Unit -3

CPU design

Instruction cycle

- **Memory address registers(MAR)** : It is connected to the address lines of the system bus. It specifies the address in memory for a read or write operation.
- **Memory Buffer Register(MBR)** : It is connected to the data lines of the system bus. It contains the value to be stored in memory or the last value read from the memory.
- **Program Counter(PC)** : Holds the address of the next instruction to be fetched.
- **Instruction Register(IR)** : Holds the last instruction fetched.

In computer organization, an instruction cycle, also known as a fetch-decode-execute cycle, is the basic operation performed by a central processing unit (CPU) to execute an instruction. The instruction cycle consists of several steps, each of which performs a specific function in the execution of the instruction. The major steps in the instruction cycle are:

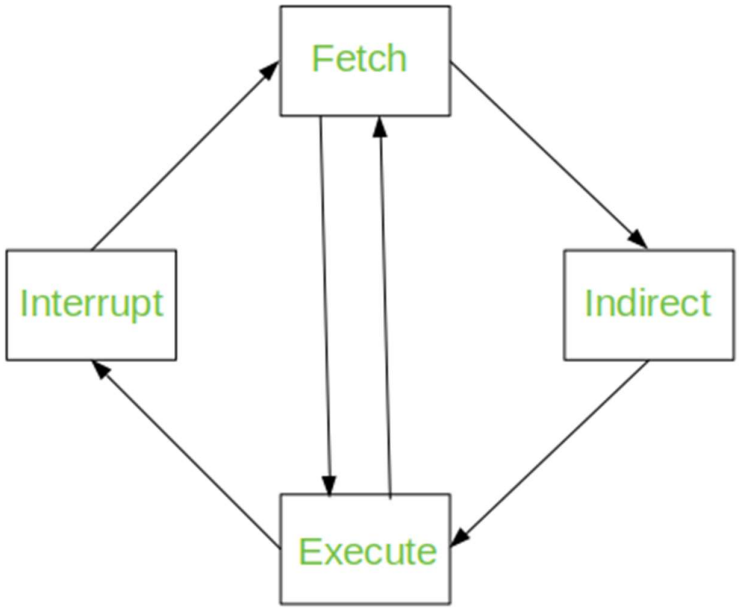
1. Fetch: In the fetch cycle, the CPU retrieves the instruction from memory. The instruction is typically stored at the address specified by the program counter (PC). The PC is then incremented to point to the next instruction in memory.
2. Decode: In the decode cycle, the CPU interprets the instruction and determines what operation needs to be performed. This involves identifying the opcode and any operands that are needed to execute the instruction.
3. Execute: In the execute cycle, the CPU performs the operation specified by the instruction. This may involve reading or writing data from or to memory, performing arithmetic or logic operations on data, or manipulating the control flow of the program.
4. There are also some additional steps that may be performed during the instruction cycle, depending on the CPU architecture and instruction set:

5. Fetch operands: In some CPUs, the operands needed for an instruction are fetched during a separate cycle before the execute cycle. This is called the fetch operands cycle.
6. Store results: In some CPUs, the results of an instruction are stored during a separate cycle after the execute cycle. This is called the store results cycle.
7. Interrupt handling: In some CPUs, interrupt handling may occur during any cycle of the instruction cycle. An interrupt is a signal that the CPU receives from an external device or software that requires immediate attention. When an interrupt occurs, the CPU suspends the current instruction and executes an interrupt handler to service the interrupt.

These cycles are the basic building blocks of the CPU's operation and are performed for every instruction executed by the CPU. By optimizing these cycles, CPU designers can improve the performance and efficiency of the CPU, allowing it to execute instructions faster and more efficiently.

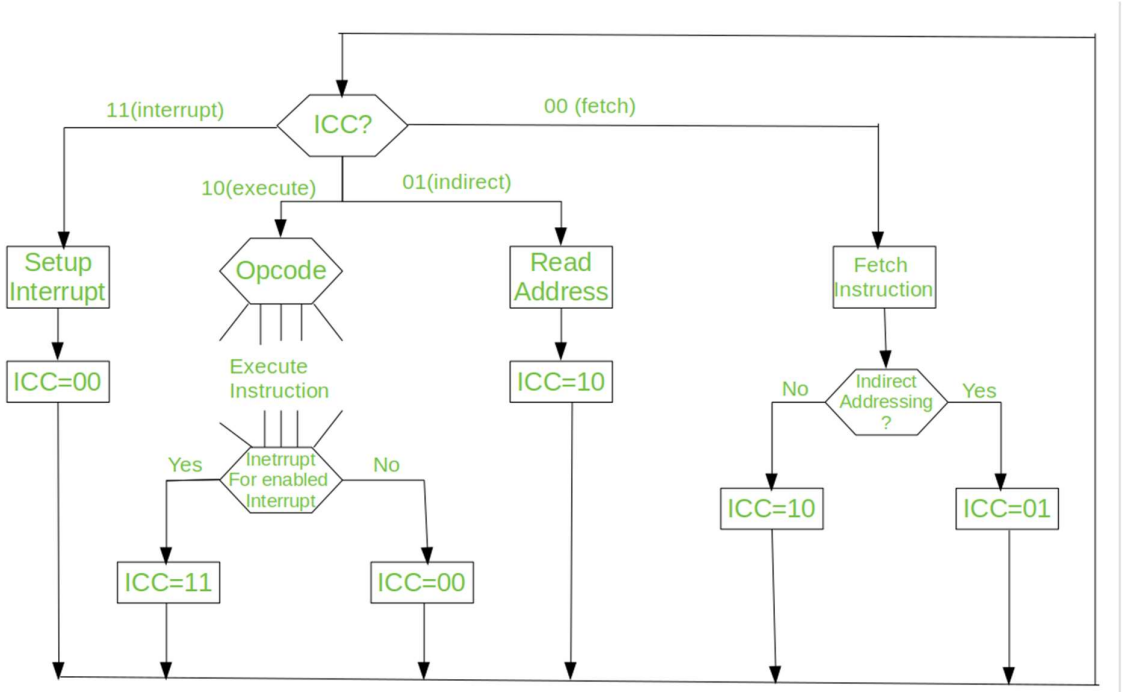
The Instruction Cycle –

Each phase of Instruction Cycle can be decomposed into a sequence of elementary micro-operations. In the above examples, there is one sequence each for the *Fetch, Indirect, Execute and Interrupt Cycles*.



The Instruction Cycle

The *Indirect Cycle* is always followed by the *Execute Cycle*. The *Interrupt Cycle* is always followed by the *Fetch Cycle*. For both fetch and execute cycles, the next cycle depends on the state of the system.

