

Computer Organization (Input-output)

KR Chowdhary
Professor & Head
Email: kr.chowdhary@gmail.com
webpage: krchowdhary.com

Department of Computer Science and Engineering
MBM Engineering College, Jodhpur

November 14, 2013

Introduction to IO

- ▶ IO of computer makes it possible to have interaction with the outside world. IO are the devices outside the CPU and memory, to which CPU/memory communicate to get the information/data/programs into computer as well as send these outside.
- ▶ IO devices are: Keyboard, mouse, monitor(computer display), hard-disk, CD ROM, pendrive, printer, network card, etc.
- ▶ Due to the large variation in speed, IO devices cannot be connected to the bus directly.

Typical speeds are: keyboard: 5-10 char per sec., printer 100s - 1000s char per sec., hard disk: millions of char (bytes per sec.), etc. If CPU is designed to interact with devices directly through the bus, its complexity increases excessively.

- ▶ Different IO devices store the data in different formats. Consequently, they are connected to the bus via IO controllers (called IO modules).
- ▶ One side of these controllers is connected to the bus and other to device.

Synchronous and Asynchronous I/O

- ▶ **Asynchronous(non-blocking) I/O:** is a form of I/O processing that permits other processing to continue before the transmission has finished.
- ▶ I/O operations on a computer can be extremely slow compared to the processing of data. An I/O device can incorporate mechanical devices that must physically move, such as a hard drive seeking a track to read or write; this is often far slower than the switching of electric current. For example, during a disk operation that takes ten

milliseconds to read/write, a processor that is clocked at 1 GHz could have performed ten million instruction-processing cycles. (These approach are either **polled or interrupt driven**)

- ▶ A simpler approach to I/O would be to start the IO and then wait for it to complete, called (**synchronous or blocking I/O**), would block the execution of instructions while the communication (data transfer) is in progress, leaving system resources idle (For example in**DMA**)

Asynchronous data transfer

If registers in the interface share common clock with CPU register, then traffic between the two is synchronous.

- ▶ Asynchronous data transfer takes place between two independent units.
- ▶ One way of achieving this is by means of **strobe pulse** to indicate the intention of data transfer.
- ▶ In below, CPU is data source and initiator of strobe.

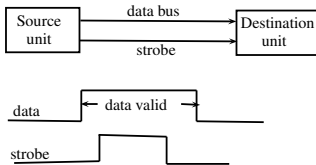


Figure 1: Timing diagram(TD) for memory write (source initiated strobe)

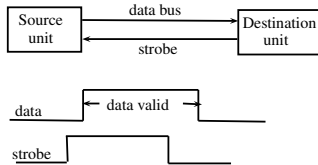


Figure 2: TD for memory read (destination initiated strobe)

- ▶ In above CPU (data destination) initiates strobe, & memory releases data.
- ▶ In both cases, source unit has no way to know that destination has received the data, and destination has no way to know that source has sent the data.

Asynchronous data transfer: Hand shake

The data transfer between an interface and I/O device is commonly controlled by a set of handshaking lines.

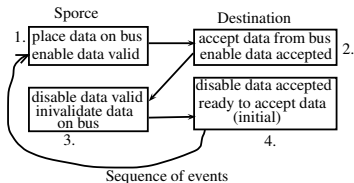
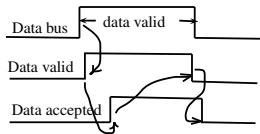
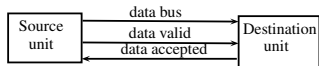


Figure 3: Source initiated transfer

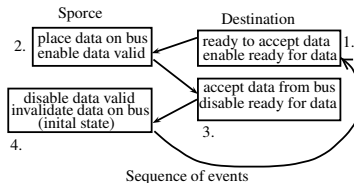
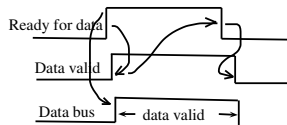
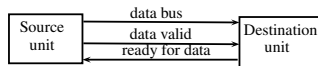


Figure 4: Destination initiated transfer

- ▶ Hand shaking scheme provides a high degree of flexibility and reliability (the successful data transfer relies on the active participation of both parties)
- ▶ If one unit is faulty, data transfer cannot be initiated and completed. Such error can be detected by time out mechanism.

