Computer – CPU (Central Processing Unit)

The features of the Central Processing Unit (CPU) are:

- CPU is considered as the brain of the computer.
- It performs all data processing operations.
- It stores data, intermediate results, and instructions (program).
- It controls the operation of all parts of the computer.

The CPU has following three components.

1. Memory or Storage Unit
2. Control Unit
3. ALU(Arithmetic Logic Unit)

The Memory or Storage Unit

This unit can store instructions, data, and intermediate results. It supplies information to other units of the computer when needed. It is known as internal storage unit or(the main memory or the primary storage or Random Access Memory (RAM)).

Its size affects speed, power, and capability. Primary memory and secondary memory are two types of memories in the computer.

Functions of the memory unit are to:

- Control the transfer of data and instructions among other units of a computer
- Manages and coordinates all the units of the computer.
- Obtains the instructions from the memory, interprets them, and directs the operation of the computer
- Communicates with Input/Output devices for transfer of data or results from storage

The ALU (Arithmetic Logic Unit)

This unit consists of two subsections namely:

- Arithmetic Section
- Logic Section

Arithmetic Section

- Perform operations like addition, subtraction, multiplication, and division.
- All complex operations are done by making repetitive use of the above operations.

Logic Section

Perform operations such as comparing, selecting, matching, and merging of data.

Computer Organization and Architecture

Computer ***Architecture*** refers to those attributes of a system that have a direct impact on the logical execution of a program. Architecture is those attributes visible to the programmer. Examples:

- the instruction set
- the number of bits used to represent various data types of I/O mechanisms
- memory addressing techniques

Computer ***Organization*** refers to the operational units and their interconnections that realize the architectural specifications. Organization is how the features are implemented

Examples are things that are transparent to the programmer:

- Control signals, interfaces, memory technology; interfaces between computer and peripherals o the memory technology being used.

For example, the availability of a multiply instruction is architecture issue. How that multiply is implemented is organization issue. For instance: Is there hardware multiply unit or is it done by repeated addition?

- All Intel x86 family share the same basic architecture
- The IBM System/370 family shares the same basic architecture
- This enables code compatibility; At least backwards
- However, Organization most often differs between different versions.

Computer Structure and Function

- Structure is the way in which components relate to each other
- Function is the operation of individual components as part of the structure
- All computer functions are:
  - **Data processing**: Computer must be able to process data which may take a wide variety of forms and the range of processing.
  - **Data storage**: Computer stores data either temporarily or permanently.
  - **Data movement**: Computer must be able to move data between itself and the outside world.
  - **Control**: There must be a control of the above three functions.

Computer Components

- The Control Unit (CU), Arithmetic and Logic Unit (ALU) constitute the Central Processing Unit (CPU)
- Input/output (I/O module) enable Data and instructions to get into the system and results to get out
- The main memory (RAM), enable temporary storage of code and results.

