet on the 199Ø, the TP (Timed Pulse) output, which is the source of the 199Ø interrupts, can be programmed to oscillate with a 5Ø% duty cycle at one of three frequencies. These frequencies are selected by strobing the appropriate data into I/O port BASE+2. The data to be strobed out to the clock port and the corresponding oscillation frequency of the 199Ø TP line are shown below:

To set TP to the desired time, strobe the following bytes consecutively to I/O port BASE+2 of GROUP Ø. (Note that the last column indicates time between interrupts.)

Table 1Ø-4: Setting the Timed Pulse

Output string to BASE+2	TP Frequency	Interrupts
3ØH, 1ØH, 3ØH	64 Hz	15.Ø msec
31H, 11H, 31H	256 Hz	3.9 msec
34H, 14H, 34H	2,Ø48 Hz	.488 msec

NOTE: Bits 6 and 7 of I/O port BASE+2 of GROUP Ø control the Ribbon Lift Line of the DAISY printer port and Printer Select Line. These bits should not be carelessly altered when outputting to the clock.

1Ø.9. Clearing the Timed Interrupts

Any input instruction directed at I/O port BASE+2 clears the interrupt request generated by the 199Ø. This action does not involve the 199Ø clock chip, but clears the flip-flop through which the 199Ø TP output is latched and converted to a constant level before reaching the 8259-A PIC. The data obtained from this instruction may be ignored.

1Ø.1Ø. A Good Random Bit

The output of the 199Ø TP has a 5Ø% duty cycle; it is at a high logic state for the same length of time it is at a low logic state. The state of this line may be examined at any time by reading bit-1 of I/O port BASE+2 of GROUP Ø, the same port used for reading and writing clock data. If examined immediately after the occurrence of a TP interrupt, the line will be high since it is the high-going edge of TP that generates the interrupt.

10.11. Generating Interrupts at Non-standard Intervals

If the interval selection available on the 1990 does not fit the user's application, a broader selection is possible by using an on-board 8250 ACE - just program the ACE to generate an interrupt whenever the Transmitter Buffer is empty.

11. LIST OF REFERENCES

1. INS 825Ø Asynchronous Communications Element, (National Semiconductor Corporation, 1978).

2. 8259A Programmable Interrupt Controller, (Intel Corporation, 1978).

3. MOS Digital Integrated Circuit PD 199ØC, (NEC Electron, Inc., undated).

4. Standard Specifications for S-1ØØ Bus Interface Devices, (IEEE, 1979).

5. Mult I/O User's Reference Manual, (Morrow Designs, preliminary edition available only).

6. Model 12ØØ Hytype Printer Reference Manual, (Diablo Systems, undated).

APPENDIX A

SOME NOTES AND CAUTIONS

In situations where one ISR is interrupted by another ISR, care should be taken to preserve CPU registers which might be altered, and so, sabotage the interrupted service routine. The same holds for routines that are time-dependent. They should be written to preserve their integrity in case they are interrupted. For example, if two routines use the same ACE device, it is possible for a routine to check, say, the TBE status bit, find the device to be ready, prepare to send data to the device, get interrupted, and proceed, when control is regained, to send data to a device that may no longer be ready.

If the CPU sends an INTA pulse (an Interrupt Acknowledge) to the master PIC when no IRQ line on the PIC is asserted, the PIC will issue the CALL vector associated with IRQ7. It is very easy to induce this situation by grounding by hand the vectored interrupt lines.

The CP/M * operating system contains a ultility program, DDT, which can be useful in developing software. This program has the provocative feature of enabling interrupts (issuing an EI command) whenever the "G" command is given. Under the right circumstances this can cause havoc if the user is caught unaware.

The following page gives a graphic illustration of the program flow which occurs when a program is interrupted and the ISR which results is itself interrupted.

*CP/M is a copyright of Digital Research

ILLUSTRATIONS OF PRIORITY INTERRUPT LEVELS

```
     Main program -->
 ---------------------A1---------------------->
                      / \
                     /   \
                    /     \
                   /       \
                  /         \
                 /           \
                /             \
               /    ISR A -->  \
              /                 \
             /--A2--A3-B1---A4---A5
                      / \
                     /   \
                    /     \
                   /       \
                  /         \
                 /           \
                /             \
               /    ISR B -->  \
              /                 \
             /--B2--B3------B4---B5
```

A1: Main program is interrupted by Interrupt Request A and PIC vectors program off to Interrupt Service Routine A (ISR A).

A2: ISR A removes the cause of its interrupt.

A3: ISR A issues an EI (Enable Interrupts) command to the CPU. This permits the servicing of a HIGHER priority interrupt.

B1: IRQ B (Interrupt Request B), a higher priority than IRQ A, causes ISR A to be interrupted, and the PIC vectors the program OFF to ISR B.

B2: ISR B removes the cause of its interrupt.

B3: ISR B issues an EI command to the CPU. ISR B may now in turn be interrupted by a higher priority IRQ.

B4: ISR B issues an EOI (End of Interrupt) command to the PIC. ISR B may be interrupted by SAME or LOWER priority IRQ.

B5: ISR B exits its service routine with a RET instruction. Control returns to ISR A.

A4: ISR A issues an EOI command to the PIC.

A5: ISR A exits its service routine with a RET instruction. Control returns to the main program.

APPENDIX B

WB I/O CONNECTORS, SWITCHES AND JUMPER OPTIONS

The following is a list of connectors, switch settings and jumper options and their function:

The WB I/O board has the following I/O connectors available at the rear of the board. As viewed from the rear of the Decision 1 cabinet they are left to right:

Connector	PC Location	Function
P4	12E - 13E	Auxiliary 8-bit multi-purpose bi-directional parallel port.
P5	8E - 12E	Although not actually visible from the rear panel, this 5Ø-pin header on the WB I/O is the connection for the daisy-wheel printer.
P3	9E - 1ØE	ACE Serial Device #3 - This port is usually reserved for printers in systems which require a serial printer.
P2	6E - 7E	ACE Serial Device #2 - normally the second CRT terminal port.
P1	2E - 3E	ACE Serial Device #1 - This port is the standard console I/O port for all Morrow Designs software.
P6	1C	Although not visible from the rear this connector is visible when the Decision 1 cover is open. This connector is the power input to the WB I/O. See table below for pin configuration.

Pin Configuration - Power Input

1	-	+ 16V
2	-	+ 16V
3	-	+ 8V
4	-	+ 8V
7	-	ground
8	-	ground
9	-	ground
1Ø	-	ground

Switch at board location 7C is used by Morrow Design's software to set the BASE port address, wait states and polarity of auxiliary parallel port handshaking inputs. The normal base address for all Morrow Designs software is 48 hex. The following summarizes this switch:

Paddle	Function
1	ON causes the WB I/O to generate a wait state on I/O and Interrupt Acknowledged cycles during which the WB I/O has been selected. The ACE and PIC chips have a minimum access time of 25Ø ns. Systems which require faster access times should have this switch ON. This switch is normally ON in Decision 1 systems.
2	Maps to CPU address line A7 for address of BASE port (normally ON for Morrow Designs software).
3	Maps to CPU address line A6 for address of BASE Port (normally OFF for Morrow Designs software).
4	Maps to CPU address line A5 for address of BASE port (normally ON for Morrow Designs software).
5	Maps to CPU address line A4 for address of BASE port (normally ON for Morrow Designs software).
6	Maps to CPU address line A3 for address of BASE port (normally OFF for Morrow Designs software).
7	When OFF allows parallel handshake latch to respond to a strobe of negative polarity.
8	When OFF allows parallel handshake latch to respond to a strobe of negative polarity.

Switch at board location 1ØA is used to determine the baud rate for the on-board serial channels. The software reads these switches (at GROUP Ø BASE+1) after a power-up or reset sequence and initializes the proper baud rates to perform the following:

Paddle	Function
1	Serial channels baud rate select - normally ON
2	Serial channels baud rate select - normally ON
3	Serial channels baud rate select - normally ON
4	Not yet dedicated
5	Not yet dedicated
6	Not yet dedicated
7	Not connected
8	Not connected

The baud rates are determined as follows:

Paddle 1	Paddle 2	Paddle 3	Baud rate
OFF	OFF	OFF	110
OFF	OFF	ON	300
OFF	ON	OFF	1200
OFF	ON	ON	2400
ON	OFF	OFF	4800
ON	OFF	ON	9600 (default)
ON	ON	OFF	19200
ON	ON	ON	Automatic

Jumpers on the WB I/O Board

Jumper	Board location	Function
J1	8A	IN causes the data read from the auxiliary parallel port input latch (BASE+3 of GROUP Ø) to be latched into the auxiliary parallel port output latch (BASE+3 of GROUP Ø). Normally this jumper is not installed.
J2	8C	Jumper between B and C of the WB I/O PIC is not a master and is not to respond to the CPU Interrupt Acknowledge signal. Jumper between A and B if the WB I/O is the master PIC and is to recognize the Interrupt Acknowledge line. This jumper is normally installed between A and B.
J3	8C	IN allows the INTR output of the PIC to drive the S-1ØØ PINT line. This jumper must be IN if the WB I/O PIC is to be the master. If this PIC is a slave, the pad closest to chip 8C is connected to one of the S-1ØØ VI lines at location 3C and the jumper is removed. This jumper is normally installed. Remove in systems where no interrupts are used.
J4	2C	Selects which S-1ØØ vectored interrupt line (if any) will be monitored by the PIC of the WB I/O. Pad A connects to PIC IRQØ line. Pad B connects to PIC IRQ1 line. Pad C connects to PIC IRQ2 line. The pc etch has these lines hard wired to the VIØ - VI2 lines respectively so no jumpers are required for normal operation. Pads are provided for user reconfiguration if necessary.
J5	13A	Battery backup for the WB I/O on-board clock. A 3 - 5 volt source (5 V battery maximum) with 15 to 2ØK Ohm series resistors for circuit protection may be connected to J5 to supply power to the clock when AC power has been removed from the system. The connector is labeled for correct polarity, please take note.
J6	8E	RESET switch inputs to the WB I/O. Shorting switch across these pins causes RESET of the CPU board and most bus slaves. The front panel RESET switch of the Decision 1 connects to these lines.

