Computer Organization and Architecture Syllabus

Stored Program Concept, Components of a Computer System, Machine Instruction, Op codes and Operands, Instruction Cycle,Organization of Central Processing Unit: ALU, Hardwired & Micro programmed Control Unit, General Purpose and Special Purpose Registers, Memory Organization, I/O Organization,Functioning of CPU, Instruction Formats, Instruction Types,Addressing Modes, Common Microprocessor Instructions, Multi-core Architecture, Multiprocessor and Multicomputer.

Computer Organization and Architecture

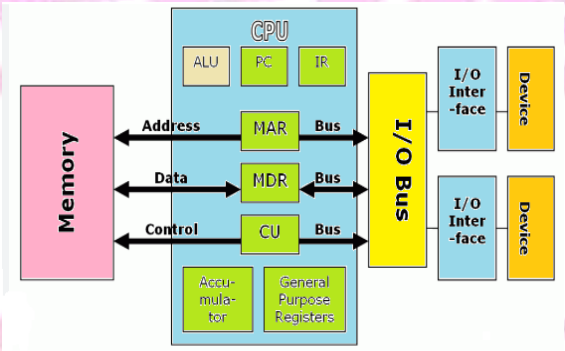
Computer Architecture

Study of the structure, functional behaviour and design of computers.

Computer Organization

Study of interconnection of operational units that realize the architectural specifications.

CO + CA= Systematic approach to solve problems.



Computer Architecture

John Von Neumann Architecture

Non John von neumann Architecture

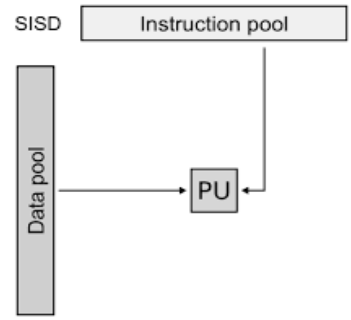
Harvard Architecture

Non Harvard Architecture

Flynn's Taxonomy

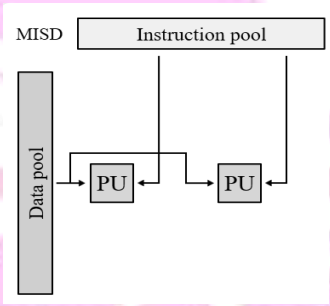
Single Instruction, Single Data (SISD):

- This is a uniprocessor machine which is capable of executing a single instruction, operating on a single data stream.
- The 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 by the rate at which the computer can transfer information internally.



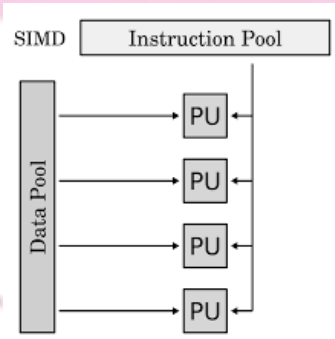
Multiple Instruction, Single Data (MISD):-

- An MISD computing system is a multiprocessor machine capable of executing different instructions on different Processing Elements but all of them operating on the same dataset.



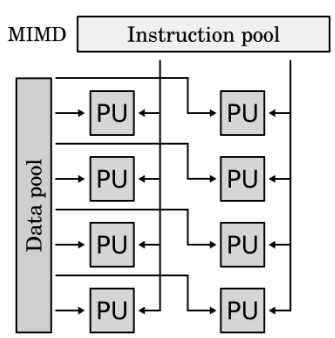
Single Instruction, Multiple Data (SIMD):

- This 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 and each PE can process one data set.



Multiple Instruction, Multiple Data (MIMD):

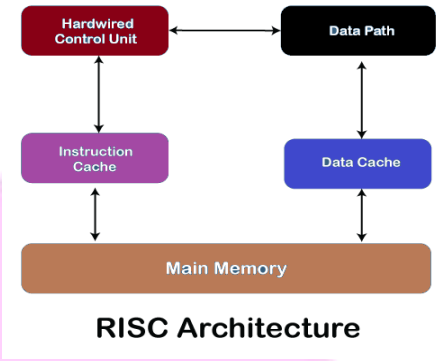
- This is capable of executing multiple instructions on multiple data sets.
- Each PE in the MIMD model has separate instruction and data streams; therefore machines built using this model are capable to any kind of application.
- Unlike SIMD and MISD machines, PEs in MIMD machines work asynchronously.



RISC and CISC architecture

Reduced Instruction Set Computer

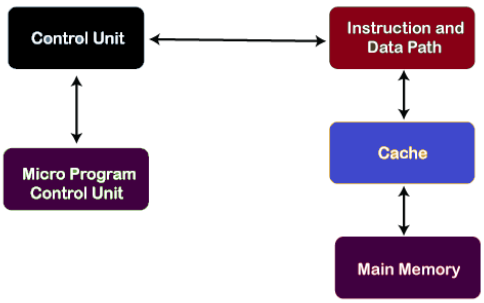
- RISC processors work with more instructions; however, the number of cycles an instruction may take to execute is minimized.
- It is built to minimize the instruction execution time by optimizing and limiting the number of instructions.
- A RISC machine takes approx one CPU cycle to complete one instruction.
- Examples: Alpha, ARC, ARM, AVR, MIPS, PA-RISC, PIC, Power Architecture, SPARC etc



RISC Architecture

Complex Instruction Set Computer

- In this architecture, combining multiple simple commands into one complex instruction.
- Examples: AMD, Intel x86, VAX, and System/360.



CISC Architecture

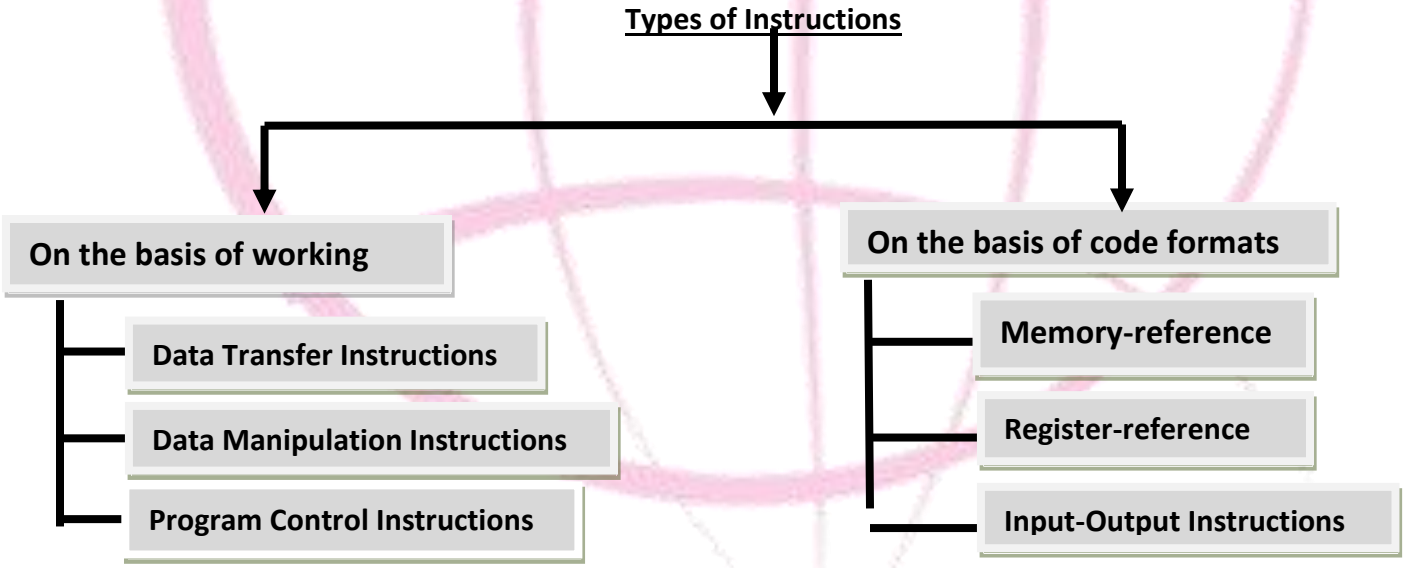
Differences between CISC and RISC processor architectures:

SNo.	CISC	RISC
1.	The CSIC architecture processes complex instructions that require several clock cycles for execution. On average, it takes two to five clock cycles per instruction (CPI).	The RISC architecture executes simple yet optimized instructions in a single clock cycle. It processes instructions at an average speed of 1.5 clock cycles per instruction (CPI).
2.	Implementation of complex instructions is enabled through memory units.	RISC lacks special memory and thus utilizes specialized hardware to execute instructions.
3.	CISC devices are installed with a microprogramming unit.	RISC devices are embedded with a hardwired programming unit.
4.	CISC uses a variety of instructions to accomplish complex tasks.	RISC is provided with a reduced instruction set, which is typically primitive in nature.
5.	A CISC architecture uses one cache to store data as well as instructions. However, recent CISC designs employ split caches to divide data and instructions.	The RISC architecture relies on split caches, one for data and the other for instructions.
6.	CISC processors use a memory-to-memory framework to execute instructions such as ADD, LOAD, and even STORE.	RISC processors rely on a register-to-register mechanism to execute ADD, STORE, and independent LOAD instructions.
7.	The CISC architecture uses only one register set.	The RISC architecture utilizes multiple registers sets.
8.	CISC processors are capable of processing high-level programming language statements more efficiently, thanks to the support of complex	Since RISC processors support a limited set of addressing modes, complex instructions are synthesized through software codes.

	addressing modes.	
9.	CISC does not support parallelism and pipelining. As such, CISC instructions are less pipelined.	RISC processors support instruction pipelining.
10.	CISC instructions require high execution time.	RISC instructions require less time for execution.
11.	In the CISC architecture, the task of decoding instructions is quite complex.	In RISC processors, instruction decoding is simpler than in CISC.
12.	Examples: Intel x86 CPUs, System/360, VAX, PDP-11, Motorola 68000 family, and AMD.	Examples: Alpha, ARC, ARM, AVR, MIPS, PA-RISC, PIC, Power Architecture, and SPARC.
13.	CISC processors are used for low-end applications such as home automation devices, security systems, etc.	RISC processors are suitable for high-end applications, including image and video processing, telecommunications, etc.