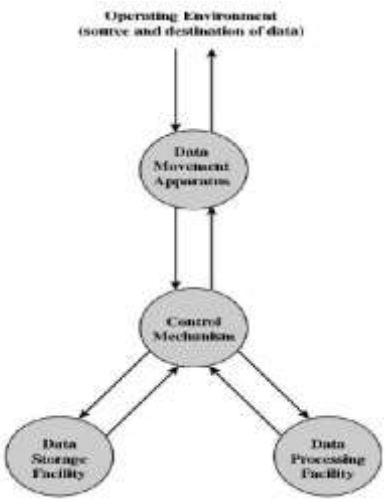


Fig: Data movement operation

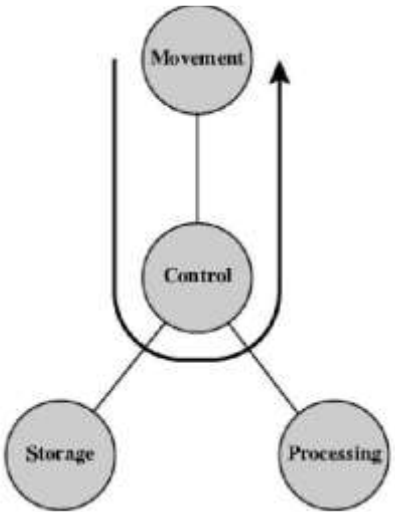


Fig: Storage Operation

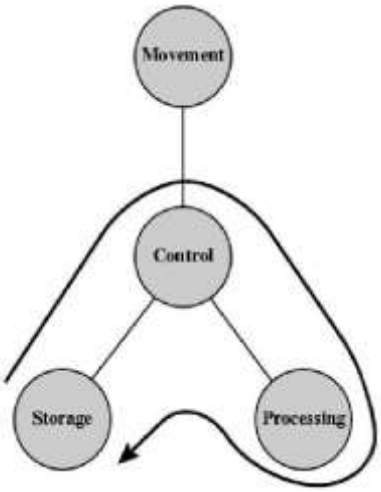


Fig: Processing from / to storage

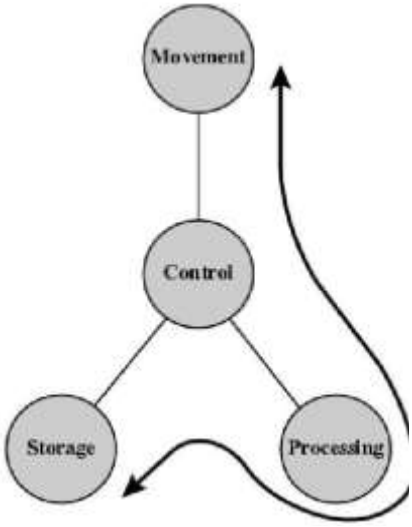


Fig: Processing from storage to i/o

Four main structural components:

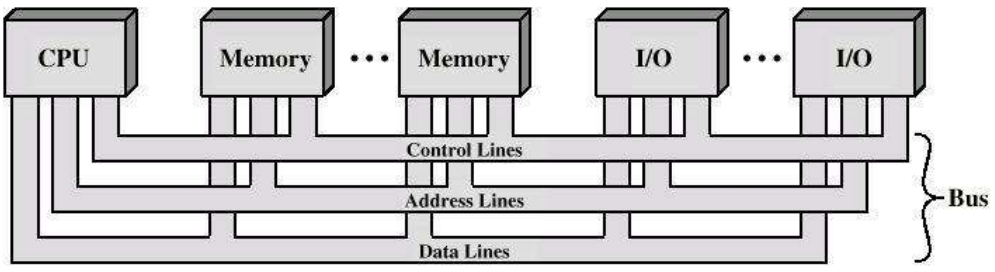
- Central processing unit (CPU)
- Main memory
- I/O
- System interconnections

CPU structural components are:

- Control unit
- Arithmetic and logic unit (ALU)
- Registers
- CPU interconnections

Bus Interconnection

• A bus is a communication pathway connecting two or more devices
• Usually broadcast (all components see signal)
• Often grouped
    - A number of channels in one bus
    - e.g. 32 bit data bus is 32 separate single bit channels
• Power lines may not be shown
• There are a number of possible interconnection systems
    • Single and multiple BUS structures are most common
        • e.g. Control/Address/Data bus (PC)
        • e.g. Unibus (DEC-PDP)
• Lots of devices on one bus leads to:
    o Propagation delays
    o Long data paths mean that co-ordination of bus can adversely affect performance
    o If aggregate data transfer approaches bus capacity
• Most systems use multiple buses to overcome these problems



We therefore have the following buses:

Data Bus

- Carries data
  Note that there is no difference between "data" and "instruction" at this level
- Width is a key determinant of performance; we have:
  8, 16, 32, 64 bit

Address Bus

- Identify the source or destination of data
  - e.g. CPU needs to read an instruction (data) from a given location in memory
- Bus width determines maximum memory capacity of system
  e.g. 8080 has 16 bit address bus giving 64k address space

Control Bus

- Performs: Control and timing information
- Memory read and Memory write
- I/O read and I/O write
- Transfer ACK
- Bus request and Bus grant
- Interrupt request and Interrupt ACK
- Clock
- Reset

CPU Registers

- In *computer architecture*, a processor **register** is a very fast computer memory used to speedup the execution of computer programs by providing quick access to commonly used values, typically, the values in the midst of a calculation at a given point in time.
- These **registers** are at the top of the memory hierarchy, and offer the fastest way for the system to manipulate data.
- In a very simple *microprocessor*, it consists of a single memory location, usually called an ***accumulator.***
- **Registers** are built from fast multi-ported memory cell.
- It drive its data onto an internal bus in a single clock cycle.
- The result of ALU operation is stored here and could be re-used in a subsequent operation or saved into memory.
- Registers are normally measured by the number of bits they can hold, for example, an "8-bit register" or a "32-bit register".
- Registers are now usually implemented as a register file, but they have also been implemented using individual flip-flops, high speed core memory, thin film memory, and other ways in various machines.
- The term 'Register' is often used to refer only to the group of registers that can be directly indexed for input or output of an instruction, as defined by the instruction set.
- More properly, these are called the "*architected registers*".
  - For instance, the x86 instruction set defines a set of eight 32-bit registers, but a CPU that implements the X86 instruction set will contain many more hardware registers than just these eight.

