4th Edition
Revised & Updated

A Handbook on

Computer Science & IT

Contains well illustrated formulae & key theory concepts

for

GATE, PSUs

& OTHER COMPETITIVE EXAMS

MADE EASY PUBLICATIONS

MADE EASY Publications Pvt. Ltd.

Corporate Office: 44-A/4, Kalu Sarai (Near Hauz Khas Metro Station), New Delhi-110016

Contact: 9021300500

E-mail: infomep@madeeasy.in

Visit us at: www.madeeasypublications.org

A Handbook on Computer Science & IT

First Edition: 2014
Reprint: 2017
Second Edition: 2018
Reprint: 2019
Reprint: 2020
Third Edition: 2022

Fourth Edition: 2023

Director's Message

B. Singh (Ex. IES)

During the current age of international competition in Science and Technology, the Indian participation through skilled technical professionals have been challenging to the world. Constant efforts and desire to achieve top positions are still required.

I feel every candidate has ability to succeed but competitive environment and quality guidance is required to achieve high level goals. At MADE EASY, we help you to discover your hidden talent and success quotient to achieve your ultimate goals. In my opinion IAS, ESE, GATE & PSU's exams are tool to enter in to main stream of Nation serving. The real application of knowledge and talent starts, after you enter in to the working system. Here in MADE EASY you are also trained to become winner in your life and achieve job satisfaction.

MADE EASY aluminae have shared their winning stories of success and expressed their gratitude towards quality guidance of MADE EASY. Our students have not only secured All India First Ranks in ESE, GATE and PSU entrance examinations but also secured top positions in their career profiles. Now, I invite you to become aluminae of MADE EASY to explore and achieve ultimate goal of your life. I promise to provide you quality guidance with competitive environment which is far advanced and ahead than the reach of other institutions. You will get the guidance, support and inspiration that you need to reach the peak of your career.

I have true desire to serve Society and Nation by way of making easy path of the education for the people of India.

After a long experience of teaching in Computer Science & IT over the period of time MADE EASY team realised that there is a need of good *Handbook* which can provide the crux of Computer Science & IT in a concise form to the student to brush up the formulae and important concepts required for GATE and other competitive examinations. This *handbook* contains all the formulae and important theoretical aspects of Computer Science & IT. It provides much needed revision aid and study guidance before examinations.

B. Singh (Ex. IES)
CMD, MADE EASY Group

A Handbook on

Computer Science & IT

Contents

Contents

■■■■

***** Marked are not in GATE syllabus***

1 Discrete and Engineering Mathematics

I Mathematical Logic

Introduction

- **Proposition:** It is a declarative statement either TRUE or FALSE.
- **Compound Proposition:** It is a proposition formed using the logical operators (Negation ($\neg$), Conjunction ($\wedge$), Disjunction ($\vee$), etc.) with the existing propositions.
- **Logical Operators:**
  - (i) Negation of p : $\neg p$ or $\bar{p}$ or $\sim p$
  - (ii) Conjunction of p and q : $p \wedge q$
  - (iii) Disjunction of p and q : $p \vee q$
  - (iv) Implication/Conditional : $p \rightarrow q$ (if p, then q)
  - (v) Bi-conditional : $p \leftrightarrow q$
- Precedence order of logical operators from high to low: $\neg$, $\wedge$, $\vee$, $\rightarrow$, $\leftrightarrow$
- $P \oplus R = PR' + P'R$, $P \leftrightarrow R = P'R' + PR$
- Number of distinct boolean expression with n variable = 2^{2^n}.
- **Normal form:** PCNF ($\vee$) = (POS = 0), PDFL ($\wedge$) = (SOP = 1)
  Total size = 2^n with n variable.

Note:

- ☑ Converse of $p \rightarrow q$ is : $q \rightarrow p$
- ☑ Inverse of $p \rightarrow q$ is : $\neg p \rightarrow \neg q$
- ☑ Contrapositive of $p \rightarrow q$ is : $\neg q \rightarrow \neg p$

Tautology

If compound proposition is always true then it is tautology.

Example: $p \vee \neg p$

Contradiction

If compound proposition is always false then it is contradiction.

