**Computer Organization**

**UNIT-III**

**Central Processing Unit**

**General register organization:**

The number of registers in a processor unit may vary from just one processor register to as many as 64 registers or more.

1. One of the CPU registers is called as an accumulator AC or 'A' register. It is the main operand register of the ALU.
2. The data register (DR) acts as a buffer between the CPU and main memory. It is used as an input operand register with the accumulator.
3. The instruction register (IR) holds the opcode of the current instruction.
4. The address register (AR) holds the address of the memory in which the operand resides.
5. The program counter (PC) holds the address of the next instruction to be fetched for execution.

     Additional addressable registers can be provided for storing operands and address. This  can be viewed as replacing the single accumulator by a set of registers. If the registers are used for many purpose, the resulting computer is said to have general register organization. In the case of processor registers, a registers is selected by the multiplexers that form the buses.

 When a large number of registers are included in the CPU, it is most efficient to connect them through a common bus system. The registers communicate with each other not only for direct data transfers, but also while performing various micro-operations. Hence it is necessary to provide a common unit that can perform all the arithmetic, logic and shift micro-operation in the processor.

**Stack organization:**

**Stack** is a storage structure that stores information in such a way that the last item stored is the first item retrieved. It is based on the principle of LIFO (Last-in-first-out). The **stack** in digital computers is a group of memory locations with a register that holds the address of top of element.

**Register Stack:-**

A stack can be placed in a portion of a large memory as it can be organized as  a collection of a finite number of memory words as register.

 address

C

B

AA

63

DR

EMPTY

FULL

EMPTY

SP

3

2

1

In a 64- word stack, the stack pointer contains 6 bits because 26 = 64.

The one bit register FULL is set to 1 when the stack is full, and the one-bit register EMTY is set to 1 when the stack is empty. DR is the data register that holes the binary data to be written into on read out of the stack.

Initially, SP is decide to O, EMTY is set to 1, FULL = 0, so that SP points to the word at address O and the stack is masked empty and not full.

**PUSH**       SP ® SP + 1              increment stack pointer
M [SP] ® DR             unit item on top of the Stack
It (SP = 0) then (FULL ® 1) check it stack is full
EMTY ® 0          mask the stack not empty.

**POP**          DR ® [SP]         read item trans the top of stack
SP ® SP –1               decrement SP
It (SP = 0)                then (EMTY ® 1) check it stack is empty

FULL ® 0          mark the stack not full. A stack can be placed in a portion of a large memory or it can be organized as
a collection of a finite number of memory words or registers. Figure X shows the
organization of a 64-word register stack. The stack pointer register SP contains a
binary number whose value is equal to the address of the word that is currently on top
of the stack. Three items are placed in the stack: A, B, and C in the order. item C is
on the top of the stack so that the content of sp is now 3. To remove the top item, the
stack is popped by reading the memory word at address 3 and decrementing the content of SP. Item B is now on top of the stack since SP holds address 2. To insert a
new item, the stack is pushed by incrementing SP and writing a word in the next
higher location in the stack. Note that item C has read out but not physically removed.
This does not matter because when the stack is pushed, a new item is written in its
place.
 In a 64-word stack, the stack pointer contains 6 bits because 26
=64.sinceSP has only six bits, it cannot exceed a number grater than 63(111111 in binary). When 63 is incremented by 1, the result is 0 since 111111 + 1 =1000000 in binary, but SP
can accommodate only the six least significant bits. Similarly, when 000000 is
decremented by 1, the result is 111111. The one bit register Full is set to 1 when the
stack is full, and the one-bit register EMTY is set to 1 when the stack is empty of
items. DR is the data register that holds the binary data to be written in to or read out
ofstack .Initially, SP is cleared to 0, Emty is set to 1, and Full is cleared to 0, so that SP points
to the word at address o and the stack is marked empty and not full. if the stack is not
full , a new item is inserted with a push operation.

SP ←SP + 1 (Increment stack pointer)
M(SP) ← DR (Write item on top of the stack)
if (sp=0) then (Full ← 1) (Check if stack is full)
Emty ← 0 ( Marked the stack not empty)

The stac pointer is incremented so that it points to the address of the next-higher
word. A memory write operation inserts the word from DR into the top of the stack.
Note that SP holds the address of the top of the stack and that M(SP) denotes the
memory word specified by the address presently available in SP, the first item stored
in the stack is at address 1. The last item is stored at address 0, if SP reaches 0, the stack is full of item, so FULLL is set to 1. This condition is reached if the top item
prior to the last push was in location 63 and after increment SP, the last item stored in
location 0. Once an item is stored in location 0, there are no more empty register in
the stack. If an item is written in the stack, Obviously the stack can not be empty, so
EMTY is cleared to 0.

DR← M[SP] Read item from the top of stack
SP ← SP-1 Decrement stack Pointer
if( SP=0) then (Emty ← 1) Check if stack is empty
FULL ← 0 Mark the stack not full

The top item is read from the stack into DR. The stack pointer is then decremented. if
its value reaches zero, the stack is empty, so Emty is set to 1. This condition is
reached if the item read was in location 1. once this item is read out , SP is
decremented and reaches the value 0, which is the initial value of SP. Note that if a
pop operation reads the item from location 0 and then SP is decremented, SP changes
to 111111, which is equal to decimal 63. In this configuration, the word in address 0
receives the last item in the stack. Note also that an erroneous operation will result if
the stack is pushed when FULL=1 or popped when EMTY =1.

**Instruction formats:**

The most common fields found in instruction format are:-

(1)    An operation code field that specified the operation to be performed
(2)    An address field that designates a memory address or a processor registers.
(3)    A mode field that specifies the way the operand or the effective address is determined.

Computers may have instructions of several different lengths containing varying number of addresses. The number of address field in the instruction format of a computer depends on the internal organization of its registers. Most computers fall into one of three types of CPU organization.

(1)    Single Accumulator organization ADD X  AC ® AC + M [×]
(2)    General Register Organization ADD R1, R2, R3 R ® R2 + R3
(3)    Stack Organization             PUSH X

**Three address Instruction**

Computer with three addresses instruction format can use each address field to specify either processor register are memory operand.

ADD  R1, A, B     A1 ® M [A] + M [B]
ADD R2, C, D     R2 ® M [C] + M [B]    X = (A + B) \* (C + A)
MUL X, R1, R2    M [X] R1 \* R2

The advantage of the three address formats is that it results in short program when evaluating arithmetic expression. The disadvantage is that the binary-coded instructions require too many bits to specify three addresses.

**Two Address Instruction**

Most common in commercial computers. Each address field can specify either a processes register on a memory word.

MOV      R1, A         R1 ® M [A]
ADD      R1, B         R1 ® R1 + M [B]
MOV      R2, C         R2 ® M [C]            X = (A + B) \* ( C + D)
ADD      R2, D         R2 ® R2 + M [D]
MUL      R1, R2        R1 ® R1 \* R2
MOV      X1 R1         M [X] ® R1

**One Address instruction**

It used an implied accumulator (AC) register for all data manipulation. For multiplication/division, there is a need for a second register.

LOAD    A              AC ® M [A]
ADD      B              AC ® AC + M [B]
STORE T              M [T] ® AC           X = (A +B) × (C + A)

All operations are done between the AC register and a memory operand. It’s the address of a temporary memory location required for storing the intermediate result.

  LOAD    C              AC ® M (C)
ADD      D              AC ® AC + M (D)
ML        T              AC ® AC + M (T)
STORE  X              M [×]® AC

**Zero – Address Instruction**

A stack organized computer does not use an address field for the instruction ADD and MUL. The PUSH & POP instruction, however, need an address field to specify the operand that communicates with the stack (TOS ® top of the stack)

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

Addressing modes:

1. Addressing modes are nothing but the different ways in which the location of an operand can be specified in an instruction. The number of addressing modes that a processor supports changes according to the instruction set it is based on, however there are a few generic ones that are present in almost all processors and are thus of utmost importance.
2. They are as follows:

i. Immediate mode

ii. Register mode

iii. Indirect mode

iv. Index mode

**Immediate mode**: In this mode, the operand is specified in the instruction itself.

E.g. Move #200,R0

The above instruction places the value 200 in the register R0.Clearly Immediate mode can be used only to specify the source operand.

**Register mode:** The operand is the contents of a register. We specify the operand in this case by specifying the name of the register in the instruction. Processor registers are often used for intermediate storage during arithmetic operations. This addressing mode is used at that time to access the registers.

E.g. Move R0, R1

Contents of the R0 register are moved to R1 register.

**Absolute mode:** The operand is in a memory location; the address of the operand is passed explicitly in the instruction. Global variables are represented using this addressing mode.

E.g. Move LOC, R0

Here LOC corresponds to the address from where the contents will be accessed by the processor and placed in R0.

**Indirect mode:** The effective address (E.A.) of the operands is the contents of a register (see Figure 3(b)) or the memory location whose address appears in the instruction (see Figure 3(a)). The name of the register or the memory address is placed in parentheses to denote indirection or in other words that the contents are addresses of the operands.

E.g. Add (R1), R0 (this mode is often called as register indirect mode) Add (B), R0

This instruction fetches the operand from the address, pointed by the contents of the register R1 or of the memory location ‘B’ and adds them to R0.

**Index mode**: The effective address of the operand is calculated by adding a constant value to the contents of a register, which is clearly shown in Figure 4. The address can be in a register used specially for this purpose or any of the general purpose registers. In either case it is called as an index register.

E.g. Move X (R0), R1

Here, Contents at address X+R0 are moved to R1 .X contains a constant value.

**Relative mode:** For relative addressing, also called PC-relative addressing, the implicitly referenced register is the program counter (PC). That is, the next instruction address is added to the address field to produce the EA. typically, the address field is treated as a twos complement number for this operation. Thus, the effective address is a displacement relative to the address of the instruction as shown in the example below.

Relative addressing exploits the concept of locality.

E.g. Move X (PC), R1

Here, Contents at address X+PC are moved to R1 .X contains a constant value.

**Auto increment mode:** The effective address of the operand is the contents of a register specified in the instruction. After accessing the operand, the contents of this register are automatically incremented to the next value. This increment is 1 for byte sized operands, 2 for 16 bit operands and so on.

E.g. Add (R2) +, R0

Here are the contents of R2 are first used as an E.A. then they are incremented.

**Auto decrement mode:** The effective address of the operand is the contents of a register specified in the instruction. Before accessing the operand, the contents of this register are automatically decremented and then the value is accessed.

E.g. Add - (R2), R0

Here are the contents of R2 are first decremented and then used as an E.A. for the operand which is added to the contents of R0. The auto increment addressing mode and the auto decrement addressing mode are widely used for the implementation of data structures like Stack. There may be other addressing modes that are unique to some processors. However the addressing modes mentioned above are common to many of the popular processors out there.

Program control:

**Status Bit Conditions**

Most CPU architectures maintain a number of status bits that indicate the results from the most recent ALU operation. These bits are usually stored in a *status register*, which is not directly accessible as an argument in machine instructions. The bits are set automatically by many instructions, and used by conditional branch instructions that follow.

V (overflow) indicates overflow in 2's complement.

C (carry) indicates unsigned overflow or shift out.

S (sign) indicates a negative result. (Also called N)

Z (zero) indicates a result of 0. ( all bits are 0 )

**Program status word (PSW):**

The collection of all status bit conditions I the CPU is sometimes called a PSW.

## RISC Characteristics:

      A computer with few instructions and simple construction is called reduced instruction set computer or RISC. RISC architecture is simple and efficient. The major characteristics of RISC architecture are,

1. Relatively few instructions
2. Relatively few addressing modes
3. Memory access limited to load and store instructions
4. All operations are done within the registers of the CPU
5. Fixed-length and easily-decoded instruction format.
6. Single cycle instruction execution
7. Hardwired and micro programmed control

**Parallel processing:**

**Parallel Processing Systems** are designed to speed up the execution of programs by dividing the program into multiple fragments and processing these fragments simultaneously. Such systems are multiprocessor systems also known as tightly coupled systems. Parallel systems deal with the simultaneous use of multiple [computer](http://ecomputernotes.com/fundamental/introduction-to-computer/what-is-computer) resources that can include a single computer with multiple processors, a number of [computers](http://ecomputernotes.com/fundamental/introduction-to-computer/what-is-computer) connected by a network to form a parallel processing cluster or a combination of both.

Parallel systems are more difficult to program than computers with a single processor because the architecture of parallel computers varies accordingly and the processes of multiple CPUs must be coordinated and synchronized. Several models for connecting processors and [memory](http://ecomputernotes.com/fundamental/input-output-and-memory/what-are-the-different-types-of-ram-explain-in-detail) modules exist, and each topology requires a different programming model. The three models that are most commonly used in building parallel computers include synchronous processors each with its own memory, asynchronous processors each with its own memory and asynchronous processors with a common, shared memory. Flynn has classified the computer systems based on parallelism in the instructions and in the data streams. These are:

  1.       Single instruction stream, single data stream (SISD).

 2.         Single instruction stream, multiple data stream (SIMD).

 3.         Multiple instruction streams, single data stream (MISD).

 4.         Multiple instruction stream, multiple data stream (MIMD).

The above classification of parallel computing system is focused in terms of two independent factors: the number of data streams that can be simultaneously processed, and the number of instruction streams that can be simultaneously processed. Here 'instruction stream' we mean an algorithm that instructs the computer what to do whereas 'data stream' (i.e. input to an algorithm) we mean the data that are being operated upon.

Even though Flynn has classified the computer 'systems into four types based on parallelism but only two of them are relevant to parallel computers. These are SIMD and MIMD computers.

SIMD computers are consisting of ‘n' processing units receiving a single stream of instruction from a central control unit and each processing unit operates on a different piece of data. Most SIMD computers operate synchronously using a single global dock.

The block diagram of SIMD computer is shown below:



MIMD computers are consisting of 'n' processing units; each with its own stream of instruction and each processing unit operate on unit operates on a different piece of data. MIMD is the most powerful computer system that covers the range of multiprocessor systems.

**Pipelining:**

In computers, a pipeline is the continuous and somewhat overlapped movement of [instruction](http://searchcio-midmarket.techtarget.com/definition/instruction) to the [processor](http://searchcio-midmarket.techtarget.com/definition/processor) or in the arithmetic steps taken by the processor to perform an instruction. Pipelining is the use of a pipeline. Without a pipeline, a computer processor gets the first instruction from memory, performs the operation it calls for, and then goes to get the next instruction from memory, and so forth. While fetching (getting) the instruction, the arithmetic part of the processor is idle. It must wait until it gets the next instruction. With pipelining, the computer architecture allows the next instructions to be fetched while the processor is performing arithmetic operations, holding them in a [buffer](http://searchcio-midmarket.techtarget.com/definition/buffer) close to the processor until each instruction operation can be performed. The staging of instruction fetching is continuous. The result is an increase in the number of instructions that can be performed during a given time period.

 Computer processor pipelining is sometimes divided into an instruction pipeline and an arithmetic pipeline. The instruction pipeline represents the stages in which an instruction is moved through the processor, including its being fetched, perhaps buffered, and then executed. The arithmetic pipeline represents the parts of an arithmetic operation that can be broken down and overlapped as they are performed.

**Arithmetic pipeling:**

The arithmetic pipeline units are usually found in very high speed computers. They are used to implement floating point operations, multiplications of fixed point numbers and similar computations encountered in specific problems. The floating point operations are easily decomposed into a pipeline unit for floating point addition and subtraction.

The sub operations that are performed in 4 segments are:

1. Compare the exponents

2. Align mantissa

3. Add or subtract the mantissa

4. Normalize the result

The exponents are compared by subtracting them to determine their difference. The larger exponent is chosen as the exponent of the result..The exponent difference how many times the mantissa associated with the smaller exponent must be shifted to the right. This produces an alignment of the two mantissas.

 The two mantissas are added or subtracted in segment3.The result is normalized in segment4.When an overflow occurs, the mantissa of the sum or difference is shifted right ad exponent is increased by 1.

 If an underflow occurs, the number of leading zeros in mantissa determines the number of left shifts I the mantissa and the number must be subtracted from the exponent.

**Instruction pipeline:**

**Instruction pipelining** is a technique that implements a form of parallelism called **instruction**-level parallelism within a single processor. It therefore allows faster CPU throughput (the number of **instructions** that can be executed in a unit of time) than would otherwise be possible at a given clock rate.

Inthe most general case, the computer needs to process each instruction with the following sequence of steps:

* 1. Fetch the instruction from memory
	2. Decode the instruction from memory
	3. Calculate the effective address
	4. Fetch the operand
	5. Execute the instruction
	6. Store the result in the proper place.

Pipeline processing can occur not only in the data stream but in the instruction stream as well. An instruction pipeline reads consecutive instructions from memory while previous instructions are being executed in other segments. This causes the instruction fetch and executes phases to overlap and perform simultaneous operations.

