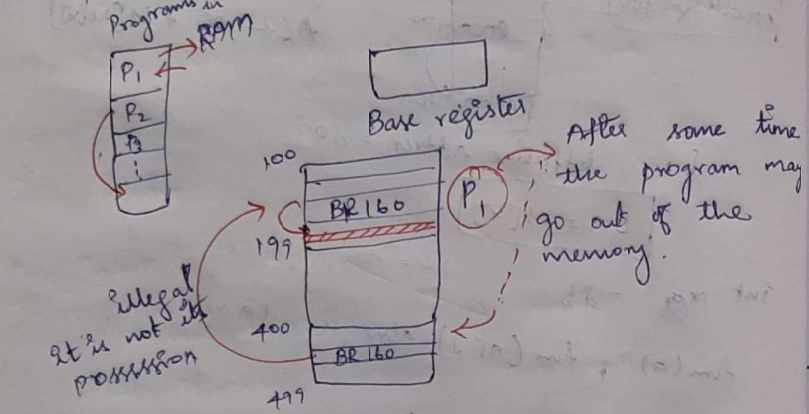
Benefits:

* Used for program relocation, while the pc goes sequentially
* Reduce the address size

BASE REGISTER ADDRESSING:-

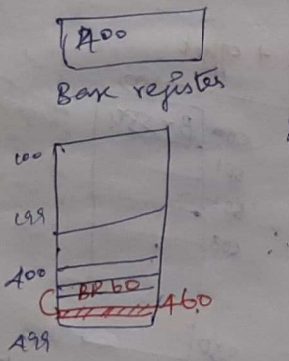


160 = 100 + 60

∴ BR to 60

EA = Base Register + Displacement

So



Displacement = 60

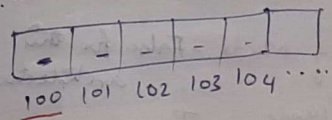
Base register = 400

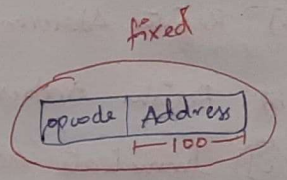
∴ EA = 460

INDEXED ADDRESSING MODE:-

* Use to access or implement array efficiently
* Multiple registers required to implement (one of them is index register)
* Any element can be accessed without changing instruction

eg:-





4 = 100 + 6
= 106
Index register

Effective Address =
Base Address + IR
100 + 4
= 104

Qn:

A certain architecture supports indirect, direct and register addressing mode for use in identifying operands for arithmetic instructions. which one of the following can not be achieved with a single instruction.

a) Specifying a register no in instruction such that register contain value of an operand that will be used by operation

b) Specifying a register no in instruction such that register will serve as destination of operations output

c) Specifying an operand value in instruction such that value will be used by operation

d) Specifying a memory location such that it contains value of operand that will be used by operation.

Ans:-

a] Register Addressing Mode - True

b] Register Addressing Mode - True

c] Immediate Addressing Mode - False for this architecture

d] Direct Addressing Mode - True

2) Column - I

- Indirect addressing
- Immediate addressing
- Auto-decrement

Column - II

- Loops
- pointers
- Constants

3). In the absolute addressing mode

1) The operand is inside the instruction

2) The address of the operand is inside the instruction

3). The register containing the address of the operand is specified inside the instruction

4) The location of the operand is implicit

Ans: (2)

(4). Which of the following addressing modes are suitable for program relocation at run time?

