Some important questions

|  |
| --- |
| What is asynchronous data transfer? Describe it with strobe-control method and hand-shaking method. |
| Explain DMA based data transfer technique for I/O devices. |
| With a proper diagram explain Daisy chaining priority interrupt in detail. |
| Explain RISC characteristics. |
| Explain DMA in detail. |
| Draw the block diagram of a computer with I/O processor and discuss the process of CPU-IOP communication in detail. |
| What do you mean by Control word? |

**11.1. Peripheral Devices**

A peripheral device is any device that sends data to or receives data from the CPU. (Keyboard, mouse, monitor, printer, disk, etc.)

Some I/O devices are what we call *Human Interface Devices (HIDs)*. Such devices must be able to input and output data in a form suitable for humans. ( Text, sound, motion, etc. )

ASCII (American Standard Code for Information Interchange) is the original 7-bit character encoding standard. It assigns a 7-bit binary code (not a number!) to each letter of the alphabet, digit, punctuation symbol, etc. It is used to input and output data in human-readable form.

0-31 Control chars. Originally designed to control text-only

printers. Adopted by ASCII terminals (vt100, xterm)

for analogous functions.

4 EOT (ctrl+d)

7 BEL (ctrl+g) \007 Beep printer/terminal

8 BS (ctrl+h) \b Move head/cursor left

May shift rest of line on terminal

9 TAB (ctrl+i) \t Move head/cursor to next tab stop

10 LF (ctrl+j) \n Move head/cursor down (scroll)

12 FF (ctrl+l) \f Scroll to start of next page

13 CR (ctrl+m) \r Move head/cursor to col 1

Terminals: CR+LF goes to start of new line. Unix adds CR to each LF by default, so we write "Hello, world!\n" instead of "Hello, world!\n\r".

32-126 Printable characters. Print/display char and move

head/cursor right. Wrap at eol of printer/terminal

supports it.

32 Space

33 !

...

48 '0'

49 '1'

...

65 'A'

66 'B'

...

97 'a'

98 'b'

127 DEL Delete char under cursor

Examples:

char ch = 'A';

ch: .byte 'A'

What is in the variable ch? (binary)

ISO (International Standards Organization, now the International Organization for Standardization, extended the ASCII set to include non-English characters, graphic symbols, etc. The basic Latin-based ISO sets are 8 bits.

All ISO character sets are backward-compatible with ASCII, i.e. the character codes for the first 128 characters are the same as ASCII.

Some common ISO character sets: ISO-latin1, ISO-latin2

Unicode Transformation Format (UTF) is a set of multibyte character encodings that extend beyond 8 bits. UTF-8 is also backward-compatible with ASCII, and is the dominant character set used on the WEB.

## 11.2. Input-Output Interface

It would not be practical for every I/O device to be wired to the computer in a different way, so we must have a scheme where the hardware connections are fixed, and yet the communication with the device is flexible, so that the widely varying needs of devices can all be met.

An I/O device, from the viewpoint of the CPU, is a set of registers. The CPU communicates with and controls the I/O device by reading and writing these registers. For example, SPIM, the MIPS simulator, uses two registers to communicate with the keyboard.

* The keyboard data register contains the ASCII code of the last key pressed.
* The keyboard control register indicates when a new key has been pressed. If bit 0 is one, a key has been pressed since the last character was read. The keyboard controller sets this bit when a key is pressed. It clears this bit when the keyboard data register is read.

The CPU can find out whether a new character is available by reading the keyboard control register and testing bit 0. If bit 0 is 1, it then reads the keyboard data register to get the new key.

For another simple example, see the 10-bit analog to digital converter in the [PIC 18f8520 spec sheet](http://www.cs.uwm.edu/classes/cs458/Lecture/39609b.pdf).

Accessing I/O devices at the hardware level is a lot like accessing memory. The registers in the I/O devices are connected to the CPU using buses. We need an address bus to specify which I/O device register is to be accessed. We need control lines to specify what kind of access is desired (read, write, reset, etc.) Finally, we need a data bus to transfer the data between the CPU and the device.