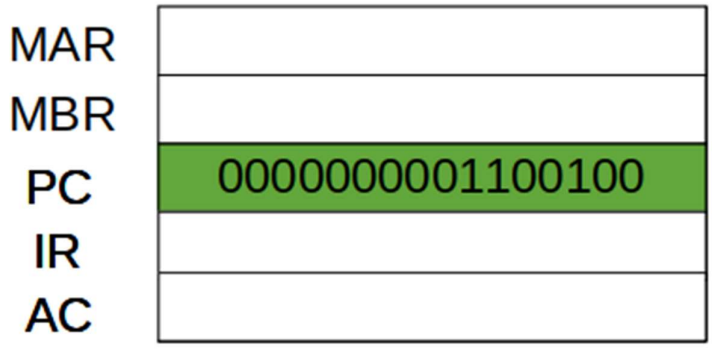


Flowchart for Instruction Cycle

We assumed a new 2-bit register called *Instruction Cycle Code* (ICC). The ICC designates the state of processor in terms of which portion of the cycle it is in:-
00 : Fetch Cycle
01 : Indirect Cycle
10 : Execute Cycle
11 : Interrupt Cycle

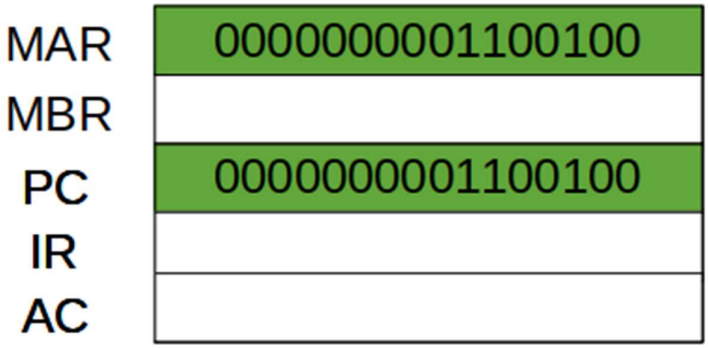
At the end of the each cycles, the ICC is set appropriately. The above flowchart of *Instruction Cycle* describes the complete sequence of micro-operations, depending only on the instruction sequence and the interrupt pattern(this is a simplified example). The operation of the processor is described as the performance of a sequence of micro-operation.
Different Instruction Cycles:

- **The Fetch Cycle –**
  At the beginning of the fetch cycle, the address of the next instruction to be executed is in the *Program Counter*(PC).



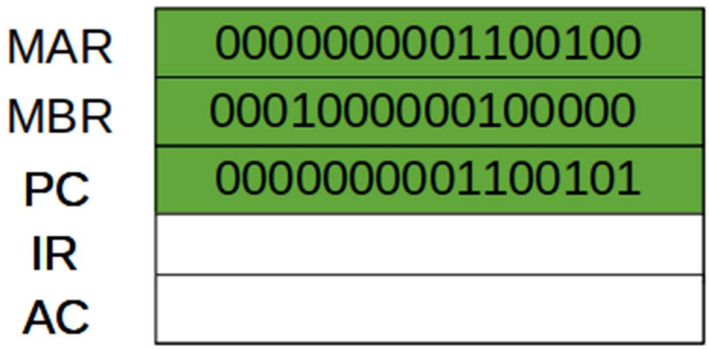
BEGINNING

- Step 1: The address in the program counter is moved to the memory address register(MAR), as this is the only register which is connected to address lines of the system bus.



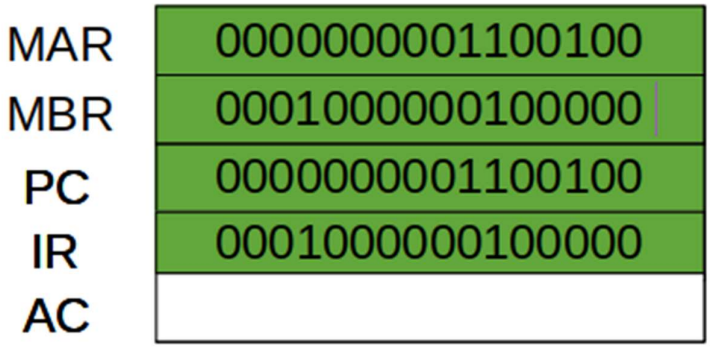
FIRST STEP

- Step 2: The address in MAR is placed on the address bus, now the control unit issues a READ command on the control bus, and the result appears on the data bus and is then copied into the memory buffer register(MBR). Program counter is incremented by one, to get ready for the next instruction. (These two action can be performed simultaneously to save time)



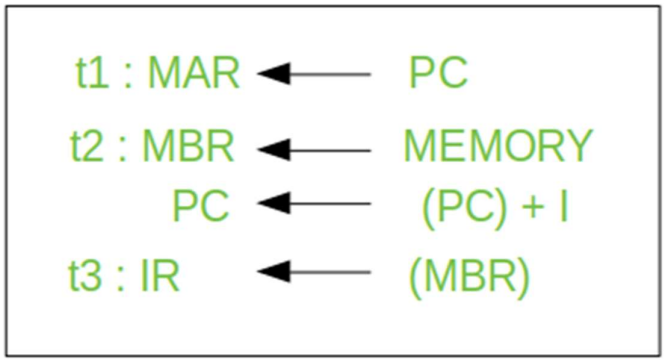
SECOND STEP

- Step 3: The content of the MBR is moved to the instruction register(IR).



FIRST STEP

- Thus, a simple *Fetch Cycle* consist of three steps and four micro-operation. Symbolically, we can write these sequence of events as follows:-



- Here 'I' is the instruction length. The notations (t1, t2, t3) represents successive time units. We assume that a clock is available for timing purposes and it emits regularly spaced clock pulses. Each clock pulse defines a time unit. Thus, all time units are of equal duration. Each micro-operation can be performed within the time of a single time unit.
  First time unit: Move the contents of the PC to MAR.
  Second time unit: Move contents of memory location specified by MAR to MBR. Increment content of PC by I.
  Third time unit: Move contents of MBR to IR.
  Note: Second and third micro-operations both take place during the second time unit.

- **The Indirect Cycles –**
  Once an instruction is fetched, the next step is to fetch source operands. *Source Operand* is being fetched by indirect addressing(it can be fetched by any addressing mode, here its done by indirect addressing). Register-based operands need not be fetched. Once the opcode is executed, a similar process may be needed to store the result in main memory. Following *micro-operations* takes place:-

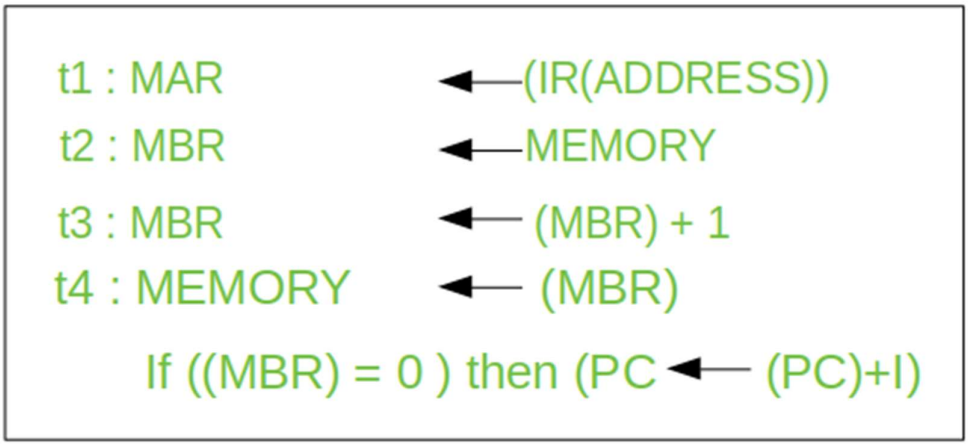
- Step 1: The address field of the instruction is transferred to the MAR. This is used to fetch the address of the operand.
  Step 2: The address field of the IR is updated from the MBR.(So that it now contains a direct addressing rather than indirect addressing)
  Step 3: The IR is now in the state, as if indirect addressing has not been occurred.
  Note: Now IR is ready for the execute cycle, but it skips that cycle for a moment to consider the *Interrupt Cycle* .