1) Absolute 2) Base 3) Relative

4) Indirect

Ans: (2) & (3)

(5). Array Implementation
Writing relocatable code
passing array as parameter

- indirect addressing
- indexed
- Base register

PIPELINING IN COMPUTER ARCHITECTURE:-

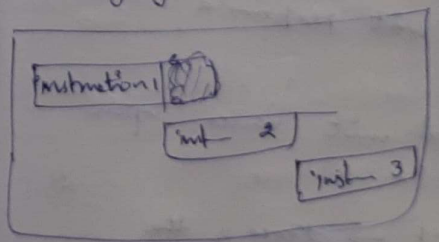
⇒ A program consists of several number of instructions

Execution of Instruction

- Non-pipelined execution
- Pipelined

Non-pipelined execution

* All the instruction of a program are executed sequentially one after other
* new instruction executes only after the previous one has executed completely
* highly inefficient



Time taken = $n \times t$
$= 3 \times t$
$= 3t$

$t \rightarrow$ time for one execution

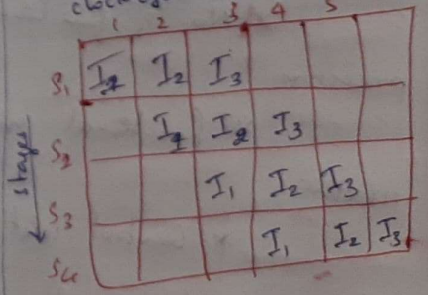
Pipelined

* Multiple instructions are executed parallely
* highly efficient

Four stage pipline:

- I. Fetch (IF)
- I. decode (ID)
- I. Execute (IE)
- Write Back (WB)

Phase time diagram:



Time taken = 6 - clock cycle

Calculation:- for 3 Instruction:-

non-pipelined

for

Time taken = 3 × t

= 3 × 4

= 12 clock cycles

one instruction

- IF
- ID
- IE
- WB

4 clock cycles

pipelined

Time taken = 6 clock cycles

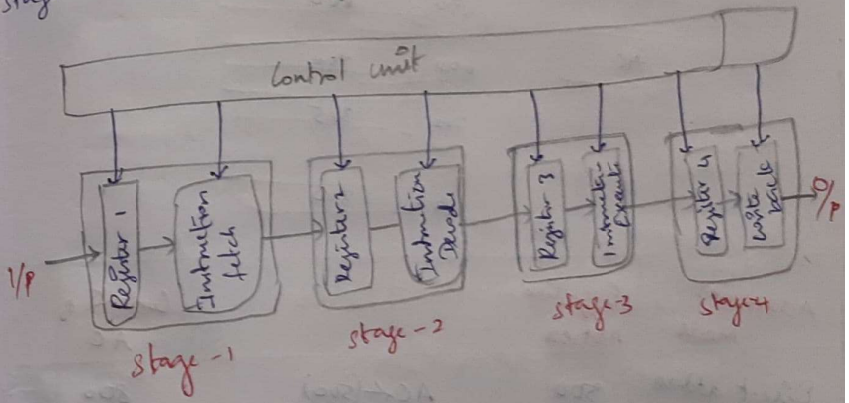
= (time taken for the first instruction to complete + (n-1) × 1 cycles)

INSTRUCTION PIPELINING:-

* It is a technique that implements a form of parallelism called as instruction level parallelism within a single processor

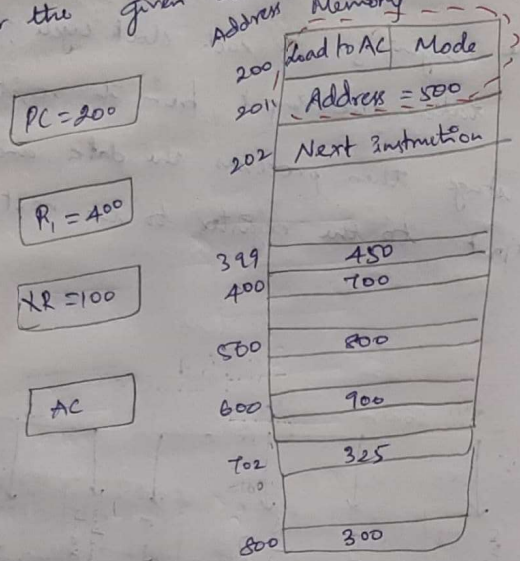
PIPELINED ARCHITECTURE:-

* The hardware of the CPU is split up into several functional units
* Each functional unit performs a dedicated task
* No. of functional unit varies upon processors
* functional unit → stages of the pipeline
* Control unit manages all the state using control signal
* Registers in each stage holds the data
* Global clock that synchronizes the working of all the stages.
* At the beginning of each clock cycle, each stage takes the input from it register
* Each stage then processes the data and feed its output to the register to the next stage



