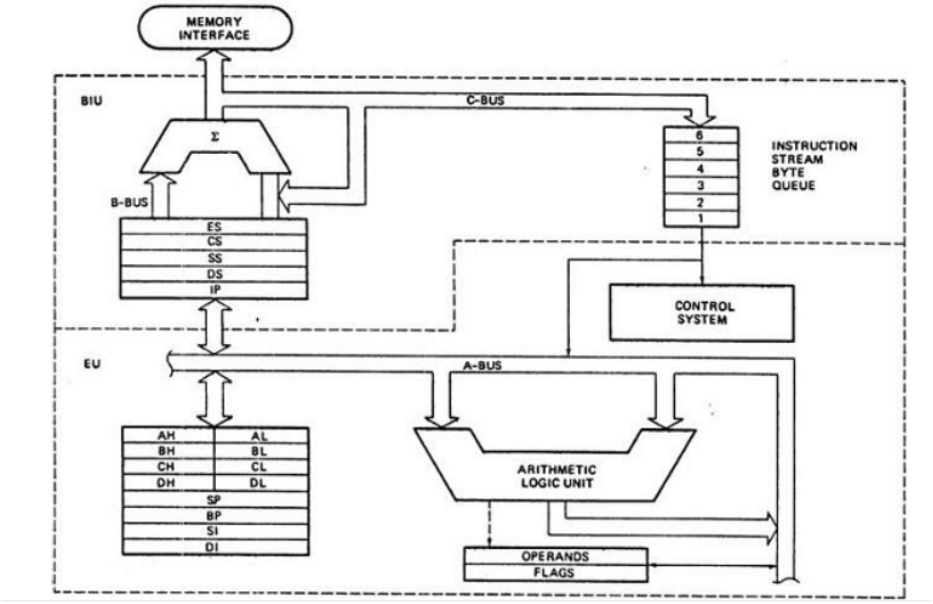
COA IMPORTANT TOPICS

1) 8086 Architecture

8086 Microprocessor is an enhanced version of 8085 Microprocessor that was designed by Intel in 1976. It is a 16-bit Microprocessor having 20 address lines and16 data lines that provides up to 1MB storage. It consists of powerful instruction set, which provides operations like multiplication and division easily.

It supports two modes of operation, i.e. Maximum mode and Minimum mode. Maximum mode is suitable for system having multiple processors and Minimum mode is suitable for system having a single processor.

8086 Microprocessor is divided into two functional units, i.e., EU (Execution Unit) and BIU (Bus Interface Unit).



EU (Execution Unit)

Execution unit gives instructions to BIU stating from where to fetch the data and then decode and execute those instructions. Its function is to control operations on data using the instruction decoder & ALU. EU has no direct connection with system buses as shown in the above figure, it performs operations over data through BIU.

Let us now discuss the functional parts of 8086 microprocessors.

1) ALU

It handles all arithmetic and logical operations, like +, −, ×, /, OR, AND, NOT operations.

Flag Register.It is a 16-bit register that behaves like a flip-flop, i.e. it changes its status according to the result stored in the accumulator. It has 9 flags and they are divided into 2 groups − Conditional Flags and Control Flags.

Conditional Flags

It represents the result of the last arithmetic or logical instruction executed. Following is the list of conditional flags −

Carry flag − This flag indicates an overflow condition for arithmetic operations.

Auxiliary flag − When an operation is performed at ALU, it results in a carry/barrow from lower nibble (i.e. D0 – D3) to upper nibble (i.e. D4 – D7), then this flag is set, i.e. carry given by D3 bit to D4 is AF flag. The processor uses this flag to perform binary to BCD conversion.

Parity flag − This flag is used to indicate the parity of the result, i.e. when the lower order 8-bits of the result contains even number of 1's, then the Parity Flag is set. For odd number of 1's, the Parity Flag is reset.
Zero flag − This flag is set to 1 when the result of arithmetic or logical operation is zero else it is set to 0.
Sign flag − This flag holds the sign of the result, i.e. when the result of the operation is negative, then the sign flag is set to 1 else set to 0.
Overflow flag − This flag represents the result when the system capacity is exceeded.

Control Flags
Control flags controls the operations of the execution unit. Following is the list of control flags −
Trap flag − It is used for single step control and allows the user to execute one instruction at a time for debugging. If it is set, then the program can be run in a single step mode.

Interrupt flag − It is an interrupt enable/disable flag, i.e. used to allow/prohibit the interruption of a program. It is set to 1 for interrupt enabled condition and set to 0 for interrupt disabled condition.
Direction flag − It is used in string operation. As the name suggests when it is set then string bytes are accessed from the higher memory address to the lower memory address and vice-a-versa.
General purpose register
There are 8 general purpose registers, i.e., AH, AL, BH, BL, CH, CL, DH, and DL. These registers can be used individually to store 8-bit data and can be used in pairs to store 16bit data. The valid register pairs are AH and AL, BH and BL, CH and CL, and DH and DL. It is referred to the AX, BX, CX, and DX respectively.
AX register − It is also known as accumulator register. It is used to store operands for arithmetic operations.
BX register − It is used as a base register. It is used to store the starting base address of the memory area within the data segment.
CX register − It is referred to as counter. It is used in loop instruction to store the loop counter.
DX register − This register is used to hold I/O port address for I/O instruction.

Stack pointer register
It is a 16-bit register, which holds the address from the start of the segment to the memory location, where a word was most recently stored on the stack.

BIU (Bus Interface Unit)
BIU takes care of all data and addresses transfers on the buses for the EU like sending addresses, fetching instructions from the memory, reading data from the ports and the memory as well as writing data to the ports and the memory. EU has no direction connection with System Buses so this is possible with the BIU. EU and BIU are connected with the Internal Bus.

It has the following functional parts −
Instruction queue − BIU contains the instruction queue. BIU gets upto 6 bytes of next instructions and stores them in the instruction queue. When EU executes instructions and is ready for its next instruction, then it simply reads the instruction from this instruction queue resulting in increased execution speed.Fetching the next instruction while the current instruction executes is called pipelining.

Segment register − BIU has 4 segment buses, i.e. CS, DS, SS& ES. It holds the addresses of instructions and data in memory, which are used by the processor to access memory locations. It also contains 1 pointer register IP, which holds the address of the next instruction to executed by the EU.
CS − It stands for Code Segment. It is used for addressing a memory location in the code segment of the memory, where the executable program is stored.
DS − It stands for Data Segment. It consists of data used by the program andis accessed in the data segment by an offset address or the content of other register that holds the offset address.
SS − It stands for Stack Segment. It handles memory to store data and addresses during execution.
ES − It stands for Extra Segment. ES is additional data segment, which is used by the string to hold the extra destination data.
Instruction pointer − It is a 16-bit register used to hold the address of the next instruction to be executed.

2) 8086 instructions

Data Transfer Instructions

These instructions are used to transfer the data from the source operand to the destination operand. Following are the list of instructions under this group −

Instruction to transfer a word

MOV − Used to copy the byte or word from the provided source to the provided destination.
PPUSH − Used to put a word at the top of the stack.
POP − Used to get a word from the top of the stack to the provided location.
PUSHA − Used to put all the registers into the stack.
POPA − Used to get words from the stack to all registers.
XCHG − Used to exchange the data from two locations.
XLAT − Used to translate a byte in AL using a table in the memory.

Instructions for input and output port transfer

IN − Used to read a byte or word from the provided port to the accumulator.
OUT − Used to send out a byte or word from the accumulator to the provided port.

Instructions to transfer the address

LEA − Used to load the address of operand into the provided register.
LDS − Used to load DS register and other provided register from the memory
LES − Used to load ES register and other provided register from the memory.

Instructions to transfer flag registers

LAHF − Used to load AH with the low byte of the flag register.
SAHF − Used to store AH register to low byte of the flag register.
PUSHF − Used to copy the flag register at the top of the stack.
POPF − Used to copy a word at the top of the stack to the flag register.

Arithmetic Instructions

These instructions are used to perform arithmetic operations like addition, subtraction, multiplication, division, etc.
Following is the list of instructions under this group −

Instructions to perform addition

ADD − Used to add the provided byte to byte/word to word.

ADC − Used to add with carry.
INC − Used to increment the provided byte/word by 1.
AAA − Used to adjust ASCII after addition.
DAA − Used to adjust the decimal after the addition/subtraction operation.

Instructions to perform subtraction

SUB − Used to subtract the byte from byte/word from word.
SBB − Used to perform subtraction with borrow.
DEC − Used to decrement the provided byte/word by 1.
NPG − Used to negate each bit of the provided byte/word and add 1/2's complement.
CMP − Used to compare 2 provided byte/word.
AAS − Used to adjust ASCII codes after subtraction.
DAS − Used to adjust decimal after subtraction.

Instruction to perform multiplication

MUL − Used to multiply unsigned byte by byte/word by word.
IMUL − Used to multiply signed byte by byte/word by word.
AAM − Used to adjust ASCII codes after multiplication.

Instructions to perform division

DIV − Used to divide the unsigned word by byte or unsigned double word by word.
IDIV − Used to divide the signed word by byte or signed double word by word.
AAD − Used to adjust ASCII codes after division.
CBW − Used to fill the upper byte of the word with the copies of sign bit of the lower byte.
CWD − Used to fill the upper word of the double word with the sign bit of the lower word.

Bit Manipulation Instructions

These instructions are used to perform operations where data bits are involved, i.e. operations like logical, shift, etc.
Following is the list of instructions under this group −

Instructions to perform logical operation

NOT − Used to invert each bit of a byte or word.
AND − Used for adding each bit in a byte/word with the corresponding bit in another byte/word.
OR − Used to multiply each bit in a byte/word with the corresponding bit in another byte/word.
XOR − Used to perform Exclusive-OR operation over each bit in a byte/word with the corresponding bit in another byte/word.
TEST − Used to add operands to update flags, without affecting operands.

Instructions to perform shift operations

SHL/SAL − Used to shift bits of a byte/word towards left and put zero(S) in LSBs.
SHR − Used to shift bits of a byte/word towards the right and put zero(S) in MSBs.
SAR − Used to shift bits of a byte/word towards the right and copy the old MSB into the new MSB.

Instructions to perform rotate operations

ROL − Used to rotate bits of byte/word towards the left, i.e. MSB to LSB and to

Carry Flag [CF].
ROR − Used to rotate bits of byte/word towards the right, i.e. LSB to MSB and to Carry Flag [CF].
RCR − Used to rotate bits of byte/word towards the right, i.e. LSB to CF and CF to MSB.
RCL − Used to rotate bits of byte/word towards the left, i.e. MSB to CF and CF to LSB.

String Instructions

String is a group of bytes/words and their memory is always allocated in a sequential order.
Following is the list of instructions under this group −
REP − Used to repeat the given instruction till CX ≠ 0.
REPE/REPZ − Used to repeat the given instruction until CX = 0 or zero flag ZF = 1.
REPNE/REPNZ − Used to repeat the given instruction until CX = 0 or zero flag ZF = 1.
MOVS/MOVSB/MOVSW − Used to move the byte/word from one string to another.
COMS/COMPSB/COMPSW − Used to compare two string bytes/words.
INS/INSB/INSW − Used as an input string/byte/word from the I/O port to the provided memory location.
OUTS/OUTSB/OUTSW − Used as an output string/byte/word from the provided memory location to the I/O port.
SCAS/SCASB/SCASW − Used to scan a string and compare its byte with a byte in AL or string word with a word in AX.
LODS/LODSB/LODSW − Used to store the string byte into AL or string word into AX.

Program Execution Transfer Instructions (Branch and Loop Instructions)

These instructions are used to transfer/branch the instructions during an execution. It includes the following instructions −

Instructions to transfer the instruction during an execution without any condition-

CALL − Used to call a procedure and save their return address to the stack.
RET − Used to return from the procedure to the main program.
JMP − Used to jump to the provided address to proceed to the next instruction.

