## *PC Architectures*

## *History of the PC: PC, PC/XT & PC/AT*

# INTRODUCTION

Today's PCs are backward compatible with the original IBM PC and PC/AT designs. This has placed some constraints on the designs of new PCs but it has paid off as many customers have stayed loyal to PC products for these very backward compatibility reasons.

To understand this legacy aspect of today's PC designs we will look at the early PC and PC/AT products. Also we find that many I/O devices used today are functionally the same as in the early products. This holds true for the Memory Map and the I/O Map.

EARLY PCs

IBM PC: i8088 based
4.77 MHz
8-bit PC-bus
No hard disk – one or two floppies

IBM PC/XT: As above, but included a 10MB Hard Disk, and a better floppy.

Many PC and PC/XT Clones used the i8086 for better performance

IBM PC/AT: i80286 based
6 MHz (first release)
16-bit AT-bus (which later became the **ISA** Standard)

## PC Brief History

*Family One Systems*

| Year | Model | CPU | Comment |
|---|---|---|---|
| 1981 | IBM PC | I8088 | No Hard Disk |
| 1983 | IBM PC/XT | I8088 | 10Mbyte Hard Disk |
| 1984 | IBM PC/AT | I80286 | 20MB Hard Disk |

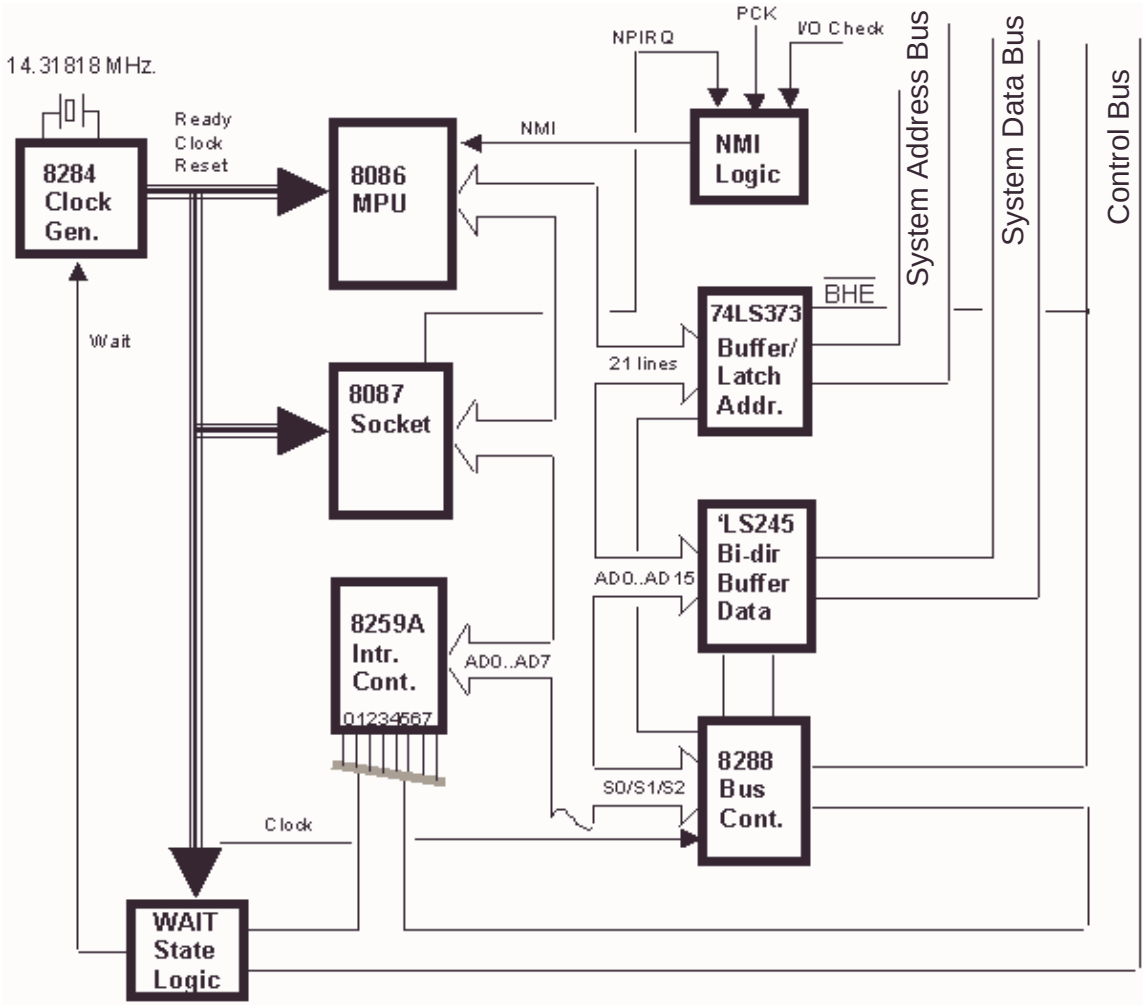
Industry builds IBM Compatible Computers

*Family Two Systems*

| Year | Model | CPU | Bus | Video |
|---|---|---|---|---|
| 1987 | PS/2 | I8086 | 8-bit ISA | MCGA |
| 1987 | PS/2 50 | I80286 | MCA/16 | VGA |
| 1990 | PS/1 286 | I80286 | 16-bit ISA | VGA |
| 1991 | PS/1 SX | I386DX | 16-bit ISA | VGA |
| 1991 | PS/2 55SX | I386SX | MCA/16 | VGA |
| 1991 | PS/2 70/386 | I386DX | MCA/32 | VGA |
| 1991 | PS/2 70/486 | I486DX | MCA/32 | VGA |
| 1993 | PS/2 E | I486SLC | 16-bit ISA | VGA |

The computer industry ignored MCA and developed EISA. First EISA bus products appeared in 1989.

# EARLY PC/XT ARCHITECTURE




The original PC was based on the i8088 processor. The PC/XT was an improved design employing an i8086 processor and a Fixed Disk drive feature (10MBytes). The diagram shows a standard i8086 MAXIMUM mode processor configuration. An Arithmetic Co- processor (i8087) socket is included as well as an 8 level interrupt controller, the 8259A. The NMI logic responds to three sources of NMI interrupts: I/OCHCK indicates that an adapter card has a fault; NPIRQ is an 'Numeric Processor' interrupt from the 8087; PCK is a Parity check error interrupt.

The i8086 bus (Address, data, status, control) in the PC/XT design is referred to as the **LOCAL BUS**, as shown. The de-multiplexed Bus is referred to as the **SYSTEM BUS** in the PC/XT design and consists of the **System Address Bus** A0..A19, the **System Data Bus** D0..D15 and the Control Bus.

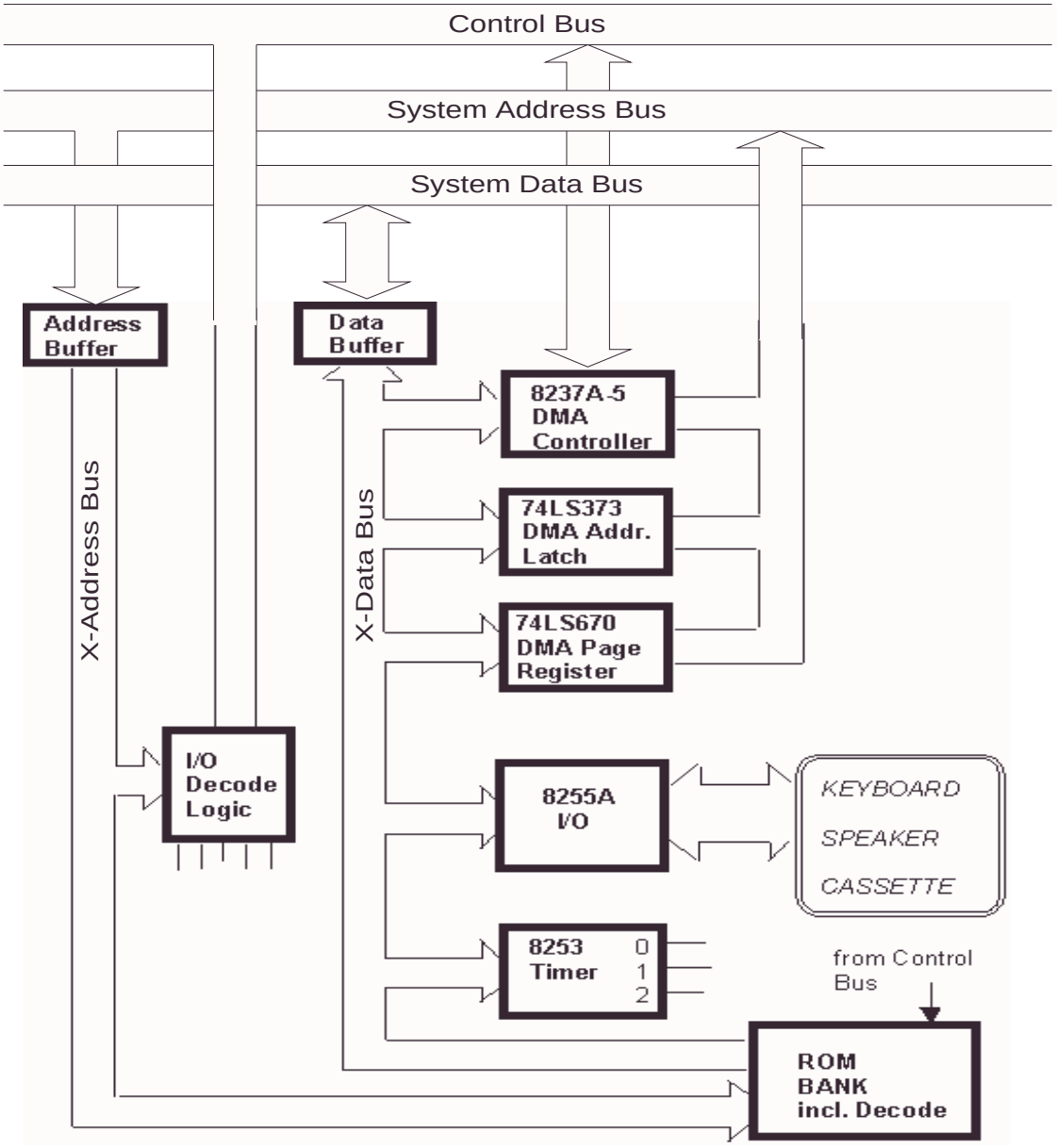
The i8086 processor has limited current drive capability. The Address latches and data bus transceivers, as well as implementing the Local bus demultiplexing function, also provide buffering of the signals to provide a higher current drive capability.

## EARLY PC Support Circuitry

The support circuits on the PC/XT system board (motherboard) include an 8237A DMA controller with associated latches and register, an i8255A parallel I/O device to interface, and devices such as the keyboard, the speaker and cassette player, an i8253 timer/counter and a ROM memory.

To reduce loading on the System bus, which is used to drive the RAM memory buffers and the

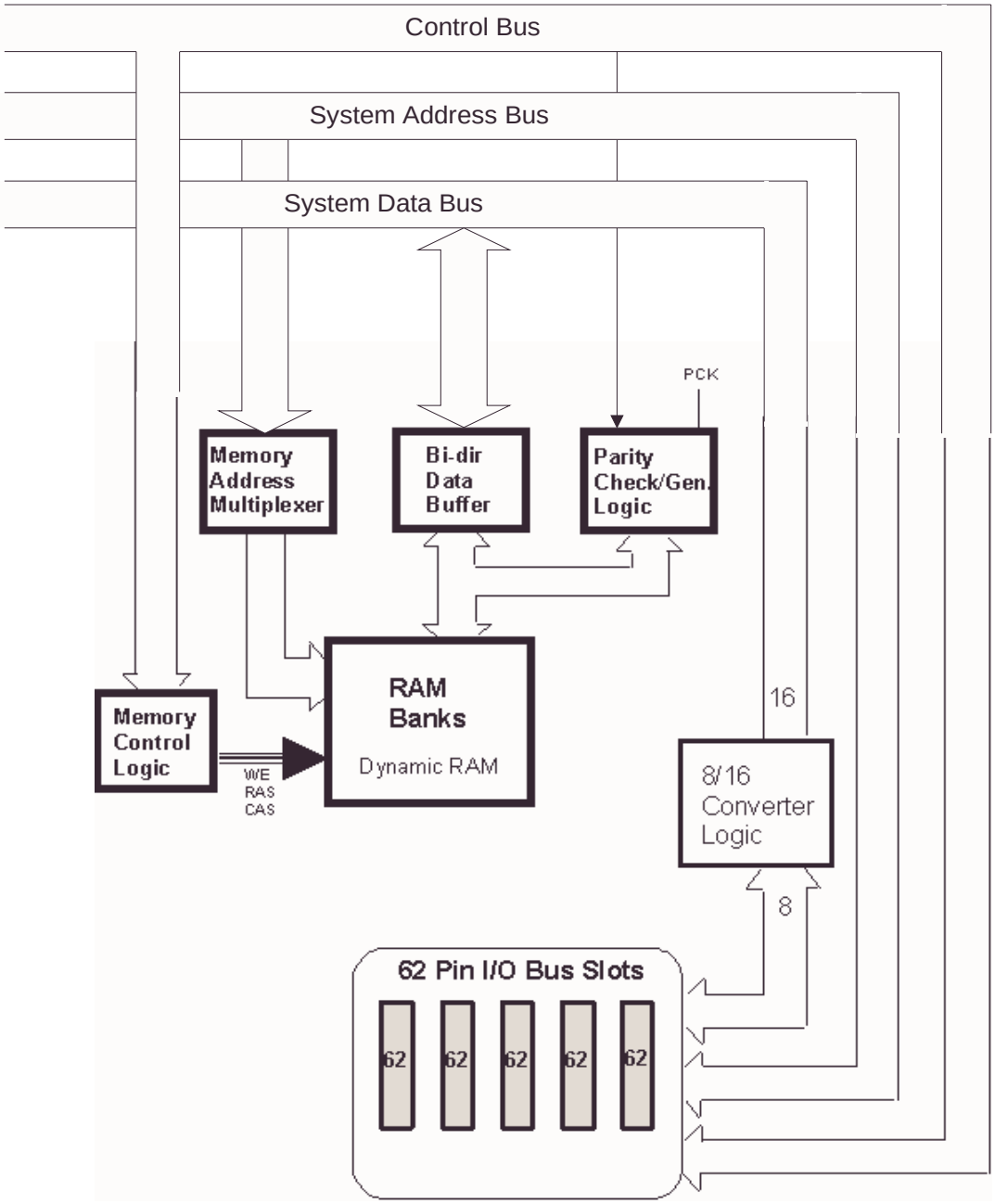
Add-in cards (Adapter cards) in the bus slots, the bus driving these support chips is again buffered. The buffered bus is known as the External, or **X-bus**. There is an **X-address bus** and an **X-data bus**, as shown below.




The PC/XT RAM memory uses Dynamic RAM devices (DRAMs). The system address bus is multiplexed to provide the correct RAS and CAS addressing with some memory control logic providing the necessary control signals (WE, RAS CAS). This multiplexed bus is referred to as **Memory Address Bus**. The memory data bus is also buffered as shown and is referred to as the **Memory Data Bus**. Note, the byte level parity check logic associated with the RAM memory.

To support Adapter cards a number of 62-Pin bus slot connectors are provided. The System Bus is connected to these connectors. Since the 62-Pin connector supports an 8-bit data bus only, an 8-/16-bit converter is required to convert from 16 bits to 8 bits when writing (or OUT) to the connector and logic to convert from 8-bits to 16-bits when reading (or IN) from the connector.

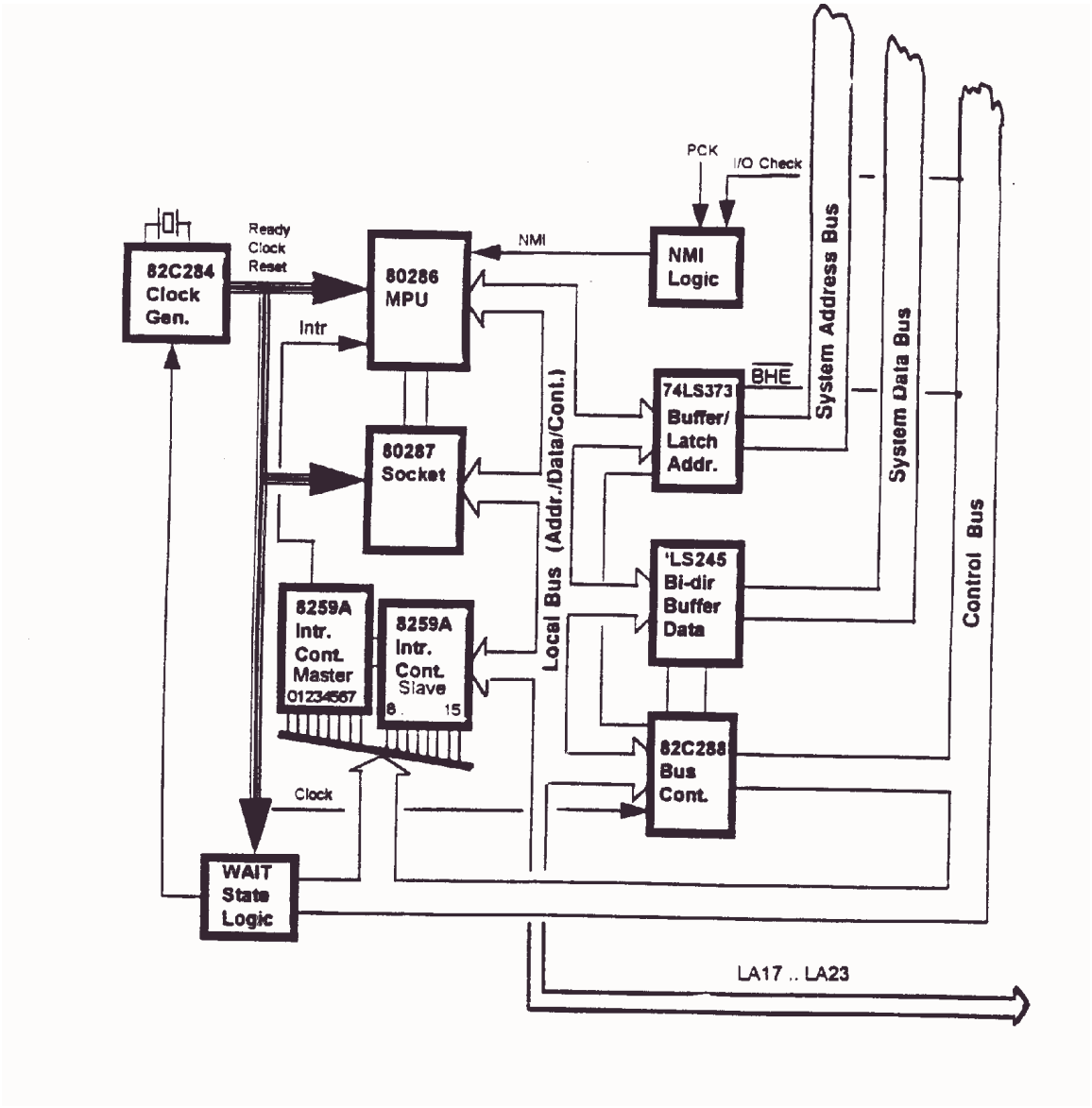
**EARLY PC/XT RAM Memory and I/O Bus Slots**



