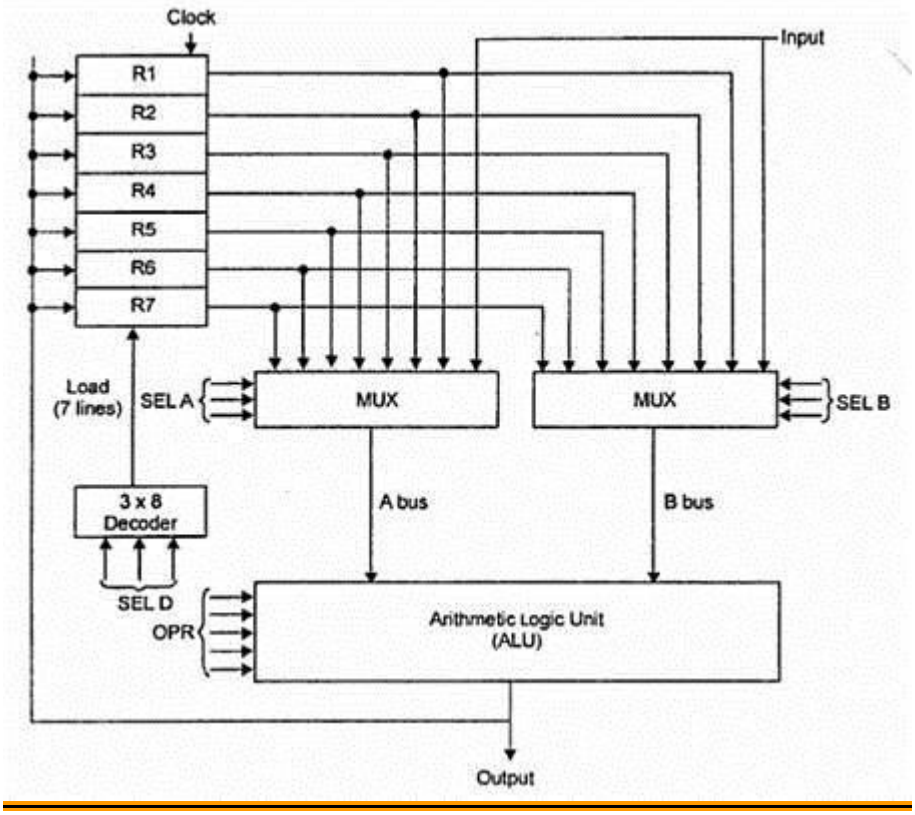
General Register Organization:-

When we are using multiple general-purpose registers, instead of a single accumulator register, in the CPU Organization then this type of organization is known as General register-based CPU Organization. In this type of organization, the computer uses two or three address fields in their instruction format.



The control unit that operates the CPU bus system directs the information flow through the registers and ALU by selecting the various components in the systems.

R1 ® R2 + R3

(1) MUX A selection (SEC A): to place the content of R2 into bus A
(2) MUX B selection (sec B): to place the content of R3 into bus B
(3) ALU operation selection (OPR): to provide the arithmetic addition (A + B)
(4) Decoder destination selection (SEC D): to transfer the content of the output bus into R1

These form the control selection variables are generated in the control unit and must be available at the beginning of a clock cycle. The data from the two source registers propagate through the gates in the multiplexer and the ALU, to the output bus, and into the into of the destination registers, all during the clock cycle intervals.

The advantages of General register-based CPU organization –

- The efficiency of the CPU increases as there are a large number of registers are used in this organization.
- Less memory space is used to store the program since the instructions are written in a compact way.

The disadvantages of General register-based CPU organization –

- Care should be taken to avoid unnecessary usage of registers. Thus, compilers need to be more intelligent in this aspect.
- Since a large number of registers are used, thus extra cost is required in this organization.

For Example-

MULT R1, R2, R3

This is an instruction of an arithmetic multiplication written in assembly language. It uses three address fields R1, R2, and R3. The meaning of this instruction is:

R1 <-- R2 * R3

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 the top of the element. This register that holds the address of the top element of the stack is called Stack Pointer.

Stack Operations:-

The two operations of a stack are:

Push: Insert an item on top of the stack.
Pop: Deletes an item from the top of the stack.

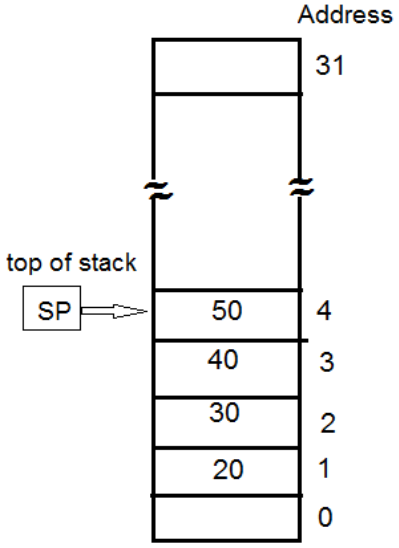
Implementation of Stack

In digital computers, the stack can be implemented in two ways:

- Register Stack
- Memory Stack

Register Stack

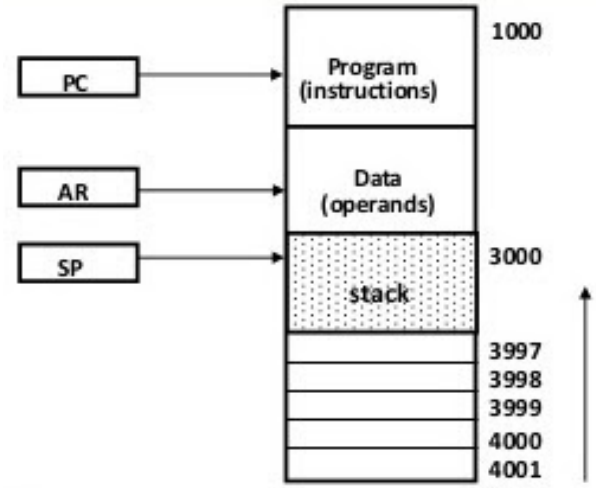
A stack can be organized as a collection of the finite numbers of registers that are used to store temporary information during the execution of a program. The stack pointer (SP) is a register that holds the address of top the element of the stack.



1. Four elements stored in the stack.
2. The data element 50 is top of the stack, therefore the content Of SP is now 4.
3. The stack pointer is a 5-bit register, because 2^5=32.
4. Initially it is clear to 0 and the stack is said to be empty.
5. When the data element is pushed on the stack, SP increments.

Memory Stack

A stack can be implemented in a random access memory (RAM) attached to a CPU. The implementation of a stack in the CPU is done by assigning a portion of memory to a stack operation and using a processor register as a stack pointer. The starting memory location of the stack is specified by the processor register as the stack pointer.



ı a

Addressing Modes:-

The term addressing modes refers to the way in which the operand of an instruction is specified. The addressing mode specifies a rule for interpreting or modifying the address field of the instruction before the operand is actually executed.

The address of the memory can be specified directly within the instruction. There are some situation when we need to change the memory address dynamically.

Addressing modes:-

1. Implied mode
2. Immediate mode
3. Register mode
4. Register indirect mode
5. Auto-Increment or Auto-Decrement mode
6. Direct addressing mode
7. Indirect addressing mode
8. Relative addressing mode
9. Indexed addressing mode
10. Based Register addressing mode

Implied mode:

In implied addressing the operand is specified in the instruction itself. In this mode, the data is 8 bits or 16 bits long and data is the part of instruction. Zero address instructions are designed with implied addressing mode.

Ex- The instruction "Complement Accumulator (CNA)" is an implied-mode.

Immediate addressing mode (symbol #):

In this mode, data is present in the address field of instruction . Designed like one address instruction format.

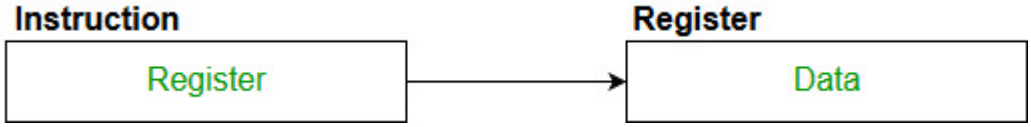
Note: Limitation in the immediate mode is that the range of constants is restricted by the size of the address field.

Register mode:

In register addressing the operand is placed in one of 8-bit or 16-bit general-purpose registers. The data is in the register that is specified by the instruction.

Here one register reference is required to access the data.

Ex-MOV AX,BX



Register Indirect mode:

In this addressing the operand's offset is placed in any one of the registers BX, BP, SI, DI as specified in the instruction. The effective address of the data is in the base register or an index register that is specified by the instruction.

Here two registered reference is required to access the data.

Ex- MOV R1, [R2](move the contents of memory location s addressed by the register BX to the register AX)

Autoincrement or Autodecrement 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.

Example of Autoincrement:-

Add R1, (R2)+ // OR

R1 = R1 +M[R2]

R2 = R2 + d

Example of Autodecrement:-

Add R1,-(R2) //OR

R2 = R2-*d*
R1 = R1 + M[R2]

Direct addressing mode (symbol [ ]):

The operand's offset is given in the instruction as an 8 bit or 16-bit displacement element. In this addressing mode, the 16-bit effective address of the data is part of the instruction. Here only one memory reference operation is required to access the data.

Example: ADD AL,[0301] //add the contents of offset address 0301 to AL

Effective address = Where op/data stored in

Indirect addressing Mode (symbol @ or ()):

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

Effective address = Address part of instruction + Content of CPU Register

Relative address mode:-

In this mode, the content of the program counter is added to the address part of the instruction in order to obtain the effective address.

Effective address = Address part of instruction + Content of PC

Indexed addressing mode:

The operand's offset is the sum of the content of an index register SI or DI and an 8-bit or 16-bit displacement.

Effective address = Address part of instruction + Content of index register
Example:MOV AX, [SI +05]

Based Indexed Addressing:

The operand's offset is some of the content of a base register BX or BP and an index register SI or DI.

Effective address = Address part of instruction + Content of the base register

Example: ADD AX, [BX+SI]

Advantages of Addressing Modes

- To give programmers to facilities such as Pointers, counters for loop controls, indexing of data, and program relocation.
- To reduce the number of bits in the addressing field of the Instruction.

Difference between RISC & CISC:-

RISC:

It stands for Reduced Instruction Set Computer. It is a type of microprocessor architecture that uses a small set of instructions of uniform length. These are simple instructions that are generally executed in one clock cycle. RISC chips are relatively simple to design and inexpensive.

Examples: SPARC, POWER PC, etc.

CISC:

It stands for Complex Instruction Set Computer. These processors offer the users, hundreds of instructions of variable sizes. CISC architecture includes a complete set of special-purpose circuits that carry out these instructions at a very high speed. These instructions interact with memory by using complex addressing modes.

Examples: Intel architecture, AMD

CISC	RISC
A large number of instructions are present in the architecture.	**Very fewer instructions are present. The number of instructions are generally less than 100.**
Some instructions with long execution times. These include instructions that copy an entire block from one part of memory to another and others that copy multiple registers to and from memory.	**No instruction with a long execution time due to very simple instruction set. Some early RISC machines did not even have an integer multiply instruction, requiring compilers to implement multiplication as a sequence of additions.**

Variable-length encodings of the instructions. Example: IA32 instruction size can range from 1 to 15 bytes.	Fixed-length encodings of the instructions are used. Example: In IA32, generally all instructions are encoded as 4 bytes.
Multiple formats are supported for specifying operands. A memory operand specifier can have many different combinations of displacement, base and index registers.	Simple addressing formats are supported. Only base and displacement addressing is allowed.
CISC supports array.	RISC does not supports array.
Arithmetic and logical operations can be applied to both memory and register operands.	Arithmetic and logical operations only use register operands. Memory referencing is only allowed by load and store instructions, i.e. reading from memory into a register and writing from a register to memory respectively.
Implementation programs are hidden from machine level programs. The ISA provides a clean abstraction between programs and how they get executed.	Implementation programs exposed to machine level programs. Few RISC machines do not allow specific instruction sequences.
Condition codes are used.	No condition codes are used.