Each device has one or more control, status, and data registers at various I/O addresses. A hypothetical example:

Address Register

ff00 keyboard status

ff01 keyboard data

ff02 display status

ff03 display data

ff04 disk status

ff05 disk block address

ff06 disk block size

ff07 disk data address

...

I/O read and write operations can be more complex than memory read and write operations, but the basic idea is the same. I/O control generally involves more than just read and write control lines. In a sense, memory can be viewed as a very simple, fast I/O device.

Whereas memory is just a large pool of slow, inexpensive registers for storing data, each I/O device register has a unique purpose in controlling a specific I/O device. This does not affect how the CPU accesses them at the hardware level, but it does affect how they are used by software.

Simple device control, such as stating whether an I/O register is to be read or written, can be done over the control lines. More complex devices are often controlled by sending special data blocks called Peripheral Control Blocks (PCBs) over the data lines. This is the primary method for communicating with disk drives, for example.

Since I/O devices are of a very different nature than CPU circuits, there must be interface hardware to connect each device to the CPU.

The interface hardware shown as a black-box in figure 11-1 is often divided between I/O chips tied closely to the CPU and controller boards embedded in the I/O device. ( Show SATA card and DVD drive )

Methods for designing a CPU's I/O interface generally fall into one of the following categories:

* Completely separate memory and I/O buses
* Common buses, separate control lines
* Common buses and control lines

Regardless of the CPU's interface, the connections to the I/O device controllers are the same. In fact, the same devices are often used on multiple architectures with vastly different CPUs, e.g. in an x86 PC and a PowerPC Mac.

What is the advantage of using separate I/O and memory buses?

What is the advantage of using common buses for memory and I/O?

Peripherals are usually much slower than CPUs (keyboards, disks, printers), but are occasionally faster (high-speed networks, some video controllers). The high speed of some of today's I/O devices has driven the need for PCI-E serial network buses to replace older parallel buses like ISA, PCI, PCI-X, AGP.

### 11.2.1. Isolated vs. Memory-mapped

If memory and I/O share address and data bus, there are two ways to distinguish them.

* In isolated I/O, common address and data bus lines are used to address both memory and I/O devices. Separate read/write control lines enable either memory or an I/O device, both never both, to accept the address. This scheme allows an I/O device and memory to use same address, since only one of them will be activated during any given clock cycle. In other words, memory and I/O time-share the buses, and have separate memory spaces. A given address can refer to a memory cell at one time and to an I/O device at another.

Systems using isolated I/O and systems with separate memory and I/O buses have distinct I/O instructions that activate the I/O read or write control lines. Memory reference instructions activate the memory read or write control lines.

+-------+

| | Address

| |---/--------+----------+

| | | |

| | Data | |

| |<-----/---+-|------+ |

| CPU | | | | |

| | v v | |

| | read +-------+ | |

| |------>| | | |

| |------>| Mem | | |

| | write +-------+ | |

| | +--------+ |

| | | +----------+

| | | |

| | v v

| | read +-------+

| |------>| |

| |------>| I/O |

| | write +-------+

+-------+

* In memory-mapped I/O, there is only one address space, and it is divided between memory cells and I/O device registers.
* Address Purpose
* 0000 Memory
* ... ...
* fff0 Memory
* fff1 Keyboard control register
* fff2 Keyboard data register
* fff3 Disk control register
* ...

Any given address is wired to either to a memory cell or an I/O device register, and cannot refer to anything else.

In memory-mapped I/O systems, there are no distinct I/O instructions. The CPU does not distinguish between memory access and I/O operations. The only difference between them is the address! Hence, we can used load and store instructions to access the registers in I/O devices on a load-store architecture. We can use almost any instruction (move, add, sub, etc.) to perform I/O on a memory-to-memory or register-memory computer.

