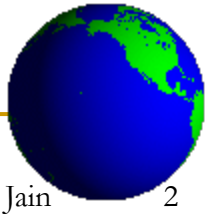
Input-Output Organization

Pooja Jain
Senior Lecturer, JUIT
Waknaghat

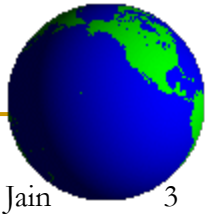
INPUT-OUTPUT ORGANIZATION

- **Peripheral Devices**
- **Input-Output Interface**
- **Asynchronous Data Transfer**
- **Modes of Transfer**
- **Priority Interrupt**
- **Direct Memory Access**
- **Input-Output Processor**
- **Serial Communication**



Peripheral

- **What are connected online devices ?**
  - Devices that are under the direct control of computer.
  - Designed to Read/write information to memory on CPU command
- **What are peripherals?**
  - Examples



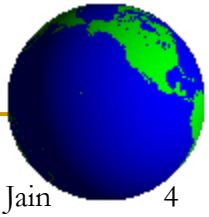
PERIPHERAL DEVICES

Input Devices

- Keyboard
- Optical input devices
  - Card Reader
  - Paper Tape Reader
  - Bar code reader
  - Digitizer
  - Optical Mark Reader
- Magnetic Input Devices
  - Magnetic Stripe Reader
- Screen Input Devices
  - Touch Screen
  - Light Pen
  - Mouse
- Analog Input Devices

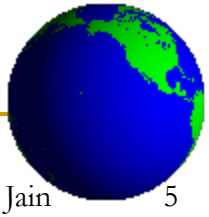
Output Devices

- Card Puncher, Paper Tape Puncher
- CRT
- Printer (Impact, Ink Jet, Laser, Dot Matrix)
- Plotter
- Analog
- Voice



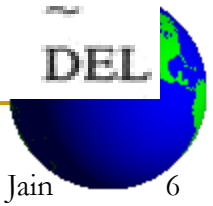
ASCII alphanumeric characters

- The American Standard Code for Information Interchange (ASCII) is the standard binary code for the alphanumeric characters.
- It consists of 94 printable characters and 34 non-printable characters which are used for various control functions.
- The control characters are used for routing data and arranging the printed text into a prescribed format. There are 3 types
  - Format effectors
  - Information separators
  - Communication control characters



ASCII Characters

$b_4 b_3 b_2 b_1$ \ $b_7 b_6 b_5$	000	001	010	011	100	101	110	111
0000	NUL	DLE	SP	0	@	P	`	p
0001	SOH	DC1	!	1	A	Q	a	q
0010	STX	DC2	"	2	B	R	b	r
0011	ETX	DC3	#	3	C	S	c	s
0100	EOT	DC4	$	4	D	T	d	t
0101	ENQ	NAK	%	5	E	U	e	u
0110	ACK	SYN	&	6	F	V	f	v
0111	BEL	ETB	'	7	G	W	g	w
1000	BS	CAN	(	8	H	X	h	x
1001	HT	EM	)	9	I	Y	i	y
1010	LF	SUB	*	:	J	Z	j	z
1011	VT	ESC	+	;	K	[	k	{
1100	FF	FS	,	<	L	\	l	|
1101	CR	GS	-	=	M	]	m	}
1110	SO	RS	.	>	N	^	n	~
1111	SI	US	/	?	O	_	o	DEL

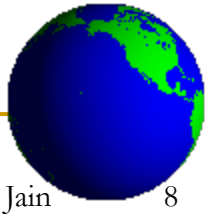


ASCII Control Characters

NUL	Null	DLE	Data link escape
SOH	Start of heading	DC1	Device control 1
STX	Start of text	DC2	Device control 2
ETX	End of text	DC3	Device control 3
EOT	End of transmission	DC4	Device control 4
ENQ	Enquiry	NAK	Negative acknowledge
ACK	Acknowledge	SYN	Synchronous idle
BEL	Bell	ETB	End of transmission block
BS	Backspace	CAN	Cancel
HT	Horizontal tab	EM	End of medium
LF	Line feed	SUB	Substitute
VT	Vertical tab	ESC	Escape
FF	Form feed	FS	File separator
CR	Carriage return	GS	Group separator
SO	Shift out	RS	Record separator
SI	Shift in	US	Unit separator
SP	Space	DEL	Delete

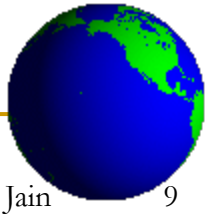
INPUT/OUTPUT INTERFACE

- Provides a method for transferring information between internal storage (such as memory and CPU registers) and external I/O devices
- Resolves the *differences* between the computer and peripheral devices
  - Peripherals - Electromechanical and electromagnetic Devices
  - CPU or Memory - Electronic Device
  - Data Transfer Rate
    - Peripherals - Usually slower
    - CPU or Memory - Usually faster than peripherals
      - Some kinds of Synchronization mechanism may be needed
  - Unit of Information
    - Peripherals – Byte, Block, …
    - CPU or Memory – Word
  - The operating modes of peripherals are different from each other and each must be controlled so as to not disturb the operation of other peripherals connected to the CPU

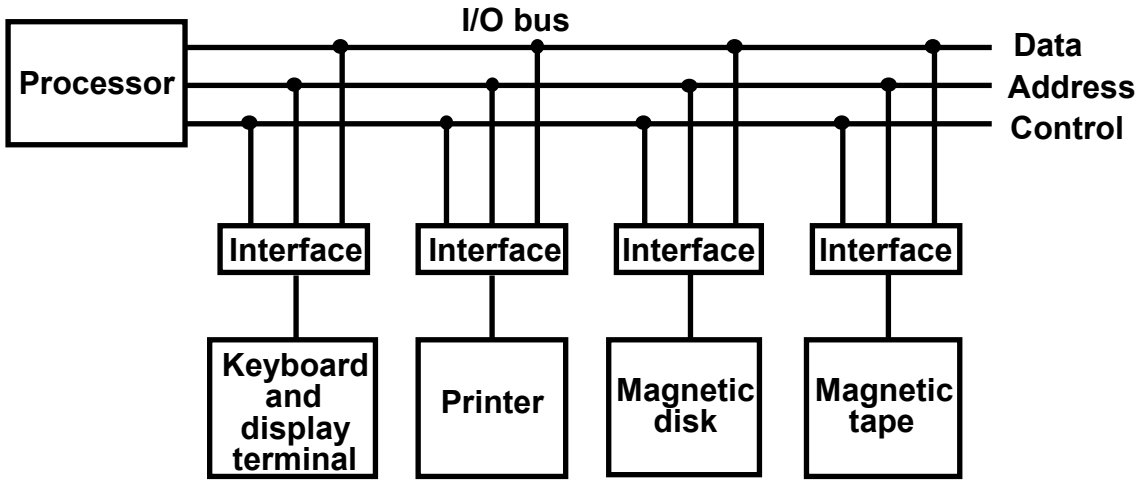


INTERFACE

- To resolve these problems system needs a unit to act as interface between the CPU and peripheral devices. Such components are called as **INTERFACE.**
  - **To supervise and synchronize all inputs and output transfers**
- In addition to Interface each device has its own Controller that **controls the specific operations** in peripherals.It can be in built or external
- Interface decodes the address on the address lines and control line and then provide appropriate signals to controller



I/O BUS AND INTERFACE MODULES

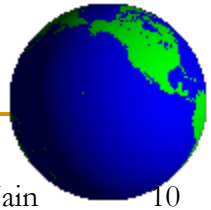


Each peripheral has an interface module associated with it

Interface

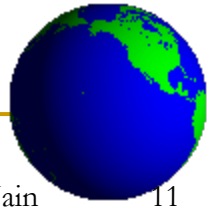
- **Decodes the device address (device code)**
- **Decodes the commands (operation)**
- **Provides signals for the peripheral controller**
- **Synchronizes the data flow and supervises the transfer rate between peripheral and CPU or Memory**

Typical I/O instruction



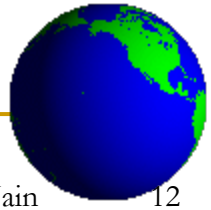
Commands

- There are four types of commands that an interface may receive. They are classified as:-
  - Control
  - Status
  - Data output
  - Data input



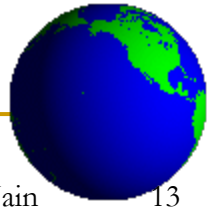
I/O Bus

- **Data Lines**
  - Data read or written
- **Address Lines**
  - Each interface has a unique address
- **Control Lines**
  - Passes commands to interface



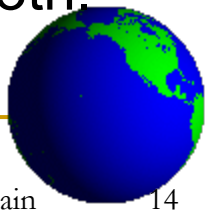
I/O Bus and Interface Modules

- Each interface has own address
- Processor places address on address lines
- Corresponding peripheral responds while others are deactivated
- Same time processor provides function code on control lines
- Interface executes function on peripheral



I/O vs. Memory Bus

- There are 3 ways that computer buses can be used to communicate with I/O and memory
  - Use two separate buses, one for memory and the other for I/O
    - Separate data, address, and control bus for memory and I/O
    - Used in computers which have separate IOP and CPU
  - Use common bus for both memory and I/O but have separate control lines for each.
  - Common bus and control lines for memory and I/O both.