- ▶ Devices are connected to the bus through I/O controller (I/O Module).

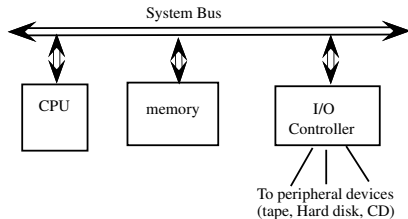


Figure 5: Connection of I/O devices and I/O controller

I/O controllers functional units

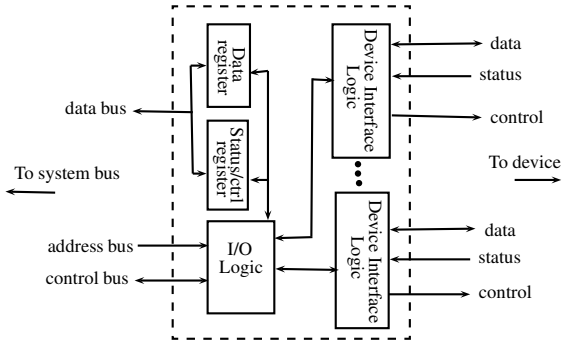


Figure 6: I/O device Interface

- Functions performed I/O controller are:
 1. Control and timing of data read/write
 2. Communication with processor and devices
 3. Data buffering (to handle speed mismatch)
 4. Error detection

Typical I/O Interface unit

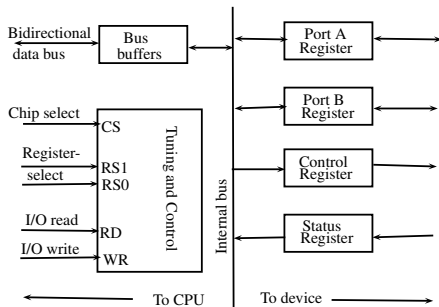


Figure 7: IO device Interface

| CS | RS1 | RS0 | Register selected |
|----|-----|-----|----------------------------------|
| 0 | x | x | None: data bus in high impedance |
| 1 | 0 | 0 | Port A register |
| 1 | 0 | 1 | Port B register |
| 1 | 1 | 0 | Control register |
| 1 | 1 | 1 | Status register |

Detailed functions of I/O controller

- ▶ **Control and timing of data R/W:** Coordinate the flow of traffic between internal resources and external devices.

1. The processor interrogates the I/O controller to check the status of the attached device.
2. The I/O controller returns the device status.
3. If device is ready to transmit, processor requests transfer of data by means of a command to I/O controller.
4. I/O controller obtains a unit of data (e.g., 8 or 16 bits) from the external device.

5. The data are transferred from I/O controller to the processor.

- ▶ **Processor communication involves:**

1. Command decoding: I/O controller (for disk) accepts commands from processor: READ SECTOR, WRITE SECTOR, SEEK track number, etc.
2. Data are exchanged between processor and I/O controller
3. Status reporting: BUSY and READY.
4. Address recognition: I/O controller must recognize address of each peripheral.

- ▶ **Data buffering:** (handles speed mismatch) Buffering makes possible the communication at the speed of CPU/ memory/ device.
- ▶ I/O controller responsible for **error detection and reporting to processor:** paper jam, bad disk track, changes to the bit pattern (parity check)
- ▶ **IO Channel/Processor:** An I/O controller taking most of the detailed processing burden, presenting a high-level interface to the processor, and used on mainframes.
- ▶ I/O Operations: Control, test, read, write.