- **The Execute Cycle**
  The other three cycles(*Fetch, Indirect and Interrupt*) are simple and predictable. Each of them requires simple, small and fixed sequence of micro-operation. In each case same micro-operation are repeated each time around.
  Execute Cycle is different from them. Like, for a machine with N different opcodes there are N different sequence of micro-operations that can occur.
  Lets take an hypothetical example :-
  consider an add instruction:

- Here, this instruction adds the content of location X to register R. Corresponding micro-operation will be:-

- We begin with the IR containing the ADD instruction.
  Step 1: The address portion of IR is loaded into the MAR.
  Step 2: The address field of the IR is updated from the MBR, so the reference memory location is read.
  Step 3: Now, the contents of R and MBR are added by the ALU.
  Lets take a complex example :-

- Here, the content of location X is incremented by 1. If the result is 0, the next instruction will be skipped. Corresponding sequence of micro-operation will be :-



- Here, the PC is incremented if (MBR) = 0. This test (is MBR equal to zero or not) and action (PC is incremented by 1) can be implemented as one micro-operation.
  Note : This test and action micro-operation can be performed during the same time unit during which the updated value MBR is stored back to memory.

- **The Interrupt Cycle**:
  At the completion of the Execute Cycle, a test is made to determine whether any enabled interrupt has occurred or not. If an enabled interrupt has occurred then Interrupt Cycle occurs. The nature of this cycle varies greatly from one machine to another. Lets take a sequence of micro-operation:-

- Step 1: Contents of the PC is transferred to the MBR, so that they can be saved for return.
  Step 2: MAR is loaded with the address at which the contents of the PC are to be saved.
  PC is loaded with the address of the start of the interrupt-processing routine.
  Step 3: MBR, containing the old value of PC, is stored in memory.
  Note: In step 2, two actions are implemented as one micro-operation. However, most processor provide multiple types of interrupts, it may take one or more micro-operation to obtain the save_address and the routine_address before they are transferred to the MAR and PC respectively.

Uses of Different Instruction Cycles :

Here are some uses of different instruction cycles:

1. **Fetch cycle:** This cycle retrieves the instruction from memory and loads it into the processor's instruction register. The fetch cycle is essential for the processor to know what instruction it needs to execute.
2. **Decode cycle:** This cycle decodes the instruction to determine what operation it represents and what operands it requires. The decode cycle is important for the processor to understand what it needs to do with the instruction and what data it needs to retrieve or manipulate.
3. **Execute cycle:** This cycle performs the actual operation specified by the instruction, using the operands specified in the instruction or in other registers. The execute cycle is where the processor performs the actual computation or manipulation of data.
4. **Store cycle:** This cycle stores the result of the operation in memory or in a register. The store cycle is essential for the processor to save the result of the computation or manipulation for future use.

The advantages and disadvantages of the instruction cycle depend on various factors, such as the specific CPU architecture and the instruction set used. However, here are some general advantages and disadvantages of the instruction cycle:

Advantages:

1. Standardization: The instruction cycle provides a standard way for CPUs to execute instructions, which allows software developers to write programs that can run on multiple CPU architectures. This standardization also makes it easier for hardware designers to build CPUs that can execute a wide range of instructions.
2. Efficiency: By breaking down the instruction execution into multiple steps, the CPU can execute instructions more efficiently. For example, while the CPU is performing the execute cycle for one instruction, it can simultaneously fetch the next instruction.
3. Pipelining: The instruction cycle can be pipelined, which means that multiple instructions can be in different stages of execution at the same time. This improves the overall performance of the CPU, as it can process multiple instructions simultaneously.

Disadvantages:

1. Overhead: The instruction cycle adds overhead to the execution of instructions, as each instruction must go through multiple stages before it can be executed. This overhead can reduce the overall performance of the CPU.
2. Complexity: The instruction cycle can be complex to implement, especially if the CPU architecture and instruction set are complex. This complexity can make it difficult to design, implement, and debug the CPU.
3. Limited parallelism: While pipelining can improve the performance of the CPU, it also has limitations. For example, some instructions may depend on the results of previous instructions, which limits the amount of parallelism that can be achieved. This can reduce the effectiveness of pipelining and limit the overall performance of the CPU.

Issues of Different Instruction Cycles :

Here are some common issues associated with different instruction cycles:

1. **Pipeline hazards:** Pipelining is a technique used to overlap the execution of multiple instructions by breaking them into smaller stages. However, pipeline hazards occur when one instruction depends on the completion of a previous instruction, leading to delays and reduced performance.

2. **Branch prediction errors:** Branch prediction is a technique used to anticipate which direction a program will take when encountering a conditional branch instruction. However, if the prediction is incorrect, it can result in wasted cycles and decreased performance.
3. **Instruction cache misses:** Instruction cache is a fast memory used to store frequently used instructions. Instruction cache misses occur when an instruction is not found in the cache and needs to be retrieved from slower memory, resulting in delays and decreased performance.
4. **Instruction-level parallelism limitations:** Instruction-level parallelism is the ability of a processor to execute multiple instructions simultaneously. However, this technique has limitations as not all instructions can be executed in parallel, leading to reduced performance in some cases.
5. **Resource contention:** Resource contention occurs when multiple instructions require the use of the same resource, such as a register or a memory location. This can lead to delays and reduced performance if the processor is unable to resolve the contention efficiently.

Data representation

Data Representation in Computer Organization

In computer organization, data refers to the symbols that are used to represent events, people, things and ideas.

Data Representation

The data can be represented in the following ways:

Data

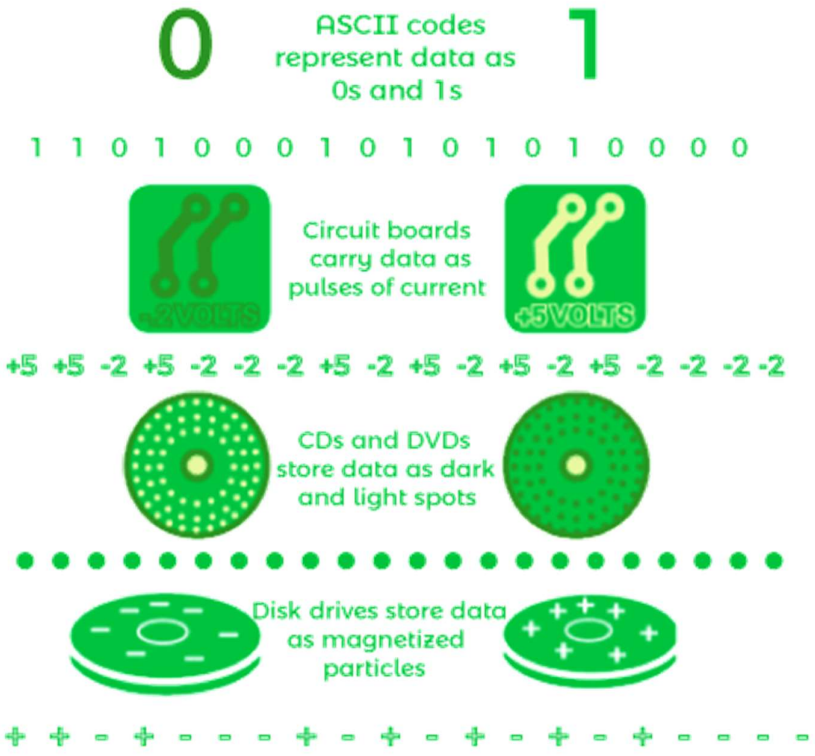
Data can be anything like a number, a name, notes in a musical composition, or the color in a photograph. Data representation can be referred to as the form in which we stored the data, processed it and transmitted it. In order to store the data in digital format, we can use any device like computers, smartphones, and iPads. Electronic circuitry is used to handle the stored data.