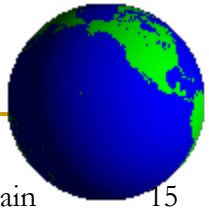
ISOLATED vs MEMORY MAPPED I/O

Isolated I/O

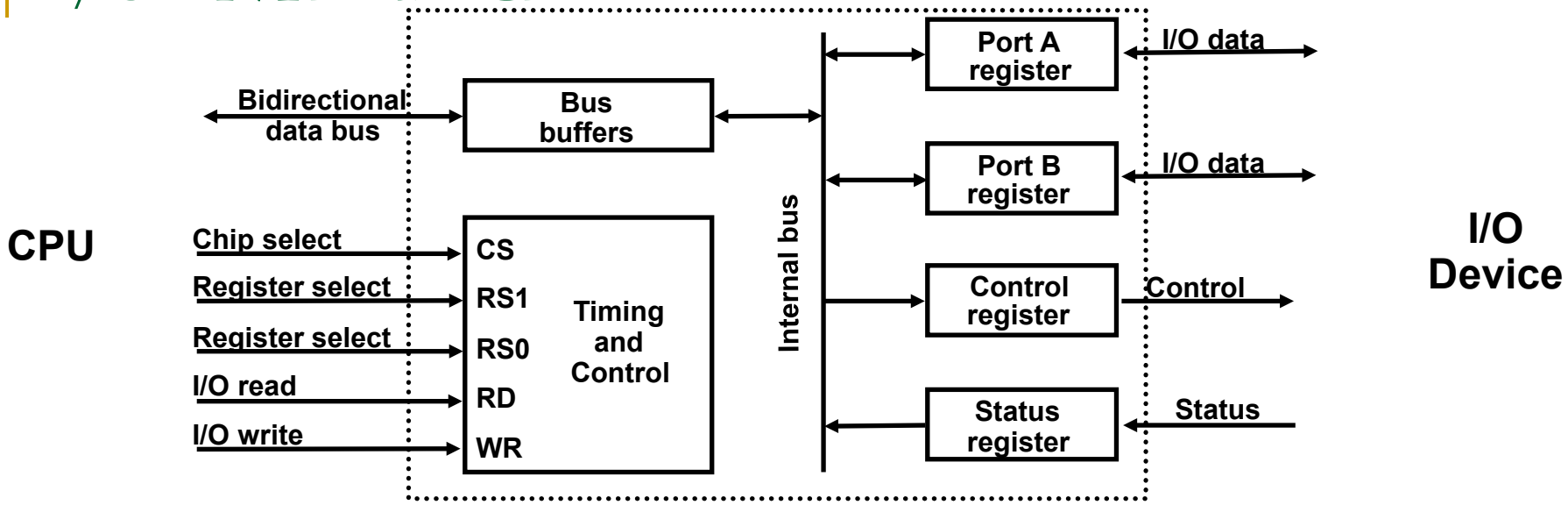
- Separate I/O read/write control lines in addition to memory read/write control lines
- Separate (isolated) memory and I/O address spaces
- Distinct input and output instructions

Memory-mapped I/O

- A single set of read/write control lines (no distinction between memory and I/O transfer)
- Memory and I/O addresses share the common address space
  -> reduces memory address range available
- No specific input or output instruction
  -> The same memory reference instructions can be used for I/O transfers
- Considerable flexibility in handling I/O operations



I/O INTERFACE



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

Programmable Interface

- Information in each port can be assigned a meaning depending on the mode of operation of the I/O device
  → Port A = Data; Port B = Command; Port C = Status

- CPU initializes(loads) each port by transferring a byte to the Control Register
  → Allows CPU can define the mode of operation of each port
  → *Programmable Port*: By changing the bits in the control register, it is possible to change the interface characteristics

ASYNCHRONOUS DATA TRANSFER

Synchronous and Asynchronous Operations

Synchronous - All devices derive the timing information from common clock line

Asynchronous - No common clock

Asynchronous Data Transfer

Asynchronous data transfer between two independent units requires that *control signals* be transmitted between the communicating units *to indicate the time at which data is being transmitted*

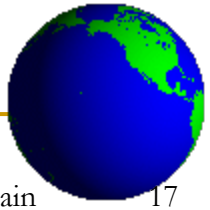
Two Asynchronous Data Transfer Methods

Strobe pulse

- **A strobe pulse is supplied by one unit to indicate the other unit when the transfer has to occur**

Handshaking

- **A control signal is accompanied with each data being transmitted to indicate the presence of data**
- **The receiving unit responds with another control signal to acknowledge receipt of the data**

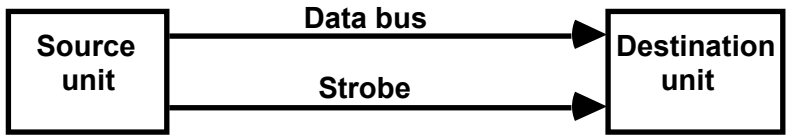


STROBE CONTROL

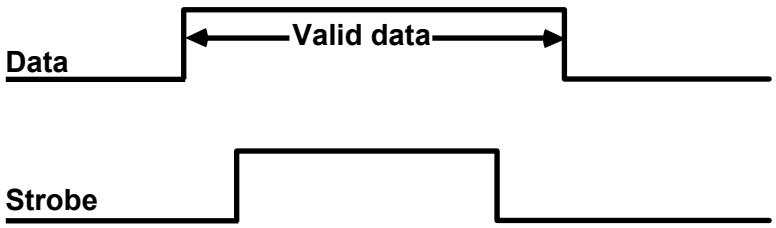
* Employs a single control line to time each transfer
* The strobe may be activated by either the source or the destination unit

Source-Initiated Strobe for Data Transfer

Block Diagram

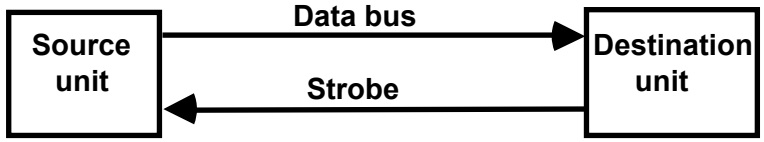


Timing Diagram

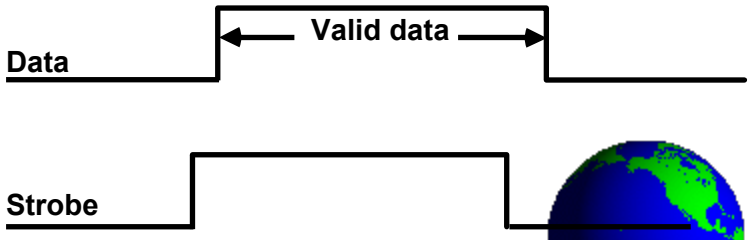


Destination-Initiated Strobe for Data Transfer

Block Diagram



Timing Diagram



HANDSHAKING

Strobe Methods

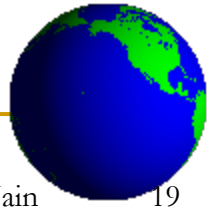
Source-Initiated

The source unit that initiates the transfer has no way of knowing whether the destination unit has actually received data

Destination-Initiated

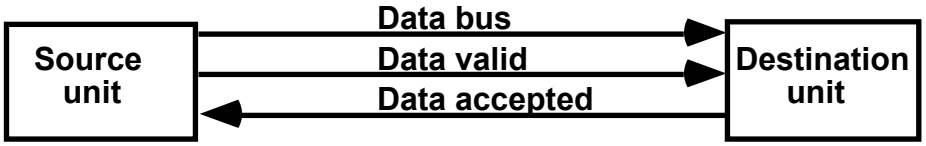
The destination unit that initiates the transfer no way of knowing whether the source has actually placed the data on the bus

To solve this problem, the *HANDSHAKE* method introduces a second control signal to provide a *Reply* to the unit that initiates the transfer