Instructions to transfer the instruction during an execution with some conditions-

JA/JNBE − Used to jump if above/not below/equal instruction satisfies.
JAE/JNB − Used to jump if above/not below instruction satisfies.
JBE/JNA − Used to jump if below/equal/ not above instruction satisfies.
JC − Used to jump if carry flag CF = 1
JE/JZ − Used to jump if equal/zero flag ZF = 1
JG/JNLE − Used to jump if greater/not less than/equal instruction satisfies.
JGE/JNL − Used to jump if greater than/equal/not less than instruction satisfies.
JL/JNGE − Used to jump if less than/not greater than/equal instruction satisfies.
JLE/JNG − Used to jump if less than/equal/if not greater than instruction satisfies.
JNC − Used to jump if no carry flag (CF = 0)
JNE/JNZ − Used to jump if not equal/zero flag ZF = 0

JNO − Used to jump if no overflow flag OF = 0
JNP/JPO − Used to jump if not parity/parity odd PF = 0
JNS − Used to jump if not sign SF = 0
JO − Used to jump if overflow flag OF = 1
JP/JPE − Used to jump if parity/parity even PF = 1
JS − Used to jump if sign flag SF = 1

Processor Control Instructions

These instructions are used to control the processor action by setting/resetting the flag values.
Following are the instructions under this group −
STC − Used to set carry flag CF to 1
CLC − Used to clear/reset carry flag CF to 0
CMC − Used to put complement at the state of carry flag CF.
STD − Used to set the direction flag DF to 1
CLD − Used to clear/reset the direction flag DF to 0
STI − Used to set the interrupt enable flag to 1, i.e., enable INTR input.
CLI − Used to clear the interrupt enable flag to 0, i.e., disable INTR input.

Iteration Control Instructions

These instructions are used to execute the given instructions for number of times.
Following is the list of instructions under this group −
LOOP − Used to loop a group of instructions until the condition satisfies, i.e., CX = 0
LOOPE/LOOPZ − Used to loop a group of instructions till it satisfies ZF = 1 & CX = 0
LOOPNE/LOOPNZ − Used to loop a group of instructions till it satisfies ZF = 0 & CX = 0
JCXZ − Used to jump to the provided address if CX = 0

Interrupt Instructions

These instructions are used to call the interrupt during program execution.
INT − Used to interrupt the program during execution and calling service specified.
INTO − Used to interrupt the program during execution if OF = 1
IRET − Used to return from interrupt service to the main program

3) 8086 addressing modes

The 8086 microprocessors have 8 addressing modes.Two addressing modes have been provided for instructions which operate on register or immediate data.

1. Register Addressing

In this, the operands / values are stored within any of the internal registers itself. So, the processing on these operands is done either in those registers or by shifting their values to some other registers. The operand is placed in one of the 16-bit or 8-bit general purpose registers.
Example
● MOV AX, CX
● ADD AL, BL
● ADD CX, DX

2. Immediate Addressing

In this addressing mode, the operands are specified within the instructions which means is that the instruction will either contain the values itself or will contain the operands whose values are required.

Example

- MOV AL, 35H
- MOV BX, 0301H
- MOV [0401], 3598H

3. Direct Addressing

In direct addressing mode, the operands offset is given in the instruction as an 8-bit or 16-bit displacement element.

Example

- ADD AL, [0301]

The instruction adds the content of the offset address 0301 to AL. the operand is placed at the given offset (0301) within the data segment DS.

4. Register Indirect Addressing

The operand's offset is placed in any one of the registers BX, BP, SI or DI as specified in the instruction.

Example

- MOV AX, [BX]

It moves the contents of memory locations addressed by the register BX to the register AX.

5. Based Addressing

The operand's offset is the sum of an 8-bit or 16-bit displacement and the contents of the base register BX or BP. BX is used as base register for data segment, and the BP is used as a base register for stack segment.Effective address (Offset) = [BX + 8-bit or 16-bit displacement].

Example

- MOV AL, [BX+05]; an example of 8-bit displacement.
- MOV AL, [BX + 1346H]; example of 16-bit displacement.

6. Indexed Addressing

The offset of an operand is the sum of the content of an index register SI or DI and an 8-bit or 16-bit displacement. Offset (Effective Address) = [SI or DI + 8-bit or 16-bit displacement]

Example

- MOV AX, [SI + 05]; 8-bit displacement.
- MOV AX, [SI + 1528H]; 16-bit displacement.

7. Base Indexed Addressing

The offset of operand is the sum of the content of a base register BX or BP and an index register SI or DI. Effective Address = [BX or BP] + [SI or DI] Here, BX is used for data segment, and BP is used for stack segment.

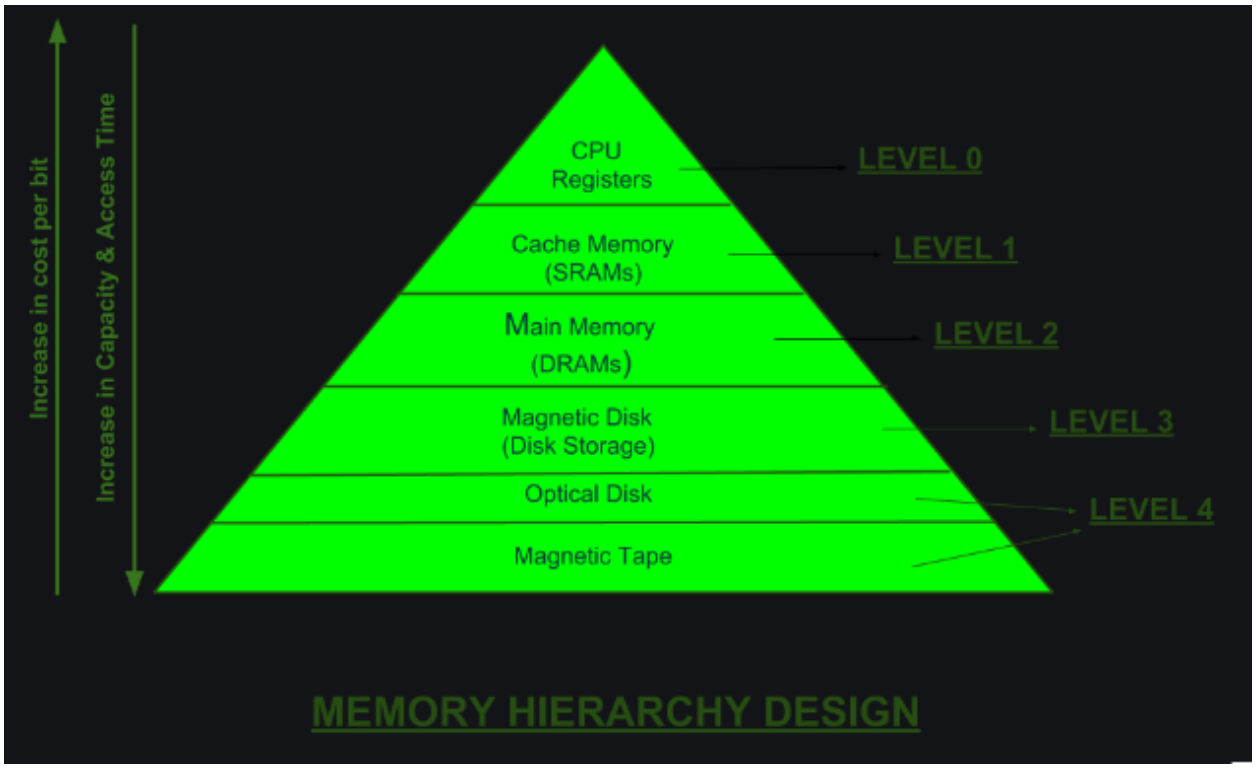
Example

- ADD AX, [BX + SI]
- MOV CX, [BX + SI]

8. Base Indexed with displacement Addressing
In this mode of addressing, the operand's offset is given by:
Effective Address (Offset) = [BX or BP] + [SI or DI] + 8-bit or 16-bit displacement
Example
● MOV AX, [BX + SI + 05]; 8-bit displacement
● MOV AX, [BX + SI + 1235H]; 16-bit displacement
Other Addressing modes:

● Program Memory Addressing Mode: These types of addressing modes are used in branch instructions like JMP or CALL instruction.
Example: JMP 0006H
● Stack Memory Addressing Mode: The stack memory addressing mode is used whenever you perform a push or pop operation.
Example: PUSH BX

4) Memory hierarchy
In the Computer System Design, Memory Hierarchy is an enhancement to organize the memory such that it can minimize the access time. The Memory Hierarchy was developed based on a program behavior known as locality of references.The figure below clearly demonstrates the different levels of memory hierarchy :



This Memory Hierarchy Design is divided into 2 main types:

External Memory or Secondary Memory – Comprising of Magnetic Disk, Optical Disk, Magnetic Tape i.e. peripheral storage devices which are accessible by the processor via I/O Module.
Internal Memory or Primary Memory – Comprising of Main Memory, Cache Memory & CPU registers. This is directly accessible by the processor.

There are typically four levels of memory in a memory hierarchy:
Registers: Registers are small, high-speed memory units located in the CPU. They are used to store the most frequently used data and instructions. Registers have the fastest access time and the smallest storage capacity, typically ranging from 16 to 64 bits.
Cache Memory: Cache memory is a small, fast memory unit located close to the CPU. It stores frequently used data and instructions that have been recently accessed from the main memory. Cache memory is designed to minimize the time it takes to access data by providing the CPU with quick access to frequently used data.
Main Memory: Main memory, also known as RAM (Random Access Memory), is the primary memory of a computer system. It has a larger storage capacity than cache memory, but it is slower. Main memory is used to store data and instructions that are currently in use by the CPU.

Types of Main memory:
Static RAM: It stores the binary information in flip flops and information remains valid until power is supplied. It has faster access time and is used in implementing cache memory.
Dynamic RAM – It stores the binary information as a charge on the capacitor. It requires refreshing circuitry to maintain the charge on the capacitors after few milliseconds. It contains more memory cells per unit area as compared to SRAM.

Secondary Storage: Secondary storage, such as hard disk drives (HDD) and solid-state drives (SSD), is a non-volatile memory unit that has a larger storage capacity than main memory. It is used to store data and instructions that are not currently in use by the CPU. Secondary storage has the slowest access time and is typically the least expensive type of memory in the memory hierarchy.

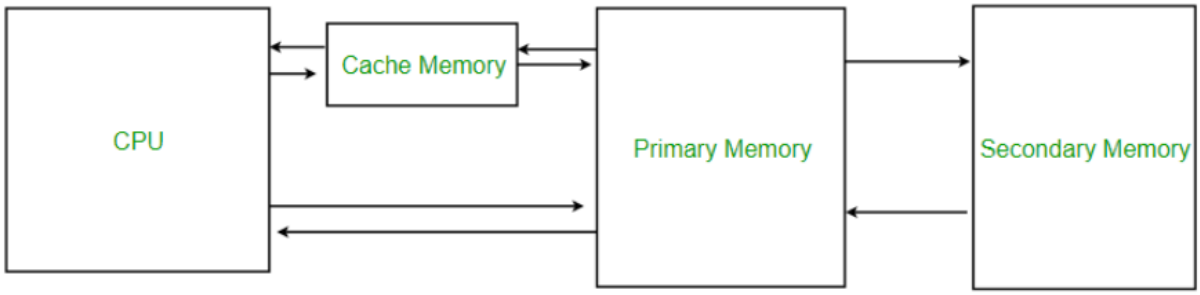
We can infer the following characteristics of Memory Hierarchy Design from above figure:
Capacity: It is the global volume of information the memory can store. As we move from top to bottom in the Hierarchy, the capacity increases.
Access Time: It is the time interval between the read/write request and the availability of the data. As we move from top to bottom in the Hierarchy, the access time increases.
Performance: Earlier when the computer system was designed without Memory Hierarchy design, the speed gap increases between the CPU registers and Main Memory due to large difference in access time. This results in lower performance of the system and thus, enhancement was required. This enhancement was made in the form of Memory Hierarchy Design because of which the performance of the system increases. One of the most significant ways to increase system performance is minimizing how far down the memory hierarchy one has to go to manipulate data.
Cost per bit: As we move from bottom to top in the Hierarchy, the cost per bit increases i.e. Internal Memory is costlier than External Memory.