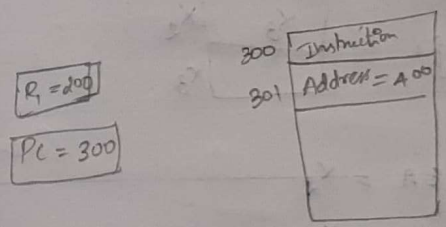
MODE	ALGORITHM	ADVANTAGE	DISADVANTAGE
Immediate	Operand = I	No memory reference	Limited operand magnitude
Direct	EA = A	Simple	Limited address space
Indirect	EA = (A)	Large Address space	multiple memory reference
Register	EA = R	No memory Reference	Limited address space
Register Indirect	EA = (R)	Large address space	Extra memory space
Displacement	EA = A + (R)	Flexibility	Complexity
Stack	EA = Top of stack	No memory Reference	Limited Applicability

Find the effective address and the content of AC for the given data



Addressing Mode	Effective Address		Content of AC
Direct address	500	AC ← (500)	800
Immediate operand	201	AC ← 500	500
Indirect address	800	AC ← ((500))	300
Relative	702 (PC contains the address of next instruction)	AC ← (PC + 500)	325
Indexed	600	AC ← (XR + 500)	900
Register	-	AC ← R_1	400
Register Indirect	400	AC ← (R_1)	700
Auto increment	400	AC ← (R_1) +	700
Auto decrement	399	AC ← -(R_1)	450

2). An instruction is stored at location 300 with its address field at location 301. The address field has value 400. A processor register R_1 contains the number 200. Evaluate the EA if the addressing mode of the instruction is (a) direct (b) immediate (c) relative (d) register indirect (e) index with R_1 as the index register.



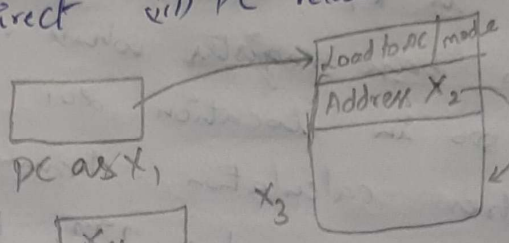
S.No	Mode	EA		Content of AC
a)	direct	500	AC ← [500]	[500] data @ 500
b)	immediate	301	AC ← 400	400
c)	relative	700	AC ← (PC + 400)	[700]
d)	register indirect	200	AC ← (R_1)	[200]
e)	indexed	600	AC ← (R_1 + 400)	[600]

3) Let the address stored in the program counter designated by the symbol X_1. The instruction stored in X_1 has the address part (operand reference) X_2. The operand needed to execute the instruction is stored in the memory word with address X_3. An index register contains

the value X_4. What is the relationship b/w these various quantity if the addressing mode of the instruction is

(a) direct b) indirect iii) PC relative

d) Indexed



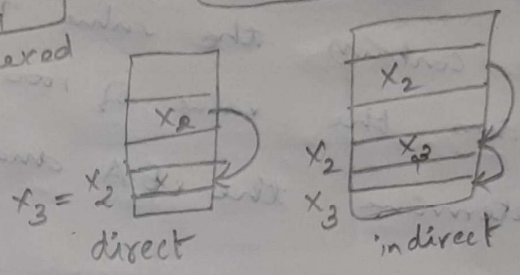
a) $X_3 = X_2$

b) $X_3 = (X_2)$

c) $X_3 = (X_1 + 1) + X_2$

next instruction

d) $X_3 = X_2 + X_4$



4). An addressing field in an instruction contains decimal value 14. where is the corresponding operand located for

a) Immediate addressing - 14 (The address field)

b) Direct addressing - (14) memory location 14.

c) Indirect - The memory location whose address is in memory location (14)

d) Register - Register 14.

e) Register indirect - The memory location whose address is in register 14.