Machine Instructions

- Machine instructions are machine code programs or commands which instruct the processor to perform a specific task.
- Machine instruction is a sequence of binary bits (0/1) which is executed by processor.
- An instruction can be divided in 3 parts:
  - Opcode
  - Operand
  - Addressing Mode



On the basis of working
Data Transfer Instructions

- Transfer instructions allow data to be transferred from one place to another without modifying its content.
- A shared transfer may occur between the memory and processor registers or between the processor registers and input/output.

LOAD: Transfer of data from memory into the register.
IN: Receives data from an input device.
MOVE: Data transfer between registers.
OUT: Outputs data from the register.

PUSH: Pushes data from a register towards the top of the stack.
STORE: Transfer of data from the register to the memory.
POP: Fetches top data from register or stack memory.
XCHG: Transfers information between registers and memory.

Instructions for Data Manipulation

It is also called computation instruction.

a. **Arithmetic Instructions:-** It is used for adding, subtracting, multiplying, dividing, increment and decrement.
   Example:- ADD, INC, MUL, DEC, SUB, DIV, etc.
b. **Arithmetic and Logical Instructions:-** The function performs "arithmetic + shift left" and "arithmetic + shift right".
   Example:- SAR,SAL;
c. **Logical Instructions:-** It is used for bit-wise operations like AND, NOT, Exclusive-OR, OR, shift, and rotate are performed.
   Example: NOT, AND, ROL, XOR, OR, SHL, ROR SHR, etc.

Program Control Instructions

a. **Compare Test Instructions:- To compare and test status flags we use Condition Code Register (CCR) which has some flag bits:**
   - 5 status bits: used to indicate the situation after the arithmetic and logic operation.
     Example:- Carry flag (C), Overflow flag (V), Negative Flag (N),Half Carry (H), Zero Flag (Z)
   - One Stop Disable bit.
   - Two Interrupt Masking bits.

b. **Unconditional Branch Instructions: It causes an unconditional change of execution sequence to a new location.**
   Example : JUMP, goto etc.
c. **Conditional Branch Instructions: These instructions are used to examine the values stored in the condition code register to determine whether the specific code exists and to branch if it does.**

Examples :

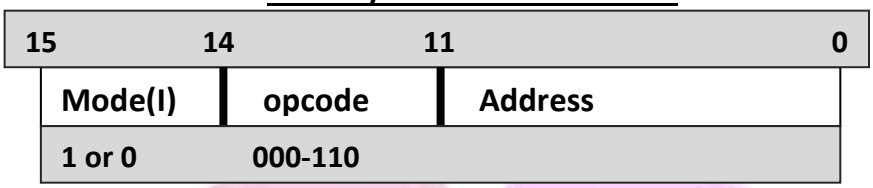
BL – Branch in Less than condition – (<)
BGE – Branch in greater than equal to condition – (>=)
BNE – Branch if not equal condition– (!=) etc.

d. **Subroutines:** It is a program fragment that lives in user space, performs a well-defined task if it is invoked by another user programs and returns control to the calling program when finished.
   Example : CALL, RET instructions.
e. **Halting and Interrupt instructions:** NOP Instruction, HALT Instruction, Interrupt Instructions(RESET, TRAP, INTR etc)

On the basis of code format

- The instructions are made up of a specific number of bits.
- In general an instruction has 16 bits which is divided in 3 parts(mode, opcode and address).
- The first 12 bits of memory (0-11) specify an operation address.
- The next three bits (12-14) specify an opcode,
- The last bit (15) specifies the addressing mode.
- It refers to memory oprand.

Memory-Reference Instruction

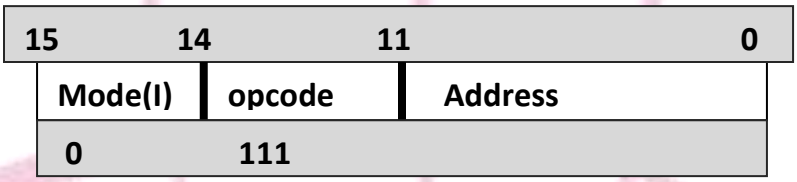


Here the opcode is between 000 to 110. (000,001,010,011,100,101,110)

- If I is 0, it specifies a direct addressing mode,
- If I is 1, it specifies an indirect addressing mode.

		Hexadecimal		
SNo.	**Instruction**	**I=0 direct**	**I=1 Indirect**	**Description**
1.	AND	0XXX	8XXX	Performs AND operation between memory and accumulator.
2.	ADD	1XXX	9XXX	Performs add operation between memory and accumulator.
3.	LDA	2XXX	AXXX	Load accumulator in memory
4.	STA	3XXX	BXXX	Store accumulator in memory
5.	BUN	4XXX	CXXX	Branch unconditionally
6.	BSA	5XXX	DXXX	Branch and save return address
7.	INC	6XXX	EXXX	Increment

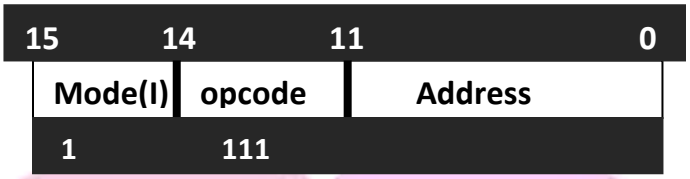
Register-reference instructions



- The opcode for a register reference instruction is always 111.
- The mode bit is always zero for register reference instruction.
- The remaining 12 bits specifies an operation on or a test of the accumulator register.
- It does not refer to the memory for operand.

SNo.	**Instructions**	**I=0**	**Description**
1.	CLA	7800	Clear AC
2.	CLE	7400	Clear E(overflow bit)
3.	CMA	7200	Complement AC
4.	CME	7100	Complement E
5.	CIR	7080	Circulate right AC and E
6.	CIL	7040	Circulate left AC and E
7.	INC	7020	Increment AC
8.	SPA	7010	Skip next instruction if AC > 0
9.	SNA	7008	Skip next instruction if AC < 0
10.	SZA	7004	Skip next instruction if AC = 0
11.	SZE	7002	Skip next instruction if E = 0
12.	HLT	7001	Halt computer

Input-Output Insruction



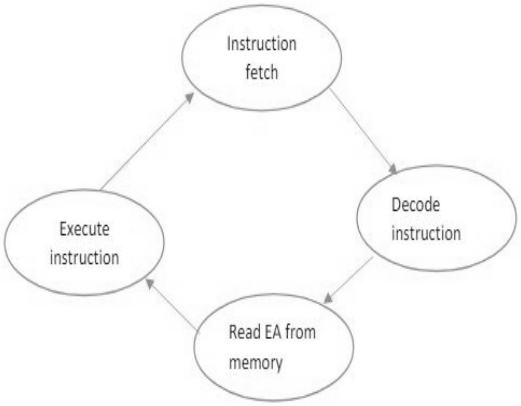
- The opcode for an I/O reference instruction is always 111.
- The mode bit is always 1 for I/O reference instruction.
- The remaining 12 bits specifies the type of I/O operation or a test performed.

SNo.	Instructions	I=1	Description
1.	INP	F800	Input character to AC
2.	OUT	F400	Output character from AC
3.	SKI	F200	Skip on input flag
4.	SKO	F100	Skip on output flag
5.	ION	F080	Interrupt On
6.	IOF	F040	Interrupt Off

Instruction set completeness

All the required number of instructions to perform a task is said to be complete instruction set.
A complete instruction set may include:-
1. Data transfer instructions
2. Arithmetic, logical, and shift instructions.
3. Program control Instructions
4. Input and Output instructions

Instruction cycle



- The instruction cycle is a process in which a computer system fetches an instruction from memory, decodes it, and then executes it.
- It is also called Fetch-Decode-Execute Cycle.
- Each phase of Instruction Cycle can be decomposed into a sequence of elementary micro-operations called microinstructions.
- Each instruction cycle consists of the following phases:
  1. Fetch instruction from memory.
  2. Decode the instruction.
  3. Read the effective address from memory.
  4. Execute the instruction.
- Upon the completion of step 4, the control goes back to step 1 to fetch, decode, and execute the next instruction.
- This cycle repeats indefinitely unless a HALT instruction is encountered

Registers needed for each instruction cycle

1. **Program Counter(PC):-** Holds the address of the next instruction to be fetched.
2. **Memory address registers (MAR):-** It is connected to System Bus address lines. It specifies the address of a read or write operation in memory.
3. **Memory Buffer Register (MBR):-** It is connected to the data lines of the system bus. It holds the memory value to be stored, or the last value read from the memory.
4. **Instruction register (IR):-** Holds the last(recent) instruction fetched.

Before starting instruction cycle the program counter has address of first instruction and sequence counter is zero.

1. Fetch cycle:-

Step1:- The address of the next instruction is fetched from Program Counter(PC) and transferred to the Memory Address Register(MAR). It is done at T0 clock pulse.

Step 2:- The address in MAR is put on the address bus and the control unit request a memory read and fetch the data of that address and send it back using data bus and data is copied into the memory buffer register.
Now Program counter is incremented by one, to get ready for the next instruction.
It is done at T1 clock pulse.

Step 3: The content of the MBR is moved to the instruction register(IR) to decode. It is done at T2 clock pulse.

Fetch and decode sequence:-
T0: AR←PC
T1: IR←M[AR] , PC<- PC+1
T2: D0 ,D1 ..D7←Decode IR(12-14), AR←IR(0- 11), I←IR(15)

2. Decode cycle:- When the instruction is in IR then it will be decoded by decoder at T2 clock pulse.

- Here we get either register reference or I/O reference or memory (direct/ indirect) reference instruction depends upon the value of D7 and I.
- Indirect bit is transferred to flipflop I & address part is transferred to AR.