The factory configuration in brief:

Switch 7C Paddle:

1 - ON
2 - ON
3 - OFF
4 - ON
5 - ON
6 - OFF
7 - OFF
8 - OFF

Switch 1ØA Paddle:

1 - ON
2 - ON
3 - ON
4 - ON
5 - OFF
6 - ON
7 - OFF
8 - OFF

J1 Not installed

J2 Jumpered A to B

J3 Installed

J4 No jumpers

J5 Battery - user supplied

J6 Connected to front panel reset

APPENDIX C
TIME SET SOFTWARE

The following program sets and reads the clock/calendar. The program runs under CP/M and assumes the I/O board to be addressed at I/O port 48h.

To set the time using this program, type:

TIME www MMM dd hh mm ss (pm/am)

where www are the first three letters of the day, MMM are the first three letters of the month, dd are the decimal minutes of the hour and ss the decimal seconds of the minute.

A twelve hour format may be used if either am or pm is typed at the end of the string. Otherwise data is assumed to be in 24 hour format. Spaces should separate the data fields. Day of week and month of year may exceed three characters but only the first three are analyzed. Leading zeros may also be omitted as long as one character appears in the field in question.

For example, typing:

TIME MON NOV 17 7 3Ø Ø AM

would set the clock/calendar to Monday, November 17, 7:3Ø:ØØ a.m.

To read the clock, simply type:

TIME

```
                                SUBTTL '(c) Morrow Designs Inc.'
                                Title  'Decision 1 Real-time Clock Software'

                        ;*****************************************************************
                        ;*                                                               *
                        ;* Time display/set program for Thinker Toys WBI/O board.        *
                        ;*                                                               *
                        ;* Bobby Dale Gifford.                                           *
                        ;* 9/25/80                                                       *
                        ;*                                                               *
                        ;* Revised for Decision I/O on 10/5/81  BJG                      *
                        ;*                                                               *
                        ;*****************************************************************

 0000'                          aseg

 000A                   rev     equu    10              ;Revision # x.x

 0048                   base    equ     48h             ;Base of Mult I/O ports
 004F                   grpsel  equ     base+7          ;Group select
 004A                   clk     equ     base+2          ;Clock port
 0002                   clkclk  equ     2               ;Clock clk bit
 0008                   clkc1   equ     8               ;Clock c1 bit
 000C                   rclk    equ     0ch             ;Read clock command
 0020                   cstb    equ     20h             ;Clock strobe bit
 0004                   shft    equ     4               ;Shift bits command
 0010                   tp64    equ     10h             ;Output tick pulse at 64 hz
 0000                   reghld  equ     0               ;Register hold command
 0008                   wclk    equ     8               ;Write clock command

 0005                   bdos    equ     5               ;Bdos entry point
 0081                   cbuff   equ     81h             ;Command buffer string
 0080                   clen    equ     80h             ;Command length byte
 0000                   wboot   equ     0               ;Warm boot location
 000B                   const   equ     11              ;Get constat function #
 0009                   pstr    equ     9               ;Print string function #
 000A                   readcon equ     10              ;Read console buffer
 000D                   acr     equ     0dh             ;Carriage return
 000A                   alf     equ     0ah             ;Line feed

                                org     100h            ;Transient program area

 0100   2A 0006         start:  lhld    bdos+1          ;Set up stack
 0103   F9                      sphl
 0104   CD 03B6                 call    skipb           ;Skip command line blanks
 0107   CA 0261                 jz      display         ;No command line
```

```
 010A     21 03F4             sett:   lxi     h,days          ;Array of string pointers to match
 010D     CD 0218                     call    match3          ;Look for match
 0110     CA 0380                     jz      exit            ;No match
 0113     11 FC0C                     lxi     d,0 -  days     ;Form index
 0116     19                          dad     d
 0117     7D                          mov     a,l             ;Get low byte
 0118     37                          stc                     ;Clear the carry
 0119     3F                          cmc
 011A     1F                          rar                     ;Divide index by 2
 011B     32 03F3                     sta     wekmon          ;Day of week finished

 011E     21 044B                     lxi     h,months        ;Array of string pointers to match
 0121     CD 0218                     call    match3          ;Look for match
 0124     CA 0380                     jz      exit            ;No match
 0127     11 FBB5                     lxi     d,0 - months    ;Form index
 012A     19                          dad     d
 012B     7D                          mov     a,l             ;Get low byte
 012C     37                          stc                     ;Clear the carry
 012D     3F                          cmc
 012E     17                          ral
 012F     17                          ral
 0130     17                          ral
 0131     47                          mov     b,a             ;Save in B
 0132     3A 03F3                     lda     wekmon          ;Or in with day
 0135     B0                          ora     b
 0136     32 03F3                     sta     wekmon
 0139     CD 01CE                     call    bcd2            ;Scan for two valid bcd digits
 013C     DA 0380                     jc      exit
 013F     32 03F2                     sta     date            ;New date
 0142     CD 01CE                     call    bcd2            ;Scan for two more valid bcd digits
 0145     DA 0380                     jc      exit
 0148     32 03F1                     sta     hour            ;New hour
 014B     CD 01CE                     call    bcd2            ;Scan for two more valid bcd digits
 014E     DA 0380                     jc      exit
 0151     32 03F0                     sta     minutes         ;New minutes
 0154     CD 01CE                     call    bcd2            ;Scan for last valid bcd digits
 0157     DA 0380                     jc      exit
 015A     32 03EF                     sta     seconds         ;New seconds
 015D     CD 03B6                     call    skipb           ;Skip trailing blanks
 0160     CA 017B                     jz      noap
 0163     CD 03D0                     call    scan
 0166     FE 50                       cpi     'P'             ;Check for AM or PM
 0168     F5                          push    psw
 0169     CC 0395                     cz      uphrs
 016C     F1                          pop     psw
```

```
 016D     FE 41                          cpi     'A'
 016F     CC 03A1                        cz      dwnhrs
 0172     CD 03AC                        call    skipc
 0175     CD 03B6                        call    skipb
 0178     C2 0380                        jnz     exit            ;If anything remaining, then error

 017B     3E 00                  noap:   mvi     a,reghld        ;Issue register hold command
 017D     CD 0360                        call    setup
 0180     3E 10                          mvi     a,tp64          ;Set up clock pulse
 0182     CD 0360                        call    setup
 0185     11 0513                        lxi     d,waitmsg       ;Wait for carriage return
 0188     CD 0389                        call    pmsg
 018B     11 0534                        lxi     d,ibuff         ;Read console
 018E     0E 0A                          mvi     c,readcon
 0190     CD 0005                        call    bdos
 0193     CD 01A2                        call    writec          ;Write the time
 0196     11 04CE                        lxi     d,acralf
 0199     CD 0389                        call    pmsg
 019C     CD 0276                        call    displ1          ;Display the current time
 019F     C3 0000                        jmp     wboot           ;All done

                                 ;*****************************************************************
                                 ;*                                                               *
                                 ;* Writec does the actual clock time writing. This routine must  *
                                 ;* not be interupted.                                            *
                                 ;*                                                               *
                                 ;*****************************************************************

 01A2     AF                     writec: xra     a               ;Select group 0
 01A3     D3 4F                          out     grpsel
 01A5     3E 04                          mvi     a,shft          ;Shift command
 01A7     CD 0360                        call    setup
 01AA     E5                             push    h               ;Save clock data address
 01AB     1E 08                  wbyte:  mvi     e,8             ;Bit shift counter
 01AD     23                             inx     h               ;Bump to next byte of data
 01AE     7E                     wbit:   mov     a,m             ;Get current byte of data
 01AF     1F                             rar                     ;LSB into carry
 01B0     77                             mov     m,a             ;Save current byte
 01B1     17                             ral                     ;Carry into LSB
 01B2     E6 01                          ani     1               ;Through away useless bits
 01B4     E3                             xthl                    ;Recover address of clock data
 01B5     B6                             ora     m               ;Get current state
 01B6     E3                             xthl                    ;Recover current byte counter
 01B7     CD 0346                        call    clkstb          ;Strobe in one bit
 01BA     1D                             dcr     e               ;Update bit counter
 01BB     C2 01AE                        jnz     wbit            ;Same byte ?
```

```
 01BE    15                          dcr     d               ;Update bye counter
 01BF    C2 01AB                     jnz     wbyte           ;All done ?
 01C2    E1                          pop     h               ;Recover address of clock data
 01C3    7E                          mov     a,m             ;Get current state
 01C4    F6 08                       ori     wclk            ;Set write clock bit
 01C6    CD 0344                     call    clkcmd          ;Issue write time command
 01C9    EE 08                       xri     wclk            ;Turn off write time command
 01CB    C3 0344                     jmp     clkcmd

                             ;*****************************************************************
                             ;*                                                               *
                             ;* Bcd2 scans the command line for up to two valid ascii digits  *
                             ;* and returns the result as a packed bcd byte in reg A.         *
                             ;*                                                               *
                             ;*****************************************************************

 01CE    CD 03B6             bcd2:   call    skipb           ;Skip any preceeding blanks
 01D1    CD 03D0                     call    scan            ;Get first char of day of month
 01D4    37                          stc                     ;Carry is error
 01D5    C8                          rz
 01D6    FE 3A                       cpi     ':'
 01D8    CA 01CE                     jz      bcd2
 01DB    FE 2C                       cpi     ','
 01DD    CA 01CE                     jz      bcd2
 01E0    CD 020E                     call    digit           ;Check for valid decimal digit
 01E3    D8                          rc
 01E4    47                          mov     b,a             ;Save in B
 01E5    CD 03D0                     call    scan
 01E8    CA 020A                     jz      okd
 01EB    FE 2C                       cpi     ','             ;Check for end of day of month
 01ED    CA 020A                     jz      okd
 01F0    FE 20                       cpi     ' '
 01F2    CA 020A                     jz      okd
 01F5    FE 3A                       cpi     ':'
 01F7    CA 020A                     jz      okd
 01FA    CD 020E                     call    digit
 01FD    D8                          rc
 01FE    37                          stc                     ;Clear the carry
 01FF    3F                          cmc
 0200    F5                          push    psw             ;Save low nibble
 0201    78                          mov     a,b
 0202    17                          ral                     ;Put previous digit into high nibble
 0203    17                          ral
 0204    17                          ral
 0205    17                          ral
 0206    47                          mov     b,a             ;Save in B
```

```
 0207     F1                        pop     psw             ;Recover low digit
 0208     B0                        ora     b               ;Form byte
 0209     47                        mov     b,a             ;Save in B
 020A     78                okd:    mov     a,b             ;Recover day of month
 020B     37                        stc                     ;No error
 020C     3F                        cmc
 020D     C9                        ret

                            ;*****************************************************************
                            ;*                                                               *
                            ;* Digit checks if the char in reg A is a valid ascii digit.     *
                            ;*                                                               *
                            ;*****************************************************************

 020E     FE 30             digit:  cpi     '0'             ;Less than 0
 0210     D8                        rc
 0211     FE 3A                     cpi     '9'+1           ;Greater than 9
 0213     3F                        cmc
 0214     D8                        rc
 0215     D6 30                     sui     '0'             ;Strip off ascii bias
 0217     C9                        ret

                            ;*****************************************************************
                            ;*                                                               *
                            ;* Match3 guarentees that at least three characters are matched  *