### 11.3. Asynchronous Data Transfer

We know that, the internal operations in individual unit of digital system are synchronized by means of clock pulse, means clock pulse is given to all registers within a unit, and all data transfer among internal registers occur simultaneously during occurrence of clock pulse.Now, suppose any two units of digital system are designed independently such as CPU and I/O interface.

And if the registers in the interface(I/O interface) share a common clock with CPU registers, then transfer between the two units is said to be synchronous.But in most cases, the internal timing in each unit is independent from each other in such a way that each uses its own private clock for its internal registers.In that case, the two units are said to be asynchronous to each other, and if data transfer occur between them this data transfer is said to be **Asynchronous Data Transfer**.

But, the Asynchronous Data Transfer between two independent units requires that control signals be transmitted between the communicating units so that the time can be indicated at which they send data.

This asynchronous way of data transfer can be achieved by two methods:

1. One way is by means of strobe pulse which is supplied by one of the units to other unit.When transfer has to occur.This method is known as “**Strobe Control**”.
2. Another method commonly used is to accompany each data item being transferred with a control signal that indicates the presence of data in the bus.The unit receiving the data item responds with another signal to acknowledge receipt of the data.This method of data transfer between two independent units is said to be “**Handshaking**”.

The strobe pulse and handshaking method of asynchronous data transfer are not restricted to I/O transfer.In fact, they are used extensively on numerous occasion requiring transfer of data between two independent units.So, here we consider the transmitting unit as source and receiving unit as destination.

As an example: The CPU, is the source during an output or write transfer and is the destination unit during input or read transfer.

And thus, the sequence of control during an asynchronous transfer depends on whether the transfer is initiated by the source or by the destination.

So, while discussing each way of data transfer asynchronously we see the sequence of control in both terms when it is initiated by source or when it is initiated by destination.In this way, each way of data transfer, can be further divided into parts, source initiated and destination initiated.

We can also specify, asynchronous transfer between two independent units by means of a timing diagram that shows the timing relationship that exists between the control and the data buses.

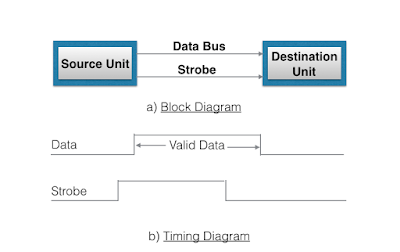
Now, we will discuss each method of asynchronous data transfer in detail one by one.

### 1. Strobe Control:

     The Strobe Control method of asynchronous data transfer employs a single control line to time each        transfer .This control line is also known as strobe and it may be achieved either by source or                    destination, depending on which initiate transfer.

#### Source initiated strobe for data transfer:

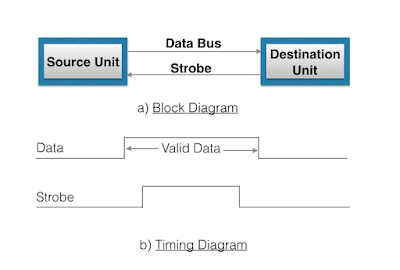
     The block diagram and timing diagram of strobe initiated by source unit is shown in figure below:

[](https://1.bp.blogspot.com/-HsusyfVCiBw/V8-PN4sD4KI/AAAAAAAAAOw/J1Hq9blUGS0HBLJ0JPuM5GYo6Yfe-i9jwCLcB/s1600/Screen%2BShot%2B2016-09-07%2Bat%2B09.22.32.png)

     In block diagram we see that strobe is initiated by source, and as shown in timing diagram, the                source unit first places the data on the data bus.After a brief delay to ensure that the data settle to a        steady value, the source activates a strobe pulse.The information on data bus and strobe control            signal remain in the active state for a sufficient period of time to allow the destination unit to receive        the data.Actually, the destination unit, uses a falling edge of strobe control to transfer the contents of        data bus to one of its internal registers.The source removes the data from the data bus after it                  disables its strobe pulse.New valid data will be available only after the strobe is enabled again.