3. Indirect Cycle:-

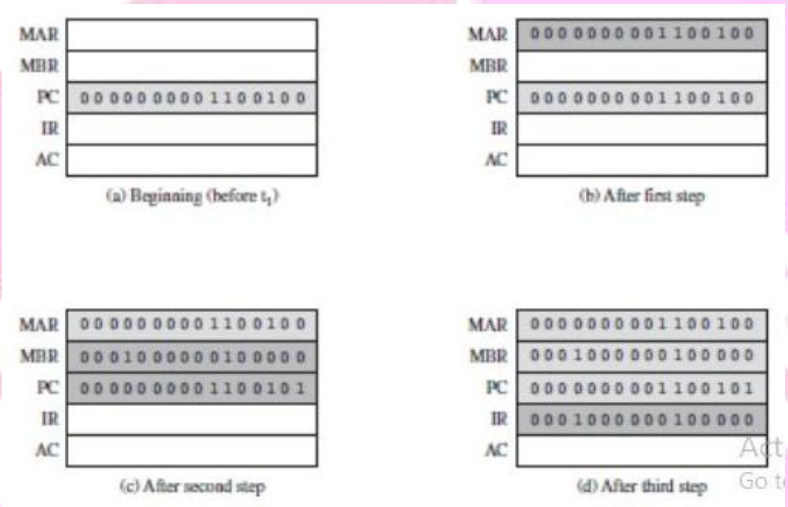
- The indirect cycle is sometimes required when instructions involve accessing memory locations that contain addresses or pointers to the actual data or another address.
- During this cycle, the CPU may use an address obtained from the previous instruction to access another memory location, which holds the data or another address to be used in the next cycle.
- The indirect cycle enables the CPU to follow memory references and retrieve the actual data required for execution.

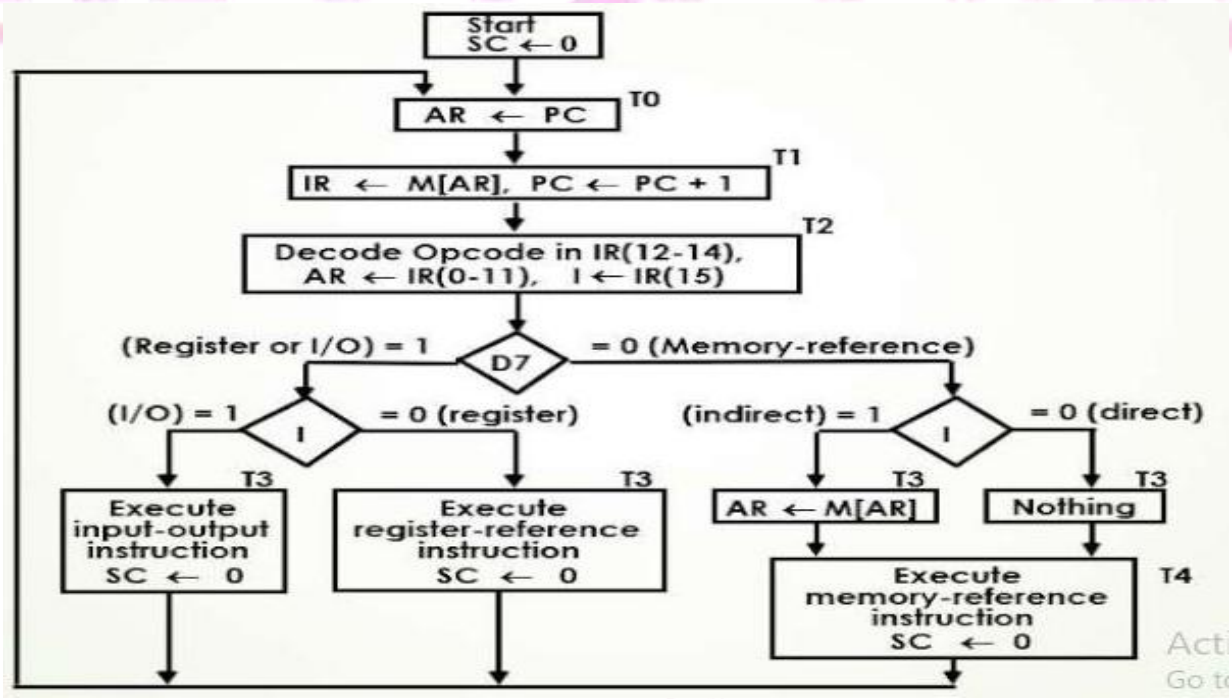
4. Execute cycle:-

Case1:- In case of a memory instruction (direct or indirect) the execution phase will be in the next clock pulse T3.
Case1.1- If the instruction is direct, nothing is done at T3 clock pulse.
Case1.2-If the instruction has an indirect address, the effective address is read from main memory, and any required data is fetched from main memory to be processed and then placed into data registers(Clock Pulse: T3).
Case2:-If the instruction is an I/O reference or a Register reference, the operation is performed (executed) at clock Pulse: T3.

5. Interrupt cycle:-

- When an interrupt occur the current contents of the PC must be saved so that the processor can resume normal activity after the interrupt.
- Thus, the contents of the PC are transferred to the MBR to be written into memory.
- The special memory location reserved for this purpose is loaded into the MAR from the control unit.
- It might, for example, be a stack pointer.
- The PC is loaded with the address of the interrupt routine. As a result, the next instruction cycle will begin by fetching the appropriate instruction.





Uses of Different Instruction Cycles

- **Fetch cycle:** Grabs instructions from memory.
- **Decode cycle:** Figures out what the instruction means and what data it needs.
- **Indirect cycle:** It retrieves data from memory using a memory address stored in a register, enabling flexible data manipulation and addressing capabilities in computer processing.
- **Execute cycle:** Does the actual work of the instruction, like math or moving data around.
- **Interrupt cycle:** It pause the computer's current tasks to deal with urgent events, like input from devices or errors, ensuring smooth and timely operation.

- **Store cycle:** Saves the result of the instruction.

Advantages

- **Standardization:** Provides a consistent way for CPUs to work, making it easier for software and hardware to work together.
- **Efficiency:** Breaks down tasks into smaller steps, helping CPUs work faster.
- **Pipelining:** Lets CPUs work on multiple instructions at once, improving speed.

Disadvantages

- **Overhead:** Adds extra steps to every instruction, slowing things down a bit.
- **Complexity:** Can be hard to design and understand, especially for complex CPUs.
- **Limited parallelism:** Sometimes, instructions depend on each other, so they can't all be done at once, slowing things down.

Issues of Different Instruction Cycles

- **Pipeline hazards:** Delays caused by instructions depending on each other in a pipeline.
- **Branch prediction errors:** Guessing wrong about which way a program will go, wasting time.
- **Instruction cache misses:** Instructions not found in the fast memory, causing delays.
- **Instruction-level parallelism limitations:** Not all instructions can be done at once, slowing down some tasks.
- **Resource contention:** Delays caused by instructions fighting over the same resources.

Instruction Format

- Instruction formats in computer architecture refer to the way instructions are encoded and represented in Machine Language.
- The instruction formats are a sequence of bits (0 and 1) which contains some fields (addressing mode, opcode and operands).
- Each field of the instruction provides specific information to the CPU related to the mode, operation and location of the operands.
- In other words we can say the instruction format also defines the layout of an instruction in bits.
- The instruction format depends upon the CPU organization and CPU depends upon the Instructions Set Architecture implemented by the processor.
- As per CPU organization instructions may have several different lengths containing varying number of addresses (3, 2, 1 or 0).
- The number of address field in the instruction format depends on the internal organization of its registers.

CPU organizations

CPU organization is the general structure and behaviour of a computer. The CPU is made up of three main parts:

- Register set
- Control unit (CU)
- Arithmetic and logic unit (ALU)

Types of CPU Organization

On the basis of registers used by processor it is divided in following types:-

1. Single Accumulator Organization
2. General Register Organization
3. Stack Organization

1. Single Accumulator Organization

- In single accumulator organization, accumulator is used implicitly for processing all instructions of a program and storing the results into the accumulator.
- This indicates that one of the operands is implied to be in the accumulator and the other operand is specified along with the instruction.
- Due to accumulator storage instructions are short which minimizes the internal complexity of the computer.
- The instruction format uses only one address field (i.e. one-address instruction format).

Example:

```
LOAD A          //AC <- M[A]
ADD A           //AC <- AC+M [A]
STORE C         //M [C] <- AC
```

Advantages:

- **Simplicity:** It's easier to design and understand because there's only one main register.
- **Cost-effective:** Requires fewer hardware components, making it cheaper to produce.

Disadvantages:

- **Limited performance:** Processing speed may be slower since only one accumulator is available.
- **Reduced flexibility:** Might not be suitable for handling complex operations efficiently compared to architectures with multiple registers.

2. General Register Organization

- It has set of general purpose registers (R0, R1, R2...) to process and store data.
- Each address field may specify a processor register or a memory operand.
- It is the most widely used models for computer architecture
- It specifies all operands explicitly.
- The instruction format needs 2 or 3 address fields according to the operation (i.e. 2 or 3-address instruction Format).

Example:

```
ADD R1, R2, R3          //R1 <- R2+R3
```

- We can also write the above instruction with two address fields if the destination register is the same as one of the source registers.

```
ADD R1, R2          //R1 <- R1+R2
MOV R1, R2          //R1 <- R2
ADD R1, A           //R1 <- R1 + M [A]
```

Advantages:

- Using lots of registers makes the CPU work faster.
- Programs take up less memory because they're written more efficiently.

Disadvantages:

- Compilers have to be smart to avoid using too many registers.
- It costs more because of all the extra registers.

3. Stack Organization

- As we know stack maintains a stack pointer which points the top of the stack to access the data.
- The operands are implicitly specified on top on the stack.
- It does not use address field for data manipulation instructions (add, mul, sub etc).

- It is also called zero address instruction formats.
- Stack operations involve two main actions: Push and Pop both are performed at the TOS and required an address field.

Example:-

```
Push A          // SP<- A
Push B          // SP <-B
Mul             // SP<- A*B
Push C          // C <- SP
```

Advantages:-

- Efficient computation of complex arithmetic expressions.
- Execution of instructions is fast because operand data are stored in consecutive memory locations.
- Length of instruction is short as they do not have address field

Disadvantages:-

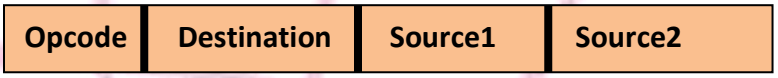
- The size of the program increases.

Types of Instruction formats

1. **Three-address instruction format**
2. **Two-address instruction format**
3. **One-address instruction format**
4. **Zero-address instruction format**

1. Three-address instruction format

- It has three address fields. One address field is used for destination and two address fields for source.
- Each address field may specify a processor register or a memory operand. It means More than one addressing mode can be used for address fields.