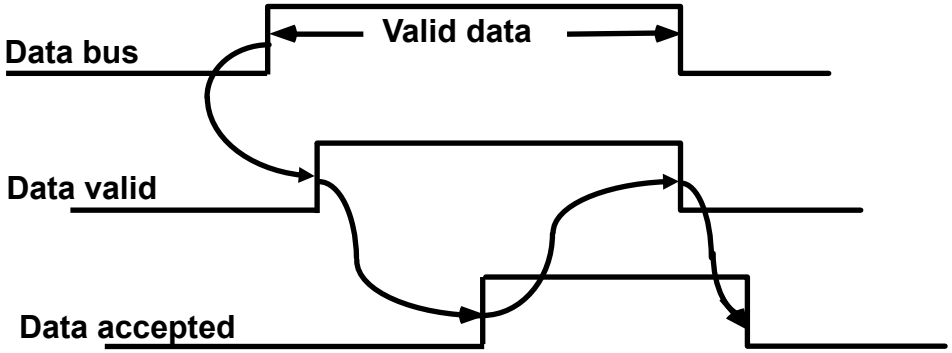


SOURCE-INITIATED TRANSFER USING HANDSHAKE

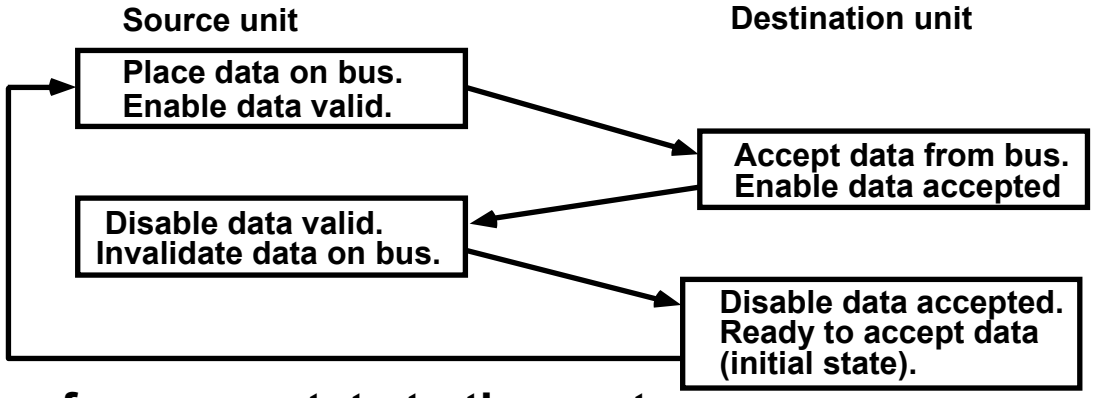
Block Diagram



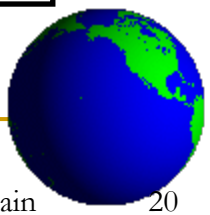
Timing Diagram



Sequence of Events

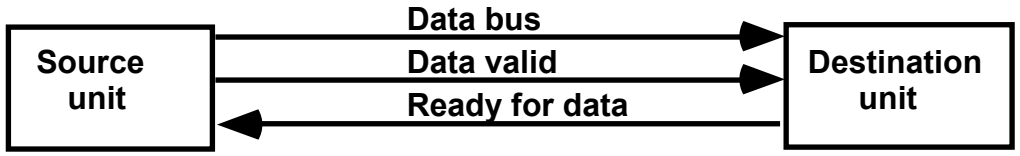


* Allows arbitrary delays from one state to the next
* Permits each unit to respond at its own data transfer rate
* The rate of transfer is determined by the slower unit

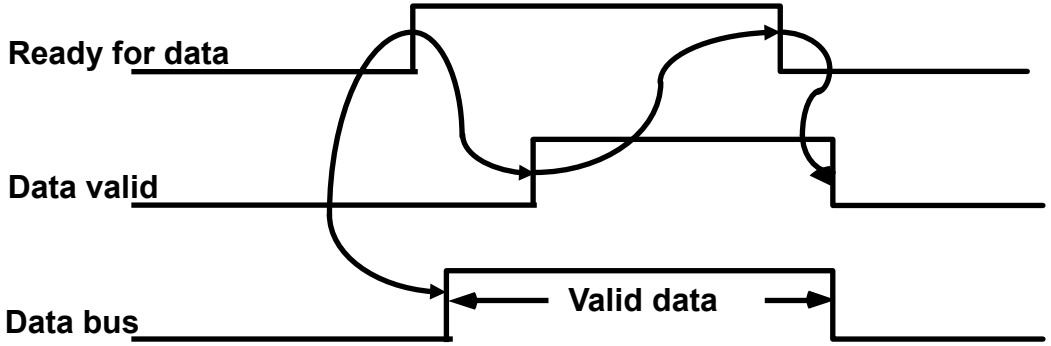


DESTINATION-INITIATED TRANSFER USING HANDSHAKE

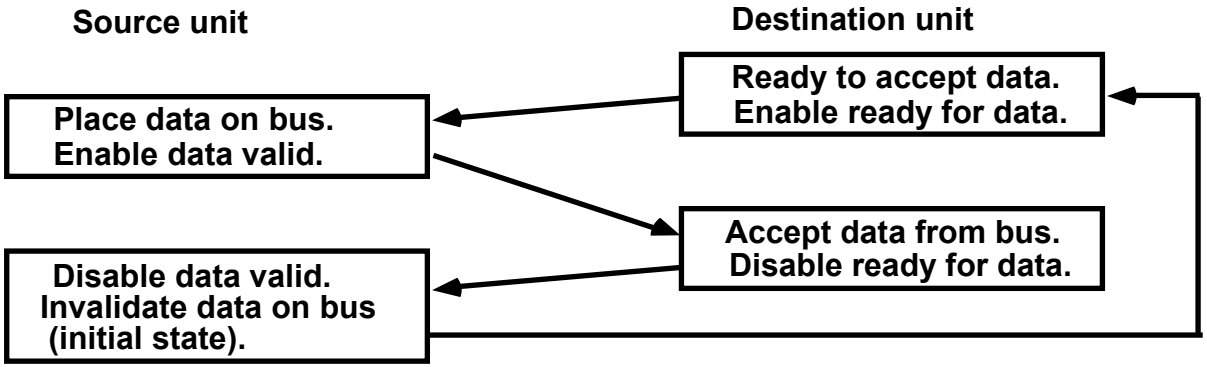
Block Diagram



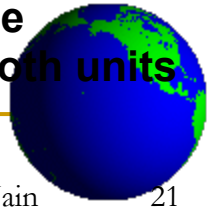
Timing Diagram



Sequence of Events



* Handshaking provides a high degree of flexibility and reliability because the successful completion of a data transfer relies on active participation by both units
* If one unit is faulty, data transfer will not be completed
  -> Can be detected by means of a *timeout* mechanism



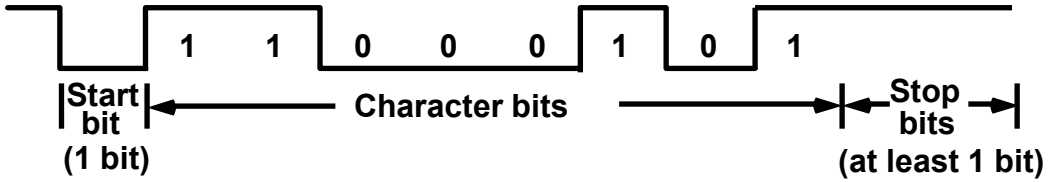
ASYNCHRONOUS SERIAL TRANSFER

Four Different Types of Transfer

Asynchronous serial transfer
Synchronous serial transfer
Asynchronous parallel transfer
Synchronous parallel transfer

Asynchronous Serial Transfer

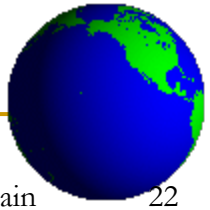
- **Employs special bits which are inserted at both ends of the character code**
- **Each character consists of three parts; Start bit; Data bits; Stop bits.**



A character can be detected by the receiver from the knowledge of 4 rules;

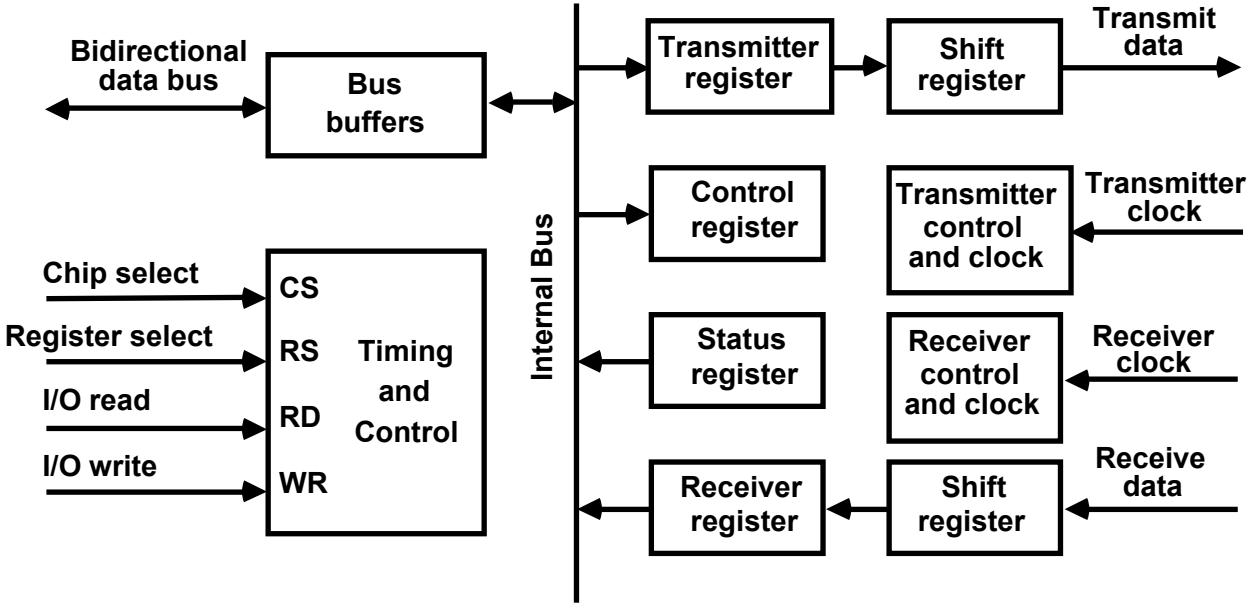
- **When data are not being sent, the line is kept in the 1-state (idle state)**
- **The initiation of a character transmission is detected by a *Start Bit* , which is always a 0**
- **The character bits always follow the *Start Bit***
- **After the last character , a *Stop Bit* is detected when the line returns to the 1-state for at least 1 bit time**