The following are other classes of registers:

(a) **Accumulator**: It is most frequently register used to store data taken from memory. Its number varies from microprocessor to microprocessor.

(b) **General Purpose registers**: are used to store data and intermediate results during program execution. Its contents can be accessed through assembly programming.

(c) **Special purpose Registers**: used by computer system at the time of program execution (not accessible to Users).

Some other types of special purpose registers are given below:

- **Memory Address Register (MAR**): - stores address of data or instructions to be fetched from memory.

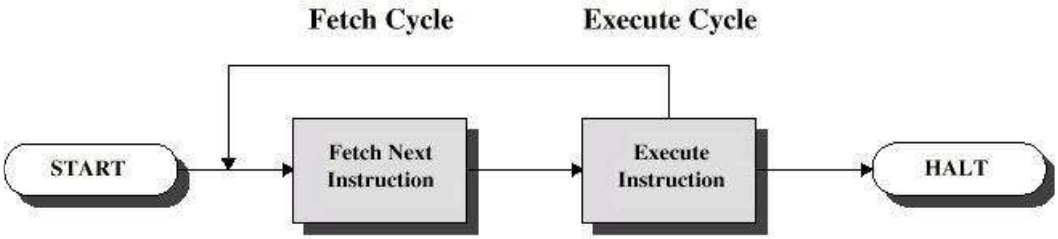
- **Memory Buffer Register (MBR)**: - stores instruction and data received from the memory or sent to the memory.
- **Instruction Register (IR):** store instructions. When one instruction is completed, next instruction is fetched in memory for processing.
- **Program Counter (PC):** - keep count instructions.

Computer Function

The basic function performed by a computer is execution of a program (a set of instructions stored in memory).

• The two steps of Instructions Cycle are:

- Fetch
- Execute

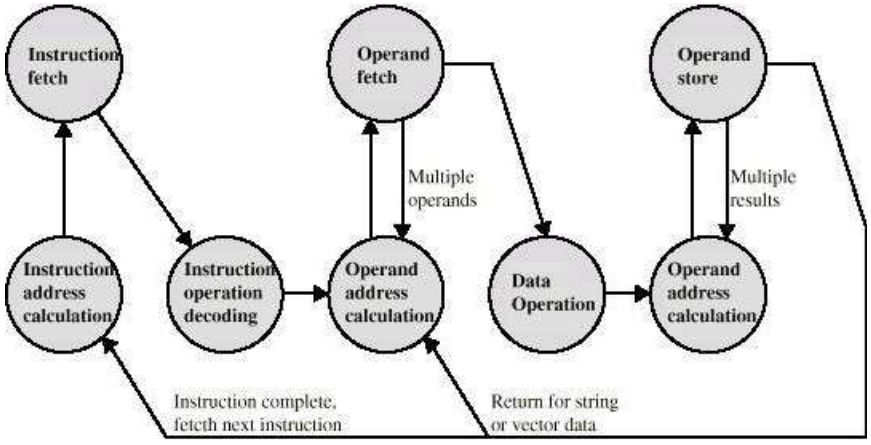


Fetch Cycle:

- Program Counter (PC) holds address of next instruction to fetch,
- Processor fetches instruction from memory location pointed to by PC,
- PC is increment;
  Unless told otherwise,
- The Instruction is loaded into Instruction Register (IR)

Execute Cycle:

- The Processor interprets instruction and performs required actions, such as:
  - data transfer between CPU and main memory
  - Data transfer between CPU and I/O module
  - Arithmetic and logical operations on data
  - Alteration of sequence of operations; e.g. jump (branch)



Fetch Instruction (Cal. Inst. Address) → Decode (Interpreter) → Fetch Operand (Cal. Data Address) → Data Operation (Process) → Store Operand (Memory)

There are two parts in instruction: - **opcode** and **operand**.

In fetch cycle the **opcode** of instruction is brought into CPU.

The operand, at first, is sent to Data Register (DR), then to Instruction Register (IR).

Decoder accesses the opcode, decodes it and type of operation is declared to CPU for execution cycle is started.

Instruction Formats (Representation)

- Computer can perform a specific task, by following specified steps to complete the task.
- A collection of such ordered steps forms a 'program'.
- The ordered steps are called the instructions.
- Computer instructions are stored in central memory locations and are executed sequentially one at a time.
- The Control unit of CPU reads an instruction from a specific address in memory and executes it.
- It then continues by reading the next instruction in sequence and executes it until the completion of the program.

A computer usually has a variety of Instruction Code Formats.