5) Cache mapping techniques

Cache Memory is a special very high-speed memory. The cache is a smaller and faster memory that stores copies of the data from frequently used main memory locations. There are various different independent caches in a CPU, which store instructions and data. The most important use of cache memory is that it is used to reduce the average time to access data from the main memory.

Characteristics of Cache Memory

1) Cache memory is an extremely fast memory type that acts as a buffer between RAM and the CPU.
2) Cache Memory holds frequently requested data and instructions so that they are immediately available to the CPU when needed.
3) Cache memory is costlier than main memory or disk memory but more economical than CPU registers.
4) Cache Memory is used to speed up and synchronize with a high-speed CPU.



Levels of Memory

Level 1 or Register: It is a type of memory in which data is stored and accepted that are immediately stored in the CPU. The most commonly used register is Accumulator, Program counter, Address Register, etc.

Level 2 or Cache memory: It is the fastest memory that has faster access time where data is temporarily stored for faster access.

Level 3 or Main Memory: It is the memory on which the computer works currently. It is small in size and once power is off data no longer stays in this memory.

Level 4 or Secondary Memory: It is external memory that is not as fast as the main memory but data stays permanently in this memory.

Cache Performance

When the processor needs to read or write a location in the main memory, it first checks for a corresponding entry in the cache.

1) If the processor finds that the memory location is in the cache, a Cache Hit has occurred and data is read from the cache.
2) If the processor does not find the memory location in the cache, a cache miss has occurred. For a cache miss, the cache allocates a new entry and copies in data from the main memory, then the request is fulfilled from the contents of the cache.

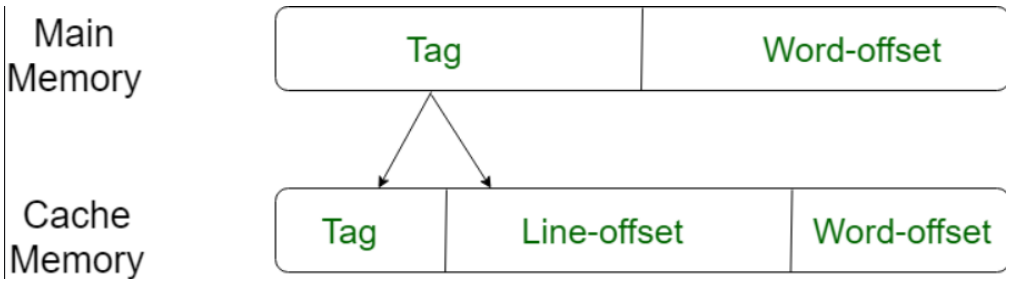
Cache Mapping

There are three different types of mapping used for the purpose of cache memory which is as follows:

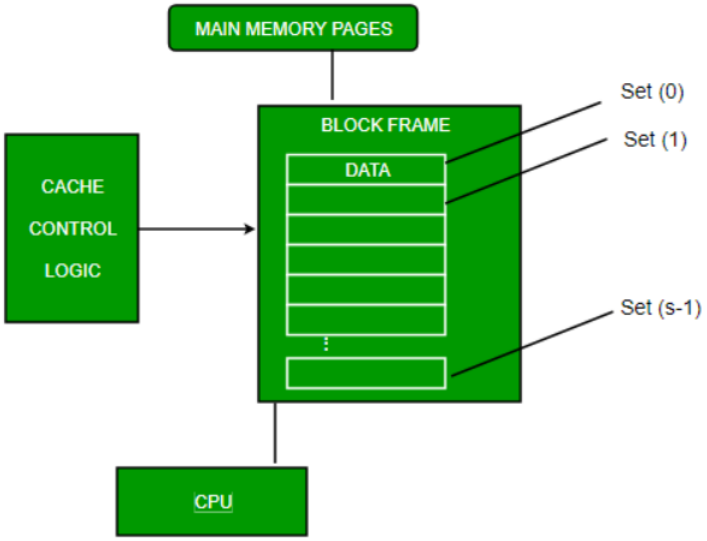
Direct Mapping
Associative Mapping
Set-Associative Mapping

1. Direct Mapping

The simplest technique, known as direct mapping, maps each block of main memory into only one possible cache line. or In Direct mapping, assign each memory block to a specific line in the cache. If a line is previously taken up by a memory block when a new block needs to be loaded, the old block is trashed. An address space is split into two parts index field and a tag field. The cache is used to store the tag field whereas the rest is stored in the main memory. Direct mapping`s performance is directly proportional to the Hit ratio.



For purposes of cache access, each main memory address can be viewed as consisting of three fields. The least significant w bits identify a unique word or byte within a block of main memory. In most contemporary machines, the address is at the byte level. The remaining s bits specify one of the 2s blocks of main memory. The cache logic interprets these s bits as a tag of s-r bits (the most significant portion) and a line field of r bits. This latter field identifies one of the m=2r lines of the cache. Line offset is index bits in the direct mapping.

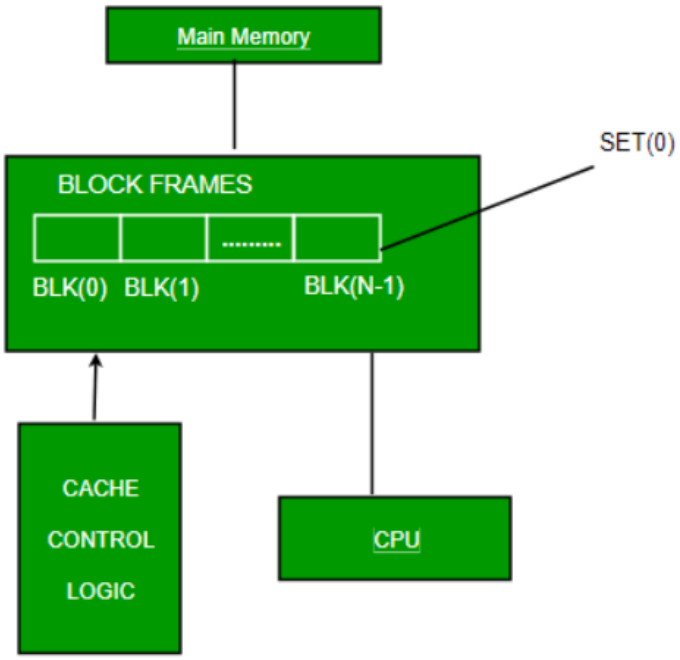


Direct mapping structure

2. Associative Mapping

In this type of mapping, associative memory is used to store the content and addresses of the memory word. Any block can go into any line of the cache. This means that the word id bits are used to identify which word in the block is needed, but the tag becomes

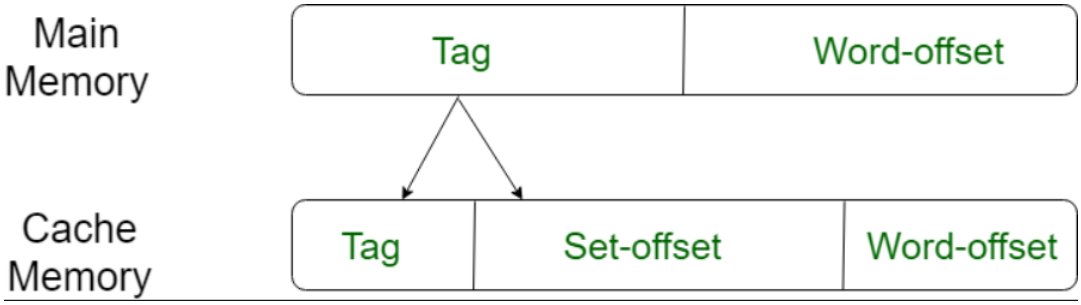
all of the remaining bits. This enables the placement of any word at any place in the cache memory. It is considered to be the fastest and most flexible mapping form. In associative mapping, the index bits are zero.



Associative mapping structure

3. Set-Associative Mapping

This form of mapping is an enhanced form of direct mapping where the drawbacks of direct mapping are removed. Set associative addresses the problem of possible thrashing in the direct mapping method. It does this by saying that instead of having exactly one line that a block can map to in the cache, we will group a few lines together creating a set. Then a block in memory can map to any one of the lines of a specific set. Set-associative mapping allows each word that is present in the cache can have two or more words in the main memory for the same index address. Set associative cache mapping combines the best of direct and associative cache mapping techniques. In set associative mapping the index bits are given by the set offset bits. In this case, the cache consists of a number of sets, each of which consists of a number of lines.



Application of Cache Memory

Here are some of the applications of Cache Memory.

1) **Primary Cache:** A primary cache is always located on the processor chip. This cache is small and its access time is comparable to that of processor registers.

2) **Secondary Cache:** Secondary cache is placed between the primary cache and the rest of the memory. It is referred to as the level 2 (L2) cache. Often, the Level 2 cache is also housed on the processor chip.
3) **Spatial Locality of Reference:** Spatial Locality of Reference says that there is a chance that the element will be present in close proximity to the reference point and next time if again searched then more close proximity to the point of reference.
4) **Temporal Locality of Reference:** Temporal Locality of Reference uses the Least recently used algorithm will be used. Whenever there is page fault occurs within a word will not only load the word in the main memory but the complete page fault will be loaded because the spatial locality of reference rule says that if you are referring to any word next word will be referred to in its register that's why we load complete page table so the complete block will be loaded.

Advantages of Cache Memory

1) Cache Memory is faster in comparison to main memory and secondary memory.
2) Programs stored by Cache Memory can be executed in less time.
3) The data access time of Cache Memory is less than that of the main memory.
4) Cache Memory stored data and instructions that are regularly used by the CPU, therefore it increases the performance of the CPU.

Disadvantages of Cache Memory

1) Cache Memory is costlier than primary memory and secondary memory.
2) Data is stored on a temporary basis in Cache Memory.
3) Whenever the system is turned off, data and instructions stored in cache memory get destroyed.
4) The high cost of cache memory increases the price of the Computer System.

6) DMA

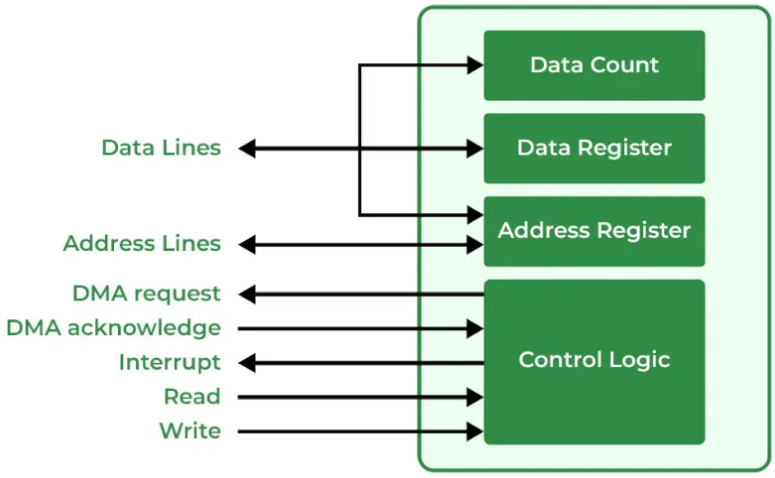
DMA Controller is a hardware device that allows I/O devices to directly access memory with less participation of the processor. DMA controller needs the same old circuits of an interface to communicate with the CPU and Input/Output devices.