#### Destination-initiated strobe for data transfer:

     The block diagram and timing diagram of strobe initiated by destination is shown in figure below:

[](https://3.bp.blogspot.com/-ch3GPPYxY6g/V8-PnxBgGlI/AAAAAAAAAO0/vrwrYk8AeIwm4rHFzuq_vSt6KmsGjivQQCLcB/s1600/Screen%2BShot%2B2016-09-07%2Bat%2B09.24.17.png)

     In block diagram, we see that, the strobe initiated by destination, and as shown in timing diagram, the      destination unit first activates the strobe pulse, informing the source to provide the data.The source          unit responds by placing the requested binary information on the data bus.The data must be valid            and remain in the bus long enough for the destination unit to accept it.The falling edge of strobe              pulse can be used again to trigger a destination register.The destination unit then disables the                  strobe.And source removes the data from data bus after a per determine time interval.

     Now, actually in computer, in the first case means in strobe initiated by source - the strobe may be a        memory-write control signal from the CPU to a memory unit.The source, CPU, places the word on          the data bus and informs the memory unit, which is the destination, that this is a write operation.

     And in the second case i.e, in the strobe initiated by destination - the strobe may be a memory read        control from the CPU to a memory unit.The destination, the CPU, initiates the read operation to                inform the memory, which is a source unit, to place selected word into the data bus.

### 2. Handshaking:

     The disadvantage of strobe method is that source unit that initiates the transfer has no way of                  knowing whether the destination has actually received the data that was placed in the bus.Similarly, a      destination unit that initiates the transfer has no way of knowing whether the source unit, has actually      placed data on the bus.

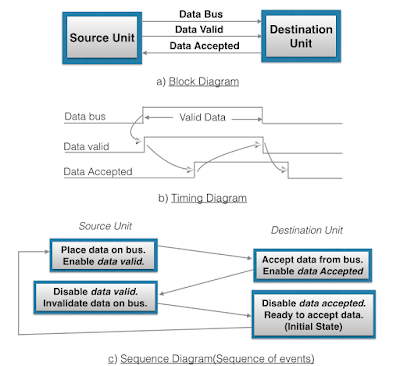
     This problem can be solved by handshaking method.

     Hand shaking method introduce a second control signal line that provides a replay to the unit that            initiates the transfer.

     In it, one control line is in the same direction as the data flow in the bus from the source to                        destination.It is used by source unit to inform the destination unit whether there are valid data in the        bus.The other control line is in the other direction from destination to the source.It is used by the              destination unit to inform the source whether it can accept data.And in it also, sequence of control            depends on unit that initiate transfer.Means sequence of control depends whether transfer is initiated      by source and destination.Sequence of control in both of them are described below:

#### Source initiated Handshaking:

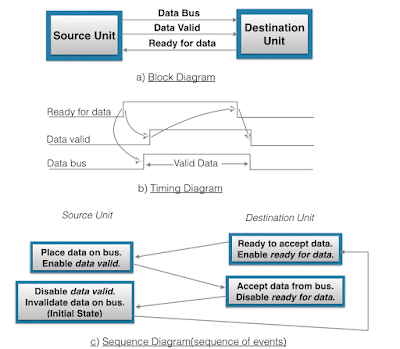
     The source initiated transfer using handshaking lines is shown in figure  below:

[](https://1.bp.blogspot.com/-mK55LGTjugA/V9FY7CoXDtI/AAAAAAAAAPs/uMNIQaCBrVg7e8R2HpL1Gc72vFzyErx3ACLcB/s1600/Screen%2BShot%2B2016-09-08%2Bat%2B14.59.30.png)

     In its block diagram, we se that two handshaking lines are "data valid", which is generated by the            source unit, and "data accepted", generated by the destination unit.  
  