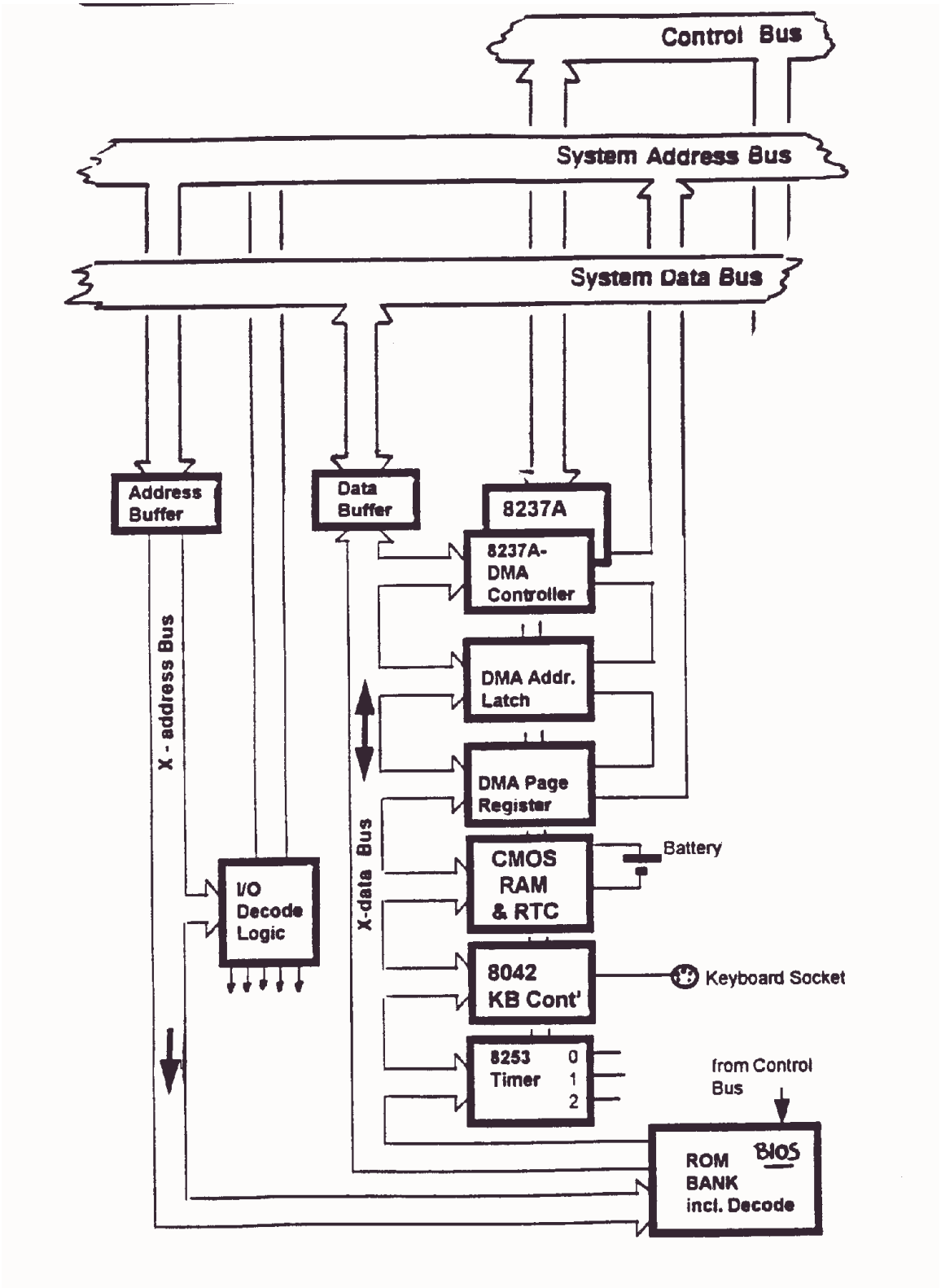

## The PC/AT

The original PC/AT was based on the 16-bit i80286 microprocessor. In the early PC and PC/AT designs peripheral devices such as the Floppy Disk Controller, Hard Disk Controller, Video Controller, Memory Expansion etc were provided on separate adapter boards, which plugged into the bus slots. With engineering advances these devices became smaller, sometimes integrated into a single chip, and it became possible to include them on the system board (motherboard).

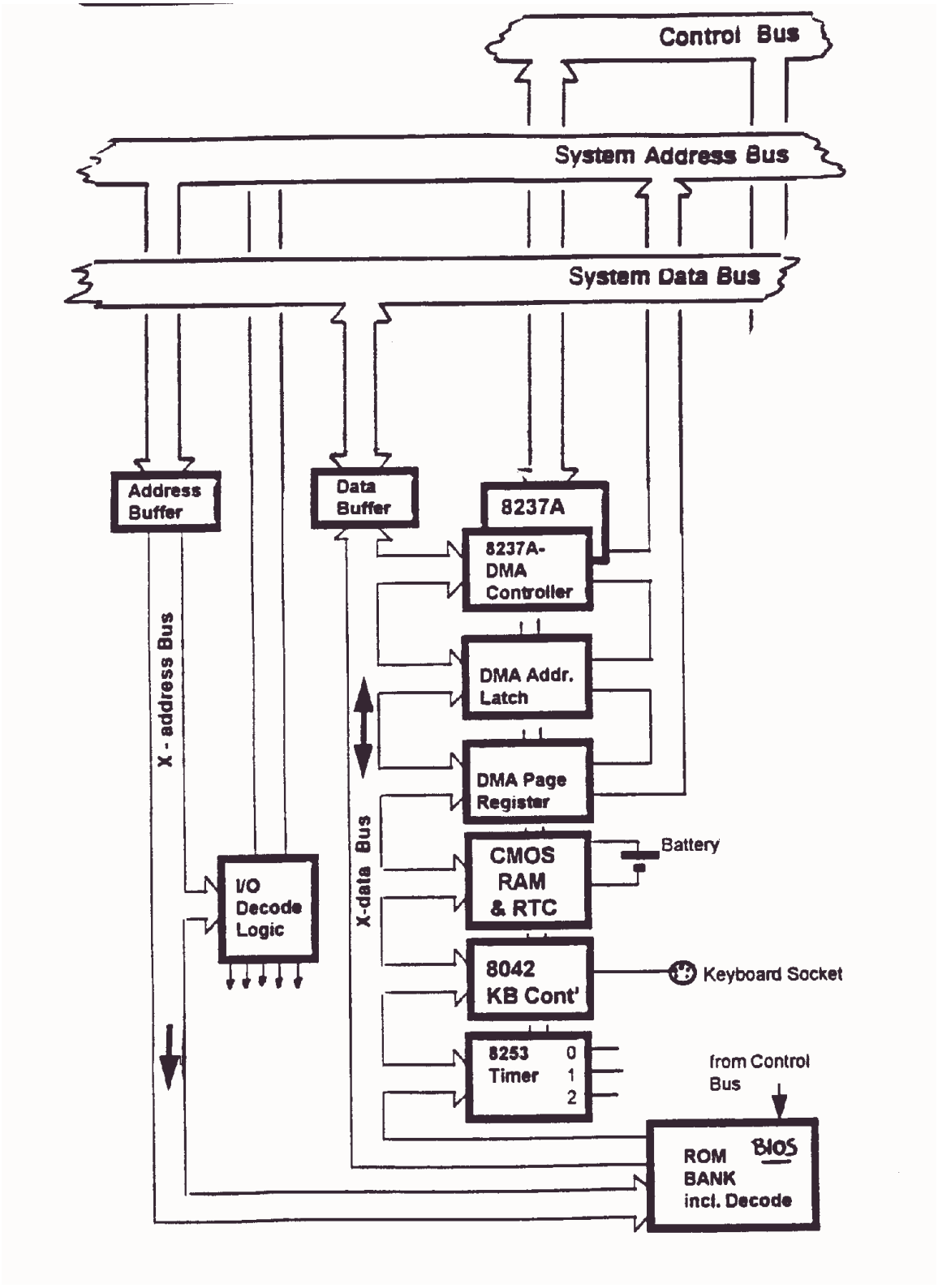
### PC/AT Processor Section


82C284
Clock
Gen.
Ready
Clock
Reset
80286
MPU
NMI
NMI
Logic
PCK
I/O Check
System Address Bus
System Data Bus
Control Bus
Intr
74LS373
Buffer/
Latch
Addr.
BHE
80287
Socket
Local Bus (Addr./Data/Cont.)
'LS245
Bi-dir
Buffer
Data
8259A
Intr.
Cont.
Master
01234567
8259A
Intr.
Cont.
Slave
8 15
82C288
Bus
Cont.
Clock
WAIT
State
Logic
LA17 .. LA23

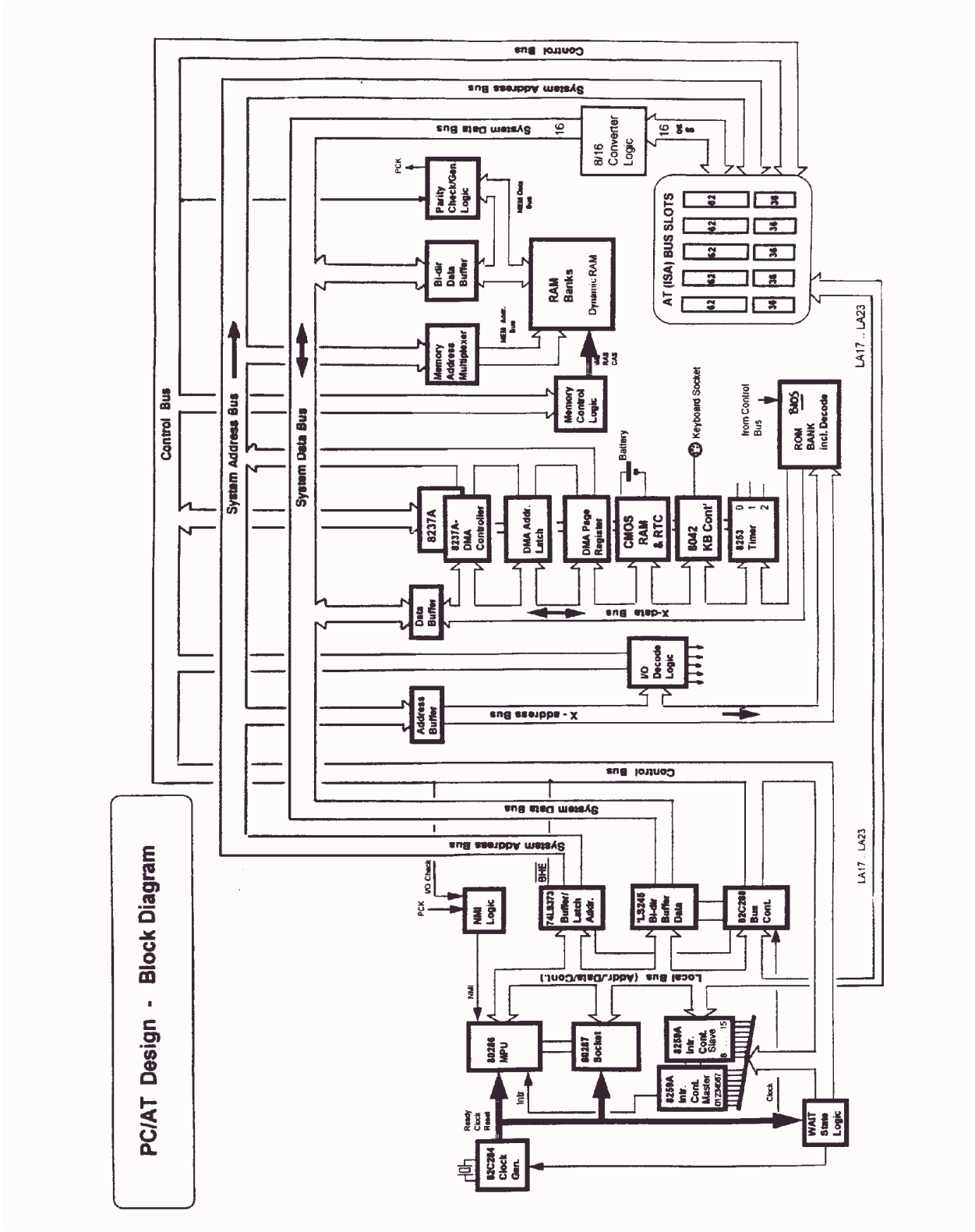
## PC/AT Support Circuitry




## PC/AT RAM Memory and I/O Bus Slots




**PC/AT – Block Diagram**


PC/AT Design - Block Diagram
Control Bus
System Address Bus
System Data Bus
8/16 Converter Logic
AT (ISA) BUS SLOTS
Parity Check/Gen. Logic
Bi-dir Data Buffer
RAM Banks
Dynamic RAM
Memory Address Multiplexer
Memory Control Logic
ROM BANK incl. Decode
Keyboard Socket
from Control Bus
Battery
8237A DMA Controller
DMA Addr. Latch
DMA Page Register
CMOS RAM & RTC
8042 KB Cont'
8253 Timer
Data Buffer
X-data Bus
I/O Decode Logic
Address Buffer
X - address Bus
LA17 .. LA23
I/O Check
NMI Logic
74LS373 Buffer/ Latch Addr.
LS245 Bi-dir Buffer Data
82C288 Bus Cont.
Local Bus (Addr./Data/Cont.)
80286 MPU
80287 Socket
8259A Intr. Cont. Slave
8259A Intr. Cont. Master
WAIT State Logic
82C284 Clock Gen.

# PC/AT BUS STANDARDS

The **AT-Bus** is a 16-bit (data) bus and is a development of the 8-bit **PC-Bus**.

The **AT-Bus** is a de-facto industry standard. Adapter cards, which use the full bus (two connectors), are referred to as '**16-bit adapter cards**'. Cards, which use just one connector (PC bus), are referred to as '**8-bit adapter cards'.**

The **ISA** (Industry Standard Architecture) standard formalised the AT-Bus (16-bit) standard as an industry standard, and this standard is commonly referred to as the '**ISA Bus**' standard. The ISA standardisation is based on the IEEE P996 defined standard.

A more advanced bus, the **EISA** bus (Enhanced Industry Standard Architecture) is a standard, which is upwards compatible to the ISA bus.

**The PC Bus**

The ISA (AT) bus uses two physical connector slots, a 62-pin and a 36-pin connector. The PC bus (8-bit) uses just the 62-pin connector. ISA did not concern itself with the PC bus but the ISA design is upwards compatible with the PC 62 pin connector. All PC signals necessary to interface the processor to external devices are listed below. These signals are connected to a physical bus, which uses a 62-pin connector as shown in figure 'The 62-Pin Bus Connector'.

Note, the EIA committee changed the names of some of the 62-Pin connector signals from the original PC/XT signal names, and did not otherwise modify the 62 Pin slot. E.g.: MEMR and MEMW are named SMEMR and SMEMW, the 20 address lines are named SA0-SA19, the 8 data lines are named SD0-SD7, ALE is named BALE, RDY is named I/O CH RDY. The ISA names are used here.

**PC-BUS SIGNAL NAMES (Using ISA Naming)**

**arrow -> System board drives signal**
**<-> Bi-directional signal**
**<- Add-in board drives the signal**

**SA0 -SA19** -> 20-bit System Address bus (driven by MPU or DMA).

**SD0 -SD7** <-> System 8-bit data bus (bi-directional).

**SMEMR** -> Memory read control signal (active low)

**SMEMW** -> Memory write control signal (active low)

**IOR** -> Read control signal (active low)

**IOW** -> I/O write control signal (active high)

**BALE** -> Address Latch Enable. Indicates that the address bus is valid for the beginning of a bus cycle. Can be used to latch valid addresses.

**I/OCHCK** <- I/O Channel Check. Signals an error condition on the add-in card. This signal is connected to the MPU's NMI line.

**I/OCHRDY** <- I/O Channel Ready. Extends the length of bus cycles for access to slow memory or I/O devices. Circuitry on the add-in card may force the I/O CH RDY line low to cause MPU or DMA 'wait states'.

**SRDY** <- Synchronous Ready (0WS is AT nomenclature). Adapter card indicates that it can operate without 'wait states'.

**REFRESH** -> Indicates a memory refresh operation is current on system board.

**SYSCLK** -> Processor clock (but became 8 MHz. standard clock signal).

**OSC** -> 14.31818 MHz.clock signal to add-in board. Required for some types of video display boards.

**RESET** -> Provides a power-on reset for add-in cards. This is an active high signal.

**IRQ9** (IRQ2) |
**IRQ3** |
**IRQ4** |
**IRQ5** | <- 6 Interrupt request lines (active high)
**IRQ6** |
**IRQ7** |

**DRQ1** |
**DRQ2** | <- 3 DMA channel request lines
**DRQ3** |

**DACK1**|
**DACK2**| -> DMA acknowledge, indicates that a DMA operation can begin.
**DACK3** |

**TC** -> Terminal Count. Signals that one of the DMA channels has reached its pre-programmed number of transfer cycles.

**AEN** -> Issued by DMA logic on system board when a DMA transfer is in progress.

| | | |
|---|---|---|
| **+5V** | **\|** | |
| **-5V** | **\|** | |
| **-12V** | **\|** | Power supply and ground lines |
| **+12V** | **\|** | |
| **GND** | **\|** | |

62-Pin Bus Connector

| | | | |
|---|---|---|---|
| 0V | B01 | A01 | IOCHCHK# |
| RESET | B02 | A02 | SD7 |
| +5V | B03 | A03 | SD6 |
| IRQ9 | B04 | A04 | SD5 |
| -5V | B05 | A05 | SD4 |
| DRQ2 | B06 | A06 | SD3 |
| -12V | B07 | A07 | SD2 |
| SRDY# | B08 | A08 | SD1 |
| +12V | B09 | A09 | SD0 |
| 0V | B10 | A10 | IOCHRDY |
| SMEMW# | B11 | A11 | AEN |
| SMEMR# | B12 | A12 | SA19 |
| IOW# | B13 | A13 | SA18 |
| IOR# | B14 | A14 | SA17 |
| DACK3# | B15 | A15 | SA16 |
| DRQ3 | B16 | A17 | SA15 |
| DACK1# | B18 | A18 | SA14 |
| DRQ1 | B19 | A19 | SA13 |
| REFRESH# | B20 | A20 | SA12 |
| SYSCLK | B21 | A21 | SA11 |
| IRQ7 | B01 | A01 | SA10 |
| IRQ6 | B22 | A22 | SA9 |
| IRQ5 | B23 | A23 | SA8 |
| IRQ4 | B24 | A24 | SA7 |
| IRQ3 | B25 | A25 | SA6 |
| DACK2# | B26 | A26 | SA5 |
| TC | B27 | A27 | SA4 |
| BALE | B28 | A28 | SA3 |
| +5V | B29 | A29 | SA2 |
| OSC | B30 | A30 | SA1 |
| 0V | B31 | A31 | SA0 |

**The ISA BUS (AT-BUS)**

The PC bus is suited to the original design based on the Intel 8088 processor. However, the PC/ AT product was based on the Intel 80286 processor. The 80286 processor has an address space of 16 Megabytes (24 bits) and a 16-bit external data bus. Hence it was required to extend the PC bus specification so that it could use the 24-bit address bus and transfer data on a 16-bit wide data bus.