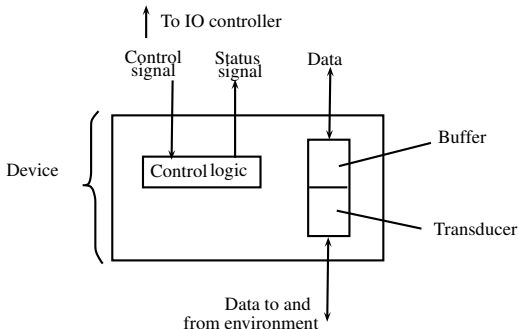


Figure 8: I/O device with Interfaces

- ▶ The control provide communication between controller and device
- ▶ Environment is magnetic surface in the case of hard disk, optical surface in case of CD-ROM.

Types of I/O (or Modes of Transfer)

► Polled I/O (or programmed I/O)

The cpu polls the device continuously after some interval, whether the device wants to transfer the data. If yes, the cpu transfer a byte to or receives a byte from it.

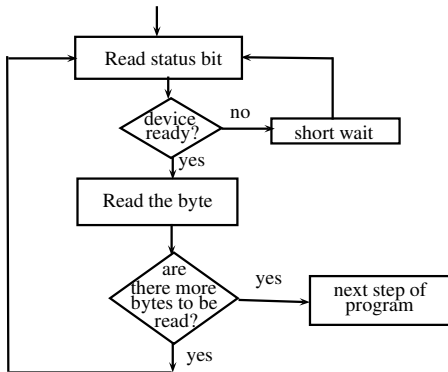


Figure 9: Read operation from device in polled I/O.

Polled I/O write operation

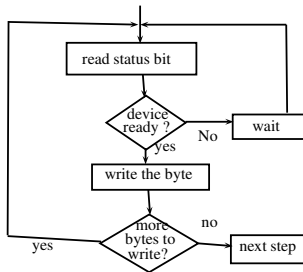


Figure 10: Write operation from device in polled I/O.

- ▶ CPU reads the status bit, if it is set (indicating that device is ready), the byte is written by CPU to I/O, else it waits for some time and again tries.
- ▶ **Is it efficient method?**

Polled I/O: Memory-mapped v/s Isolated I/O

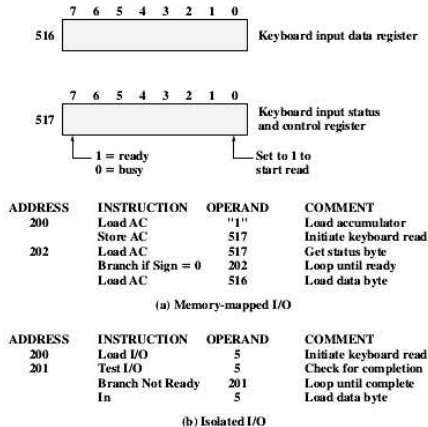


Figure 11: Memory-mapped and Isolated I/O.

Polled I/O: for 8085

- ▶ Repeatedly tests the status of I/O device, hence it wastes the cpu time. However, its architecture is simple to implement.

Example: busy waiting then input

```
wait: IN 1 ; read io device status
      CPI ready ; if ready, set Z=1, else set z=0
      JNZ wait ; io device waiting
      IN 2 ; read data into accumulator
```

;Transfer a block in 8085

```
LXI H, 10
MVI B, 100
loop: IN 7 ; read from port 7
      MOV M, A
      INX H
      DCR B
      JNZ loop
```


Memory Mapped v/s Isolated I/O

- ▶ I/O devices are connected through I/O module or I/O controller to bus.
- ▶ Each I/O port of the module has an address. If that is address of device, it is **Isolated I/O**. Alternatively, address space of I/O can be treated as memory locations. Thus, address space of memory gets reduced by the magnitude I/O address. This is **Memory mapped I/O**.
- ▶ In isolated I/O only IN addr /OUT addr instructions exist.

For 8-bit I/O address total 256 I/O addresses exists in 8085.

- ▶ In memory mapped IO all the memory reference instruction can be executed for IO ports: *STA addr, LDA addr, MOV A, M; MOV B, M; MOV M, C; ADD M; ADI M; ORA M; ORI M; SUB M; SBI M* (8085 processor). Hence, memory mapped IO is more flexible to use compared to Isolated IO. Now, at least in theory, entire memory can space can be used as IO address space.