     The timing diagram shows the timing relationship of exchange of signals between the two                        units.Means as shown in its timing diagram, the source initiates a transfer by placing data on the bus      and enabling its data valid signal.The data accepted signal is then activated by destination unit after it      accepts the data from the bus.The source unit then disable its data valid signal which invalidates the        data on the bus.After this, the destination unit disables its data accepted signal and the system goes        into initial state.The source unit does not send the next data item until after the destination unit shows      its readiness to accept new data by disabling the data accepted signal.  
  
     This sequence of events described in its sequence diagram, which shows the above sequence in            which the system is present, at any given time.

#### Destination initiated handshaking:

     The destination initiated transfer using handshaking lines is shown in figure  below:

[](https://1.bp.blogspot.com/-G10r82h91tU/V9FaG0KQcKI/AAAAAAAAAPw/Bn25tXqutUUfDD9ZwEAnO5R1a7RgC8OhQCLcB/s1600/Screen%2BShot%2B2016-09-08%2Bat%2B15.02.22.png)

     In its block diagram, we see that the two handshaking lines are "data valid", generated by the source      unit, and "ready for data" generated by destination unit.Note that the name of signal data accepted          generated by destination unit has been changed to ready for data to reflect its new meaning.

     In it, transfer is initiated by destination, so source unit does not place data on data bus until it                    receives ready for data signal from destination unit.After that, hand shaking process is some as that        of source initiated.

     The sequence of event in it are shown in its sequence diagram and timing relationship between              signals is shown in its timing diagram.

     Thus, here we can say that, sequence of events in both cases would be identical.If we consider              ready for data signal as the complement of data accept.Means, the only difference between source          and destination initiated transfer is in their choice of initial state.

## Mode of Transfer

The method that is used to transfer information between internal storage and external I/O devices is known as I/O interface. The CPU is interfaced using special communication links by the peripherals connected to any computer system. These communication links are used to resolve the differences between CPU and peripheral. There exists special hardware components between CPU and peripherals to supervise and synchronize all the input and output transfers that are called interface units.

Mode of Transfer:

The binary information that is received from an external device is usually stored in the memory unit. The information that is transferred from the CPU to the external device is originated from the memory unit. CPU merely processes the information but the source and target is always the memory unit. Data transfer between CPU and the I/O devices may be done in different modes.

Data transfer to and from the peripherals may be done in any of the three possible ways

1. Programmed I/O.
2. Interrupt- initiated I/O.
3. Direct memory access( DMA).

Now let’s discuss each mode one by one.

1. **Programmed I/O:** It is due to the result of the I/O instructions that are written in the computer program. Each data item transfer is initiated by an instruction in the program. Usually the transfer is from a CPU register and memory. In this case it requires constant monitoring by the CPU of the peripheral devices.

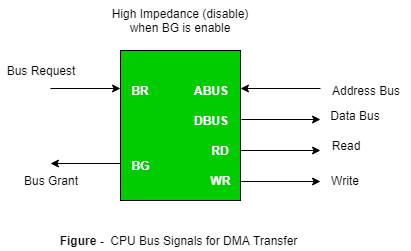
**Example of Programmed I/O:** In this case, the I/O device does not have direct access to the memory unit. A transfer from I/O device to memory requires the execution of several instructions by the CPU, including an input instruction to transfer the data from device to the CPU and store instruction to transfer the data from CPU to memory. In programmed I/O, the CPU stays in the program loop until the I/O unit indicates that it is ready for data transfer. This is a time consuming process since it needlessly keeps the CPU busy. This situation can be avoided by using an interrupt facility. This is discussed below.

1. **Interrupt- initiated I/O:** Since in the above case we saw the CPU is kept busy unnecessarily. This situation can very well be avoided by using an interrupt driven method for data transfer. By using interrupt facility and special commands to inform the interface to issue an interrupt request signal whenever data is available from any device. In the meantime the CPU can proceed for any other program execution. The interface meanwhile keeps monitoring the device. Whenever it is determined that the device is ready for data transfer it initiates an interrupt request signal to the computer. Upon detection of an external interrupt signal the CPU stops momentarily the task that it was already performing, branches to the service program to process the I/O transfer, and then return to the task it was originally performing.