                            ;* with the command line.                                        *
                            ;*                                                               *
                            ;*****************************************************************

 0218     3E 03             match3: mvi     a,3             ;Clear match count
 021A     32 0542                   sta     mcnt
 021D     5E                        mov     e,m             ;Get current string pointer
 021E     23                        inx     h
 021F     56                        mov     d,m
 0220     23                        inx     h
 0221     7B                        mov     a,e             ;Check if all done
 0222     B2                        ora     d
 0223     C8                        rz                      ;No match
 0224     E5                        push    h               ;Save current array pointer
 0225     2A 0540                   lhld    scanpnt         ;Save current scan pointer
 0228     E5                        push    h
 0229     3A 0080                   lda     clen            ;Save current command length
 022C     F5                        push    psw
 022D     CD 03D0           mtchmo: call    scan            ;Scan and convert to upper case
 0230     CA 0255                   jz      nomatch         ;No match if out of chars
 0233     CD 03E5                   call    toupper
```

```
0236    47                      mov     b,a             ;Save in B
0237    1A                      ldax    d               ;Get next char in string
0238    13                      inx     d               ;Bump string pointer
0239    CD 03E5                 call    toupper         ;Convert to upper case
023C    B8                      cmp     b               ;Does it match ?
023D    C2 0255                 jnz     nomatch         ;No match
0240    3A 0542                 lda     mcnt            ;Get match count
0243    3D                      dcr     a               ;Matched three ?
0244    32 0542                 sta     mcnt            ;Save match count
0247    C2 022D                 jnz     mtchmo          ;Match more ?
024A    CD 03AC                 call    skipc           ;Skip rest of characters
024D    E1                      pop     h               ;Through away old scan pointer
024E    E1                      pop     h               ;Through away old command length
024F    E1                      pop     h               ;Recover array pointer
0250    2B                      dcx     h               ;Backup array pointer
0251    2B                      dcx     h
0252    C0                      rnz                     ;No error return
0253    3C                      inr     a               ;No error return
0254    C9                      ret
0255                    nomatch:
0255    F1                      pop     psw             ;Recover command length
0256    32 0080                 sta     clen            ;Restore command length
0259    E1                      pop     h               ;Recover scan pointer
025A    22 0540                 shld    scanpnt         ;Restore scan pointer
025D    E1                      pop     h               ;Recover array pointer
025E    C3 0218                 jmp     match3          ;Try again

                        ;*****************************************************************
                        ;*                                                               *
                        ;*  Display continually displays the time as long as nothing is  *
                        ;*  typed on the console.                                        *
                        ;*                                                               *
                        ;*****************************************************************

0261                    display:
0261    CD 0276                 call    displ1          ;Display one time line
0264    0E 0B                   mvi     c,const         ;Check console for char
0266    CD 0005                 call    bdos
0269    A7                      ana     a               ;If anything typed then reboot
026A    C2 0000                 jnz     wboot
026D    11 04CC                 lxi     d,acrmsg        ;Print carriage return only
0270    CD 0389                 call    pmsg
0273    C3 0261                 jmp     display         ;Go print the time again

                        ;*****************************************************************
                        ;*                                                               *
```

```
                                ;* Displ1 displays the current time once.                         *
                                ;*                                                                *
                                ;******************************************************************

 0276     CD 031B               displ1: call    readc           ;Read the clock - watch out if interupts are on
 0279     3A 03F3                       lda     wekmon          ;Get the day of the week
 027C     E6 07                         ani     7               ;Through away irrelevent bits
 027E     17                    okday:  ral                     ;Multiply by 2
 027F     5F                            mov     e,a             ;Form 16 bit offset
 0280     16 00                         mvi     d,0
 0282     21 03F4                       lxi     h,days          ;Array of string pointers
 0285     19                            dad     d               ;Form absolute address of string
 0286     5E                            mov     e,m             ;Get low string address byte
 0287     23                            inx     h               ;Point to high byte
 0288     56                            mov     d,m             ;Get high byte
 0289     7B                            mov     a,e             ;Check for invalid day
 028A     B2                            ora     d
 028B     CA 0276                       jz      displ1          ;Start over again if invalid
 028E     CD 0389                       call    pmsg            ;Print the day

 0291     3A 03F3                       lda     wekmon          ;Get the month
 0294     1F                            rar                     ;Adjust for proper offset
 0295     1F                            rar
 0296     1F                            rar
 0297     E6 1E                         ani     1eh             ;Multiply by two and through out
                                                                ;       irrelevent bits
 0299     5F                            mov     e,a             ;Form 16 bit offset
 029A     16 00                         mvi     d,0
 029C     21 044B                       lxi     h,months        ;Array of string pointers
 029F     19                            dad     d               ;Form absolute address of string
 02A0     5E                            mov     e,m             ;Get low string address byte
 02A1     23                            inx     h               ;Point to high byte
 02A2     56                            mov     d,m             ;Get high byte
 02A3     7A                            mov     a,d             ;Check for invalid month
 02A4     B3                            ora     e
 02A5     CA 0276                       jz      displ1          ;Start over again if invalid
 02A8     CD 0389                       call    pmsg            ;Print the month

 02AB     21 04D1                       lxi     h,tbuff         ;Pointer to temporary storage
 02AE     E5                            push    h               ;Save for printing
 02AF     3A 03F2                       lda     date            ;Convert the date to ascii
 02B2     1F                            rar                     ;Get high digit into low nibble
 02B3     1F                            rar
 02B4     1F                            rar
 02B5     1F                            rar
 02B6     E6 0F                         ani     0fh
```

```
 02B8   C4 0379                   cnz     putlow         ;Don'T print leading zero
 02BB   3A 03F2                   lda     date           ;Get the low digit
 02BE   CD 0379                   call    putlow         ;Stuff it in the buffer
 02C1   3E 2C                     mvi     a,','          ;And the comma and space
 02C3   CD 037D                   call    put
 02C6   3E 20                     mvi     a,' '
 02C8   CD 037D                   call    put

 02CB   3A 03F1                   lda     hour           ;Get the hour
 02CE   FE 13                     cpi     13h            ;Check for AM or PM
 02D0   D4 038E                   cnc     subhr          ;Convert PM from 13-24 into 0-12
 02D3   B7                        ora     a              ;Check for 12 midnight
 02D4   CC 0392                   cz      mak12
 02D7   CD 0370                   call    puthi          ;Put both digits into the buffer
 02DA   3E 3A                     mvi     a,':'          ;Put the colon in the buffer
 02DC   CD 037D                   call    put
 02DF   3A 03F0                   lda     minutes        ;Get the minutes
 02E2   CD 0370                   call    puthi          ;Put both minutes digits in the buffer
 02E5   3E 3A                     mvi     a,':'          ;Put another colon in the buffer
 02E7   CD 037D                   call    put
 02EA   3A 03EF                   lda     seconds        ;Get the seconds
 02ED   CD 0370                   call    puthi          ;Put both second digits in the buffer
 02F0   3E 20                     mvi     a,' '          ;One space into the buffer
 02F2   CD 037D                   call    put
 02F5   3A 03F1                   lda     hour           ;Check hours for AM or PM
 02F8   FE 12                     cpi     12h
 02FA   3E 61                     mvi     a,'a'          ;Print 'A' or 'P'
 02FC   DA 0301                   jc      isam
 02FF   3E 70                     mvi     a,'p'
 0301   CD 037D           isam:   call    put            ;Put the 'A' or 'P' in the buffer
 0304   3E 6D                     mvi     a,'m'          ;Put the 'M' in the buffer
 0306   CD 037D                   call    put
 0309   7E                sploop: mov     a,m            ;Get the next char in the buffer
 030A   FE 24                     cpi     '$'            ;Is it the end ?
 030C   CA 0317                   jz      endsp          ;All done
 030F   3E 20                     mvi     a,' '          ;Get a space
 0311   CD 037D                   call    put            ;Put it in the buffer
 0314   C3 0309                   jmp     sploop         ;Finishing padding with spaces

 0317   D1                endsp:  pop     d              ;Recover the Buffer address
 0318   C3 0389                   jmp     pmsg           ;Print the buffer

                          ;***************************************************************
                          ;*                                                             *
                          ;* Readc does the actual clock reading (40 bits) from the      *
                          ;* hardware. If interupts are enabled, then care must be taken *
```

```
                              ;* to assure that this routine is not interupted until it      *
                              ;* completes.                                                  *
                              ;*                                                             *
                              ;***************************************************************

 031B   AF              readc:  xra     a               ;Select group zero
 031C   D3 4F                   out     grpsel
 031E   3E 0C                   mvi     a,rclk          ;Read clock into 40 bit shift register
 0320   CD 0360                 call    setup
 0323   E5                      push    h               ;Save address of clkdata
 0324   EE 08                   xri     clkc1           ;Issue shift command
 0326   CD 0344                 call    clkcmd
 0329   1E 08           rbyte:  mvi     e,8             ;Prep for 8 bits
 032B   23                      inx     h               ;Bump to next address of clock data

 032C   AF              rbit:   xra     a
 032D   D3 4F                   out     grpsel
 032F   DB 4A                   in      clk             ;Read one bit
 0331   1F                      rar                     ;Put bit into carry
 0332   7E                      mov     a,m             ;Get partially assembled byte
 0333   1F                      rar                     ;Shift in the bit just read
 0334   77                      mov     m,a             ;Save partially assembled byte
 0335   E3                      xthl                    ;Get address of clkdata
 0336   7E                      mov     a,m             ;Get clock data
 0337   E3                      xthl                    ;Save address of clock data
 0338   CD 0346                 call    clkstb          ;Strobe the shift register
 033B   1D                      dcr     e               ;All done with this byte ?
 033C   C2 032C                 jnz     rbit            ;Read another bit if not
 033F   15                      dcr     d               ;Completely done ?
 0340   C2 0329                 jnz     rbyte           ;Read another byte if not
 0343   E1                      pop     h               ;Recover address of clkdata
 0344   0E 20           clkcmd: mvi     c,cstb          ;Get clock strobe bit
 0346   F5              clkstb: push    af
 0347   3E 00                   mvi     a,0
 0349   D3 4F                   out     grpsel
 034B   F1                      pop     af
 034C   D3 4A                   out     clk             ;Output strobe low
 034E   CD 0369                 call    delay           ;Wait for chip to see the strobe low
 0351   A9                      xra     c               ;Turn strobe high
 0352   D3 4A                   out     clk             ;Output strobe high