The receiver knows in advance the transfer rate of the bits and the number of information bits to expect

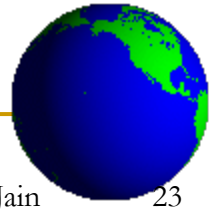


UNIVERSAL ASYNCHRONOUS RECEIVER-TRANSMITTER - UART -

A typical asynchronous communication interface available as an IC

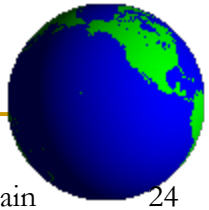


CS	RS	Oper.	Register selected
0	x	x	None
1	0	WR	Transmitter register
1	1	WR	Control register
1	0	RD	Receiver register
1	1	RD	Status register



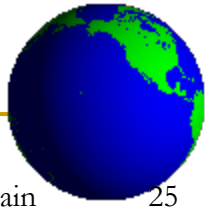
UART cont…

- Two bits in the status register are used as flags
  - Whether transmitter register is empty
  - Whether the receiver register is full
- Status Register Bits
  - Used for I/O flags and for recording errors
- Control Register Bits
  - Define baud rate, no. of bits in each character, whether to generate and check parity, and no. of stop bits



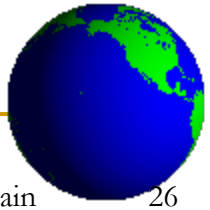
Transmitter register

- Accepts a data byte (from CPU) through the data bus when its empty (designated by the flag bit)
- Transferred to a shift register for serial transmission
- The first bit in the transmitter shift register is set to 0 to generate a start bit
- The character is transferred in parallel from transmitter register to the shift register and the appropriate number of stop bits are appended into the shift register
- The transmitter register is then marked empty



Receiver register

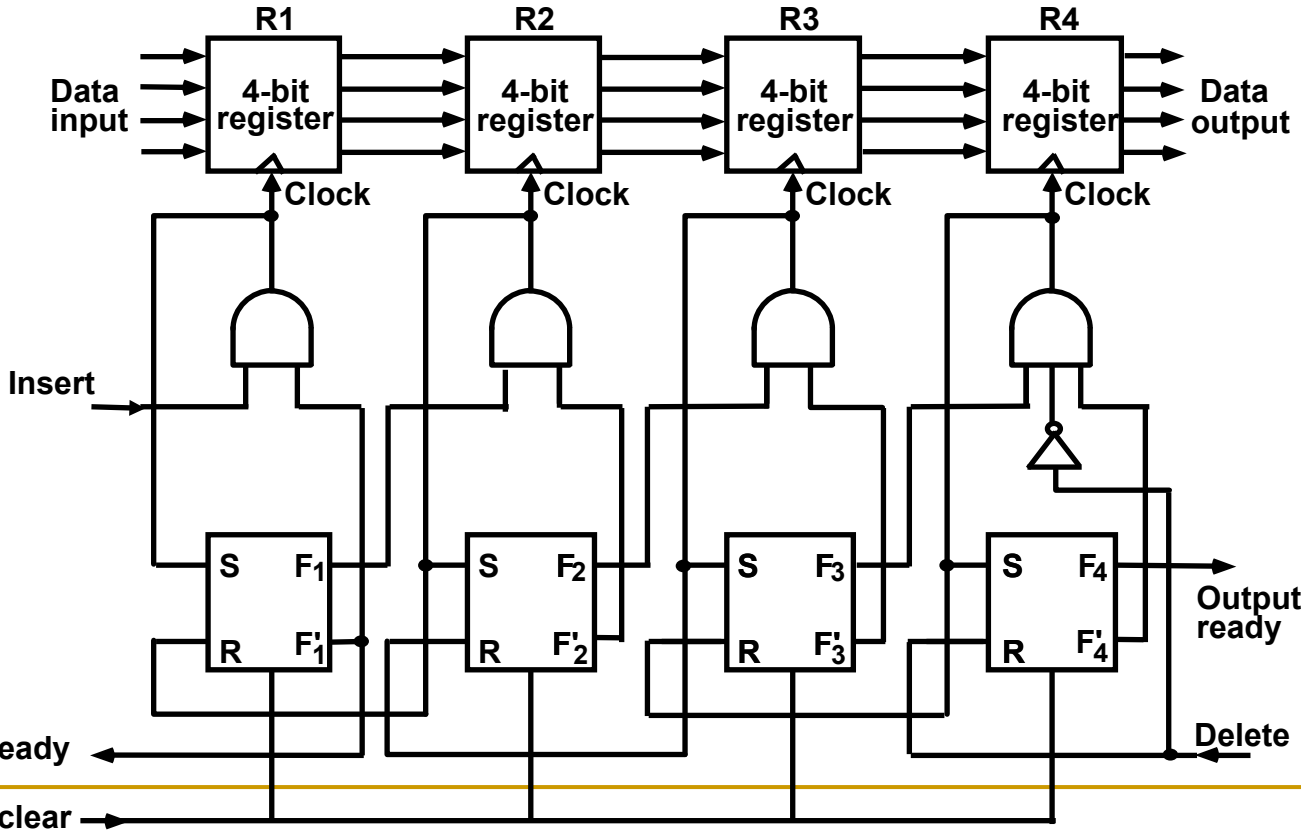
- Receives serial information into another shift register
- Complete data byte is sent to the receiver register
- The receiver controller monitors the receive-data line for a 0 signal to detect the occurrence of a start bit
- Once a start bit is detected, the incoming bits of the character are shifted into the shift register at the prescribed baud rate.
- After receiving the data bits, the interface checks for the parity and stop bits.
- The character without the start and stop bits is then transferred in parallel from the shift register to the receiver register.
- The flag in the status register is set to 1 to indicate that the receiver register is full.



FIRST-IN-FIRST-OUT(FIFO) BUFFER

* Input data and output data at two different rates
* Output data are always in the same order in which the data entered the buffer.
* Useful in some applications when data is transferred asynchronously

4 x 4 FIFO Buffer (4 4-bit registers Ri),
4 Control Registers(flip-flops Fi, associated with each Ri)



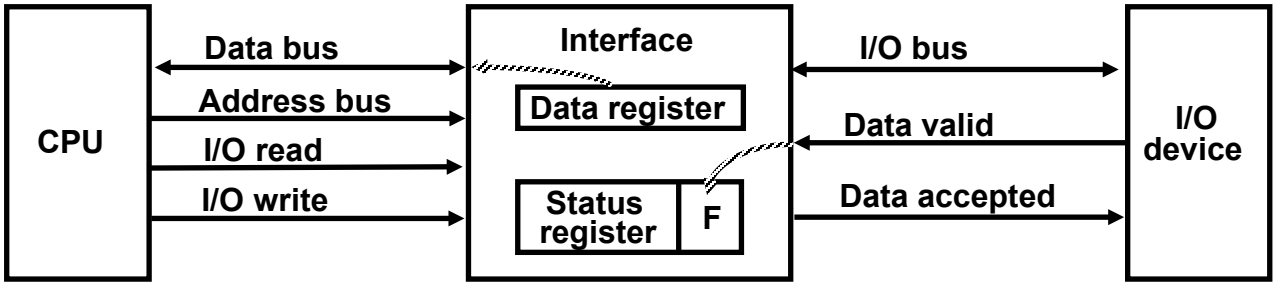
MODES OF TRANSFER - PROGRAM-CONTROLLED I/O

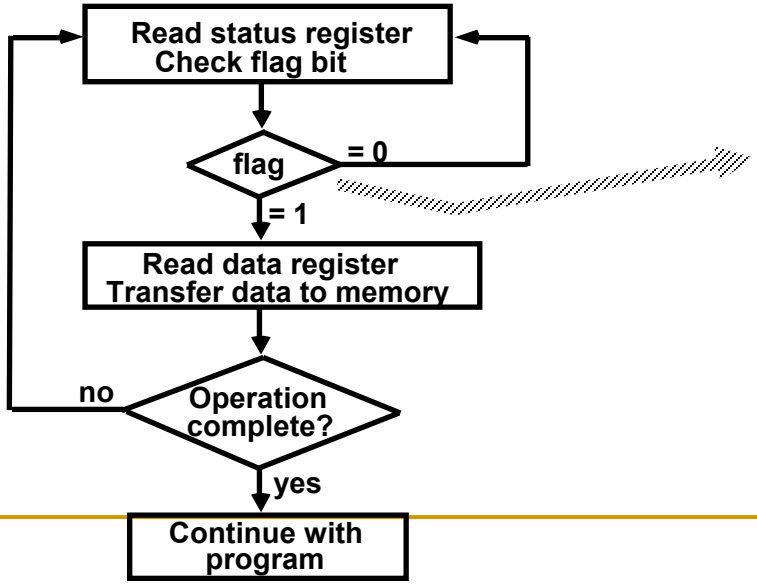
-

3 different Data Transfer Modes between the central computer(CPU or Memory) and peripherals;