**Note:** Both the methods programmed I/O and Interrupt-driven I/O require the active intervention of the  
processor to transfer data between memory and the I/O module, and any data transfer must transverse  
a path through the processor. Thus both these forms of I/O suffer from two inherent drawbacks.

* + The I/O transfer rate is limited by the speed with which the processor can test and service a  
    device.
  + The processor is tied up in managing an I/O transfer; a number of instructions must be executed  
    for each I/O transfer.

1. **Direct Memory Access**: The data transfer between a fast storage media such as magnetic disk and memory unit is limited by the speed of the CPU. Thus we can allow the peripherals directly communicate with each other using the memory buses, removing the intervention of the CPU. This type of data transfer technique is known as DMA or direct memory access. During DMA the CPU is idle and it has no control over the memory buses. The DMA controller takes over the buses to manage the transfer directly between the I/O devices and the memory unit.



**Bus Request :** It is used by the DMA controller to request the CPU to relinquish the control of the buses.

**Bus Grant :** It is activated by the CPU to Inform the external DMA controller that the buses are in high impedance state and the requesting DMA can take control of the buses. Once the DMA has taken the control of the buses it transfers the data. This transfer can take place in many ways.

**Types of DMA transfer using DMA controller:**

**Burst Transfer :**  
DMA returns the bus after complete data transfer. A register is used as a byte count,  
being decremented for each byte transfer, and upon the byte count reaching zero, the DMAC will  
release the bus. When the DMAC operates in burst mode, the CPU is halted for the duration of the data  
transfer.  
Steps involved are:

* + Bus grant request time.
  + Transfer the entire block of data at transfer rate of device because the device is usually slow than the  
    speed at which the data can be transferred to CPU.
  + Release the control of the bus back to CPU  
    So, total time taken to transfer the N bytes  
    = Bus grant request time + (N) \* (memory transfer rate) + Bus release control time.

Where,

X µsec =data transfer time or preparation time (words/block)

Y µsec =memory cycle time or cycle time or transfer time (words/block)

% CPU idle (Blocked)=(Y/X+Y)\*100

% CPU Busy=(X/X+Y)\*100

**Cyclic Stealing :**  
In this DMA controller transfers one word at a time after which it must return the control of the buses to the CPU. The CPU merely delays its operation for one memory cycle to allow the direct memory I/O transfer to “steal” one memory cycle.  
Steps Involved are:

* + Buffer the byte into the buffer
  + Inform the CPU that the device has 1 byte to transfer (i.e. bus grant request)
  + Transfer the byte (at system bus speed)
  + Release the control of the bus back to CPU.

Before moving on transfer next byte of data, device performs step 1 again so that bus isn’t tied up and  
the transfer won’t depend upon the transfer rate of device.  
So, for 1 byte of transfer of data, time taken by using cycle stealing mode (T).  
= time required for bus grant + 1 bus cycle to transfer data + time required to release the bus, it will be  
N x T

In cycle stealing mode we always follow pipelining concept that when one byte is getting transferred then Device is parallel preparing the next byte. “The fraction of CPU time to the data transfer time” if asked then cycle stealing mode is used.

Where,

X µsec =data transfer time or preparation time

(words/block)

Y µsec =memory cycle time or cycle time or transfer

time (words/block)

% CPU idle (Blocked) =(Y/X)\*100

% CPU busy=(X/Y)\*100

**Interleaved mode:** In this technique , the DMA controller takes over the system bus when the  
microprocessor is not using it.An alternate half cycle i.e. half cycle DMA + half cycle processor.  
  
  
**Note:** In Gate Exam you can directly apply above formula for different mode of DMA transfer.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.