Interrupts driven I/O

Interrupt driven I/O

- ▶ The program controlled I/O degrades the system performance as the cpu gets tied down to the I/O.
 - ▶ The interrupt mechanism greatly improves the performance of the CPU.
 - ▶ In interrupt driven IO, timing of I/O is controlled by the device, and the cpu remains occupied in its own job for rest of the time.
 - ▶ IO device interrupts the CPU when IO is required. On this CPU saves its status including the PC, and control is transferred to an ISR (interrupt service routine), which performs I/O. On return from ISR, CPU resumes at its previous operation.
- ▶ CPU gets an interrupt when, for example,
 - a character is entered on keyboard,
 - monitor is ready for next refresh
 - a block transfer complete from memory to I/O or I/O to memory
 - HW interrupts are due to division by zero, or an attempt to execute a privileged instruction by user program.

Classes of Interrupts:

1. **Program:** Generated by some conditions which occurs as a result of instruction execution:
 - ▶ arithmetic over flow
 - ▶ divide by zero
 - ▶ attempt to execute illegal machine instruction
 - ▶ reference to outside user's allowed memory space
2. **Timer Interrupt:** generated by timer of processor; allows to perform certain functions at regular intervals
3. **IO:** Interrupt generated by IO controller to: signal normal

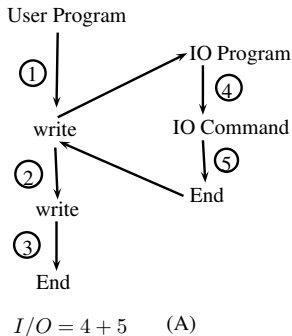
completion/start of an IO, or to send variety of error conditions

4. **HW failure:** Power failure, memory parity error.

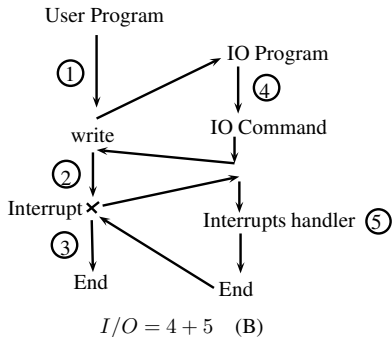
Advantages of Interrupts

- ▶ Improves processing efficiency due to slow IO and fast CPU
- ▶ User program does not need any special code for interrupt
- ▶ Processor and OS are responsible to suspend the program, and cause it to return back

Interrupt Processing



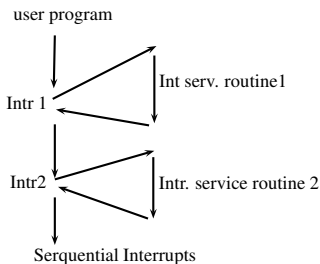
- ▶ User program waits till the interrupt is processed and control returns back to it.
- ▶ 4 = save context, 5 Interrupt handler + context restore.
- ▶ (A): Control returns back immediately to user program



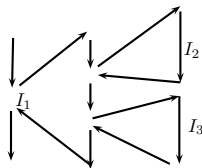
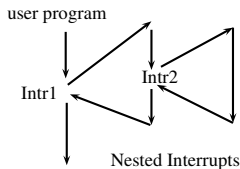
- ▶ (B): user program is called on the completion of processing of interrupt handler.
- ▶ What is difference between two?

Handling Multiple Interrupts

- ▶ Program may be receiving data from communication line, and send them to printer. So communication line will cause interrupt, as well as the printer.
- ▶ Two approaches to handle multiple interrupts: 1) **Disable interrupts**, 2) **Priority interrupts**



I_1 : Printer int, I_2 : communication serv. Int, I_3 : Disk intr



Working of Interrupt

- ▶ Interrupt request pending, if any, is tested by cpu after execution of every instruction. If pending, it is serviced and cpu resumes normal execution, else cpu continue with the next instruction.