- The control unit within the CPU to interpret each instruction code and provide the necessary control functions needed to process the instruction.
- An n bit instruction has k bits address field and m bits operation code field, can addressed 2k location directly and specify 2m different operation
- The bits of the instruction are divided into groups called fields.
- The most common fields in instruction formats are:
    - An ***Operation code*** field that specifies the operation to be performed.
    - An ***Address*** field that designates a memory address or a processor register.
    - A ***Mode*** field that specifies the way the operand (or the effective address) is determined.

Fig: Instruction format with mode field

- The operation code field (Opcode) of an instruction is a group of bits that define various processor operations such as *add, subtract, complement, shift etc.*
- The bits that define the mode field of an instruction code specify a variety of alternatives for choosing the operands from the given address.
- Operation specified by an instruction is executed on some data stored in the ***processor register*** or in the ***memory location***.
    - Operands residing in memory are specified by their ***memory address***.
    - Operands residing in processor register are specified with a ***register address***.

- An instruction is represented as sequence of bits, for example:

1001	0010	0000	0011	1011	1011	1000	0001
9	2	0	3	B	B	8	1
Opcode		Operand1				Operand2	

The Instruction above is divided into fields:

- Opcode: - *indicates the operation* to be performed, eg., 92 above indicates a copy operation. we need two operands; one ***source*** and other ***destination***

- Operand - represents:
    - *nature of operands* (data or address) above; operand 1 is address and operand 2 is data
    - mode(register or memory), operand 1 is memory, and operand 2 is immediate data

Types of Instruction

- Computers may have instructions of different lengths, containing varying number of addresses.
- The number of address fields in the instruction format of a computer depends on the internal organization of its registers.

Most computers fall into one of 3 types of CPU organizations:

Single accumulator organization:

- All the operations are performed with an accumulator register.
- The instruction format in this type of computer uses one address field.
  For example: ADD X, where X is the address of the operands.

General register organization:

- The instruction format in this type of computer needs three register address fields.
  For example: ADD R1, R2, R3

Stack organization:

The instruction in a stack computer consists of an operation code with no address field.
This operation has the effect of popping the 2 top numbers from the stack, operating the numbers and pushing the sum into the stack.
For example: ADD

The following are the types of instructions:

1. Three address Instruction

In this type, each instruction specifies two operand location and a result location. A temporary location T is used to store some intermediate result so as not to alter any of the operand location.
The three address instruction format requires a very complex design to hold the three address references.
Format: Op X, Y, Z; X ← Y Op Z
Example: ADD X, Y, Z; X ←Y + Z

- Advangage: It results in short programs when evaluating arithmetic expressions.
- Disadvantage: The instructions require too many bits to specify 3 addresses.

2. Two address instruction

This is the most common in commercial computers.
Here each address field can specify either a processor register, or a memory word.
One address must do double duty as both operand and result.
The two address instruction format reduces the space requirement. To avoid altering the value of an operand, a MOV instruction is used to move one of the values to a result or temporary location T, before performing the operation.
Format: Op X, Y; X ← X Op Y
Example: SUB X, Y; X ← X - Y

3. One address Instruction

This was generally used in earlier machine with the implied address been a CPU register known as accumulator.
So the One-address instruction uses an implied accumulator (Ac) register for all data manipulation.
All operations are done between the AC register and a memory operand.
We use LOAD and STORE instruction for transfer to/from memory and Ac register.
Format: Op X; Ac ←Ac Op X
Example: MUL X; Ac ←Ac * X

4. Zero address Instruction

This does not use any address field for the instruction like ADD, SUB, MUL, DIV, etc.
However, PUSH and POP instructions need an address field to specify the operand that communicates with the stack.
The name "Zero" address is given because of the absence of an address field in the computational instruction.
Format: Op; TOS ←TOS Op (TOS – 1)
Example: DIV; TOS ←TOS DIV (TOS – 1)

Let us illustrate the influence of the number of addresses on computer programs, we will evaluate the arithmetic statement X=(A+B) *(C+D) using Zero, one, two, or three address instructions.

1. Three-Address Instructions:

ADD R1, A, B; R1 ←M[A] + M[B]
ADD R2, C, D; R2 ←M[C] + M[D]
MUL X, R1, R2; M[X] ← R1 * R2

It is assumed that the computer has two processor registers R1 and R2. The symbol M[A] denotes the operand at memory address symbolized by A.