opcode	Address = 14

) Consider a 16-bit processor in which the following appears in main memory, starting at location 200

200	load to AC mode
201	500
202	Next instruction

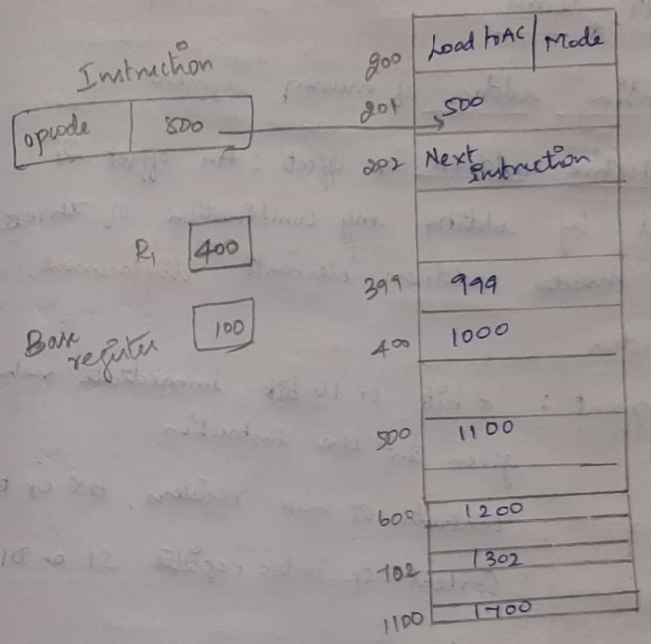
The first part of the first word. The mode field specifies an addressing mode and

if appropriate indicates a source register, assuming that when used, the source register is R_1; which has a value of 400. There is base register whose value is 100. The value of 500 in location 201 may be part of the address calculation. Assume that location 399 contains the value 999, location 400 contains the value of 1000. and so on.

Determine the EA and operand.

Mode	EA	Operand
Direct	500	1100
Immediate	201	500
Indirect	1100	1700
PC relative	201+1+500=702	1302
Displacement	500+100=600	1200
Register	R_1	400
Register indirect	400	1000
Autoindexing with increment using R_1	400	1000

* R_1 is incremented after the execution of the instruction



b). Memory values and a one address machine with an accumulator is given as

Word 20 contains 40
Word 30 contain 50
Word 40 contains 60
Word 50 contains 70

20	40
30	50
40	60
50	70

What values to the following instructions load into the accumulator

a. Load IMMEDIATE 20 — AC ← 20
b. Load DIRECT 20 — AC ← 40
c. Load INDIRECT 20 — AC ← 60
d. Load IMMEDIATE 30 — AC ← 30
e. Load DIRECT 30 — AC ← 50
f. Load INDIRECT 30 — AC ← 70

IMPORTANT terms:-

* Starting address of memory segment
* Effective address or offset: An offset is determined by adding any combination of three addressing modes address elements: displacement, Base, index.

Displacement : 8 bit or 16 bit immediate value given in the instruction

Base : Contents of base registers, BX or BP

Index : Content of index register SI or DI.

Symbols:-

for immediate addressing : #

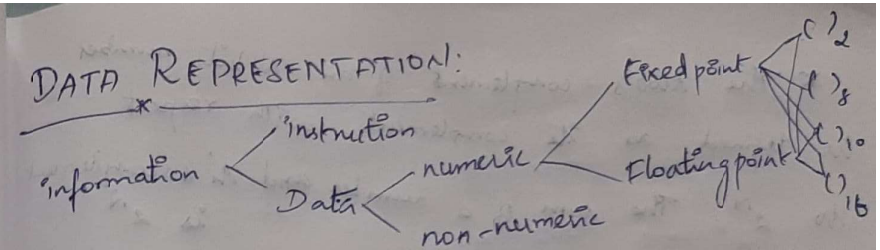
Direct / absolute addressing : [ ]

indirect addressing : @ or ( )