Example: $p \wedge \neg p$

Contingency

Neither tautology nor contradiction.

Example: p

Logical Equivalence

$P \Leftrightarrow Q$ is tautology iff P and Q are logically equivalent.

Functionally Complete

If any formula can be written as an equivalent formula containing only the connectives in a set of operators, then such a set of operators is called as functionally complete.

Example:

$\{\uparrow\}$, $\{\downarrow\}$, $\{\neg, \vee\}$, $\{\neg, \wedge\}$, $\{\neg, \vee, \wedge\}$ are functionally complete (NAND).

Consistent

If $H_1 \wedge H_2 \wedge H_3 \wedge .... \wedge H_n$ is satisfiable then H_1, H_2, ... and H_n are consistent (Tautology, contingency but not contradiction).

Inconsistent

If $H_1 \wedge H_2 \wedge H_3 \wedge .... \wedge H_n$ is unsatisfiable then H_1, H_2, ... and H_n are inconsistent (only contradiction).

- **Valid:** Tautology, **Satisfiable:** Tautology + contingency, Invalid: contradiction + contingency, **Unsatisfiable:** Contradiction
- Sufficient ($\rightarrow$), necessary ($\leftarrow$), but = and, if = when = whenever, is = will = would = are = p unless q.
- $p \rightarrow q \equiv q$ unless $\neg p$ = "q is true unless p is false" either p is not true or q is true.
- p is necessary but not sufficient for $q = (q \rightarrow p) \wedge (p \rightarrow q)' = p \wedge q'$.

Equivalences

$$P \vee (P \wedge Q) \equiv P$$

$$P \rightarrow Q \equiv \neg P \vee Q \equiv \neg Q \rightarrow \neg P$$

$$P \wedge (P \vee Q) \equiv P$$

$$P \leftrightarrow Q \equiv (P \rightarrow Q) \wedge (Q \rightarrow P)$$

$$P \leftrightarrow Q \equiv (P \wedge Q) \vee (\neg P \wedge \neg Q)$$

$$P \leftrightarrow Q \equiv \neg P \leftrightarrow \neg Q$$
$$P \rightarrow (Q \rightarrow R) \equiv (P \wedge Q) \rightarrow R$$
$$\neg(P \leftrightarrow Q) \equiv P \leftrightarrow (\sim Q) \equiv (\sim P) \leftrightarrow Q \equiv P \oplus Q$$
$$(P \rightarrow Q) \wedge (P \rightarrow R) \equiv P \rightarrow (Q \wedge R)$$
$$(P \rightarrow R) \wedge (Q \rightarrow R) \equiv (P \vee Q) \rightarrow R$$
$$(P \rightarrow Q) \vee (P \rightarrow R) \equiv P \rightarrow (Q \vee R)$$
$$(P \rightarrow R) \vee (Q \rightarrow R) \equiv (P \wedge Q) \rightarrow R$$
$$P \vee Q \equiv \neg P \rightarrow Q$$
$$P \wedge Q \equiv \neg(P \rightarrow \neg Q)$$
$$\neg(P \rightarrow Q) \equiv (P \wedge \neg Q)$$

Identity Laws: $(i)\ P \wedge T = P$, $(ii)\ P \vee F = P$

Domination Laws: $(i)\ P \vee T = T$, $(ii)\ P \wedge F = F$

Idempotent Laws: $(i)\ P \wedge P = P$, $(ii)\ P \vee P = P$

Commutative Laws:

$(i)\ P \vee Q = Q \vee P$, $(ii)\ P \wedge Q = Q \wedge P$

Associative Laws:

$(i)\ (P \vee Q) \vee R = P \vee (Q \vee R)$

$(ii)\ (P \wedge Q) \wedge R = P \wedge (Q \wedge R)$

Distributive Laws:

$(i)\ P \vee (Q \wedge R) = (P \vee Q) \wedge (P \vee R)$

$(ii)\ P \wedge (Q \vee R) = (P \wedge Q) \vee (P \wedge R)$

Demorgan's Laws:

$(i)\ \neg(P \wedge Q) = \neg P \vee \neg Q$

$(ii)\ \neg(P \vee Q) = \neg P \wedge \neg Q$

Absorption Laws:

$(i)\ P \vee (P \wedge Q) = P$, $(ii)\ P \wedge (P \vee Q) = P$

Negation Laws:

$(i)\ P \vee \neg P = T$, $(ii)\ P \wedge \neg P = F$

Double Negation Laws: $\neg(\neg P) = P$

Rules of Inference (Tautological Implications)

Simplification:

$$(P \wedge Q) \Rightarrow P$$
$$(P \wedge Q) \Rightarrow Q$$

Addition:

$$P \Rightarrow (P \vee Q)$$
$$Q \Rightarrow (P \vee Q)$$

Disjunctive Syllogism: $(\sim P, P \vee Q) \Rightarrow Q$

Modus Ponens: $(P, P \rightarrow Q) \Rightarrow Q$

Modus Tollens: $(\sim Q, P \rightarrow Q) \Rightarrow \sim P$

Hypothetical Syllogism: $(P \rightarrow Q, Q \rightarrow R) \Rightarrow (P \rightarrow R)$

Conjunctive Syllogism: $((P \vee Q), P) \Rightarrow \sim Q$

Dilemma: $(P \vee Q, P \rightarrow R, Q \rightarrow R) \Rightarrow R$

Constructive Dilemma: $(P \vee Q, P \rightarrow R, Q \rightarrow S) \Rightarrow R \vee S$

Destructive Dilemma: $(\sim R \vee \sim S, P \rightarrow R, Q \rightarrow S) \Rightarrow \sim P \vee \sim Q$

Other rules:

$$\sim P \Rightarrow (P \rightarrow Q)$$

$$Q \Rightarrow (P \rightarrow Q)$$

$$\sim(P \rightarrow Q) \Rightarrow P$$

$$\sim(P \rightarrow Q) \Rightarrow \sim Q$$

Exactly one = $\exists!$ or $\exists x\,[P(x) \wedge P(y) \Rightarrow y = x]$,

$$\forall x[\exists y\,(B(x, y) \wedge (B(x, z) \rightarrow y = z)$$

$p \Rightarrow q \Rightarrow r \equiv (p \wedge q) \rightarrow r = q \Rightarrow (p \Rightarrow r)$

Principle Conjunctive Normal Form (PCNF)

Product of sums (max term)

PCNF: $[P(x_1) \vee P(x_2)] \wedge [P(x_3) \vee P(x_4)]$

Principle Disjunctive Normal Form (PDNF)

Sums of products (min term)

PDNF: $[P(x_1) \wedge P(x_2)] \vee [P(x_3) \wedge P(x_4)]$

Number of non equivalent propositional functions with n-propositional variables are = 2^{2^n}.

- $\forall x\,(\alpha \rightarrow \beta) \Rightarrow (\forall x\,\alpha \Rightarrow \forall x\,\beta)$ true only with properties always use and but not $\rightarrow$.

Predicate Logic

Quantifiers

- **Universal ($\forall$) :** "for all" or "for every"
- **Existential ($\exists$) :** "there exist"

Predicates

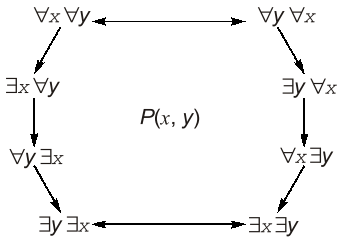
- $P(x)$: Propositional statement with one variable.
- $Q(x, y)$: Propositional statement with two variables.

Note:

☑ $\neg\exists x\, P(x) = \forall x\, \neg P(x)$

☑ $\neg\forall x\, P(x) = \exists x\, \neg P(x)$

Logical Equivalences



1. $\forall x\,[P(x) \wedge Q(x)] \equiv \forall x\, P(x) \wedge \forall x\, Q(x)$
2. $\exists x[P(x) \vee Q(x)] \equiv \exists x P(x) \vee \exists x\, Q(x)$
3. $\forall x\,(P(x) \vee Q) \equiv \forall x\, P(x) \vee Q$
4. $\forall x\,(P(x) \wedge Q) \equiv \forall x\, P(x) \wedge Q$
5. $\exists x\,(P(x) \vee Q) \equiv \exists x\, P(x) \vee Q$
6. $\exists x\,(P(x) \wedge Q) \equiv \exists x\, P(x) \wedge Q$
7. $\forall x\, P(x) \wedge \exists y Q(y) \equiv \forall x \exists y[P(x) \wedge Q(y)]$
8. $\forall x\, P(x) \vee \exists y Q(y) \equiv \forall x \exists y[P(x) \vee Q(y)]$