The extra address and data lines are provided for by the addition of the extra 36 pin physical connector as shown in the figure ' ISA (AT) BUS CONNECTOR'

The ISA (AT) bus also supports some additional interrupt lines and DMA channels.

The ISA (AT) bus can also accommodate 8-bit adapter cards. The signal MEMCS16 (for memory devices) or IOCS16 (for I/O devices) is used to 'tell' the MPU if the adapter card is a 16-bit device.

**36-PIN CONNECTOR SIGNAL NAMES**
**------------------------------**

**LA17-LA23** -> 7 most significant address lines for use with 80286 or 80386 processors to access a 128kB section within a 16 MByte address space. These addresses are latched on the Adapter card.

**SD8-SD15** <-> high-byte data lines to support 16-bit word access.

**MEMR** -> Memory read control signal for address space 0M to 16M

**MEMW**-> Memory write control signal for address space 0M to 16M

**MEMCS16** <- Indicates memory device addressed on the adapter board has 16-bit wide data.
**I/OCS16** <- Indicates that the addressed I/O device has 16-bit wide data

**IRQ10** |
**IRQ11** |
**IRQ12** | <- 5 Interrupt request lines
**IRQ14** |
**IRQ15** |

**DRQ0** |
**DRQ5** | <- 4 x DMA channels: request (DMA 0 is 8-bit, DMA 5 to 7 are 16-bit)
**DRQ6** |
**DRQ7** |

**DACK0** |
**DACK5** | -> DMA acknowledge, indicates that a DMA operation can begin.
**DACK6** |
**DACK7** |

**SBHE** <- Indicates that a byte is being transferred on the higher byte (SD8-SD15) lines.
**MASTER** <- Indicates that the, using DMA is taking control of the MPU bus.

**+5V** | Additional power line and ground line.
**GND** |

## ISA (AT) BUS CONNECTOR

| | | | |
|---|---|---|---|
| 0V | B01 | A01 | IOCHCHK# |
| RESET | B02 | A02 | SD7 |
| +5V | B03 | A03 | SD6 |
| IRQ9 | B04 | A04 | SD5 |
| -5V | B05 | A05 | SD4 |
| DRQ2 | B06 | A06 | SD3 |
| -12V | B07 | A07 | SD2 |
| SRDY# | B08 | A08 | SD1 |
| +12V | B09 | A09 | SD0 |
| 0V | B10 | A10 | IOCHRDY |
| SMEMW# | B11 | A11 | AEN |
| SMEMR# | B12 | A12 | SA19 |
| IOW# | B13 | A13 | SA18 |
| IOR# | B14 | A14 | SA17 |
| DACK3# | B15 | A15 | SA16 |
| DRQ3 | B16 | A17 | SA15 |
| DACK1# | B18 | A18 | SA14 |
| DRQ1 | B19 | A19 | SA13 |
| REFRESH# | B20 | A20 | SA12 |
| SYSCLK | B21 | A21 | SA11 |
| IRQ7 | B01 | A01 | SA10 |
| IRQ6 | B22 | A22 | SA9 |
| IRQ5 | B23 | A23 | SA8 |
| IRQ4 | B24 | A24 | SA7 |
| IRQ3 | B25 | A25 | SA6 |
| DACK2# | B26 | A26 | SA5 |
| TC | B27 | A27 | SA4 |
| BALE | B28 | A28 | SA3 |
| +5V | B29 | A29 | SA2 |
| OSC | B30 | A30 | SA1 |
| 0V | B31 | A31 | SA0 |

| | | | |
|---|---|---|---|
| MEMCS16# | D01 | C01 | SBHE# |
| IOCS16# | D02 | C02 | LA23 |
| IRQ10 | D03 | C03 | LA22 |
| IRQ11 | D04 | C04 | LA21 |
| IRQ12 | D05 | C05 | LA20 |
| IRQ15 | D06 | C06 | LA19 |
| IRQ14 | D07 | C07 | LA18 |
| DACK0# | D08 | C08 | LA17 |
| DRQ0 | D09 | C09 | MEMR# |
| DACK5# | D10 | C10 | MEMW# |
| DRQ5 | D11 | C11 | SD8 |
| DACK6# | D12 | C12 | SD9 |
| DRQ6 | D13 | C13 | SD10 |
| DACK7# | D14 | C14 | SD11 |
| DRQ7 | D15 | C15 | SD12 |
| +5V | D16 | C16 | SD13 |
| MASTER# | D17 | C17 | SD14 |
| 0V | D18 | C18 | SD15 |

## Some general background on servicing multiple interrupts

### Establishing the Source of an Interrupt

When there are a number of devices in a system, which can request an interrupt, the microprocessor must have a method for establishing which one of the devices caused the interrupt. There are a number of methods for achieving this; some of these methods are listed below:

**Multiple Interrupt Lines**
Some microprocessors have a number of interrupt lines. A separate interrupt line could be assigned to each I/O device. However, the most microprocessors use a single maskable interrupt request line, which is shared by a number of I/O devices.

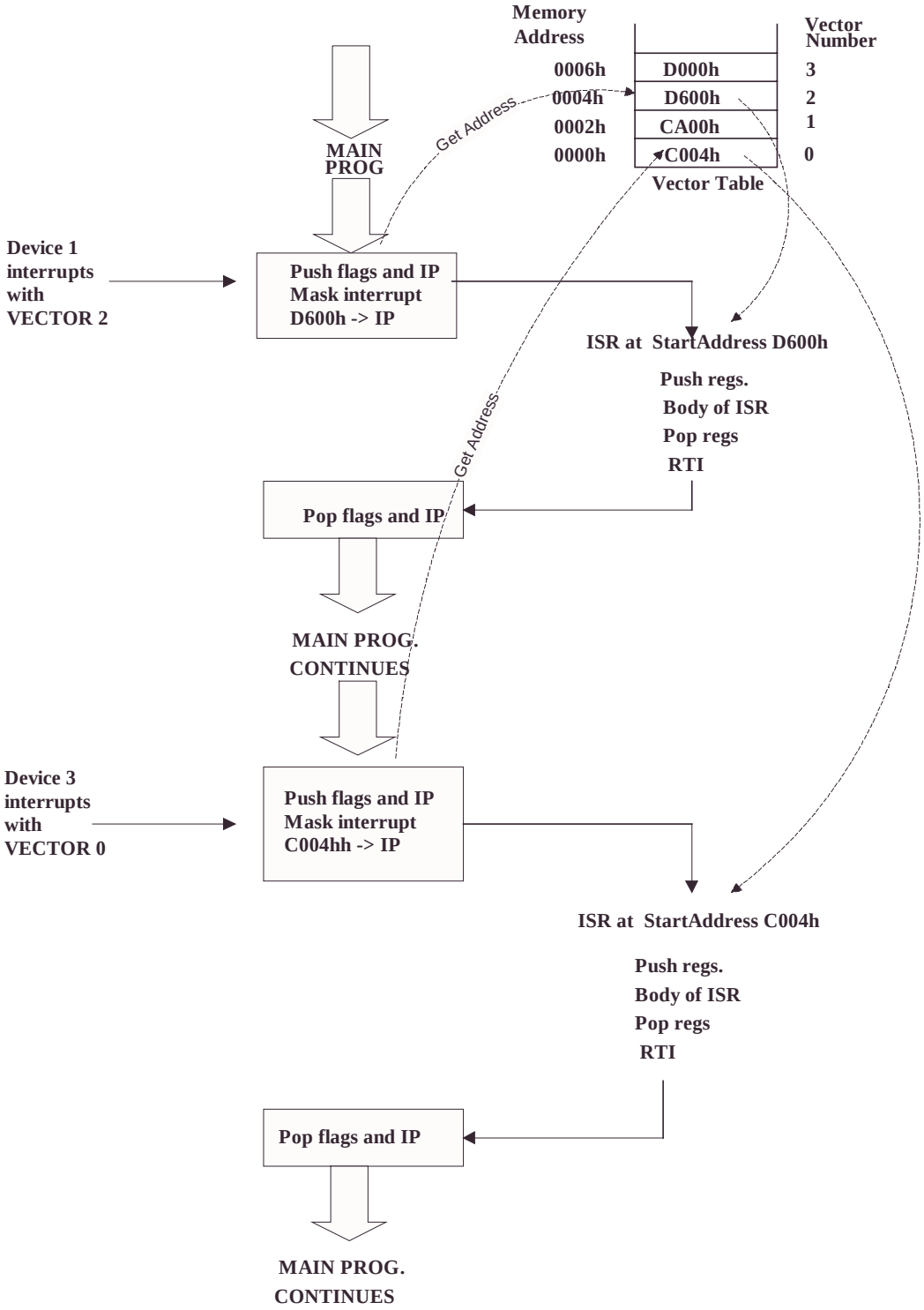
**Software Device Polling**
In the software polling method, when the microprocessor receives an interrupt, the first operation of the ISR is to establish which I/O device caused the interrupt. Each I/O device will have an interrupt status bit to indicate whether the I/O device generated an interrupt. The software polls each device in turn until it finds the device, which caused the interrupt. The program then jumps to a section of code, which will handle the I/O device operation. The software polling method is slow, as a number of devices may have to be tested before the required device is identified.

**Vectored Interrupt**
The interrupting device itself passes a *vector* interrupt address pointer to the microprocessor. The vector pointer is used to access a unique location in the *interrupt vector table*. The interrupt vector table is usually located sequentially in program memory and each location contains the start address of the interrupt service routine for each device.

The vectored interrupt scheme allows a number of devices to be connected to one interrupt request line. When the microprocessor accepts an interrupt request it generates a hardware interrupt acknowledge signal (**INTA**). This **INTA** signal is used to gate the vector pointer from the interrupting device on to the microprocessor data bus. This vector pointer will allow the microprocessor to identify the correct entry in the vector table where the address of the ISR routine is contained. The vectored interrupt method is a **very fast** method of identifying the interrupting device.

The figure 'Example of Interrupt Vector Use (16 bit Address System)' shows how the vector table is used to find the ISR start address in a small 16-bit address system. The system programmer initialises the table to define the vectors. Each location of the vector table contains a 16-bit address; hence each location is two bytes in size.

**EXAMPLE OF INTERRUPT VECTOR USE (16 bit Address System)**




## Priority Schemes for Servicing Multiple Interrupts

Where multiple I/O devices exist, there is the possibility that more than one device will issue an interrupt request at the same time. Further, while one interrupt is being serviced by the MPU a number of other interrupt requests may queue for service. Different devices will have different priority needs so the determination of priority is critical. Some devices will need immediate attention; others may be able to wait for longer lengths of time.

**Priority Encoder**

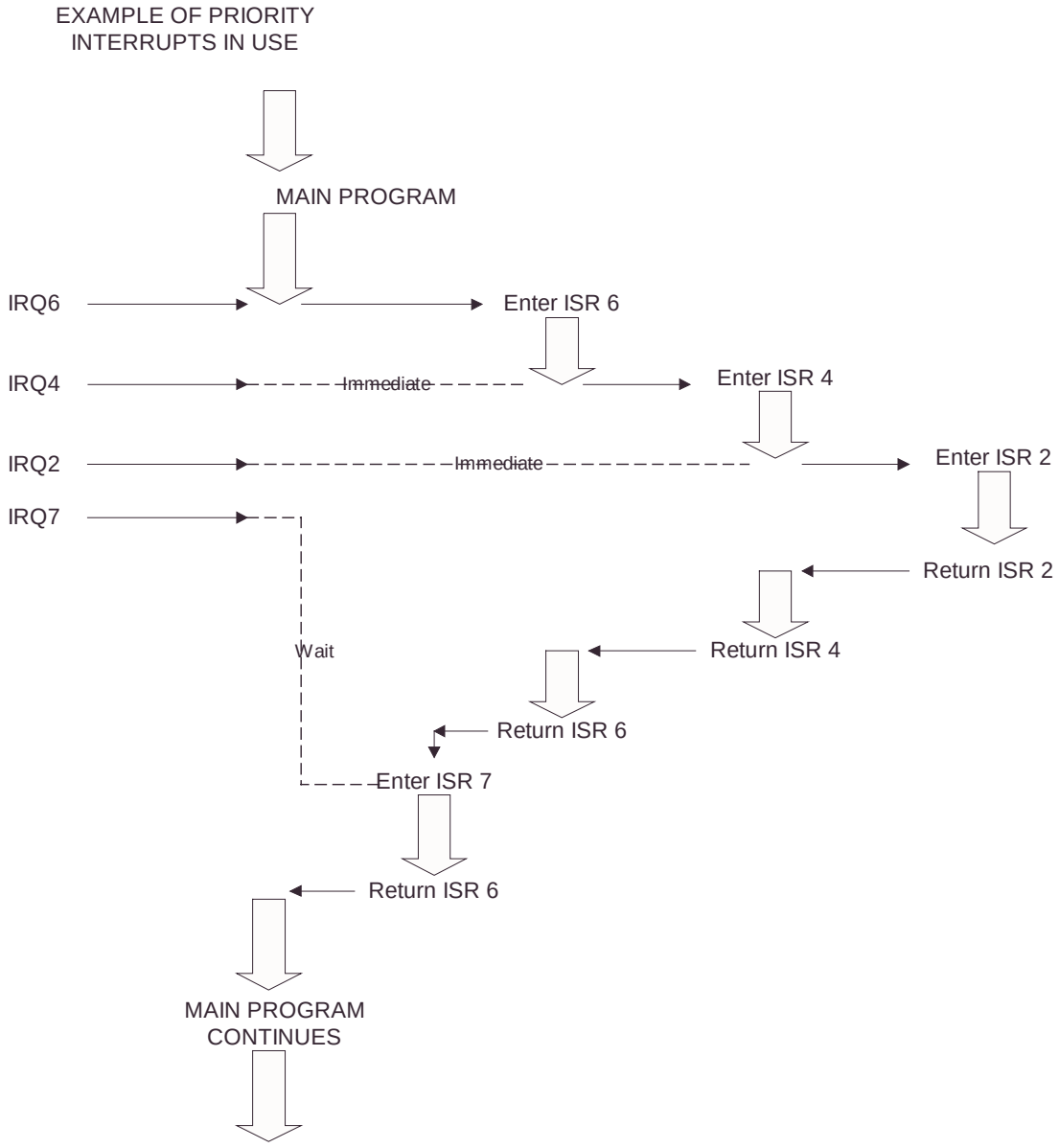
A priority encoder circuit is used to resolve priority. Each interrupt request input line is assigned a priority level. The encoder will pass the highest priority interrupt to the MPU and mask interrupts with have lower priority levels than the level currently being serviced. If an interrupt request is made at a higher priority than the interrupt currently being serviced then the current interrupt is interrupted to allow the higher priority interrupt to be serviced. The priority encoder can be programmed to support different priority modes, but it is common to find some fixed priority assignment as follows:

**FIXED PRIORITY**

**IR0 - Highest priority**
**IR1**
**IR2**
**IR3**
**IR4**
**IR5**
**IR6**
**IR7 - Lowest priority**

**Servicing Multiple Interrupts in a Priority Based System.**

Under a priority interrupt scheme, as stated, an interrupt service routine itself can be interrupted by an interrupt request from a higher priority device. If an interrupt service routine does not want to allow any interrupts it can disable interrupts by control of the interrupt mask bit (IE flag). Interrupts are usually numbered with the highest priority being INTERRUPT 0.
See figure 'Example of Priority Interrupts in Use'




## Interrupts on the PC

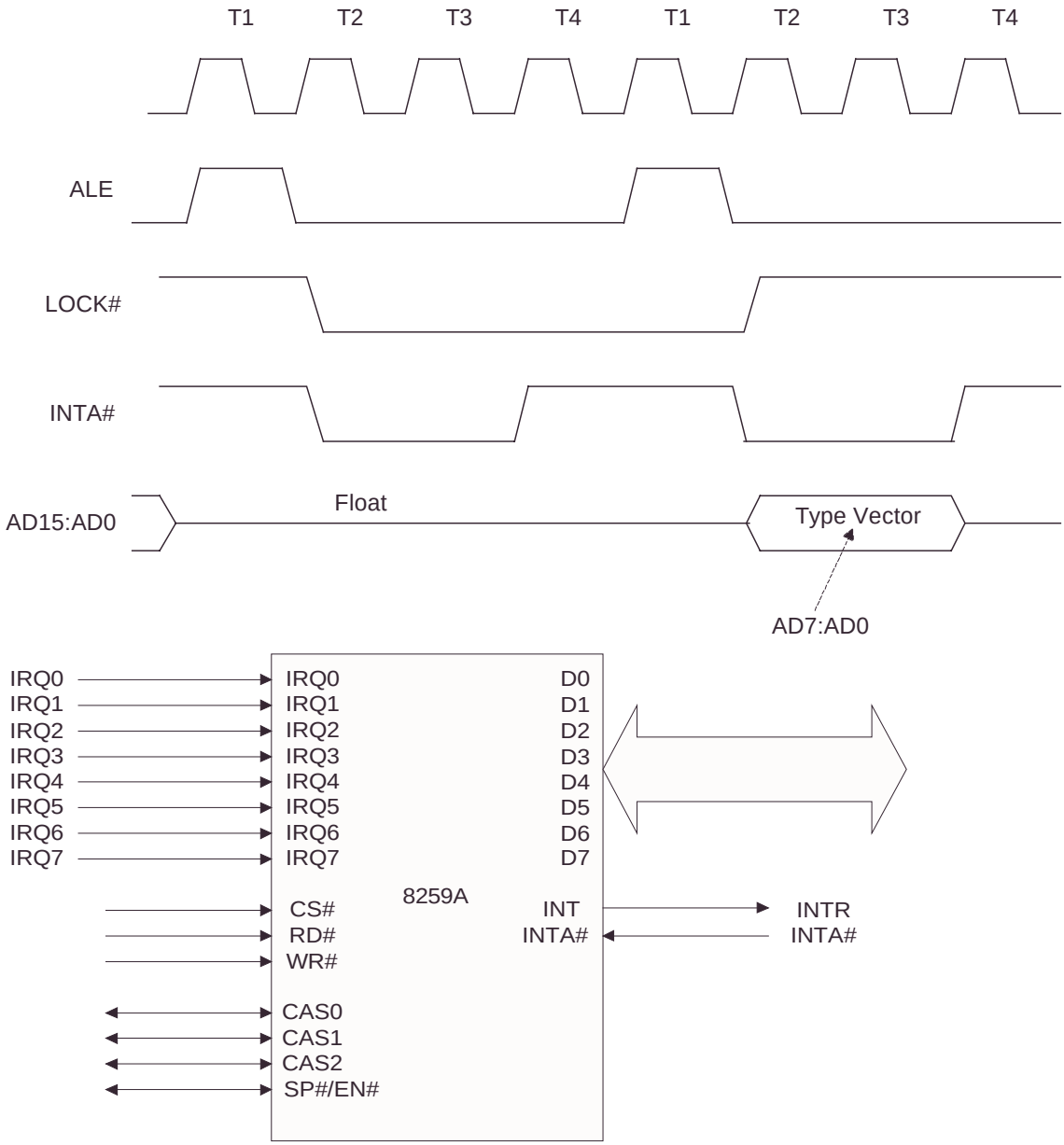
The PC uses the Intel i8259 interrupt controller chip, see figure 'i8259 Interrupt Controller'.

The i8259 has 8 interrupt request inputs IR0..IR7, or IRQ0..IRQ7in the PC design. The Interrupt Request Register latches any interrupt requests. The requests go to priority resolving logic and the priority requests go to the Interrupt Service Register. The Interrupt Mask Register and associated logic will allow masking of individual interrupt requests under program control. The i8259 contains a number of internal programmable registers (detail covered elsewhere).

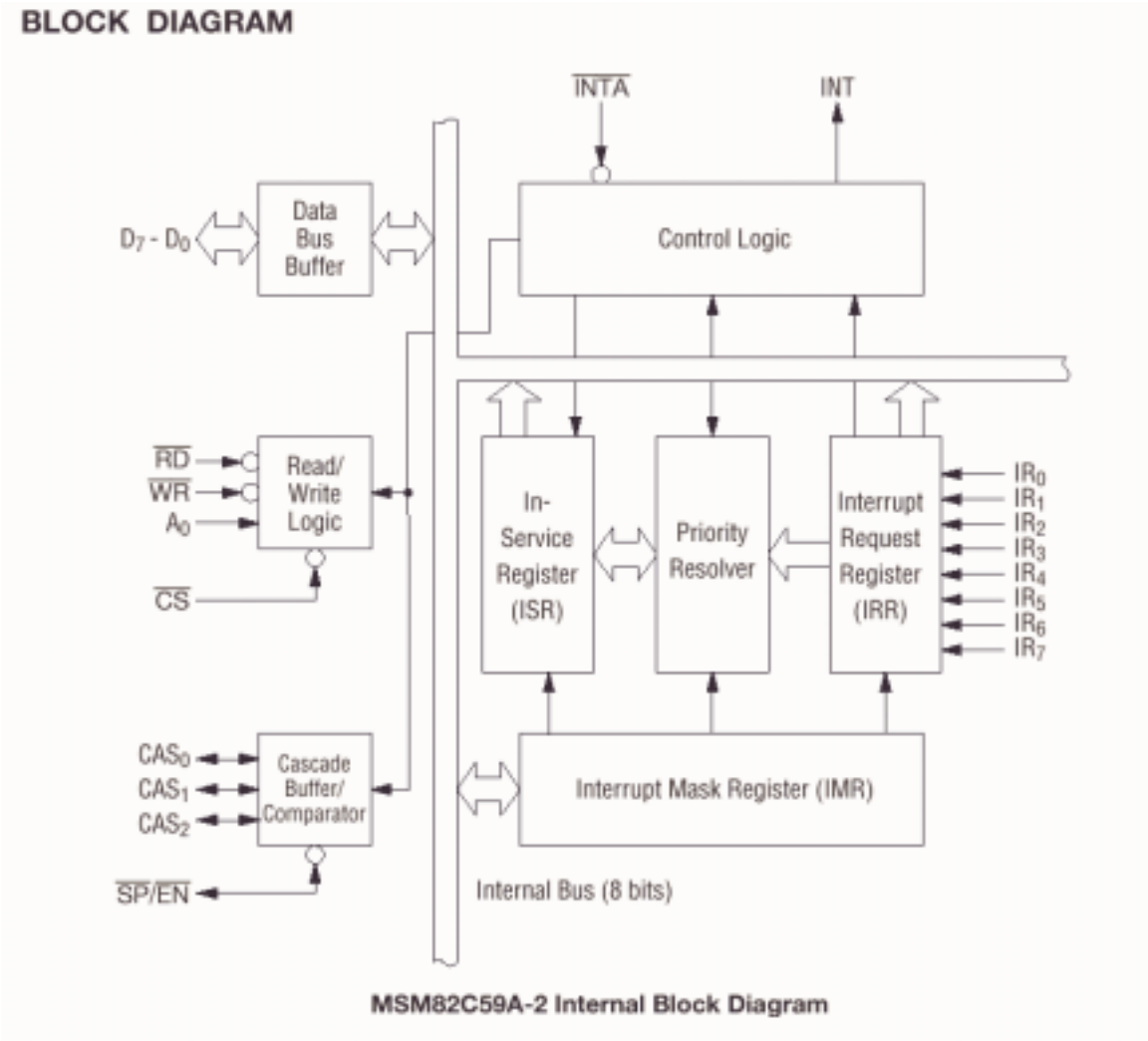
**Timing of Events**
On receipt of an interrupt request, the request is prioritised along with any other pending interrupt requests. If this is the only pending request, or the highest priority request, then the i8259 sends an interrupt request (INTR) to the MPU.

The MPU sends two INTA response pulses to the i8259, the first freezes priority and the second allows the i8259 to send the 8-bit vector information to the MPU, along the data bus AD0..AD7.

The MPU 'vectors' to the ISR address as discussed earlier.

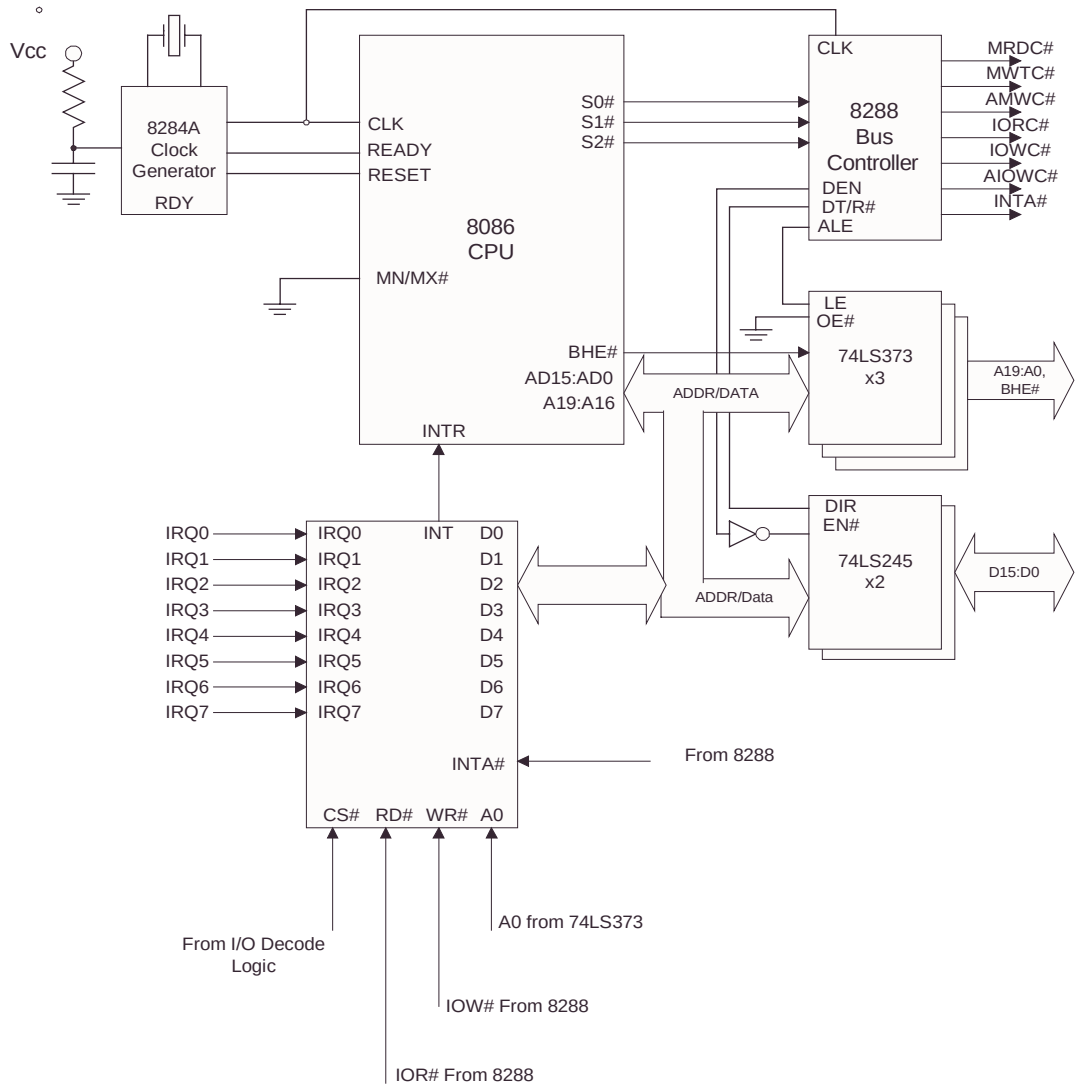



## i8259A Interrupt Controller




## The i8259 in the PC Circuit

The figure below shows how the i8259 is used in the PC circuit. The i8259 is wired to the Local Bus AD0 to AD7.The i8259 is accessed by the MPU for reading and writing of the internal registers. These registers are seen as I/O devices and are decoded to the PC I/O map (see later).




## CASCADING i8259s in the PC/AT

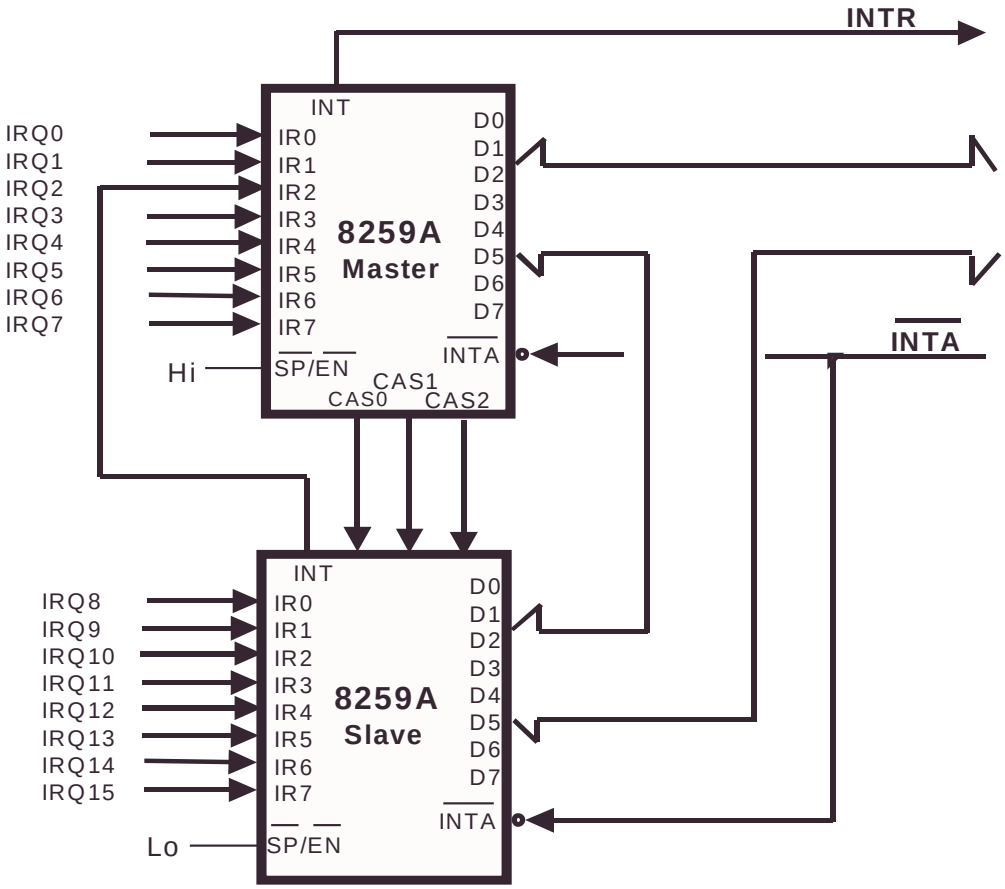
Two i8259 Interrupt Controllers are cascaded to provide fifteen IRQ inputs.
The original PC design employs a single i8259 interrupt controller device to support eight interrupt levels. However, the PC/AT supports more hardware interrupt request lines. Figure 'Cascaded 8259s' shows how the 8259s are wired in the PC/AT design. The interrupt request signal of the slave 8259 is connected to IRQ2 of the master 8259. The master passes control signals to the slave via the connections CAS0..CAS2.

By redirecting the INT output from the slave 8259 to the master at input IRQ2, the resulting interrupt priorities of the PC/AT can be listed as follows:

| Priority | | Use of PC/AT Interrupt |
|---|---|---|
| Highest | IRQ0 | Timer 0 |
| | IRQ1 | Keyboard |
| | IRQ2 | Redirection from slave 8259 |
| | IRQ8 | Real time clock |
| | IRQ9 | * |
| | IRQ10 | |
| | IRQ11 | |
| | IRQ12 | |
| | IRQ13 | C0-processor |
| | IRQ14 | Hard disk controller |
| | IRQ15 | |
| | IRQ3 | COM2 port |
| | IRQ4 | COM1 port |
| | IRQ5 | LPT2 |
| | IRQ6 | Floppy disk controller |
| Lowest | IRQ7 | LPT1 |

* IRQ9 interrupt is redirected to IRQ2 vector

## Cascaded i8259As




## Interrupt Vectors on the PC/AT

In the i80x86 processor design the interrupt vector table is defined at the bottom of the memory map. The table is 1024 (400h) bytes in size and can contain 256 vectors. Each vector location is 4 bytes long *(segment:offset address*). Figure 'PC/AT Vector Table' shows the standard vector table for the PC/AT.

An interrupt can be triggered by one of three types of events:

**- EXCEPTIONS: Hardware interrupt, INTERNALLY generated**
Generated by the MPU itself to flag fault (or other machine related conditions) such as arithmetic overflow etc.

Examples:

| | |
|---|---|
| **Interrupt 0** | Quotient exceeds the maximum value that the division instruction allows. This is a fault condition. |
| **Interrupt 1** | The Trap Fla g (TF) must be set in the flags register. Each instruction causes a 'trap' to an ISR routine. Used by DEBUG type software. |
| **Interrupt 2** | NMI, the non-maskable interrupt input. |
| **Interrupt 3** | MPU can issue a special one-byte interrupt instruction to act as a break point. Used by DEBUG type software. |
| **Interrupt 4** | Interrupt on overflow |

**- Hardware interrupt, EXTERNALLY generated**
Peripheral devices usually generate external interrupt requests as asynchronous events. Figure 'PC/AT Hardware External Interrupt' shows a simplified example of the process. The vector pointer sent to the MPU on the second INTA pulse of the interrupt cycle is defined as follows:

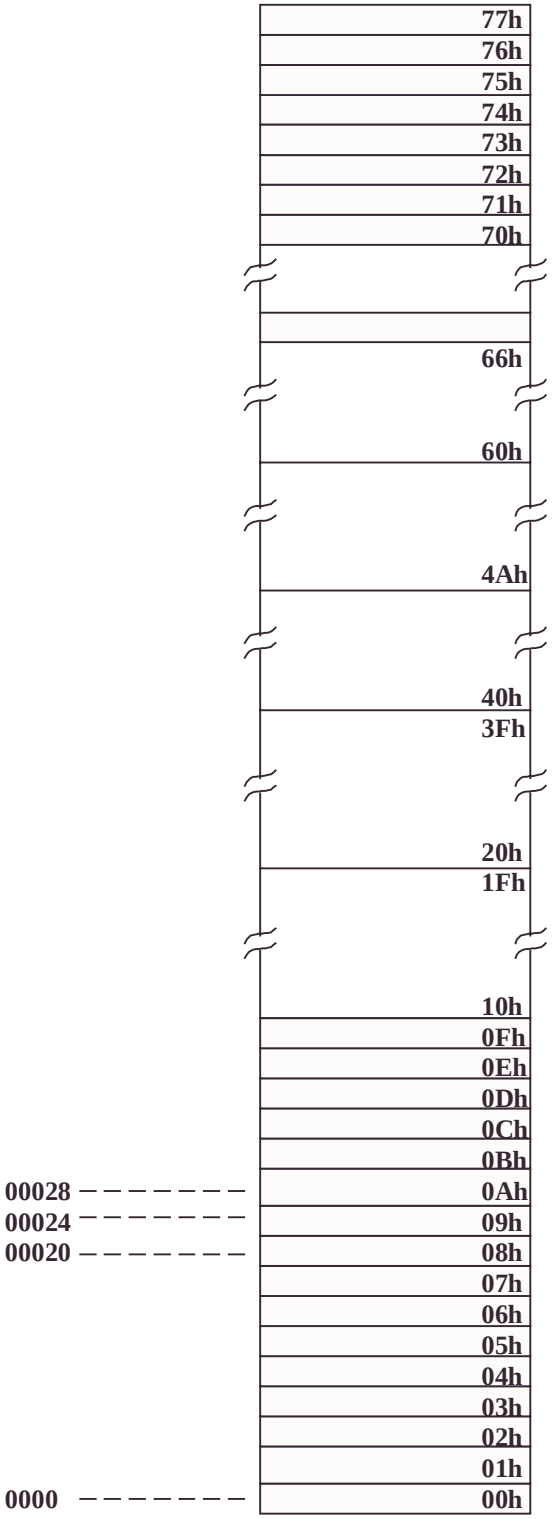
*Vector Pointer:* $A_9$ $A_8$ $A_7$ $A_6$ $A_5$ $L_3$ $L_2$ $L_1$

$A_9..A_5$ locates the position of the 32 byte area within the vector table. Each 8259 will have an associated 32 byte area to hold the eight 4-byte vector addresses.