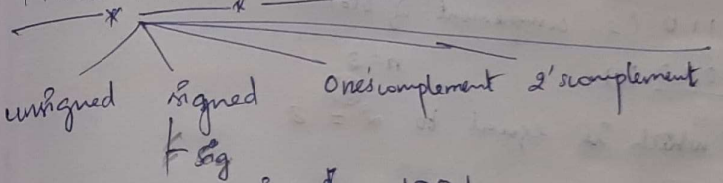
Examples:-

1) Implied : CLC (used to reset carry flag to 0)
2) Immediate : MOV AL, 35H
3) Register : MOV AX, CX
4) Register indirect : MOV AX, [BX]
5) Auto indexed :
   - increment mode : Add R_1, $(R_2)+$
   - decrement mode : Add R_1, $-(R_1)$
6) Direct : Add AL, [0301]
7) Indirect : Add AL, @301 (301)
8) Based indexed addressing : MOV AX, [BX+SI]
9) Indexed addressing : Add AX, [SI+05]

DATA REPRESENTATION:



FIXED POINT REPRESENTATION:



eg: 9 → unsigned 1001

signed: +9 ⇒ 01001 (sign & magnitude)
⇒ 01110 (1's complement)
⇒ 01111 (2's complement)

-9 ⇒ 11001 (S&M)
⇒ 10110 (1's)
⇒ 10111 (2's)

Reason for calling it as 1's complement:-

* Complementing a single bit ≡ to subtracting it from 1

0's = 1 and 1 − 0 = 1

1's = 0 and 1 − 1 = 0

1's/9's

* Complementing each bit of an n-bit number is equivalent to subtracting it from $2^n - 1$

eg: 01101, here $n = 5 \Rightarrow 2^n - 1 = 2^5 - 1 = (32)_{10} = (11111)_2$

```
 11111
 01101   (-)
------
 10010
```

* The two's complement of an N-bit number is defined as its complement with respect to 2^N. The sum of a number and its 2's complement is 2^N

eg:

```
  010   (+)
  110   (2's complement of 010)
 1000   (8)
```

n = 3

8 which is equal to $2^3 = 8$

1's complement Addition:-

* perform binary addition
* If there is carry, Add 1 to the result
* check overflow: overflow occurs if the result is opposite sign of A & B

eg:-

Note:- * first do unsigned addition with the numbers, including the sign bit

* Add the carry to the result

```
  0111   (+7)
(+) 1011   (-4)
 10010

  0010
(+)  1
  0011   (+3)
```

```
  0011   (+3)
(+) 0010   (+2)
  0 0101

  0101
(+)  0
  0101   (-5)
```

1's complement Subtraction:-

* Take 1's complement of B by inverting all bits
* Add the 1's complement of B to A.

$$A - B = A + (-B)$$

2's complement Addition:-

* Perform Binary addition
* Ignore the carry out of the MSB
* Check for overflow: Overflow occurs if the 'carry in' and 'carry out' of the MSB are different, or if result is opposite sign of A and B

2's complement subtraction:-

$$A - B = A + (-B)$$

* Take 2's complement of B by inverting all the bits and adding 1
* Add the 2's complement of B to A.

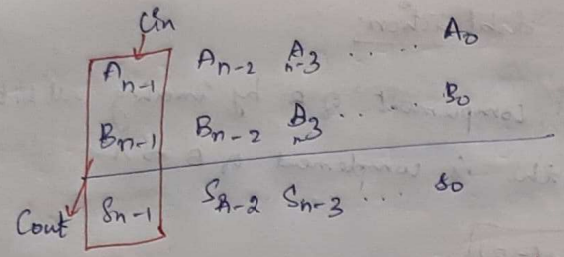
Overflow detection:-

Occurs when:

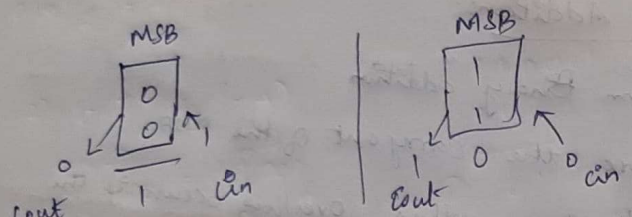
(i) 2 negative numbers are added and an answer comes as +ve.