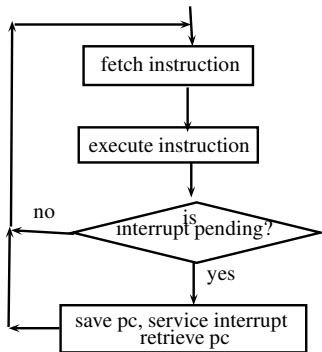


Figure 12: Interrupt servicing

Working of interrupt...

▶ Steps:

1. CPU identifies the source of interrupt (may require polling of I/O device)
2. CPU obtains address of ISR. (may be supplied by device along with Interrupt request)
3. PC, CPU status saved
4. PC loaded with ISR address

- ▶ Usually, the ISR has DI instruction at its beginning and EI (Enable intr.) at its end, followed by return

```
ISR: DI
      high priority ISR
      EI
      RET
```

▶ **Interrupt selection/Device Identification:**

1. Multiple interrupt lines (limits interrupts)
2. Software polling (test I/O, read adr register): time consuming
3. Vectored Interrupt (daisy chaining/HW polling:)

Working of interrupt

1. Multi-line interrupt:

- Called multi-level interrupt. No need of HW or SW to scan ports.
- Unless some other technique is used, CPU may have to execute the program that fetches the ISR address. Can be eliminated by vectored technique. (fixed priority)

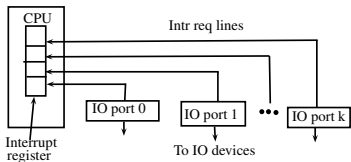


Figure 13: Multi-line interrupts

2. Single-line interrupt system:

- All the interrupts are ORed together. An interrupt from any device will set the interrupt flag in CPU. On knowing that, CPU determines source of interrupt. (programmable priority?)

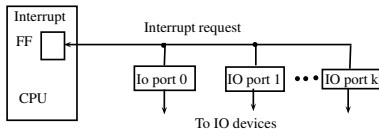


Figure 14: Single-line interrupt

SW poll and interrupt vectoring

- ▶ **Software poll:** On detecting interrupt, the processor branches to a service routine, which polls IO modules to determine who has interrupted, and then serves ISR of that.
 - Disadv: time wasted in polling.
- ▶ **Daisy Chaining:**
 1. Hardware polled. All IO modules share a common interrupt line and Int. ack. like is daisy chained through these modules.
 2. Requesting module places a word on the data bus (address of IO module - called vector)
 3. Vector is used a pointer to ISR (called vectored interrupt)

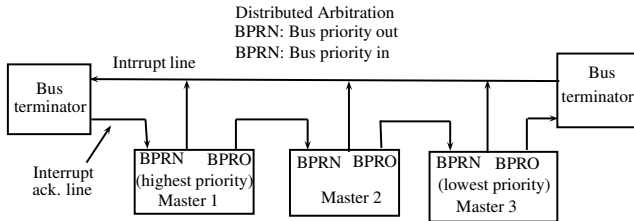


Figure 15: Vectored interrupt

Vectored Interrupt

- ▶ The device sets $INT=1$ when it wants to cause the interrupt
- ▶ Interrupt vector: An 8-bit signal for device to identify itself, which is used as an entry into a interrupt vector table to get the starting address of the ISR (Interrupt service routine).

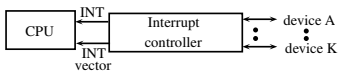


Figure 16: Vectored interrupt

- ▶ Most flexible and fastest response to interrupt is there when interrupt request causes direct HW implemented transition to current interrupt handling program.
- ▶ This requires that interrupting device supply to cpu the starting address or transfer vector of that program. The technique is called vectoring.

Vectored Interrupt contd.

- ▶ In the figure shown below, the interrupt vector is supplied by the device itself via the data bus
- ▶ Each I/O port may request the services of many different programs.
- ▶ Address on data bus modifies PC. Takes control of data bus temporarily. Alternatively send instruction *call x*

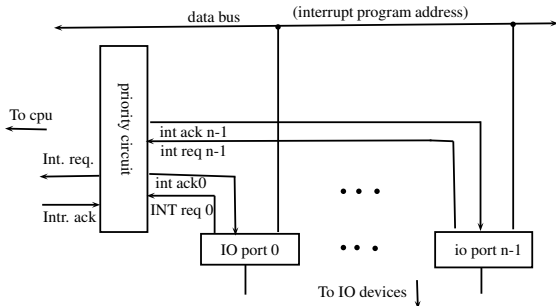


Figure 17: Another implementation of vectored interrupt

Vectored Interrupt with masking

- ▶ The k masked interrupt signals are fed into a priority encoder that produces a $\lceil \log_2 k \rceil$ -bit address, which is then inserted into program counter as a sub-field.

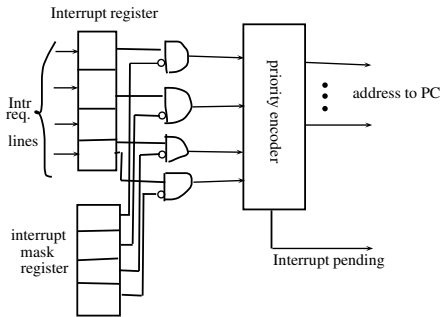


Figure 18: Intr. vectoring with masking of interrupts.

Vectored Interrupt ...

- ▶ When int req. from port i is received, priority encoder generates 2-bit address, which is inserted into PC. Rest of the bits of PC are zeroed.
- ▶ Thus, 2 bit will generate addresses: 0-3. This is multiplied by 4 to get (0, 4, 8, 12) as the position of interrupt vectors.
- ▶ Fig. 19 shows that first four locations (words, each 32 bits) are assigned to interrupt vectors.

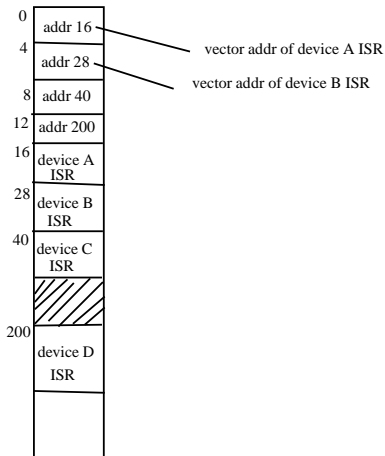


Figure 19: Interrupt vectors are stored in memory

IO controller

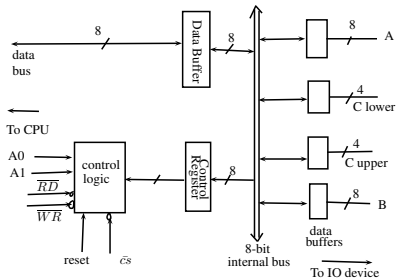


Figure 20: IO controller (intel 8255)/ PPI-programmable peripheral Interface.

- ▶ The Intel 8255 is called IO controller/IO module. There are three IO ports: A, B, C.
- ▶ The port C can be used as two 4-bit ports or one 8-bit ports. A, B are always 8-bit ports. All these ports can be programmed as Input, output, bidirectional in polled mode or as interrupt mode.
- ▶ Programming is done by writing a 8-bit control word at port address 11_2 . In polled mode certain lines can be used for handshaking.

- ▶ Following tables show the direction of ports:

| A_0 | A_1 | \overline{RD} | \overline{WR} | \overline{CS} | Direction |
|-------|-------|-----------------|-----------------|-----------------|-------------------------------|
| 0 | 0 | 0 | 1 | 0 | $A \rightarrow databus$ |
| 0 | 1 | 0 | 1 | 0 | $B \rightarrow databus$ |
| 1 | 0 | 0 | 1 | 0 | $C \rightarrow databus$ |
| 0 | 0 | 1 | 0 | 0 | $databus \rightarrow A$ |
| 0 | 1 | 1 | 0 | 0 | $databus \rightarrow B$ |
| 1 | 0 | 1 | 0 | 0 | $databus \rightarrow C$ |
| 1 | 1 | 1 | 0 | 0 | $databus \rightarrow control$ |

- ▶ Three are 3-modes of operation for PPI 8255:

mode 0: Basic input/output, mode 1: strobed I/O; mode 2: bidirectional bus

- ▶ various bits of control word:

$D_7 = 1 \Rightarrow$ mode set flag, 1=active.