Program-Controlled I/O
Interrupt-Initiated I/O
Direct Memory Access (DMA)

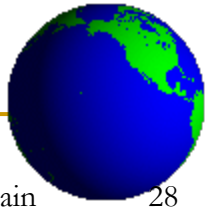
Program-Controlled I/O(Input Dev to CPU)





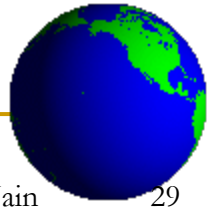
Polling or Status Checking

- **Continuous CPU involvement**
- **CPU slowed down to I/O speed**
- **Simple**
- **Least hardware**



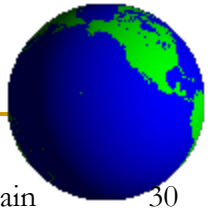
Programmed control

- The transfer of each byte requires three instructions:-
  - Read the status register
  - Check the status of the flag bit and branch to step 1 if not set or to step 3 if set
  - Read the data register



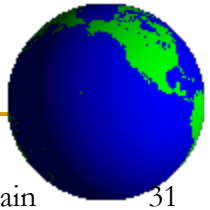
Interrupt Initiated I/O

- **Polling takes valuable CPU time**
- **Open communication only when some data has to be passed -> *Interrupt*.**
- **- I/O interface, instead of the CPU, monitors the I/O device**
- **- When the interface determines that the I/O device is ready for data transfer, it generates an *Interrupt Request* to the CPU**
- **Upon detecting an interrupt, CPU stops momentarily the task it is doing, branches to the service routine to process the data transfer, and then returns to the task it was performing**



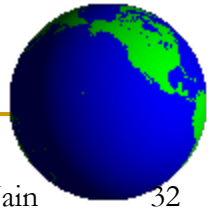
Vectored interrupt

- There are two ways that the processor chooses the branch address of the service routine:-
  - In a Non vectored interrupt, the branch address is assigned to a fixed location in memory
  - In Vectored interrupt, the source that interrupts supplies the branch information to the computer



PRIORITY INTERRUPT

- Data transfer between the CPU and an I/O device is initiated by the CPU. However, the CPU cannot start the transfer unless the device is ready to communicate with the CPU.
- The readiness of the device can be determined from an interrupt signal. The CPU responds to the interrupt request by storing the return address in a stack



PRIORITY INTERRUPT

Priority

- Determines which interrupt is to be served first when two or more requests are made simultaneously

- Also determines which interrupts are permitted to interrupt the computer while another is being serviced

- Higher priority interrupts can make requests while servicing a lower priority interrupt

Priority Interrupt by Software(Polling)

- Priority is established by the order of polling the devices(interrupt sources)

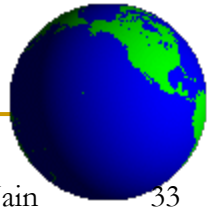
- One common branch address for all the interrupts

- The highest priority source are tested first, and if its interrupt signal is on, control branches to a service routine for this source

- Flexible since it is established by software

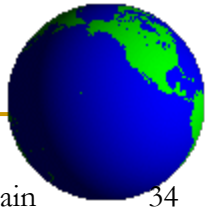
- Low cost since it needs a very little hardware

- Very slow

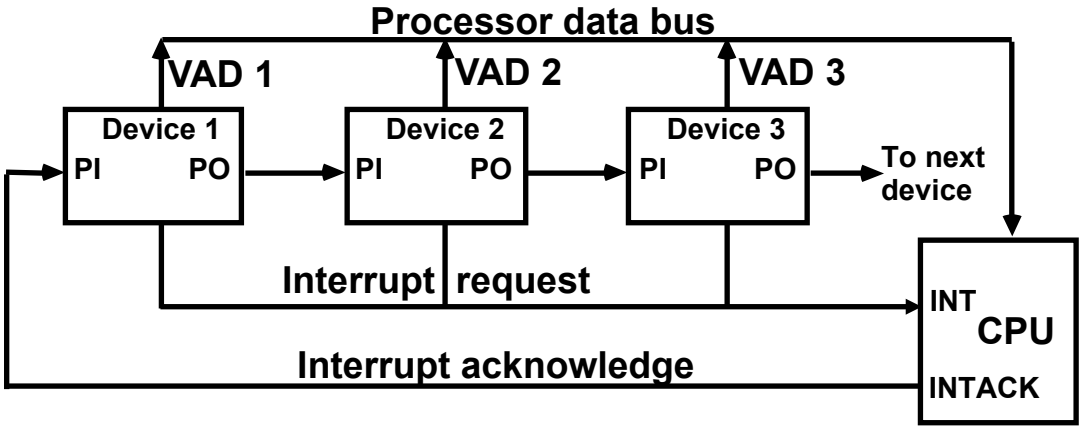


Hardware priority- interrupt unit

- It functions as an overall interrupt manager which accepts all the interrupt requests to determine the highest priority request
- Fast since identification of the highest priority interrupt request is identified by the hardware and no polling is required
- Fast since each interrupt source has its own interrupt vector to access directly to its own service routine
- The hardware priority function can ve established by either a serail (daisy chain) or a parallel connection (parallel priority interrupt) of interrupt lines.

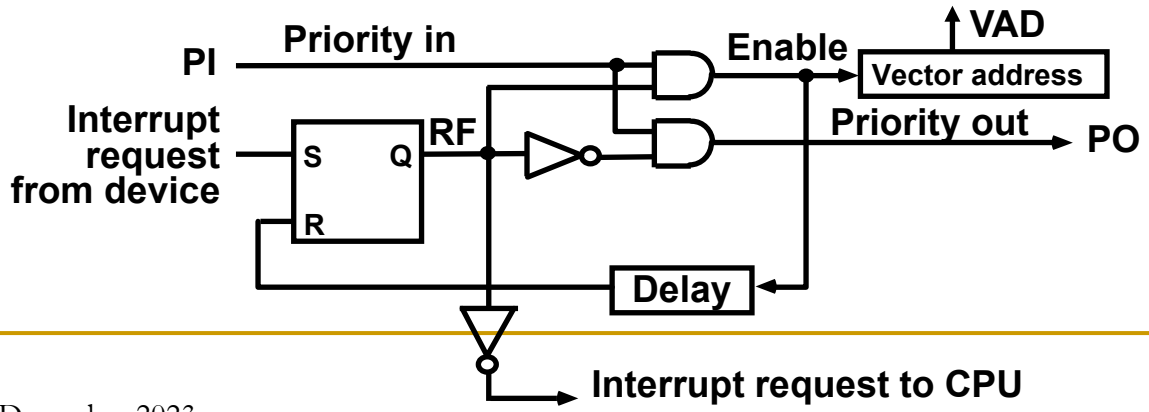


HARDWARE PRIORITY INTERRUPT - DAISY-CHAIN -



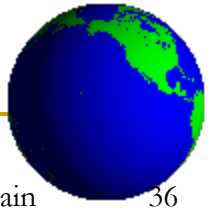
PI	RF	PO	Enable
0	0	0	0
0	1	0	0
1	0	1	0
1	1	0	1

One stage of the daisy chain priority arrangement



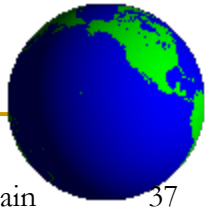
Daisy- chaining priority

- The interrupt request line is common to all devices
- If any device has its interrupt signal in the low-level state, the interrupt line goes to the low-level state and enables the interrupt input in the CPU
- When no interrupts are pending, the interrupt line remains in the high level state- negative OR operation
- The farther the device is from the first position, the lower is its priority.