2. Two-Address Instructions:

MOV R1, A; R1 ← M[A]
ADD R1, B; R1 ← R1 + M[B]
MOV R2, C; R2 ← M[C]
ADD R2, D; R2 ← R2 + M[D]
MUL R1, R2; R1 ← R1 * R2
MOV X, R1; M[X] ← R1

3. One-Address Instruction:

LOAD A; Ac ←M[A]
ADD B; Ac ←Ac + M[B]
STORE T; M[T] ←Ac
LOAD C; Ac ←M[C]
ADD D; Ac ←Ac + M[D]
MUL T; Ac ←Ac * M[T]
STORE X; M[X] ←Ac

Here, T is the temporary memory location required for storing the intermediate result.

4. Zero-Address Instructions:

PUSH A; TOS ←A
PUSH B; TOS ←B
ADD; TOS ←(A + B)
PUSH C; TOS ←C
PUSH D; TOS ←D
ADD; TOS ←(C + D)
MUL; TOS ←(C + D) * (A + B)
POP X ; M[X] ←TOS

So basically we have the following Instruction Types

Not all instructions require two operands

- 3-address instructions

  Operation Source1, Source2, Destination
  e.g. Add A, B, C ; C = A + B

- 2-address instructions

  Operation Source, Destination
  e.g. Move B, C; C = B
  Add A, C; C = C + A
  Here Source2 is implicitly the destination

- 1-address instructions

  e.g. Load A
  Store C

- 0-address instructions

e.g. Stop

Addressing Modes

Specifies a rule for interpreting or modifying the address field of the instruction before the operand is actually referenced.

Computers use addressing mode techniques for the purpose of accommodating the following purposes:-

- To give programming versatility to the user by providing such facilities as pointers to memory, counters for loop control, indexing of data and various other purposes.
- To reduce the number of bits in the addressing field of the instructions.
  - Some computers use a single binary for operation & Address mode.

- The mode field is used to locate the operand.
- Address field may designate a memory address or a processor register.
- There are 2 modes that need no address field at all (Implied & immediate modes).

Effective address (EA):

- The effective address is defined to be the memory address obtained from the computation dictated by the given addressing mode.
- The effective address is the address of the operand in a computational-type instruction.

The well known addressing modes are:

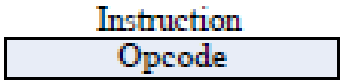
- Implied Addressing Mode.
- Immediate Addressing Mode
- Register Addressing Mode
- Register Indirect Addressing Mode
- Auto-increment or Auto-decrement Addressing Mode
- Direct Addressing Mode
- Indirect Addressing Mode
- Displacement Address Addressing Mode
- Relative Addressing Mode
- Index Addressing Mode
- Stack Addressing Mode

Implied Addressing Mode:

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

For example- CMA - "complement accumulator" is an implied-mode instruction because the operand in the accumulator register is implied in the definition of the instruction.

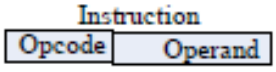
In fact, all register reference instructions that use an accumulator are implied-mode instructions.



Advantage: no memory reference. Disadvantage: limited operand

Immediate Addressing mode:

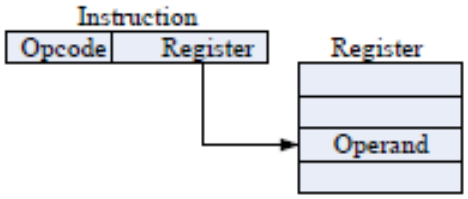
- In this mode the operand is specified in the instruction itself. In other words, an immediate-mode instruction has an operand field rather than an address field.
- The operand field contains the actual operand to be used in conjunction with the operation specified in the instruction.
- These instructions are useful for initializing register to a constant value;
  For example MVI B, 50H



It was mentioned previously that the address field of an instruction may specify either a memory word or a processor register. When the address field specifies a processor register, the instruction is said to be in register-mode.
Advantage: no memory reference. Disadvantage: limited operand

Register direct addressing mode:

- In this mode, the operands are in registers that reside within the CPU.
- The particular register is selected from the register field in the instruction. For example MOV A, B

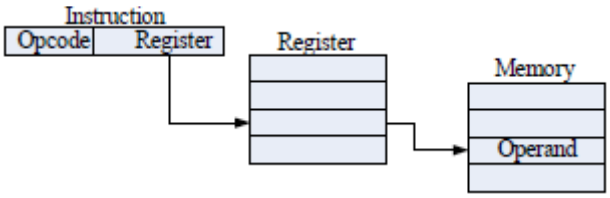


Effective Address (EA) = R
Advantage: no memory reference. Disadvantage: limited address space

Register Indirect Addressing Mode:

- In this mode the instruction specifies a register in the CPU whose contents give the address of the operand in the memory.
- In other words, the selected register contains the address of the operand rather than the operand itself.
- Before using a register indirect mode instruction, the programmer must ensure that the memory address of the operand is placed in the processor register with a previous instruction.

For example LDAX B



Effective Address (EA) = (R)
Advantage: Large address space.
The address field of the instruction uses fewer bits to select a register than would have been required to specify a memory address directly.
Disadvantage: Extra memory reference

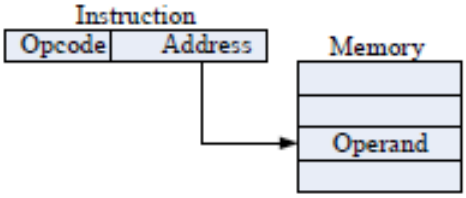
Auto increment or Auto decrement Addressing Mode:

- This is similar to register indirect mode except that the register is incremented or decremented after (or before) its value is used to access memory.
- When the address stored in the registers refers to a table of data in memory, it is necessary to increment or decrement the registers after every access to the table.
- This can be achieved by using the increment or decrement instruction. In some computers it is automatically accessed.
- The address field of an instruction is used by the control to obtain the operands from memory.
- Sometimes the value given in the address field is the address of the operand, but sometimes it is the address from which the address has to be calculated.

Direct Addressing Mode

- In this mode the effective address is equal to the address part of the instruction. The operand resides in memory and its address is given directly by the address field of the instruction.

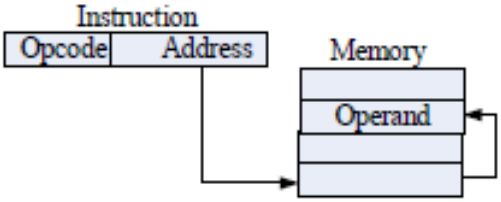
For example LDA 4000H



Effective Address (EA) = A
Advantage: Simple. Disadvantage: limited address field

Indirect Addressing Mode

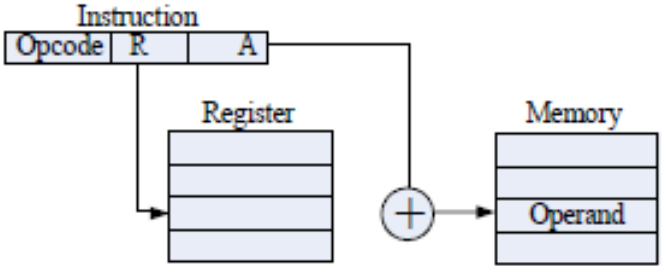
- In this mode the address field of the instruction gives the address where the effective address is stored in memory.
- Control unit fetches the instruction from the memory and uses its address part to access memory again to read the effective address.



Effective Address (EA) = (A)
Advantage: Flexibility. Disadvantage: Complexity

Displacement Addressing Mode

- A very powerful mode of addressing which combines the capabilities of direct addressing and register indirect addressing.
- The address field of instruction is added to the content of specific register in the CPU.



Effective Address (EA) = A + (R)
Advantage: Flexibility. Disadvantage: Complexity

Relative Addressing Mode

- In this mode the content of the program counter (PC) is added to the address part of the instruction in order to obtain the effective address.
- The address part of the instruction is usually a signed number (either a +ve or a –ve number).
- When the number is added to the content of the program counter, the result produces an effective address whose position in memory is relative to the address of the next instruction.

Effective Address (EA) = PC + A

Indexed Addressing Mode

- In this mode the content of an index register (XR) is added to the address part of the instruction to obtain the effective address.
- The index register is a special CPU register that contains an index value.
- Note: If an index-type instruction does not include an address field in its format, the instruction is automatically converted to the register indirect mode of operation.

Effective Address (EA) = XR + A

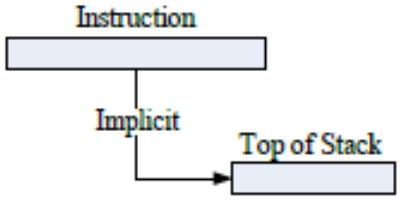
Base Register Addressing Mode

- In this mode the content of a base register (BR) is added to the address part of the instruction to obtain the effective address.
- This is similar to the indexed addressing mode except that the register is now called a base register instead of the index register.
- The base register addressing mode is used in computers to facilitate the relocation of programs in memory i.e. when programs and data are moved from one segment of memory to another.

Effective Address (EA) = BR + A

Stack Addressing Mode

The stack is the linear array of locations. It is some times referred to as push down list or last in First out (LIFO) queue. The stack pointer is maintained in register.



Effective Address (EA) = TOS

Addressing Mode	Effective Address	Content of AC
Direct address	500	800
Immediate operand	201	500
Indirect address	800	300
Relative address	702	325
Indexed address	600	900
Register	—	400
Register indirect	400	700
Autoincrement	400	700
Autodecrement	399	450

Fig: Tabular list of Numerical Example

Data Transfer And Manipulation

Data transfer instructions cause transfer of data from one location to another without changing the binary information. The most common transfer are between the

- Memory and Processor registers
- Processor registers and input output devices
- Processor registers themselves

Typical Data Transfer Instructions

Name	Mnemonic
Load	LD
Store	ST
Move	MOV
Exchange	XCH
Input	IN
Output	OUT
Push	PUSH
Pop	POP

Data Manipulation Instructions

Data manipulation instructions perform operations on data and provide the computational capabilities for the computer. These instructions perform arithmetic, logic and shift

Arithmetic Instructions

Name	Mnemonic
Increment	INC
Decrement	DEC
Add	ADD
Subtract	SUB
Multiply	MUL
Divide	DIV
Add with carry	ADDC
Subtract with borrow	SUBB
Negate (2's complement)	NEG

LOGICAL AND BIT MANIPULATION INSTRUCTIONS

Name	Mnemonic
Clear	CLR
Complement	COM
AND	AND
OR	OR
Exclusive-OR	XOR
Clear carry	CLRC
Set carry	SETC
Complement carry	COMC
Enable interrupt	EI
Disable interrupt	DI

SHIFT INSTRUCTIONS

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

Program Control Instructions

The program control instructions provide decision making capabilities and change the path taken by the program when executed in computer. These instructions specify conditions for altering the content of the program counter. The change in value of program counter as a result of execution of program control instruction causes a break in sequence of instruction execution. Some typical program control instructions are:

Name	Mnemonic
Branch	BR
Jump	JMP
Skip	SKP
Call	CALL
Return	RET
Compare (by subtraction)	CMP
Test (by ANDing)	TST

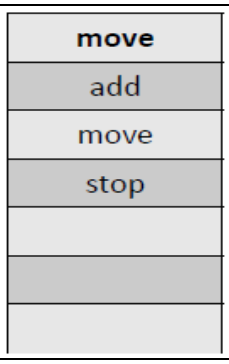
Simple Instruction Set

Assume we have a processor whose Instruction Set consists of four machine language instructions

- Move from a memory location to a data register in CPU
- Move from a data register in CPU to a memory location
- Add the contents of a memory location to a data register
- Stop

Suppose our program for Z = X + Y looks like:

```
Move X, D0
Add Y, D0
Move D0, Z
Stop
```



This program is coded into machine instruction and is loaded into memory starting at location $0000 0000

How does the CPU know which instruction to execute?

There is a dedicated register in CPU called ***Program Counter (PC)*** that points to the memory location where next instruction is stored

Therefore, at start PC = $0000 0000

- Instruction is in Main Memory –it is to be transferred (fetched) to CPU to be executed
- CPU has an Instruction Register (IR) that holds the instruction
- What kind of instruction is to be executed?
- CPU has its own Instruction Interpreter (Decoder)
- Followed by Instruction execution
- Next instruction follows. PC is incremented by length of instruction just completed

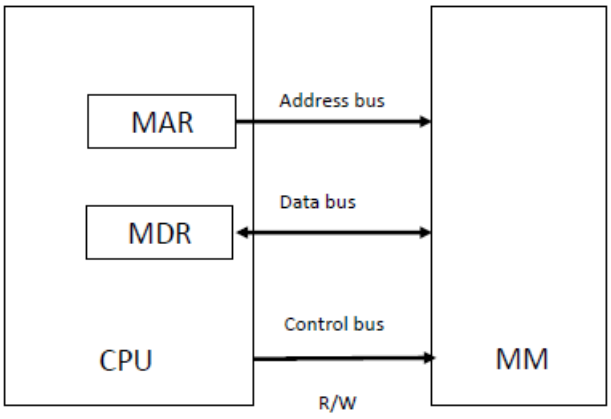
Mechanism of Transferring Data from MM to CPU

CPU has an external bus that connects it to the Memory and I/O devices.