(ii) 2 +ve numbers are added and an answer comes as -ve.

⇒ So overflow can be detected by checking MSB of two operand and answer.



* Overflow occurs when $C_{in} \neq C_{out}$



* Overflow bit in 2's complement subtraction is ignored

Overflow → when the addition of 2 binary numbers yields a result that is greater than the max. possible value

Underflow → when the addition / subtraction of 2 binary numbers yields a result that is less than the min. possible value

Overflow

Underflow

Assume 4 bit restriction and 2's complement

Max. possible value:

$2^{4-1} - 1 = 7$

$6_{10} + 3_{10} = 9_{10}$

0110_2 6

$+ \quad 0011_2$ +3

1001_2 -7 ×

Min. possible value:

$-(2^{4-1}) = -8$

$-5_{10} + -5_{10} = -10_{10}$

1011_2 -5

$+ \quad 1011_2$ -5

0110_2 +6 ×

Cases:

i) Carry generated but no overflow

cin → 1

1101 (-3)

+ 1011 (-5)

cout → 1 1000 (-8)

ii) No carry and no overflow

1101 (-3)

+ 0010 (+2)

0 1111 (-1)

iii) carry and overflow

1101 (-3)

+ 1010 (-6)

1 0111 (+7)

iv) no-carry but overflow

0111 (+7)

+ 0011 (+3)

0 1010 (-6)

Cin ⊕ Cout ⇒ overflow

Shortcut to check the overflow:-

* x & y are the sign bits of two numbers
* z is the sign bit of result

$\bar{x} \cdot \bar{y} \cdot z + x \cdot y \cdot \bar{z} = 0$ (no overflow)

$\bar{x} \cdot \bar{y} \cdot z + x \cdot y \cdot \bar{z} = 1$ (overflow

Binary Arithmetic using 2's complement:

* If the resultant sum contains carry out from the sign bit, then discard it in order to get the correct value.

* If Resultant sum is +ve, keep it as it is. If it is -ve, then take 2's complement of the result in order to get the magnitude

eg:

1) $(+7) + (+4)$

$+7_{10} = 00111_2$

$+4_{10} = 00100_2$

$(+7) + (+4) = \underbrace{0}_{+ve}1011_2$

$01011_2 = +11_{10}$

2) $(-7) + (-4)$

$-7_{10} = 11001_2$ (extra bit must also be 1)

$-4_2 = 11100_2$

11001

11100

$\overline{(1)10101}$

Carry out from the sign bit discard it

$(-7) + (-4) = 10101_2$

Since the result is in -ve then 2's complement for 0101 is $1010 + 1 = 1011_2 = (11)_{10}$

$\therefore (-7) + (-4) = 10101_2 = -11$

3) $(+7) - (+4)$

$7 - 4 = 7 + (-4)$

2's complement of

$+7_{10} = 00111_2$

$+4_{10} = 00100_2$

$-4_{10} = 11100_2$

$(+7) + (-4)$

00111

11100

$\overline{(1)00011_2 = 3_{10}}$

4) $(+4) - (+7)$

$-7_{10} = 11001_2$

$(+4) + (-7) \Rightarrow 00100$

11001

$\overline{(1)11101}$

$(1)1101_2 = -3_{(10)}$

Decimal	S.M	1's complement	2's complement
7	0111	0111	0111
6	0110	0110	0110
5	0101	0101	0101
4	0100	0100	0100
3	0011	0011	0011
2	0010	0010	0010
1	0001	0001	0001
0	0000	0000	0000
-0	1000	1111	—
-1	1001	1110	1111
-2	1010	1101	1110
-3	1011	1100	1101
-4	1100	1011	1100
-5	1101	1010	1011
-6	1110	1001	1010
-7	1111	1000	1001
-8	—	—	1000

* +ve number is same in all types of representation
* 2's complement → asymmetric range +7 to −8.
* 2's complement addition algorithm is simple