Example:-

1. ADD R1, R2, R3 //R1 <- R2 + R3
2. ADD R1, A, B //R1 <- M[A] + M[B]
3. Mul A, B, R1 // M[A] <- M[B] * R1

EVALUATE C=A*B (using Three address instructions format)

```
MUL C,A,B          // M[C] <- M[A] * M[B]
```

2. Two address instructions format

- Mostly computers uses two address field instructions format.
- Here we use two address fields for operand locations.
- More than one addressing mode can be used for address fields.

Example:

1. ADD R1, R2 // R1 <- R1 + R2
2. MOV R1, R2 // R1 <- R2
3. Mul R1, X // R1 <- R1 * M[X]

EVALUATE C=A*B (using Two address instructions format)

```
MUL A, B           // M[A] <- M[A] * M[B]
STORE C,A          // M[C] <- M[A]
```

3. One-address instruction format

- It has only one address field and it implied ACCUMULATOR (AC) register for all operations.
- The operand specified in the instruction may be source or destination, depending on instruction.
- All operations done between accumulator register and memory operand.

Example:
ADD A // AC <- AC + M[A]

EVALUATE C=A*B (using One address instructions)
LOAD A // AC <- M[A]
ADD B // AC <- AC + M[B]
STORE C // M[C] <- AC

- ✓ Here in LOAD instruction given operand A act as a source operand location and AC register is destination location.
- ✓ STORE instruction given operand C act as a destination location and AC register is source location.
- ✓ LOAD and STORE instruction is used to transfer to and from memory location to AC register.

4. Zero address instruction

- In zero address instructions Stack is used. Also called stack based organization.
- A stack based computer does not use an address field in the data manipulation instructions (like ADD, MUL)
- However, PUSH and POP instructions need an address field to specify the operand that placed in stack.
- To evaluate an arithmetic expression first it is converted to Revere Polish Notation i.e. Postfix Notation.

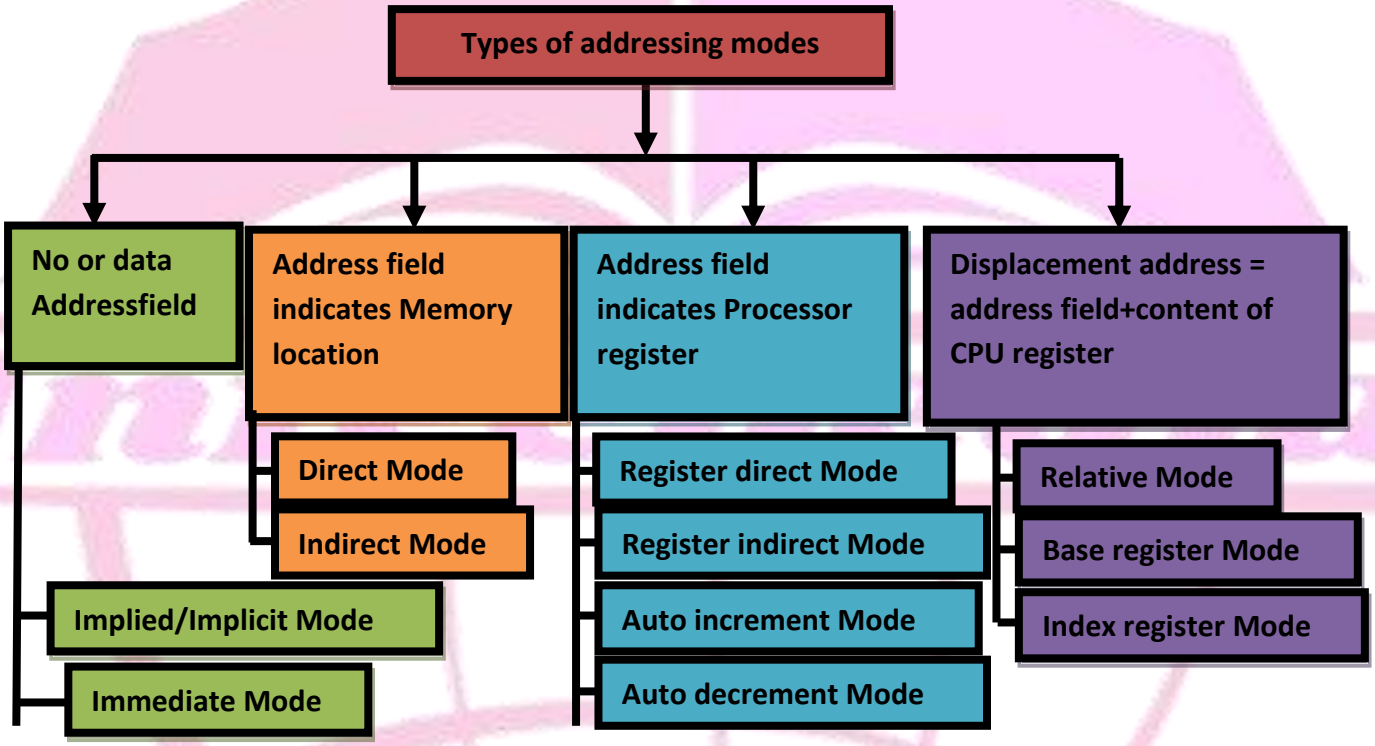
EVALUATE C=A*B (using Zero address instructions)
Postfix Notation: AB*
PUSH A // SP <- A
PUSH B // SP <- B
MUL // SP-> (A*B)
POP C // M[C] <- SP

Important Points

- In instruction format instruction length is always a challenge. The longer the instruction, the longer will be the time taken to fetch the instruction.
- The number of bits of address is directly proportional to the memory range. i.e., if the larger range is required, larger number of address bits is required.
- If a system supports the virtual memory, then the memory range that needs to be addressed by the instruction will be larger than the physical memory.
  Instruction length should be equal to or the multiple of data bus length.

Instruction Addressing Mode

- Mode field indicates the location of the operand of address field.
- The addressing mode specifies a rule for interpreting or modifying the address field of the instruction before the operand is actually executed.
- Indicating the location of an operand in an instruction in different ways are called as addressing mode.
- Addressing mode is used by processors to calculate the effective memory address or operand location for data operations.

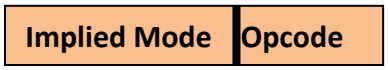


Types of Addressing modes

1. Implied Mode	8. Auto increment Mode
2. Immediate Mode	9. Relative mode
3. Direct Address Mode	10. Base register mode
4. Indirect Address Mode	11. Indexed Mode
5. Register direct Mode	
6. Register Indirect Mode	
7. Auto decrement Mode	

1. Implied/Implicit Addressing Mode

In this mode operands are stored implicitly in the definition of instruction.



Example:-

- **CMA:-** In the instruction "Complement Accumulator", the operand in AC in the definition of instruction.
- **CLA:-** Clear Accumulator
- All register reference instructions that use an accumulator are implied-mode instructions (one-address instruction format).

  In a stack organized system, Zero Address Instructions are implied mode instructions.

Here operands are always implied to be present on the top of the stack.

2. Immediate Mode

The operand is specified in the instruction explicitly.

Instead of address of location, a data value is present in the instruction address field.

It is represented as '#'.

Immediate Mode	Opcode	operands

Example:-

ADD #15 // AC <- AC+15

It will increment the value stored in the accumulator by 15.

MOV R #20 //R <- 20

It initializes register R to a constant value 20.

Advantages:

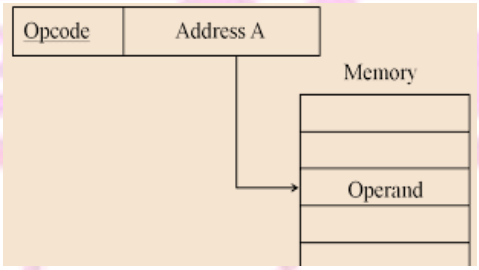
It is fast mode

Limitation:-

The range of constants is restricted by the size of the address field.

3. Direct Addressing Mode

- It is also called absolute addressing mode(symbol used [])
- In this mode, the address field of the instruction specifies the direct memory address of the operand.
- Here the effective address is equal to the address field. Only one memory reference operation is required to access the data.
- **Example:-**
  ADD X // AC <- AC + m[X]
  MUL A,B // m[A] <- m[A] + m[B]

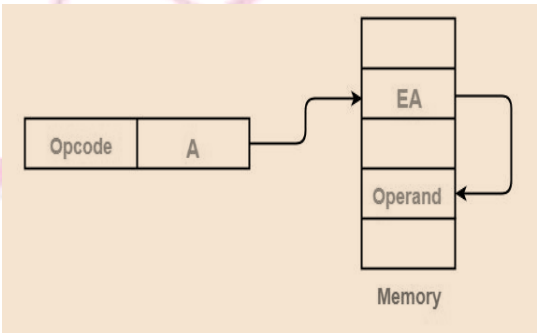


4. Indirect Addressing Mode

- In this mode address field of instruction contains the address of the address of the operand (effective address).
- Here two references are required:
  - 1st reference to get effective address.
  - 2nd reference to access the data.

Example:-

ADD (X) or ADD @X //AC ← AC + m[m[X]]



Advantages:-

Largest address space is available

Disadvantages:-

Slower than direct due to multiple memory reference to find data.

5. Register Direct Addressing Mode

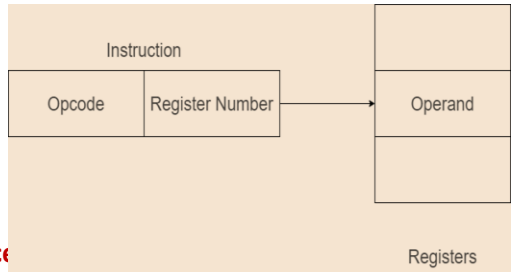
The operand is stored in a register set. The address field of the instruction refers to a CPU register that contains the operand.

No reference to memory is required to fetch the operand.