What is a DMA Controller?

Direct Memory Access uses hardware for accessing the memory, that hardware is called a DMA Controller. It has the work of transferring the data between Input Output devices and main memory with very less interaction with the processor. The direct Memory Access Controller is a control unit, which has the work of transferring data.

DMA Controller Diagram in Computer Architecture

DMA Controller is a type of control unit that works as an interface for the data bus and the I/O Devices. As mentioned, DMA Controller has the work of transferring the data without the intervention of the processors, processors can control the data transfer. DMA Controller also contains an address unit, which generates the address and selects an I/O device for the transfer of data. Here we are showing the block diagram of the DMA Controller.



Types of Direct Memory Access (DMA)

There are four popular types of DMA.

1) Single-Ended DMA
2) Dual-Ended DMA
3) Arbitrated-Ended DMA
4) Interleaved DMA

Single-Ended DMA: Single-Ended DMA Controllers operate by reading and writing from a single memory address. They are the simplest DMA.

Dual-Ended DMA: Dual-Ended DMA controllers can read and write from two memory addresses. Dual-ended DMA is more advanced than single-ended DMA.

Arbitrated-Ended DMA: Arbitrated-Ended DMA works by reading and writing to several memory addresses. It is more advanced than Dual-Ended DMA.

Interleaved DMA: Interleaved DMA are those DMA that read from one memory address and write from another memory address.

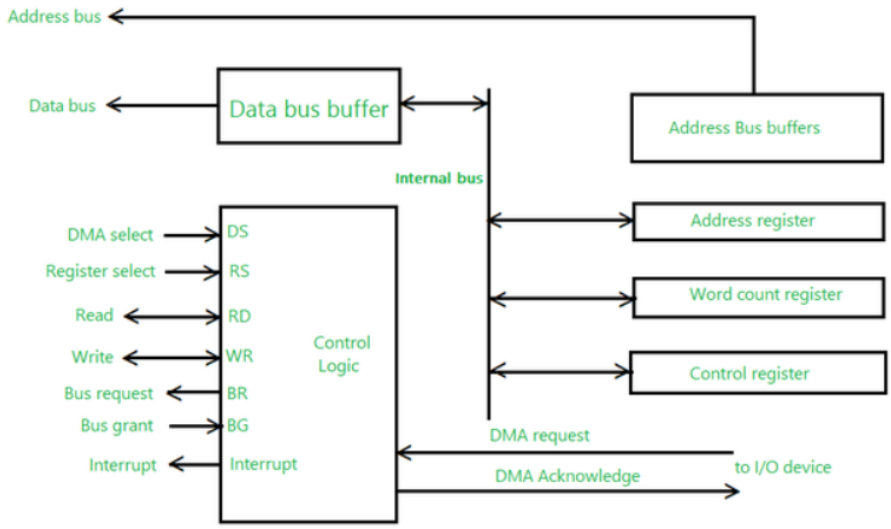
Working of DMA Controller

The DMA controller registers have three registers as follows.

1) **Address register** – It contains the address to specify the desired location in memory.
2) **Word count register** – It contains the number of words to be transferred.
3) **Control register** – It specifies the transfer mode.

Note: All registers in the DMA appear to the CPU as I/O interface registers. Therefore, the CPU can both read and write into the DMA registers under program control via the data bus.

The figure below shows the block diagram of the DMA controller. The unit communicates with the CPU through the data bus and control lines. Through the use of the address bus and allowing the DMA and RS register to select inputs, the register within the DMA is chosen by the CPU. RD and WR are two-way inputs. When BG (bus grant) input is 0, the CPU can communicate with DMA registers. When BG (bus grant) input is 1, the CPU has relinquished the buses and DMA can communicate directly with the memory.



Explanation: The CPU initializes the DMA by sending the given information through the data bus. The starting address of the memory block where the data is available (to read) or where data are to be stored (to write).It also sends word count which is the number of words in the memory block to be read or written.Control to define the mode of transfer such as read or write.A control to begin the DMA transfer

Advantages of DMA Controller

1) Data Memory Access speeds up memory operations and data transfer.
2) CPU is not involved while transferring data.
3) DMA requires very few clock cycles while transferring data.
4) DMA distributes workload very appropriately.
5) DMA helps the CPU in decreasing its load.

Disadvantages of DMA Controller

1) Direct Memory Access is a costly operation because of additional operations.
2) DMA suffers from Cache-Coherence Problems.
3) DMA Controller increases the overall cost of the system.
4) DMA Controller increases the complexity of the software.

7) Pipeline hazards

1. Pipeline hazards are situations that prevent the next instruction in the instruction stream from executing during its designated clock cycles.
2. Any condition that causes a stall in the pipeline operations can be called a hazard.
3. There are primarily three types of hazards:
i. Data Hazards
ii. Control Hazards or instruction Hazards
iii. Structural Hazards.

1. Data Hazards

A data hazard is any condition in which either the source or the destination operands of an instruction are not available at the time expected in the pipeline.
As a result of which some operation has to be delayed and the pipeline stalls. Whenever there are two instructions one of which depends on the data obtained from the other.
A=3+A
B=A*4

For the above sequence, the second instruction needs the value of ‘A’ computed in the first instruction. Thus the second instruction is said to depend on the first.
If the execution is done in a pipelined processor, it is highly likely that the interleaving of these two instructions can lead to incorrect results due to data dependency between the instructions. Thus the pipeline needs to be stalled as and when necessary to avoid errors.

2. Structural Hazards
This situation arises mainly when two instructions require a given hardware resource at the same time and hence for one of the instructions the pipeline needs to be stalled.
The most common case is when memory is accessed at the same time by two instructions. One instruction may need to access the memory as part of the Execute or Write back phase while other instruction is being fetched.
In this case if both the instructions and data reside in the same memory. Both the instructions can’t proceed together and one of them needs to be stalled till the other is done with the memory access part.
Thus in general sufficient hardware resources are needed for avoiding structural hazards.

3. Branch Hazards
The instruction fetch unit of the CPU is responsible for providing a stream of instructions to the execution unit. The instructions fetched by the fetch unit are in consecutive memory locations and they are executed.
However the problem arises when one of the instructions is a branching instruction to some other memory location.
Thus all the instruction fetched in the pipeline from consecutive memory locations are invalid now and need to removed(also called flushing of the pipeline).This induces a stall till new instructions are again fetched from the memory address specified in the branch instruction.
Thus the time lost as a result of this is called a branch penalty.
Often dedicated hardware is incorporated in the fetch unit to identify branch instructions and compute branch addresses as soon as possible and reducing the resulting delay as a result.

8) Flip Flops

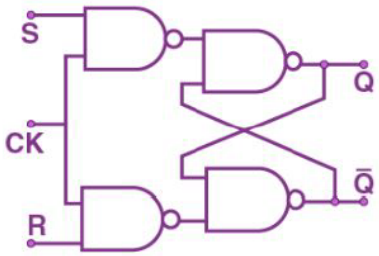
● A Flip Flop is an electronic circuit that is capable of storing one bit of information.
● It is a memory element that is capable of storing one bit of information.
● It has two stable states either 0 or 1.

Flip flops are of different types depending on how their inputs and clock pulses cause transition between two states.
There are 4 basic types of flip flops-
1. SR Flip Flop
2. JK Flip Flop
3. D Flip Flop
4. T Flip Flop

SR Flip flop

● SR flip flop is the simplest type of flip flops.
● It stands for Set Reset flip flop.
● It is a clocked flip flop.
● Following are the two methods for constructing a SR flip flop-
1. By using NOR latch
2. By using NAND latch

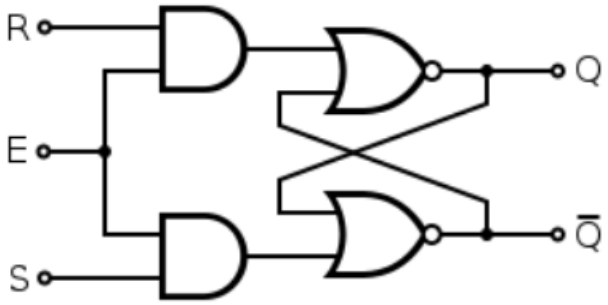
1. Construction of SR Flip Flop By Using NAND Latch-



Truth Table

S	R	Q_N	Q_{N+1}
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	-
1	1	1	-

2. Construction of SR Flip Flop By Using NOR Latch-

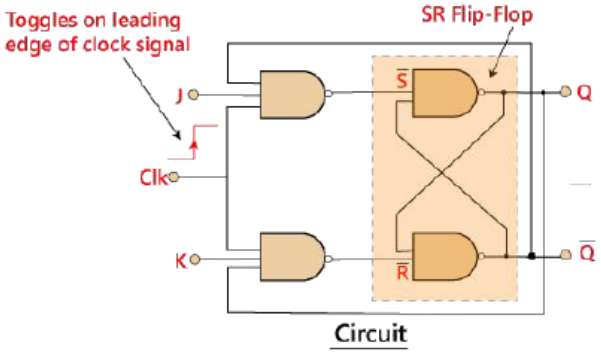


JK Flip flop

● JK flip flop is a refined & improved version of SR flip flop that has been introduced to solve the problem of indeterminate state that occurs in SR flip flop when both the inputs are 1.

● There are following two methods for constructing a JK flip flop

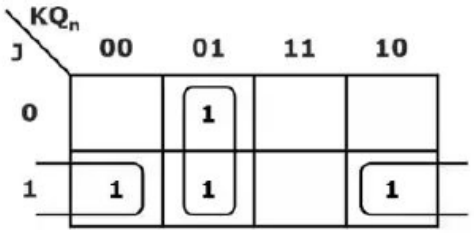
JK Flip flop



J	K	Q_n	Q_{n+1}	State
0	0	0	0	Q_n (Hold)
0	0	1	1	
0	1	0	0	Reset
0	1	1	0	
1	0	0	1	Set
1	0	1	1	
1	1	0	1	Toggel
1	1	1	0	

JK Flip flop

J	K	Q_n	Q_{n+1}	State
0	0	0	0	Q_n (Hold)
0	0	1	1	
0	1	0	0	Reset
0	1	1	0	
1	0	0	1	Set
1	0	1	1	
1	1	0	1	Toggel
1	1	1	0	

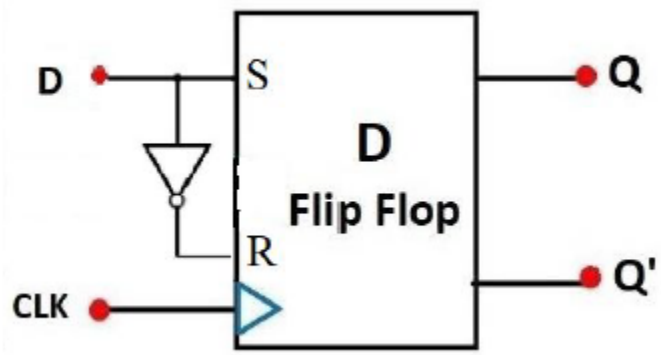


∴ Characterstic equation is $Q_{n+1} = J\bar{Q}_n + \bar{K}Q_n$

D Flip flop

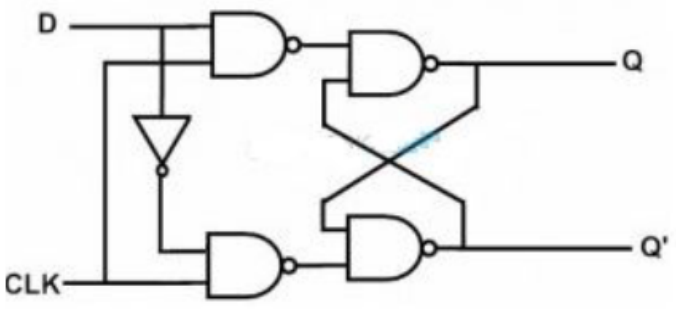
- D stands for data thus it is used to store data.
- The basic building block of D flip flop is SR flip flop.
- In SR flip flop if both i/p are the same, o/p is either no change or it is invalid (i/o -> 00, No change and o/p -> 1 1, Invalid).
- These conditions can be avoided by making them complement of each other. This modified SR flip flop is called D flip flop.