Digitization

Digitization is a type of process in which we convert information like photos, music, number, text into digital data. Electronic devices are used to manipulate these types of data. The digital revolution has evolved with the help of 4 phases, starting with the big, expensive standalone computers and progressing to today's digital world. All around the world, small and inexpensive devices are spreading everywhere.

Binary Digits

The **binary digits** or bits are used to show the digital data, which is represented by 0 and 1. The binary digits can be called the smallest unit of information in a computer. The main use of binary digit is that it can store the information or data in the form of 0s and 1s. It contains a value that can be on/off or true/false. On or true will be represented by the 1, and off or false will be represented by the 0. The digital file is a simple file, which is used to collect data contained by the storage medium like the flash drive, CD, hard disk, or DVD.



Representing Numbers

The number can be represented in the following way:

Numeric Data

Numeric data is used to contain numbers, which helps us to perform arithmetic operations. The digital devices use a binary number system so that they can represent numeric data. The binary number system can only be represented by two digits 0 and 1. There can't be any other digits like 2 in the system. If we want to represent number 2 in binary, then we will write it as 10.

Representing Numbers

DECIMAL (BASE 10)	BINARY (BASE 2)
0	0
1	1
2	10
3	11
4	100
5	101
6	110
7	111
8	1000
9	1001
10	1010
11	1011
1000	1111101000

Representing Text

The text can be represented in the following ways:

Character Data

Character data can be formed with the help of symbols, letters, and numerals, but they can?t be used in calculations. Using the character data, we can form our address, hair colour, name, etc. Character data normally takes the data in the form of text. With the help of the text, we can describe many things like our father name, mother name, etc.

Digital Devices

Several types of codes are employed by the **digital devices** to represent character data, including Unicode, ASCII, and other types of variants. The full form of ASCII is American Standard Code for Information Interchange. It is a type of character encoding standard, which is used for electronic communication. With the help of telecommunication equipment, computers and many other devices, ASCII code can represent the text. The ASCII code needs 7 bits for each character, where the unique character is represented by every single bit. For the uppercase letter A, the ASCII code is represented as 1000001.

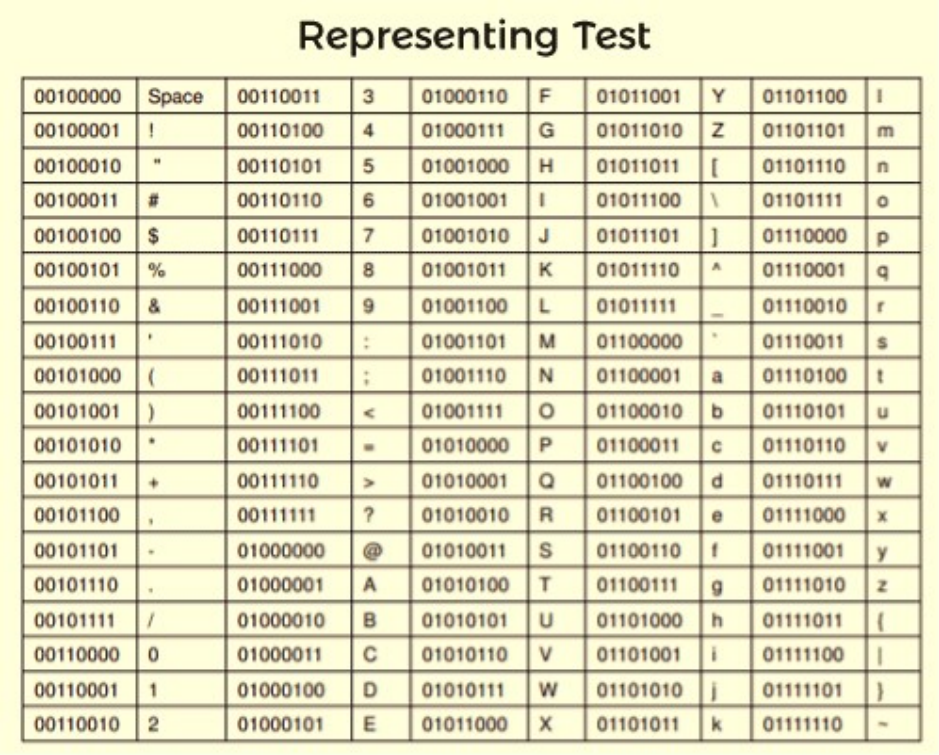
Extended ASCII

Extended ASCII can be described as a superset of ASCII. The ASCII set uses 7 bits to represent every character, but the Extended ASCII uses 8 bits to represent each character. The extended ASCII contains 7 bits of ASCII characters and 1 bit for additional characters. Using the 7 bits, the ASCII code provides code for 128 unique symbols or characters, but Extended ASCII provides code for 256 unique symbols or characters. For the uppercase letter A, the Extended ASCII code is represented as 01000001.

Unicode

Unicode is also known as the universal character encoding standard. Unicode provides a way through which an individual character can be represented in the form of web pages, text files, and other documents. Using ASCII, we can only represent the basic English characters, but with the help of Unicode, we can represent characters from all languages around the World.

ASCII code provides code for 128 characters, while Unicode provide code for roughly 65,000 characters with the help of 16 bits. In order to represent each character, ASCII code only uses 1 bit, while Unicode supports up to 4 bytes. The Unicode encoding has several different types, but UTF-8 and UTF-16 are the most commonly used. UTF-8 is a type of variable length coding scheme. It has also become the standard character encoding, which is used on the web. Many software programs also set UTF-8 as their default encoding.



Representing Test

00100000	Space	00110011	3	01000110	F	01011001	Y	01101100	l
00100001	!	00110100	4	01000111	G	01011010	Z	01101101	m
00100010	"	00110101	5	01001000	H	01011011	[	01101110	n
00100011	#	00110110	6	01001001	I	01011100	\	01101111	o
00100100	$	00110111	7	01001010	J	01011101	]	01110000	p
00100101	%	00111000	8	01001011	K	01011110	^	01110001	q
00100110	&	00111001	9	01001100	L	01011111	_	01110010	r
00100111	'	00111010	:	01001101	M	01100000	`	01110011	s
00101000	(	00111011	;	01001110	N	01100001	a	01110100	t
00101001	)	00111100	<	01001111	O	01100010	b	01110101	u
00101010	*	00111101	=	01010000	P	01100011	c	01110110	v
00101011	+	00111110	>	01010001	Q	01100100	d	01110111	w
00101100	,	00111111	?	01010010	R	01100101	e	01111000	x
00101101	-	01000000	@	01010011	S	01100110	f	01111001	y
00101110	.	01000001	A	01010100	T	01100111	g	01111010	z
00101111	/	01000010	B	01010101	U	01101000	h	01111011	{
00110000	0	01000011	C	01010110	V	01101001	i	01111100	|
00110001	1	01000100	D	01010111	W	01101010	j	01111101	}
00110010	2	01000101	E	01011000	X	01101011	k	01111110	~