$D_6D_5 = 00 \Rightarrow mode\ 0, 01 \Rightarrow mode\ 1, 1x \Rightarrow mode\ 2$

$D_4 = 1 \Rightarrow$ port A I/P, 0: O/P,

D_3 for port C upper, D_2 for mode 0 & 1, D_1 for port B, D_0 for port C lower. Any bit can be set or reset.

Interrupt Controller

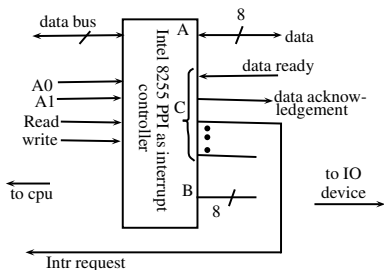


Figure 21: Programmable peripheral Interface as Interrupt Controller.

- ▶ 8255 chip is used as interrupt controller. The port C is used as interrupt port for strobing. Port A is used as data port for data input or output.
- ▶ Interrupt signal is stored in cpu register which is periodically tested by the cpu. Usually, the interrupts are assigned priorities based on the priorities of IO devices or services they are performing.

DMA(Direct Memory Access)

- ▶ CPU relinquishes the bus, and get itself isolated. The IO takes place between memory and IO device at clock speed. When I/O is complete, DMA controller removes the bus request line, CPU takes over the bus, and processing resumes at the point it was left.
- ▶ IO transfers are limited by the speed by which the CPU can

test and service IO.

- ▶ The testing IO status and executing IO commands can be better used for processing tasks.
- ▶ DMA request by IO device is for demand of BUS and not CPU (in interrupt it is reverse).
- ▶ The DMA request can be granted BUS at the end of any **CPU Cycles**(fig.)

Introduction to DMA

- ▶ **DMA v/s Interrupts:**
 - DMA break points are after each of following to opcode fetch, decode opcode, fetch operand (if any), execute instruction. However, the interrupt break point is after instruction is executed, and not in between.
- ▶ Why there is difference in break points of these?

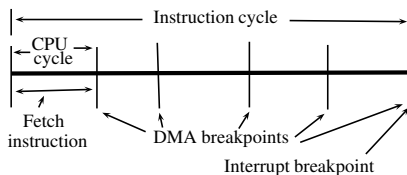


Figure 22: DMA and Instruction breakpoints during an instruction cycle.

Functional blocks of DMA

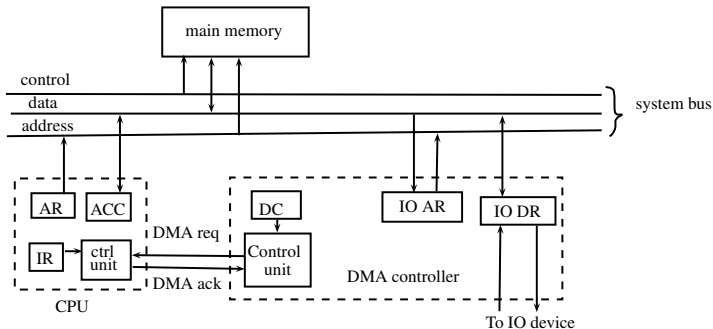


Figure 23: DMA controller block diagram.

- ▶ Functional blocks: DC (data count) keeps initial count of no. of words /bytes to be transferred, IOAR: is memory address from that location onward data is to be transferred to/from memory, IODR: IO data reg., for holding word/byte while it is being transferred to IO/RAM.

Working of DMA

- ▶ DMA is used for bulk data transfer between memory and IO.
- ▶ The cpu relinquishes the control of the bus, and surrenders it to DMA for doing this data transfer
- ▶ The transfer is initiated by CPU, and DMA controller interrupts the cpu to indicate that DMA is over
- ▶ DMA steps:
 1. CPU executes two IO instructions which loads IOAR and DC. (IOAR is base addr of main memory, DC = words count to be transferred)
 2. DMA controller gives bus request when ready to transfer to data. On this DMA acknowledges (grants the bus to DMA controller) (bus priority control is used when request are too many)
 3. DMA controller transfers the data.
 4. If IO device is not ready but $DC > 0$, the DMA controller deactivates the DMA request. On this CPU acknowledges bus grant low, and resumes normal operation.