The input D goes directly to i/p S and its complement is applied to R i/p. Due to this, only two i/p conditions exist:
S=0 & R=1 or S=1 & R=0.



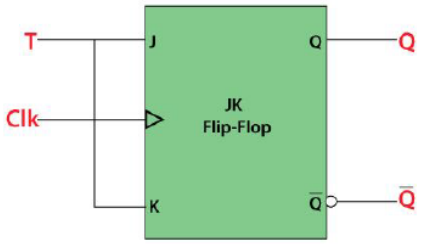
D	S	R	Q	State
	0	0	Previous State	No Change
0	0	1	0	Reset
1	1	0	1	Set
	1	1	?	Forbidden

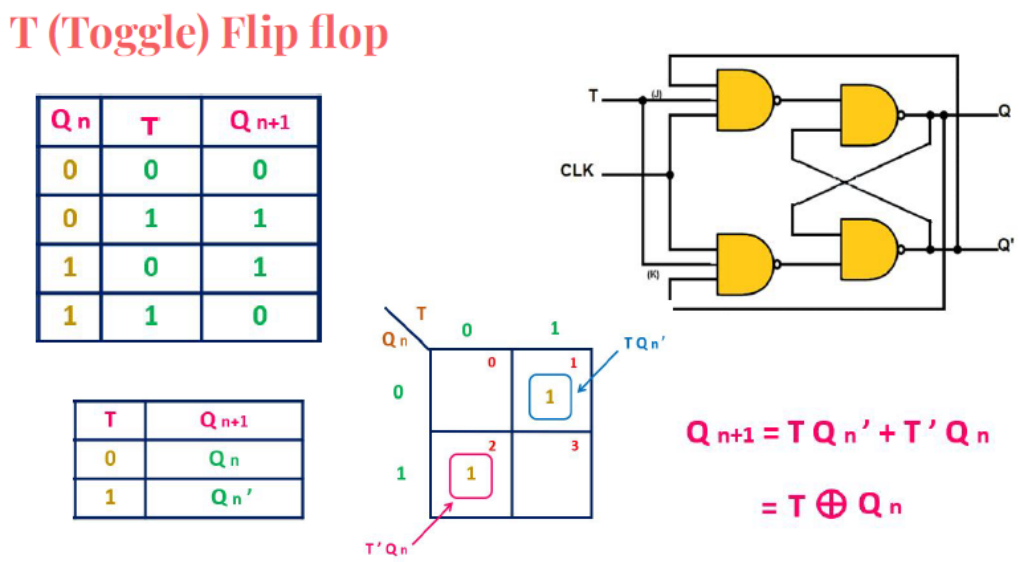
SR & D Flip Flop TruthTable



T (Toggle) Flip flop

● We can design the T flip flop by making simple modifications to the JK flip flop.
● The T flip flop is a single input device and hence by connecting J and K inputs together and giving them with a single input called T, we can convert a JK flip flop into T flip flop.
● So, a T flip flop is sometimes called a single input JK flip flop.





Applications of Flip-Flops:

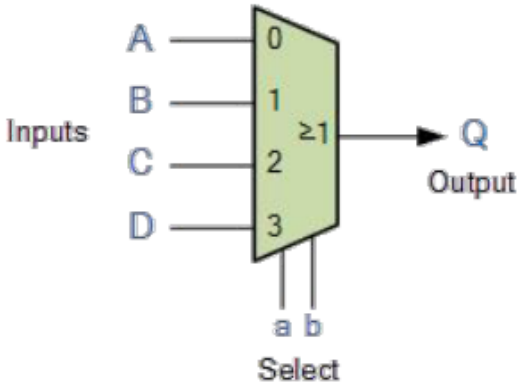
Counters
Frequency Dividers
Shift Registers
Storage Registers
Bounce elimination switch
Data storage
Data transfer
Latch
Registers
Memory

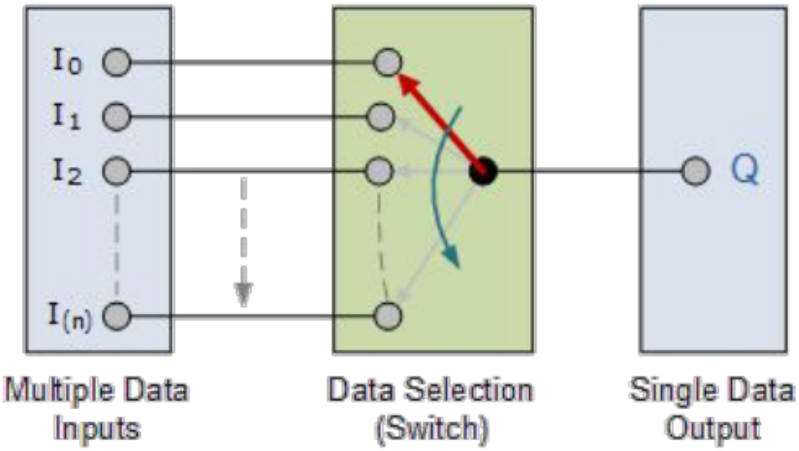
9) Mux-Demux

Multiplexer

● A multiplexer is a combinational circuit that has 2n input lines and a single output line.
● Simply, the multiplexer is a multi-input and single-output combinational circuit.
● The binary information is received from the input lines and directed to the output line. On the basis of the values of the selection lines, one of these data inputs will be connected to the output. A multiplexer is also treated as Mux.

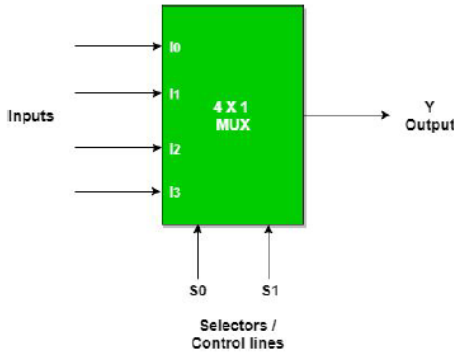
Multiplexer symbol-





4X1 Multiplexer

In 4×1 multiplexer, there are only 4 inputs, 2 selection line and single outputs.

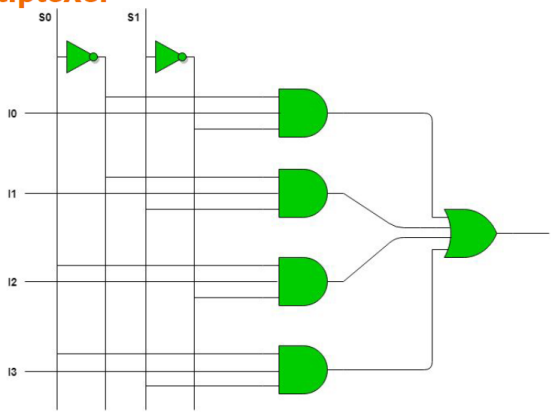


Truth Table

S0	S1	Y
0	0	I0
0	1	I1
1	0	I2
1	1	I3

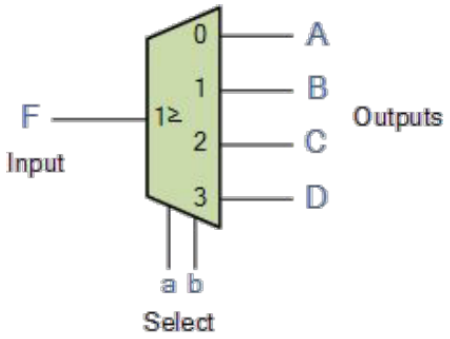
So, final equation,
Y = S0'.S1'.I0 + S0'.S1.I1 + S0.S1'.I2 + S0.S1.I3

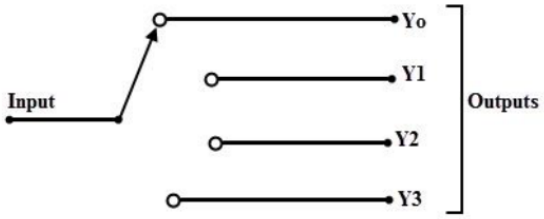
4X1 Multiplexer

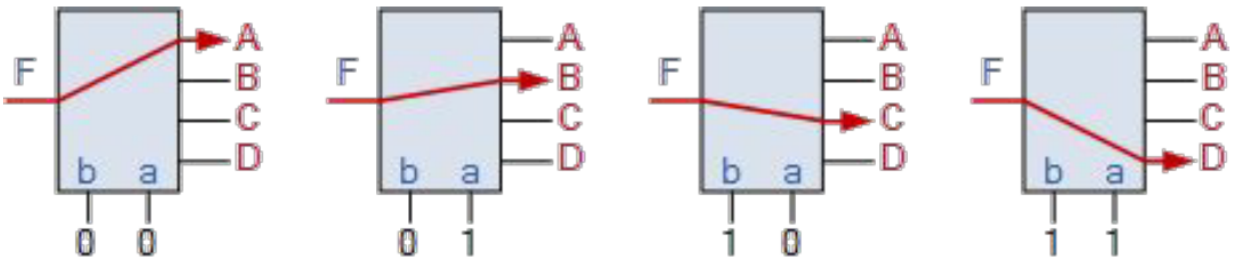


Demultiplexer

- De-Multiplexer is a combinational circuit that performs the reverse operation of Multiplexer.
- It has single input, ‘n’ selection lines and a maximum of 2n outputs.
- The input will be connected to one of these outputs based on the values of selection lines.
- Demultiplexer is also called De-Mux.

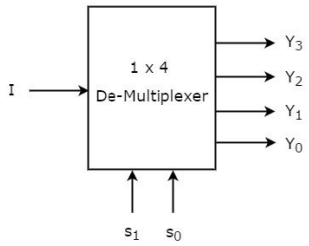






1:4 Demultiplexer

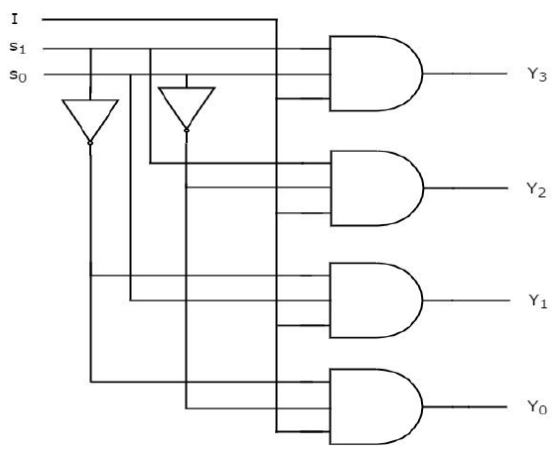
- 1x4 De-Multiplexer has one input I, two selection lines, s1 & s0 and four outputs Y3, Y2, Y1 &Y0.
- The block diagram of 1x4 De-Multiplexer is shown in the following figure.



The single input 'I' will be connected to one of the four outputs, Y_3 to Y_0 based on the values of selection lines s_1 & s0. The **Truth table** of 1x4 De-Multiplexer is shown below.