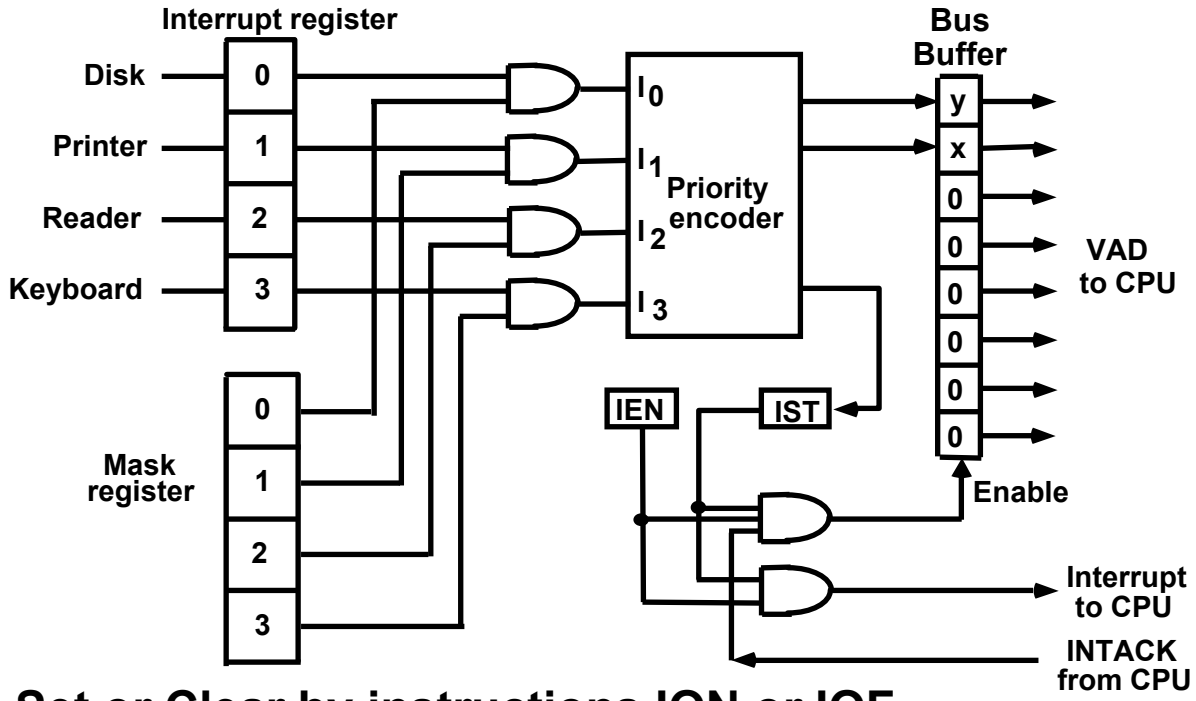


Daisy chaining- internal logic

- The device sets its RF flip flop when it wants to interrupt the CPU
- PI=0 both PO and enable line to VAD are equal to 0 irrespective of RF
- PI=1, RF=0 then PO=1 and VAD is disabled. This condition passes the acknowledge signal to the next device through PO
- PI=1, RF=1 device is active. PO=0 and enables the VAD for the data bus
- The RF flip flop is reset after a sufficient delay to ensure that the CPU has received the VAD



PARALLEL PRIORITY INTERRUPT



IEN: Set or Clear by instructions ION or IOF

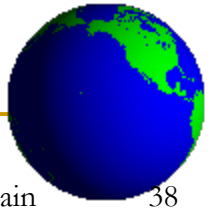
IST: Represents an unmasked interrupt has occurred. INTACK enables tri state Bus Buffer to load VAD generated by the Priority Logic

Interrupt Register:

- Each bit is associated with an Interrupt Request from different Interrupt Source - different priority level
- Each bit can be cleared by a program instruction

Mask Register:

- Mask Register is associated with Interrupt Register
- Each bit can be set or cleared by an Instruction

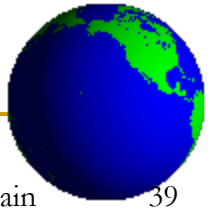


INTERRUPT PRIORITY ENCODER

Determines the highest priority interrupt when more than one interrupts take place

Priority Encoder Truth table

Inputs				Outputs			Boolean functions
I_0	I_1	I_2	I_3	x	y	IST	
1	d	d	d	0	0	1	
0	1	d	d	0	1	1	
0	0	1	d	1	0	1	$x = I_0' \; I_1'$
0	0	0	1	1	1	1	$y = I_0' \; I_1 + I_0' \; I_2'$
0	0	0	0	d	d	0	$(IST) = I_0 + I_1 + I_2 + I_3$



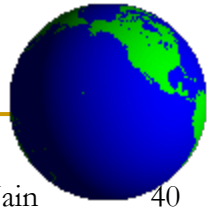
INTERRUPT CYCLE

At the end of each Instruction cycle

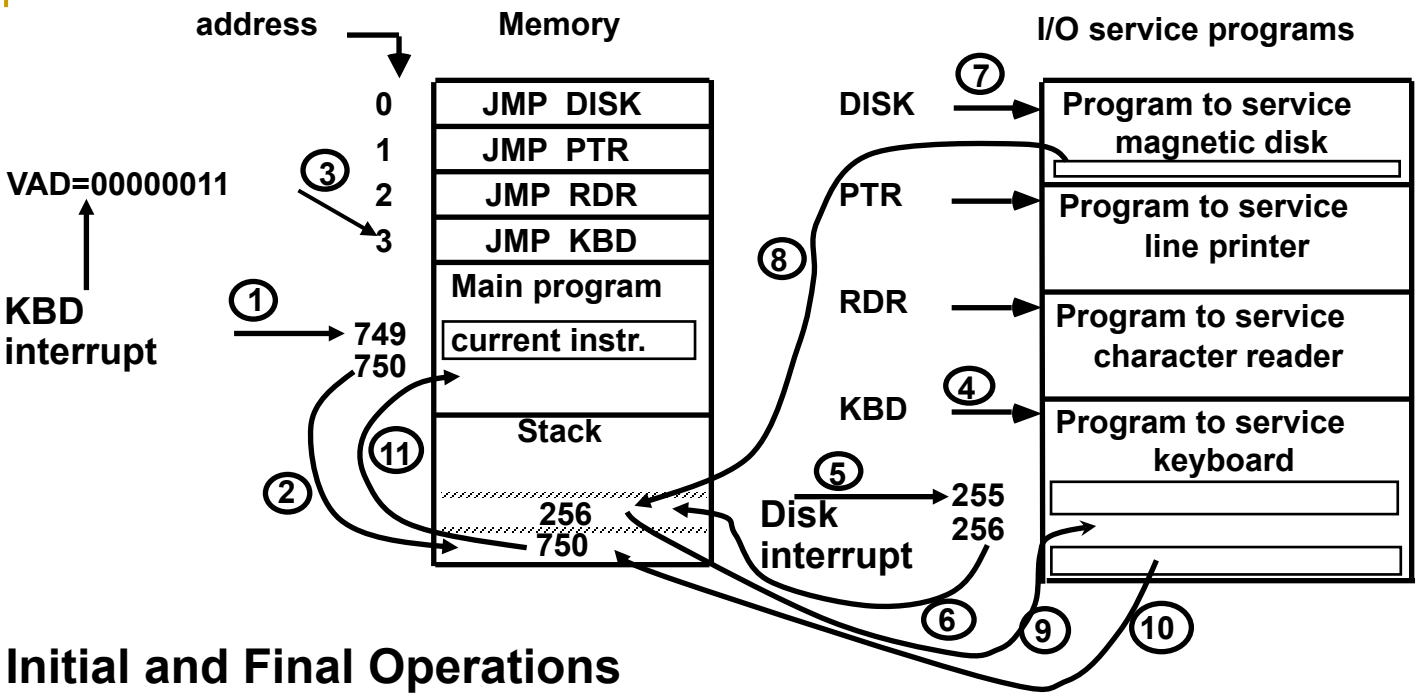
- CPU checks IEN and IST

- If IEN • IST = 1, CPU -> Interrupt Cycle

SP ← SP - 1	**Decrement stack pointer**
M[SP] ← PC	**Push PC into stack**
INTACK ← 1	**Enable interrupt acknowledge**
PC ← VAD	**Transfer vector address to PC**
IEN ← 0	**Disable further interrupts**
Go To Fetch	**to execute the first instruction in the interrupt service routine**



INTERRUPT SERVICE ROUTINE



Initial and Final Operations

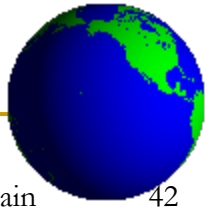
Each interrupt service routine must have an initial and final set of operations for controlling the registers in the hardware interrupt system

Initial Sequence
- [1] Clear lower level Mask reg. bits
- [2] IST <- 0
- [3] Save contents of CPU registers
- [4] IEN <- 1
- [5] Go to Interrupt Service Routine

Final Sequence
- [1] IEN <- 0
- [2] Restore CPU registers
- [3] Clear the bit in the Interrupt Reg
- [4] Set lower level Mask reg. bits
- [5] Restore return address, IEN <- 1

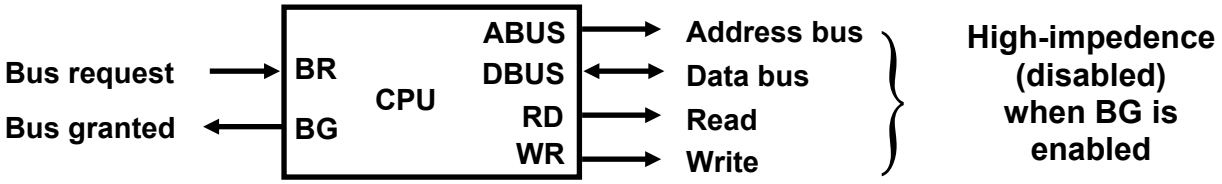
DMA (Direct Memory Access)