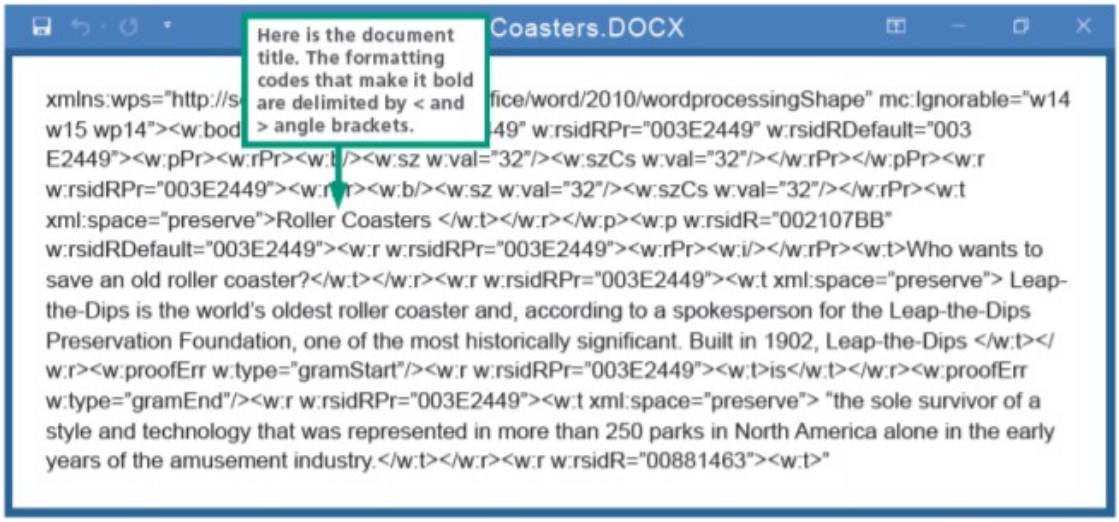
ASCII Code

ASCII code can be used for numerals like phone numbers and social security numbers. ASCII text contains plain and unformatted text. This type of file will be saved in a text file format, which contains a name ending with .txt. These files are labelled differently on different systems, like Windows operating system labelled these files as "Text document" and Apple devices labelled these files as "Plain Text". There will have no formatting in the ASCII text files. If we want to make the documents with styles and formats, then we have to embed formatting codes in the text.

Microsoft Excel

Microsoft word is used to create formatted text and documents. It uses the **DOCX format** to do this. If we create a new document using the Microsoft Word 2007 or later version, then it always uses DOCX as the default file format. **Apple pages** use **PAGES format** to produce the documents. As compared to Microsoft Word, it is simpler to create and edit documents using page format. **Adobe Acrobat** uses the **PDF format** to create the documents. The files that saved in the PDF format cannot be modified. But we can easily print and share these files. If we save our document in PDF format, then we cannot change that file into the Microsoft Office file or any other file without specified software.

HTML is the hypertext markup language. It is used for document designing, which will be displayed in a web browser. It uses **HTML format** to design the documents. In HTML, hypertext is a type of text in any document containing links through which we can go to other places in the document or in other documents also. The markup language can be called as a computer language. In order to define the element within a document, this language uses tags.



Representing Bits and Bytes

The bits and bytes can be represented in the following ways:

Bits and Bytes

In the field of digital communication or computers, bits are the most basic unit of information or smallest unit of data. It is short of binary digit, which means it can contain only one value, either 0 or 1. So bits can be represented by 0 or 1, - or +, false or true, off or on, or no or yes. Many technologies are based on bits and bytes, which is extensively useful to describe the network access speed and storage capacity. The bit is usually abbreviated as a lowercase b.

In order to execute the instructions and store the data, the bits are grouped into multiple bits, which are known as bytes. Bytes can be defined as a group of eight bits, and it is usually abbreviated as an uppercase B. If we have four bytes, it will equal 32 bits (4*8 = 32), and 10 bytes will equal 80 bits (8*10 = 80).

Uses

Bits are used for data rates like speeds while movie download, speed while internet connection, etc. Bytes are used to get the storage capacity and file sizes. When we are reading something related to digital devices, it will be frequently encountered

references like 90 kilobits per second, 1.44 megabytes, 2.8 gigahertz, and 2 terabytes. To quantify digital data, we have many options such as Kilo, Mega, Giga, Tera and many more similar terms, which are described as follows:

104 KB: Kb is also called a kilobyte or Kbyte. It is mostly used while referring to the size of small computer files.

56 Kbps: Kbps is also called kilobit, Kbit or Kb. The 56 kbps means 56 kilobits per second which are used to show the slow data rates. If our internet speed is 56 kbps, we have to face difficulty while connecting more than one device, buffering while streaming videos, slow downloading, and many other internet connectivity problems.

50 Mbps: Mbps is also called Megabit, MB or Mbit. The 50 Mbps means 50 Megabit per second, which are used to show the faster data rates. If our internet speed is 50 Mbps, we will experience online activity without any buffering, such as online gaming, downloading music, streaming HD, web browsing, etc. 50 Mbps or more than that will be known as fast internet speed. With the help of fast speed, we can easily handle more than one online activity for more than one user at a time without major interruptions in services.

3.2 MB: 3.2 MB is also called Megabyte, MB or MByte. It is used when we are referring to the size of files, which contains videos and photos.

100 Gbit: 100 Gbit is also called Gigabit or GB. It is used to show the really fast network speeds.

16 GB: 16 GB is also called Gigabyte, GB or GByte. It is used to show the storage capacity.

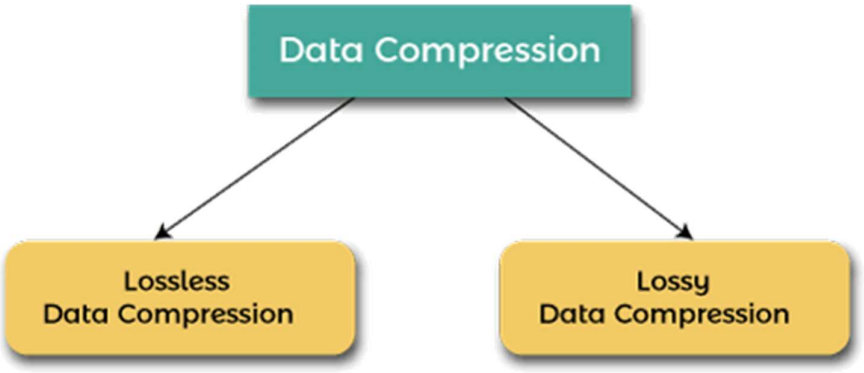
Bites and Bytes

Bit	One binary digit
Byte	8 bits
Kilobit	1,024 or 2^{10} bits
Kilobyte	1,024 or 2^{10} bytes
Megabit	1,048,576 or 2^{20} bits
Megabyte	1,048,576 or 2^{20} bytes
Gigabit	2^{30} bits
Gigabyte	2^{30} bytes
Terabyte	2^{40} bytes
Petabyte	2^{50} bytes
Exabyte	2^{60} bytes

Data Compression

The digital data is compressed to reduce transmission times and file size. Data compression is the process of reducing the number of bits used to represent data. Data compression typically uses encoding techniques to compress the data. The compressed data will help us to save storage capacity, reduce costs for storage hardware, increase file transfer speed.

Compression uses some programs, which also uses algorithms and functions to find out the way to reduce the data size. Compression can be referred "zipping". The process of reconstructing files will be known as unzipping or extracting. The compressed files will contain .gz, or.tar.gz, .pkg, or .zip at the end of the files. Compression can be divided into two techniques: Lossless compression and Lossy compression.



Lossless Compression

As the name implies, lossless compression is the process of compressing the data without any loss of information or data. If we compressed the data with the help of lossless compression, then we can exactly recover the original data from the compressed data. That means all the information can be completely restored by lossless compression.