Selection Inputs		Outputs			
S_1	S_0	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	I
0	1	0	0	I	0
1	0	0	I	0	0
1	1	I	0	0	0

From the above truth table, the Boolean Expressions for the outputs as follows:



$Y0 = S_1'S_0'I$

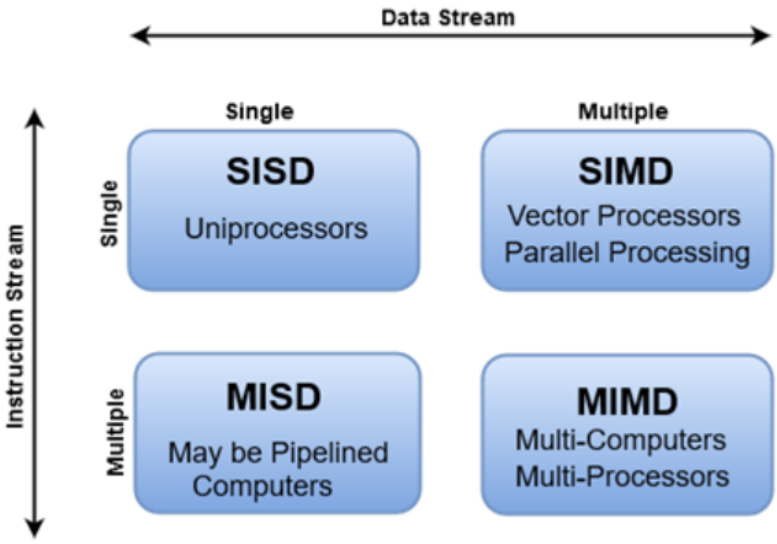
$Y1 = S_1'S_0I$

$Y2 = S_1S_0'I$

$Y3 = S_1S_0I$

10) Flynn's classification

Flynn's Classification of Computers



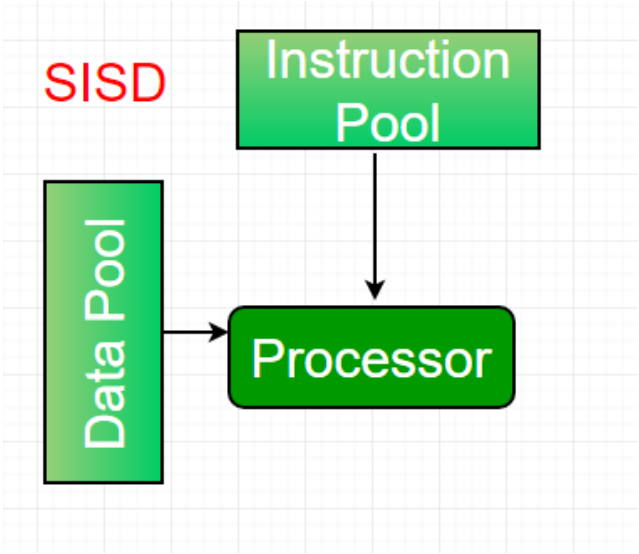
(In Short)
Single Instruction Single Data (SISD): In an SISD architecture, there is a single processor that executes a single instruction stream and operates on a single data stream. This is the simplest type of computer architecture and is used in most traditional computers.

Single Instruction Multiple Data (SIMD): In a SIMD architecture, there is a single processor that executes the same instruction on multiple data streams in parallel. This type of architecture is used in applications such as image and signal processing.

Multiple Instruction Single Data (MISD): In a MISD architecture, multiple processors execute different instructions on the same data stream. This type of architecture is not commonly used in practice, as it is difficult to find applications that can be decomposed into independent instruction streams.

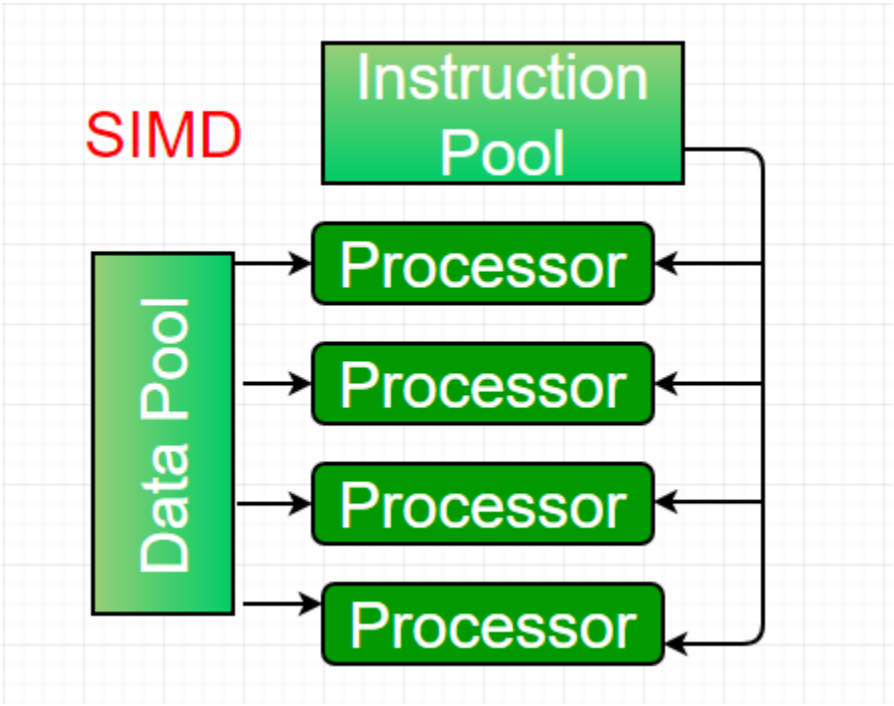
Multiple Instruction Multiple Data (MIMD): In a MIMD architecture, multiple processors execute different instructions on different data streams. This type of architecture is used in distributed computing, parallel processing, and other high-performance computing applications.

(In brief)
1. **Single-instruction, single-data (SISD) systems –** An SISD computing system is a uniprocessor machine which is capable of executing a single instruction, operating on a single data stream. In SISD, machine instructions are processed in a sequential manner and computers adopting this model are popularly called sequential computers. Most conventional computers have SISD architecture. All the instructions and data to be processed have to be stored in primary memory.



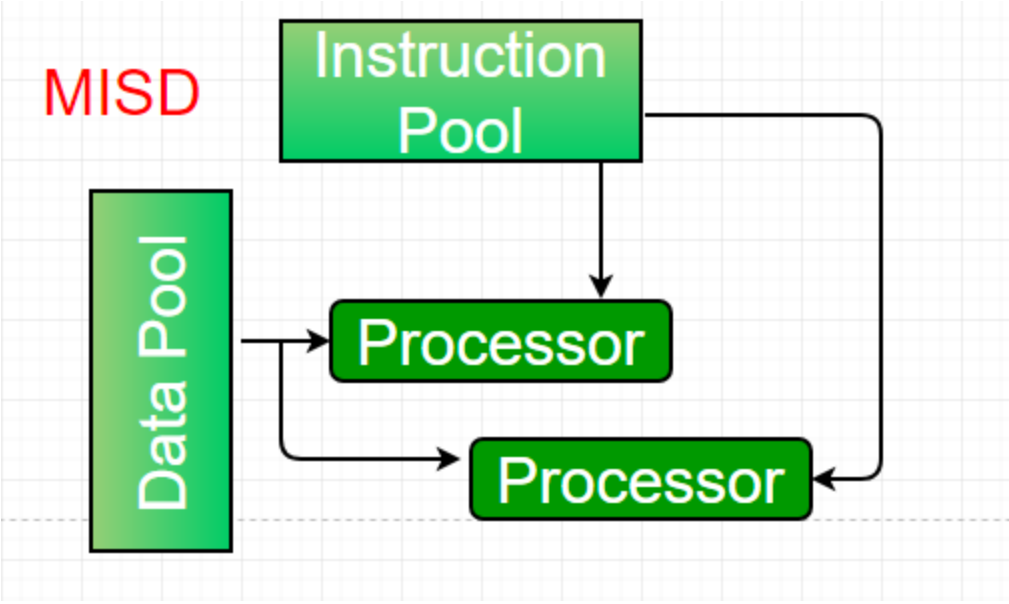
The speed of the processing element in the SISD model is limited(dependent) by the rate at which the computer can transfer information internally. Dominant representative SISD systems are IBM PC, workstations.

2. **Single-instruction, multiple-data (SIMD) systems –** An SIMD system is a multiprocessor machine capable of executing the same instruction on all the CPUs but operating on different data streams. Machines based on an SIMD model are well suited to scientific computing since they involve lots of vector and matrix operations. So that the information can be passed to all the processing elements (PEs) organized data elements of vectors can be divided into multiple sets(N-sets for N PE systems) and each PE can process one data set.



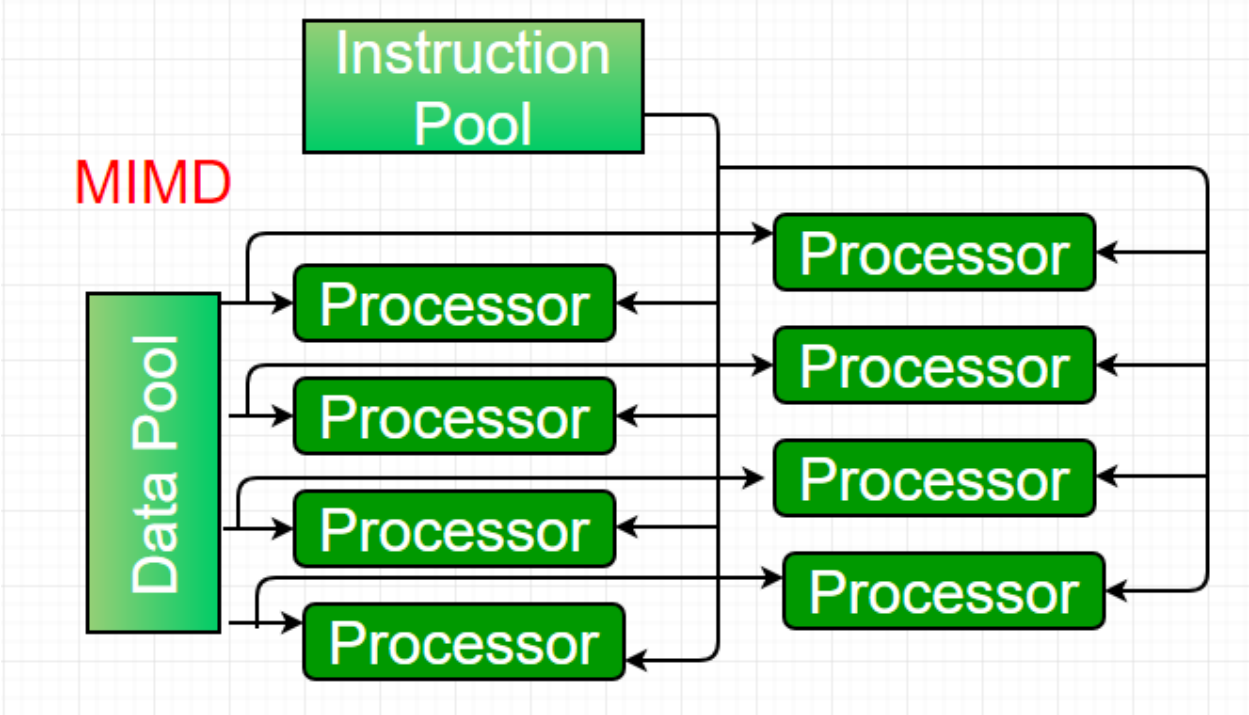
Dominant representative SIMD systems is Cray's vector processing machine.

3. **Multiple-instruction, single-data (MISD) systems –** An MISD computing system is a multiprocessor machine capable of executing different instructions on different PEs but all of them operating on the same dataset .



Example Z = sin(x)+cos(x)+tan(x) The system performs different operations on the same data set. Machines built using the MISD model are not useful in most of the application, a few machines are built, but none of them are available commercially.