$L_3..L_1$ defines the interrupt level, 1 of 8, from the 8259.

The vector pointer is multiplied by 4 by the MPU to form the 10-bit address of the starting byte of the particular vector 4 byte location.

## PC/AT VECTOR TABLE




**IRQ15 Reserved**
**IRQ14 Fixed Disk Controller**
**IRQ13 80x87**
**IRQ12 Reserved**
**IRQ11...Reserved**
**IRQ10 ..Reserved**
**IRQ9 Directed to IRQ2**
**IRQ8 CMOS RTC**

**EMM**

**User Interrupts**

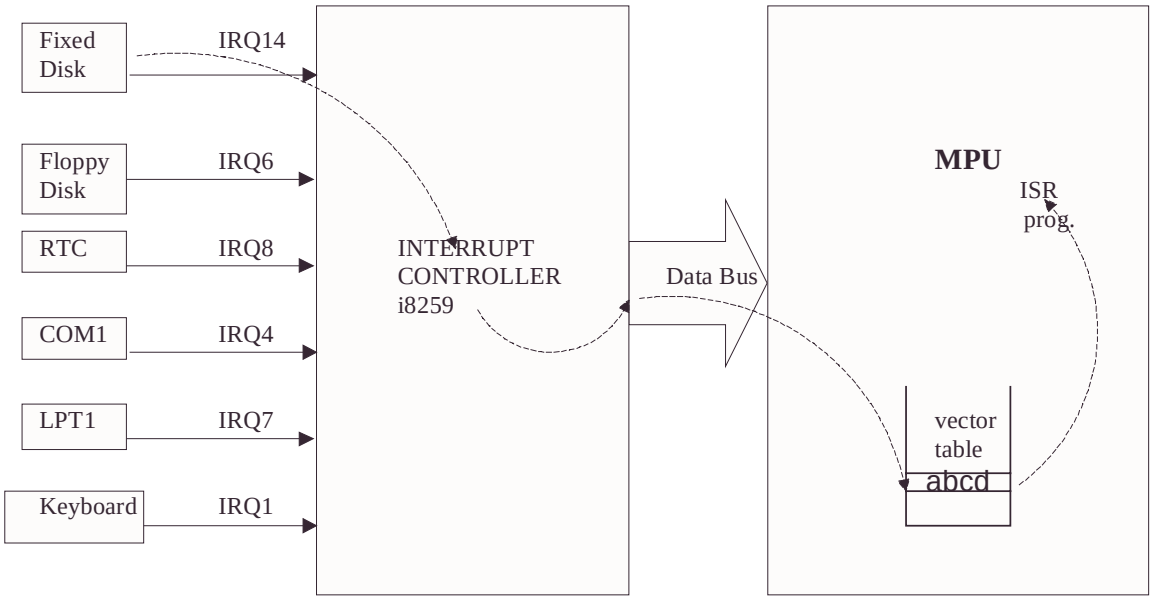
**ROM BIOS and VIDEO**

**MSDOS SWI (30-3F reserved)**

**ROM BIOS SWI**

**IRQ 7 LPT1**
**IRQ6 Floppy Disk**
**IRQ5 LPT2**
**IRQ4 COM1 Port**
**IRQ3 COM2 Port**
**IRQ2 Cascade from Slave 8259**
**IRQ1 Keyboard**
**IRQ0 Timer tick**
**80x87 not present**
**Invalid opcode**
**Print screen (BIOS)**
**Overflow**
**Break point instruction**
**NMI**
**Single step**
**Divide by zero**

# PC/AT HARDWARE EXTERNAL INTERRUPT (Simplified Diagram)




**Peripheral** **IRQ#** **Forces Vector pointer** **Finds Address of ISR and transfers control**

a) Peripheral device causes a hardware interrupt

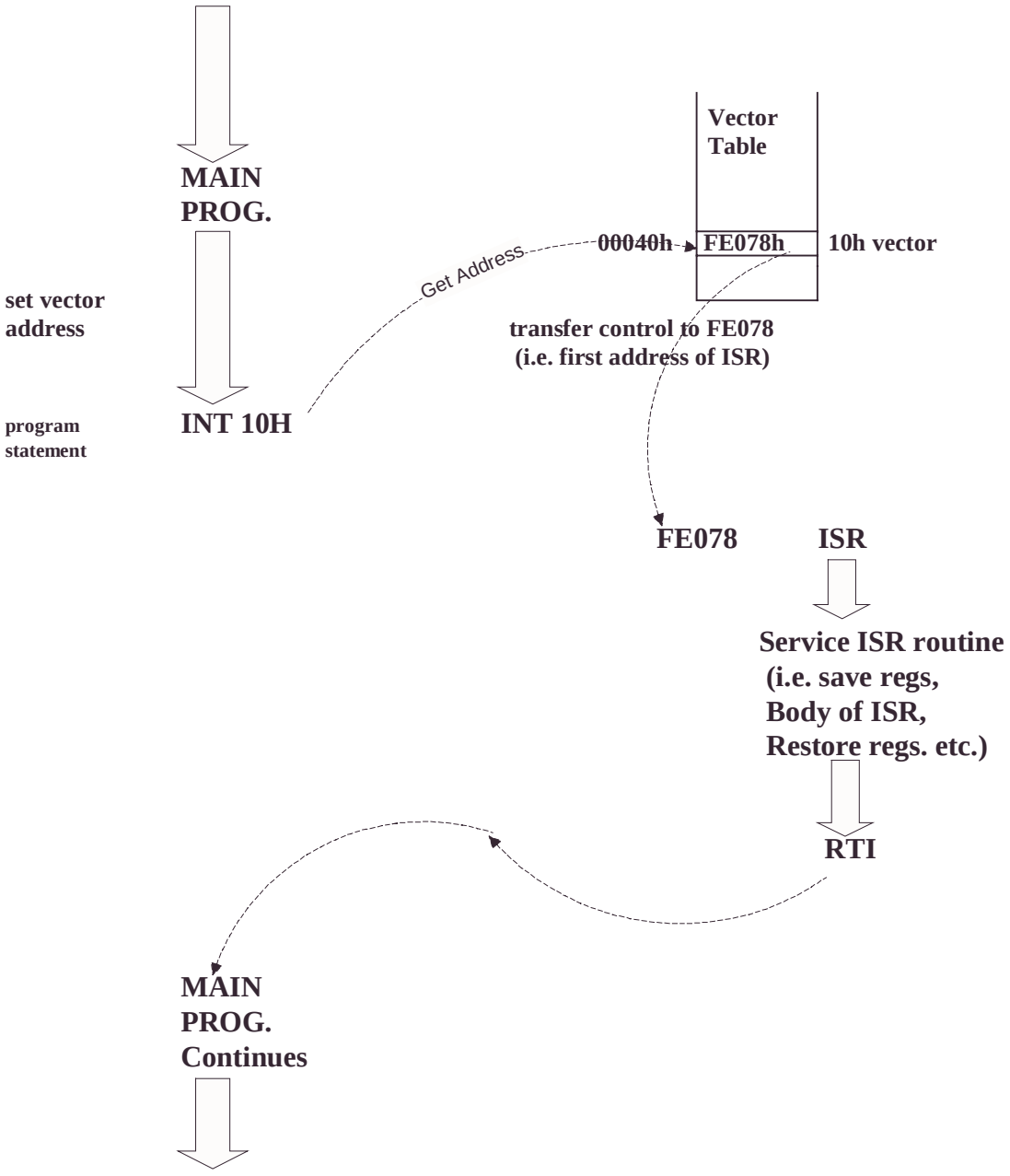
b) Interrupt Controller forces the correct vector pointer on data bus

c) Vector table is used to locate start of ISR

**- Software interrupts**

A synchronous event triggered by a running program, see figure 'Example Program Flow for a Software Interrupt'. The programmer makes the interrupt call by including the 'INT X' statement within the program. The software interrupt 'vectors' to a specific ISR just like the hardware interrupt does. MS-DOS and BIOS system calls are made using software interrupt.
Example of such calls will be described later.

## EXAMPLE PROGRAM FLOW FOR A SOFTWARE INTERRUPT




## The i8237A DMA Controller

- The i8237A contains four independent DMA channels.

- The original PC/XT design included a single i8237A device to provide four DMA channels.

- The PC/AT design extended the number of DMA channels to seven by using two i8237As

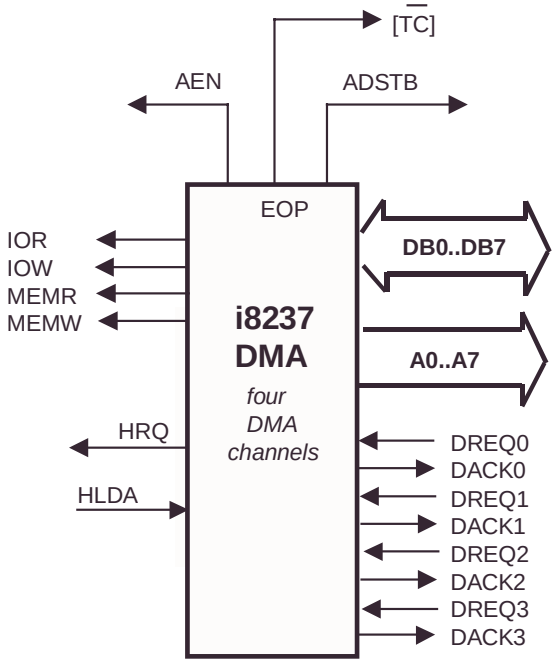
- The i8237A is a programmable device and can be configured for single transfers, block transfers, READS, WRITES or Memory-to-Memory transfers.

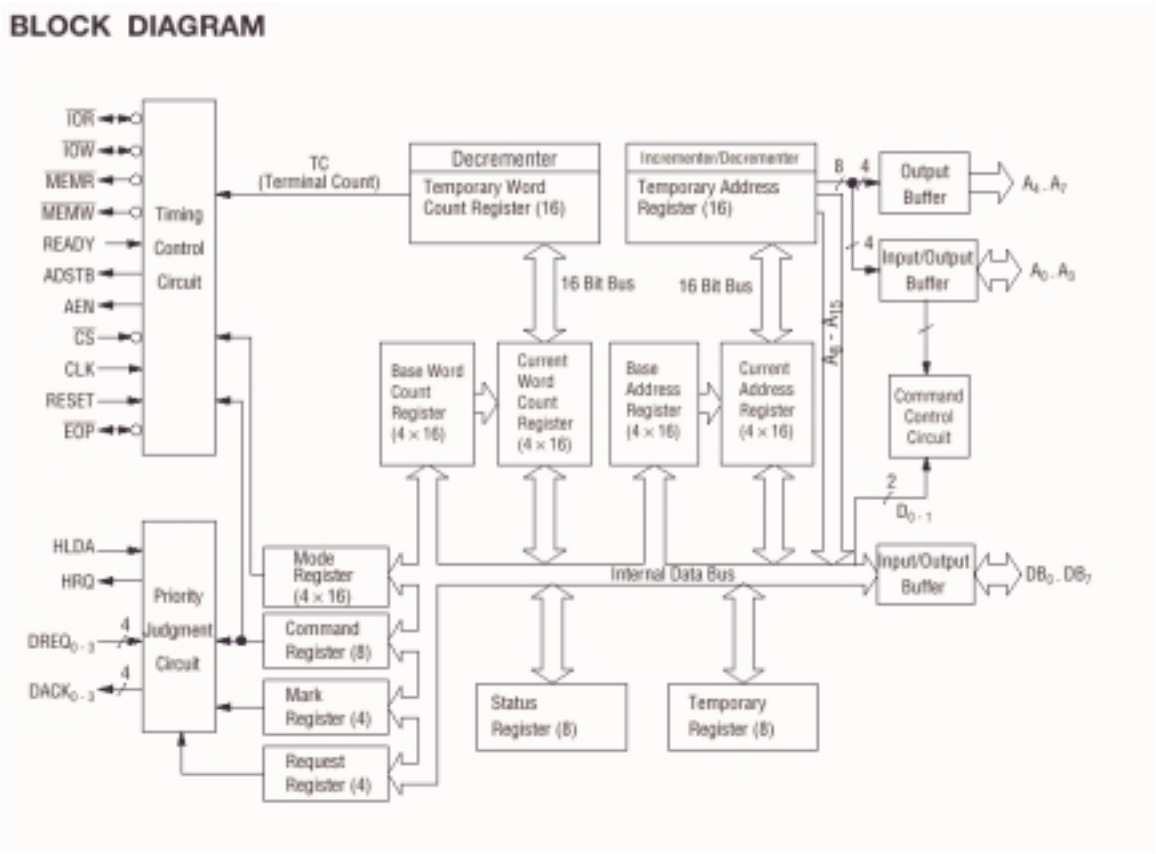
- The i8237A was designed to allow byte addressing for 8-bit data transfers. In the PC/AT design, a contrived 16-bit transfer design is implemented using the i8237A.

- The i8237A provides control lines such as IOR, IOW, MEMR, MEMW, HRQ (hold request), HLDA (hold acknowledge), DREQ0..DREQ3 (the four DMA request lines) and DACK0..DACK3 (the four DMA acknowledge lines).

- The i8237A uses a multiplexed address and data bus to reduce the device pin count. The DB0..DB7 lines contain the data bus along with the low byte of the 16-bit address bus. An external latch is required to demultiplex the address lines.

**i8237A Logic Symbol**





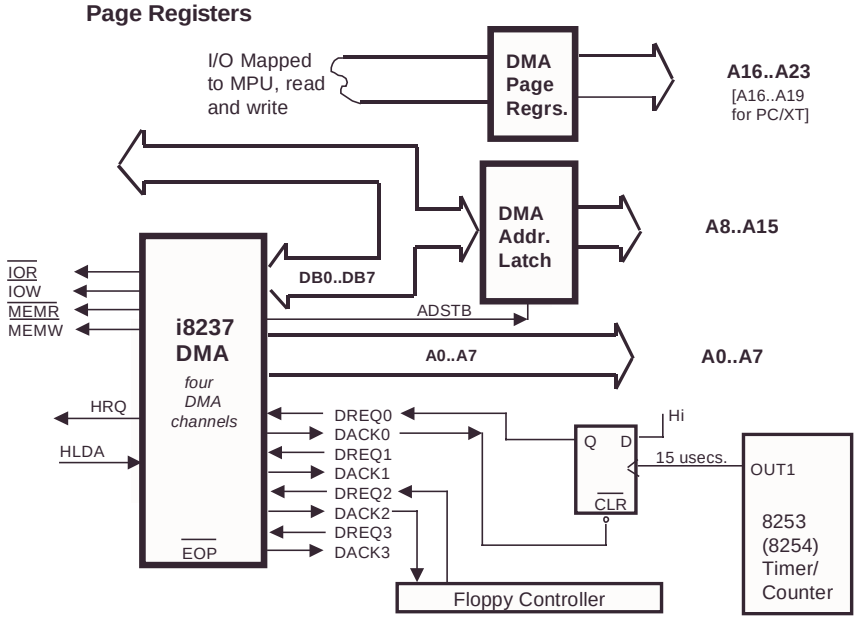

## i8237A Address Latch and Page Registers

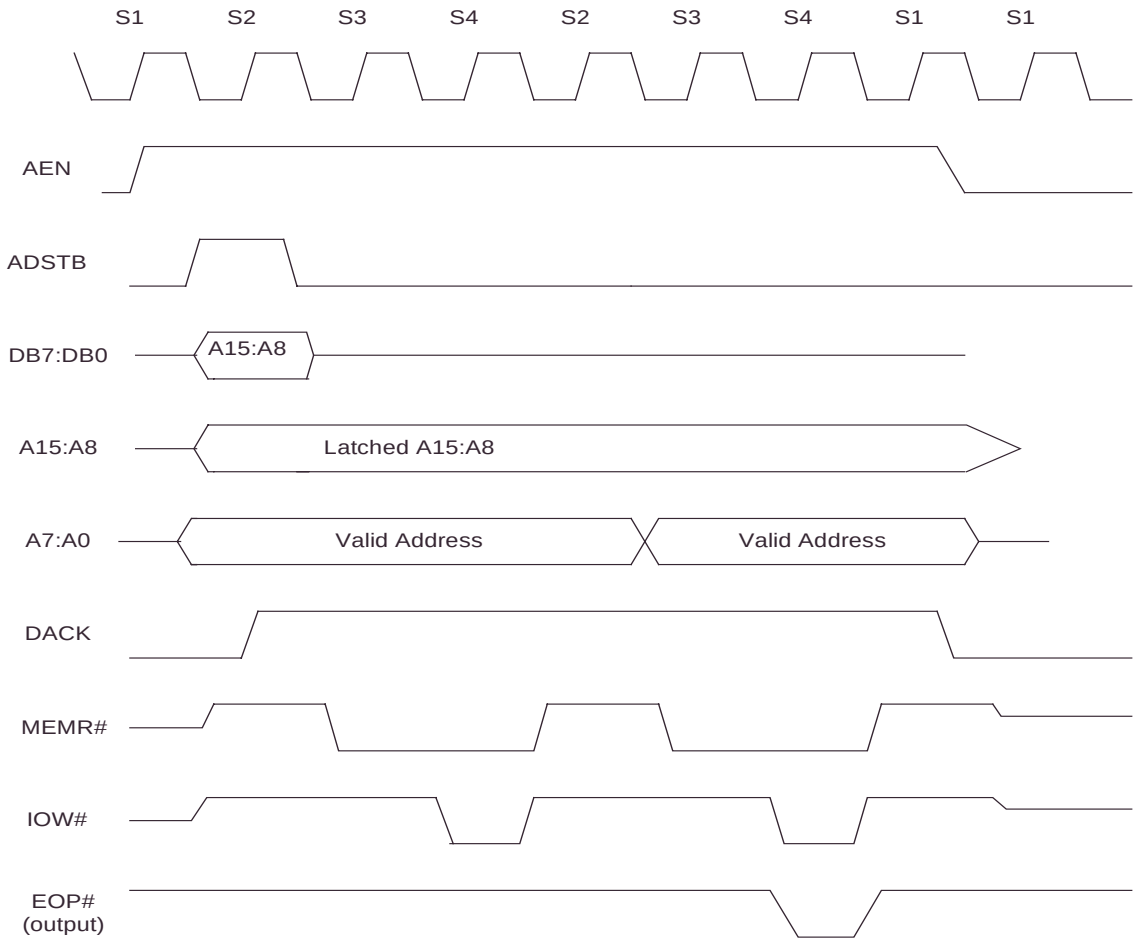
**The Address Latch**
An 8-bit latch is required with the i8237A circuit to de-multiplex the lower byte of the address bus as shown below. Figure 'i8237A DMA Read Transfer Timing' shows how **ADSTB** (Address Strobe) is used to latch the A8..A15 lines, in a timing diagram.