Classification of DMAs

▶ DMA types:

1. Block transfer DMA
2. Cycle stealing DMA (bus cycles are stolen by DMA, during the time cpu is inactive, to carry out DMA)

▶ DMA block Transfer:

- Data of arbitrary length are transferred in a single continuous burst.
- DMA controller is bus master
- It is used when secondary memory devices are mag. disk, and cannot be stopped or slowed down without loss of data.
- Supports maximum transfer rate
- CPU has to remain inactive for long period

▶ Cycle stealing DMA:

- allows DMA controller to use system bus to transfer one or several words/bytes, and returns control back to CPU
- long strings are split
- reduces the DMA speed but also the interference of DMA to CPU





1. Indicate whether the following constitutes a control, status, or data transfer commands:
 - 1.1 Skip next instruction if flag is set
 - 1.2 seek a given record from a magnetic disk
 - 1.3 check if IO device is ready
 - 1.4 Move printer paper to beginning of next page
 - 1.5 Read interface status register
2. Why does the DMA has priority over CPU when both request a memory transfer?
3. How many 8-bit characters can be transmitted per second over a 9600 baud serial communication link using asynchronous mode of transmission with one start bit, eight data bits, two stop bits, and one parity bit?
(a) 600 (b) 800 (c) 876 (d) 1200
4. A DMA module is transferring characters to memory using cycle stealing, from the device transmitting at 14400 bps. The processor is fetching instructions at the rate of 1 million instructions per seconds (1 MIPS). By how much will the processor be slowed down due to due to the DMA activity?

5. Consider the system in which bus cycles takes 500 nsec. Transfer of bus control in either direction from processor to I/O device or vice-versa, takes 250 nsec. One of the I/O devices has a data transfer rate of 75 KB/s and employs DMA. Data are transferred one byte at a time.
 - 5.1 Suppose we employ DMA in a burst mode. That is, the DMA interface gains bus mastership prior to the start of a block transfer and maintains control of the bus until the whole block is transferred. How long would the device tie up the bus when transferred a block of 256 bytes?
 - 5.2 Repeat the above for cycle stealing mode.
6. An asynchronous link between two computers uses the start-stop scheme, with one start bit and one stop bit, and transmission rate of 38.8 kilobits per sec. What is the effective transmission rate as seen by the two computers?

7. A DMA controller serves four receive only telecommunications links (one DMA per channel) having speed of 64 kbps each.
 - 7.1 Would you operate the controller in burst-mode or cycle stealing mode?
 - 7.2 What priority scheme would you employ for service of the DMA channel?

8. A processor and I/O device D are connected to main memory M via a shared bus having width of one word. CPU can execute 10^6 instructions per sec. An average instruction requires five machine cycles, three of which use the memory bus. A memory read or write operation uses one machine cycle. Suppose that processor is continuously executing background program that requires 95% of its instruction execution rate but not any I/O instructions. Assume that one processor cycle equals one bus cycle. Now suppose that the I/O device is to be used to transfer very large blocks data between M and D .
 - 8.1 If programmed I/O is used and each one-word I/O transfer requires the processor to execute two instructions, estimate the maximum I/O data-transfer rate, in words per sec., possible through D .
 - 8.2 Estimate the same rate if DMA is used.

9. A typical CPU allows most interrupts to be enabled and disabled under software control. In contrast, so cpu provides facilitates to disable DMA request signals. Explain why it is so?

-  John P. Hayes, "Computer Architecture and Organization", 2nd Edition, McGraw-Hill, 1988.
-  William Stalling, "Computer Organization and Architecture", 8th Edition, Pearson, 2010.
-  M. Morris Mano, "Computer System Architecture-3rd Edition", Pearson, 2006 (chapter 11).
-  <http://krchowdhary.com/co/co.html>