II Combinatorics

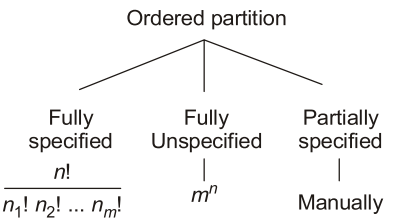
Permutations (Ordered Selection/Arrangement)

- The number of permutations of n-objects:
  1. Taken 'r' at a time = ${}^{n}P_{r} = P(n, r) = (n)_{r}$ (arrangement).
  2. Taken 'r' at a time = n^{r} (with repetition) (arrangement).
  3. Taken all at a time = $n!$

4. Taken not more than 'r' = $\frac{(n^{r+1}-1)}{n-1}$ (with repetition).
5. Taken 'r' (atleast one repeated) = All – none = $n^r - {}^nP_r$
6. Taken all at a time, in which 'r' of them are alike (identical) = $\frac{n!}{r!}$.
7. Taken all at a time, in which n_1 are alike, n_2 are alike, ..., n_r are alike
   $= \frac{n!}{n_1!\, n_2! \ldots n_r!}$
8. Taken 'r' at a time, in which m-particular objects are:
   (a) Never included = ${}^{(n-m)}P_r$.
   (b) Always included ($n \geq m$ and $r \geq m$) = ${}^{(n-m)}P_{(r-m)} \cdot {}^{r}P_m$.
9. ${}^nP_r = P(n-1, r) + r \cdot P(n-1, r-1)$
10. ${}^nP_r = r! \cdot {}^nC_r = n \times {}^{(n-1)}P_{(r-1)}$
11. n types of objects (each of infinity in number)
    $\underbrace{n_1 + n_2 + n_3 + n_4 + \ldots\ldots + n_n}_{n} = r = {}^{n-1+r}C_r$
12. m boys and n girls are arrange in straight line = $(m + n - 1)!$

- **The Number of Circular Permutations of n object:**
  1. Taken all at a time = $(n-1)!$; (If not directed)
     $= \frac{(n-1)!}{2}$; (If direction (clockwise/anti-clockwise) is given)
  2. Taken 'r' at a time = $\frac{{}^nP_r}{r}$; (If not directed), and $\frac{1}{2} \cdot \frac{{}^nP_r}{r}$; (if directed)
  3. Necklace with m identical beads, n identical beads in circular number of way = $\frac{(m+n-1)}{m! \times n! \times 2!}$.

Distinct ↓ Distinct ↓ Ordered partition	Distinct ↓ Indistinct ↓ Unordered	Indistinct ↓ Distinct ↓ Ball in box
$\frac{n!}{n_1!\, n_2! \ldots n_n!}$	$\frac{n!}{(G!)^P}$	$\frac{n!}{(G!)^P \times P!}$



Digital Logic

2

I Logic Functions

Logic Gates

- OR, AND, NOT are Basic Gates
- NAND, NOR are Universal Gates
- EX-OR, EX-NOR are Arithmetic Gates

NOT Gate

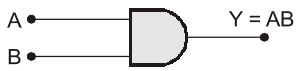
- Also referred to as "Inversion" or "Complementation".