4. **Multiple-instruction, multiple-data (MIMD) systems –** An MIMD system is a multiprocessor machine which is capable of executing multiple instructions on multiple data sets. Each PE in the MIMD model has separate instruction and data streams; therefore machines built using this model are capable to any kind of application. Unlike SIMD and MISD machines, PEs in MIMD machines work asynchronously.



MIMD machines are broadly categorized into **shared-memory MIMD** and **distributed-memory MIMD** based on the way PEs are coupled to the main memory. In the **shared memory MIMD** model (tightly coupled multiprocessor systems), all the PEs are connected to a single global memory and they all have access to it. The communication between PEs in this model takes place through the shared memory, modification of the data stored in the global memory by one PE is visible to all other PEs. Dominant representative shared memory MIMD systems are Silicon Graphics machines and Sun/IBM's SMP (Symmetric Multi-Processing). In **Distributed memory MIMD** machines (loosely coupled multiprocessor systems) all PEs have a local memory. The communication between PEs in this model takes place through the interconnection network (the inter process communication channel, or IPC). The network connecting PEs can be configured to tree, mesh or in accordance with the requirement. The

shared-memory MIMD architecture is easier to program but is less tolerant to failures and harder to extend with respect to the distributed memory MIMD model. Failures in a shared-memory MIMD affect the entire system, whereas this is not the case of the distributed model, in which each of the PEs can be easily isolated. Moreover, shared memory MIMD architectures are less likely to scale because the addition of more PEs leads to memory contention. This is a situation that does not happen in the case of distributed memory, in which each PE has its own memory. As a result of practical outcomes and user's requirement , distributed memory MIMD architecture is superior to the other existing models.

11) Assembler directives

Assembler directives are the directions to the assembler which indicate how an operand or section of the program is to be processed. These are also called pseudo operations which are not executable by the microprocessor. The following section explains the basic assembler directives for 8086.

ASSEMBLER DIRECTIVES:

The various directives are explained below.

1. ASSUME : The ASSUME directive is used to inform the assembler the name of the logical segment it should use for a specified segment.

Ex: ASSUME DS: DATA tells the assembler that for any program instruction which refers to the data segment ,it should use the logical segment called DATA.

2.DB -Define byte. It is used to declare a byte variable or set aside one or more storage locations of type byte in memory.

For example, CURRENT_VALUE DB 36H tells the assembler to reserve 1 byte of memory for a variable named CURRENT_ VALUE and to put the value 36 H in that memory location when the program is loaded into RAM .

3. DW -Define word. It tells the assembler to define a variable of type word or to reserve storage locations of type word in memory.

4. DD(define double word) :This directive is used to declare a variable of type double word or restore memory locations which can be accessed as type double word.

5.DQ (define quadword) :This directive is used to tell the assembler to declare a variable 4 words in length or to reserve 4 words of storage in memory .

6.DT (define ten bytes):It is used to inform the assembler to define a variable which is 10 bytes in length or to reserve 10 bytes of storage in memory.

7. EQU –Equate It is used to give a name to some value or symbol. Every time the assembler finds the given name in the program, it will replace the name with the value or symbol we have equated with that name

8.ORG -Originate : The ORG statement changes the starting offset address of the data.

It allows to set the location counter to a desired value at any point in the program.For example the statement ORG 3000H tells the assembler to set the location counter to 3000H.

9 .PROC- Procedure: It is used to identify the start of a procedure. Or subroutine.

10. END- End program .This directive indicates the assembler that this is the end of the program module.The assembler ignores any statements after an END directive.

11. ENDP- End procedure: It indicates the end of the procedure (subroutine) to the assembler.

12.ENDS-End Segment: This directive is used with the name of the segment to indicate the end of that logical segment.

12) Nano pgm

In most microprogrammed processors, an instruction fetched from memory is interpreted by a micro program stored in a single control memory (CM).In some microprogrammed processors, the micro instructions are not directly used by the decoder to generate control signals. They use second control memory called a nano control memory (nCM).

So they are two levels of control memories,

- A higher level control memories is known as micro control memory (μCM)
- A lower level control memories is known as nano control memory (nCM).

The μCM stores micro instructions whereas nCM stores nano instructions. The instruction is fetched from the main memory into instruction register IR.

- Using its opcode we load address of its first micro-instruction into μPC,

● Using this address we fetch the micro-instruction from micro control memory (μCM) into micro instruction register μIR.

● This is in vertical form and decoded by a decoder.

● The decoded output loads a new address in a nano program counter (nPC).

● By using this address, the nano-instruction is fetched from nano-control memory (nCM) into nano instruction register (nIR).

● This is in horizontal form and can directly generate control signals which can be multiple at a time.

● Such a combination gives advantage of both techniques.

● The size of the control Memory is small as micro-instructions are vertical.

Micro instruction format :

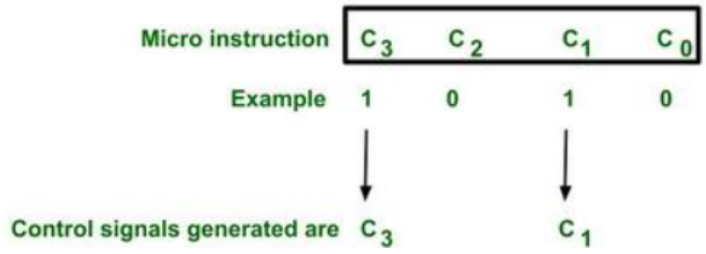
The control field of the micro-instruction decides the control signals to be produced. It is of two different formats Horizontal or Vertical. Let's discuss it one by one.

Type-1 :

Horizontal micro-instruction :

Here, we will discuss the horizontal micro-instruction format as follows.

- Every bit of the micro-instruction corresponds to a control signal.
- The bit which is 1, that corresponding control signal will be produced by the micro-instruction. If there are N bits in micro-instruction then it can produce N control signals by that micro-instruction.
- As the control signals increase, the micro-instruction grows wider. Therefore, the control memory grows horizontally.
- Executes faster because no decoder is used.
- The control memory is large because the micro-instructions are wide.

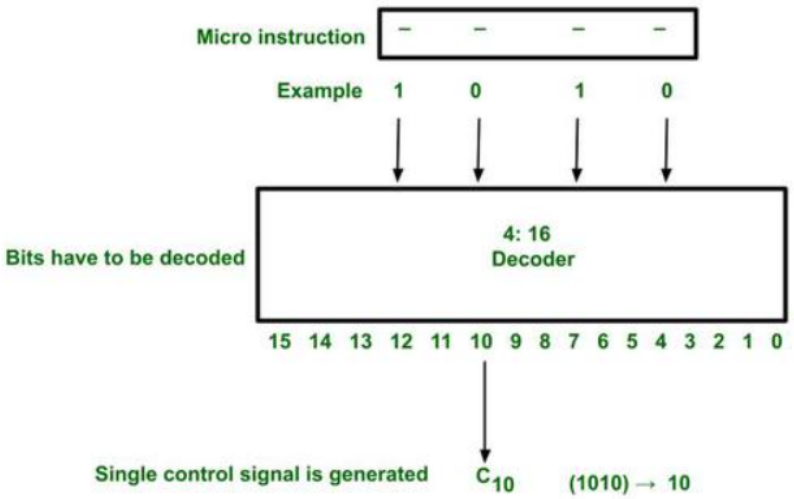


Type-2 :

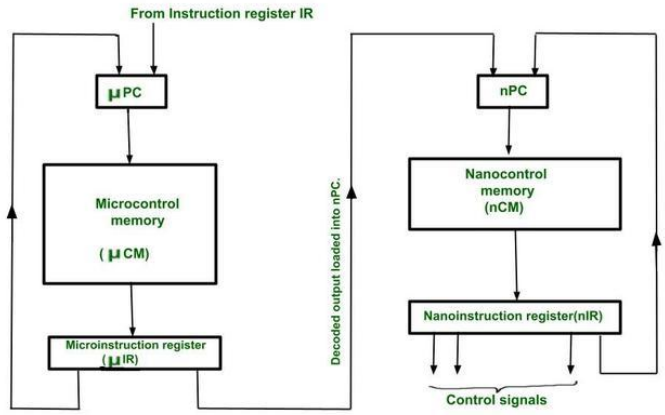
Vertical micro-instruction :

Here, we will discuss the vertical micro-instruction format as follows.

- The bits of the micro-instruction are decoded.
- The decoded output decides the control signal to be produced.
- N bits in the micro-instruction will totally generate 2N control signals.
- But one control can be generated by one micro-instruction.
- More micro-instructions are needed.
- Also, the decoding makes the execution slower.
- The circuit is complex.



1. Here we have a two-level control memory.
2. The instruction is fetched from the main memory into instruction register IR.
3. Using its opcode we load address of its first micro-instruction into μPC,
4. Using this address we fetch the micro-instruction from micro control memory (μCM) into micro instruction register μIR.
5. This is in vertical form and decoded by a decoder.
6. The decoded output loads a new address in a nano program counter (nPC).
7. By using this address, the nano-instruction is fetched from nano-control memory (nCM) into nano instruction register (nIR).
8. This is in horizontal form and can directly generate control signals which can be multiple at a time.
9. Such a combination gives advantage of both techniques.
10. The size of the control Memory is small as micro-instructions are vertical.



13) Hardwired VS Microcontroller

Hardwired Control Unit	Microprogrammed Control Unit
Hardwired control unit generates the control signals needed for the processor using logic circuits	Microprogrammed control unit generates the control signals with the help of micro instructions stored in control memory
Hardwired control unit is faster when compared to microprogrammed control unit as the required control signals are generated with the help of hardwares	This is slower than the other as micro instructions are used for generating signals here
Difficult to modify as the control signals that need to be generated are hard wired	Easy to modify as the modification need to be done only at the instruction level
More costlier as everything has to be realized in terms of logic gates	Less costlier than hardwired control as only micro instructions are used for generating control signals
It cannot handle complex instructions as the circuit design for it becomes complex	It can handle complex instructions
Only limited number of instructions are used due to the hardware implementation	Control signals for many instructions can be generated
Used in computer that makes use of Reduced Instruction Set Computers(RISC)	Used in computer that makes use of Complex Instruction Set Computers(CISC)

14) Instruction pipeline and Instruction Execution cycle

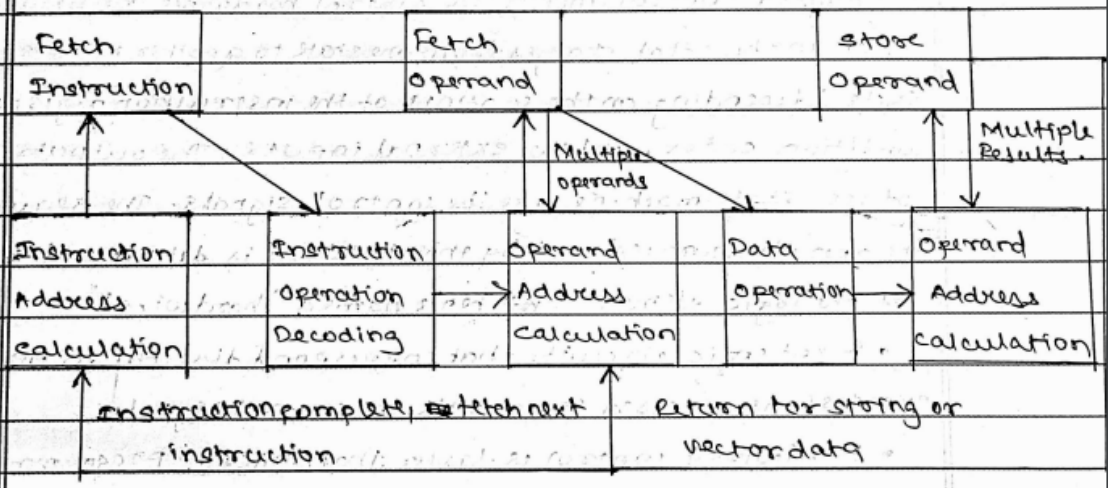
1 Draw and explain basic instruction execution cycle.