Example:-

ADD R //AC <- AC + R

MOV R1,R2 //R1 <- R2



Advantages:-

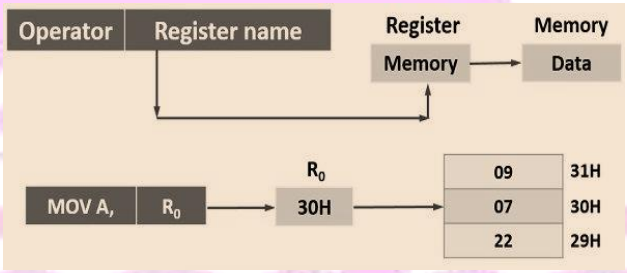
- Insruction length is short due to limited number of registers.
- Fast execution due to no memory reference.

Disadvantages:-

- Limited address space
- If we use multiple register it complex the instructions and also not cost effective.

6. Register Indirect Addressing Mode

- The address field of the instruction refers to a CPU register that contains the effective address of the operand. The operand is stored in the memory.
- Only one reference to memory is required to fetch the operand.
- Register indicates to the effective address of operand.



Example:-

ADD (R) // AC ← AC + m[R]

MUL (R1), R2 // m[R1] <- m[R1] + R2

Advantages:-

Faster than indirect mode because one less memory reference.

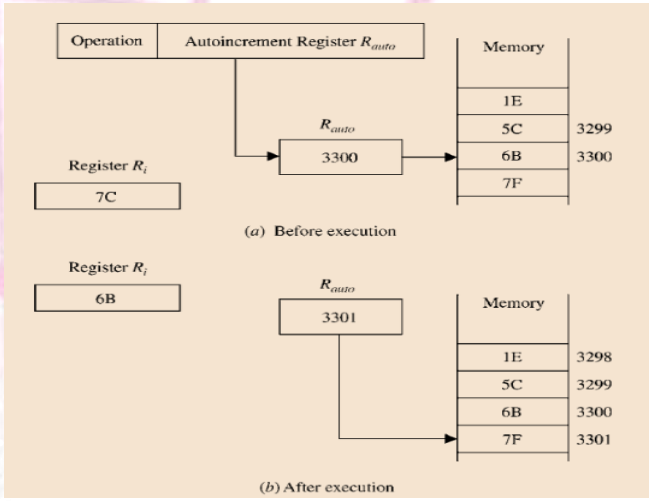
Large address space 2^n

Disadvantages:-

Slower than register direct mode.

7. Auto-increment Addressing mode

- It is also called auto indexed increment mode.
- In this mode the instruction specifies a register which points to a memory address that contains the operand.
- It is same as to register indirect mode however after accessing the operand, the contents of the register are automatically incremented to point to the next consecutive memory location (R1)+ .
- Here one register reference, one memory reference and one ALU operation is required to access the data.



Example:

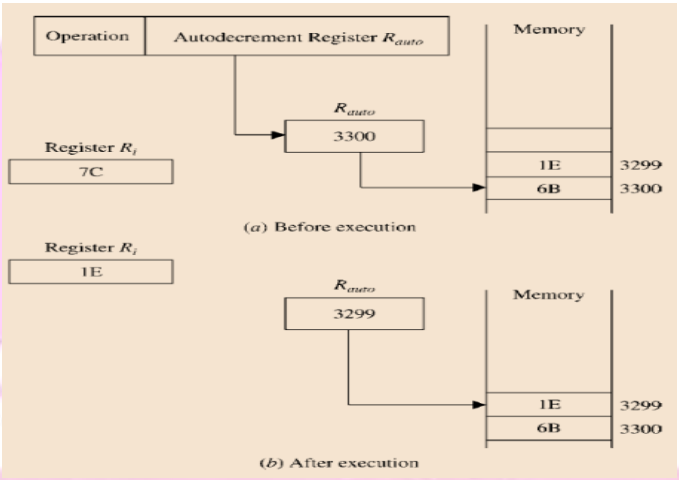
1. ADD R1, (R2)+ // R1 <- R1+M[R2], R2 <- R2+1
2. ADD R // AC <- AC + m[R], R <- R+1

8. Auto-decrement Addressing mode

- It is also called auto indexed decrement mode.
- As register indirect mode effective address of the operand is the contents of a register specified in the instruction.
- It is same as Auto increment but before accessing the operand, the contents of this register are automatically decremented to point to the previous consecutive memory location. EA= –(R1)

Example:

1. ADD R1, -(R2) //R2 <-R2-1, R1 <- R1+ M[R2]

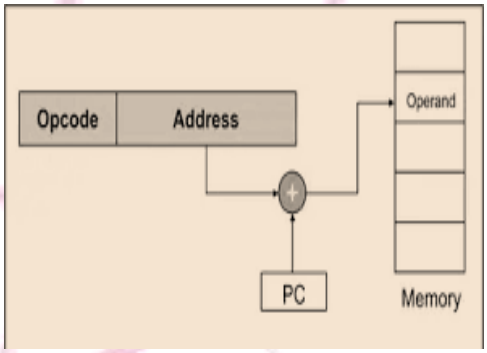


9. Relative Addressing Mode

- It is also called displacement addressing mode.
- In this mode effective address of the operand is obtained by adding the content of program counter with the address part of the instruction.
- Effective Address = Address part of instruction + address of program counter(PC).
- Its symbol $ or instruction address(PC)
- Address location is relative to PC.
- It is useful for indicating destination addresses in branch-type instructions.
- It is useful in relocation of programs in memory in runtime.

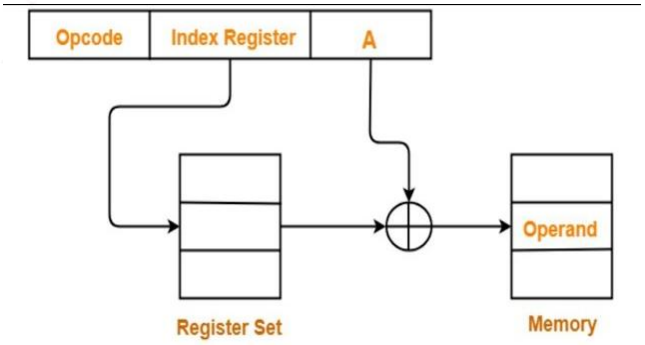
Example:

LOAD $A or LOAD A(PC) // AC <- M[PC + A]



10. Indexed Addressing Mode

- In this mode, the address of the operand is obtained by adding the contents of the address field and the index register.
- Effective Address = Address part of instruction + Content of XR
  EA = A + (XR) or EA = A + (Ri)
- Index register used here is either a special CPU register (XR) provided for this purpose or commonly, it may be anyone of a set of general purpose registers (Ri) in the processor; that contain an index value.
- Index register contains an index number that is relative to the address part of the instruction.
- It is useful for accessing Arrays (beginning address of data array in address part & index number in index register)



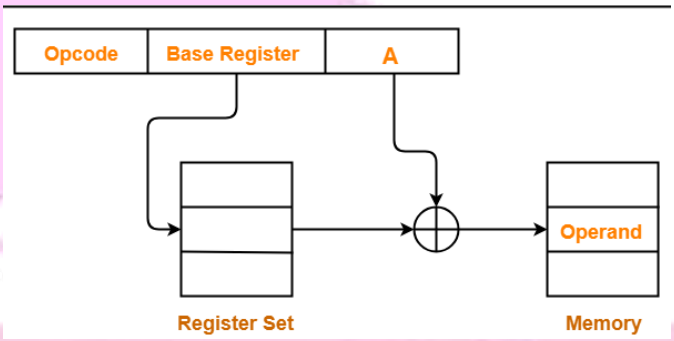
Example:

ADD R1, 10(R2) // R1 <- R1 + M[10 + R2]

ADD R1, 201(XR) // R1 <- R1 + M[201 + XR]

11. Base Register Addressing Mode

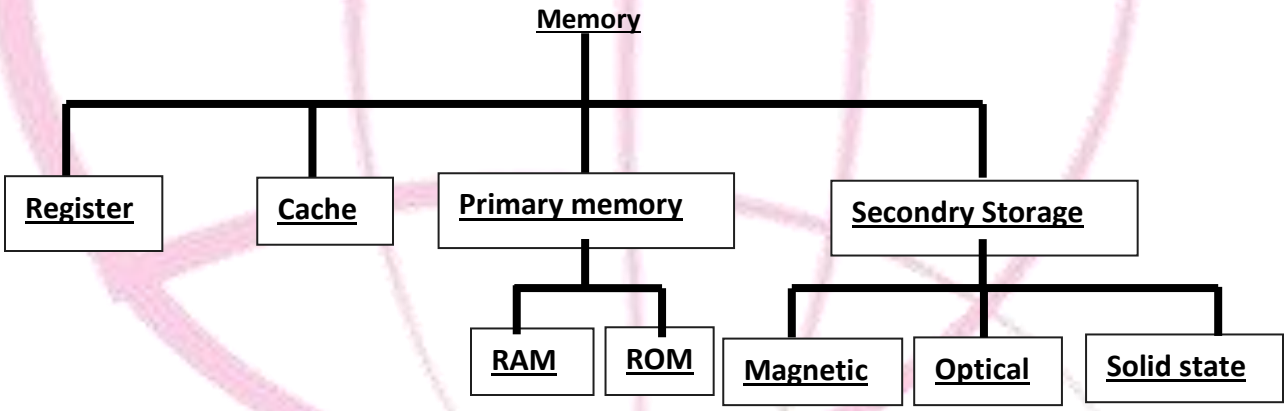
- It is same as indexed addressing mode the only difference is register.Here instead of indexed register we use base register.
- The content of address field of the instruction is added to the Base Register to obtain the effective address of operand.
- Effective Address = Content of BR + Address field of instruction
- EA = (Ri) + A
- Ri is base register that holds base address
- A holds a displacement relative to this base address.
- It is useful in relocation of programs in memory in runtime.
- It is best suitable to write position independent codes.