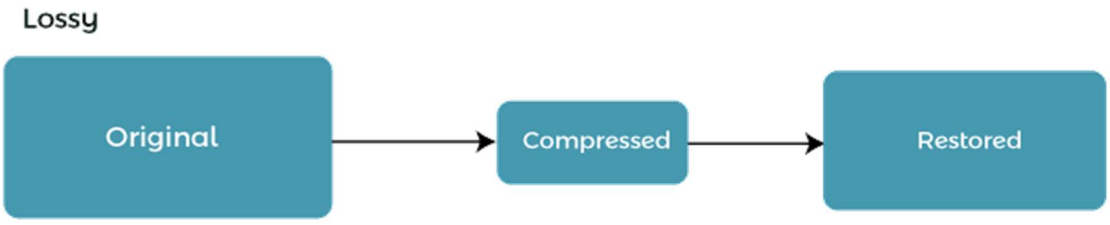
Many applications want to use data loss compression. For example, lossless compression can be used in the format of ZIP files and in the GNU tool gzip. The lossless data compression can also be used as a component within the technologies of lossy data compression. It is generally used for discrete data like word processing files, database records, some images, and information of the video.

According to this image, when we compress the original data using the lossless, we are able to restore all the original data.

Lossy Compression

Lossy compression is the process of compressing the data, but that data cannot be recovered 100% of original data. This compression is able to provide a high degree of compression, and the result of this compression will be in smaller compressed files. But in this process, some number of video frames, sound waves and original pixels are removed forever.

If the compression is greater, then the size of files will be smaller. Business data and text, which needs a full restoration, will never use lossy compression. Nobody likes to lose the information, but there are a lot of files that are very large, and we don't have enough space to maintain all of the original data or many times, we don't require all the original data in the first place. For example, videos, photos and audio recording files to capture the beauty of our world. In this case, we use lossy compression.



According to this image, when we compress the original data using the lossy, we are only able to restore some amount of data. We will not restore 100% of the original data.

Memory reference instruction

In computer systems, the concept of "memory reference instructions" is fundamental to understanding how a Central Processing Unit (CPU) interacts with memory to perform operations. These instructions provide a way for the CPU to read data from and write data to memory. Let me break it down in easy-to-understand terms:

1. **Memory**: Think of computer memory as a vast storage area where data is kept. It's like a huge bookshelf with many shelves, and each shelf can store different things (data).

2. **Central Processing Unit (CPU)**: The CPU is like the brain of the computer. It processes instructions, performs calculations, and controls all the operations.

3. **Memory Reference Instructions**: These are commands given to the CPU to interact with memory. They tell the CPU what to do with data in memory, like reading (fetching) or writing (storing) data.

Here are the two main types of memory reference instructions:

- **Load**: When the CPU receives a "load" instruction, it's like the CPU sending someone to the memory bookshelf to fetch a specific book. In computer terms, it's reading (loading) data from memory into the CPU so it can work with that data.

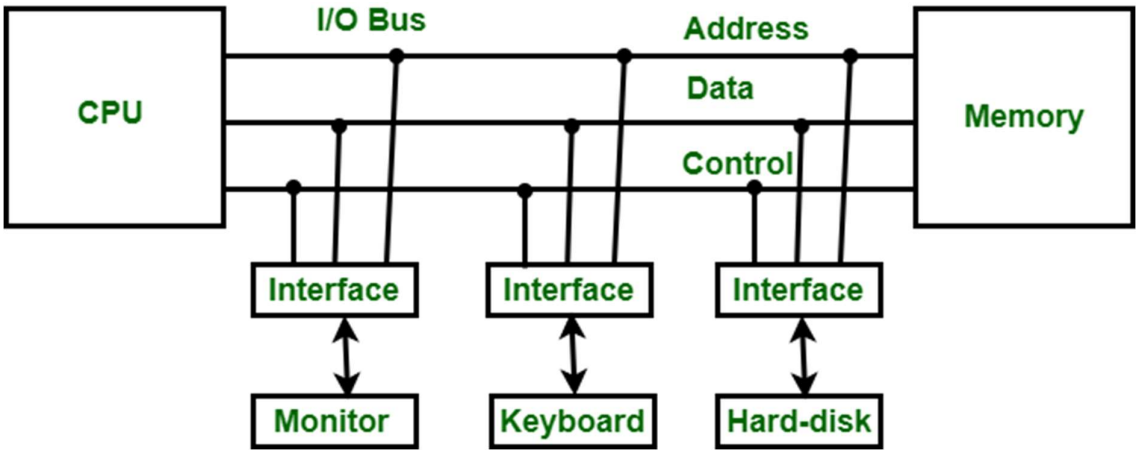
- **Store**: Conversely, when the CPU gets a "store" instruction, it's like telling someone to place a new book on the memory bookshelf. In computer terms, it's writing (storing) data into memory so that it can be used or saved for later.

So, memory reference instructions are like the communication between the CPU and the memory. They allow the CPU to access, manipulate, and store data, which is essential for running programs, performing calculations, and storing information on your computer.

In summary, memory reference instructions are the commands that the CPU uses to interact with computer memory. They enable the CPU to load data it needs for processing and to store results or other data back into memory. It's a crucial part of how a computer system functions and processes information.

Input-output

Input-Output Interface is used as an method which helps in transferring of information between the internal storage devices i.e. memory and the external peripheral device . A peripheral device is that which provide input and output for the computer, it is also called Input-Output devices. For Example: A keyboard and mouse provide Input to the computer are called input devices while a monitor and printer that provide output to the computer are called output devices. Just like the external hard-drives, there is also availability of some peripheral devices which are able to provide both input and output.



Input-Output Interface

In micro-computer base system, the only purpose of peripheral devices is just to provide **special communication links** for the interfacing them with the CPU. To resolve the differences between peripheral devices and CPU, there is a special need for communication links.
The major differences are as follows:

1. The nature of peripheral devices is electromagnetic and electro-mechanical. The nature of the CPU is electronic. There is a lot of difference in the mode of operation of both peripheral devices and CPU.
2. There is also a synchronization mechanism because the data transfer rate of peripheral devices are slow than CPU.
3. In peripheral devices, data code and formats are differ from the format in the CPU and memory.
4. The operating mode of peripheral devices are different and each may be controlled so as not to disturb the operation of other peripheral devices connected to CPU.

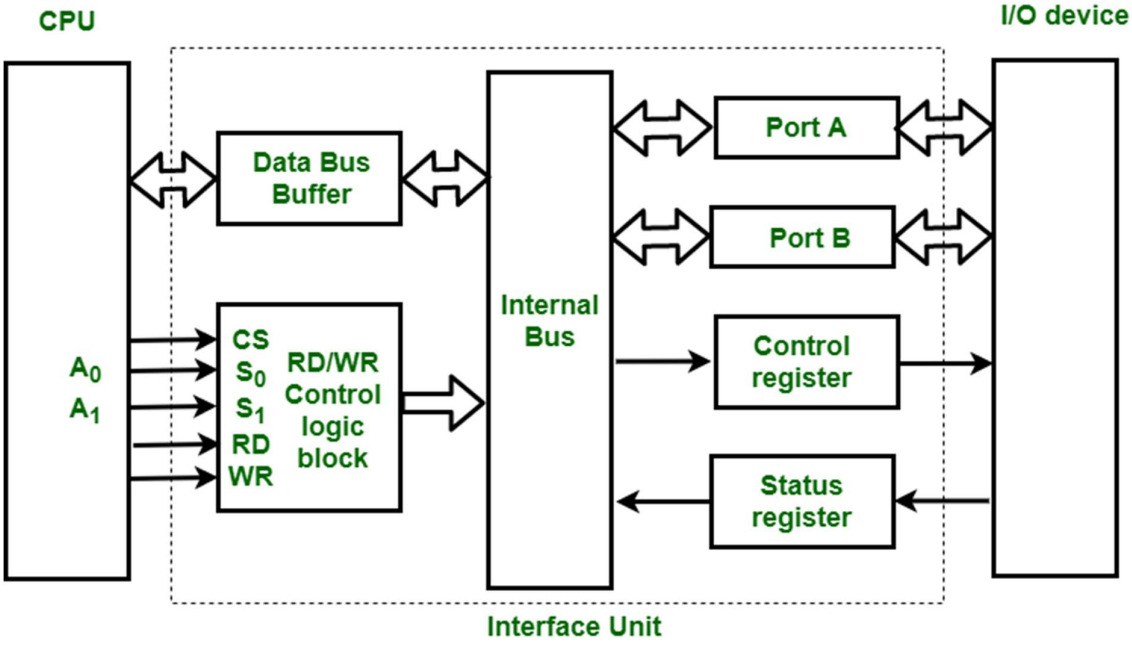
There is a special need of the additional hardware to resolve the differences between CPU and peripheral devices to supervise and synchronize all input and output devices.