- **Large blocks of data transferred at a high speed to or from high speed devices, magnetic drums, disks, tapes, etc.**
- **DMA controller**
  - **Interface that provides I/O transfer of data directly to and from the memory and the I/O device**
  - **CPU initializes the DMA controller by sending a memory address and the number of words to be transferred**
  - **Actual transfer of data is done directly between the device and memory through DMA controller**
- **Freeing CPU for other tasks**

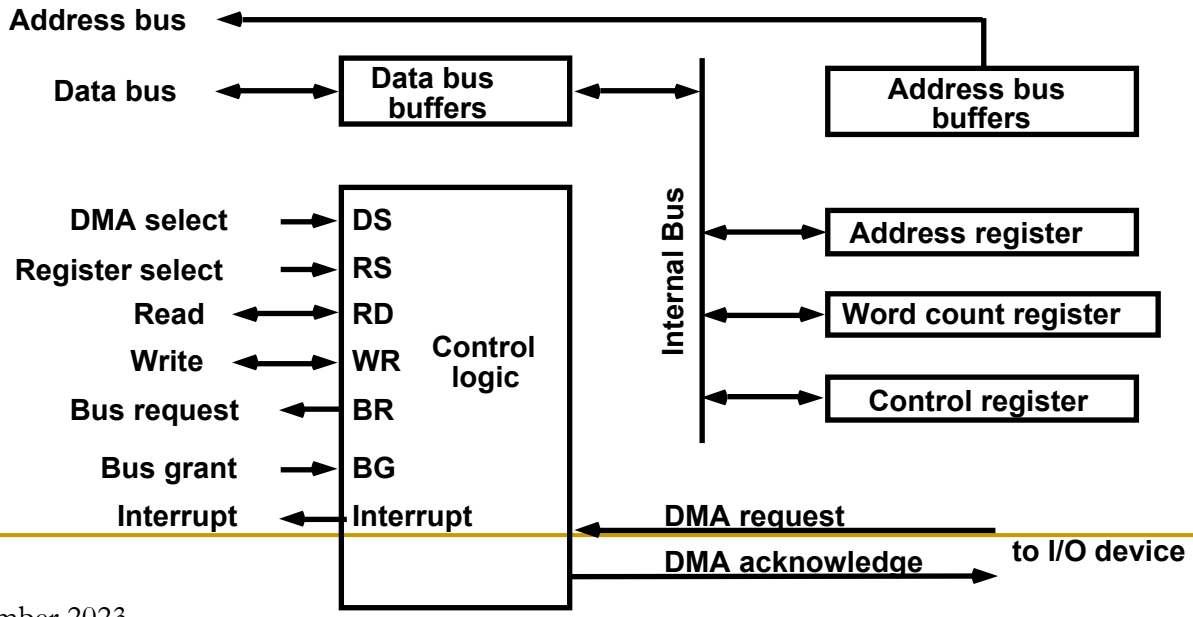


DIRECT MEMORY ACCESS

CPU bus signals for DMA transfer



Block diagram of DMA controller



DMA I/O OPERATION

Starting an I/O

- CPU executes instruction to

Load Memory Address Register
Load Word Counter
Load Function(Read or Write) to be performed
Issue a GO command

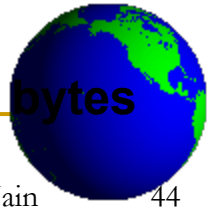
Upon receiving a GO Command DMA performs I/O operation as follows independently from CPU

Input

[1] Input Device <- R (Read control signal)
[2] Buffer(DMA Controller) <- Input Byte; and assembles the byte into a word until word is full
[4] M <- memory address, W(Write control signal)
[5] Address Reg <- Address Reg +1; WC(Word Counter) <- WC - 1
[6] If WC = 0, then Interrupt to acknowledge done, else go to [1]

Output

[1] M <- M Address, R
M Address R <- M Address R + 1, WC <- WC - 1
[2] Disassemble the word
[3] Buffer <- One byte; Output Device <- W, for all disassembled bytes
[4] If WC = 0, then Interrupt to acknowledge done, else go to [1]



CYCLE STEALING

While DMA I/O takes place, CPU is also executing instructions

DMA Controller and CPU both access Memory -> Memory Access Conflict

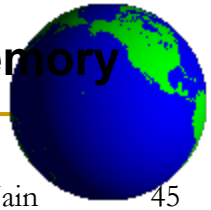
Memory Bus Controller

- Coordinating the activities of all devices requesting memory access
- Priority System

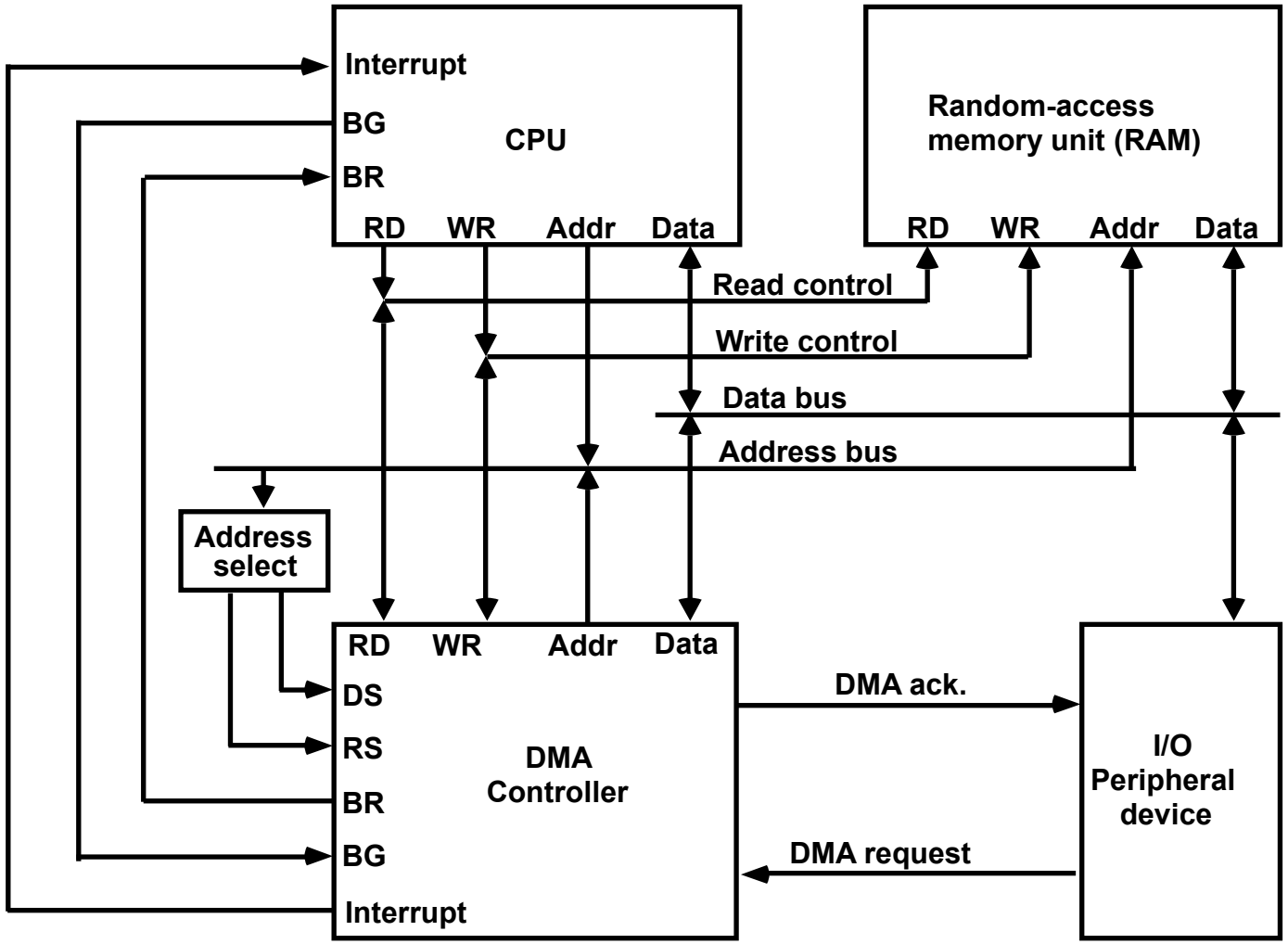
Memory accesses by CPU and DMA Controller are interwoven, with the top priority given to DMA Controller
-> Cycle Stealing

Cycle Steal

- CPU is usually much faster than I/O(DMA), thus CPU uses the most of the memory cycles
- DMA Controller steals the memory cycles from CPU
- For those stolen cycles, CPU remains idle
- For those slow CPU, DMA Controller may steal most of the memory cycles which may cause CPU remain idle long time