 0354   CD 0369                 call    delay           ;Wait for chip to see the strobe high
 0357   A9                      xra     c               ;Turn strobe low
 0358   D3 4A                   out     clk             ;Output strobe low
 035A   CD 0369                 call    delay
 035D   0E 02                   mvi     c,clkclk        ;Clock clk bit
 035F   C9                      ret
```

```
 0360    16 05              setup:  mvi     d,5             ;Count of bytes to read
 0362    21 03EE                    lxi     h,clkdata       ;Address of clock data
 0365    B6                         ora     m               ;Get current bit state
 0366    C3 0344                    jmp     clkcmd          ;Issue the command

 0369    06 01              delay:  mvi     b,1             ;Time delay
 036B    05                 delay1: dcr     b
 036C    C2 036B                    jnz     delay1
 036F    C9                         ret

                            ;*****************************************************************
                            ;*                                                               *
                            ;* Puthi puts the high and low nibbles of the bcd number in      *
                            ;* the a reg in the temporary buffer.                            *
                            ;*                                                               *
                            ;*****************************************************************

 0370    F5                 puthi:  push    psw             ;Save low nibble
 0371    1F                         rar                     ;Put high nibble into low nibble
 0372    1F                         rar
 0373    1F                         rar
 0374    1F                         rar
 0375    CD 0379                    call    putlow          ;Print the low nibble of a reg
 0378    F1                         pop     psw             ;Recover the low nibble
 0379    E6 0F              putlow: ani     0fh             ;Strip off irrelevent bits
 037B    C6 30                      adi     '0'             ;Form Ascii character
 037D    77                 put:    mov     m,a             ;Put char in buffer
 037E    23                         inx     h               ;Bump buffer pointer
 037F    C9                         ret

                            ;*****************************************************************
                            ;*                                                               *
                            ;* Exit is the standard error message for invalid command.       *
                            ;*                                                               *
                            ;*****************************************************************

 0380    11 04F9            exit:   lxi     d,badtmsg
 0383    CD 0389                    call    pmsg
 0386    C3 0000                    jmp     wboot

                            ;*****************************************************************
                            ;*                                                               *
                            ;* Pmsg is the CP/M print string function.                       *
                            ;*                                                               *
                            ;*****************************************************************
```

```
 0389     0E 09                  pmsg:   mvi     c,pstr
 038B     C3 0005                        jmp     bdos

 038E     C6 88                  subhr:  adi     88h             ;Subhr adjusts the BCD number to
 0390     27                             daa                     ;       be between 1 and 12
 0391     C9                             ret

 0392     3E 12                  mak12:  mvi     a,12h
 0394     C9                             ret

 0395     3A 03F1                uphrs:  lda     hour
 0398     FE 12                          cpi     12h
 039A     C8                             rz
 039B     C6 12                          adi     12h
 039D     32 03F1                        sta     hour
 03A0     C9                             ret

 03A1     3A 03F1                dwnhrs: lda     hour
 03A4     FE 12                          cpi     12h
 03A6     C0                             rnz
 03A7     AF                             xra     a
 03A8     32 03F1                        sta     hour
 03AB     C9                             ret

 03AC     CD 03D0                skipc:  call    scan            ;Get next char
 03AF     C8                             rz                      ;Return if no more chars
 03B0     FE 20                          cpi     ' '             ;Check for space
 03B2     C2 03AC                        jnz     skipc           ;Continue if not
 03B5     C9                             ret

 03B6     CD 03D0                skipb:  call    scan            ;Get next char
 03B9     C8                             rz                      ;Return if no characters left
 03BA     FE 20                          cpi     ' '             ;Is it a space
 03BC     CA 03B6                        jz      skipb           ;Skip it
 03BF     E5                     unscan: push    h               ;Save HL
 03C0     2A 0540                        lhld    scanpnt         ;Get command scan pointer
 03C3     2B                             dcx     h               ;Back it up
 03C4     22 0540                        shld    scanpnt         ;Save updated char
 03C7     3A 0080                        lda     clen            ;Update length
 03CA     3C                             inr     a
 03CB     32 0080                        sta     clen            ;Save updated length
 03CE     E1                             pop     h               ;Restore HL
 03CF     C9                             ret

 03D0     3A 0080                scan:   lda     clen            ;Check if anything left
```

```
 03D3     A7                        ana     a
 03D4     C8                        rz                      ;Return with Z set if no more
 03D5     3D                        dcr     a               ;Update length
 03D6     32 0080                   sta     clen
 03D9     E5                        push    h               ;Save HL
 03DA     2A 0540                   lhld    scanpnt         ;Get command pointer
 03DD     7E                        mov     a,m
 03DE     23                        inx     h               ;Update command pointer
 03DF     22 0540                   shld    scanpnt
 03E2     E1                        pop     h
 03E3     B7                        ora     a               ;Clear Z flag
 03E4     C9                        ret

 03E5                       toupper:
 03E5     FE 61                     cpi     'a'             ;Is it lower case ?
 03E7     D8                        rc
 03E8     FE 7B                     cpi     'z'+1
 03EA     D0                        rnc
 03EB     D6 20                     sui     ' '
 03ED     C9                        ret

                            ;*****************************************************************
                            ;*                                                               *
                            ;* The following are data used within the program.               *
                            ;*                                                               *
                            ;*****************************************************************

 03EE                       clkdata:
 03EE     00                        db      0               ;Current state of clk port
 03EF                       seconds:
 03EF     00                        db      0               ;Seconds read
 03F0                       minutes:
 03F0     00                        db      0               ;Minutes read
 03F1     00                hour:   db      0               ;Hours read
 03F2     00                date:   db      0               ;Date read
 03F3     00                wekmon: db      0               ;Week day and month read

                            ;*****************************************************************
                            ;*                                                               *
                            ;* Days is an array of pointers to strings, used to print the    *
                            ;* english version of the day of the week.                       *
                            ;*                                                               *
                            ;*****************************************************************

 03F4     0404              days:   dw      sun
 03F6     040D                      dw      mon
```

```
 03F8     0416                          dw      tue
 03FA     0420                          dw      wed
 03FC     042C                          dw      thu
 03FE     0437                          dw      fri
 0400     0440                          dw      sat
 0402     0000                          dw      0                   ;Illegal day

 0404     53 75 6E 64           sun:    db      'Sunday, $'
 0408     61 79 2C 20
 040C     24
 040D     4D 6F 6E 64           mon:    db      'Monday, $'
 0411     61 79 2C 20
 0415     24
 0416     54 75 65 73           tue:    db      'Tuesday, $'
 041A     64 61 79 2C
 041E     20 24
 0420     57 65 64 6E           wed:    db      'Wednesday, $'
 0424     65 73 64 61
 0428     79 2C 20 24
 042C     54 68 75 72           thu:    db      'Thursday, $'
 0430     73 64 61 79
 0434     2C 20 24
 0437     46 72 69 64           fri:    db      'Friday, $'
 043B     61 79 2C 20
 043F     24
 0440     53 61 74 75           sat:    db      'Saturday, $'
 0444     72 64 61 79
 0448     2C 20 24

                                ;*****************************************************************
                                ;*                                                               *
                                ;* Months is an array of pointers to strings, used to print the  *
                                ;* english version of the month of the year.                     *
                                ;*                                                               *
                                ;*****************************************************************

 044B     046B                  months: dw      jan
 044D     0474                          dw      feb
 044F     047D                          dw      mar
 0451     0484                          dw      apr
 0453     048B                          dw      may
 0455     0490                          dw      jun
 0457     0496                          dw      jul
 0459     049C                          dw      aug
 045B     04A4                          dw      sep
 045D     04AF                          dw      oct
```

```
045F    04B8                            dw      nov
0461    04C2                            dw      dec
0463    0000 0000                       dw      0,0,0,0                 ;Illegal months
0467    0000 0000

046B    4A 61 6E 75             jan:    db      'January $'
046F    61 72 79 20
0473    24
0474    46 65 62 75             feb:    db      'Febuary $'
0478    61 72 79 20
047C    24
047D    4D 61 72 63             mar:    db      'March $'
0481    68 20 24
0484    41 70 72 69             apr:    db      'April $'
0488    6C 20 24
048B    4D 61 79 20             may:    db      'May $'
048F    24
0490    4A 75 6E 65             jun:    db      'June $'
0494    20 24
0496    4A 75 6C 79             jul:    db      'July $'
049A    20 24
049C    41 75 67 75             aug:    db      'August $'
04A0    73 74 20 24
04A4    53 65 70 74             sep:    db      'September $'
04A8    65 6D 62 65
04AC    72 20 24
04AF    4F 63 74 6F             oct:    db      'October $'
04B3    62 65 72 20
04B7    24
04B8    4E 6F 76 65             nov:    db      'November $'
04BC    6D 62 65 72
04C0    20 24
04C2    44 65 63 65             dec:    db      'December $'
04C6    6D 62 65 72
04CA    20 24
04CC    0D 24                   acrmsg: db      acr,'$'
04CE    0D 0A 24                acralf: db      acr,alf,'$'

                                ;*****************************************************************
                                ;*                                                               *
                                ;* Tbuff is used to prepare the day of the month, hours, minutes,*
                                ;* and seconds prior to printing.                                *
                                ;*                                                               *
                                ;*****************************************************************

04D1    30 30 2C 20             tbuff:  db      '00, 00:00:00 am                $'
```

```
 04D5    30 30 3A 30
 04D9    30 3A 30 30
 04DD    20 61 6D 20
 04E1    20 20 20 20
 04E5    20 20 20 20
 04E9    20 20 20 20
 04ED    20 20 20 20
 04F1    20 20 20 20
 04F5    20 20 20 24

 04F9                        badtmsg:
 04F9    0D 0A                        db      acr,alf
 04FB    49 6E 76 61                  db      'Invalid Time specified.$'
 04FF    6C 69 64 20
 0503    54 69 6D 65
 0507    20 73 70 65
 050B    63 69 66 69
 050F    65 64 2E 24

 0513                        waitmsg:
 0513    0D 0A                        db      acr,alf
 0515    50 72 65 73                  db      'Press return to set the time: $'
 0519    73 20 72 65
 051D    74 75 72 6E
 0521    20 74 6F 20
 0525    73 65 74 20
 0529    74 68 65 20
 052D    74 69 6D 65
 0531    3A 20 24

 0534    0A 0A               ibuff:   db      10,10
 0536                                 ds      10

 0540                        scanpnt:
 0540    0081                         dw      cbuff
 0542    00                  mcnt:    db      0

                                      end
```