The data lines are connected to the processor via the Memory Data Register (MDR)

The address lines are connected to the processor via the Memory Address Register (MAR)

- Memory address from where the instruction/data is to be accessed is copied into MAR
- Contents of MAR are loaded onto address bus
- Corresponding memory location accessed
- Contents of this location put onto data bus
- Data on data bus loaded into MDR



Program Execution

Fetch Cycle:

- Processor *fetches* one instruction at a time from successive memory locations until a branch/jump occurs.
- Instructions are located in the memory location pointed to by the PC
- Instruction is loaded into the IR
- Increment the contents of the PC by the size of an instruction

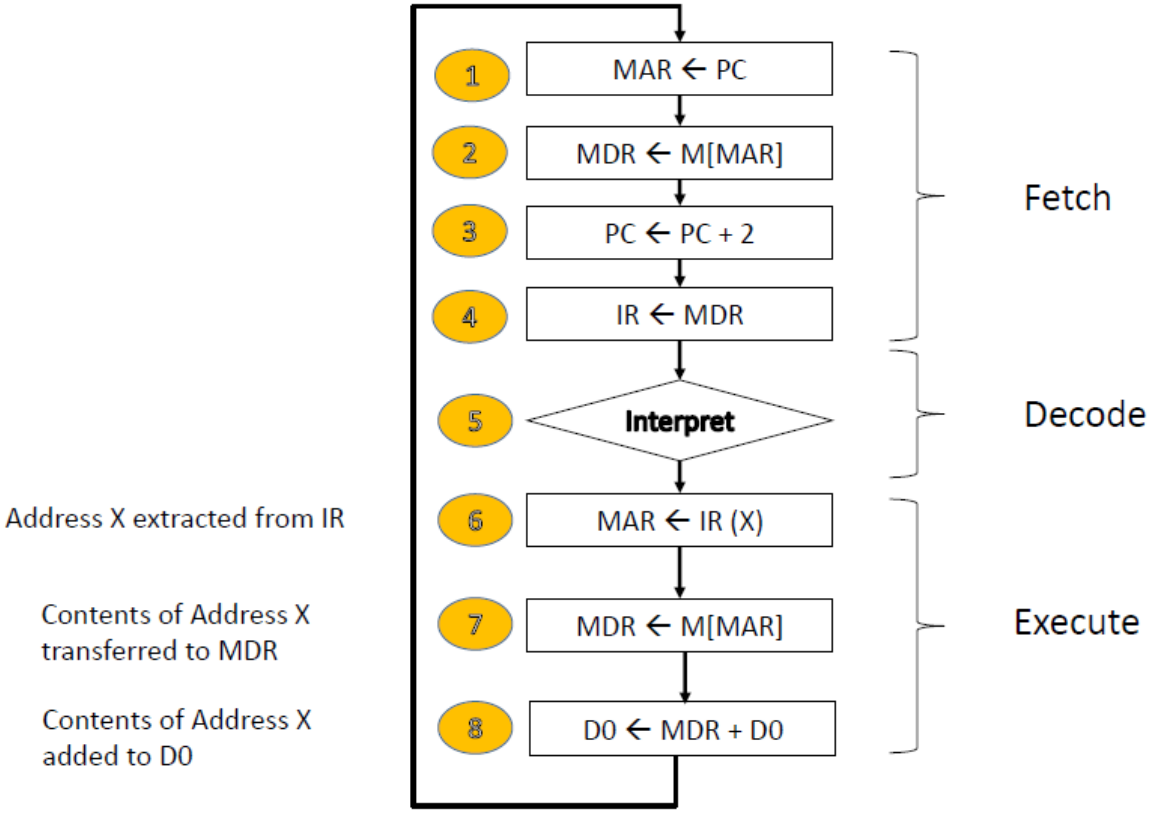
Decode Cycle:

- Instruction is decoded/interpreted, opcode will provide the type of *operation* to be performed, the nature and mode of the operands
- Decoder and control logic unit is responsible to select the registers involved and direct the data transfer.

Execute Cycle:

Carry out the actions specified by the instruction in the IR

Execution for add X, D0 in a GPR processor



Instruction Execution Time

Clock Cycles (P) – is regular time intervals defined by the CPU clock
Clock Rate, R = 1/P cycles per second (Hz)

500 MHz => P = 2ns
1.25 GHz => P = 0.8ns

For each instruction:

Fetch: Total 12 clock cycles

MAR ← PC 1
MDR ← M[MAR] 10
IR ← MBR 1

Decode: 2 clock cycles
Execute: depends on instruction

Micro Step	Number of Clock Cycles
Register Transfer	1
Decoding	2
Add	2
Multiply	5
Memory Access	10

Accumulator (Acc) Architecture

The Acc has only ONE register –accumulator (Acc) instead of the Register File

Example: Z = X + Y

Move contents of location X to Acc
Add contents of location Y to Acc
Move from Acc to location Z
Stop

- All operations and data movements are on this single register
- Most of the instructions in the instruction set require only one Operand
- Destination and Source are implicitly Acc
- Leads to shorter instructions but program may be slower to execute since there are more moves to memory for intermediate results (to free Acc)
- May lead to inefficiency