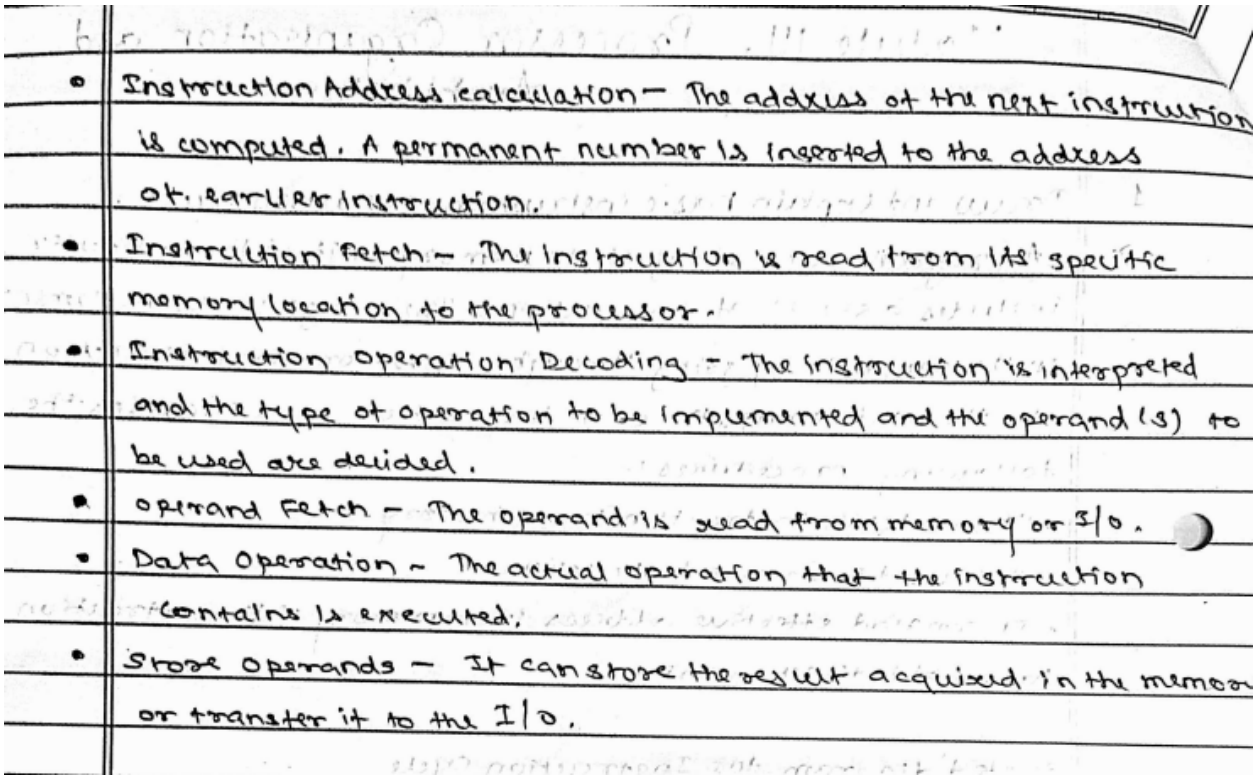
→ A program consisting of the memory unit of the computer includes a series of instructions. The program is implemented on the computer by going through a cycle for each instruction.

In the basic computer each instruction cycle includes the following procedures :-

- It can fetch instruction from memory.
- It is used to decode instruction.
- It can read effective address from memory if the instruction has an indirect access.

Stated diagram for Instruction Cycle.





15) Max VS Min mode of operations of 8086

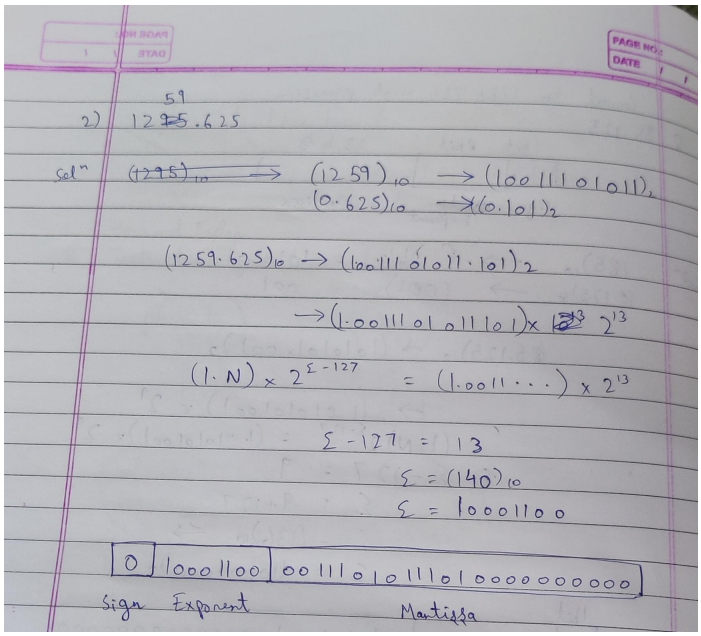
Difference between minimum mode and maximum mode in 8086 microprocessor

Minimum mode	Maximum mode
There can be only one processor.	There can be multiple processors.
Performance is slower.	Performance is faster.
The circuit is simple.	The circuit is complex.
Multiprocessing cannot be performed.	Multiprocessing can be performed.
MN/MX is 1 to indicate the minimum mode.	MN/MX is 0 to indicate the maximum mode
The 8086 generates INTA for interrupt acknowledgment.	The 8288 Bus Controller generates the interrupt acknowledgment signal (INTA).
The 8086 itself provides an ALE for the latch.	Because there are several processors, the 8288 bus controller provides ALE for the latch.
The system is more affordable.	The system costs more money.
It is used for small systems.	It is used for large systems.
The multiprocessor setup is not supported.	The multiprocessor configuration is accepted.

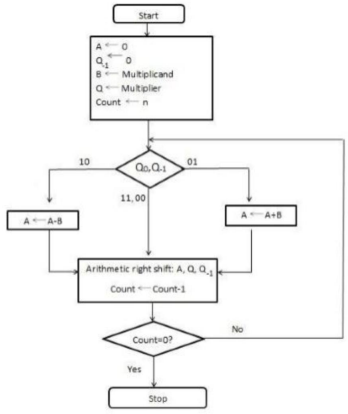
Feature	Minimum Mode	Maximum Mode
Bus width	8-bit	Multiplexed bus
Address bus width	20-bit	20-bit
Control signal	Single bus control signal	Multiple bus control signals
Additional support chips	Fewer support chips required	Additional support chips required
Coprocessor support	Not supported	Supported
Maximum number of coprocessors	0	Up to 3
Bus controller	Not required	Required
Clock generator	Required	Required
Data buffer	Not required	Required
Interrupt controller	Built-in	External
Interrupt controller	Not available	Available

16) (IEEE,Booth's algo,Restoring,Instructions)--> Pakka aa rahe 2-3Q

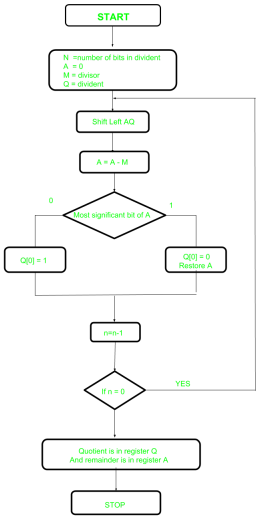
IEEE



Booth's algo



Restoring division algorithm



17) Associative and Interleaved

Interleaved memory.

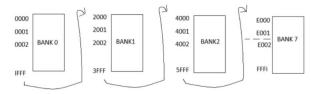
The memory system in which successive addresses are evenly spread across memory bank to compensate for the relatively slow speed of DRAM.

The contagious memory reads and writes are using each bank in term, resulting in higher memory throughputs due to reduced waiting for memory banks to become ready for desired operations.

Types of Interleaving:

1. Low order Inter leaving.

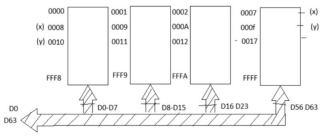
In this lower order address lines eg. A0-A12 used to identify location in each bank whereas higher order address lines are decoded to generate chip select (CS) of individual memory chip. In this low order interleaving successive address will get allocated in the same memory chip as shown below.



Here in this low order interleave memory even if contagious addresses generated by CPU it will be allowed byte by byte transfer.

2. High order Interleaving.

In this case higher order address lines, eg, A3 - A15 used to identify location in each bank where as lower order address lines eg. A0, A1, A2 will be decoded to generate CS 07 individual chip. This will allocate successive addresses in the different memory bank as shown below.

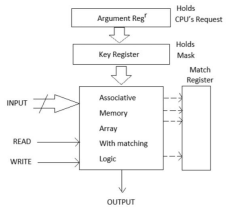


In above high order interleaved memory it can be possible to read and write 8 data words simultaneously and have data transfer speed is comparative faster.

3. Associate memory (CAM).

This memory is also known as content addressable memory. Its hardware organization consist of argument register to hold CPU's request & a key register to hold predefined "mask" which is logically 'AND'ed with the CPU's request in argument register. The result of CPU's request will be masked and remaining unmasked bits are then parallel y searched against each word in the associative memory array as shown below.

During this parallel search as soon as the match is found the corresponding match register bit will be set and the data word from this matched location is placed on the OUTPUT for CPU.



The associative memory finds an application where search time is critical and very short. For example, directory search in the computer or website search in the internet system makes used of associative memory.

In this associative memory if the matched words are contagious the will be placed on OUTPUT, one after another without any intermediate gap as a result the data transfer rate will be very FAST for the associative memory.

18) I/O ke types

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

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

2) **<u>Interrupt- initiated I/O:</u> Since in the above case we saw the CPU is kept busy unnecessarily. This situation can very well be avoided by using an interrupt driven method for data transfer. By using interrupt facility and special commands to inform the interface to issue an interrupt request**

signal whenever data is available from any device. In the meantime the CPU can proceed for any other program execution. The interface meanwhile keeps monitoring the device. Whenever it is determined that the device is ready for data transfer it initiates an interrupt request signal to the computer. Upon detection of an external interrupt signal the CPU stops momentarily the task that it was already performing, branches to the service program to process the I/O transfer, and then return to the task it was originally performing.

The I/O transfer rate is limited by the speed with which the processor can test and service a device.

The processor is tied up in managing an I/O transfer; a number of instructions must be executed for each I/O transfer.

Terms:

Hardware Interrupts: Interrupts present in the hardware pins.

Software Interrupts: These are the instructions used in the program whenever the required functionality is needed.

Vectored interrupts: These interrupts are associated with the static vector address.

Non-vectored interrupts: These interrupts are associated with the dynamic vector address.

Maskable Interrupts: These interrupts can be enabled or disabled explicitly.

Non-maskable interrupts: These are always in the enabled state. we cannot disable them.

External interrupts: Generated by external devices such as I/O.

Internal interrupts: These devices are generated by the internal components of the processor such as power failure, error instruction, temperature sensor, etc.

Synchronous interrupts: These interrupts are controlled by the fixed time interval. All the interval interrupts are called as synchronous interrupts.

Asynchronous interrupts: These are initiated based on the feedback of previous instructions. All the external interrupts are called as asynchronous interrupts.

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

Bus grant request time.

Transfer the entire block of data at transfer rate of device because the device is usually slow than the speed at which the data can be transferred to CPU.

Release the control of the bus back to CPU So, total time taken to transfer the N bytes = Bus grant request time + (N) * (memory transfer rate) + Bus release control time.

Buffer the byte into the buffer

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