**The DMA Page Registers**
Since the i8237A generates sixteen address lines only, the higher order address lines must be provided, external to the chip. In the PC/XT design an additional four address lines A16..A19 are required to provide 'page' addresses. The diagram below shows four DMA page registers. These registers are written to and read from directly by the MPU using I/O instructions. In the PC/AT design, based on the i80286 processor, a 24-bit address bus is used, so 8-bit page registers are required. In 80386 and higher processors, employing a 32-bit address bus, a wider page register, 16-bits, is required. The page register design does not allow a DMA operation to cross a 64K boundary.

Note, in the original PC/XT design page registers were provided for DMA channels 1 to 3 only. DMA channel 0 was used for memory refresh and did not require the page register.

**i8237A Address Latch and Page Registers**



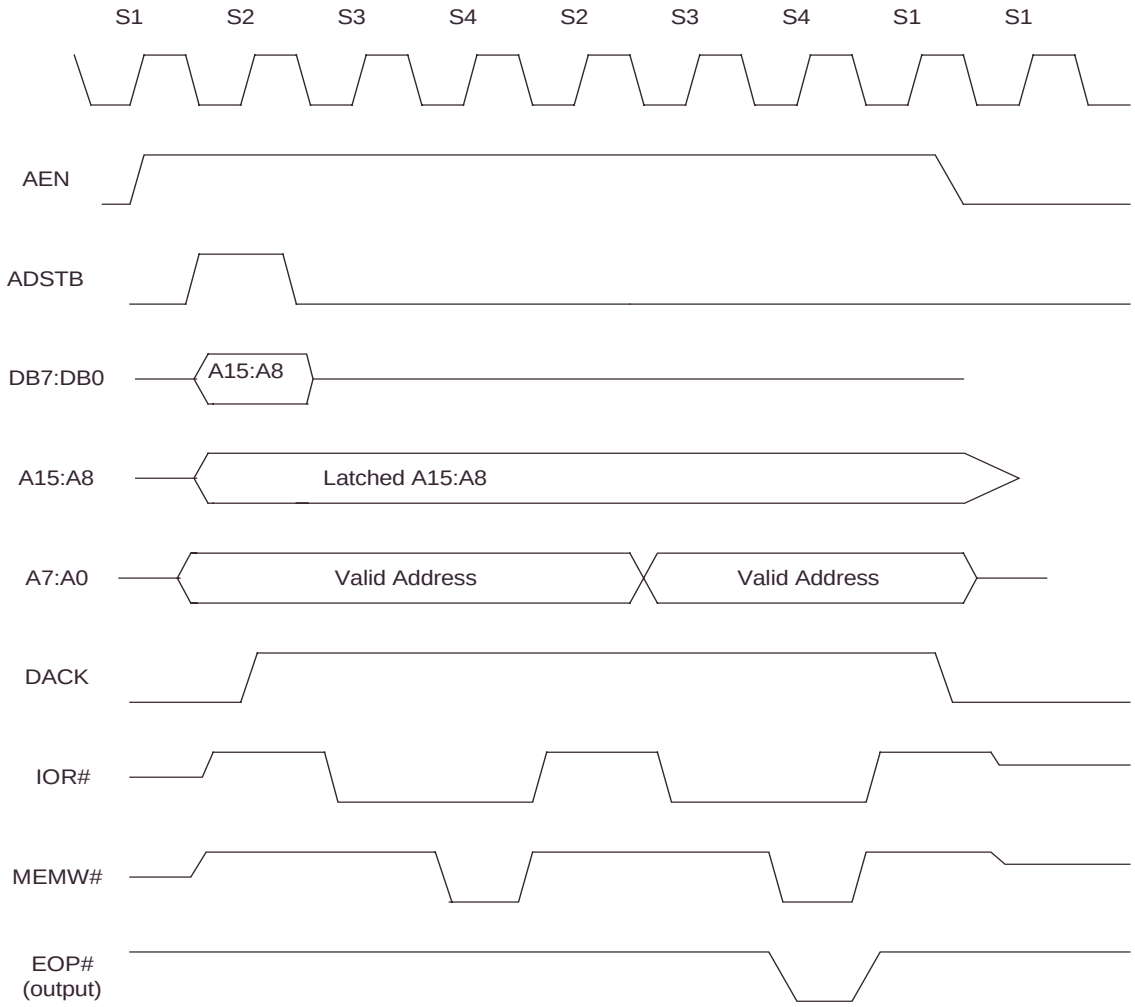




**DMA READ Transfer Timing Diagram using i8237A**

**Notes:**

1. Every a 256 bus cycles a 4-clock cycle is required to define a high-byte address.

2. Just like a microprocessor, the i8237A has a READY input so that slow devices can cause wait states. The wait states are inserted between S3 and S4.




**DMA WRITE Transfer Timing Diagram using i8237A**

## DMA Address Tracking

The i8237A has a number of read/write I/O port addressable registers to allow device initialisation and control. The programming of these registers is described in the i8237A specification. Here, we will describe the functional operation of four registers involved in the tracking of memory addresses during a DMA block transfer.

The *Base Address register* and the *Base Word Count register* are initialised with the memory starting address and the number of required transfers, respectively.

The Content of the *Base Address register*s is copied to the *Current Address register* and the content of the *Base Word Count register* is copied to the *Current Word Count register.*

During DMA operation, after each single DMA transfer cycle, the *Current Address register* is incremented (or decremented if controller is programmed to count addresses downwards), and the *Current Word Count register* is decremented. When the *Current Word Count becomes* 'zero', the internal TC (Terminal Count) flag is set to show that the full DMA block transfer is complete and the EOP signal is pulsed low to advise of the TC externally. Note, EOP can also be set as an input signal to terminate DMA operation by external control.

When the full DMA block transfer is complete the *Current Address Register* and *Current Word Count register* are automatically reloaded with the initial values from the *Base Address register* and the *Base Word Count Register*.

Note if we want to define a transfer block size of 256 we would program the *Base Word Count register* with a value of '255', since a transfer is completed before the count is decremented.

Each DMA channel will include these four 16-bit registers:

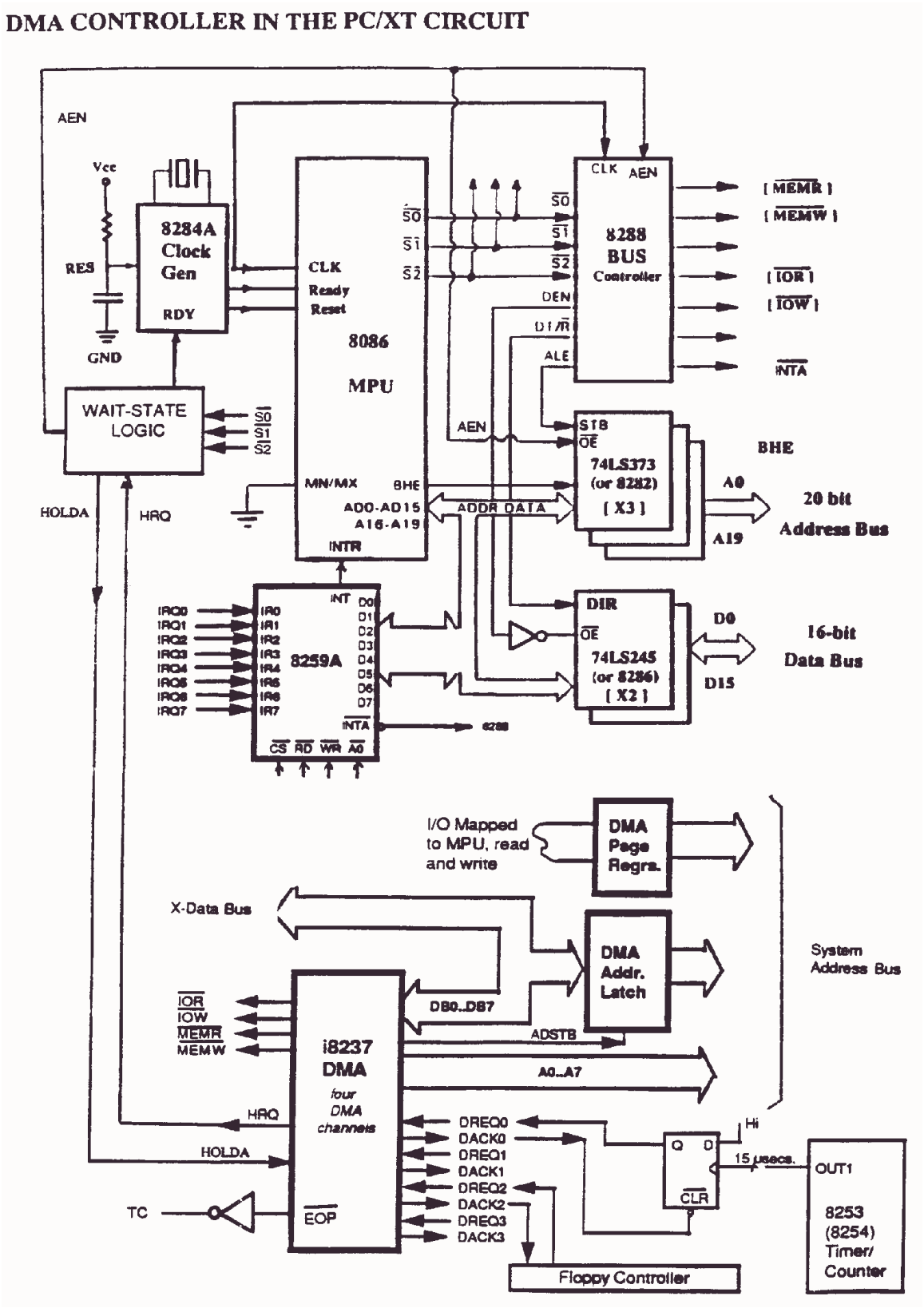
**BASE ADDRESS REGISTER**

**BASE WORD COUNT REGISTER**

**CURRENT ADDRESS REGISTER**

**CURRENT WORD COUNT REGISTER**

## The i8237A DMA Controller in the PC/XT




The DMA controller drives the System Bus address lines. The DMA controller is interfaced to the system as an I/O ported device on the system's X-bus (port addresses in the I/O map will be shown later). The DMA controller data lines are connected to the X-data bus.

Following a DMA request from a device (activation of DRQx) the i8237A raises the HRQ (hold request) signal. The Wait-State Logic circuit detects HRQ and then when an MPU passive state occurs (S2,S1,S0 = 111) the MPU ready line (through the i8284) is held low so that the MPU is held in its WAIT STATE condition. The Wait-State Logic then generates an AEN signal (The i8237A's own AEN signal is not used), which disables (tri-states) the MPU's system address bus latches (74LS373s) and tri-states the MPU's system data bus transceivers. The i8288 bus controller disables the data bus transceivers in response to the AEN signal. The Wait-State Logic then sends HOLDA (Hold acknowledge) to the DMA controller, which, in turn, activates the DACKx line to the requesting device, this DACKx signal is effectively the 'chip select' for the device. Eventually, the i8237A will release its HRQ signal and allow the MPU to come out of the WAIT-STATE condition.

Note, the i8287's EOP signal is inverted and becomes the TC signal (ISA bus signal).

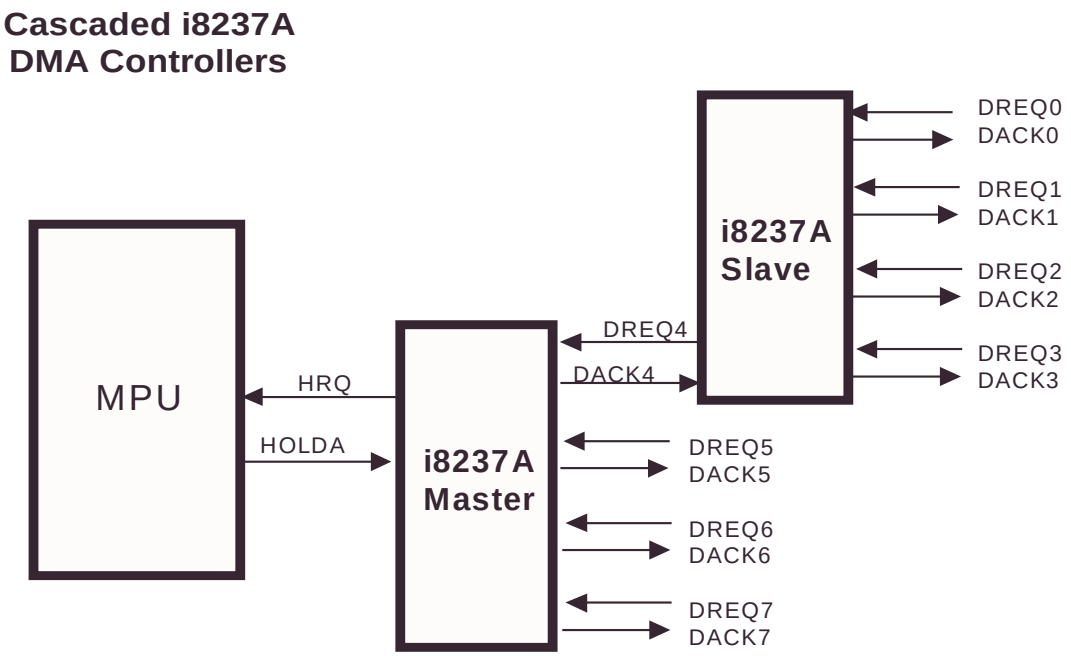
DEFINED USE OF DMA CHANNEL 0 AND CHANNEL 2

In the original PC/XT design DMA channel 0 was used for DRAM memory refresh. OUT1 from the system Timer/Counter provides a pulse at 15 microsecond intervals. This OUT1 signal drives DRQ0 by the setting of a simple latch, as shown. The DACK0 signal clears the latch. The DMA channel 0 is programmed for single-byte transfers so once every 15 microseconds the MPU is effectively stopped to allow the DMA controller to do a single memory read cycle and over time the entire DRAM memory is refreshed.

In the original PC/XT design DMA channel 2 is assigned to the floppy disk drive. The particular implementation does not yield a high performance solution. It is thought that floppy disk drive DMA was used to conveniently avoid having to disable keyboard and timer interrupts during floppy transactions, and not for high performance reasons. The floppy disk interface asserts DRQ2 and this initiates a single-byte transfer across the floppy interface.

Note: i8237A labels 'Hold Acknowledge' as HLDA, The PC/AT designers labelled this signal as HOLDA.

## Cascaded DMA Controllers in the PC/AT




The PC/AT product design includes an additional i8237A DMA controller device. The i8237A devices can be cascaded, as shown in the figure 'Cascaded i8237A DMA Controllers'. One i8237A device is the master and makes a direct 'hold request' to the MPU via the HRQ/HOLDA lines. The master's request inputs are called DREQ4..DREQ7, while the slave's request inputs are called DREQ0..DREQ3. The master's DREQ4 implements the cascading so this request input is not otherwise available.

The following list shows the DMA channel assignment for the PC/AT:

| DMA | Priority | Pre-defined Use in PC/AT | 8-bit or 16-bit |
|---|---|---|---|
| DREQ0 | Highest | Memory Refresh* | 8-bits |
| DREQ1 | | Not defined | 8-bits |
| DREQ2 | | Floppy Disk | 8-bits |
| DREQ3 | | Not defined | 8-bits |
| DREQ4 | | Cascade | not used |
| DREQ5 | | Not defined | 16-bits |
| DREQ6 | | Not defined | 16-bits |
| DREQ7 | Lowest | Not defined | 16-bits |

* Memory refresh does not use DMA in PC/AT designs, PC/XT only.

### DMA Channel Priority

With the cascaded DMA system in the PC/AT, DMA channel 0 (DREQ0) has the highest priority, with priority level descending to the lowest priority channel, DMA channel 7 (DREQ7). Note, when a DMA

transfer is in session, it cannot be 'interrupted' by another DMA request, even if the DMA request is made by a higher priority DMA channel. The current DMA transfer session will be allowed to complete before the pending DMA request is accepted.

### 8-Bit DMA Transfers

Some PC/XT products and the PC/AT products have 16-bit wide memory data buses. The 8-bit DMA channels are provided for the transfer of byte-wide data to and from 8-bit I/O devices, i.e. 8-bit adapter cards in the I/O Channel bus. The DMA controller provides byte addressing. Circuitry is required to decode the least significant bit from the address bus (A0) so that the 8-bit data can be 'steered' to/from the high (D8..D15) memory data byte for odd addresses and to/from the low memory data byte (D0..D7) for even addresses. For 32-bit wide memory data buses, the two least significant address bits (A0, A1) are decoded to select the appropriate byte to satisfy the byte addressing requirements.

### 16-Bit DMA Transfers

DMA channels 5 to 7 are stated to be 16-bit DMA channels. The i8237A is an 8-bit device, so how is the device used to transfer 16-bit data words to and from 16-bit adapter cards?

The i8237A is not required to store data internally during a DMA transfer. It does have a *Memory-to-Memory* transfer mode where it uses two DMA channels and internally buffers the data byte being transferred. But the *Memory-to-Memory* transfer mode is of little interest in the PC/AT design, the objective is to provide DMA transfer between I/O card and memory. Since the DMA controller does not need to buffer the data it can be 'fooled' into transferring 16-bit data words quite easily.

The 16-bit DMA channels provide word addressing by setting the address bus **A0** bit to 0 and shifting the i8237A's **A0..A15** bits left one bit, i.e. multiplying the address by two. This allows 128K bytes (64K words) to be transferred by the DMA controller.

**Addressing for 8-bit DMA in the PC/AT (24 address bits):**

| PAGE REGISTER (8) | DMA CONTROLLER (16) |
|---|---|
| $A_{23}$ .....$A_{16}$ | $A_{15}$.....$A_0$ |

64K bytes addressable per page.