Macros:

Symbols:

ACR	000D	ACRALF	04CE	ACRMSG	04CC	ALF	000A
APR	0484	AUG	049C	BADTMS	04F9	BASE	0048
BCD2	01CE	BDOS	0005	CBUFF	0081	CLEN	0080
CLK	004A	CLKC1	0008	CLKCLK	0002	CLKCMD	0344
CLKDAT	03EE	CLKSTB	0346	CONST	000B	CSTB	0020
DATE	03F2	DAYS	03F4	DEC	04C2	DELAY	0369
DELAY1	036B	DIGIT	020E	DISPL1	0276	DISPLA	0261
DWNHRS	03A1	ENDSP	0317	EXIT	0380	FEB	0474
FRI	0437	GRPSEL	004F	HOUR	03F1	IBUFF	0534
ISAM	0301	JAN	046B	JUL	0496	JUN	0490
MAK12	0392	MAR	047D	MATCH3	0218	MAY	048B
MCNT	0542	MINUTE	03F0	MON	040D	MONTHS	044B
MTCHMO	022D	NOAP	017B	NOMATC	0255	NOV	04B8
OCT	04AF	OKD	020A	OKDAY	027E	PMSG	0389
PSTR	0009	PUT	037D	PUTHI	0370	PUTLOW	0379
RBIT	032C	RBYTE	0329	RCLK	000C	READC	031B
READCO	000A	REGHLD	0000	REV	000A	SAT	0440
SCAN	03D0	SCANPN	0540	SECOND	03EF	SEP	04A4
SETT	010A	SETUP	0360	SHFT	0004	SKIPB	03B6
SKIPC	03AC	SPLOOP	0309	START	0100	SUBHR	038E
SUN	0404	TBUFF	04D1	THU	042C	TOUPPE	03E5
TP64	0010	TUE	0416	UNSCAN	03BF	UPHRS	0395
WAITMS	0513	WBIT	01AE	WBOOT	0000	WBYTE	01AB
WCLK	0008	WED	0420	WEKMON	03F3	WRITEC	01A2

No Fatal error(s)

PARTS LIST

3	8-pin low profile sockets
19	14-pin low profile sockets
6	16-pin low profile sockets
10	20-pin low profile sockets
1	28-pin low profile sockets
3	40-pin low profile sockets

2	3/4 inch wide heat sink
4	6-32 hex machine nuts
4	6-32 x 3/8 machine screws
1	10-pin power connector
1	2-pin reset connector
1	50-pin hooded dual inline connector
3	26-pin right angle P.C. mount (subminiature D connectors)
1	15-pin right angle P.C. mount (subminiature D connectors)
14	100-pin S-100 edge connectors
2	8 position DIP switch arrays
2	2 position .025 square connector post array
1	3 position .025 square connector post array

1	3.3 Ohm 1/4 watt resistor
2	75 Ohm 1/4 watt resistors
2	130 Ohm 1/4 watt resistors
2	220 Ohm 1/4 watt resistors
2	330 Ohm 1/4 watt resistors
1	360 Ohm 1/4 watt resistor
1	390 Ohm 1/4 watt resistor
2	1.5k Ohm 1/4 watt resistors
8	3.3k Ohm 1/4 watt resistors
4	4.7k Ohm 1/4 watt resistors
1	10k Ohm 1/4 watt resistor
3	100k Ohm 1/4 watt resistor
7	100k Ohm 1/8 watt resistor