Symbol and Truth table:

Inputs A	Output Y = $\overline{A}$
0	1
1	0

AND Gate

Symbol and Truth table:



Inputs		Output
A	B	Y = AB
0	0	0
0	1	0
1	0	0
1	1	1

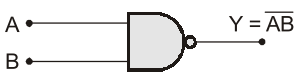
OR Gate

Symbol and Truth table:

Inputs		Output
A	B	Y = A + B
0	0	0
0	1	1
1	0	1
1	1	1

NAND Gate

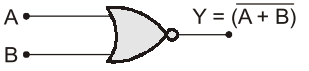
Symbol and Truth table:



Inputs		Output
A	B	Y = $\overline{AB}$
0	0	1
0	1	1
1	0	1
1	1	0

NOR Gate

Symbol and Truth table:



Inputs		Output
A	B	Y = $\overline{(A + B)}$
0	0	1
0	1	0
1	0	0
1	1	0

EX-OR Gate

- It is also called "stair case switch".
- Symbol and Truth table:

Inputs		Output
A	B	Y = A ⊕ B
0	0	0
0	1	1
1	0	1
1	1	0

- Boolean function of 2-input EXOR operation:

$$Y = A \oplus B = \overline{A}B + A\overline{B}$$

(*i*) It acts like as an "odd number of 1's detector in the input".

(*ii*) It is mostly used in "parity generation and detection".

(*iii*) When both the inputs are same, then output becomes LOW or Logic '0'.

(*iv*) When both the inputs are different, then output becomes HIGH or Logic '1'.

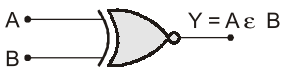
(*v*) $A \oplus A = 0, \quad A \oplus 0 = A$

$A \oplus \overline{A} = 1, \quad A \oplus 1 = \overline{A}$

- $A \oplus A \oplus A \oplus \ldots$ upto n terms = 0, when n = even
- $A \oplus A \oplus A \oplus \ldots$ upto n terms = A, when n = odd

EX-NOR Gate

- It acts like as an "even number of 1's detector".
- Symbol and Truth table:



Inputs		Output
A	B	Y = A ε B
0	0	1
0	1	0
1	0	0
1	1	1

- Boolean function of 2-input EX-NOR operation:

$$Y = A \odot B = \overline{A \oplus B} = (\overline{\bar{A}B + A\bar{B}}) = AB + \bar{A}\bar{B}$$

(*i*) When both the inputs are same, then output becomes HIGH or Logic '1'.

(*ii*) When both the inputs are different, then output becomes LOW or Logic '0'.

(*iii*) $A \odot A = 1, \quad A \odot 1 = A$

$A \odot \bar{A} = 0, \quad A \odot 0 = \bar{A}$

- $A \odot A \odot A \odot \ldots$ upto n terms = 1, when n is even
- $A \odot A \odot A \odot \ldots$ upto n terms = A, when n is odd

Note:

☑ $\bar{A} \oplus B = A \oplus \bar{B} = A \odot B$

☑ $\bar{A} \odot B = A \odot \bar{B} = A \oplus B$

☑ For odd number of inputs EX-OR and EX-NOR are same and for even number of variables they are complements to each other.

i.e. $A \oplus B \oplus C = A \odot B \odot C$

as $A \oplus B \oplus C = \overline{A \oplus B}\, C + (A \oplus B)\bar{C}$

$= (A \odot B)\, C + (A \oplus B)\bar{C}$

$= (A \odot B)C + \overline{A \odot B}\, \bar{C}$

$= A \odot B \odot C$

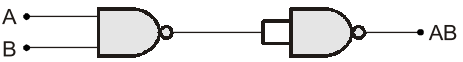
☑ $\overline{A \oplus B \oplus C} = A \odot B \oplus C = A \oplus B \odot C$

☑ $\overline{A \odot B \odot C} = A \oplus B \odot C = A \odot B \oplus C$

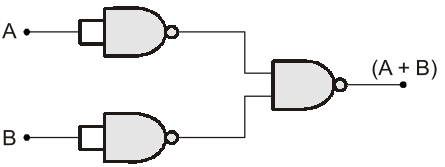
- ☑ $A \oplus B \oplus AB = A + B$
- ☑ EX-OR and EX-NOR are also called arithmetic gates as they are used in addition, subtraction, comparator circuits.