**Addressing for 16-bit DMA in the PC/AT (24 address bits):**

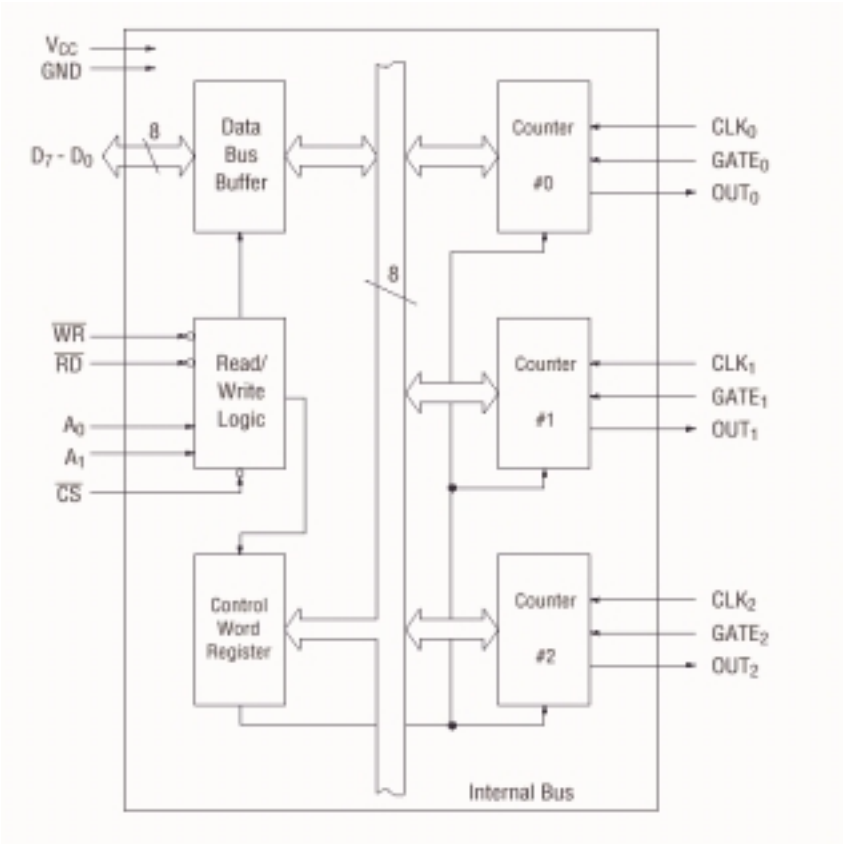
| PAGE REGISTER (7) | DMA CONTROLLER (16) | $A_0$=low |
|---|---|---|
| $A_{23}$ .....$A_{17}$ | $A_{16}$.....$A_1$ | $A_0$ |

64Kwords (128Kbytes) addressable per page

## The i8253/i8254 Timer Counter

The PC/AT design uses a timer/counter chip to provide hardware-based timer and counter functions. The i8253/i8254 is called a 'Programmable Interval Timer', or PIT for short. The i8253/i8254 device contains three identical, separate counter channels.

### i8253/i8254 Simplified Block diagram




**The i8253/i8254 in the PC/AT**

Each CLKx input is clocked at a frequency of: 1.193180 MHz. This is the PC processor, 4.77 MHz., divided by four.

**f** = 1.193180 MHz., **t** = 1/f = 0.838 micro secs.

The three counter channels are assigned as follows, in the PC/AT design:

Counter Channel 0: General System Timer
Counter Channel 1: Memory Refresh
Counter Channel 2: Audio Speaker

## The Parallel Printer Port

The parallel printer port on the PC/AT product is referred to as an LPT (Line Printer). The LPT port is designed to support the Centronics parallel printer interface standard. The Centronics standard uses a 36-pin connector but the PC/AT implementation uses just a 25-pin connector. Only a short cable between PC and peripheral is allowed.

The LPT port is often considered as a general-purpose parallel interface for connecting to a range of devices e.g. Dongle, A/D converter, D/A converter etc.

A number of LPT ports can be supported in a PC/AT design, typically up to four ports labelled: LPT1 to LPT4. The I/O addresses for LPT1 and LPT2 are defined as follows:

---

| | | LPT1 | LPT2 |
|---|---|---|---|
| **Data latch output** | **8 bits** | **378h** | **278h** |
| **Data Latch read back (verify)** | **8 bits** | **378h*** | **278h*** |
| **Printer controls output latch** | **4 bits+Int** | **37Ah** | **27Ah** |
| **Printer controls latch read back** | **4 bits+Int** | **37Ah**** | **27Ah**** |
| **Printer status, read** | **5 bits** | **379h** | **279h** |

* Not bi-directional as output latch is always enabled, but 'enhanced' designs support bi-directional read/write.

** Can be used as inputs if open-collector outputs are driven high.

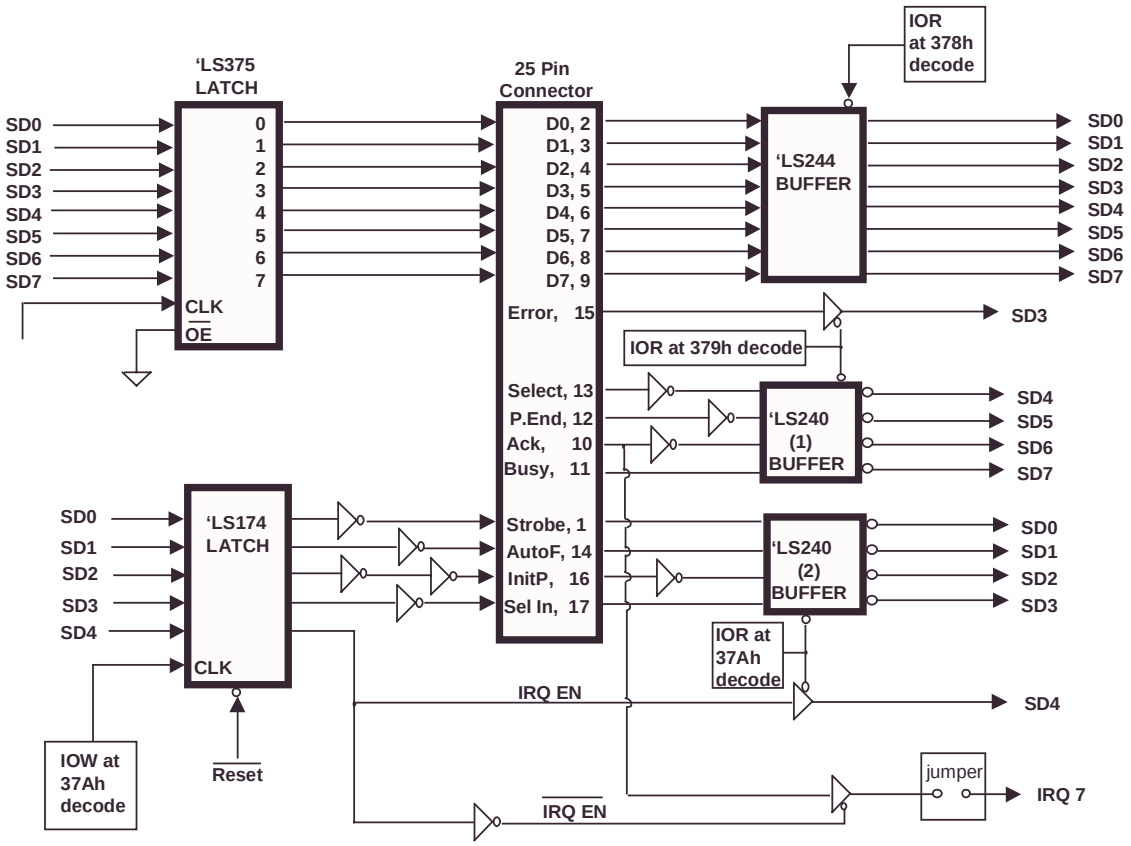
Base addresses for LPT3 and LPT4 can be found by examining the BIOS data area.

**IRQ**

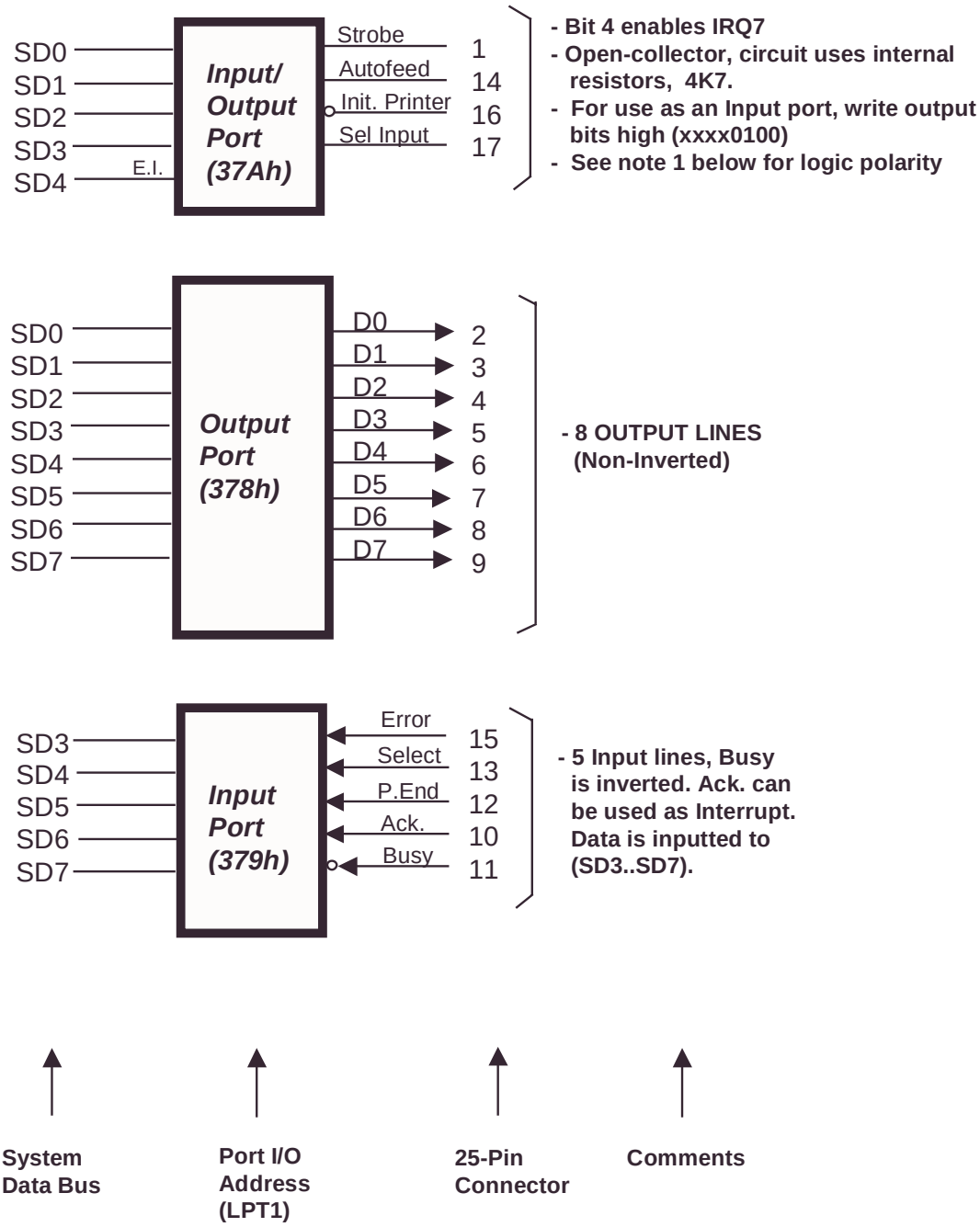
Writing a 1 bit to the LPT printer control port, bit 4, will enable the port IRQ. The status of the IRQ (IRQ EN) can be read back at bit 4 of the printer control read back input port.

Figure 'LPT1 Port Logic Diagram' shows the logic used on the original PC/AT design.

**LPT1 Port Logic**



The figure below summarises the I/O programming requirements.




***Note1: Port 37Ah*** ***- as OUTPUT Port all 4 lines are inverted, except pin 16***
***- as INPUT Port all lines are inverted, except pin 16***

**N.B.** Pins **18 to 25** are Ground pins on the 25-way connector

## Parallel Port Enhancements

The parallel port described above is the 'standard' as found on the original PC/AT. Most parallel ports exceed this 'standard'. In fact no official standard has ever been defined for the original PC Centronics Parallel Interface but some enhancements are summarised below:

### Bi-directional Type 1

Introduced with the IBM PS/2 (1987) this enhancement allowed the parallel port to operate in both directions – in and out. A status bit indicates which direction is set for data transfer. The port must be specifically configured for bi-directional data transfer. The default operation is uni-directional.

### Bi-directional, Type 3, DMA

Later PS/2 models supported the IBM defined 'Type 3' parallel port, which included DMA support. DMA enabled faster I/O transfers.

### Enhanced Parallel Port (EPP)

This is an Intel defined specification, also called fast Mode Parallel Port.

### Enhanced Compatibility Port (ECP)

Similar to EPP.

### The IEEE 1284 Standard

The IEEE 1284 Standard "Standard Signalling method for a bi-directional Parallel Peripheral Interface for Personal Computers" was created because there was no defined standard for bi-directional parallel communications between PCs and peripherals. Even today there is quite a wide range of implementations of the parallel port interface and incompatibilities are quite common. The elements of IEEE 1284 are:

- Adds standardised bi-directional capabilities to the "Centronics Parallel Interface"
- Multiple bi-directional operating modes
- Advanced operating mode can reach speeds of 2 to 4Mbyte/s (50 times the original rate)
- New electrical interface, cabling and connector for improved performance and reliability, while retaining backward compatibility
- The new IEEE1284 port uses a 36-pin connector and twisted pair cabling, while retaining backward compatibility – cable lengths can be up to 10m long
- Still not widely accepted in PC designs

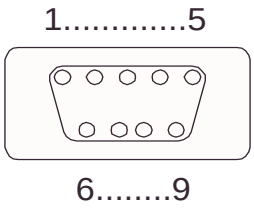
## The Serial Port

The PC/AT serial port is a programmable full-duplex asynchronous communications communication channel, based upon the EIA, RS-232-C communication's standard.

The RS-232-C standard defines a 25-way D-type physical connector. IBM, in an effort to reduce the size and cost of this connector implemented a sub-set of this standard and defined a 9-pin D-type connector which has now become a 'de-facto' standard for implementation of a *limited* serial, asynchronous communications port.

**Assignment of the 9 pins:**

| 9 Pin Conn. | 25 Pin Conn. | Signal Name |
|---|---|---|
| **1** | **8** | **DCD (data carrier detect)** |
| **2** | **3** | **RX Data (receive data)** |
| **3** | **2** | **TX Data (transmit data)** |
| **4** | **20** | **DTR (data terminal ready)** |
| **5** | **7** | **GND (signal ground)** |
| **6** | **6** | **DSR (data set ready)** |
| **7** | **4** | **RTS (request to send)** |
| **8** | **5** | **CTS (clear to send)** |
| **9** | **22** | **RI (ring indicator)** |
| **Shell** | **1, Shell** | **FG (frame ground)** |




### The 8250/16450 UART

[UART: Universal Asynchronous Receiver/Transmitter]

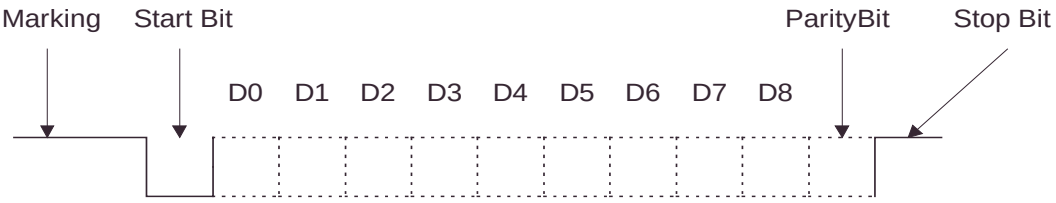
The UART is a programmable device where the communications channel parameters such as the baud rate; number of stops bits, parity bits, etc. can be programmed onto the chip.

### Note on Asynchronous Transmission

Since transmission is asynchronous, each transmitted character must be 'framed' such that the receiver can establish the starting bit and ending bit of each character.

- A single 'start bit' is used to denote the start of a character, i.e. the 'start bit'.

- Either 1, 1.5 or 2 'stop bits' can be defined.
- The character size can be 5, 7 or 8 bits long.
- Parity is optional and can be defined as 'no parity', 'even parity' or 'odd parity'.




## The Baud Rate

The baud rate is programmable. The baud rate generator is contained within the UART chip. An external 1.8432MHz clock is provided. An internal programmable divider circuit is used to select the desired baud rate.

The original PC/XT and PC/ AT products could be programmed for a maximum baud rate of 9,600 baud. Later the maximum baud rate became 19,2Kbaud. The UART chip can be directly programmed to run at a baud rate of up to 115.2KBaud but there is no guarantee that the transmission at this baud rate will be reliable, especially over long cable distances.

### RS-232 Level Shifters

The RS-232-C specification defines logic levels where a logic 0 is represented by a high voltage level, 3 to 15 volts, a *Mark* condition. A logic 1 is defined as a low voltage level, -3 to -15 volts, a *Space* condition. The PC design uses +12v and -12v voltage supplies and transmits +12v for logic 0 and -12v for logic 1, along the RS-232-C interface. RS-232-C voltage level shifters are used. Note these RS-232-C level shifters also invert the logic, converting from positive logic to negative logic.

### COM Ports

A number of serial ports can supported in the PC and are designated COM1, COM2 etc.
Each COM port has its own UART circuit, the 8250 (or similar 16450). The UART has a rich register set and the programmer via I/O ports can access these registers. The standard I/O port assignment for the COM1 and COM 2 ports are as follows:

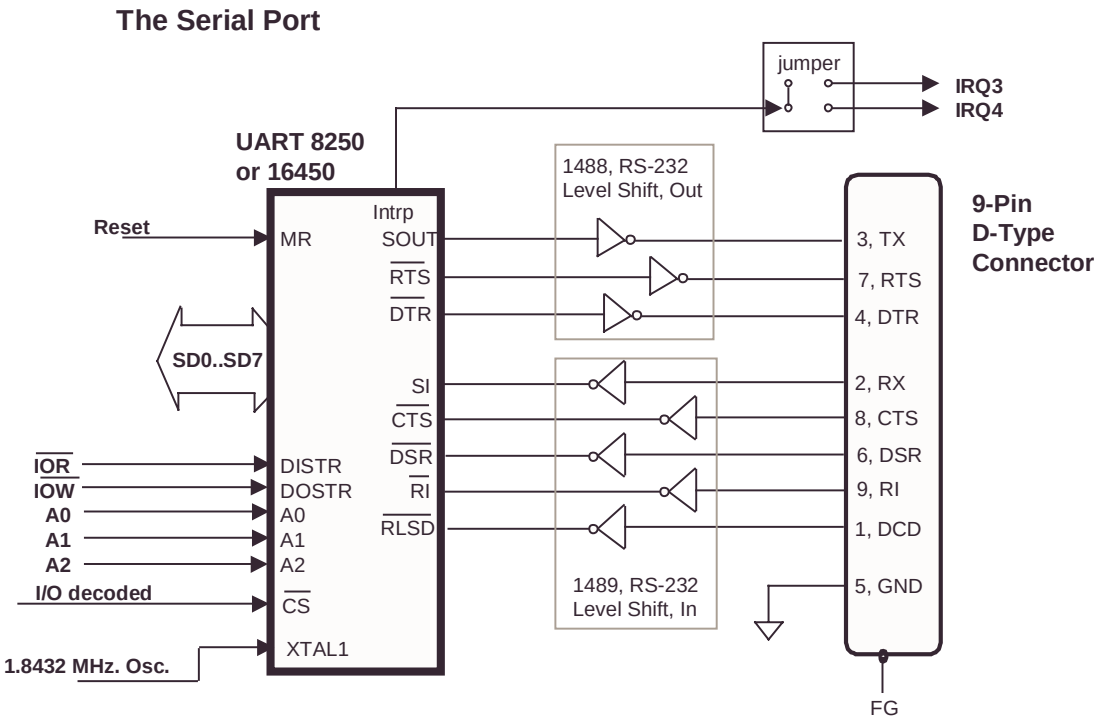
| | |
|---|---|
| **COM1** | **3F8h - 3FFh** |
| **COM2** | **2F8h - 2FFh** |

## PC UARTs

| | |
|---|---|
| 8250: | Used on original PC |
| 16450: | Used on original PC/AT |
| 16550: | Backwards compatible with earlier UARTs but includes a 16-byte buffer to improve performance |
| 16550A: | 16550 with bug fixes |

## The COM Port Circuit

The figure below shows a circuit diagram for the (Serial) COM port.