Example:

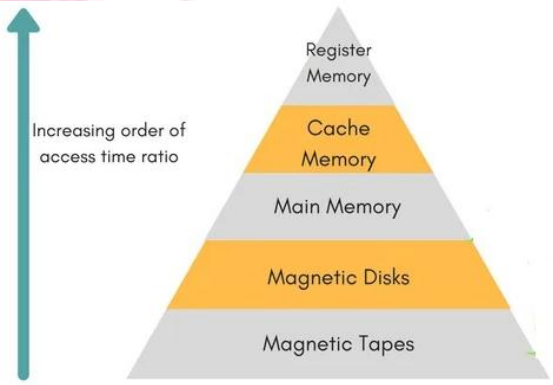
LOAD A(R1) // R1 <- M[R1 + A]

ADD R1, 201(R2) // R1 <- R1 + M[R2 + 201]



From up to down in the memory hierarchy-

- Cost decreases
- Capacity increases
- Access time increases
- Frequency of accessing memory decreases
- Data transfer rate decreases



Memory access Methods

- **Random Access:-** Where each memory location has a unique address. Using this unique address any memory location can be reached in the same amount of time in any order.
- **Sequential Access:-** This method allows memory access in a sequence or in order.
- **Direct Access:-** In this mode, information is stored in tracks, with each track having a separate read/write head.

Memory Organization

Simultaneous Access Memory Organization

Hierarchical Access Memory Organization

1. Simultaneous Access Memory Organization

- In this memory organization, All the levels of memory are directly connected to the CPU.
- Whenever CPU requires any word, it starts searching for it in all the levels simultaneously.

Hierarchical Access Memory Organization

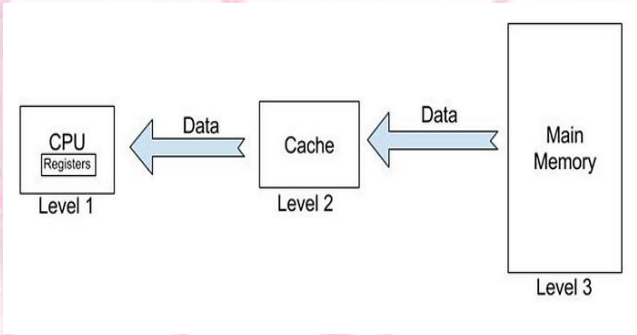
In this memory organization, memory levels are organized as-

- Level-1 is directly connected to the CPU.
- Level-2 is directly connected to level-1.
- Level-3 is directly connected to level-2 and so on.

Whenever CPU requires any word,

- It first searches for the word in level-1.
- If the required word is not found in level-1, it searches for the word in level-2.

If the required word is not found in level-2, it searches for the word in level-3 and so on.



Random Access Memory

1. SRAM (Static RAM)
2. DRAM (Dynamic RAM)
3. Non volatile RAM (Flash memory)

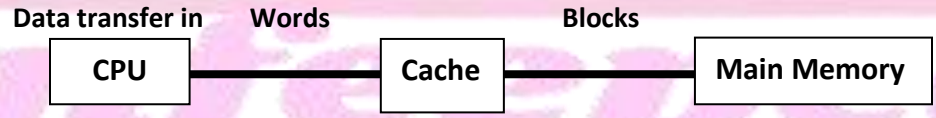
Associative Memory

- Associative memories are also commonly known as content-addressable memories (CAMs).
- In associative memory, data is stored together with additional tags or metadata that describe its content. When a search is performed, the associative memory compares the search query with the tags of all stored data, and retrieves the data that matches the query.
- It reduces the search time by using content
- It is accessed simultaneously or parallel.
- When a word is written in an associative memory, no address is given.
- This memory is capable to find an empty location to store the word.
- To read a word from associative memory the content of the word or part of the word is specified.
- The memory locates all the words which matched the specified content and marks them for reading.

- It is more expensive than RAM.
- Each cell must have storage capability and logical circuits for matching its content with external argument.
- It is used in memory allocation format.
- In the network routing tables to quickly find the path to a destination network.
- In the database management systems.
- in the image processing applications to search for specific features or patterns within an image and artificial intelligence applications such as expert systems and pattern recognition.
- It is used in page tables used by the virtual memory and used in neural networks.

Cache memory

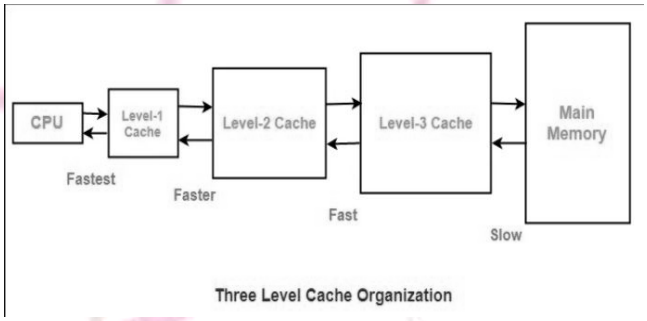
- Cache is the fastest memory.
- Currently running or ready to run process is placed in cache for fast execution.
- CPU first access cache memory if words needed.
- If the word is found in the cache, it is read from the cache.
- If the word required by CPU is not found in the cache, the main memory is accessed to read the blocks.
- Data transfer between CPU and cache is called words and between cache and main memory is called blocks.



Multilevel Cache Organization

Cache memories of different sizes are organized at multiple levels to increase the processing speed. The smaller the size of cache, the faster its speed.

- L1 cache has low storage capacity but is usually the fastest memory in any computer, up to 100 times faster than RAM. Each processor core has its own L1 cache, usually around 64KB.
- L2 cache may be several times larger than L1 but is only about 25 times as fast as RAM. Like L1, each processor core has its own L2 cache. Each is commonly 256-512KB, sometimes as high as 1MB.
- L3 cache has the largest storage capacity, often 32MB or more, but might only be twice as fast as the system memory. L3 cache is commonly embedded in the CPU, but separate from the cores.



Three Level Cache Organization

Locality of Reference

- Locality of reference is the tendency of the computer program where a program repeatedly accesses a specific set of memory locations over a short period of time.
- The term locality of reference refers to the fact that there is usually some predictability in the sequence of memory addresses accessed by a program during its execution.

There are two types of locality of reference:-

1. Temporal:- It means that a recently executed instruction is likely to be executed again very soon.

2. Spatial:- It means that instructions in close proximity to a recently executed instruction are also likely to be executed soon.

Cache Performance

- When the CPU refers to memory and finds the word in cache, it is said cache hit, Otherwise, it is a cache miss
- The performance of cache memory is measured in terms of hit ratio.

Hit ratio = hits(h) / (hit+miss) = No. of hits/total no. of CPU references

Miss ratio= miss(1-h) /(hit+miss)=No. of miss/ total no. of CPU references

Cache access time(Cache hit time)

When the word required by CPU is available in cache, the time required to access the word from the cache is called cache access time.

Cache miss penalty

When the word required by CPU is not available in cache, the main memory is referred for that block. The time required to fetch that block from main memory is called cache miss penalty.

Cache miss penalty= cache access time + main memory access time

Average access time of CPU

Hit ratio * cache access time + (1- hit ratio) * miss time penalty

= $H*T_c + (1-H) * T_m$

Cache Mapping Techniques

- As we know main memory is divided in equal size of blocks or frames.
- Cache memory is divided in lines having same size as blocks.
- In case of cache miss CPU copy a block from main memory to the cache memory lines, this technique is called cache mapping.
- There are three different types of mapping as follows:
  1. **Direct Mapping**
  2. **Associative Mapping**
  3. **K-way Set-Associative Mapping**

Direct mapping

- It is the simplest mapping and uses RAM to map with cache memory.
- A particular block of the main memory is mapped only in a certain line of the cache.
- Suppose cache memory is divided in k number of lines and main memory has n blocks then jth number block can be mapped in the cache as follows:
  Cache line number = (Address of the Main Memory Block) Modulo (Total number of lines in Cache)

  L = j mod k

Example:-

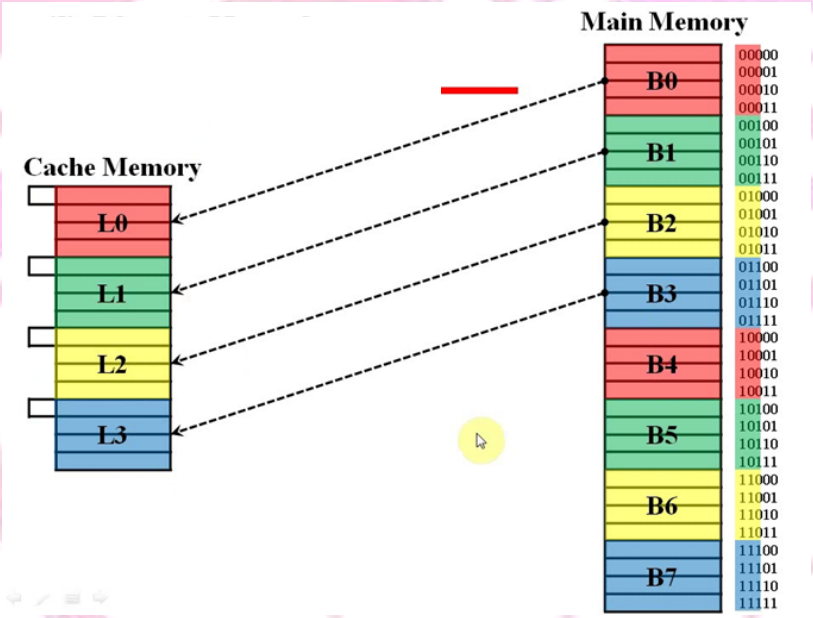
- Suppose block j of the main memory is mapped to cache with k number of lines (J mod k).
- Consider a main memory of 256 words and divided into 64 blocks.
- Number of words in each block =256/64 = 4 words/block
- Consider a cache memory of 32 words.
- As we know the size of block and line is equal. Hence the size of line is 4 words.
- Total number of lines = 32/4 = 8

L0	B0, B8,
L1	B1,
L2	B2
L3	B3
L4	B4
L5	B5
L6	B6
L7	B7

Cache memory

B0	W0	W1	W2	W3
B1	W4	W5	W6	W7
.	.	.	.	.
.	.	.	.	.
.	.	.	.	.
.	.	.	.	.
B62	W248	W249	W250	W251
B63	W252	W253	W254	W255

Main Memory



- Main memory generates physical address to represent 256 words.
  Therefore address will be of 8 bits.

6 bit	2 bit
Block No.	Block offset

- To access the data from cache CPU generate a logical address of 8 bits.

8 bit	
Tag	Index
3 bit	5 bit

- The size of cache is 32 words.
- Here to access cache the block number is further divided in tag and line number. Therefore

3 bit	3 bit	2 bit
Tag	Line number	Line offset

- The main drawback is that it is not flexible.
- There is contention problem even when cache is not full. We cannot store data if the corresponding line is not empty.
- Conflict miss increases.
- To overcome this problem we use associative mapping.
- No page replacement algorithm needed.

Fully Associative mapping

- Any block can go anywhere in cache. No conflict miss occur.
- Consider a main memory of 256 words and divided into 64 blocks.
- Number of words in each block =256/64 = 4 words/block
- Consider a cache memory of 32 words. As we know the size of block and line is equal. Hence the size of line is 4 word.
- Total number of lines = 32/4 = 8

L0	**B0,B1……., B63**
L1	**B0,B1……., B63**
L2	**B0,B1……., B63**
L3	**B0,B1……., B63**
L4	**B0,B1……., B63**
L5	**B0,B1……., B63**
L6	**B0,B1……., B63**
L7	**B0,B1……., B63**

Cache memory

B0	**W0**	**W1**	**W2**	**W3**
B1	**W4**	**W5**	**W6**	**W7**
B2	**W8**	**W9**	**W10**	**W11**
B3	**W12**	**W13**	**W14**	**W15**
.	**.**	**.**	**.**	**.**
.	**.**	**.**	**.**	**.**
.	**.**	**.**	**.**	**.**
.	**.**	**.**	**.**	**.**
B61	**W244**	**W245**	**W246**	**W247**
B62	**W248**	**W249**	**W250**	**W251**
B63	**W252**	**W253**	**W254**	**W255**

Main Memory

- Main memory generates physical address to represent 256 words.
  Therefore address will be of 8 bits.

6 bit	**2 bit**
Block No.	**Block offset**

- To access the data from cache CPU generate a logical address of 8 bits.

8 bit	
Tag	Index
6 bit	2 bit

- The size of cache is 32 words.
- Here to access cache we use tag and block offset(line offset).

Therefore:

6 bit	**2 bit**
Tag	**Line offset**

Advantage:-

- Any empty block in cache can be used,
- It is flexible it must check all tags to check for a hit

Disadvantage:-

- Expensive (parallel algorithm has been developed to speed up the process)
- Page replacement algorithm is needed.

K- way Set Associative Mapping

- Comprise between direct mapping and associative mapping
- Block in main memory maps to a set of lines in cache – direct mapping
- Cache lines are grouped into sets where each set contains k number of lines.
- Can map to any block within the set
- 2-way set associative mapping
- Consider a main memory of 256 words and divided into 64 blocks.
- Number of words in each block =256/64 = 4 words/block
- Consider a cache memory of 32 words. As we know the size of block and line is equal. Hence the size of line is 4 word.
- Total number of lines = 32/4 = 8

L0	B0, B4, B8.....	S0
L1	B0, B4, B8.....	
L2	B1	
L3	B1	S1
L4	B2,	
L5	B2,	S2
L6	B3, B7	
L7	B3, B7	S3

Cache memory

B0	W0	W1	W2	W3
B1	W4	W5	W6	W7
B62	W248	W249	W250	W251
B63	W252	W253	W254	W255

Main Memory

Use direct mapping:- block number mod number of set (n mod s)

Use associative mapping:- place block anywhere inside the set lines.

- Main memory generates physical address to represent 256 words.
  Therefore address will be of 8 bits.

6 bit	2 bit
Block No.	Block offset

- To access the data from cache CPU generate a logical address of 8 bits.

8 bit	
Tag	Index
6 bit	2 bit

- The size of cache is 32 words.
- Here to access cache we use tag and block offset(line offset).
  Therefore

4 bit	2 bit	2 bit
Tag	Set number	Line offset

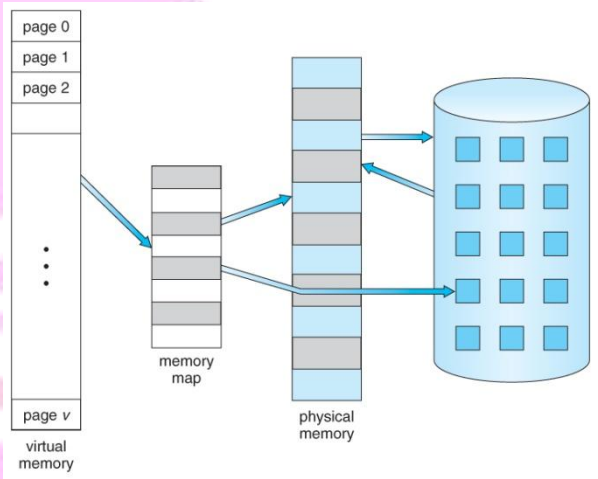
Note:

If K=1, it becomes direct mapping

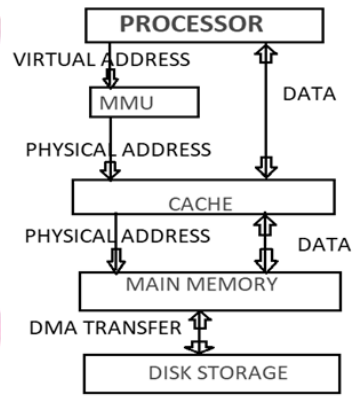
If k= no. of blocks, it become associative mapping.

Virtual Memory

- Virtual memory is a memory management technique that allows the illusion of a larger memory space than physically available.
- It separates logical memory from physical memory, assigning each program its own virtual address space.
- Page tables are used to map virtual addresses to physical addresses, with memory divided into fixed-size blocks called pages.
- When a program accesses a page not in physical memory, a page fault occurs, triggering the retrieval of the required page from secondary storage.
- Demand paging brings pages into memory only when needed, optimizing memory utilization.
- Swapping is a memory management technique where entire processes are moved between RAM and disk.
- Processes are swapped out from RAM to free up memory and stored in the swap space on disk.



- When needed, swapped-out processes are swapped back into RAM from the swap space, called swap in.
- Swap space acts as a backing store, providing additional storage when RAM is insufficient.
- Page replacement algorithms like FIFO and LRU decide which pages to evict from memory when it's full.
- Drawbacks include overhead from page faults and complexity.



Example:- The virtual address space is 4 GB and page size is 128 KB. Given that the memory address space is of 512 MB. What is the size of frame and bits required to represent virtual addresses and physical addresses.

Solution:

1. **Virtual Address Space (VAS):**
   - VAS = 4 GB = 4×2^{30} bytes = 2^{32} bytes.
   - Page size = 128 KB = 128×2^{10} bytes = 2^{17} bytes.
   - Number of pages = VAS/ Page Size=2^{32} /2^{17}= 2^{15}
   - Virtual address bits = 32 bits (since 2^{32} bytes).
2. **Physical Address Space (PAS):**
   - PAS = 512 MB = 512×2^{20} bytes = 2^{29} bytes.

- Frame size = Page size = 128 KB = 2^{17} bytes.
- Number of frames = PAS /Frame Size=$2^{29}/2^{17} = 2^{12}$
- Physical address bits = 29 bits (since 2^{29} bytes).

Thus:

- Frame size = 128 KB.
- Virtual address total bits = 32 bits.
- Physical address total bits = 29 bits.

Page Replacement algorithm

Page replacement algorithms are used in virtual memory management to decide which page to evict from physical memory when it's full.

1. FIFO (First In, First Out):

- Evicts the oldest page in memory.
- Easy to implement using a queue.
- May suffer from the "Belady's Anomaly" where increasing the number of frames can increase page faults.

Example: Suppose we have a system with 3 frames (physical memory) and a sequence of page accesses:
1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5.

		3	3	3	2	2	2	2	2	4	4
	2	2	2	1	1	1	1	1	3	3	3
1	1	1	4	4	4	5	5	5	5	5	5
F	F	F	F	F	F	F	H	H	F	F	H

If the system has 3 frames, the given reference string the using FIFO page replacement algorithm yields a total of 9 page faults.

2. LRU (Least Recently Used):-

- Evicts the least recently accessed page in memory.
- Requires maintaining a record of when each page was last accessed.
- Offers good performance by removing pages least likely to be used in the near future.

		3	3	3	2	2	2	2	2	2	5
	2	2	2	1	1	1	1	1	1	4	4
1	1	1	4	4	4	5	5	5	3	3	3
F	F	F	F	F	F	F	H	H	F	F	F

Example: Suppose we have a system with 3 frames (physical memory) and a sequence of page accesses:
1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5.

If the system has 3 frames, the given reference string the using LRU page replacement algorithm yields a total of 10 page faults.

3. LFU (Least Frequently Used)

- Evicts the page with the fewest accesses.
- Requires maintaining a count of how often each page is accessed.
- Can suffer from "frequency skew" if a page is accessed frequently initially but then becomes less used.

Example: Suppose we have a system with 3 frames (physical memory) and a sequence of page accesses: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5.

		3	3	3	2	2	2	2	2	2	2
	2	2	2	1	1	1	1	1	1	1	1
1	1	1	4	4	4	5	5	5	3	4	5
F	F	F	F	F	F	F	H	H	F	F	F

If the system has 3 frames, the given reference string the using LFU page replacement algorithm yields a total of 10 page faults.

4. Optimal Page Replacement

- Evicts the page that will not be used for the longest period in the future.
- Provides the lowest possible page fault rate but is impractical to implement as it requires knowledge of future memory accesses.

Example: Suppose we have a system with 3 frames (physical memory) and a sequence of page accesses: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5.

		3	4	4	4	5	5	5	5	5	5
	2	2	2	2	2	2	2	2	2	4	4
1	1	1	1	1	1	1	1	1	3	3	3
F	F	F	F	H	H	F	H	H	F	F	H

If the system has 3 frames, the given reference string the using Optimal page replacement algorithm yields a total of 7 page faults.

IO Organization

Input/Output Subsystem:-

- The I/O subsystem of a computer provides an efficient mode of communication between the central system and the outside environment.
- It handles all the input output operations of the computer system.