Implementation of Gates using NAND

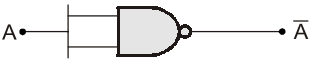
- AND gate:



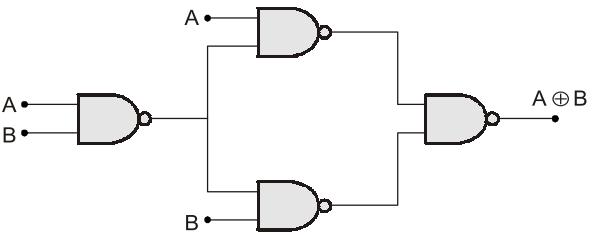
- OR gate:



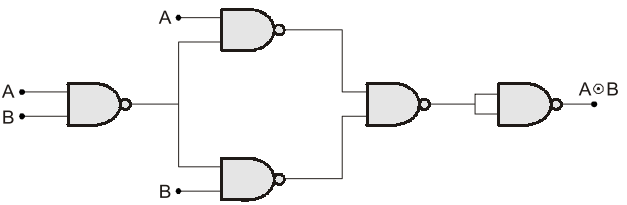
- NOT gate:



- EX-OR gate:

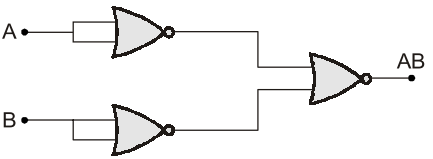


- EX-NOR Gate:

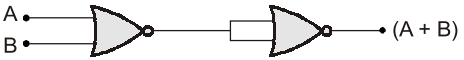


Implementation of Gates using NOR

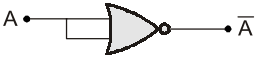
- AND gate:



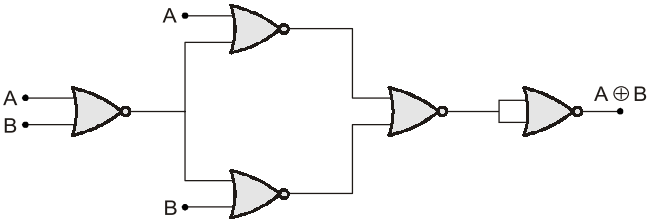
- OR gate:



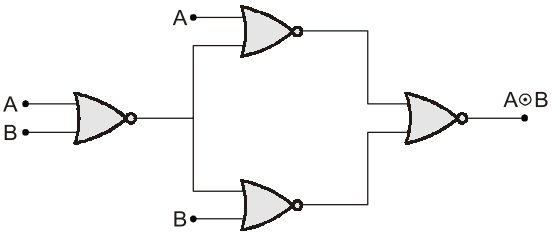
- NOT gate:



- EX-OR gate:



- EX-NOR gate:



Note:

☑ Number of NAND and NOR gates needed to implement other logic gates is shown in the following table.

Logic gate	No. of NAND gates	No. of NOR gates
NOT	1	1
AND	2	3
OR	3	2
EX OR	4	5
EX NOR	5	4

☑ Alternative Symbols of Gates

1. Bubbled – OR gate ≡ NAND gate
2. Bubbled – NAND gate ≡ OR gate
3. Bubble – NOR gate ≡ AND gate
4. Bubbled – AND gate ≡ NOR gate

Computer Organization and Architecture

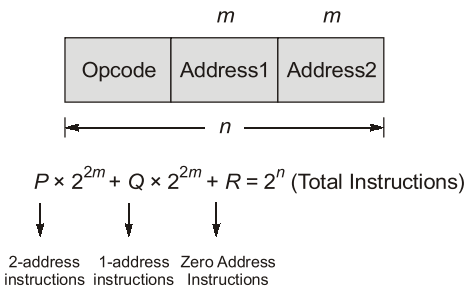
3

I Machine Instructions and Addressing Modes

Machine Instructions

Types of instructions: Zero address instructions, one address instructions, and two address instructions.

- Zero address instructions : **NOP**
- One address instructions : **PUSH, POP**
- Two address instructions : **ADD, MULT, SUB**



Addressing Modes

- Addressing modes shows the way where the required object is present. The object may be an instruction or data.
- Addressing Modes are basically classified in two types:
  - (a) **Sequential Control Flow Addressing Modes:** When the program is stored in the sequential memory locations, the program counter itself points the next instruction address, therefore such Addressing Modes are focused on data.
  - (b) **Transfer of Control Flow Addressing Modes:** When the program is stored in the random memory locations (using structured programming) there is a need of special instructions and addressing modes in order to calculate the next instruction address.