*DISTR, DOSTR = Data Input and Data Output Strobes.*

## NMI Logic

The PC/XT product has three NMI interrupt sources:

- Memory Parity Error
- I/O Channel Check
- 8087 FPU

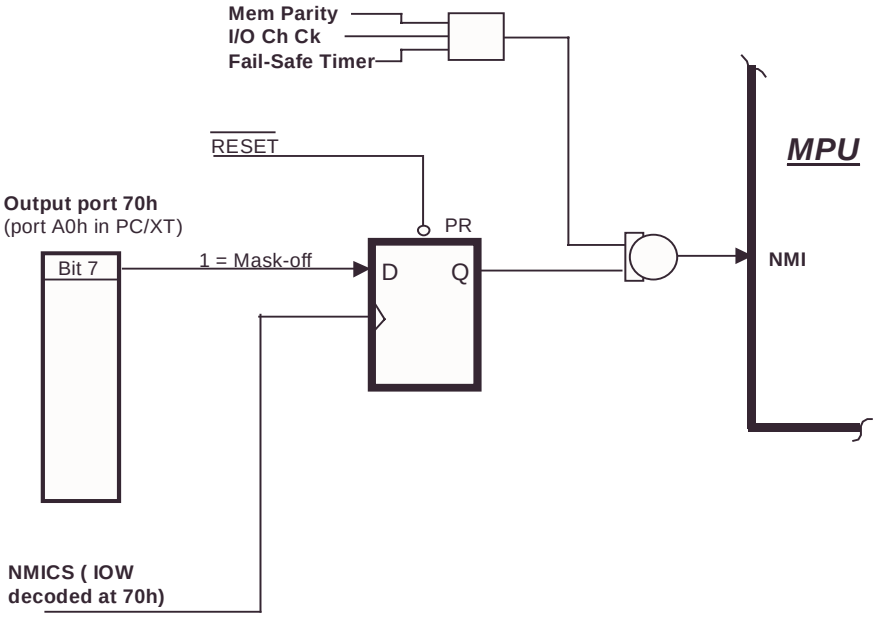
The PC/AT has the following three NMI interrupt sources:

- Memory Parity Error
- I/O Channel Check
- Fail-safe Timer (design option)

The NMI interrupt handler routine will poll the devices to establish the source of interrupt. It is important to be able to 'mask off' the interrupt source while one NMI interrupt is being serviced. It is also important to be able to 'mask off' the NMI during the boot-up process, as any one of the three devices would inhibit the boot process if the interrupt request was activated before the boot process was completed. The NMI interrupt can be disabled by clearing the NMI mask bit, and enabled by setting the NMI mask bit.

A circuit to implement the 'masking' for the non-maskable interrupt is shown in figure 'Masking the Non-Maskable'

**Masking the Non-Maskable**




## THE PC/AT KEYBOARD CONTROLLER AND INTERFACE

A typical PC/AT keyboard has 102 keys. The keyboard logic design is based on a small microcontroller, typically an 8049 microcontroller. The 8049 has its own internal ROM (2KBytes), RAM (128 Bytes) and I/O ports.

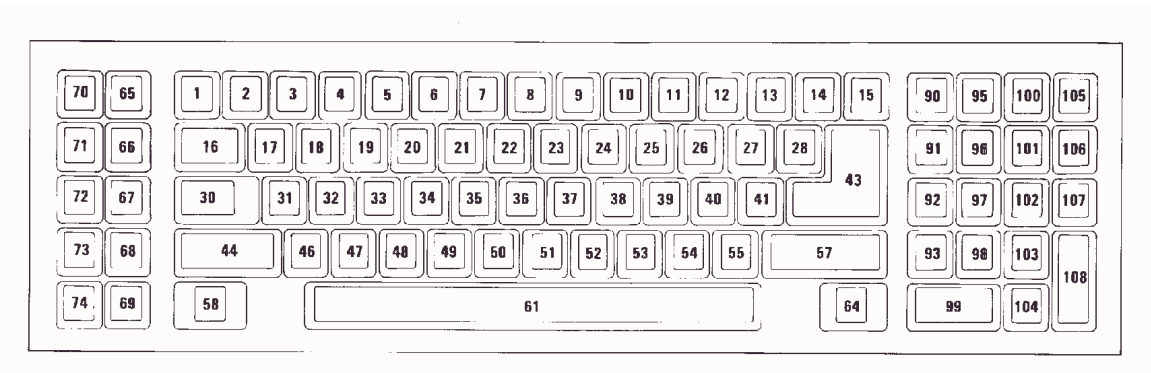


**Figure: Keyboard Circuit**

Figure 'The Keyboard Circuit' shows the circuitry for a typical 102 key keyboard.

The keyboard matrix shown is a matrix of simple contact switches. The microcontroller will continuously scan the matrix to 'see' if a user has pressed (or released) a key. The scanning is achieved by activating (active low) each column in sequence, by counting through the 74154's 'ABCD' inputs. When a column is selected (low) the matrix rows (D0..D7) are read as a single byte. If any bit within the byte is low then a 'key press' is recorded within the microcontroller's RAM memory. Referring to the 'Keyboard Matrix' section of the diagram, you will see that each key contact switch has a series diode. These diodes are used to avoid problems when more than one key is pressed at the same time. In the diagram consider what would happen if both keys within the magnified area were pressed at the same time. If the diodes did not exist, when column 'y' is activated (low), column 'x' is high, then effectively a 'short-circuit' is placed across 'x' and 'y' rendering an indeterminate logic level being read on row 'a'.

The microcontroller also provides the key 'de-bounce' function in software.

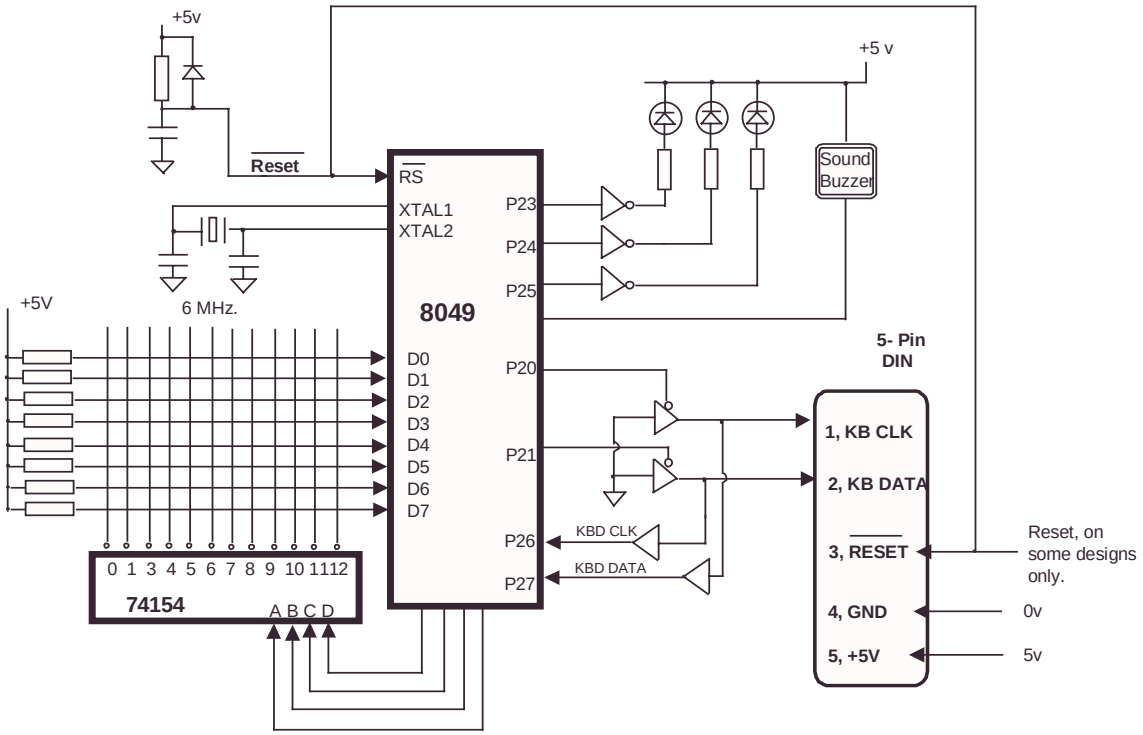
SCAN CODES

The microcontroller will record when a key is pressed, the *MAKE* condition, and when a key is released, the *BREAK* condition.

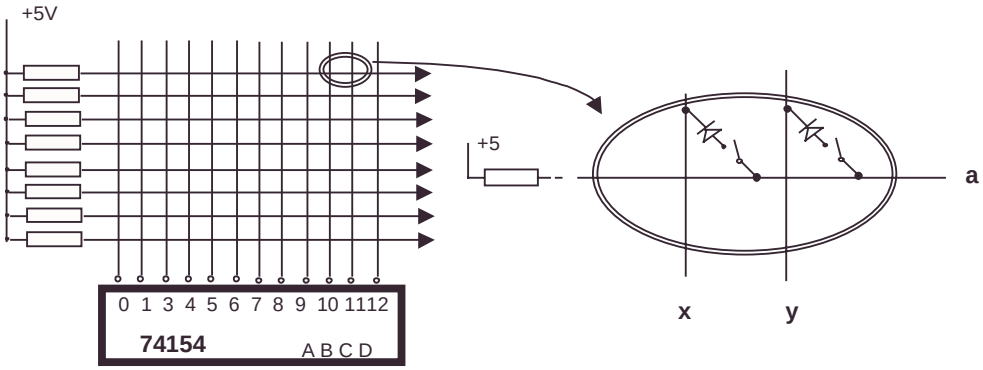
Each key has its own unique SCAN CODE, an 8-bit code that uniquely identifies each key.

For a *MAKE* condition the MSB of the key's SCAN CODE is set to 0. For a *BREAK* condition the MSB bit of the key's SCAN CODE is set to 1. Thus for a given key, if the MSB bit of the SCAN CODE is 1 there exists a *BREAK* condition, and if the MSB bit is 0 there exists a *MAKE* condition.

**The Keyboard Circuit**




**The Keyboard Matrix**




The SCAN CODE is sent along the keyboard's interface to the computer.

E.g.: The SCAN CODE for the key 'C' is 46d (2Eh) on a 102 key keyboard.

When 'C' is pressed the SCAN CODE code **2Eh** (00101110) is sent to the computer.
When 'C' is released the code **AEh** (**1**0101110) is sent.

It is the keyboard software driver, which assigns the right 'ASCII' code to the received SCAN CODE. The driver will, for example, check whether CAPS-LOCK or the SHIFT key was pressed (*MAKE* condition) to know whether to assign an upper case or lower case character to the received SCAN CODE. In fact a foreign language keyboard layout may assign some other character to the key, which we might define as the 'C' position.

Since the microcontroller can 'remember' that a key has been pressed and not yet released we can have useful key combinations representing defined functions e.g. the well known key combination **'CTRL, ALT, DEL'** etc.

**Some Other Features:**

The 8049 microcontroller drives the keyboards LEDs (**Num Lock, Caps Lock, Scroll Lock**) and the 'buzzer' through buffer circuits, as shown in the diagram. The 8049 uses an external crystal for its clock oscillator. The bi-directional communication is achieved by the control of tri-state buffers. Note, the communication is bi-directional but is **half-duplex**, i.e. data can travel in one direction only, at a time.

The keyboard connects to the computer via a 5-pin DIN connector:

| | |
|---|---|
| **1** | **Clock** |
| **2** | **Data** |
| **3** | **Spare (most often used for reset)** |
| **4** | **0v** |
| **5** | **+5v** |
| **Shield** | **Shield Ground** |

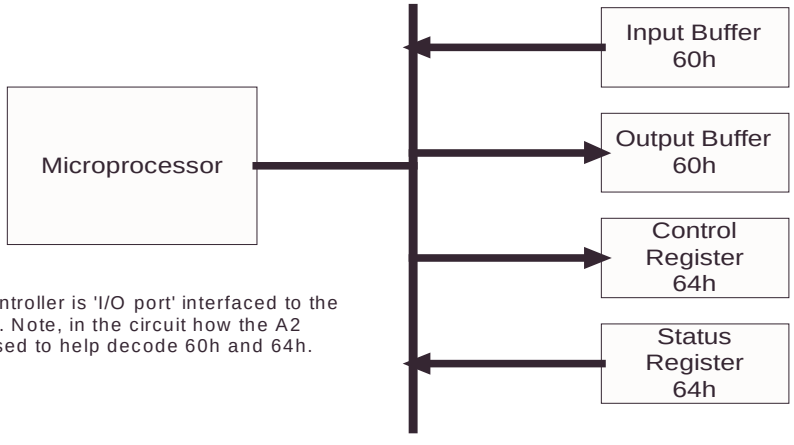
Communication between the keyboard and the PC is synchronous (data is clocked) and the data is framed similar to the asynchronous format:

| | |
|---|---|
| **Start bits** | **1** |
| **Data bits** | **8** |
| **Parity** | **Yes** |
| **Stop bits** | **1** |

## The PC/AT's Keyboard Controller

The keyboard controller logic is based on a microcontroller circuit, typically an 8042 (8742 is EPROM version) microcontroller device is used.

The 8042 microcontroller has its own internal ROM (2 Kbytes), RAM (1 Kbytes) and I/O ports. The device is interfaced to the MPU's I/O logic and is programmed to provide synchronous bi-directional serial communication via some of its I/O port bits.




The keyboard controller is 'I/O port' interfaced to the PC/AT as shown. Note, in the circuit how the A2 address line is used to help decode 60h and 64h.

## Keyboard Buffer

A 16-character keyboard buffer area is maintained in the MPU's RAM, within the BIOS data area. Each keyboard character is stored as two data bytes:

**High byte..... ASCII Character code**
**Low Byte..... Key SCAN code**

Thus, the sixteen characters are stored in a 32 byte buffer, from 40:1Eh to 40:3Dh.

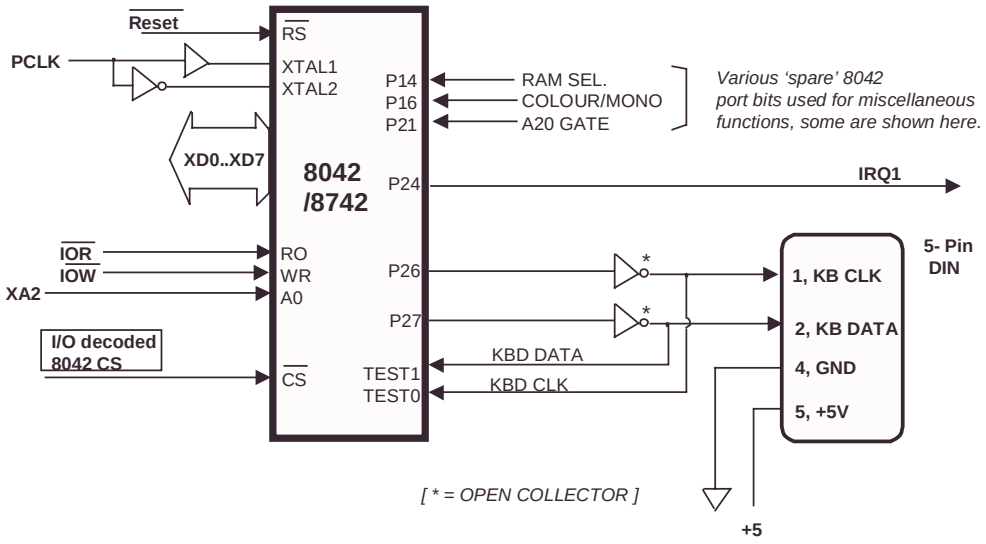
## Keyboard Interrupts

When a key is pressed (or released) IRQ1 is activated which causes a vector to INT 9. The INT 9 handler sends the scan code to the keyboard buffer in the BIOS data area.

The BIOS can access the keyboard IN/OUT functions to read scan code, check KB buffer, check status of Shift and Control keys etc. using the INT 16h BIOS interrupt.

INT 1Bh is the Keyboard Break interrupt. When the INT handler 'sees' the CTRL BREAK code it re-vectors to the INT 1Bh handler.

**The Key Board Controller**




*DISTR, DOSTR = Data Input and Data Output Strobes.*

## *THE PC/AT I/O MAP - Standard Assignments*

The PC/AT standard I/O Map assignments are shown below.

| Hex Range | Device |
|---|---|
| 000 - 01F | DMA controller 1, i8237A |
| 020 - 03F | Interrupt controller 1, i8259, master |
| 040 - 05F | Timer, i8253/i8254 |
| 060 - 06F | 8042 Keyboard controller |
| 070 - 07F | Real-time clock, NMI mask |
| 080 - 09F | DMA registers |
| 0A0 - 0BF | Interrupt controller 2, i8259A, slave |
| 0C0 - 0DF | DMA controller 2, i8237A |
| 0F0 | Clear math coprocessor busy |
| 0F1 | Reset math coprocessor |
| 0F8 - 0FF | Math coprocessor |
| IF0 - IF8 | Fixed disk |
| 200 - 207 | Game 1/0 |
| 278 - 27F | Parallel printer port 2 (LPT 2) |
| 2F8 - 2FF | Serial port 2, (COM 2 ) |
| 300 - 31F | Prototype card |
| 360 - 36F | Reserved |
| 378 - 37F | Parallel printer port 1 (LPT 1) |
| 380 - 38F | SDLC, bisynchronous 2 |
| 3A0 - 3AF | Bisynchronous 1 |
| 3B0 - 3BF | Monochrome display/printer adapter |
| 3C0 - 3CF | Reserved |
| 3D0 - 3DF | Colour/graphics monitor adapter |
| 3F0 - 3F7 | Diskette controller |
| 3F8 - 3FF | Serial port 1 (COM 1) |