Peripheral Devices:

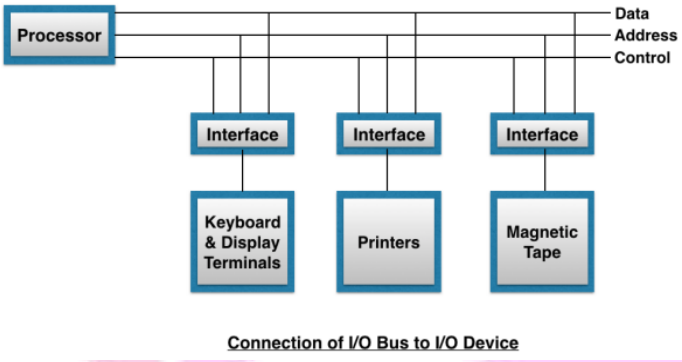
- Input or output devices that are connected to computer are called peripheral devices.
- These devices are designed to read information into or out of the memory unit upon command from the CPU and are considered to be the part of computer system.
- These devices are also called peripherals.
- For example: Keyboards, Mouse,Scanner and printers are common peripheral devices.

Interfaces

Interface is a shared boundary btween two separate components of the computer system which can be used to attach two or more components to the system for communication purposes.

There are two types of interface:

1. CPU Inteface
2. I/O Interface



Connection of I/O Bus to I/O Device

Input-Output Interface

1. Peripherals connected to a computer need special communication links for interfacing with CPU.
2. In computer system, there are special hardware components between the CPU and peripherals to control or manage the input-output transfers.
3. These components are called input-output interface units because they provide communication links between processor bus and peripherals.
4. They provide a method for transferring information between internal system and input-output devices.
5. The Input/output Interface is required because there are exists many differences between the central computer and each peripheral while transferring information.

Some major differences are:

a) Peripherals are electromechanical and electromagnetic devices and their manner of operation is different from the operation of CPU and memory, which are electronic device.Therefore, a conversion of signal values may be required.

b) The data transfer rate of peripherals is usually slower than the transfer rate of CPU, and consequently a synchronisation mechanism is needed.

c) Data codes and formats in peripherals differ from the word format in the CPU and Memory.

d) The operating modes of peripherals are differ from each other and each must be controlled so as not to disturb the operation of other peripherals connected to CPU.

- There are three ways that computer buses can be used to communicate with memory and I/O:

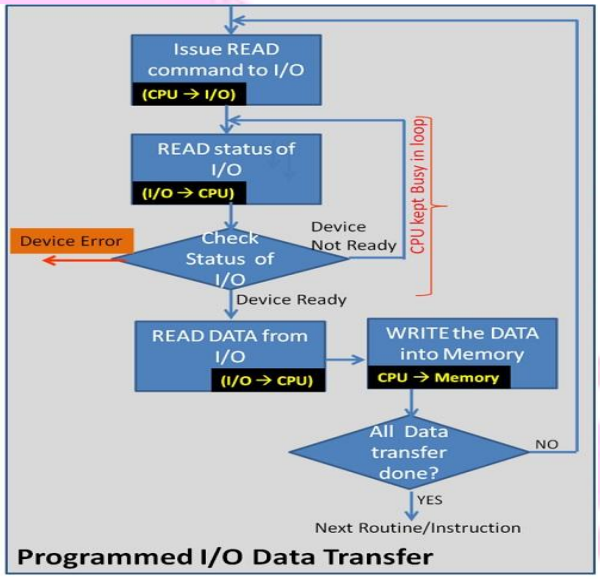
1. Use two separate buses, one for memory and the other for I/O.
2. Use one common bus for both memory and I/O but have separate control lines for each.
3. Use one common bus for memory and I/O with common control lines.

Modes of I/O Data Transfer

Data transfer between the central unit and I/O devices can be handled in generally three types of modes which are given below:

1. Programmed I/O

1. In this mode, the CPU is responsible for all data transfers between the I/O device and memory.
2. The CPU continuously checks the status of the I/O device to see if it is ready for data transfer.
3. This process is called polling.
4. The CPU waits in a loop until the I/O device signals that it is ready.
5. Once ready, the CPU executes a data transfer instruction to read from or write to the I/O device.



Advantages:

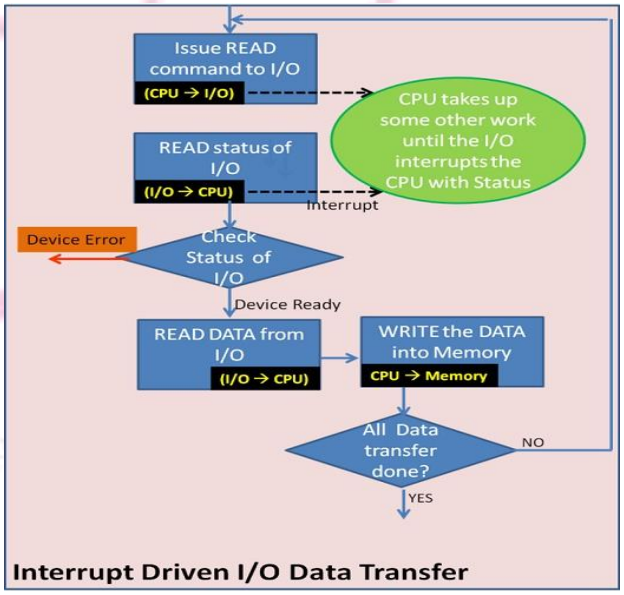
- Simple to implement.
- Direct control over I/O devices.

Disadvantages:

- CPU time is wasted in constantly checking the device status (polling).
- Inefficient as the CPU cannot perform other tasks during this waiting period.

2. Interrupt-Driven I/O

1. In this mode, the CPU is interrupted by the I/O device when the device is ready for data transfer.
2. The CPU issues a command to the I/O device and continues with other tasks.
3. When the I/O device is ready (e.g., it has finished reading or writing), it sends an interrupt signal to the CPU.
4. The CPU stops its current task, saves its state, and executes an interrupt service routine (ISR) to handle the I/O transfer.
5. After the I/O operation is complete, the CPU resumes its previous task.



Advantages:

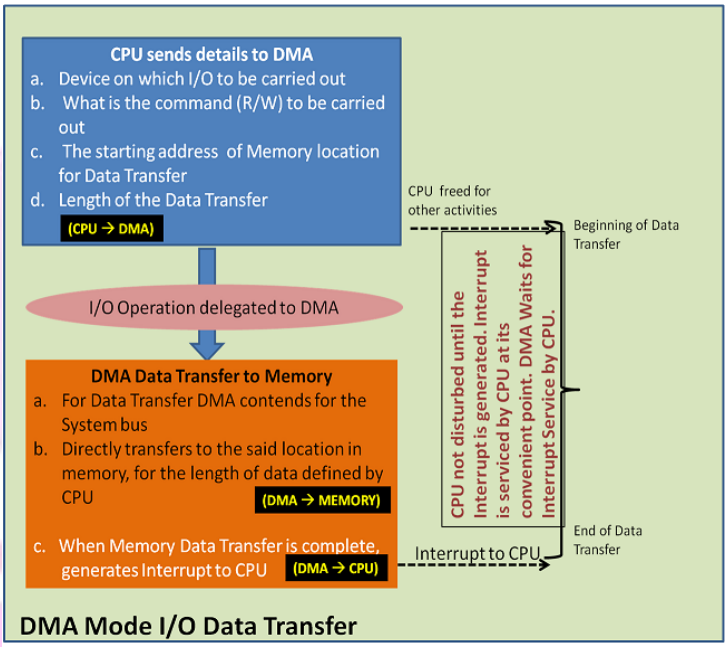
- More efficient than programmed I/O as the CPU can perform other tasks while waiting for the I/O operation.
- Reduces CPU idle time.

Disadvantages:

- More complex to implement compared to programmed I/O.
- May introduce interrupt latency if multiple interrupts occur.

3. Direct Memory Access (DMA)

1. In this mode, a separate controller (DMA controller) is used to handle data transfers directly between memory and I/O devices without involving the CPU for each byte or word of data transfer.
2. The CPU initializes the DMA controller by providing the memory address, I/O address, and the number of bytes to transfer.
3. The DMA controller takes over the bus and manages the data transfer directly between the memory and the I/O device.
4. After the transfer is complete, the DMA controller sends an interrupt signal to the CPU to inform it of the completion.



DMA Mode I/O Data Transfer

Advantages:

- Very efficient for high-speed data transfer between memory and I/O devices.
- Frees up the CPU to perform other tasks.

Disadvantages:

- Requires a separate DMA controller, adding hardware complexity.
- Possible bus contention when both CPU and DMA controller try to access the memory simultaneously.

Memory-Mapped I/O

1. In this mode, I/O devices are treated as if they are part of the system memory, and specific memory addresses are assigned to each I/O device.
2. The CPU uses regular memory instructions (like LOAD and STORE) to read or write data to the I/O device addresses.
3. No special I/O instructions are needed.
4. This allows I/O devices to be controlled and accessed just like memory.
5. A single set of read/write control lines (no distinction between memory and I/O transfer)
6. Memory and I/O addresses share the common address space which reduces memory address range available
7. No specific input or output instruction so the same memory reference instructions can be used for I/O transfers o Considerable flexibility in handling I/O operations

Advantages:

- Simplifies the I/O access mechanism as the same instructions can be used for both memory and I/O.
- Easier to program and optimize.

Disadvantages:

- Consumes a part of the memory address space for I/O operations.
- May lead to address conflicts.
- Applications:
- Common in microcontrollers and simpler computer architectures.

I/O Mapped I/O (Isolated I/O)

1. In this mode, a separate address space is reserved for I/O devices, and special instructions are used to access I/O devices.
2. Uses specific I/O instructions like IN and OUT to access the I/O device.
3. I/O devices have their own dedicated address space, separate from the system memory.
4. Separate I/O read/write control lines in addition to memory read/write control lines

5. Separate (isolated) memory and I/O address spaces
6. Distinct input and output instructions

Advantages:

- No conflict between memory and I/O addresses.
- Efficient use of memory address space.

Disadvantages:

- Requires additional instructions for I/O operations.
- May require complex programming.

Modes of Transfer

Mode	CPU Involvement	Efficiency	Use Case
Programmed I/O	High (Polling-based)	Low (CPU is idle)	Simple, low-speed devices
Interrupt-Driven I/O	Moderate (Interrupt handling)	Moderate to High	Moderate-speed devices
Direct Memory Access (DMA)	Low (Only initialization)	High	High-speed devices