12	10-pin 180 Ohm SIP resistor array
2	8-pin 3.3k Ohm SIP resistor array

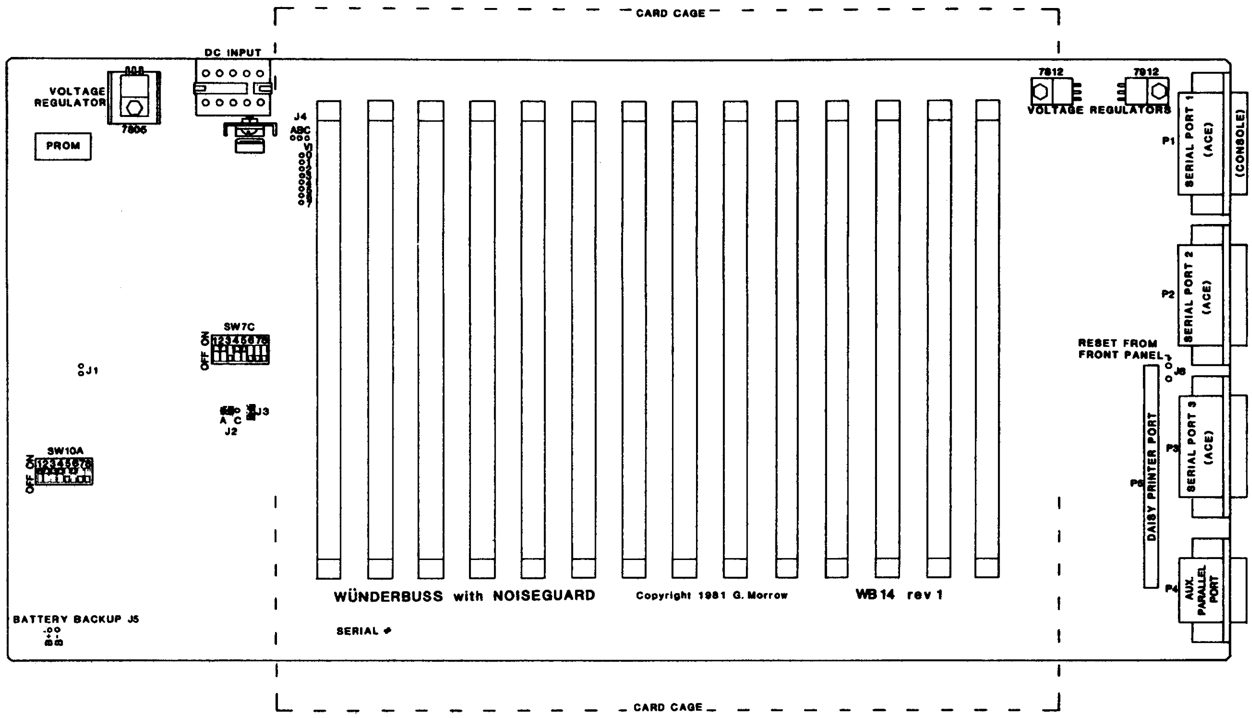
2	20 pf dipped mica capacitor
1	47 pf dipped mica capacitor
1	56 pf dipped mica capacitor
1	112 pf dipped mica capacitor
7	dipped tantalum capacitor - 20V
4	39 ufd axial tantulum 10V capacitor
22	disk ceramic by-pass capacitor

1	32.768 KHz clock crystal
1	18.432 MHz HU/18 crystal

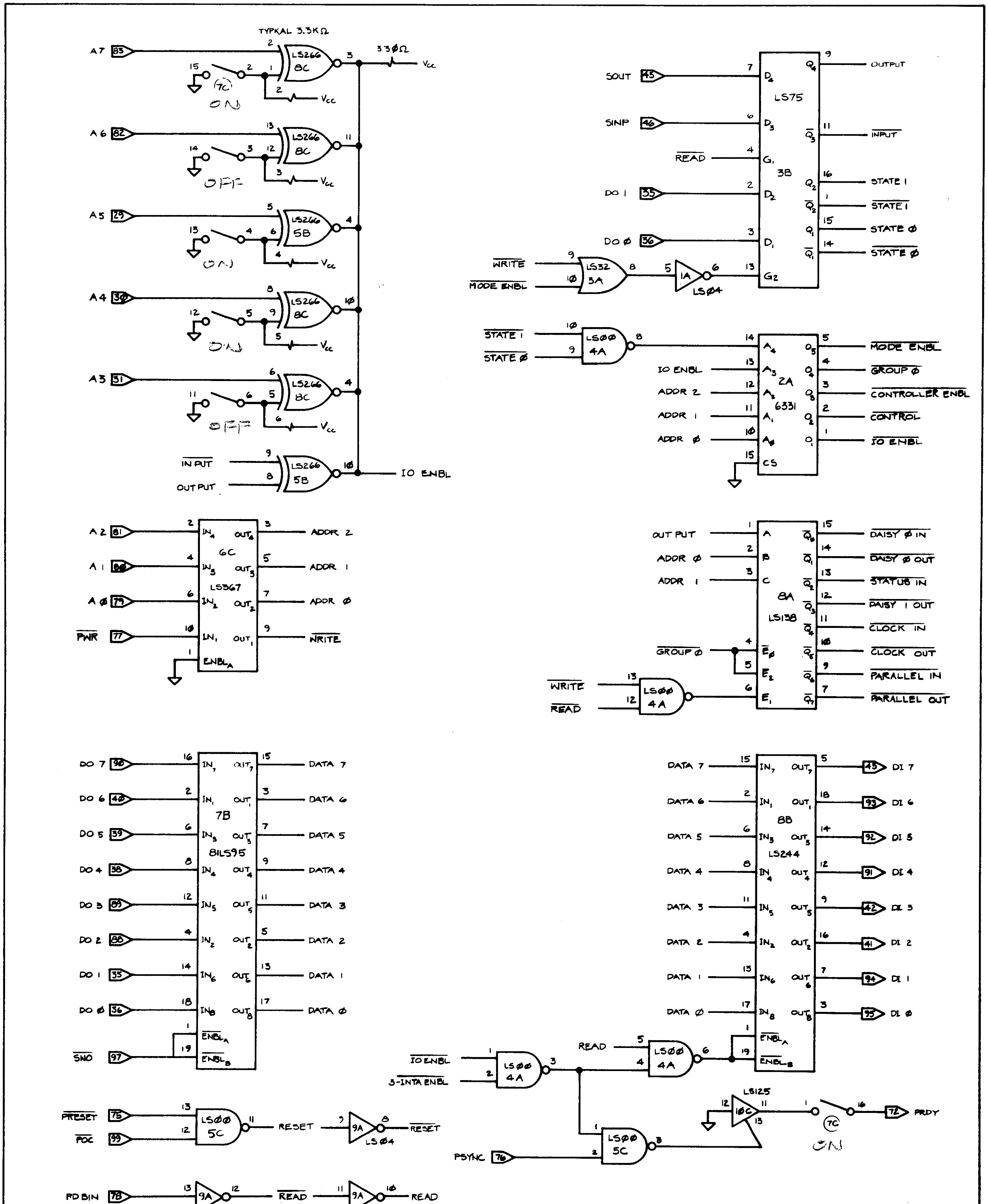
PARTS LIST CONT.

1	IN914 signal diode
1	IN5221 2.6V zener diode
2	2N3904 NPN transistor
3	2N3906 PNP transistor
1	TIP29/D44C4 NPN transistor
1	TIP30/D45C4 PNP transistor
1	7805 positive 5V regulator
1	7812 positive 12V regulator
1	7912 negative 12V regulator
1	LM201 high speed operational amplifier
4	LM1458 dual operational amplifier
3	1489 quad RS232 receiver/buffer
2	74LS00 quad 2-input NAND gate IC
4	74LS04 hex inverter IC
1	7406 hex open collector inverter/buffer IC
1	74LS32 quad 2-input OR gate IC
2	74LS74 dual D-type flip-flop IC
1	74LS75 quad dual rail transparent latch IC
2	74LS90 decade counter IC
1	74LS125 quad tri-state buffer IC
1	74LS138 1 of 8 decoder ICs
1	74LS174 hex latch with clear IC
2	74LS244 octal tri-state buffer IC
2	74LS266 quad 2-input EXNOR gate IC
1	74LS273 octal latch with clear IC
1	74LS367 hex tri-state buffer IC
3	74LS373 octal transparent latch/buffer IC
2	8ILS95 octal tri-state buffer IC
2	8ILS96 octal inverting tri-state buffer IC
1	8259A programmable interrupt controller IC
3	8250 programmable UART with baud rate generator IC
1	1990 programmable real-time clock IC
1	7611 32 x 8 bi-polar PROM

