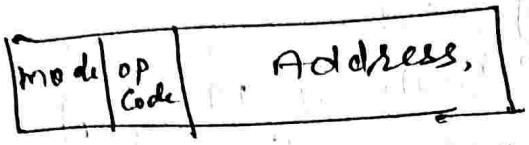
Addressing modes:→

There are Three fields in an instruction

(1)- opcode
(2)- Address field
(3)- mode field.



There are several ways to specify an operand. These all known as Addressing modes.

The way the operands are chosen during the execution of program is depend on the addressing mode of the instruction.

(1) Implied Addressing →

In this type of addressing mode operands are defined implicitly as a part of definition of the instruction itself.

ex - increment Accumulator instruction
Complement AC inst- CMA

All the register reference instruction that use Accumulator are implied Addressing mode.

(2) - Immediate Addressing →

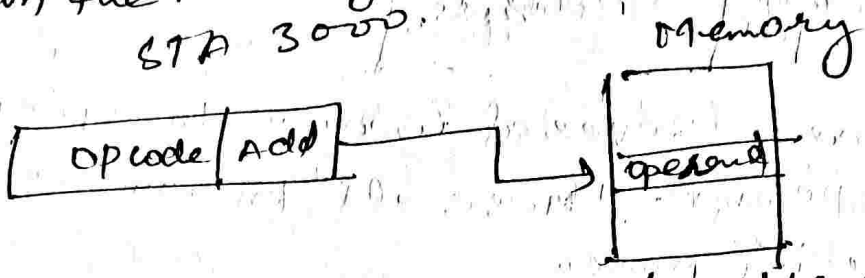
* direct operand is given.
* there is no need of memory reference other than the instruction fetch.

So it saves one memory cycle in the instration. MVI C, 05

Direct Addressing :→

In this type of Addressing the address of operand is present in the instruction. This is very simple form of Addressing. The operand is present in the memory

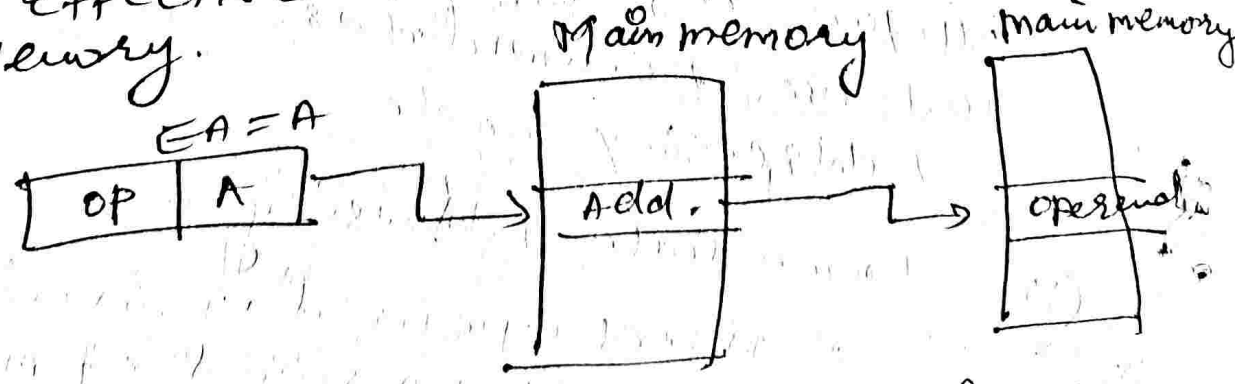
STA 3000.



It provide only a limited address space. With direct addressing the length of the address field is less than the word length, so limited Address range.

Indirect Addressing :→ In this technique address field of the instruction represents the another address field. (Rather than the actual value of operand)

In this mode the address field of the instruction gives the address where the effective address is stored in memory.



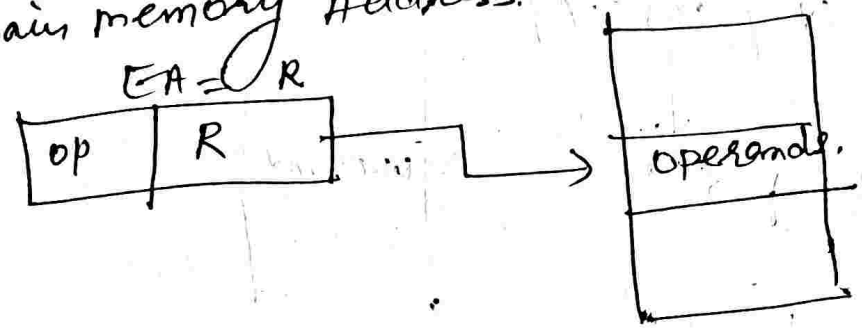
Advantage :→ the word length of n Can use 2^n Address space.

DisAdvantage - two memory Read cycle.