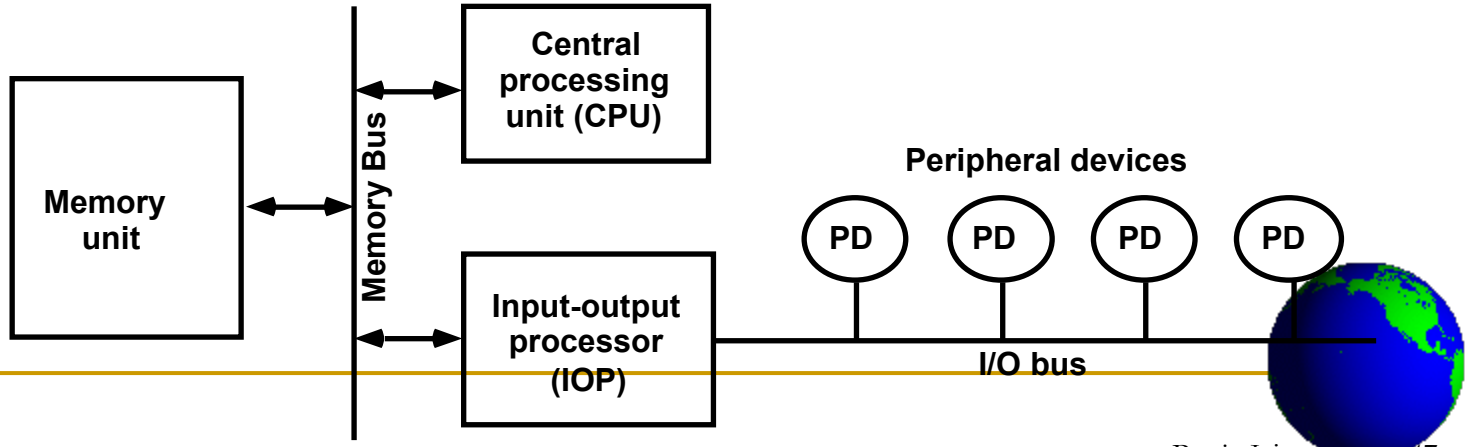
DMA TRANSFER



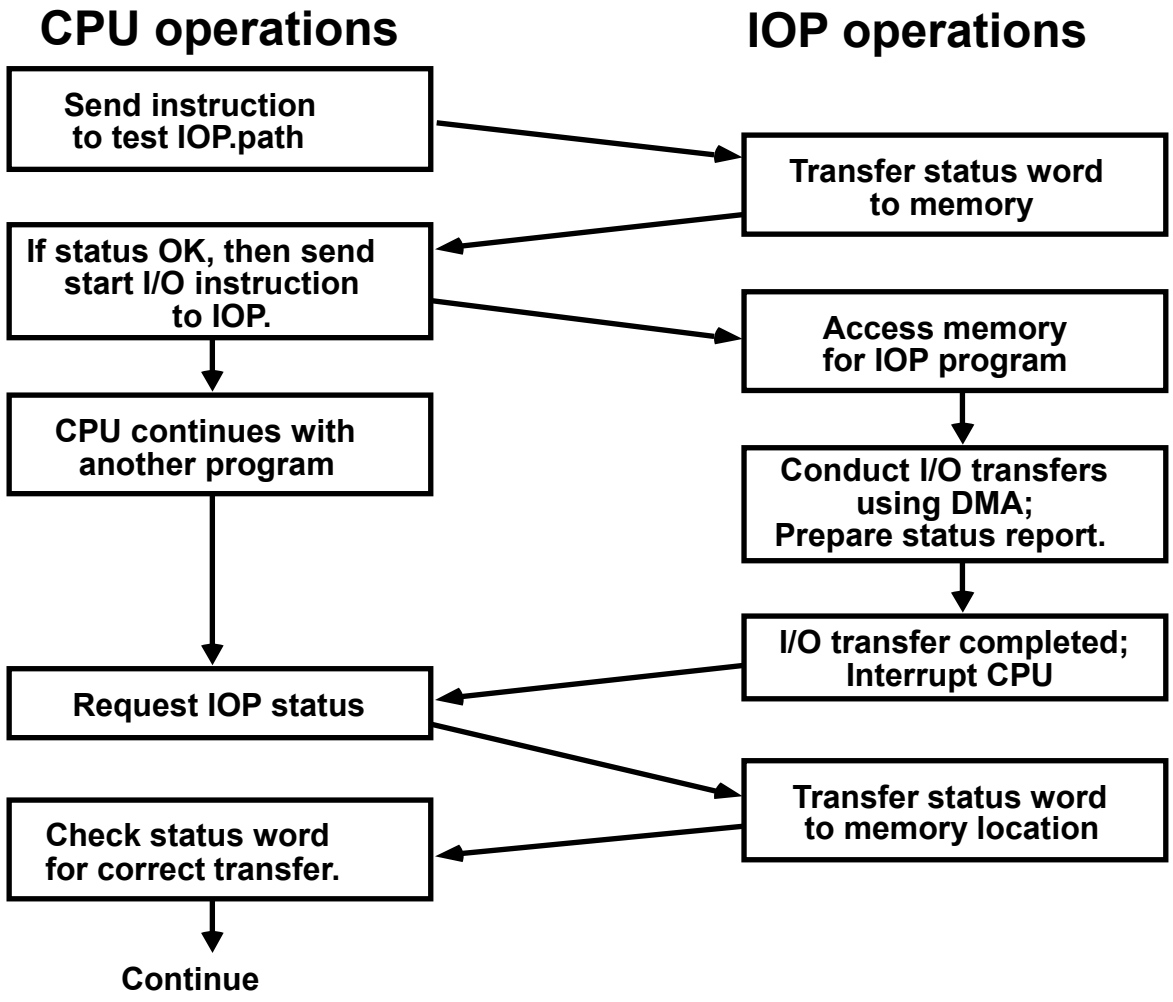
INPUT/OUTPUT PROCESSOR - CHANNEL -

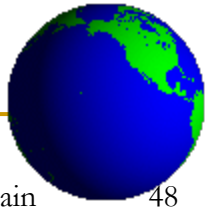
Channel

- **Processor with direct memory access capability that communicates with I/O devices**
- **Channel accesses memory by cycle stealing**
- **Channel can execute a Channel Program**
  - **Stored in the main memory**
  - **Consists of Channel Command Word(CCW)**
  - **Each CCW specifies the parameters needed by the channel to control the I/O devices and perform data transfer operations**
- **CPU initiates the channel by executing an channel I/O class instruction and once initiated, channel operates independently of the CPU**



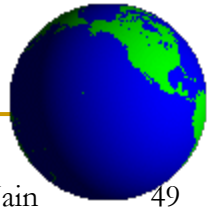
CHANNEL / CPU COMMUNICATION





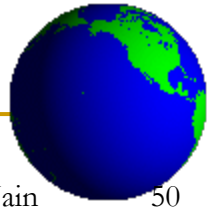
Serial communication

- A data communication processor(DCP) is an I/O processor that distributes and collects data from many remote terminals connected through telephone and other communication lines.
- IOP communicates with the peripherals through a common I/O bus, DCP communicates with each terminal through a single pair of wires.



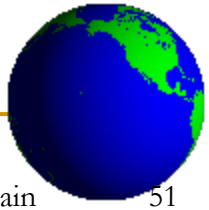
Modem

- A modem converts digital signals into audio tones to be transmitted over telephone lines and also converts audio tones from the line to digital signals for machine use.
- The modems used in synchronous transmission have internal clocks that are set to frequency that bits are being transmitted in the communication line.



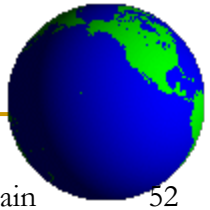
Transmission

- Simplex
- Half duplex
  - Turn around time
  - Pair of wires
- Full duplex
  - Four wire link
- Data link
- Protocol
  - Character oriented protocol
  - Bit oriented protocol



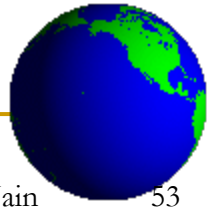
Character oriented protocol

- Character oriented protocols are also known as **byte oriented protocols** that interpret a transmission frame as a character and usually contain one byte (or eight bits). In all data link protocols, control information can be inserted separately or as an addition to existing data frames. In character oriented protocols, this information is in the form of code words. These code words carry information about the flow control, error control and line discipline.



Message format

SYN	SYN	SOH	Header	STX	Text	ETX	BCC

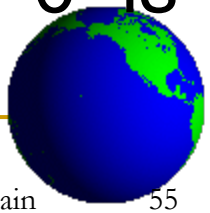


Bit oriented protocol

- A **bit-oriented protocol** is a communications protocol that sees the transmitted data as an *opaque* stream of bits with no semantics, or meaning. Control codes are defined in terms of bit sequences instead of characters. Bit oriented protocol can transfer data frames regardless of frame contents. It can also be stated as "bit stuffing" this technique allows the data frames to contain an arbitrary number of bits and allows character codes with arbitrary number of bits per character.

Frame format

- Flag 01111110
- Address – 8 bit
- Control – 8 bits
- Information field is not restricted in format or content and can be of any length.
- The frame check field is a CRC sequence.
- Zero insertion- to prevent a flag from occurring in the middle of a frame, a 0 is inserted after 5 consecutive 1's



Frame format for Bit oriented protocol

Flag	Address	Control	Data.....	FCS	FCS	Flag