COMPONENT LAYOUT/SCHEMATIC

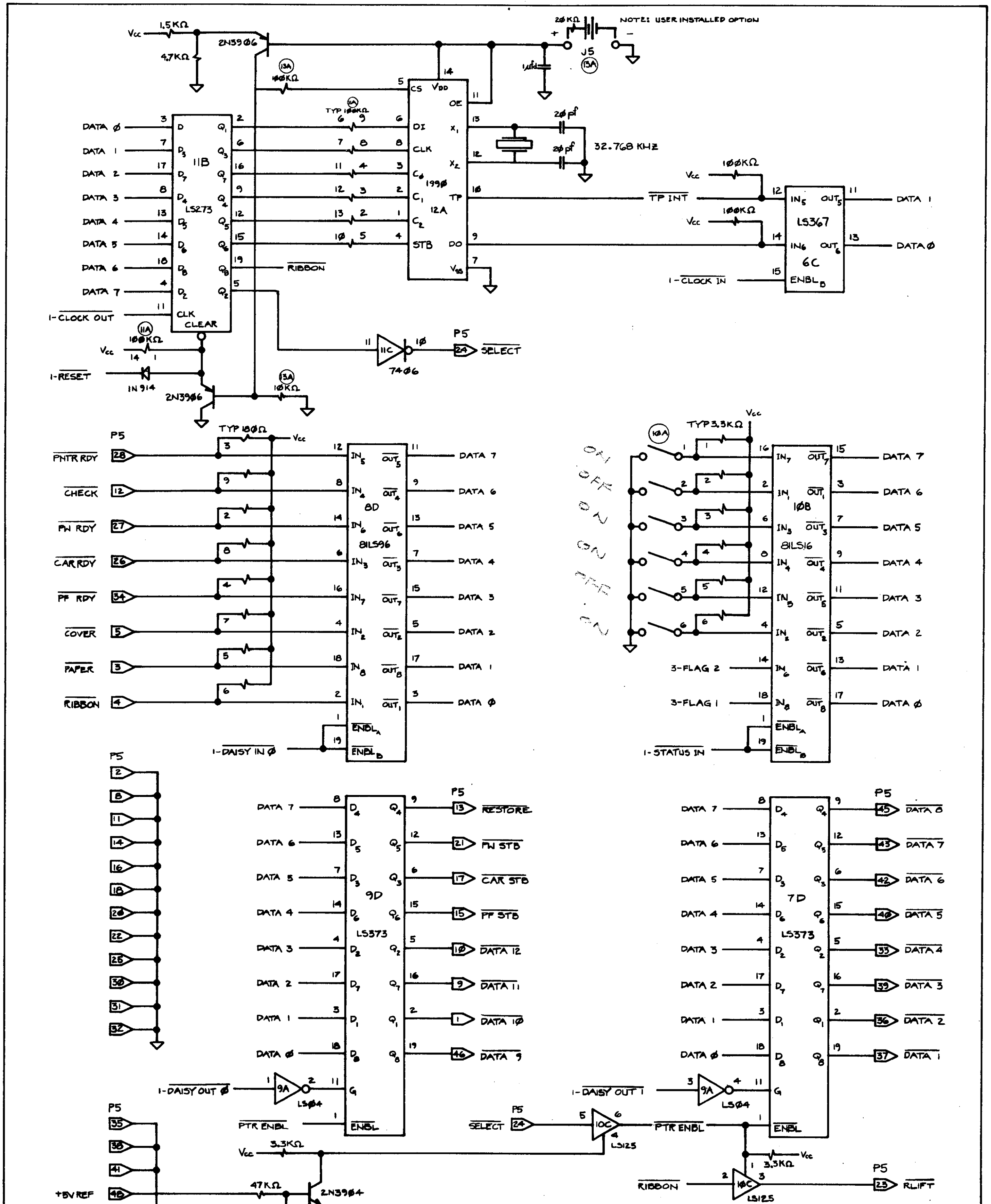


Wunderbuss Component Layout



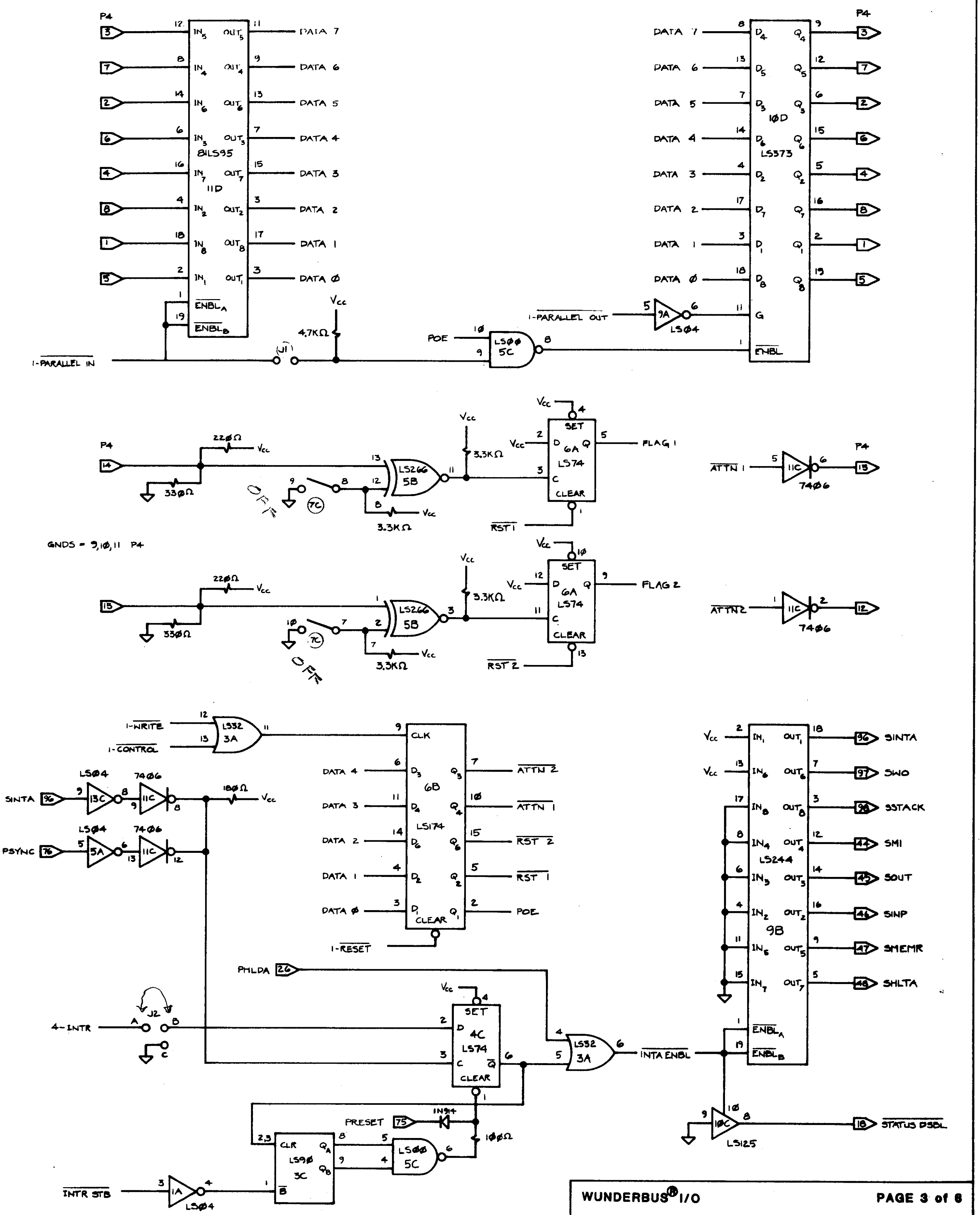
WUNDERBUS® I/O **PAGE 1 of 6**

BUFFERS, BOARD SELECT LOGIC



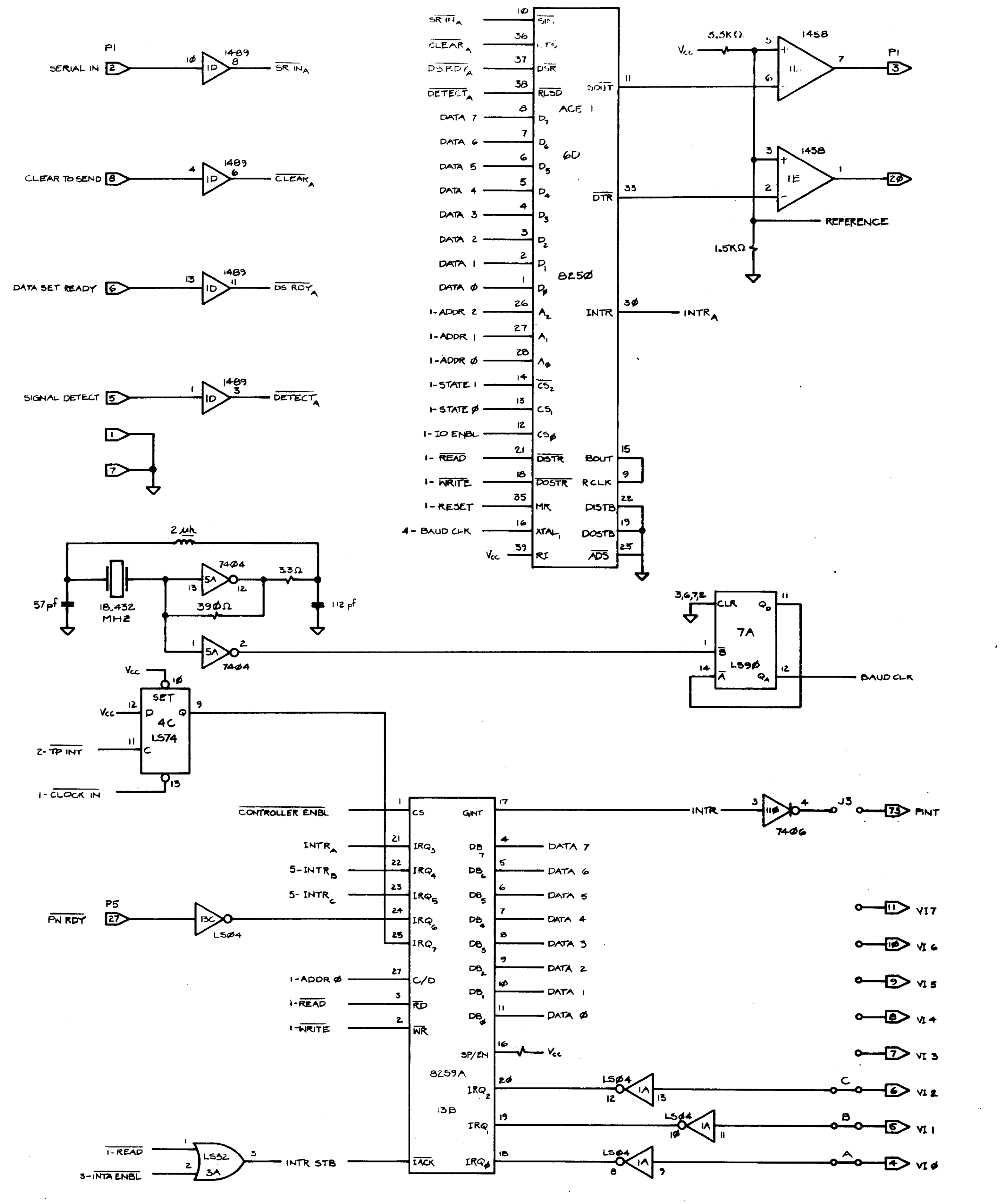
WUNDERBUS® I/O

CLOCK, STATUS, & DAISY PORT™



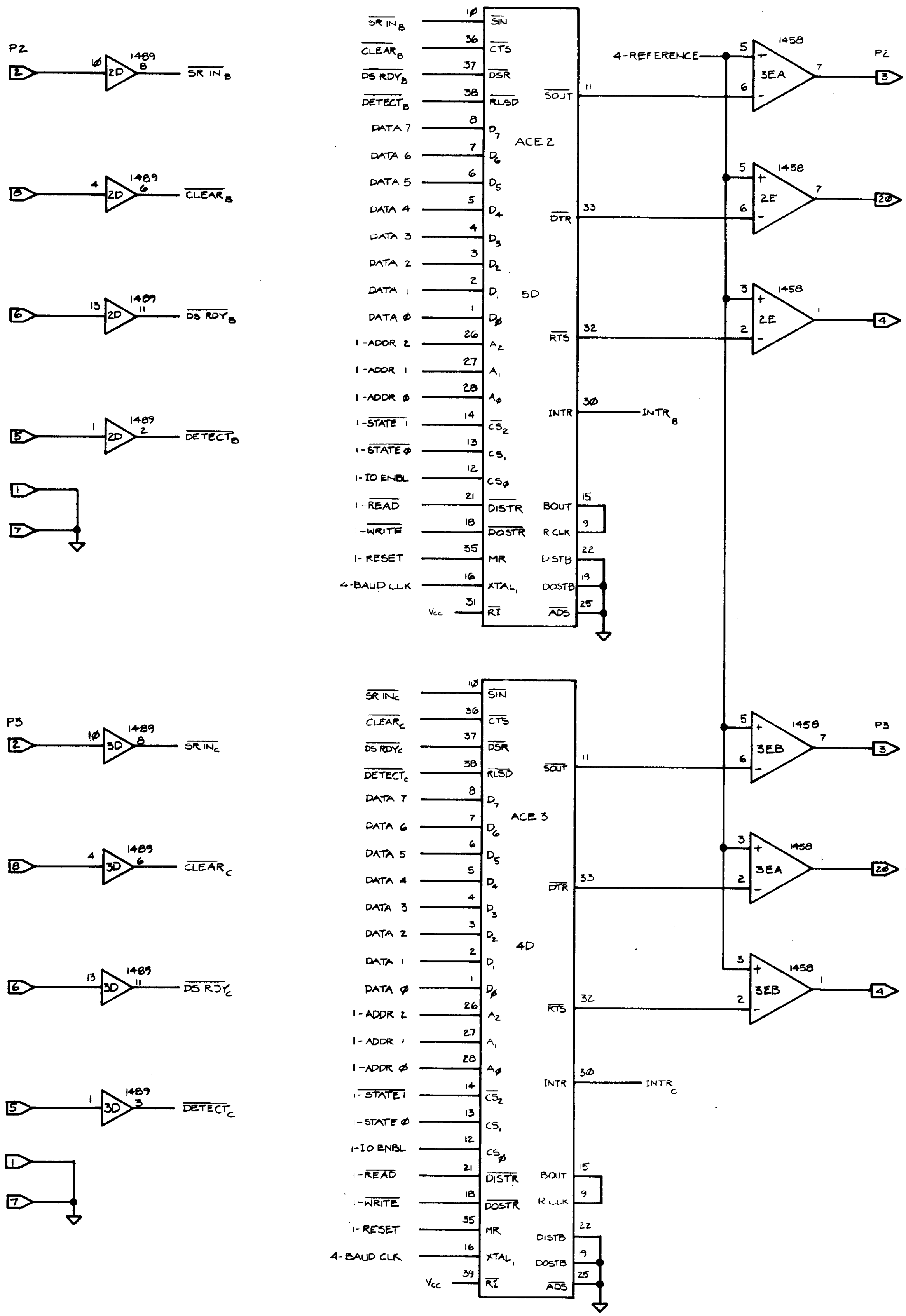
WUNDERBUS® I/O **PAGE 3 of 8**

PARALLEL PORT, INTERRUPT LOGIC



WUNDERBUS® I/O **PAGE 4 of 6**

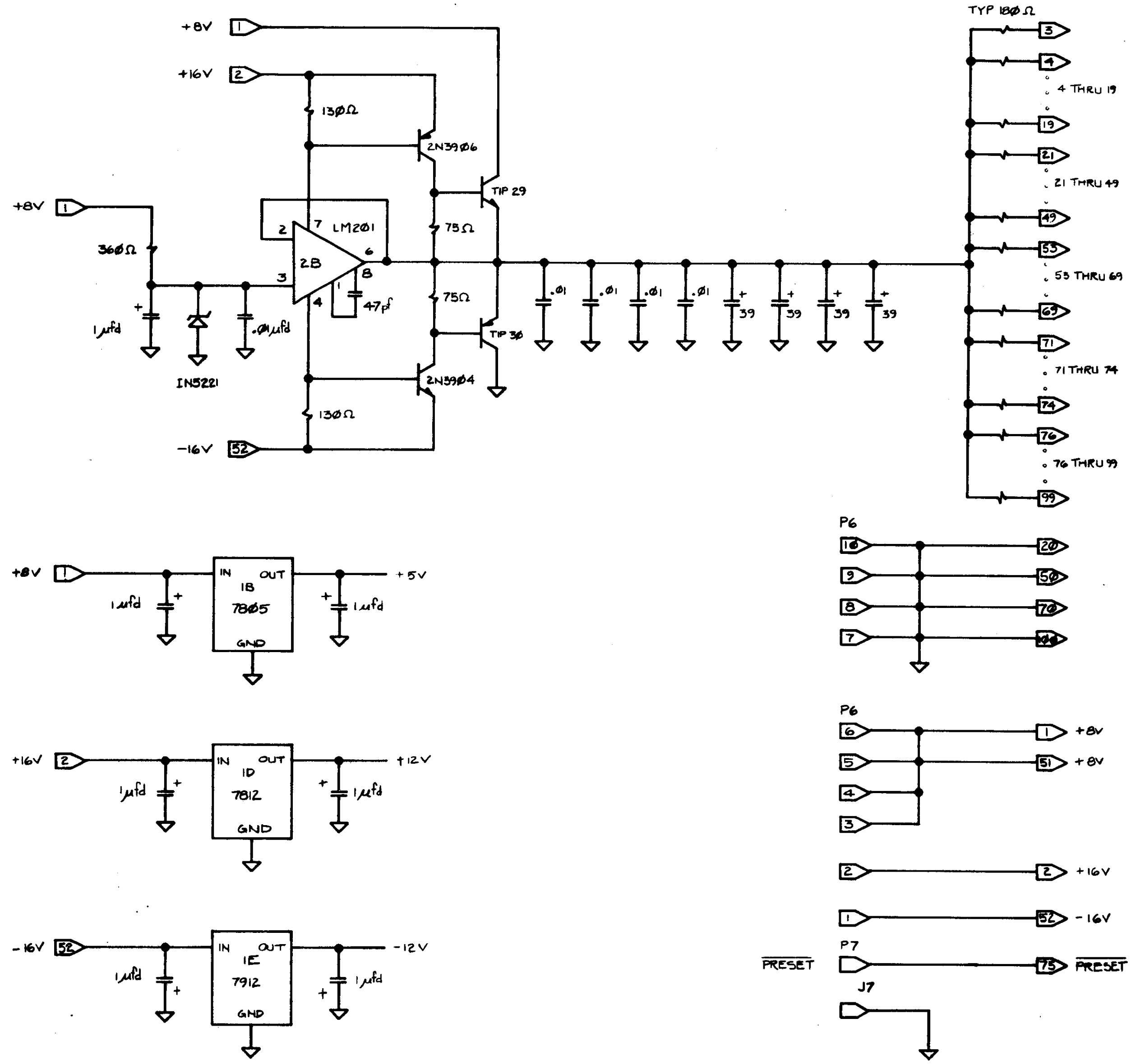
SERIAL CHANNEL A & INTERRUPT CONTROLLER



WUNDERBUS® I/O

SERIAL CHANNELS B & C

PAGE 5 of 6



WUNDERBUS® I/O — **PAGE 6 of 6**

POWER SUPPLIES & BUS TERMINATION

Subject Index