Register Addressing.


```
MOV C, B
ADD B, C
```

In this mode the operand are in registers. that reside within the CPU
It is similar to Direct Addl.
the only diff'n is that the Address field refers to a registers. Rather than a main memory Address.



Advantage:
no memory ref. is needed
So the Access time is less.
than the memory ref.

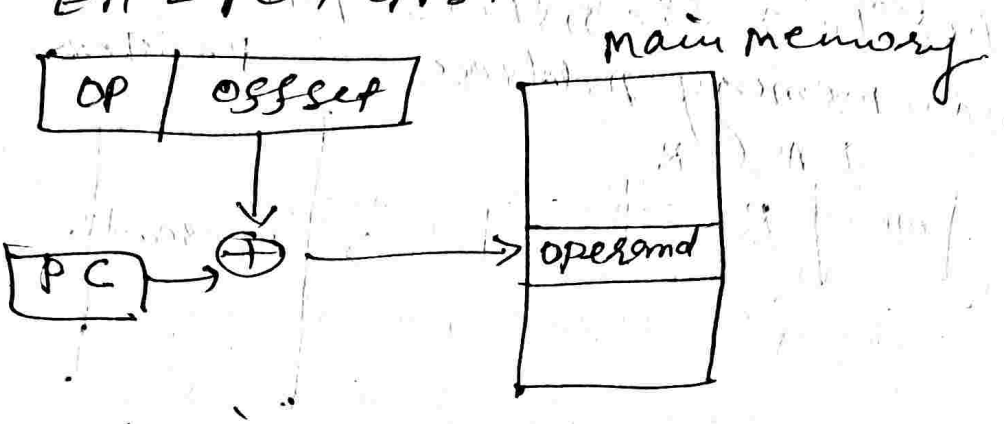
Disadvantage Address space is limited
Because no of Registers are limited.

Register indirect Addressing
Register indirect Addressing is similar to indirect Addressing. In this case address field refers to a registers. and in indirect Addressing Address field. refer to a memory location

Relative Addressing

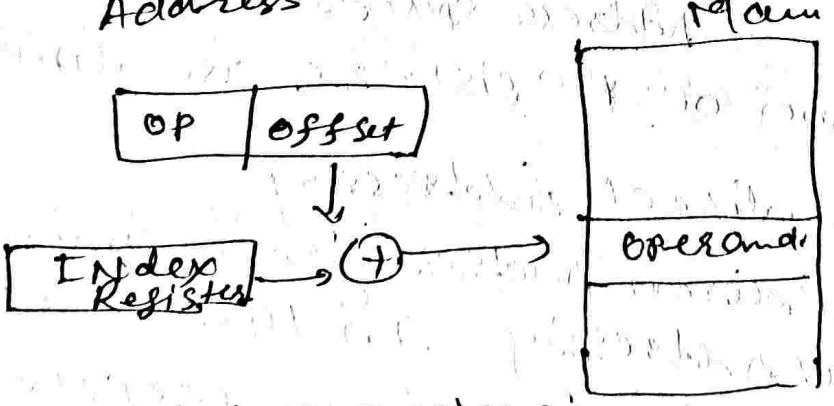
In this technique contents of Program Counter (PC) is used as a base Address Contents of program Counter is added with the address part of the instruction. In order to get the effective address.

EA = PC + offset



INDEX Addressing :→

In this addressing mode the Contents of an INDEX Register is added with the address part of the instruction to obtain the effective Address

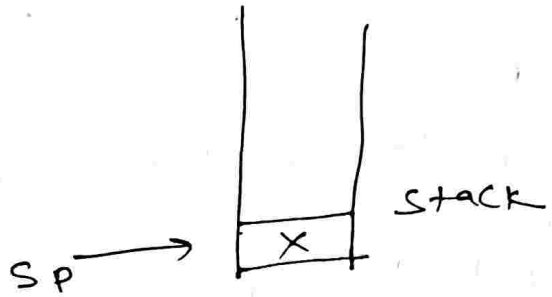


the index Register is a special CPU register that contains an index value: the distance between the beginning address and the address of the operand. The index value stored in the index Register

Ref: Computer System Arch/(P.V) page-262

Stack organisation :→

A stack is a storage device that stores information in Last in, first out list.



The register that hold the address for the stack is known as stack pointer (SP). It point top item in the stack.

There is two operation in stack

(1) - push. (push - down)

(2) - pop (pop - up)

push → inserting new item in stack

pop → deleting the item or removing the items from stack.

These operation incrementing or decrementing the stack point er Register.

Register stack :→

A stack can be placed in a portion of memory or it can be large organized as a collection of a finite number of memory word or registers

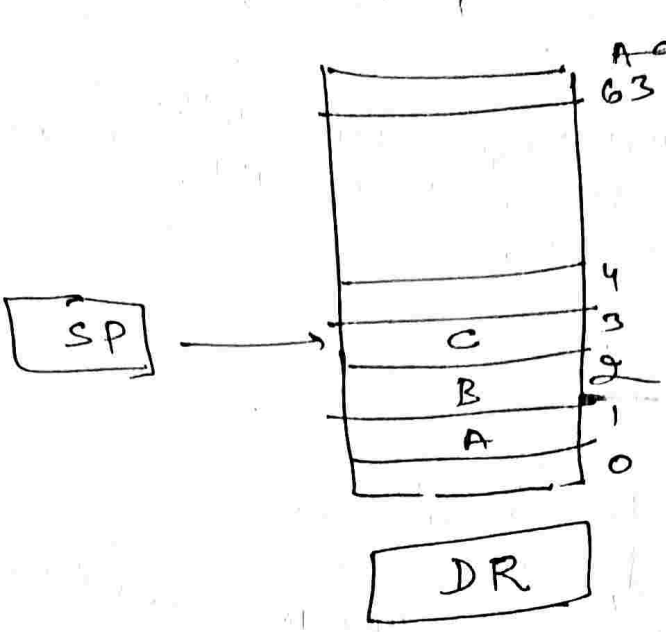


Fig: Block diagram of a 64 word s

In The stack pointer Register, sp contain a binary number whose value is equal to the address of the word that is currently on top of the stack. There is 3 items are placed in the stack: A, B and C the item C is on 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. the item B is now on top of the stack., so stack pointer has Address 2.

✓ To insert new value stack. pointer is incremented.

✱ In a 64 word stack, the stack pointer contains 6 bits because $2^6 = 64$. since sp has only six bits, it cannot exceed a.

number greater 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 one bit register EMPTY is set to 1 when the stack is empty of items. DR is the data register | EMPTY that holds the binary data to be written into or read out of the stack.

Initially, SP is cleared to 0, EMPTY is set to 1, and FULL is cleared to 0. so that SP points to the word at address 0 and the stack is marked empty and not full. if the stack is not full (if FULL=0), a new item is inserted with a push operation The push operation is implemented with the following sequence of microoperations.

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 make the stack not empty.

EMTY ← 1 ⇒ stack empty
EMTY ← 0 ⇒ full

The stack 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
Notes 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 reach 0 the stack is full of item. So full is set 0 1. This condition is reached if the top item prior to the last push was in location 63. and after incrementing SP, the last item is stored in location 0. once an item is stored in location 0, there are no more empty registers in the stack. if an item is written in the stack, obviously the stack cannot be empty, So EMTY is cleared to 0.

A new item is deleted from the stack is not empty (if EMTY=0) the pop operation consists of the following sequece of microoperation

DR ← M[SP]
SP ← SP-1
If (SP=0) then (EMTY ← 1)
FULL ← 0

field to specifies the operand that communicates with the stack. the following program shows how $X = (A+B) * (C+D)$ will be written for a stack organization (TOS stand for top of 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

To evaluate arithmetic expression in a stack computer, it is necessary to convert the expression into reverse polish notation.

The name "zero-address" is given to this type of computer because the absence of an address field in the computational instructions.

one - Address instructions.

One-address instructions use an implied accumulator (AC) register for all data manipulation. For multiplication & division there is need for a second register. However, here we neglect the second register and assume that the AC contains the result of all ~~operands~~ operations. the program to evaluate

$X = (A + B) * (C + D)$ is

~~Load~~

LOAD	A	$AC \leftarrow M[A]$
ADD	B	$AC \leftarrow AC + M[B]$
STORE	T	$M[T] \leftarrow AC$
LOAD	C	$AC \leftarrow M[C]$
ADD	D	$AC \leftarrow AC + M[D]$
MUL	T	$AC \leftarrow AC * M[T]$
STORE	X	$M[X] \leftarrow AC$.

* All operations are done between the AC Register and a memory operand.
T is the address of a temporary memory location required for storing the intermediate result.

Zero-Address instructions

A stack-organization computer does not use an address field for the instructions ADD and MUL. The PUSH and POP. instructions however, need an address.

Three-Address instructions :-

Computers with 3-address instruction formats can use each address field to specify either a processor register or a memory operand. The program in assembly language that evaluates $X = (A + B) * (C + D)$ is shown below, together with comments that explain the register transfer operation of each instruction.

ADD	R_1, A, B	$R_1 \leftarrow M[A] + M[B]$
ADD	R_2, C, D	$R_2 \leftarrow M[C] + M[D]$
MUL	X, R_1, R_2	$M[X] \leftarrow R_1 * R_2$

It is assumed that the computer has two processor Registers R_1 & R_2.

TWO - Address instructions

The two - address instructions are the most common in commercial computers. Here again each address field can specify either a processor Registers or a Memory word. The program to evaluate $X = (A + B) * (C + D)$ is as follows.

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

The MOV instruction moves or Transfers the operands to and from memory and processor Registers. The first Symbol listed in an instruction is assumed to be both a Source and the destination where the result of the operation is transferred.