Sequential Control Flow Addressing Modes

1. Register Based Addressing Modes:

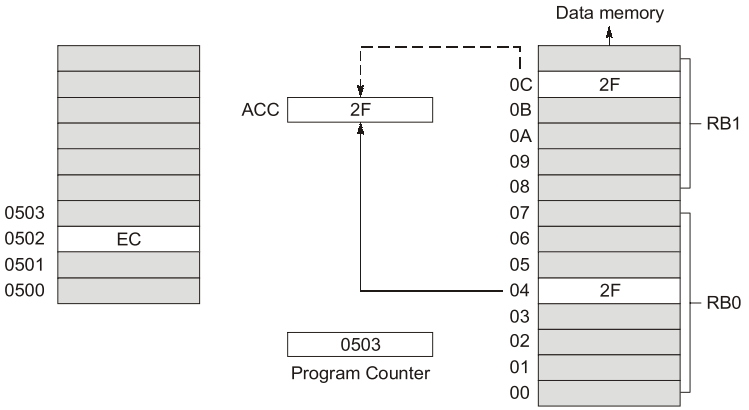
   (*i*) These Addressing Modes are used to access the data when it is available in the registers.

   (*ii*) **Register/Register Direct Addressing Mode:** This Addressing Mode is used to access the local variables. In this mode the data is present in the register, that register address is available in the address field of the instruction. Therefore, the effective address is equal to address field value. Data = [EA] = [Register Name]

   Example:

Register Direct Addressing Mode

Instruction	Opcode	Bytes	Cycles
MOV A, R4	ECH	1	1



2. **Memory Based Addressing Modes:** When the data is available in the memory, different memory based addressing modes are used to access the data. Under this, the EA is always the memory address.

   (*i*) **Implied/Implicit Addressing Modes:** In this mode the data is available in the opcode itself. Therefore, there is no effective address.

   Example: Compliment Accumulator (CMA) and all zero address instructions.

   (*ii*) **Immediate Addressing Mode:**

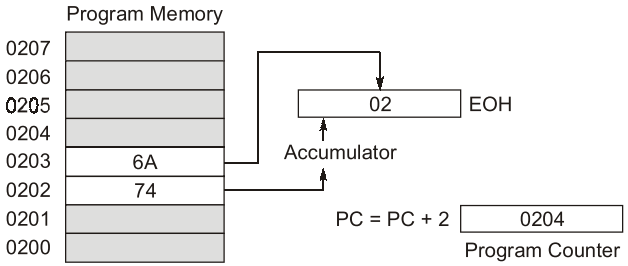
   - This mode is used to access the constants or to initialize registers to a constant value.
   - Here the data is present in the address field of the instruction.

- The range of values initialized is limited by the size of the address field.
- If the address field size is *n*-bit, the possible range of immediate constants or data is: 0 to $2^n - 1$.

Example:

Immediate Addressing Mode

Instruction	Opcode	Bytes	Cycles
MOV A, #6AH	74H	2	1



(*iii*) Direct/Absolute Addressing Mode:

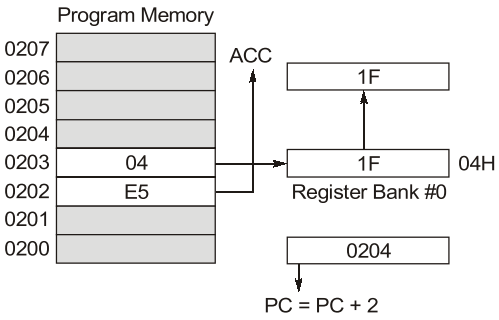
- Used to access the static variables.
- The data is present in the memory, that memory cell address is present in the address field of the instruction.

  Data = [EA] = [Memory Address]
- One memory reference is required to read or write the data by using the direct addressing mode.