Functions of Input-Output Interface:

1. It is used to synchronize the operating speed of CPU with respect to input-output devices.
2. It selects the input-output device which is appropriate for the interpretation of the input-output device.
3. It is capable of providing signals like control and timing signals.
4. In this data buffering can be possible through data bus.
5. There are various error detectors.
6. It converts serial data into parallel data and vice-versa.
7. It also convert digital data into analog signal and vice-versa.

The block diagram of an Input-Output Interface unit contain the following blocks :

1. Data Bus Buffer
2. Read/Write Control Logic
3. Port A, Port B register
4. Control and Status register



These are explained as following below.

Data Bus Buffer :
The bus buffer use bi-directional data bus to communicate with CPU. All control word data and status information between interface unit and CPU are transferred through data bus.
Port A and Port B :
Port A and Port B are used to transfer data between Input-Output device and Interface Unit. Each port consist of bi-directional data input buffer and bi-directional data output buffer. Interface unit connect directly with an input device and output disk or with device that require both input and output through Port A and Port B i.e. modem, external hard-drive, magnetic disk.
Control and Status Register :
CPU gives control information to control register on basis of control information. Interface unit control input and output operation between CPU and input-output device. Bits which are present in status register are used for checking of status conditions. Status register indicate status of data register, port A, port B and also record error that may be occur during transfer of data.
Read/Write Control Logic :
This block generates necessary control signals for overall device operations. All commands from CPU are accepted through this block. It also allow status of interface unit to be transferred onto data bus through this block accept CS, read and write control signal from system bus and S_0, S_1 from system address bus. Read and Write signal are used to define direction of data transfer over data bus.
Read Operation: CPU <---- I/O device
Write Operation: CPU ----> I/O device
The read signal direct data transfer from interface unit to CPU and write signal direct data transfer from CPU to interface unit through data bus.

Address bus is used to select to interface unit. Two least significant lines of address bus (A_0, A_1) are connected to select lines S_0, S_1. This two select input lines are used to select any one of four registers in interface unit. The selection of interface unit is according to the following criteria :

Read state :

Chip Select	Operation	Select lines	Selection of

CS	Read	Write	S_0	S_1	Interface unit
0	0	1	0	0	Port A
0	0	1	0	1	Port B
0	0	1	1	0	Control Register
0	0	1	1	1	Status Register

Write State :

Chip Select	Operation		Select lines		
CS	Read	Write	S_0	S_1	Selection of Interface unit
0	1	0	0	0	Port A
0	1	0	0	1	Port B
0	1	0	1	0	Control Register
0	1	0	1	1	Status Register

Example :

- If S_0, S_1 = 0 1, then Port B data register is selected for data transfer between CPU and I/O device.
- If S_0, S_1 = 1 0, then Control register is selected and store the control information send by the CPU.

Interrupts

The interrupt is a signal emitted by hardware or software when a process or an event needs immediate attention. It alerts the processor to a high-priority process requiring interruption of the current working process. In I/O devices one of the bus control lines is dedicated for this purpose and is called the *Interrupt Service Routine (ISR)*.

When a device raises an interrupt at let’s say process i, the processor first completes the execution of instruction i. Then it loads the Program Counter (PC) with the address of the first instruction of the ISR. Before loading the Program Counter with the address, the address of the interrupted instruction is moved to a temporary location. Therefore, after handling the interrupt the processor can continue with process i+1.

While the processor is handling the interrupts, it must inform the device that its request has been recognized so that it stops sending the interrupt request signal. Also, saving the registers so that the interrupted process can be restored in the future, increases the delay between the time an interrupt is received and the start of the execution of the ISR. This is called Interrupt Latency.

Another defination of interrupts :-

In CPU (Central Processing Unit) design, "interrupts" are a mechanism used to temporarily pause the CPU's current activities to handle higher-priority tasks or events. Think of interrupts as a way for the CPU to say, "Hold on, something important just came up, and we need to address it right away." Here's an easy-to-understand explanation of interrupts:

1. Unexpected Events:

- Imagine your CPU is like a chef cooking in a restaurant kitchen.

- While the chef is cooking, unexpected events can happen, like a fire alarm going off, a food order from a VIP customer, or a server bringing a customer complaint.

- In the CPU's world, these unexpected events are called "interrupts."

2. Temporary Pause:

- When an interrupt occurs, it's as if the chef temporarily stops cooking to address the issue.

- In CPU terms, the CPU stops its current task, saves its progress, and shifts its attention to the interrupting event.

3. Handling Priorities:

- Some interrupts are more urgent than others. For instance, a fire alarm is more critical than a customer inquiry.

- In CPU design, interrupts are prioritized. High-priority interrupts are handled before lower-priority ones.

4. Interrupt Service Routine (ISR):

- To address each type of interrupt, there is a specific set of instructions called an "Interrupt Service Routine" (ISR).

- The chef knows how to respond to different situations, like turning off stoves during a fire alarm (the "ISR" for that event).

5. Save and Resume:

- Just like the chef saves their progress and later resumes cooking, the CPU saves its current state (like the values in registers and the instruction it was executing).

- After handling the interrupt, the CPU returns to where it left off, as if nothing happened.

6. Use Cases:

- Interrupts are used in many scenarios: to handle errors, interact with external devices (like pressing a key on your keyboard), or perform multitasking in an operating system.

7. Efficient Handling:

- Efficient handling of interrupts is crucial. It ensures that the CPU can respond quickly to important events and smoothly continue regular tasks.

8. Real-Life Analogy:

- Think of interrupts like your smartphone receiving a phone call (an interrupt) while you're playing a game (your CPU's task). You pause the game, take the call (handle the interrupt), and then resume playing the game (continue with your previous task).

In summary, interrupts in CPU design are a way to manage and respond to unexpected events or high-priority tasks efficiently. They allow the CPU to pause its current activities, address the interrupting event, and then resume where it left off. It's like a chef in a restaurant kitchen temporarily stopping cooking to handle different urgent situations and then going back to cooking.

Addressing modes

Addressing modes in CPU design refer to the various methods by which a CPU can access data or operands in memory or registers. These modes determine how the CPU specifies the source or destination of data for an instruction. Here, we'll explain common addressing modes in an easy-to-understand manner:

1. **Immediate Addressing Mode:**

In this mode, the operand is part of the instruction itself. For example, if you have an instruction like "ADD R1, #5," the value 5 is the immediate operand. It's directly used by the instruction.

2. **Register Addressing Mode:**

Here, the operand is located in one of the CPU's registers. For instance, in the instruction "ADD R1, R2," both R1 and R2 are registers containing data to be operated on.

3. **Direct Addressing Mode:**

In this mode, the instruction contains the memory address of the operand. For example, "MOV R1, [0x1234]" instructs the CPU to move data from memory address 0x1234 into register R1.

4. **Indirect Addressing Mode:**

In this mode, the instruction contains a memory address pointing to the actual location of the operand. It's like having a pointer. For instance, "MOV R1, [R2]" means the data at the memory location pointed to by the value in register R2 will be moved into R1.

5. **Indexed Addressing Mode:**

This mode involves using a base address, an index, and an offset to calculate the effective address. It's often used in array operations. For example, "LOAD R1, [R2 + 5]" means that the data at the memory location calculated as R2 + 5 will be loaded into R1.

6. **Relative Addressing Mode:**

Here, the effective address is determined by adding an offset to the current program counter (PC) value. It's often used in branching or jumping instructions. For instance, "BRANCH +5" means jump forward five instructions from the current PC.

7. **Stack Addressing Mode:**

Stack-based CPUs use this mode where operands are implicitly accessed from the top of the stack. Operations like push and pop are commonly used. For example, "PUSH R1" puts the value in R1 onto the stack.

8. **Memory Indirect Addressing Mode:**

Similar to indirect addressing, but here, you have a memory location pointing to another memory location, which holds the actual operand. It's like having a pointer to a pointer.

Each addressing mode serves specific purposes and allows the CPU to work with data efficiently based on the type of operation being performed and the structure of the data. The CPU designer chooses which addressing modes to include to meet the requirements of the intended applications.

Data transfer and manupulation of data

Data transfer and manipulation are fundamental operations within a CPU (Central Processing Unit) that allow it to process information and execute instructions efficiently. Let's explore these concepts in the context of computer system organization:

1. **Data Transfer**:

Data transfer refers to the movement of data from one location to another within the CPU. This can involve transferring data between registers, memory, and I/O devices. Here are key aspects:

- **Registers**: CPUs have a set of high-speed registers where data is temporarily stored for processing. Data is transferred between these registers using data transfer instructions.

- **Memory**: Data is transferred between registers and memory. Load instructions move data from memory to registers, while store instructions move data from registers to memory.

- **I/O Devices**: CPUs communicate with external devices, such as keyboards, displays, and storage, by transferring data between registers and I/O buffers. This is typically achieved using input and output instructions.

2. **Data Manipulation**:

Data manipulation involves performing operations on data to transform or process it. The ALU (Arithmetic Logic Unit) in the CPU is responsible for executing arithmetic and logical operations. Here's how data manipulation works:

- **Arithmetic Operations**: The CPU can perform operations like addition, subtraction, multiplication, and division on data in registers.

For example, the instruction "ADD R1, R2, R3" adds the values in registers R2 and R3 and stores the result in R1.

- **Logical Operations**: Logical operations like AND, OR, and NOT are used for tasks such as bit manipulation and decision-making. For example, "AND R1, R2, R3" performs a bitwise AND between the values in registers R2 and R3 and stores the result in R1.

- **Shift and Rotate Operations**: Shifting and rotating bits within registers are common for tasks like data extraction or manipulation. "LSL R1, R2, 2" left-shifts the bits in R2 by two positions and stores the result in R1.

- **Comparison Operations**: CPUs can compare data in registers using instructions like "CMP" or "TEST." These operations set flags in the status register to indicate the result of the comparison, which is crucial for branching instructions.

- **Bitwise Operations**: Bitwise operations involve modifying individual bits in a data word. "BITWISE-AND R1, R2, R3" is an example of a bitwise AND operation.

- **Data Conversion**: Data can be converted from one format to another using various operations. For example, data can be converted from binary to decimal or vice versa.

3. **Data Flow**:

Understanding the flow of data through the CPU is vital. Instructions are fetched from memory, and their operands are read from registers or memory. After execution, results may be stored back in registers or memory. Data flow also includes considerations for pipelining and parallel processing.

In summary, data transfer and manipulation are core operations in CPU design. Efficiently moving data and performing calculations are essential for the CPU to execute instructions, process information, and perform a wide range of tasks in a computer system. These operations are finely orchestrated to ensure the CPU's proper functioning and to execute programs effectively.

Another explaination:-

Data Manipulation Instructions Data manipulation instructions perform operations on data and provide computational capabilities for the computer. The data manipulation instructions in a typical computer are usually divided into three basic types as follows.

1. Arithmetic instructions
2. Logical and bit manipulation instructions
3. Shift instructions

Let's discuss them one by one.

1. **Arithmetic instructions:** The four basic operations are addition, subtraction, multiplication, and division. Most computers provide instructions for all four operations. **Typical Arithmetic Instructions –**

Name	Mnemonic	Example	Explanation
Increment	INC	INC B	It will increment the register B by 1 B<-B+1

Decrement	**DEC**	**DEC B**	**It will decrement the register B by 1** **B<-B-1**
Add	**ADD**	**ADD B**	**It will add contents of register B to the contents of the accumulator** **and store the result in the accumulator** **AC<-AC+B**
Subtract	**SUB**	**SUB B**	**It will subtract the contents of register B from the contents of the** **accumulator and store the result in the accumulator** **AC<-AC-B**
Multiply	**MUL**	**MUL B**	**It will multiply the contents of register B with the contents of the** **accumulator and store the result in the accumulator** **AC<-AC*B**
Divide	**DIV**	**DIV B**	**It will divide the contents of register B with the contents of the** **accumulator and store the quotient in the accumulator** **AC<-AC/B**

Add with carry	**ADDC**	**ADDC B**	**It will add the contents of register B and the carry flag with the contents of the accumulator and store the result in the accumulator AC<-AC+B+Carry flag**
Subtract with borrow	**SUBB**	**SUBB B**	**It will subtract the contents of register B and the carry flag from the contents of the accumulator and store the result in the accumulator AC<-AC-B-Carry flag**
Negate(2's complement)	**NEG**	**NEG B**	**It will negate a value by finding 2's complement of its single operand. This means simply operand by -1. B<-B'+1**

1. **Logical and Bit Manipulation Instructions:** Logical instructions perform binary operations on strings of bits stored in registers. They are helpful for manipulating individual bits or a group of bits. **Typical Logical and Bit Manipulation Instructions –**

Name	Mnemonic	Example	Explanation
Clear	**CLR**	**CLR**	**It will set the accumulator to 0 AC<-0**
Complement	**COM**	**COM A**	**It will complement the accumulator AC<-(AC)'**

AND	**AND**	**AND B**	**It will AND the contents of register B with the contents of accumulator and store it in the accumulator AC<-AC AND B**
OR	**OR**	**OR B**	**It will OR the contents of register B with the contents of accumulator and store it in the accumulator AC<-AC OR B**
Exclusive-OR	**XOR**	**XOR B**	**It will XOR the contents of register B with the contents of the accumulator and store it in the accumulator AC<-AC XOR B**
Clear carry	**CLRC**	**CLRC**	**It will set the carry flag to 0 Carry flag<-0**
Set carry	**SETC**	**SETC**	**It will set the carry flag to 1 Carry flag<-1**
Complement carry	**COMC**	**COMC**	**It will complement the carry flag Carry flag<- (Carry flag)'**
Enable interrupt	**EI**	**EI**	**It will enable the interrupt**
Disable interrupt	**DI**	**DI**	**It will disable the interrupt**

1. **Shift Instructions:** Shifts are operations in which the bits of a word are moved to the left or right. Shift instructions may specify either

logical shifts, arithmetic shifts, or rotate-type operations. **Typical Shift Instructions –**

Name	Mnemonic
Logical shift right	SHR
Logical shift left	SHL
Arithmetic shift right	SHRA
Arithmetic shift left	SHLA
Rotate right	ROR
Rotate left	ROL
Rotate right through carry	RORC
Rotate left through carry	ROLC