Example:

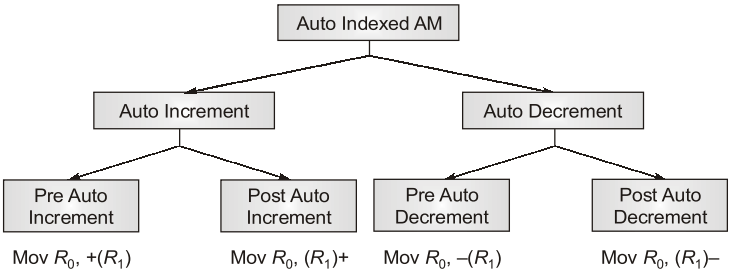
Direct Addressing Mode

Instruction	Opcode	Bytes	Cycles
MOV A, #04H	E5	2	1



(*iv*) Auto Indexed Addressing Mode:

- This mode is used to access the linear array elements. Thus, "base address" is required to access the data.
- The base address is maintained in the base register. Therefore the EA is Base Register ± Step Size.
- Step size is dependency on the amount of the data to be accessed from the memory. Data = [EA] = [[Base Register] ± Step size].



(*v*) Indirect Addressing Mode: (Array as parameter)

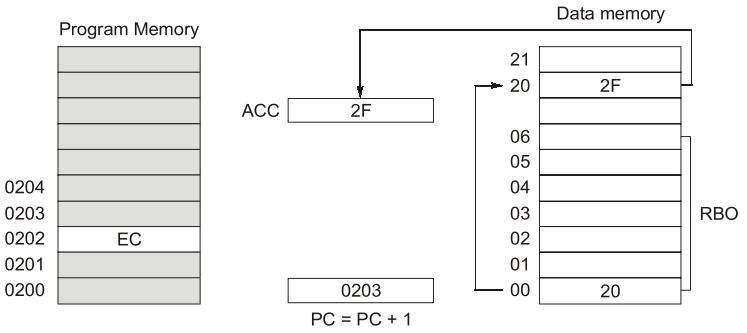
- Used to implement the pointers. The EA is available in either register or memory.
- This mode is divided into two types:

  (a) Register Indirect Addressing Mode: Here, the EA is present in the address register, that register name is available in the address field of the instruction.

  EA = [Address Field Value] ⇒ [Register Name]

Register Indirect Addressing Mode

Instruction	Opcode	Bytes	Cycles
MOV A, @R0	E6H	1	1



(b) Memory Indirect Addressing Mode: Here, the EA is present in the memory, that memory address is available in the address field of the instruction.

EA = [Address field value] ⇒ [Memory address]

Data = [EA] = [[Memory address]]

- Two memory references are required in memory indirect Addressing Mode.

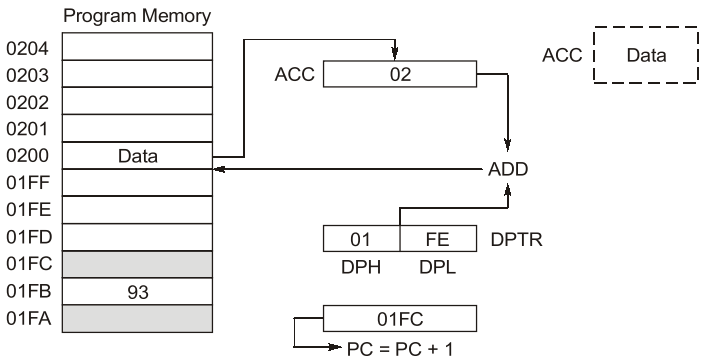
(*vi*) Indexed Addressing Mode:

- Allows to implement array indexing.

  EA = Base Address + Index Value
- The register that holds the index value is "Register Indexed Addressing Mode", it is used to access the Random array element.
- In Indirect Index Addressing Mode the base address is present in the memory, that memory address is present in the address field of the instruction.

Indexed Addressing Mode

Instruction	Opcode	Bytes	Cycles
MOVC A, @A +DPTR	93H	1	2



Transfer of Control Flow Addressing Mode

- During the execution of selection statements (if, then, else, goto, switch), iterative statements (For loop, while loop, do while loop) and subprogram concept, the control is transferred from one location to another location.
- Transfer-of-Control operations can be implemented by using three possible mnemonics: (*i*) Jump (*ii*) Branch and (*iii*) Skip.
- Transfer-of-Control instruction is divided into two types: