AKSHAYA INSTITUTE OF TECHNOLOGY,TUMKUR
Department of Electronics & Communication Engineering

Module 2 and 3 Notes for

"Addressing Modes, Instruction Sets, Stack Pointer and Programs"

[BEC405A]

Prepared by:

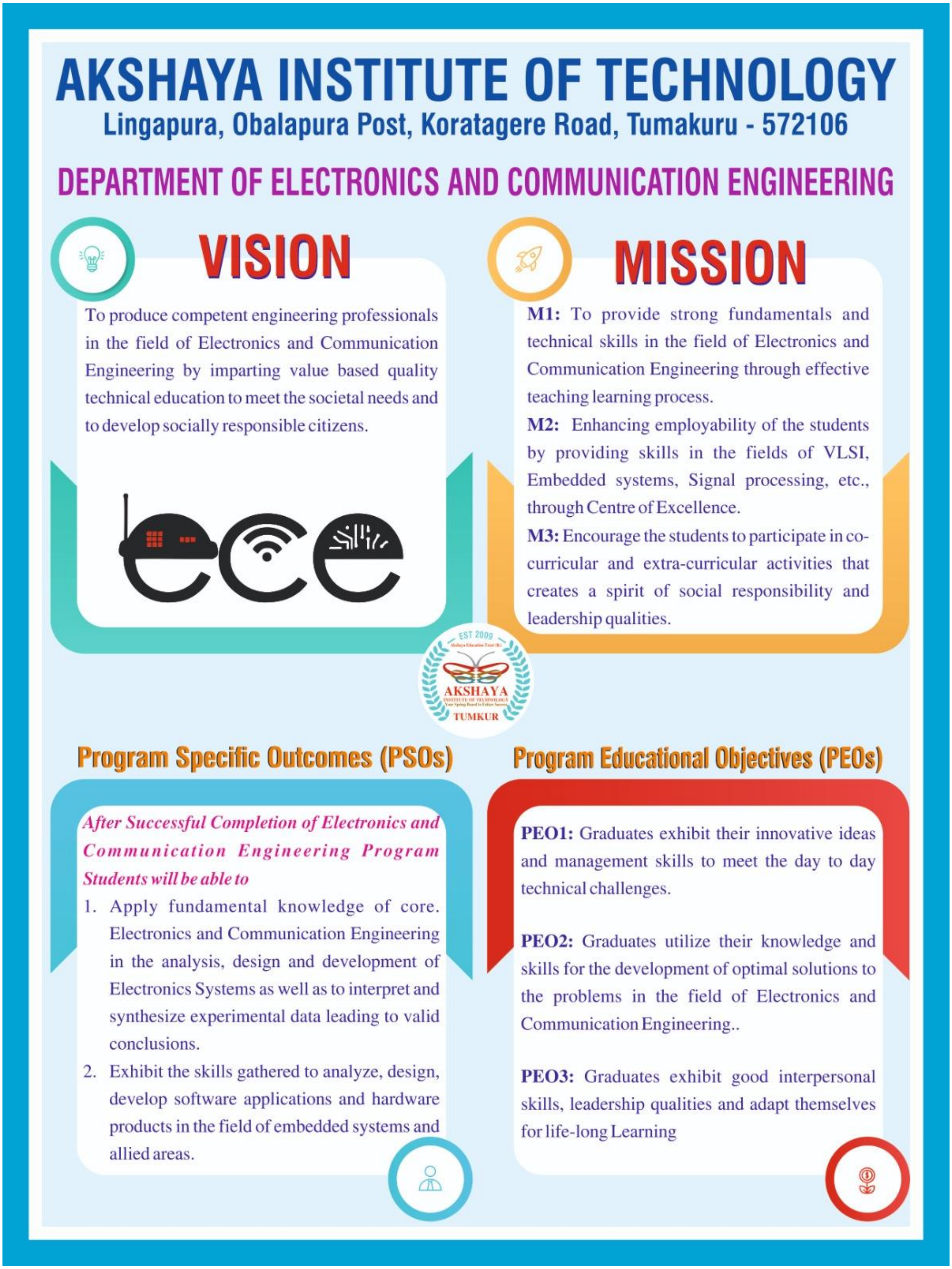
Dr. Vijaya Kumar H R

Associate Professor

Department of ECE.

Akshaya Institute of Technology

Tumakuru



AKSHAYA INSTITUTE OF TECHNOLOGY

Lingapura, Obalapura Post, Koratagere Road, Tumakuru - 572106

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

VISION

To produce competent engineering professionals in the field of Electronics and Communication Engineering by imparting value based quality technical education to meet the societal needs and to develop socially responsible citizens.

MISSION

M1: To provide strong fundamentals and technical skills in the field of Electronics and Communication Engineering through effective teaching learning process.

M2: Enhancing employability of the students by providing skills in the fields of VLSI, Embedded systems, Signal processing, etc., through Centre of Excellence.

M3: Encourage the students to participate in co-curricular and extra-curricular activities that creates a spirit of social responsibility and leadership qualities.

Program Specific Outcomes (PSOs)

After Successful Completion of Electronics and Communication Engineering Program Students will be able to

1. Apply fundamental knowledge of core. Electronics and Communication Engineering in the analysis, design and development of Electronics Systems as well as to interpret and synthesize experimental data leading to valid conclusions.
2. Exhibit the skills gathered to analyze, design, develop software applications and hardware products in the field of embedded systems and allied areas.

Program Educational Objectives (PEOs)

PEO1: Graduates exhibit their innovative ideas and management skills to meet the day to day technical challenges.

PEO2: Graduates utilize their knowledge and skills for the development of optimal solutions to the problems in the field of Electronics and Communication Engineering..

PEO3: Graduates exhibit good interpersonal skills, leadership qualities and adapt themselves for life-long Learning

MICROCONTROLLERS		Semester	4
Course Code	BEC405A	CIE Marks	50
Teaching Hours/Week(L:T:P)	3:0:0	SEE Marks	50
Total Hours of Pedagogy	40	Total Marks	100
Credits	03	Exam Hours	3
Examination type(SEE)	Theory		

Course objectives:

This course will enable students to:

- Understand the difference between Microprocessor and Microcontroller and embedded microcontrollers.
- Analyze the basic architecture of 8051microcontroller.
- Program 8051 microcontroller using Assembly Language and C.
- Understand the operation and use of inbuilt Timers/Counters and Serial port of 8051
- Understand the interrupt structure of 8051 and Interfacing I/O devices using I/O ports of 8051.

Teaching-Learning Process(General Instructions)

The samples strategies, which the teacher can use to accelerate the attainment of the various course outcomes are listed in the following:

1. Lecture method (L) does not mean only the traditional lecture method, but a different type of teaching method may be adopted to develop the outcomes.
2. Show Video/animation films to explain the functioning of various techniques.
3. Encourage collaborative(Group)Learning in the class
4. Ask at least three HOTS(Higher-order Thinking) questions in the class, which promotes critical thinking
5. Adopt Problem Based Learning (PBL), which fosters students' Analytical kills, develop thinking skills such as the ability to evaluate, generalize, and analyze information rather than simply recall it.
6. Show the different ways to solve the same problem and encourage the students to come up with their own creative ways to solve them.
7. Discuss how every concept can be applied to the real world and when that's possible, it helps improve the students' understanding.
   Give Programming Assignments.

	Level
Module-1 (8 Hrs)	
Microcontroller: Microprocessor Vs Microcontroller, Micro controller & Embedded Processors, Processor Architectures-Harvard Vs Princeton & RISC Vs CISC , 8051 Architecture- Registers, Pin diagram, I/O ports functions, Internal Memory organization. External Memory (ROM & RAM) interfacing. (Text book 1-1.1,Text book 2-1.0,1.1,3.0,3.1,3.2,3.3 Text book 3-Pg 5-9)	**L1,L2**
Module-2 (8 Hrs)	
Instruction Set: 8051 Addressing Modes, Data Transfer Instructions, Arithmetic instructions, Logical Instructions, Jump & Call Instructions Stack & Subroutine Instructions of 8051 (with examples in assembly Language). (Text book 2- Chapter 5,6,7,8, Additional reading Refer Textbook 3. Chapter 3 for complete understanding of instructions with	**L1,L2**

Module-3 (8 Hrs)	
Timers/Counters & Serial port programming: Basics of Timers & Counters, Data types & Time delay in the 8051 using C, Programming 8051 Timers, Mode 1 & Mode 2 Programming, Counter Programming (Assembly Language only). (Text book 2- 3.4, Text book 1- 7.1, 9.1,9.2) Basics of Serial Communication, 8051 Connection to RS232, Programming the 8051 to transfer data serially & to receive data serially using C.(Text book 2- 3.5, Text book 1- 10.1,10.2,10.3 except assembly language programs, 10.5)	**L1,L2, L3**
Module-4 (8 Hrs)	
Interrupt Programming: Basics of Interrupts, 8051 Interrupts, Programming Timer Interrupts, Programming Serial Communication Interrupts, Interrupt Priority in 8051(Assembly Language only) (Text book 2- 3.6, Text book 1- 11.1,11.2,11.4, 11.5)	**L1,L2, L3**
Module-5 (8 Hrs)	
I/O Port Interfacing & Programming: I/O Programming in 8051 C, LCD interfacing, DAC 0808 Interfacing, ADC 0804 interfacing, Stepper motor interfacing, DC motor control & Pulse Width Modulation (PWM) using C only. (Text book 1- 7.2, 12.1, 13.1, 13.2, 17.2, 17.3)	**L1, L2, L3**

Course outcome (Course Skill Set)

At the end of the course, students will be able to:

1. Describe the difference between Microprocessor and Microcontroller, Types of Processor Architectures and Architecture of 8051Microcontroller.
2. Discuss the types of 8051 Microcontroller Addressing modes & Instructions with Assembly Language Programs.
3. Explain the programming operation of Timers/Counters and Serial port of 8051 Microcontroller.
4. Illustrate the Interrupt Structure of 8051 Microcontroller & its programming.
5. Develop C programs to interface I/O devices with 8051 Microcontroller.

Assessment Details (both CIE and SEE)

The weightage of Continuous Internal Evaluation (CIE) is 50% and for Semester End Exam (SEE)is 50%.The minimum passing mark for the CIE is 40%of the maximum marks(20marks out of 50) and for the SEE minimum passing mark is 35% of the maximum marks (18 out of 50 marks). The student is declared as a pass in the course if he/she secures a minimum of 40% (40 marks out of 100) in the sum total of the CIE (Continuous Internal Evaluation) and SEE (Semester End Examination) taken together.

Continuous Internal Evaluation:

☐ There are 25marks for the CIE's Assignment component and 25 for the Internal Assessment Test component.

☐ Each test shall be conducted for 25marks.The first test will be administered after 40-50% of the coverage of the syllabus, and the second test will be administered after 85-90% of the coverage of the syllabus. The average of the two tests shall be scaled down to 25 marks

☐ Any two assignment methods mentioned in the 22OB2.4, if an assignment is project-based then only one assignment for the courses hall be planned. The schedule for assignments shall be planned properly by the course teacher. The teacher should not conduct two assignments at the end of the semester if two assignments are planned. Each assignment shall be conducted for 25 marks. (If two assignments are conducted then the sum of the two assignments shall be scaled down to 25 marks)

☐ The final CIE marks of the course out of 50 will be the sum of the scale-down marks of tests and assignment/s marks.

Internal Assessment Test question paper is designed to attain the different levels of Bloom's taxonomy as per the outcome defined for the course.

Semester-End Examination:

Theory SEE will be conducted by University as per the scheduled timetable, with common question papers for the course **(duration 03 hours).**

1. The question paper will have ten questions. Each question is set for 20marks.
2. There will be 2questions from each module. Each of the two questions under a module (with a maximum of 3 sub-questions), **should have a mix of topics** under that module.
3. The students have to answer 5 full questions, selecting one full question from each module.
4. Marks scored shall be proportionally reduced to50 marks

Suggested Learning Resources:

TEXT BOOKS

1. The "8051 Microcontroller and Embedded Systems – Using Assembly and C", Muhammad Ali Mazidi and Janice Gillespie Mazidi and Rollind. Mckinlay; Phi, 2006 / Pearson, 2006.
2. "The 8051 Microcontroller", Kenneth j. Ayala, 3rd edition, Thomson/Cengage Learning.
3. "Programming And Customizing The 8051 Microcontroller".,Myke Predko Tata Mc Graw-Hill Edition 1999 (reprint 2003).

REFERENCEBOOKS:

1. "The 8051 Microcontroller Based Embedded Systems", Manish K Patel, McGraw Hill, 2014, ISBN: 978-93-329-0125-4.
2. "Microcontrollers: Architecture, Programming, Interfacing and System Design", Raj Kamal, Pearson Education, 2005.

Web links and Video Lectures(e-Resources):

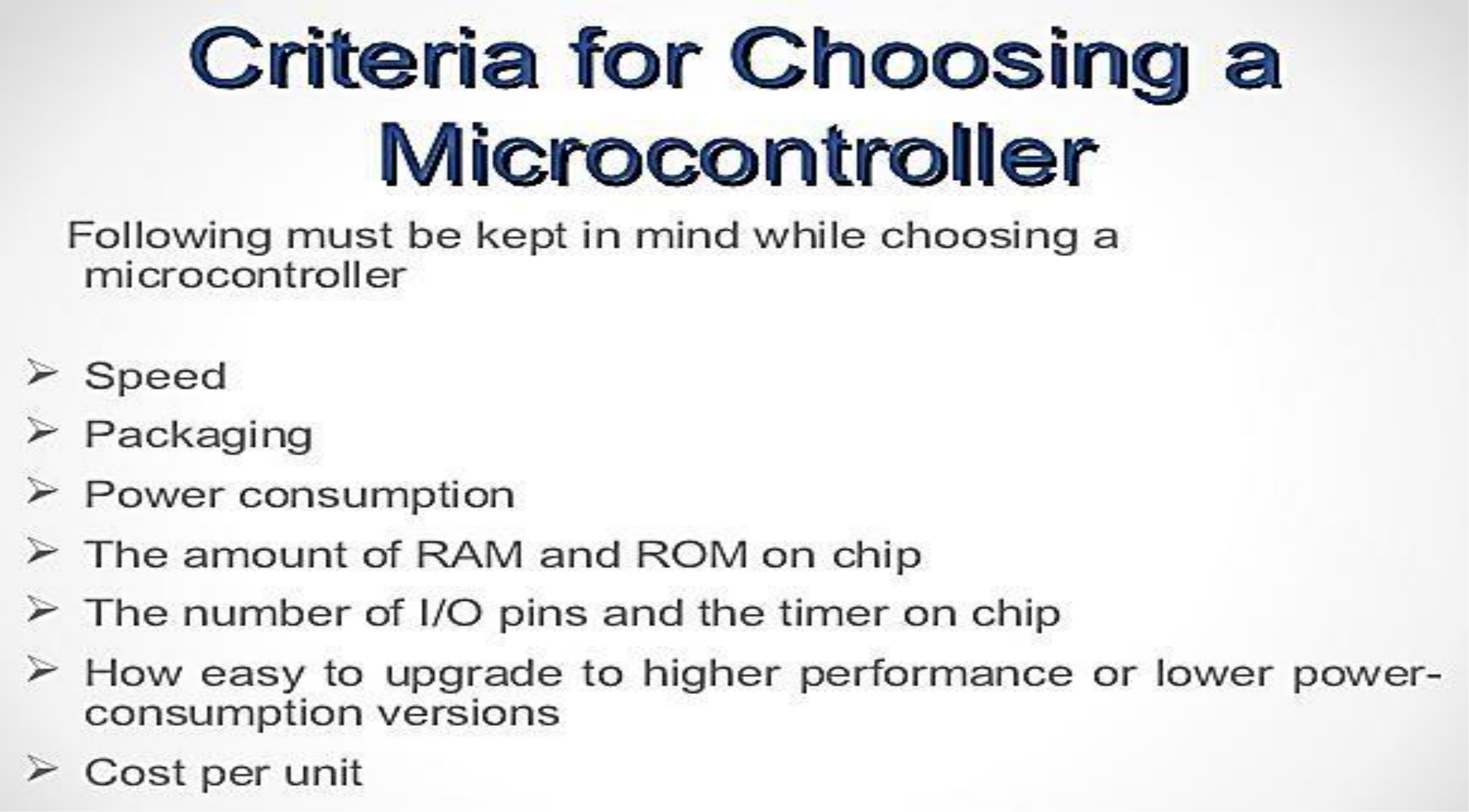
https://youtu.be/pA6K5NgWTow?si=zQqqgXQq50dVL_-s

MODULE 2 and Module 3

ADDRESSING MODES AND INSTRUCTION SETS
Stack Pointer and Programs

By

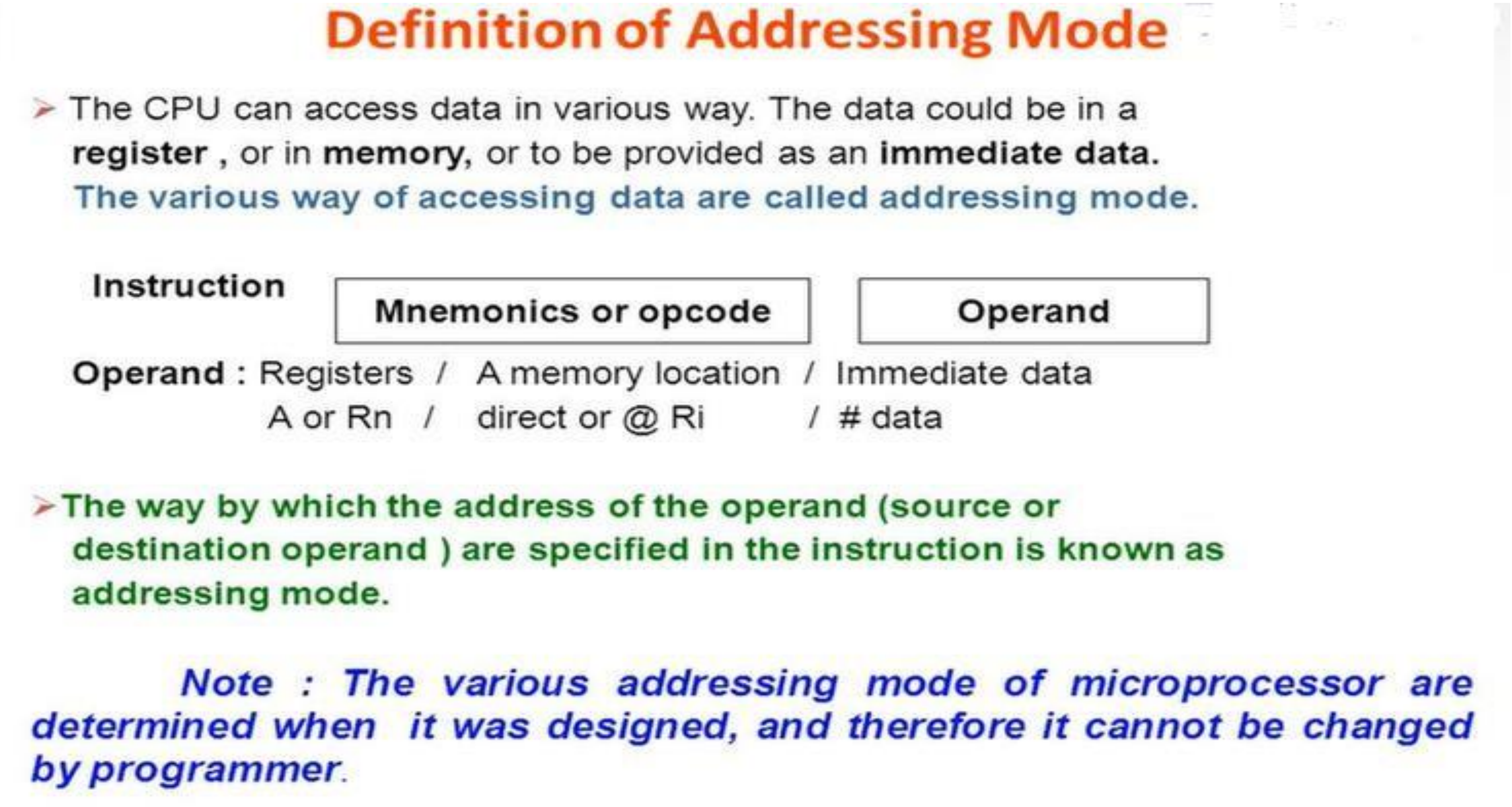
Dr. Vijaya Kumar H R
Associate Professor, Dept. of ECE
AIT, Tumkur, Karnataka



Criteria for Choosing a Microcontroller

Following must be kept in mind while choosing a microcontroller

- Speed
- Packaging
- Power consumption
- The amount of RAM and ROM on chip
- The number of I/O pins and the timer on chip
- How easy to upgrade to higher performance or lower power-consumption versions
- Cost per unit



Definition of Addressing Mode

- The CPU can access data in various way. The data could be in a **register** , or in **memory,** or to be provided as an **immediate data.** **The various way of accessing data are called addressing mode.**

Instruction **Mnemonics or opcode** **Operand**

Operand : Registers / A memory location / Immediate data

A or Rn / direct or @ Ri / # data

- **The way by which the address of the operand (source or destination operand) are specified in the instruction is known as addressing mode.**

Note : The various addressing mode of microprocessor are determined when it was designed, and therefore it cannot be changed by programmer.

Addressing Mode of 8051

- 8051 micro controller supports the following addressing modes.

1) Immediate addressing,

2) Register addressing

3) Direct addressing

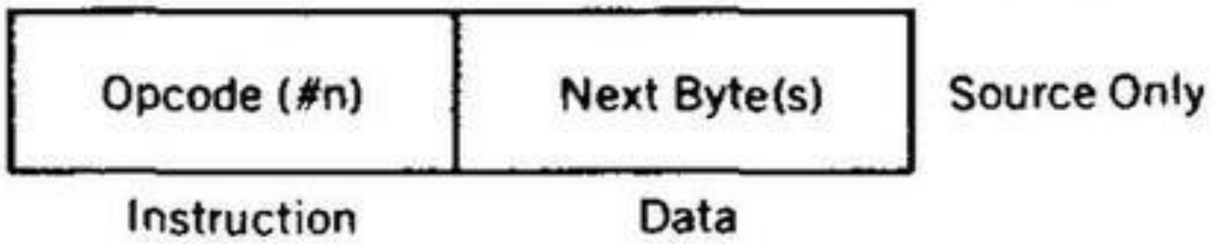
4) Indirect addressing,

5) Index addressing mode (External Data Moves)

Immediate Addressing

- In immediate addressing mode the operand is specified within the instruction itself. In this "data" is part of the instruction.

Note : ***The mnemonic for immediate data is the pound sign (#). '#' sign is used in the instruction to indicate the "immediate" data***

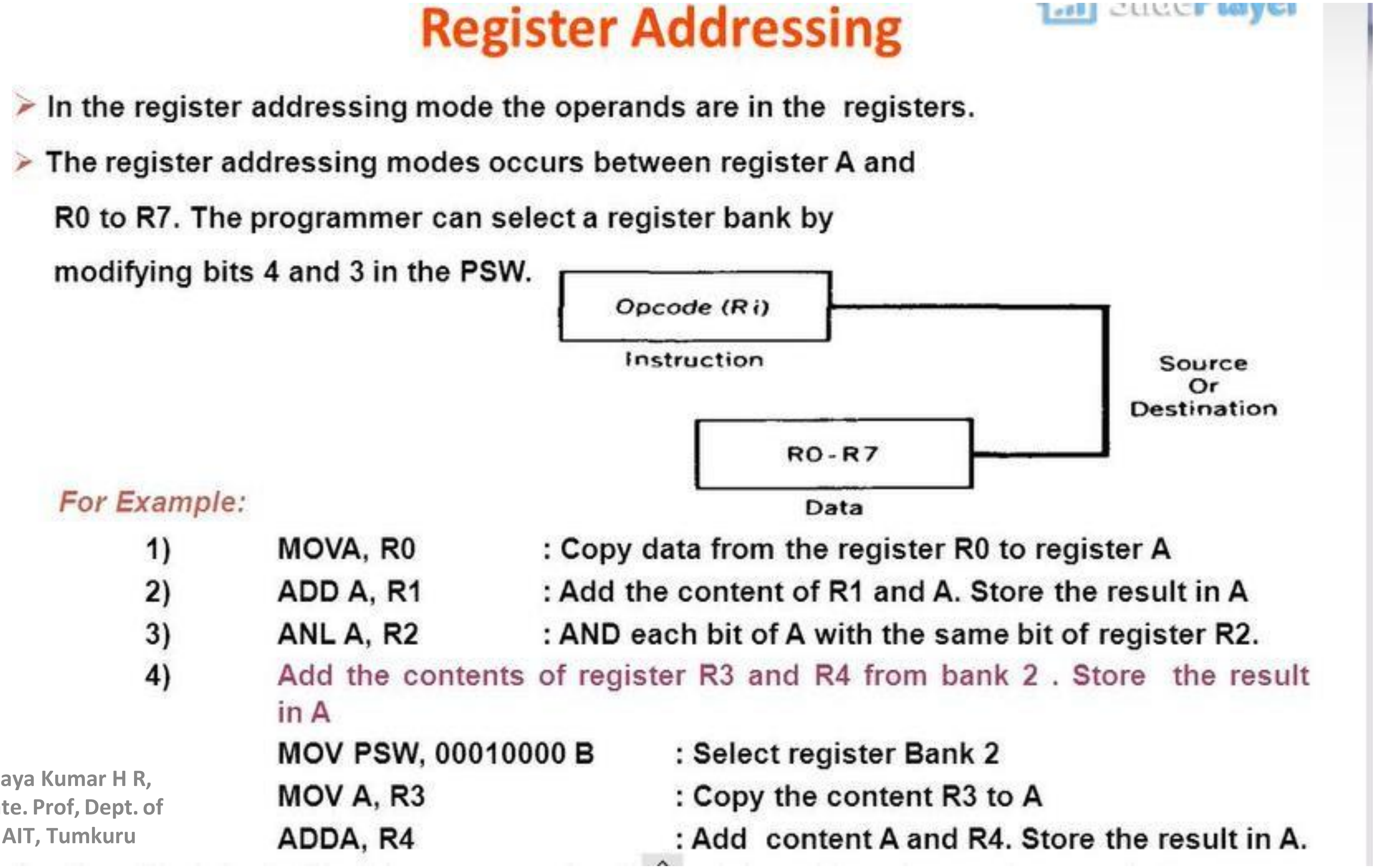


For Example:

1) **MOV A, #n** : Move 8 bit number n(n= 00 to FFH) immediately to accumulator.
   MOV A, #30H

2) **MOV Rr, #n** : Move 8 bit number n(n= 00 to FFH) immediately to Rr(Rr is R0 to R8 of current register bank,B,P0 to P3)
   MOV B,#30H MOV R0,#30H MOV R7,#30H, MOV P0,#30H MOV P3,#30H

3) **MOV DPTR, #nn** : Move the immediate 16 bit data nn(nn=0000 to FFFFH) to the DPTR
   MOV DPTR, #1234H



Register Addressing

- In the register addressing mode the operands are in the registers.
- The register addressing modes occurs between register A and R0 to R7. The programmer can select a register bank by modifying bits 4 and 3 in the PSW.

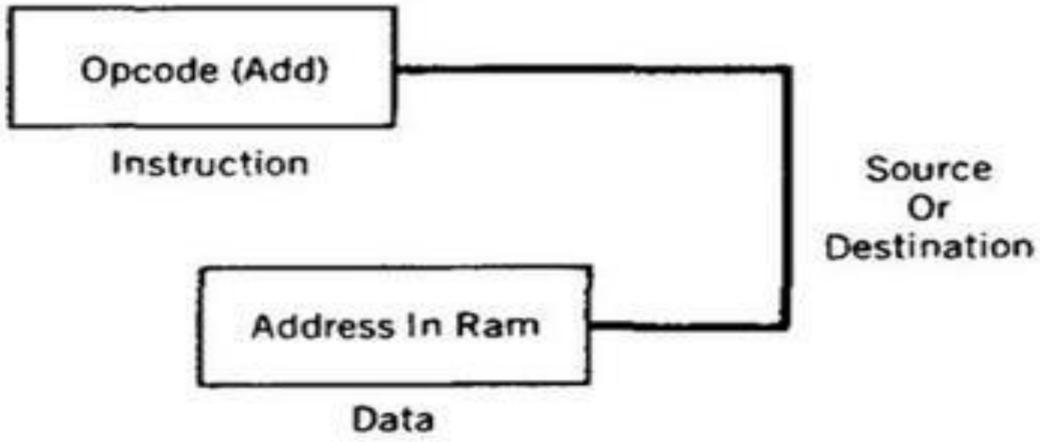
For Example:

1) MOVA, R0 : Copy data from the register R0 to register A
2) ADD A, R1 : Add the content of R1 and A. Store the result in A
3) ANL A, R2 : AND each bit of A with the same bit of register R2.
4) Add the contents of register R3 and R4 from bank 2 . Store the result in A

MOV PSW, 00010000 B : Select register Bank 2
MOV A, R3 : Copy the content R3 to A
ADDA, R4 : Add content A and R4. Store the result in A.

Direct Addressing

- In direct addressing mode, the address of the operand is specified by an 8-bit address in the instruction.
- Using this mode one can access internal data RAM and SFRs, directly. Internal RAM uses addresses from 00H to 7FH to address each byte. The SFR addresses exist from 80H to FFH.

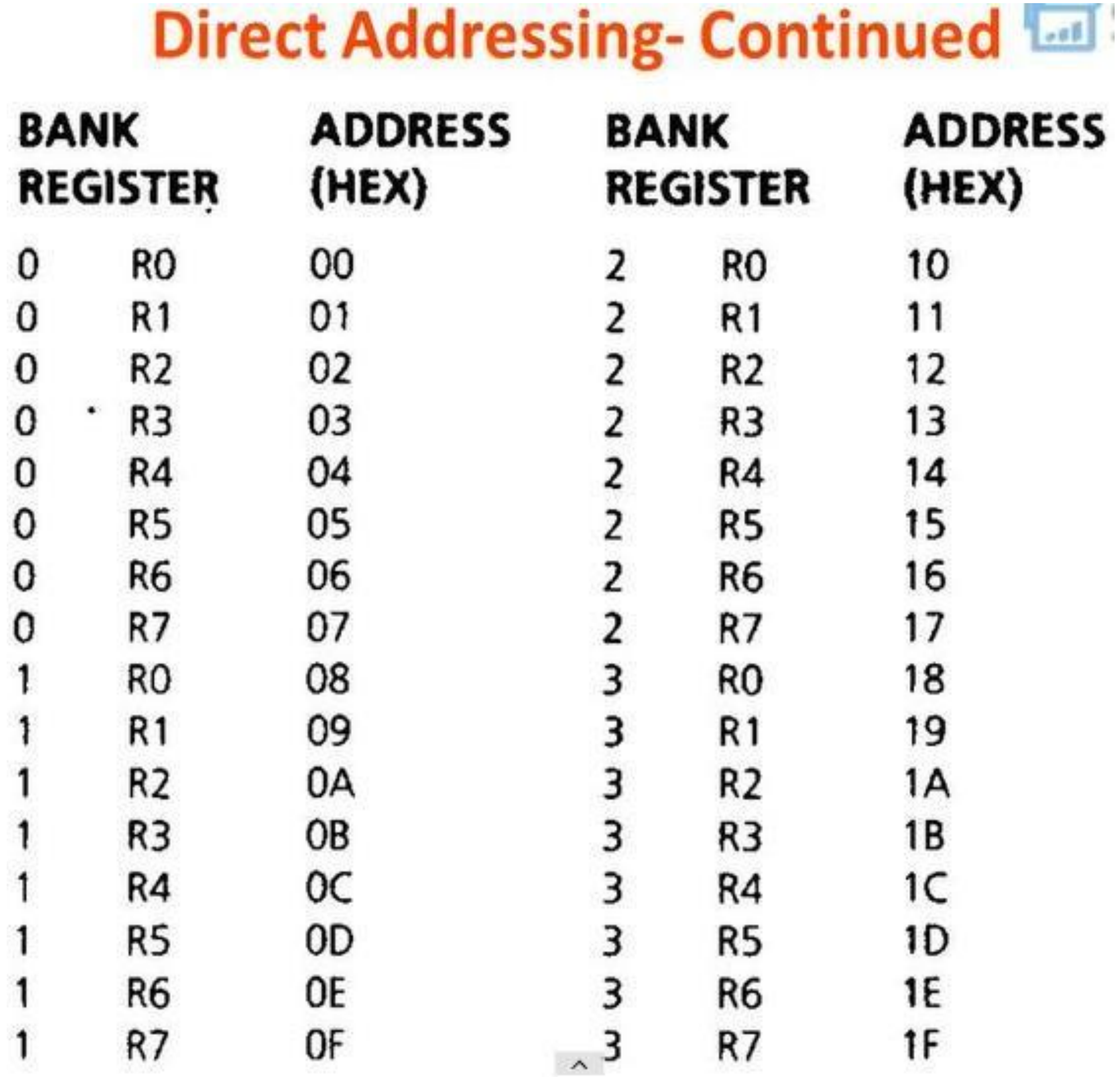


For Example:

1) MOVA, 80H : Copy data from the port 0 to register A
2) MOV 80H, A : Copy data from the register A to port 0
3) MOV 0F0, 12H : Copy data from RAM location 12H to register B
4) MOV 8CH, R7 : Copy data from the resister R7 to timer 0 high byte
5) MOV 5CH, A : Copy data from register A to RAM location 5CH
6) MOV 0A8, 77H : Copy data from RAM location 77H to IE register

Direct Addressing- Continued

SFR	ADDRESS (HEX)
A	0E0
B	0F0
DPL	82
DPH	83
IE	0A8
IP	0B8
P0	80
P1	90
P2	0A0
P3	0B0
PCON	87
PSW	0D0
SBUF	99
SCON	98
SP	81
TCON	88
TMOD	89
TH0	8C
TL0	8A
TH1	8D
TL1	8B

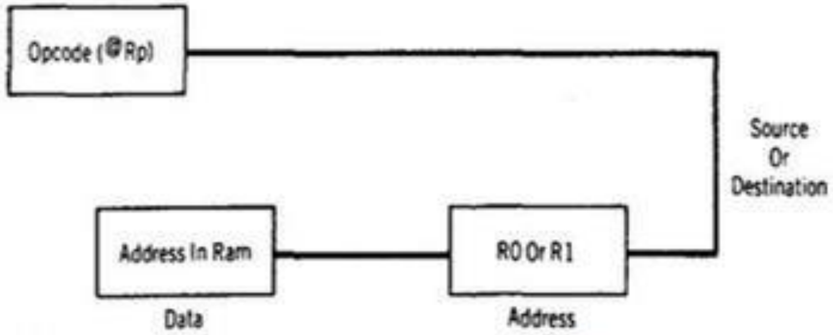


Direct Addressing- Continued

BANK	REGISTER	ADDRESS (HEX)	BANK	REGISTER	ADDRESS (HEX)
0	R0	00	2	R0	10
0	R1	01	2	R1	11
0	R2	02	2	R2	12
0	R3	03	2	R3	13
0	R4	04	2	R4	14
0	R5	05	2	R5	15
0	R6	06	2	R6	16
0	R7	07	2	R7	17
1	R0	08	3	R0	18
1	R1	09	3	R1	19
1	R2	0A	3	R2	1A
1	R3	0B	3	R3	1B
1	R4	0C	3	R4	1C
1	R5	0D	3	R5	1D
1	R6	0E	3	R6	1E
1	R7	0F	3	R7	1F

Indirect Addressing

- In indirect addressing mode instruction specifies a register which holds address of an operand.
- In this mode, only registers R0 or R1 may be used to hold the address of one of the data location in RAM from address 00H to FFH.



Note : *The symbol used for indirect addressing is the "at" sign, which is printed as @. @Rp means register R1 or R0, addressing internal RAM locations from 00H to FFH.*

For Example :

1) **MOV A, @ R0** : Copy contents of memory location, whose address is specified in R0 of selected bank to accumulator.

2) **ADDA. @ R1** : Add the contents of memory location, whose address is specified in R1 and accumulator. Store the result in A.

3) **ANLA, @ R0** : AND each bit of A with same bit of the contents of address contained in R0. Store result in A.

Only two register are used for the operation

r0 AND r1 in indirect addressing mode

In order to access @ was used with r0, r1

@r0

@r1

Example:

1. r0 <= #10h
2. 10h =→ memory location

mov a, @r0

Indirect Addressing (contd..)

- **Advantage:**

It makes accessing a data dynamics rather than static or in the case of direct addressing mode.

For examples :

```
MOV A, #55H            MOV R1, # 06      Count
MOV 40, A              MOV A, # 55H      Data
MOV 41, A              MOV R0, # 40      Memory  Location
MOV 42, A          UP  MOV @ R0, A
MOV 43, A              INC R0
MOV 44, A              DJNZ R1, UP
MOV 45, A              END
```

- **CAUTION :**

The number in register Rp must be a RAM or an SFR address

Only Registers, R0 and R1 (8-bit wide) can be used for pointer in indirect addressing mode.

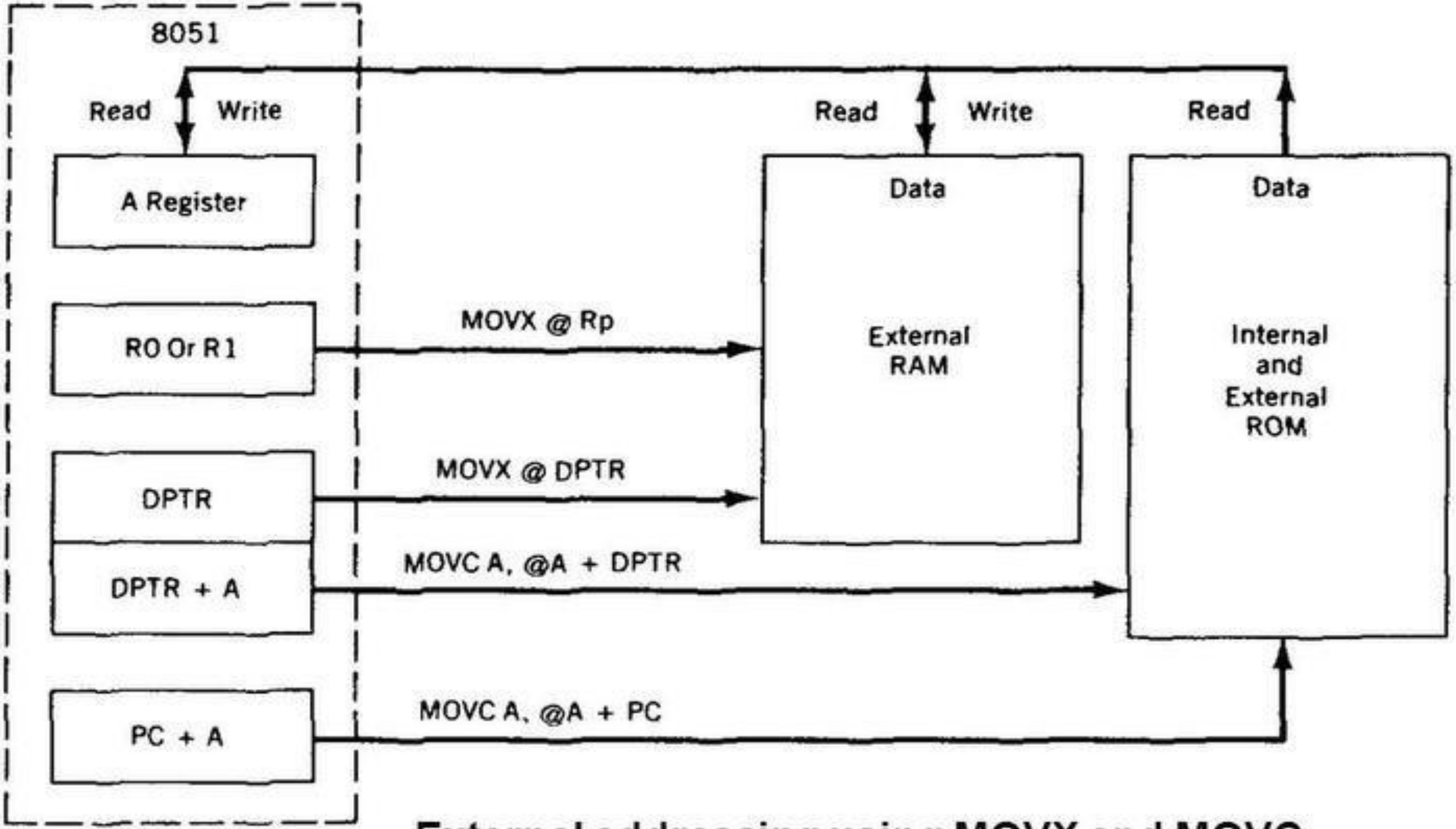
External Data Moves(Index Addressing)

- In the indexed addressing mode, only the program memory can be accessed. The program memory can only be read.
- This addressing mode is preferred for reading look up tables in the program memory.
- Either the DPTR or PC can be used as Index register.

For Examples :

1) ***MOVC A, @ A + DPTR*** *:* Copy the code byte, found at the ROM address formed by adding A and the DPTR, to A.

2) **MOVC A, @ A + PC** :Copy the code byte, found at the ROM address formed by adding A and the PC,to A



External addressing using MOVX and MOVC

movc A, @A+DPTR

➔ A(8bit) = 61h, DPTR(16bit)=1200h.

➔ A + DPTR ➔ 61h + 1200h =1261h.

➔ Due to @, then 1261h = memory location

➔ Memory location in terms of look up table

➔ A 🡐 VALUES IN LOOK UP TABLE OF memory location 1261h
if look up table contain 23h, A = 23 h

movc A, @A+PC

- → A(8bit) = 61h, PC(16bit)=1200h ← PC.
- → A + PC → 61h + 1200h =1261h.
- → Due to @, then 1261h = memory location
- → Memory location in terms of look up table
- → A ← VALUES IN LOOK UP TABLE OF memory location 1261h
  if look up table contain 23h, A = 23 h

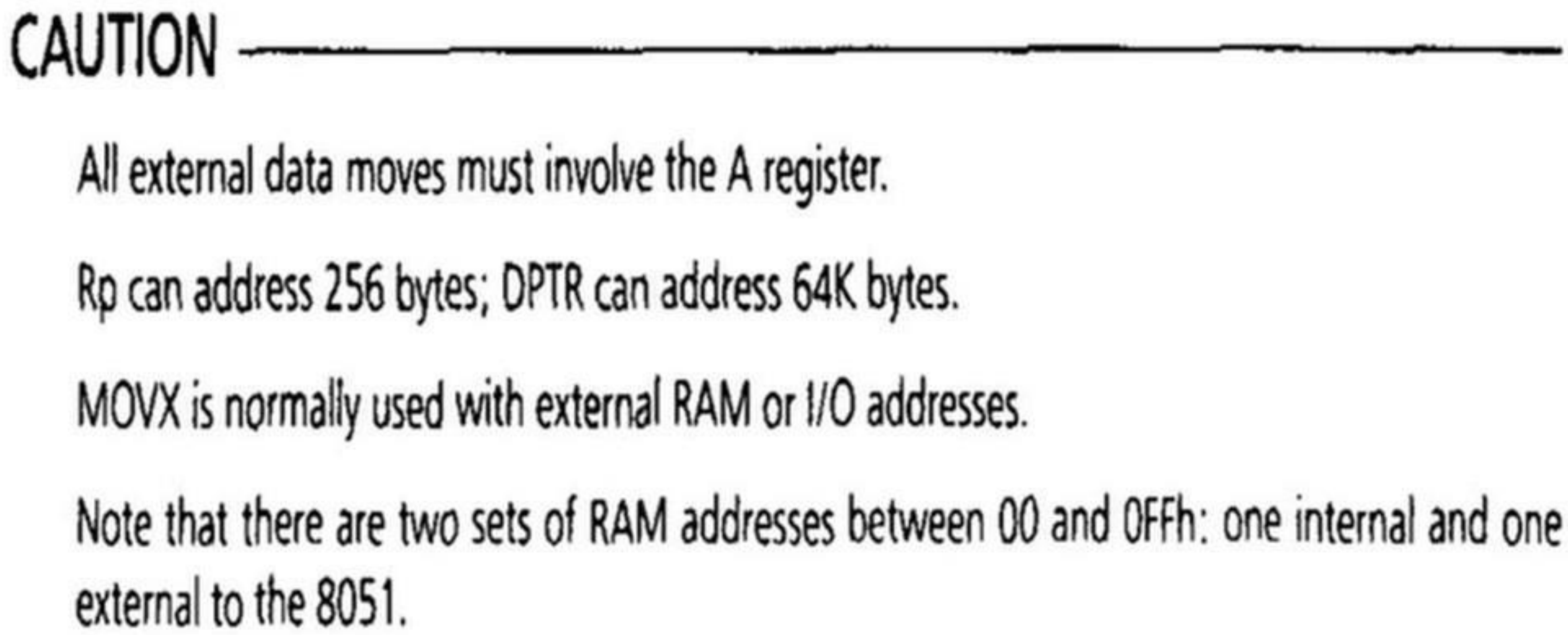
An X is added to the MOV mnemonics to serve as a reminder that the data move is external to the 8051, as shown in the following table.

Mnemonic	**Operation**
MOVX A,@Rp	Copy the contents of the external address in Rp to A
MOVX A,@DPTR	Copy the contents of the external address in DPTR to A
MOVX @Rp,A	Copy data from A to the external address in Rp
MOVX @DPTR,A	Copy data from A to the external address in DPTR

The following table shows examples of external moves using register and indirect addressing modes:

Mnemonic	**Operation**
MOVX @DPTR,A	Copy data from A to the 16-bit address in DPTR
MOVX @R0,A	Copy data from A to the 8-bit address in R0

External Data Moves(Index Addressing)



CAUTION

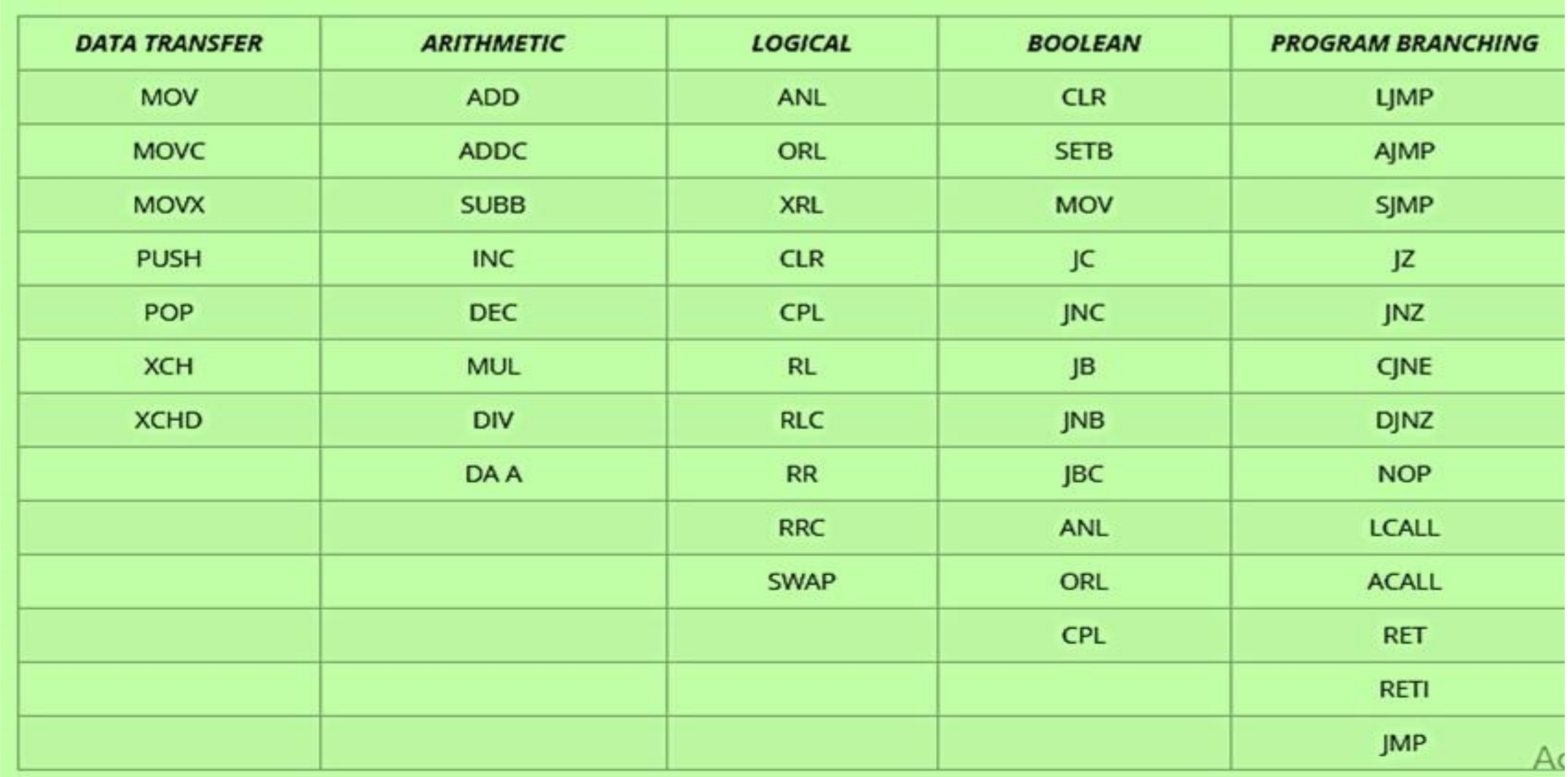
All external data moves must involve the A register.

Rp can address 256 bytes; DPTR can address 64K bytes.

MOVX is normally used with external RAM or I/O addresses.

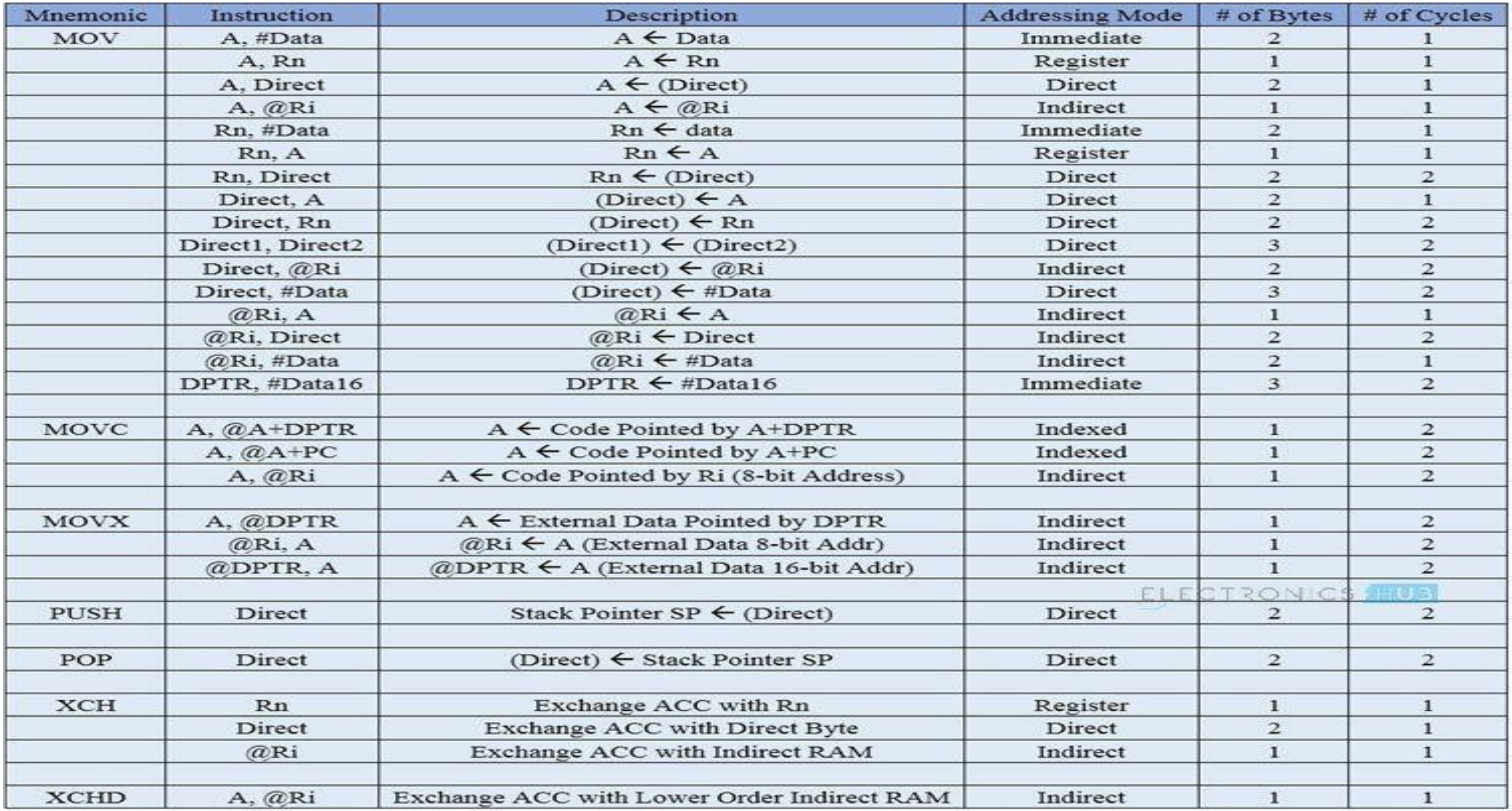
Note that there are two sets of RAM addresses between 00 and 0FFh: one internal and one external to the 8051.

8051 Instruction Sets

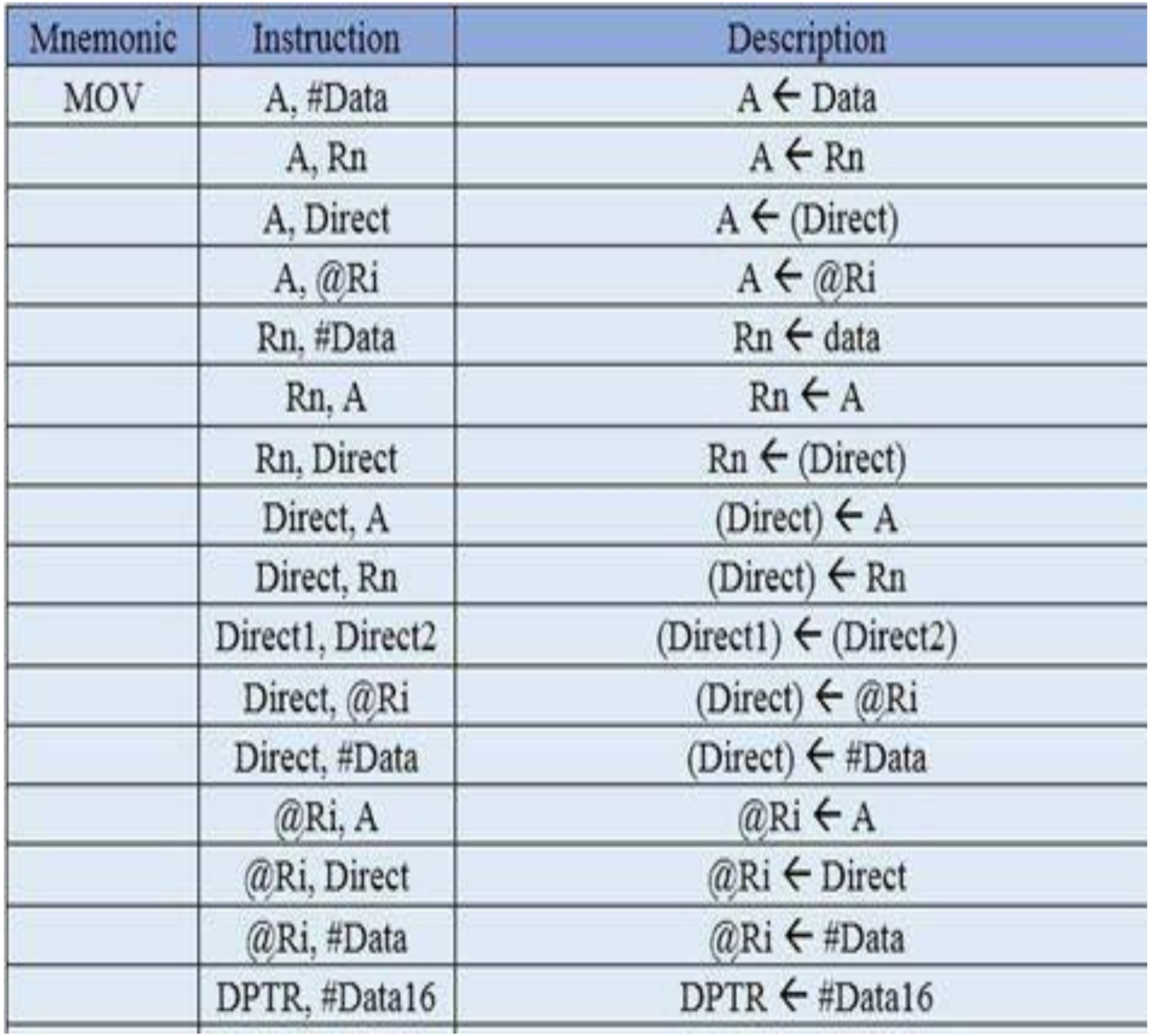


DATA TRANSFER	*ARITHMETIC*	*LOGICAL*	*BOOLEAN*	*PROGRAM BRANCHING*
MOV	ADD	ANL	CLR	LJMP
MOVC	ADDC	ORL	SETB	AJMP
MOVX	SUBB	XRL	MOV	SJMP
PUSH	INC	CLR	JC	JZ
POP	DEC	CPL	JNC	JNZ
XCH	MUL	RL	JB	CJNE
XCHD	DIV	RLC	JNB	DJNZ
	DA A	RR	JBC	NOP
		RRC	ANL	LCALL
		SWAP	ORL	ACALL
			CPL	RET
				RETI
				JMP

Data transfer Instructions



Mnemonic	Instruction	Description	Addressing Mode	# of Bytes	# of Cycles
MOV	A, #Data	A ← Data	Immediate	2	1
	A, Rn	A ← Rn	Register	1	1
	A, Direct	A ← (Direct)	Direct	2	1
	A, @Ri	A ← @Ri	Indirect	1	1
	Rn, #Data	Rn ← data	Immediate	2	1
	Rn, A	Rn ← A	Register	1	1
	Rn, Direct	Rn ← (Direct)	Direct	2	2
	Direct, A	(Direct) ← A	Direct	2	1
	Direct, Rn	(Direct) ← Rn	Direct	2	2
	Direct1, Direct2	(Direct1) ← (Direct2)	Direct	3	2
	Direct, @Ri	(Direct) ← @Ri	Indirect	2	2
	Direct, #Data	(Direct) ← #Data	Direct	3	2
	@Ri, A	@Ri ← A	Indirect	1	1
	@Ri, Direct	@Ri ← Direct	Indirect	2	2
	@Ri, #Data	@Ri ← #Data	Indirect	2	1
	DPTR, #Data16	DPTR ← #Data16	Immediate	3	2
MOVC	A, @A+DPTR	A ← Code Pointed by A+DPTR	Indexed	1	2
	A, @A+PC	A ← Code Pointed by A+PC	Indexed	1	2
	A, @Ri	A ← Code Pointed by Ri (8-bit Address)	Indirect	1	2
MOVX	A, @DPTR	A ← External Data Pointed by DPTR	Indirect	1	2
	@Ri, A	@Ri ← A (External Data 8-bit Addr)	Indirect	1	2
	@DPTR, A	@DPTR ← A (External Data 16-bit Addr)	Indirect	1	2
PUSH	Direct	Stack Pointer SP ← (Direct)	Direct	2	2
POP	Direct	(Direct) ← Stack Pointer SP	Direct	2	2
XCH	Rn	Exchange ACC with Rn	Register	1	1
	Direct	Exchange ACC with Direct Byte	Direct	2	1
	@Ri	Exchange ACC with Indirect RAM	Indirect	1	1
XCHD	A, @Ri	Exchange ACC with Lower Order Indirect RAM	Indirect	1	1



Mnemonic	Instruction	Description
MOV	A, #Data	A ← Data
	A, Rn	A ← Rn
	A, Direct	A ← (Direct)
	A, @Ri	A ← @Ri
	Rn, #Data	Rn ← data
	Rn, A	Rn ← A
	Rn, Direct	Rn ← (Direct)
	Direct, A	(Direct) ← A
	Direct, Rn	(Direct) ← Rn
	Direct1, Direct2	(Direct1) ← (Direct2)
	Direct, @Ri	(Direct) ← @Ri
	Direct, #Data	(Direct) ← #Data
	@Ri, A	@Ri ← A
	@Ri, Direct	@Ri ← Direct
	@Ri, #Data	@Ri ← #Data
	DPTR, #Data16	DPTR ← #Data16

Mov a, #12h

Mov a, r7

Mov a, 20h

Mov a, @r0, mov a, @r1,,,,,.... @ Only r0, r1 only

Mov r3, #30h

Mov r3, a

Mov r7, 25h

Mov 30h, r3

Mov 30h, r4

Mov 20h, 30h

Mov 20h, @r0, mov 35h, @r1

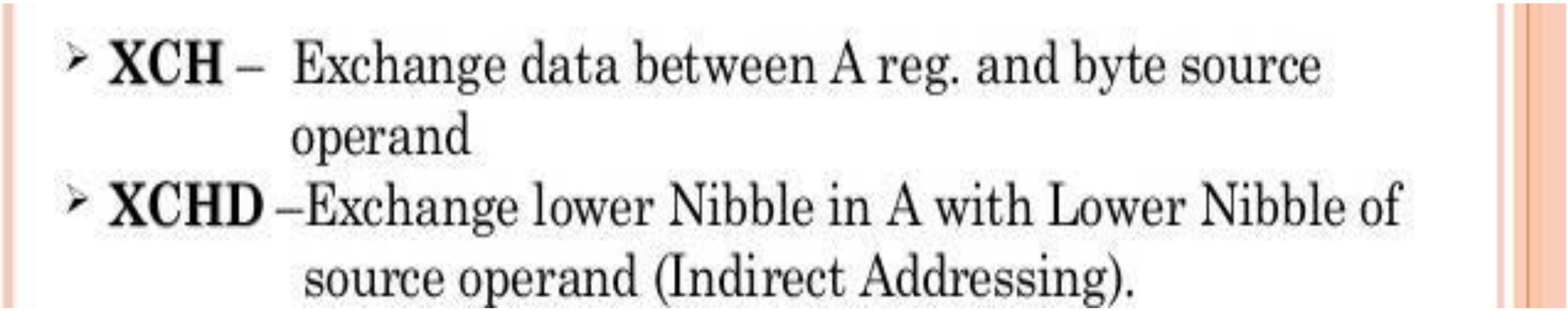
Mov 20h, #25h, mov 30h, #40h

Mov @r1, a, mov @r0, a

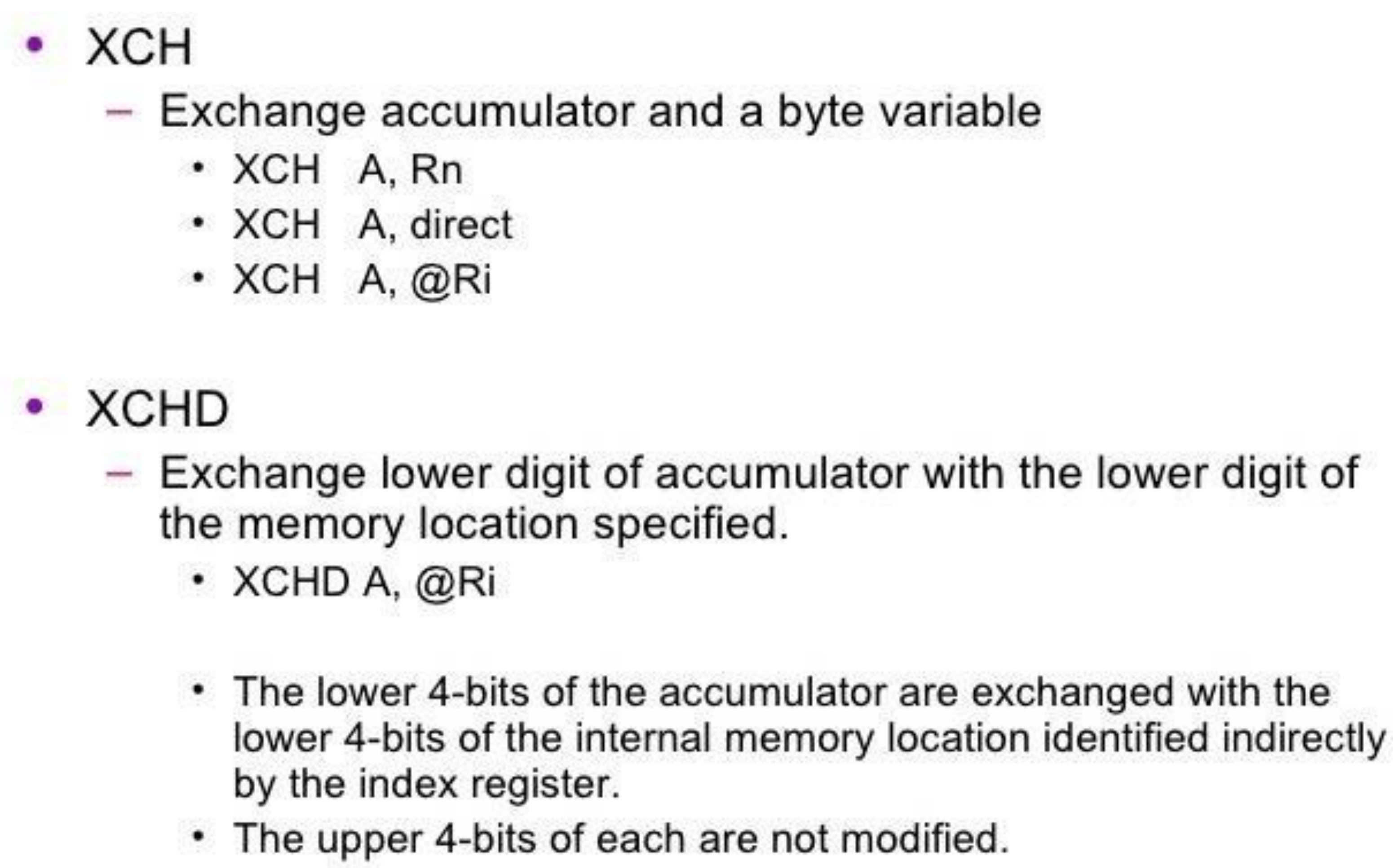
Mov @r0, 20h

Mov @r1, #0ffh

Mov dptr, #1234h



- **XCH** – Exchange data between A reg. and byte source operand
- **XCHD** –Exchange lower Nibble in A with Lower Nibble of source operand (Indirect Addressing).



- XCH
  - Exchange accumulator and a byte variable
    - XCH A, Rn
    - XCH A, direct
    - XCH A, @Ri
- XCHD
  - Exchange lower digit of accumulator with the lower digit of the memory location specified.
    - XCHD A, @Ri
    - The lower 4-bits of the accumulator are exchanged with the lower 4-bits of the internal memory location identified indirectly by the index register.
    - The upper 4-bits of each are not modified.

Example:

A = 23h, r0 = 45 h

XCH A, r0
➔A = 45h, r0 ➔23h

XCHD A, r0
➔A=25h, r0 ➔43h

XCH A, R0 === exchange instruction

Source value (right side) <=> destination (left side)

Example:

A = 23h, r0 = FA h

XCH A, r0

➔A = FAh, r0 ➔23h

XCH A, r0

XCH A, @r0

A= 23H, r0 = #10h (immediate value) => becomes mem location = 34h

➔ A = 34h, 10h =23h

XCH A, 20H

A = 12h, 20h = 35h

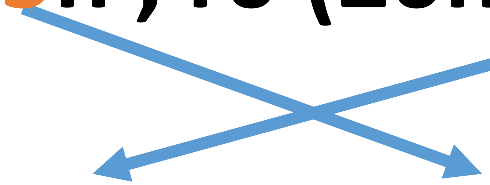
➔ A = 35h, 20h = 12h

Exchange instruction :

Here only lower nibble (4bit) will be exchanged.

XCHD A, @r0 r0=#20h → memory location = 13h

A = 25h , r0 (20h)= 13h

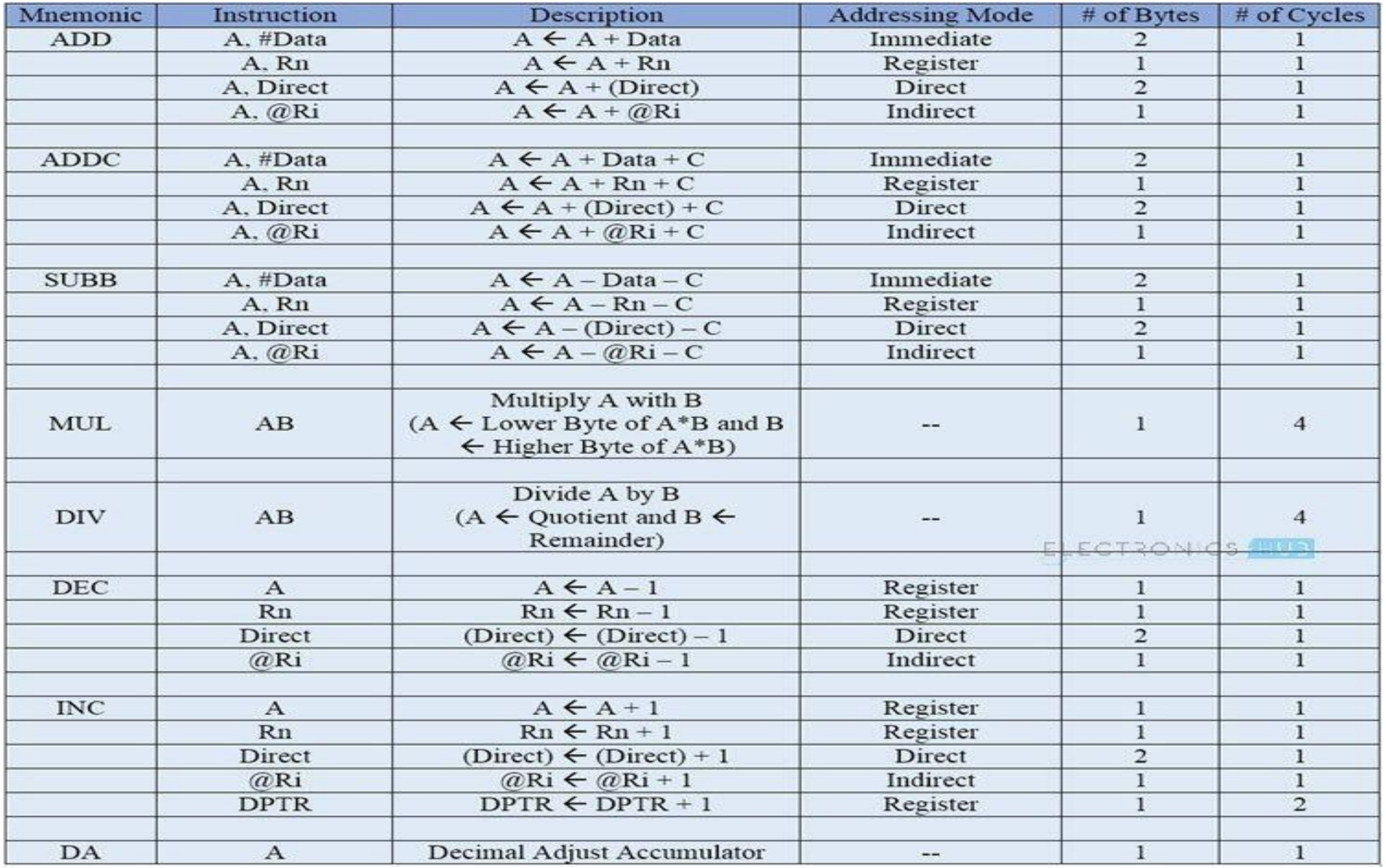


→A=23, 20h →15h

A= AFh, r0= 12h

→ A = A2h,, r0 = 1Fh

Arithmetic Instructions

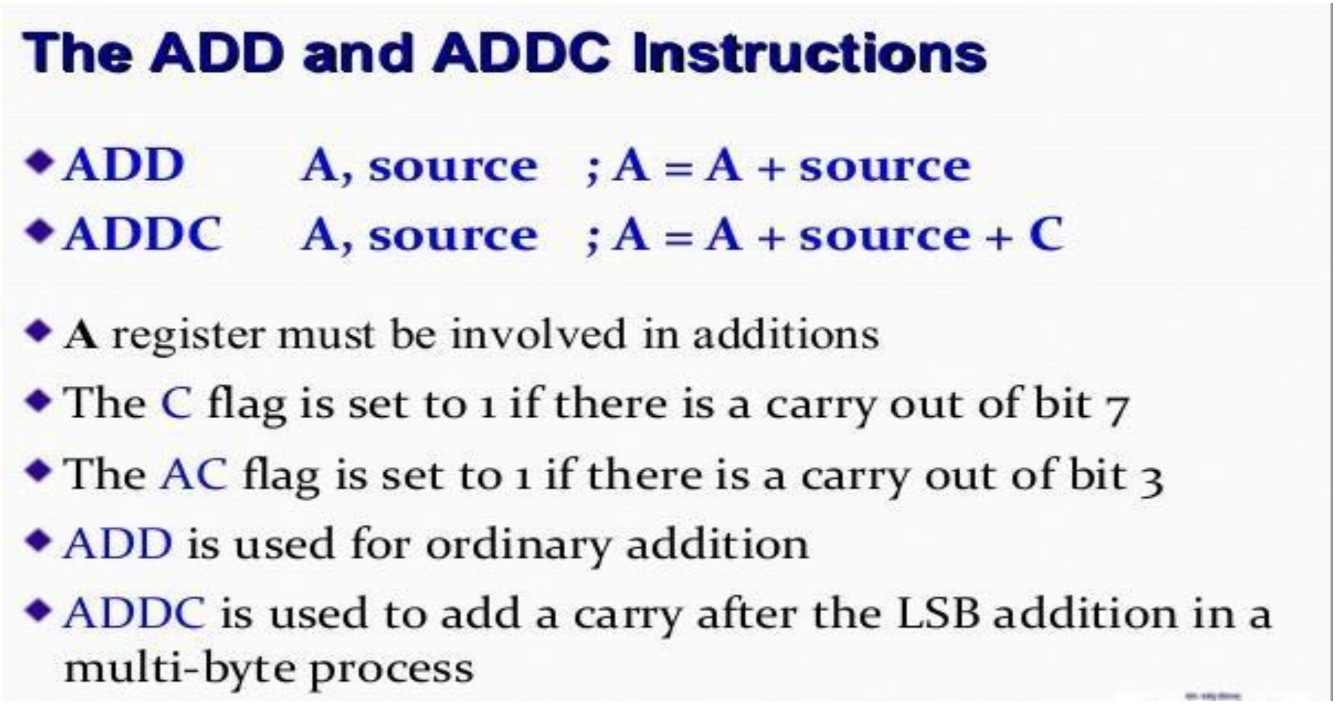


Mnemonic	Instruction	Description	Addressing Mode	# of Bytes	# of Cycles
ADD	A, #Data	A ← A + Data	Immediate	2	1
	A, Rn	A ← A + Rn	Register	1	1
	A, Direct	A ← A + (Direct)	Direct	2	1
	A, @Ri	A ← A + @Ri	Indirect	1	1
ADDC	A, #Data	A ← A + Data + C	Immediate	2	1
	A, Rn	A ← A + Rn + C	Register	1	1
	A, Direct	A ← A + (Direct) + C	Direct	2	1
	A, @Ri	A ← A + @Ri + C	Indirect	1	1
SUBB	A, #Data	A ← A – Data – C	Immediate	2	1
	A, Rn	A ← A – Rn – C	Register	1	1
	A, Direct	A ← A – (Direct) – C	Direct	2	1
	A, @Ri	A ← A – @Ri – C	Indirect	1	1
MUL	AB	Multiply A with B (A ← Lower Byte of A*B and B ← Higher Byte of A*B)	--	1	4
DIV	AB	Divide A by B (A ← Quotient and B ← Remainder)	--	1	4
DEC	A	A ← A – 1	Register	1	1
	Rn	Rn ← Rn – 1	Register	1	1
	Direct	(Direct) ← (Direct) – 1	Direct	2	1
	@Ri	@Ri ← @Ri – 1	Indirect	1	1
INC	A	A ← A + 1	Register	1	1
	Rn	Rn ← Rn + 1	Register	1	1
	Direct	(Direct) ← (Direct) + 1	Direct	2	1
	@Ri	@Ri ← @Ri + 1	Indirect	1	1
	DPTR	DPTR ← DPTR + 1	Register	1	2
DA	A	Decimal Adjust Accumulator	--	1	1

Arithmetic Flags (Conditional Flags)

- There are 4 arithmetic flags in the 8051
  - Carry (C)
  - Auxiliary Carry (AC)
  - Overflow (OV)
  - Parity (P)
- All the above flags are stored in the Program Status Word (**PSW**)

CY	AC	--	RS1	RS0	0V	--	P
PSW.7	PSW.6	PSW.5	PSW.4	PSW.3	PSW.2	PSW.1	PSW.0



The ADD and ADDC Instructions

- **ADD A, source ; A = A + source**
- **ADDC A, source ; A = A + source + C**

- **A** register must be involved in additions
- The C flag is set to 1 if there is a carry out of bit 7
- The AC flag is set to 1 if there is a carry out of bit 3
- ADD is used for ordinary addition
- ADDC is used to add a carry after the LSB addition in a multi-byte process

ADD and ADDC operation effectiveness:

8051 process only 8 bit at a time, so it will use ADD instruction since CY initially zero,, if carry generated then the cy will entering to next 8 bit in order to make complete 16bit addition where it will make use ADDC since CY is considered here. Note that for only 8 bit addition add instruction is enough to show Sum and cy

1. Add 1234 h and FB19 h

here carry is ‘1’→ 1 0←here carry is usually ‘0’

1 2 3 4 h

F B F 9 h

----

1 0 E 2 D h

3 + F = 12 h here sum is 2 and carry is 1

Add two 16bit number 1234 h and FBF9h

```
Mov  r0, #34h
Mov  a,  #0f9h
Add a, r0

Mov  r7, a

Mov r1, #12h
Mov a, #0fbh
Addc a, r1

Mov r6, a

JC vijay
Mov r5, #00h
Vijay: inc r5
end
```

here carry is ‘1’→ 1   0←here carry is usually ‘0’

1 2   3 4 h

+ F B   F 9 h

----

1   0 E   2 D h

answer stored in registers r5   r6   r7

Both input and output are playing in registers

Add two 16bit number 1234 h and FBF9h

```
Mov  r0, #34h
Mov a,  #0f9h
Add a, r0

Mov  22h, a

Mov r1, #12h
Mov a, #0fbh
Addc a, r1

Mov 21h, a

JC vijay
Mov 20h, #00h
Vijay: inc 20h
end
```

here carry is ‘1’→ 1   0←here carry is usually ‘0’

1 2   3 4 h

+ F B   F 9 h

----

1   0 E   2 D h

answer stored in registers 20h   21   22h

Input in register and output are playing in memory location

Add two 16bit number 1234 h and FBF9h

```
Mov r0, #21h
Mov r1, #31h
Mov a, @r0
Add a, @r1

Mov  22h, a

Mov r0, #20h
Mov r1, #30h
Mov a, @r0
Addc a,@r1

Mov 21h, a

JC vijay
Mov 20h, #00h
Vijay: inc 20h
end
```

here carry is ‘1’→ 1   0←here carry is usually ‘0’

	20h	21h	
	1 2	3 4 h	
	30h	31h	
	+ F B	F 9 h	

	1	0 E	2 D h
answer stored in registers	20h	21h	22h

Both input and output are playing in memory location

Add two 16bit number 1234 h and FBF9h

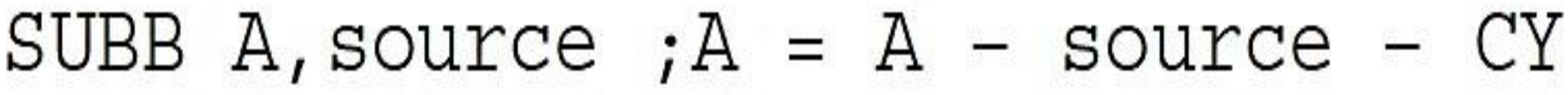
```
Mov r0, #21h
Mov r1, #31h
Mov a, @r0
Add a, @r1

Mov  r7h, a

Mov r0, #20h
Mov r1, #30h
Mov a, @r0
Addc a,@r1

Mov r6h, a

JC vijay
Mov r5h, #00h
Vijay: inc r5h
end
```

here carry is ‘1’➔ 1   0🡐here carry is usually ‘0’

```
   20h   21h
   1 2   3 4 h

   30h   31h
 + F B   F 9 h
 ---------------
1  0 E   2 D h
```

answer stored in registers   r5h   r6h   r7h

input are in memory location and output are playing in register



```
SUBB A,source ;A = A - source - CY
```

1) 5678h
-1234h

4444h

5678 h – 1234h – CY = 5678 h – 1234 h - 0 ➔ 4444h   hint: HERE CY =0

2) 1234 h
- 5678 h

34h => 0011 0100
78h => 0111 1000 => 1000 0111
+1

=>1000 1000 => 88 h

2' COMPLEMENT OPERATION USED WHEN CY = 1

therefore sub => 34h + 88h - Initial carry ➔ BCh – 0 = BC h   with CY =1

12h => 0001 0010
56h => 0101 0110 => 1010 1001
+1

=> 1010 1010 => AA h

2' COMPLEMENT OPERATION USED WHEN CY = 1

therefor sub => 12h + AA – 1 → BC h – 1 ➔ BB h

Final answer is BB BC h   WITH CARRY FLAG = 1

```
   56 78 h
-  23 AB h
 -------------
78h – AB h
1010 1011 =>              0101 0100
                                 +1
                          --------------
                          0101 0101 = 55h
78h + 55h – 0 => CD h with CY = 1

56h – 23h – 01h => 32 h
```

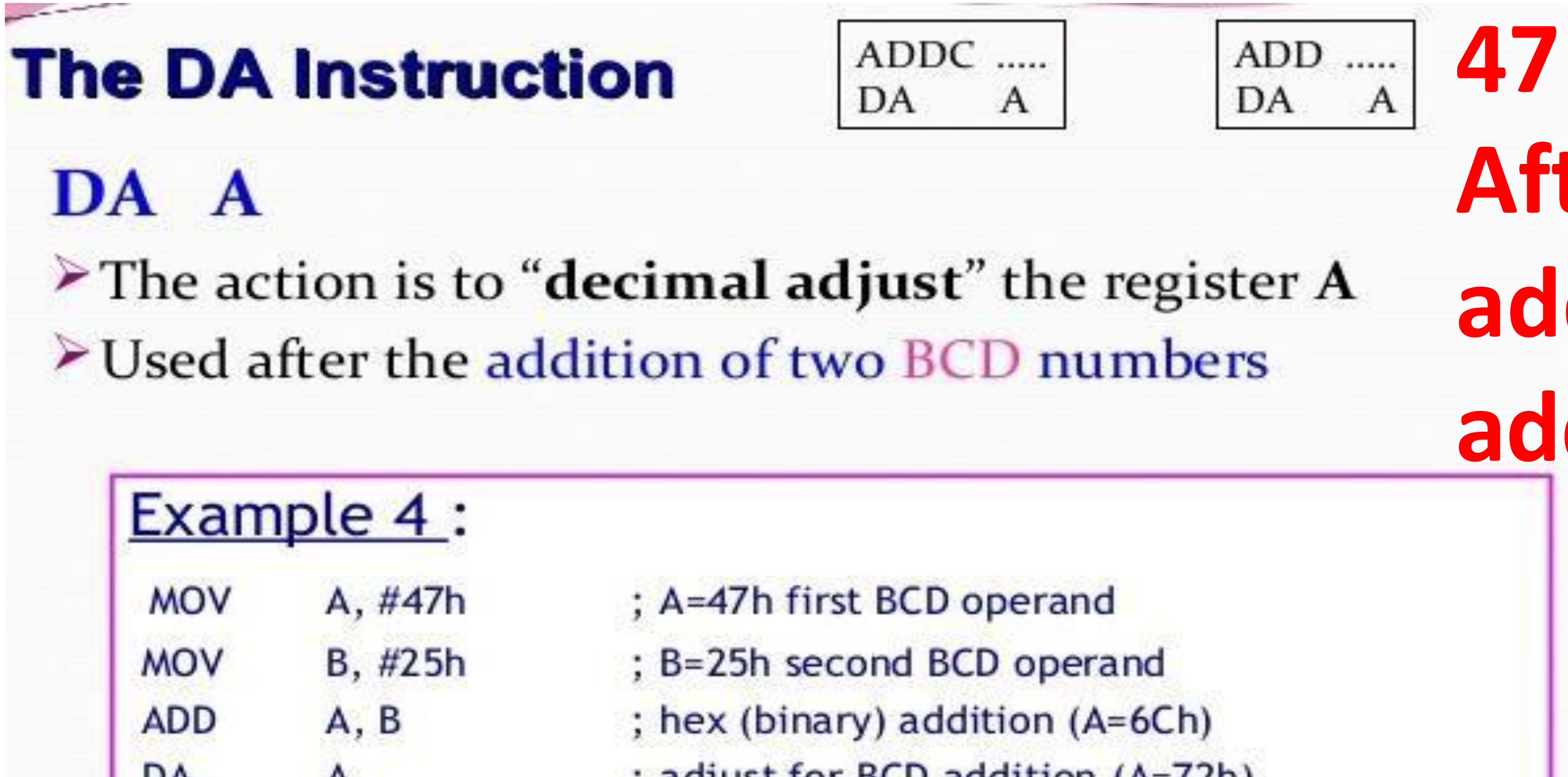
Final answer ➔ **32 CD h**

Program for 2byte 16-bit Subtraction

```
Org 00h
Clr c
Mov a, 21h
Mov r0, a
Mov b, 31h
Subb a, b // a = a – b - cy
Mov 62h, a

Mov a, 20h,
Mov r1, a
Mov b, 30h
Subb a, b  //// a = a – b - cy

Jc next2
Mov 61h, a
Mov 60h, #00h
```

```
Next2: cpl b
Add b, #01h
Mov a, b
Add a, r1
Dec a
Mov 61h, a
Mov 60h, #01h
end
```

Note: In this program Input values from 20h, 21h 30h, 31h, and output values Are stored in 60h, 61h, 62h



The DA Instruction

```
ADDC  .....
DA        A
```

```
ADD  .....
DA       A
```

DA A

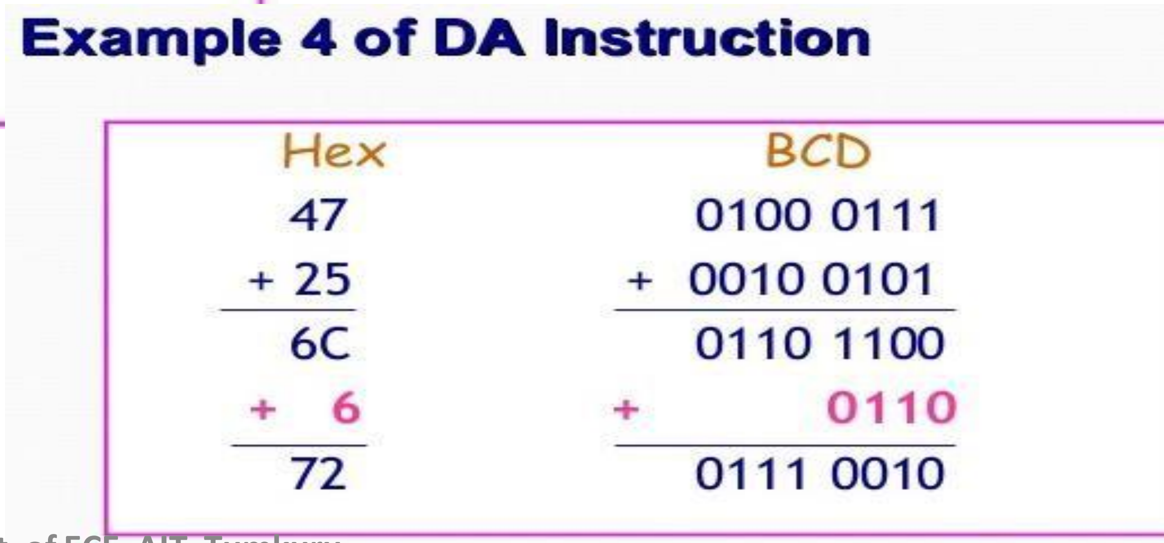
- The action is to "**decimal adjust**" the register A
- Used after the addition of two BCD numbers

Example 4 :

```
MOV      A, #47h          ; A=47h first BCD operand
MOV      B, #25h          ; B=25h second BCD operand
ADD      A, B             ; hex (binary) addition (A=6Ch)
DA       A                ; adjust for BCD addition (A=72h)
```

47 h + 25 h = 6C h

After hexadecimal addition >9 → +6 add



Example 4 of DA Instruction

Hex	BCD
47	0100 0111
+ 25	+ 0010 0101
6C	0110 1100
+ 6	+ 0110
72	0111 0010

MUL AB

8 Bit X 8 Bit

12h X 23h

A = 12h, B = 23h

Or A = 23h , B =12h

After multiplication operation

→ Result = 02 76 h

→ Result is stored such that

→ A = 76h

→ B = 02h

- The 8051 supports byte by byte multiplication only
  - The byte are assumed to be unsigned data

`MUL AB   ;AxB, 16-bit result in B, A`

```
        MOV     A,#25H     ;load 25H to reg.  A
        MOV     B,#65H     ;load 65H to reg.  B
        MUL     AB         ;25H * 65H = E99 where
                           ;B = OEH and A = 99H
```

Unsigned Multiplication Summary (MUL AB)

Multiplication	Operand1	Operand2	Result
Byte x byte	A	B	B = high byte A = low byte

- The 8051 supports byte over byte division only
  - The byte are assumed to be unsigned data

`DIV  AB   ;divide A by B, A/B`

```
        MOV     A,#95      ;load 95 to reg.  A
        MOV     B,#10      ;load 10 to reg.  B
        div     AB         ;A = 09(quotient) and
                           ;B = 05(remainder)
```

Unsigned Division Summary (DIV AB)

Division	Numerator	Denominator	Quotient	Remainder
Byte / byte	A	B	A	B

CY is always 0
If B ≠ 0, OV = 0
If B = 0, OV = 1 indicates error

1234 h X ABCD h = 0C 37 4F A4 h

16 Bit X 16 Bit

12 34 h X AB CD h

Carry➔01 29 A4 ➔ CD h X 34 h

0E 6A ➔ CD h X 12 h

22 BC ➔ AB h X 34 h

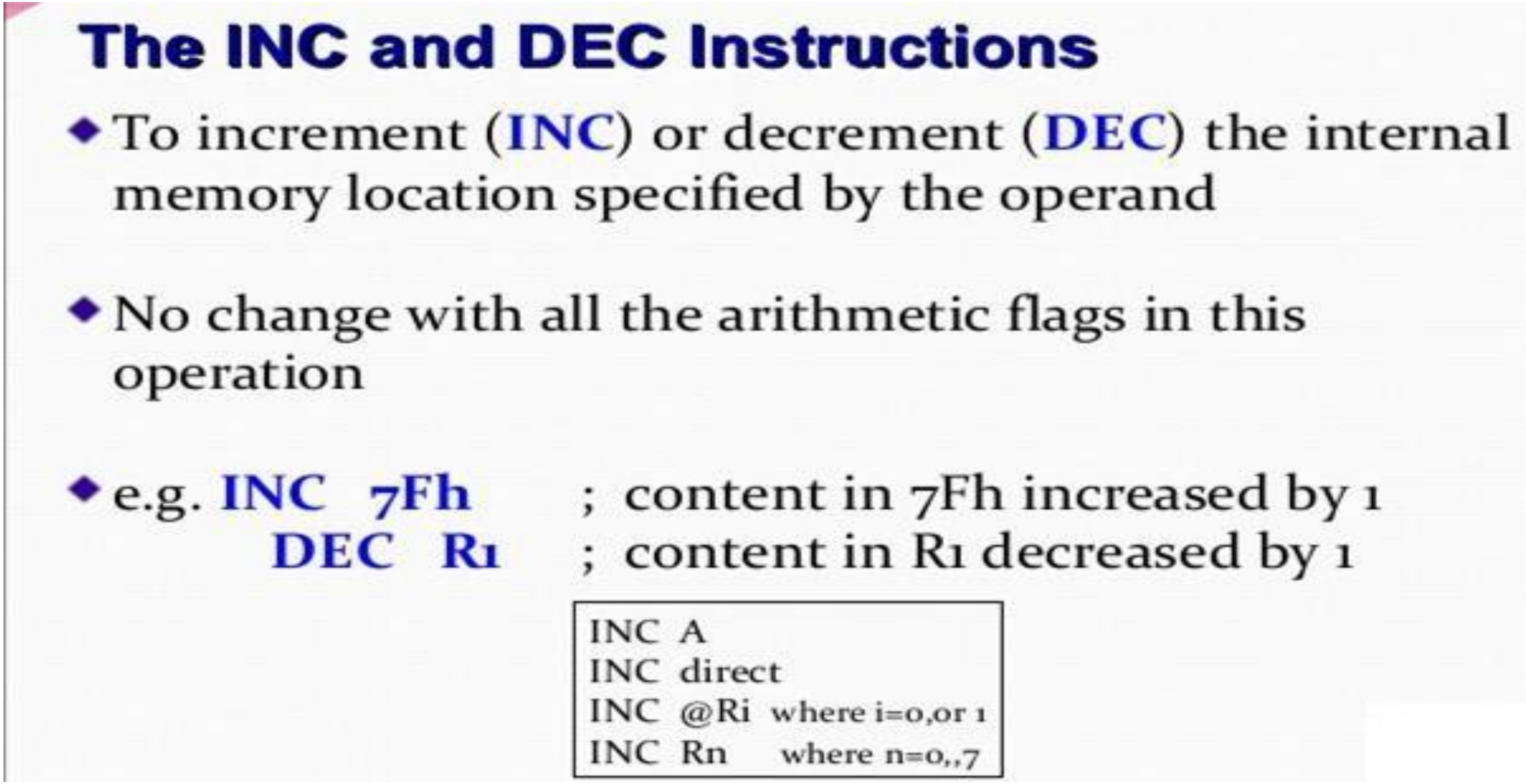
0C 06 ➔ AB h X 12 h

0C 37 4F A4

34h x CD h = 29 A4 h ➔ A = A4 h, B = 29h therefore here 29 h move as carry and A4 h is one of the direct answer

```
Org 00h
MOV r5, #0CDh
mov b, r5
Mov a, #34h
Mul AB       /// CD h X 34 h ? B = 29 h,  A = A4 h

Mov 23h,A
Mov r6,B   //// r6 = 29h

Mov a,#12h
Mov b,r5
Mul AB    //// 12 h X CD ? B = 0E h, A=6A h

clr  c
Addc a,r6 //// a = 29 h + 6A h = 93 h with carry = 0
Mov  r6, a  //// r6 = 93 h

Mov r7, B  ///// r7 = 0E h

Mov r5, #0ABh
mov b, r5
Mov a, #34h
Mul AB   ///// AB h X 34 h ? B = 22 h, A = BC
```

```
Addc a, r6  //// cy = 0 + 93 h + BC =  01 4F h
Mov 22h, a

Mov a, b    ///// a = 22 h
Addc a, r7 /// 0E h + 22h + 01h =  31h, and here cy = 1 is added
Mov r3, a /// r3 = 31h, cy = 0

Mov  r5,  #12h
Mov a, #0ABh
Mov B, r5
Mul AB   /// B = 0C h, A = 06h

Addc a, r3   ///06 h + 31 h  + (cy = 0) = 37 h
Mov 21h, a
Mov a, b
Mov 20h, a

end
```



The INC and DEC Instructions

- To increment (**INC**) or decrement (**DEC**) the internal memory location specified by the operand

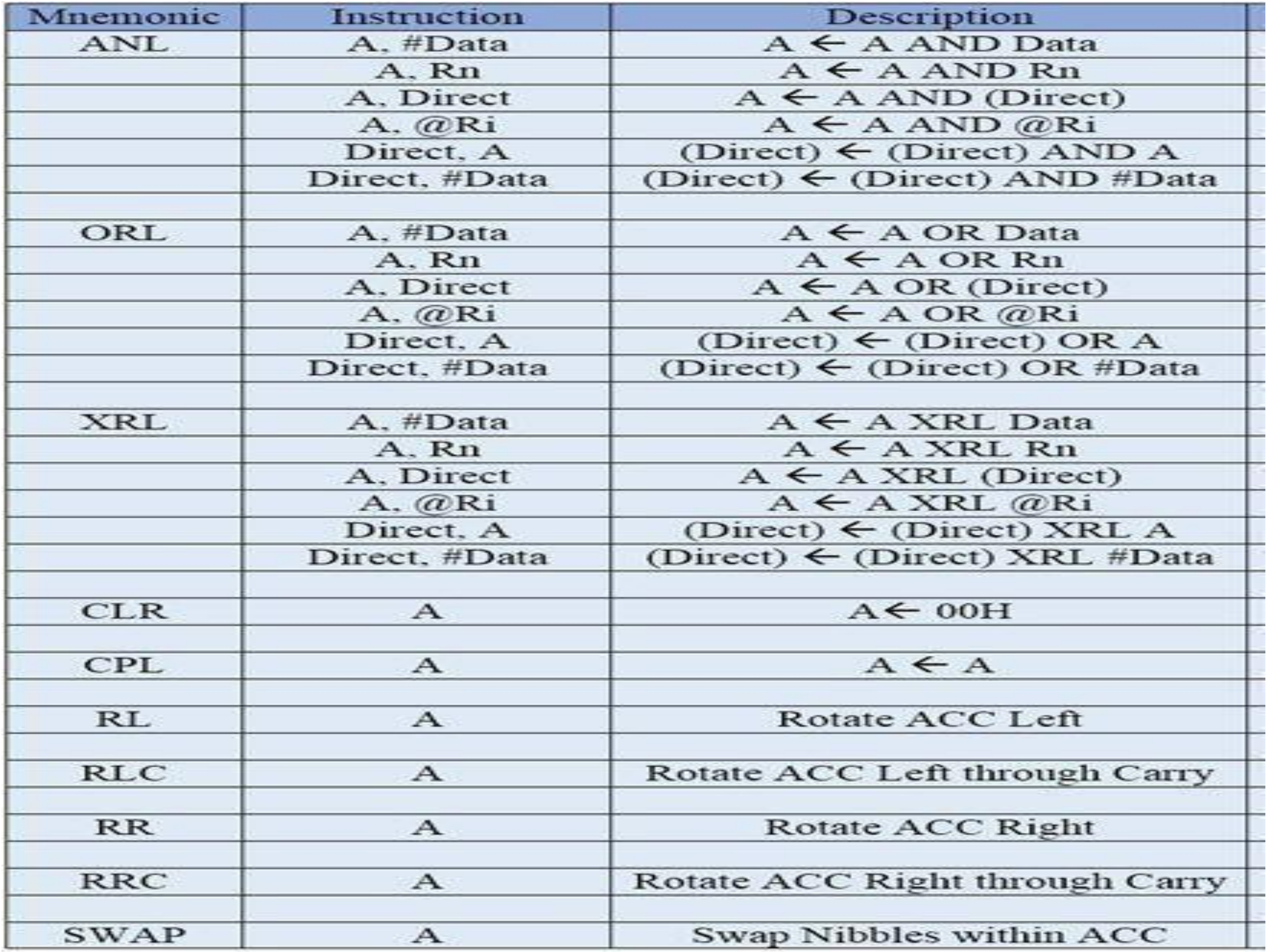
- No change with all the arithmetic flags in this operation

- e.g. **INC 7Fh** ; content in 7Fh increased by 1
  DEC R1 ; content in R1 decreased by 1

```
INC  A
INC  direct
INC  @Ri  where i=0,or 1
INC  Rn     where n=0,,7
```

Logical Instructions

Mnemonic	Instruction	Description	Addressing Mode	# of Bytes	# of Cycles
ANL	A, #Data	A ← A AND Data	Immediate	2	1
	A, Rn	A ← A AND Rn	Register	1	1
	A, Direct	A ← A AND (Direct)	Direct	2	1
	A, @Ri	A ← A AND @Ri	Indirect	1	1
	Direct, A	(Direct) ← (Direct) AND A	Direct	2	1
	Direct, #Data	(Direct) ← (Direct) AND #Data	Direct	3	2
ORL	A, #Data	A ← A OR Data	Immediate	2	1
	A, Rn	A ← A OR Rn	Register	1	1
	A, Direct	A ← A OR (Direct)	Direct	2	1
	A, @Ri	A ← A OR @Ri	Indirect	1	1
	Direct, A	(Direct) ← (Direct) OR A	Direct	2	1
	Direct, #Data	(Direct) ← (Direct) OR #Data	Direct	3	2
XRL	A, #Data	A ← A XRL Data	Immediate	2	1
	A, Rn	A ← A XRL Rn	Register	1	1
	A, Direct	A ← A XRL (Direct)	Direct	2	1
	A, @Ri	A ← A XRL @Ri	Indirect	1	1
	Direct, A	(Direct) ← (Direct) XRL A	Direct	2	1
	Direct, #Data	(Direct) ← (Direct) XRL #Data	Direct	3	2
CLR	A	A← 00H	--	1	1
CPL	A	A ← A	--	1	1
RL	A	Rotate ACC Left	--	1	1
RLC	A	Rotate ACC Left through Carry	--	1	1
RR	A	Rotate ACC Right	--	1	1
RRC	A	Rotate ACC Right through Carry	--	1	1
SWAP	A	Swap Nibbles within ACC	--	1	1



Mnemonic	Instruction	Description
ANL	A, #Data	A ← A AND Data
	A, Rn	A ← A AND Rn
	A, Direct	A ← A AND (Direct)
	A, @Ri	A ← A AND @Ri
	Direct, A	(Direct) ← (Direct) AND A
	Direct, #Data	(Direct) ← (Direct) AND #Data
ORL	A, #Data	A ← A OR Data
	A, Rn	A ← A OR Rn
	A, Direct	A ← A OR (Direct)
	A, @Ri	A ← A OR @Ri
	Direct, A	(Direct) ← (Direct) OR A
	Direct, #Data	(Direct) ← (Direct) OR #Data
XRL	A, #Data	A ← A XRL Data
	A, Rn	A ← A XRL Rn
	A, Direct	A ← A XRL (Direct)
	A, @Ri	A ← A XRL @Ri
	Direct, A	(Direct) ← (Direct) XRL A
	Direct, #Data	(Direct) ← (Direct) XRL #Data
CLR	A	A← 00H
CPL	A	A ← A
RL	A	Rotate ACC Left
RLC	A	Rotate ACC Left through Carry
RR	A	Rotate ACC Right
RRC	A	Rotate ACC Right through Carry
SWAP	A	Swap Nibbles within ACC

ANL A, #34h
ANL A, r3
ANL A, 20H
ANL A, @R1
ANL 30H, A
ANL 20H, #35H

ORL A, #34h
ORL A, r3
ORL A, 20H
ORL A, @R1
ORL 30H, A
ORL 20H, #35H

XRL A, #34h
XRL A, r3
XRL A, 20H
XRL A, @R0
XRL 30H, A
XRL 20H, #35H

CLR A

CPL A

RL A
RLC A
RLC A
RR A
RRC A

And operation: 1111 0010 →F2h
0010 1111 →2Fh

0010 0010 →22h

OR operation: 1111 0010 →F2h
0010 1111 →2Fh

1111 1111 →FFh

XOR operation: 1111 0010 →F2h
0010 1111 →2Fh

1101 1101 →CC h

CLR Clear operation: A = 1111 0010 →F2h
A = 0000 0000 →00H

CPL Complement operation: A = 1111 0010 →F2h
A = 0000 1101 →0C h

Rotate instruction

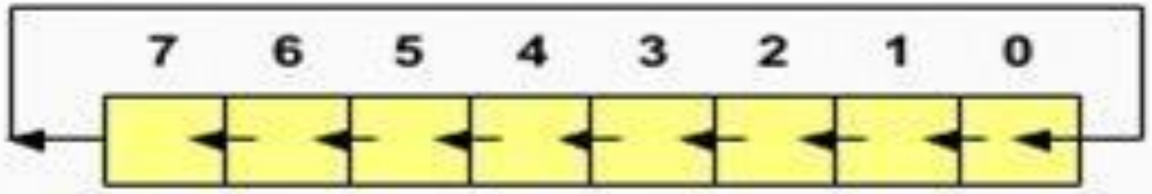
Rotate right ===→ RR carry not considered

Rotate left ===→ RL carry not considered

Rotate right with carry ===→ RRC

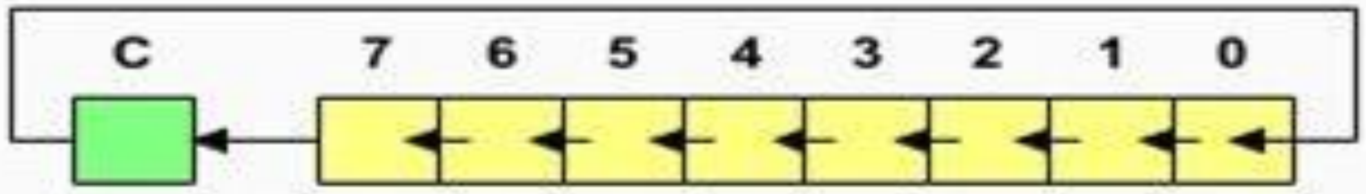
Rotate left with carry ==→ RLC

The Rotate Instructions



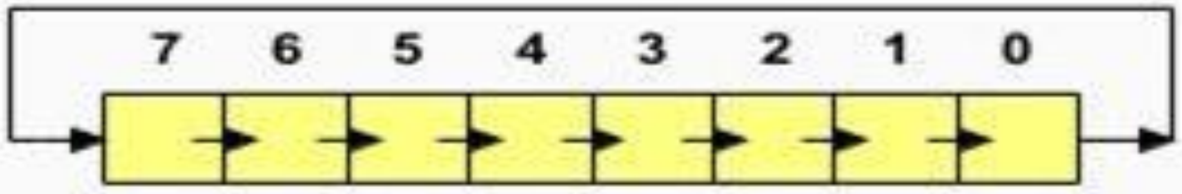
RL A

Before: 10011100
After: 00111001



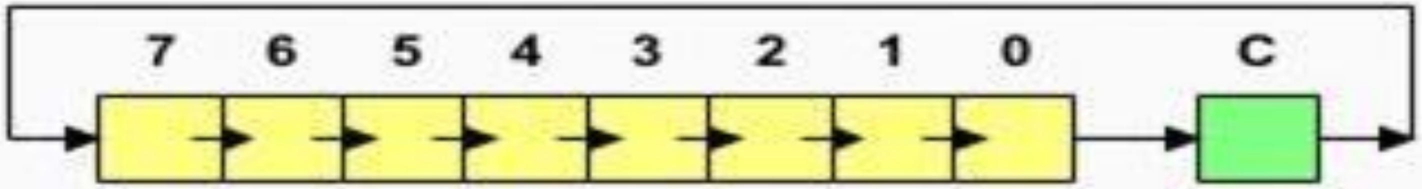
RLC A

Before: 10011100 CY = 0
After: 00111000 CY = 1



RR A

Before: 10011100
After: 01001110

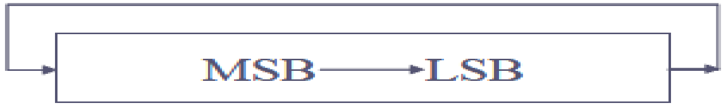


RRC A

Before: 10011100 CY = 1
After: 11001110 CY = 0

RR A ;rotate right A

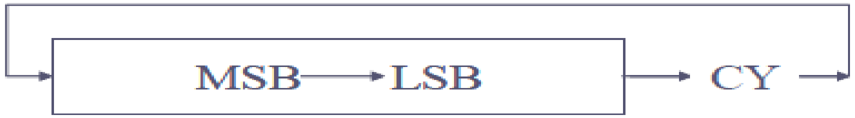
- In rotate right
  - The 8 bits of the accumulator are rotated right one bit, and
  - Bit D0 exits from the LSB and enters into MSB, D7



```
MOV  A,#36H     ;A = 0011 0110
RR   A          ;A = 0001 1011
RR   A          ;A = 1000 1101
RR   A          ;A = 1100 0110
RR   A          ;A = 0110 0011
```

RRC A ;rotate right through carry

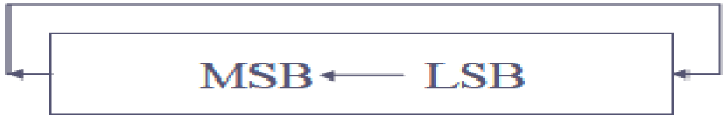
- In RRC A
  - Bits are rotated from left to right
  - They exit the LSB to the carry flag, and the carry flag enters the MSB



```
CLR C           ;make CY = 0
MOV A,#26H      ;A = 0010 0110
RRC A           ;A = 0001 0011    CY = 0
RRC A           ;A = 0000 1001    CY = 1
RRC A           ;A = 1000 0100    CY = 1
```

RL A ;rotate left A

- In rotate left
  - The 8 bits of the accumulator are rotated left one bit, and
  - Bit D7 exits from the MSB and enters into LSB, D0



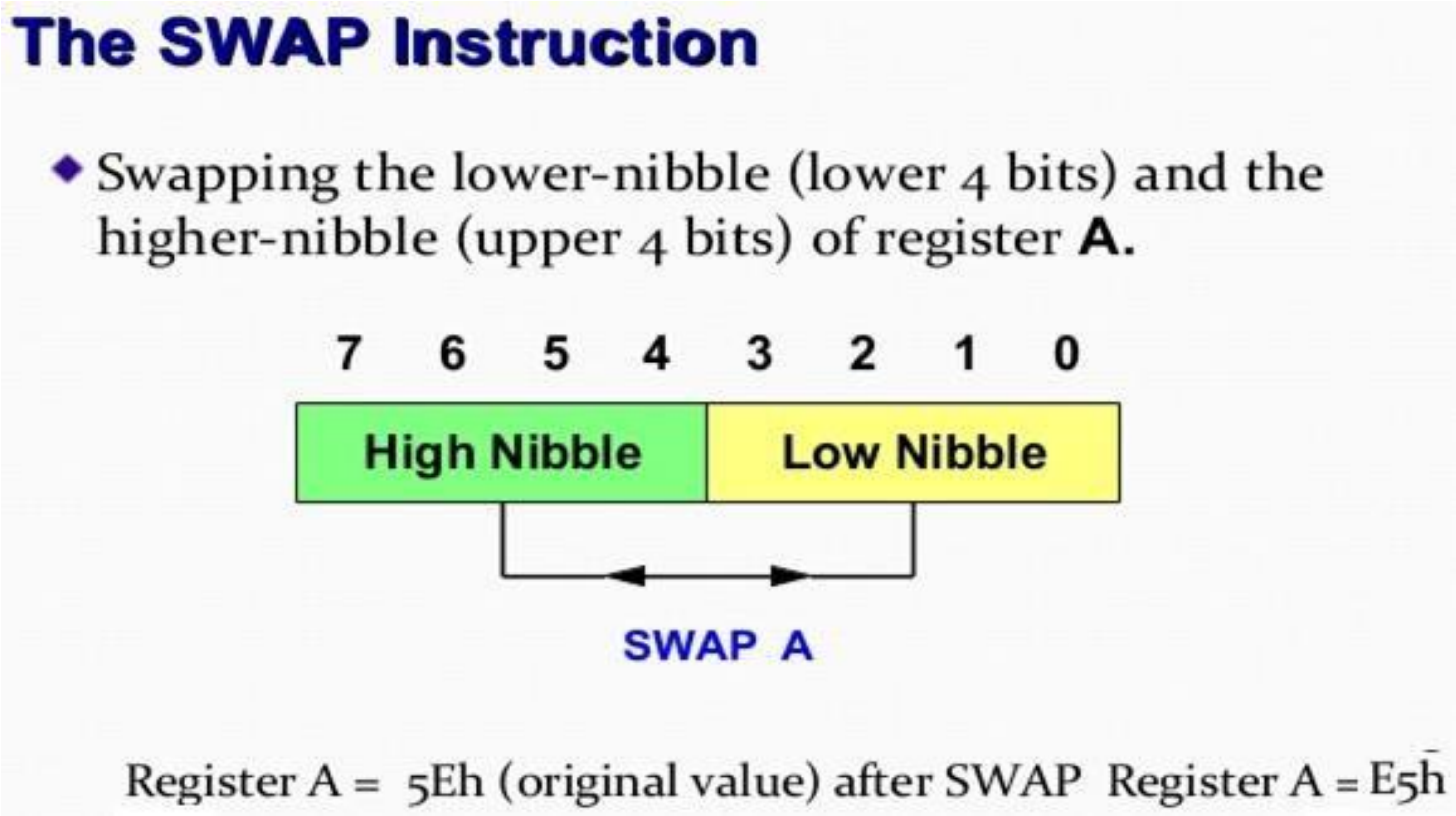
```
MOV  A,#72H     ;A = 0111 0010
RL   A          ;A = 1110 0100
RL   A          ;A = 1100 1001
```

RLC A ;rotate left through carry

- In RLC A
  - Bits are shifted from right to left
  - They exit the MSB and enter the carry flag, and the carry flag enters the LSB

Write a program that finds the number of 1s in a given byte.

```
        MOV     R1,#0
        MOV     R7,#8         ;count=08
        MOV     A,#97H
AGAIN:  RLC     A
        JNC     NEXT          ;check for CY
        INC     R1            ;if CY=1 add to count
NEXT:   DJNZ    R7,AGAIN
```



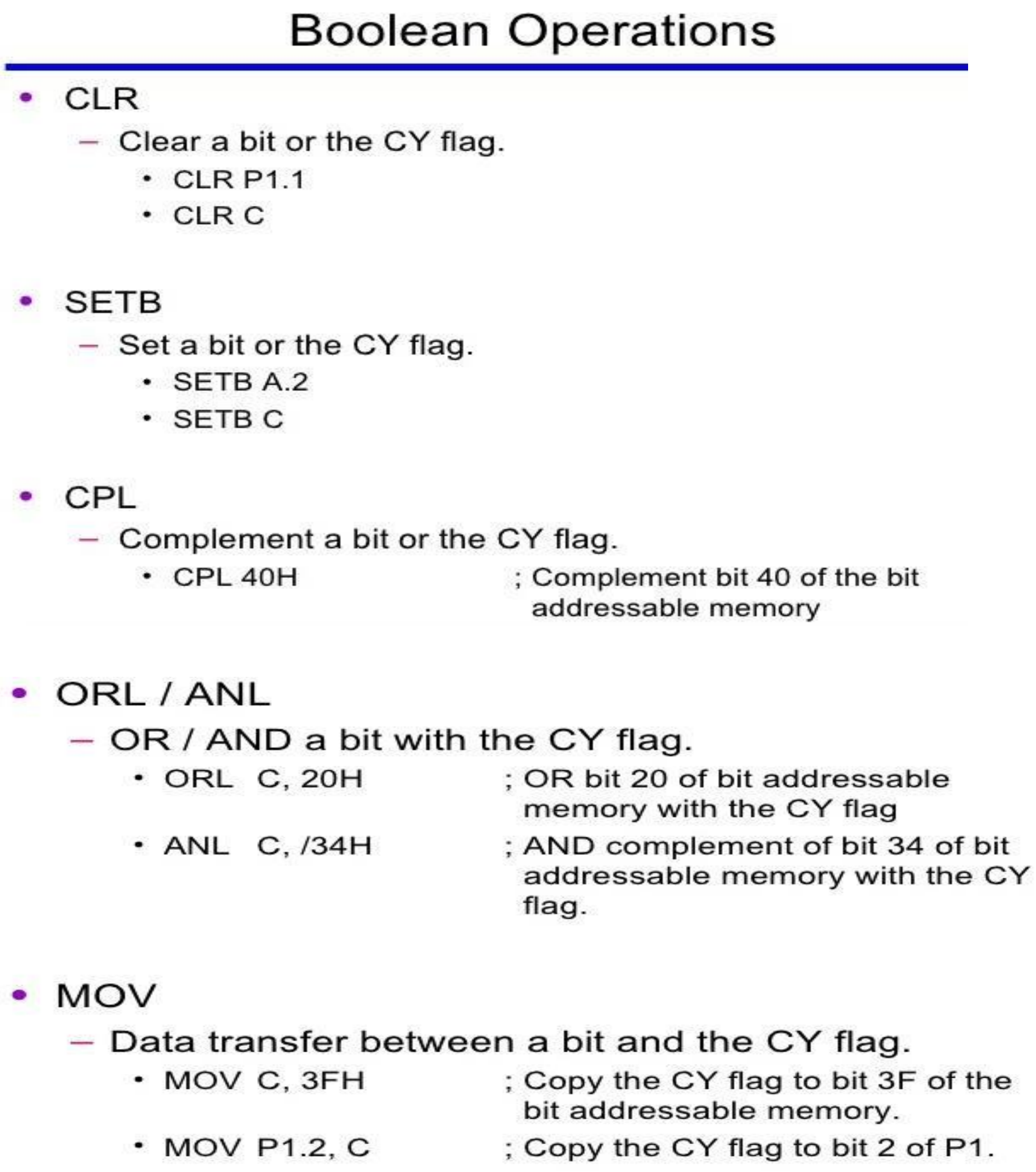
The SWAP Instruction

- Swapping the lower-nibble (lower 4 bits) and the higher-nibble (upper 4 bits) of register **A.**

Register A = 5Eh (original value) after SWAP Register A = E5h

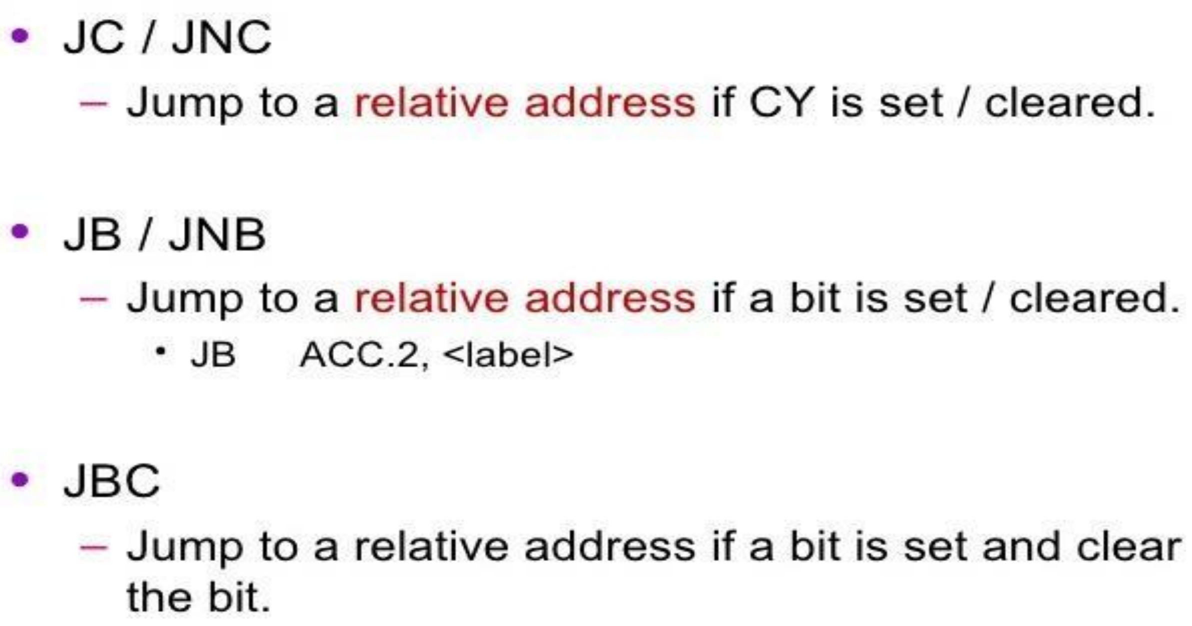
Boolean or bit instruction

Mnemonic	Instruction	Description	# of Bytes	# of Cycles
CLR	C	C ← 0 (C = Carry Bit)	1	1
	Bit	Bit ← 0 (Bit = Direct Bit)	2	1
SET	C	C ← 1	1	1
	Bit	Bit ← 1	2	1
CPL	C	C ← $\overline{C}$	1	1
	Bit	Bit ← $\overline{Bit}$	2	1
ANL	C, /Bit	C ← C. $\overline{Bit}$ (AND)	2	1
	C, Bit	C ← C . Bit (AND)	2	1
ORL	C, /Bit	C ← C + $\overline{Bit}$ (OR)	2	1
	C, Bit	C ← C + Bit (OR)	2	1
MOV	C, Bit	C ← Bit	2	1
	Bit, C	Bit ← C	2	2
JC	rel	Jump is Carry (C) is Set	2	2
JNC	rel	Jump is Carry (C) is Not Set	2	2
JB	Bit, rel	Jump is Direct Bit is Set	3	2
JNB	Bit, rel	Jump is Direct Bit is Not Set	3	2
JBC	Bit, rel	Jump is Direct Bit is Set and Clear Bit	3	2

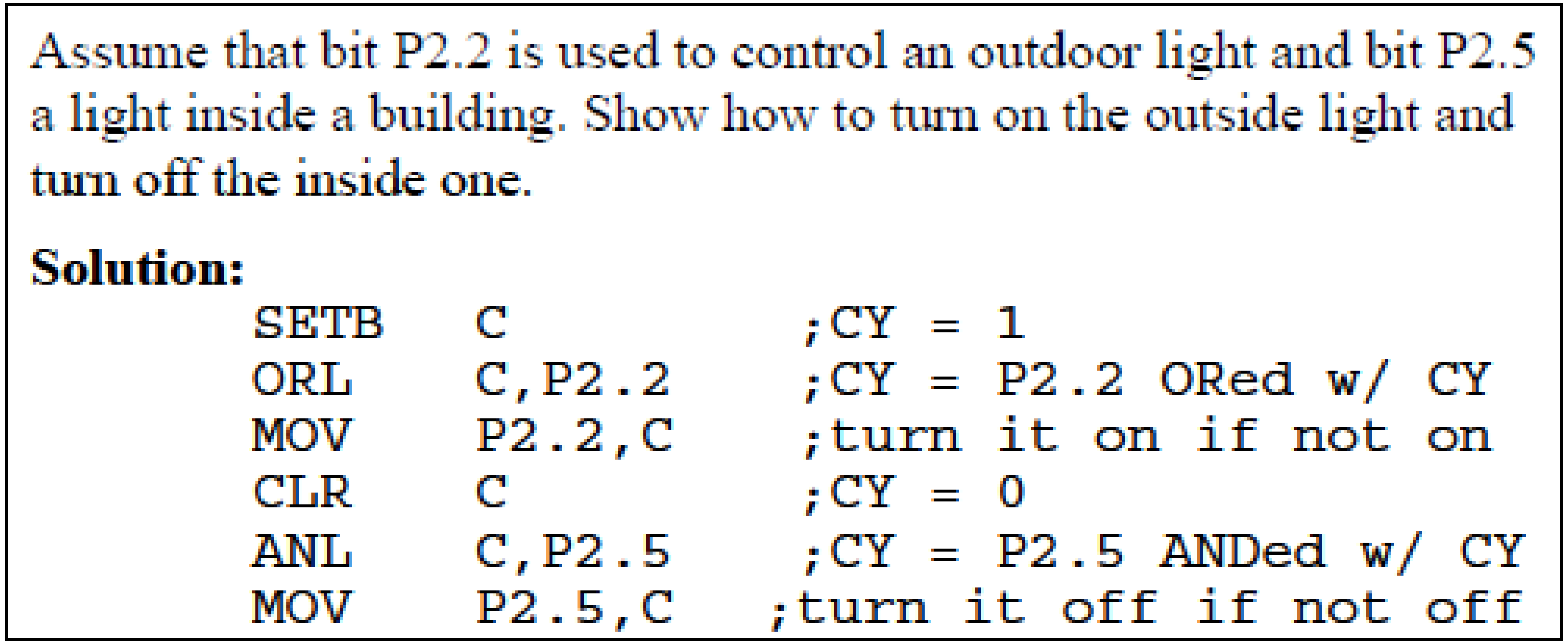


Boolean Operations

- CLR
  - Clear a bit or the CY flag.
    - CLR P1.1
    - CLR C
- SETB
  - Set a bit or the CY flag.
    - SETB A.2
    - SETB C
- CPL
  - Complement a bit or the CY flag.
    - CPL 40H ; Complement bit 40 of the bit addressable memory
- ORL / ANL
  - OR / AND a bit with the CY flag.
    - ORL C, 20H ; OR bit 20 of bit addressable memory with the CY flag
    - ANL C, /34H ; AND complement of bit 34 of bit addressable memory with the CY flag.
- MOV
  - Data transfer between a bit and the CY flag.
    - MOV C, 3FH ; Copy the CY flag to bit 3F of the bit addressable memory.
    - MOV P1.2, C ; Copy the CY flag to bit 2 of P1.



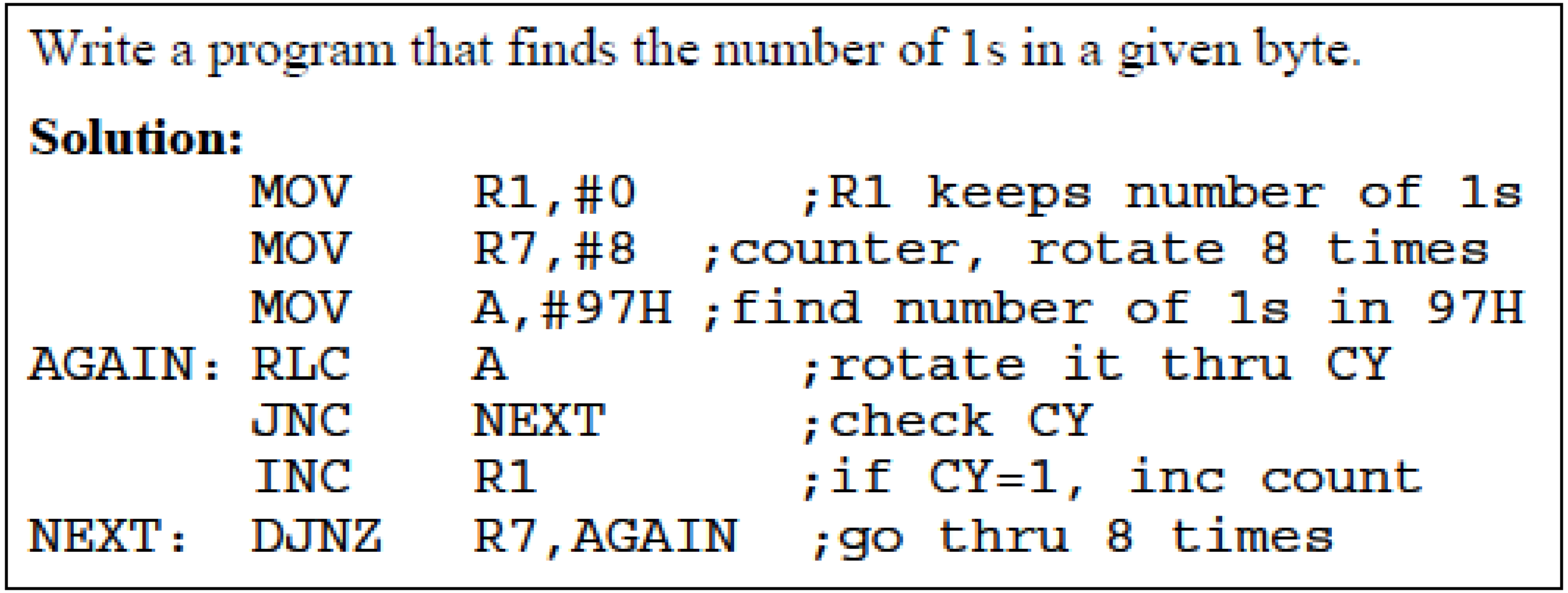
- JC / JNC
  - Jump to a relative address if CY is set / cleared.
- JB / JNB
  - Jump to a relative address if a bit is set / cleared.
    - JB ACC.2, <label>
- JBC
  - Jump to a relative address if a bit is set and clear the bit.



Assume that bit P2.2 is used to control an outdoor light and bit P2.5 a light inside a building. Show how to turn on the outside light and turn off the inside one.

Solution:

```
        SETB    C             ;CY = 1
        ORL     C,P2.2        ;CY = P2.2 ORed w/ CY
        MOV     P2.2,C        ;turn it on if not on
        CLR     C             ;CY = 0
        ANL     C,P2.5        ;CY = P2.5 ANDed w/ CY
        MOV     P2.5,C       ;turn it off if not off
```



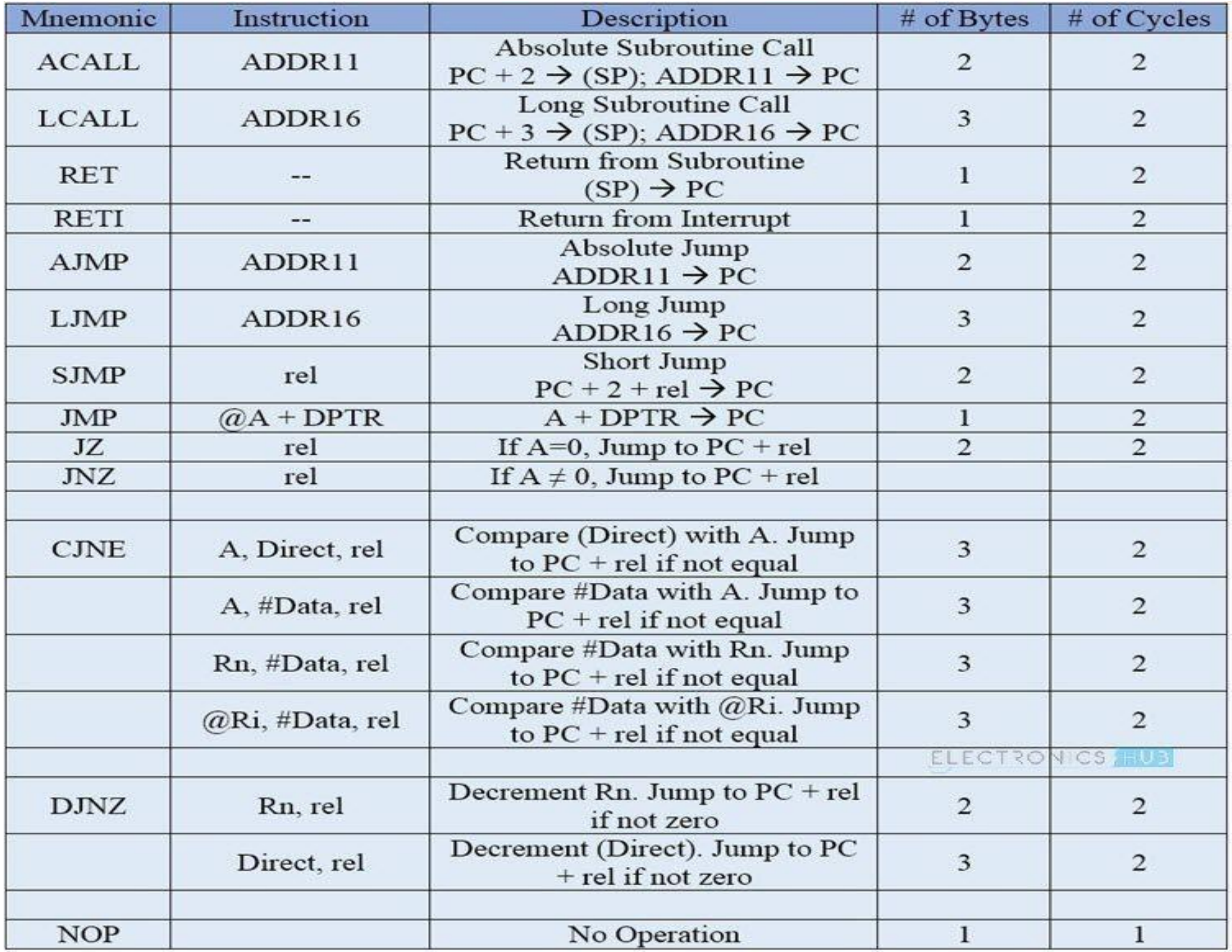
Write a program that finds the number of 1s in a given byte.

Solution:

```
        MOV     R1,#0         ;R1 keeps number of 1s
        MOV     R7,#8  ;counter, rotate 8 times
        MOV     A,#97H ;find number of 1s in 97H
AGAIN:  RLC     A             ;rotate it thru CY
        JNC     NEXT          ;check CY
        INC     R1            ;if CY=1, inc count
NEXT:   DJNZ    R7,AGAIN       ;go thru 8 times
```

Assignment: Find the number of 1's and 0's and store the results in r4 and r5

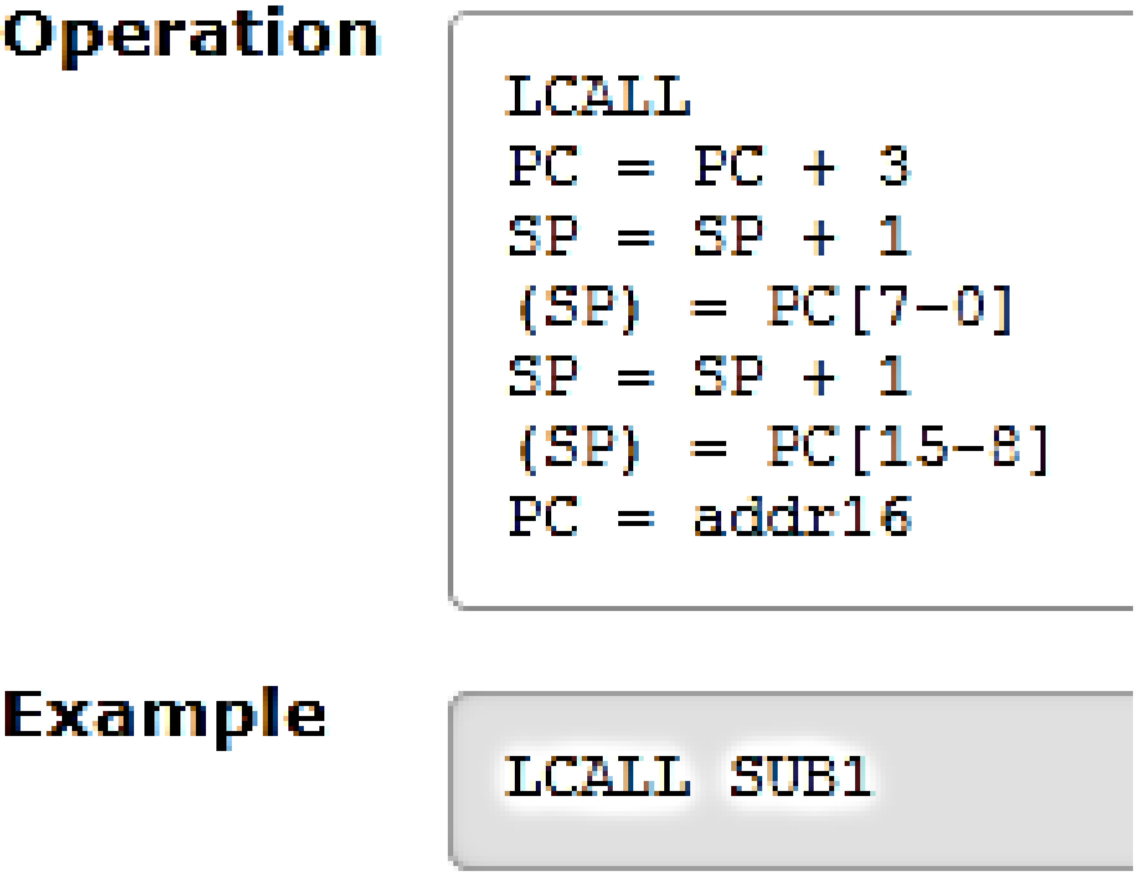
Branch instruction



Mnemonic	Instruction	Description	# of Bytes	# of Cycles
ACALL	ADDR11	Absolute Subroutine Call PC + 2 → (SP); ADDR11 → PC	2	2
LCALL	ADDR16	Long Subroutine Call PC + 3 → (SP); ADDR16 → PC	3	2
RET	--	Return from Subroutine (SP) → PC	1	2
RETI	--	Return from Interrupt	1	2
AJMP	ADDR11	Absolute Jump ADDR11 → PC	2	2
LJMP	ADDR16	Long Jump ADDR16 → PC	3	2
SJMP	rel	Short Jump PC + 2 + rel → PC	2	2
JMP	@A + DPTR	A + DPTR → PC	1	2
JZ	rel	If A=0, Jump to PC + rel	2	2
JNZ	rel	If A ≠ 0, Jump to PC + rel		
CJNE	A, Direct, rel	Compare (Direct) with A. Jump to PC + rel if not equal	3	2
	A, #Data, rel	Compare #Data with A. Jump to PC + rel if not equal	3	2
	Rn, #Data, rel	Compare #Data with Rn. Jump to PC + rel if not equal	3	2
	@Ri, #Data, rel	Compare #Data with @Ri. Jump to PC + rel if not equal	3	2
DJNZ	Rn, rel	Decrement Rn. Jump to PC + rel if not zero	2	2
	Direct, rel	Decrement (Direct). Jump to PC + rel if not zero	3	2
NOP		No Operation	1	1

ACALL – 11BIT ADDRESS == 2k Bytes
LCALL – 16 BIT ADDRESS == unlimited

Syntax:
ACALL delay
LCALL delay
Delay : It’s just a name of subroutine or loop name or subprogram name

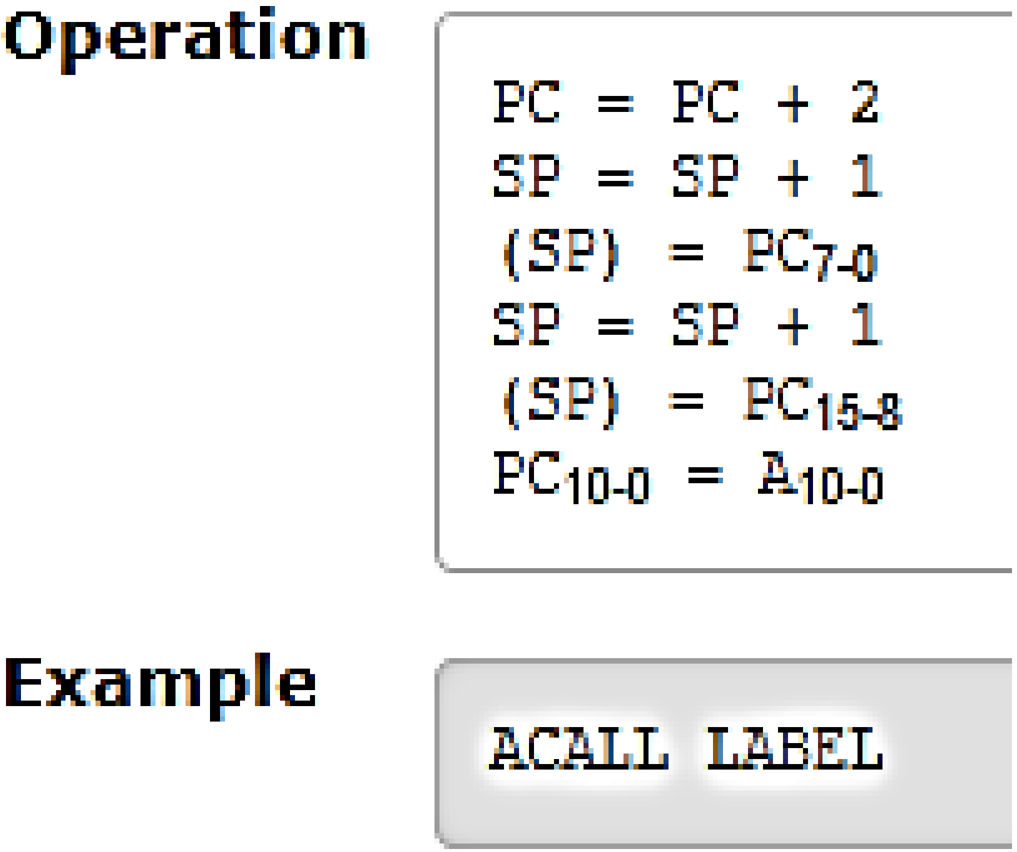


Operation

```
LCALL
PC = PC + 3
SP = SP + 1
(SP) = PC[7-0]
SP = SP + 1
(SP) = PC[15-8]
PC = addr16
```

Example

```
LCALL SUB1
```



Operation

$PC = PC + 2$

$SP = SP + 1$

$(SP) = PC_{7-0}$

$SP = SP + 1$

$(SP) = PC_{15-8}$

$PC_{10-0} = A_{10-0}$

Example

```
ACALL LABEL
```

LCALL =16 BIT address == 0 to 15, ACALL = 11BIT address = 0 to 10,

PC to SP using CALL
And SP to PC using RET

1➔1200h : mov a, r0
2➔1201h: mov b, #12h
3➔1202h:acall delay
3➔1203h: add a, b
4➔end

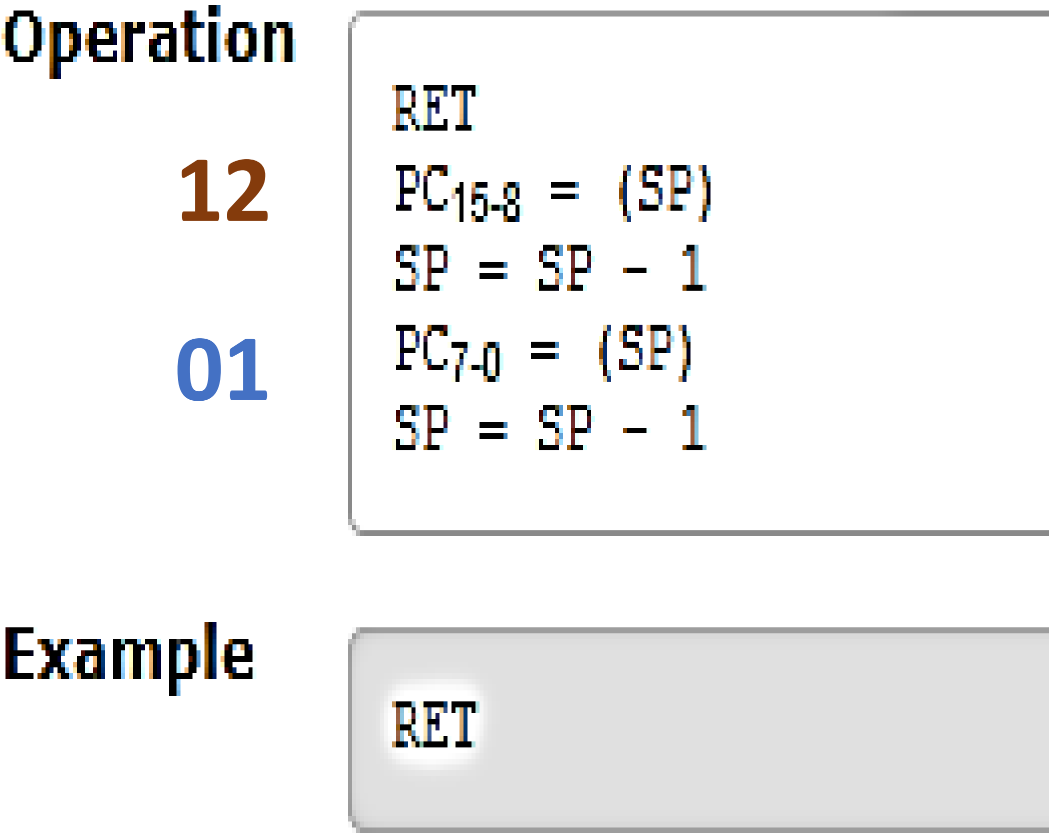
SP = 8bit ➔12 01 h
SP + 1 => 01 h
SP + 1 => 12 h

Subprogram:
1➔delay: mov a, b
Mov 20h, a
ret ==➔ entering to 2 step to tell next address is
1203 fetch

ACALL = 11BIT address = 0 to 10
LCALL =16 BIT address == 0 to 15

- The 8051 provides 2 forms for the return instruction:
  - Return from subroutine – RET
    - Pop the return address from the stack and continue execution there.
  - Return from ISR – RETI
    - Pop the return address from the stack.
    - Restore the interrupt logic to accept additional interrupts at the same priority level as the one just processed.
    - Continue execution at the address retrieved from the stack.
    - RETI must be used for interrupt service routine.

SP → PC → 12 01 h



Operation

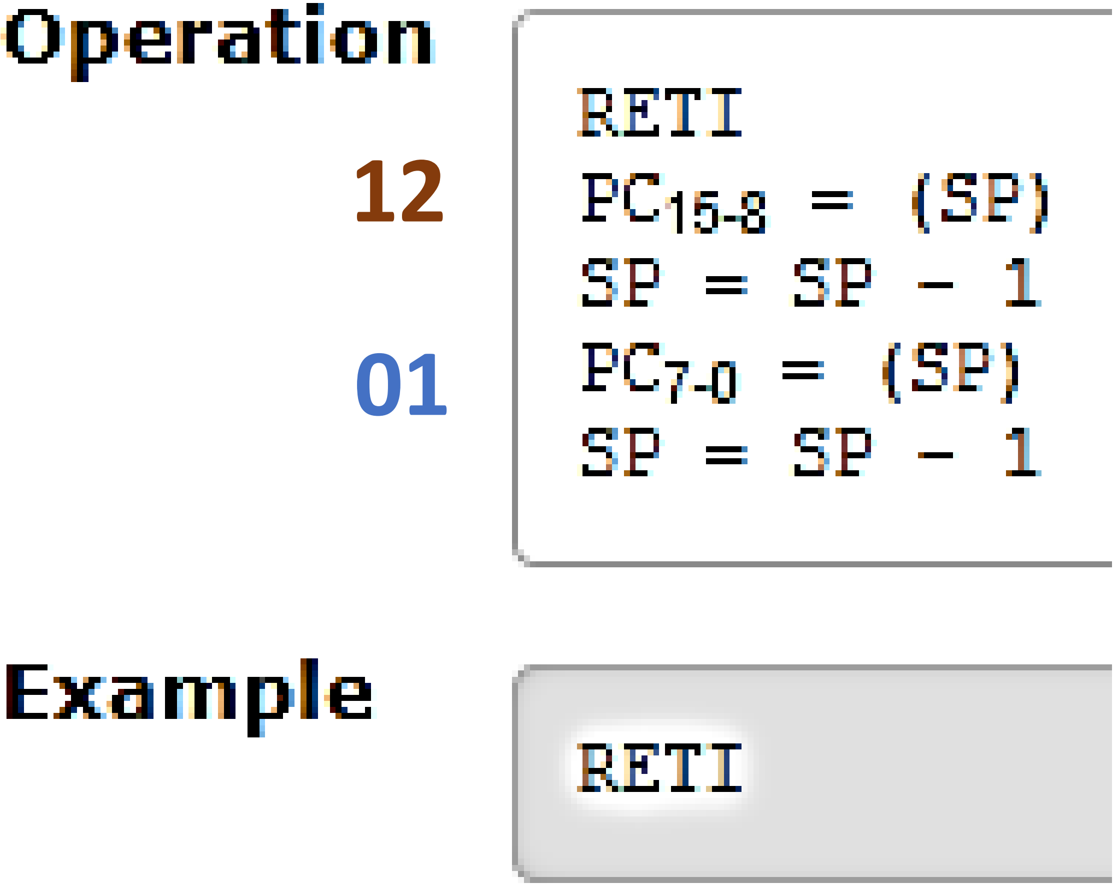
12

01

RET
PC_{15-8} = (SP)
SP = SP - 1
PC_{7-0} = (SP)
SP = SP - 1

Example

RET



Operation

12

01

RETI
PC_{15-8} = (SP)
SP = SP - 1
PC_{7-0} = (SP)
SP = SP - 1

Example

RETI

PC =→12 01 h

SP = 07H 08H 09H

→01H 12H

PC = 12H 01H

SP + 1

SP + 1

09H	**12H**
08H	01H
07H	

CALL

09H	**12H**
08H	01H
07H	

SP - 1

SP - 1

RET

- The 8051 supports 5 different conditional jump instructions.
  - ALL conditional jump instructions use an 8-bit signed offset.
  - Jump on Zero – JZ / JNZ
    - Jump if the A == 0 / A != 0
      - The check is done at the time of the instruction execution.

Ex: Add a, r0 // A = 00H

JZ next1

JNZ vijay

- – Jump on Carry – JC / JNC
  - Jump if the C flag is set / cleared.

JZ next and JNZ next

JC next and JNC next

- – Jump on Bit – JB / JNB
  - Jump if the specified bit is set / cleared.
  - Any addressable bit can be specified.
- – Jump if the Bit is set then Clear the bit – JBC
  - Jump if the specified bit is set.
  - Then clear the bit.

JB next and JNB next1

- Compare and Jump if Not Equal – CJNE
  - Compare the magnitude of the two operands and jump if they are not equal.
    - The values are considered to be unsigned.
    - The Carry flag is set / cleared appropriately.

    - CJNE A, direct, rel
    - CJNE A, #data, rel
    - CJNE Rn, #data, rel
    - CJNE @Ri, #data, rel

- Decrement and Jump if Not Zero – DJNZ
  - Decrement the first operand by 1 and jump to the location identified by the second operand if the resulting value is not zero.

    - DJNZ Rn, rel
    - DJNZ direct, rel

- No Operation
  - NOP

CJNE A, #20H, Rel
Mov b, a
Rel: mov r0, a
Ex: A = 20H

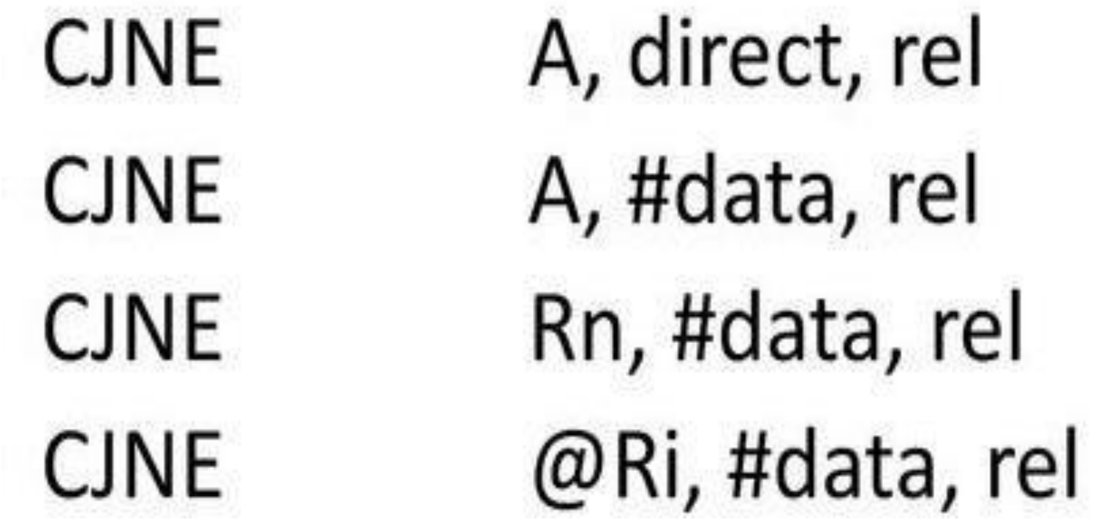
A ⇔ 20H /// EQUAL

Next instruction executed

Ex: A = 15h, A<=>20h ////
UNEQUAL

It go to Rel loop to execute

CJNE A, #20H, Rel
CJNE A, 30H, Vijay
CJNE r2, #05h, raki
CJNE @r1, #0Fh, 4th_ece



CJNE	A, direct, rel
CJNE	A, #data, rel
CJNE	Rn, #data, rel
CJNE	@Ri, #data, rel

- The 8051 provides four different types of unconditional jump(jump to the specified address without any condition) instructions:
  - Short Jump – SJMP

    Here: sjmp Here

    - Uses an 8-bit signed offset relative to the 1st byte of the next instruction.
    - In this jump the target address must be within -128 to +127 bytes(00-FF) of the program counter of the instruction.
    - It is two byte instruction.
  - Long Jump – LJMP

    Here: ljmp Here

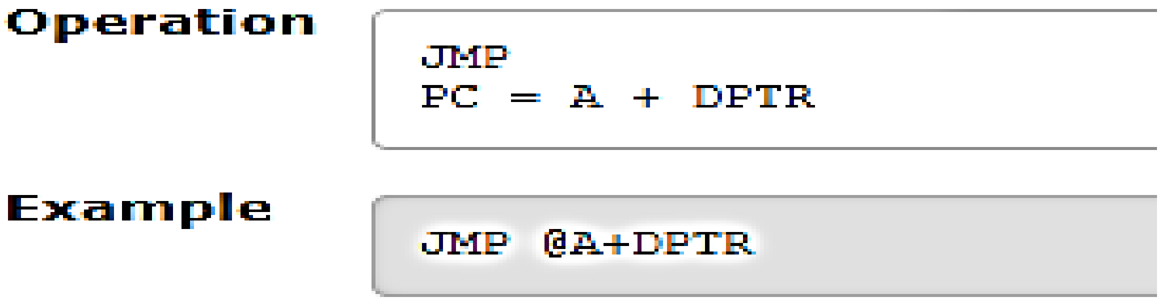
    - Uses a 16-bit address(0000-FFFF).
    - 3 byte instruction capable of referencing any location in the entire 64K of program memory.

- Absolute Jump – AJMP(Jump can be within single page)
  - Uses an 11-bit address.
  - 2 byte instruction
    - The upper 3-bits of the address combine with the 5-bit opcode to form the 1st byte and the lower 8-bits of the address form the 2nd byte.
  - The 11-bit address is substituted for the lower 11-bits of the PC to calculate the 16-bit address of the target.
    - The location referenced must be within the 2K Byte memory page containing the AJMP instruction.

Here: ajmp Here

The JMP instruction transfers execution to the address generated by adding the 8-bit value in the accumulator to the 16-bit value in the DPTR register.

- Indirect Jump – JMP(This instruction is not widely used)
  - JMP @A + DPTR



Operation

```
JMP
PC = A + DPTR
```

Example

```
JMP @A+DPTR
```

A= 08 h, DPTR = 1100 h
JMP @A +DPTR ➔
Jump to address == PC = a + dptr = 1108h

Stack in the 8051

- The stack is a section of RAM used by the CPU to store information temporarily.
- The stack is in the 8051 RAM location 08H to 1FH.
- How the stack is accessed by the CPU ?
- The answer is SP (Stack Pointers) .
  - SP is an 8-bit register.
  - SP always points to the last used location.
  - SP stores the address of top data.

STACK IN 8051

- SP =>Stack pointer => 8bit => used to store address or mem or information.
- It will take Location of last step.
- SP => Starting address = 07h => 08h to 1Fh
- Push => store mem or address or information and Location to SP are
- Pop=> Retrieve mem. Location from SP
- Push and Pop are playing with only Register bank
- => RB0, RB1, RB2, RB3

Default=> RB0 register bank is used.

Mov r2, #0f1h

Mov r4, #1ah

Mov r5, #15h

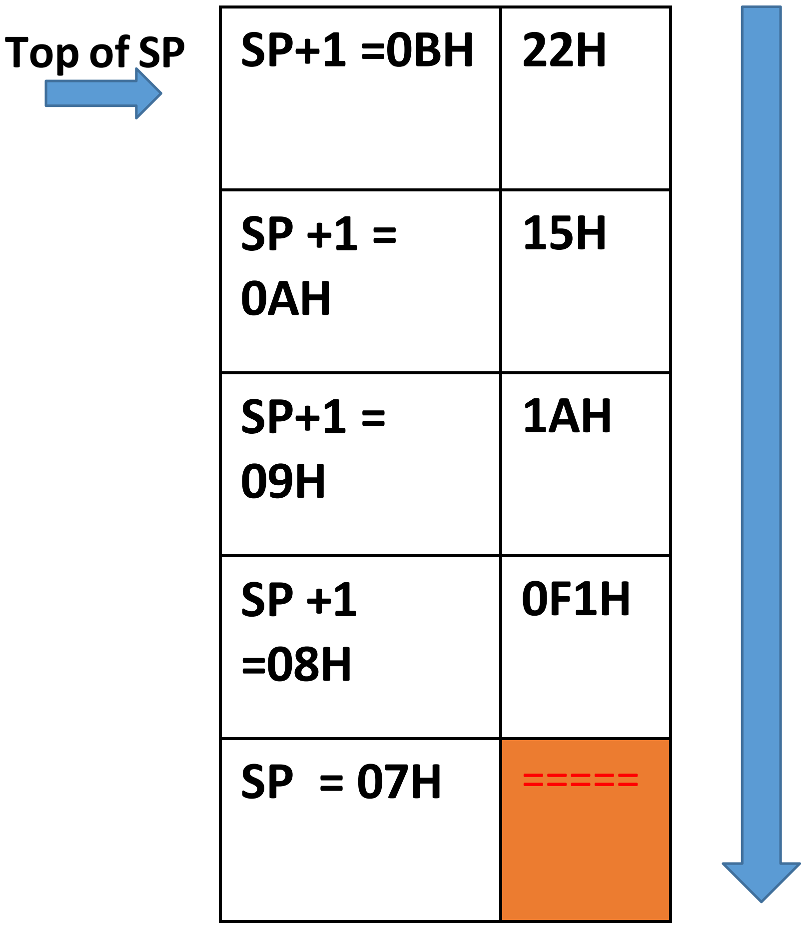
Mov r7, #22h

Push 2 → r2

Push 4 → r4

Push 5 → r5

Push 7 → r7



Top of SP

SP+1 =0BH	22H
SP +1 = 0AH	15H
SP+1 = 09H	1AH
SP +1 =08H	0F1H
SP = 07H	=====

Pop 7 → r7

Pop 5 → r5

Pop 4 → r4

Pop 2 → r2

If I am using different register bank then suppose RB2

RS1 = PSW.4 =1, RS0 = PSW.3 = 0 , SETB, CLR instruction is to be used.

SETB PSW.4

CLR PSW.3

Mov r2, #0f1h

Mov r4, #1ah

Mov r5, #15h

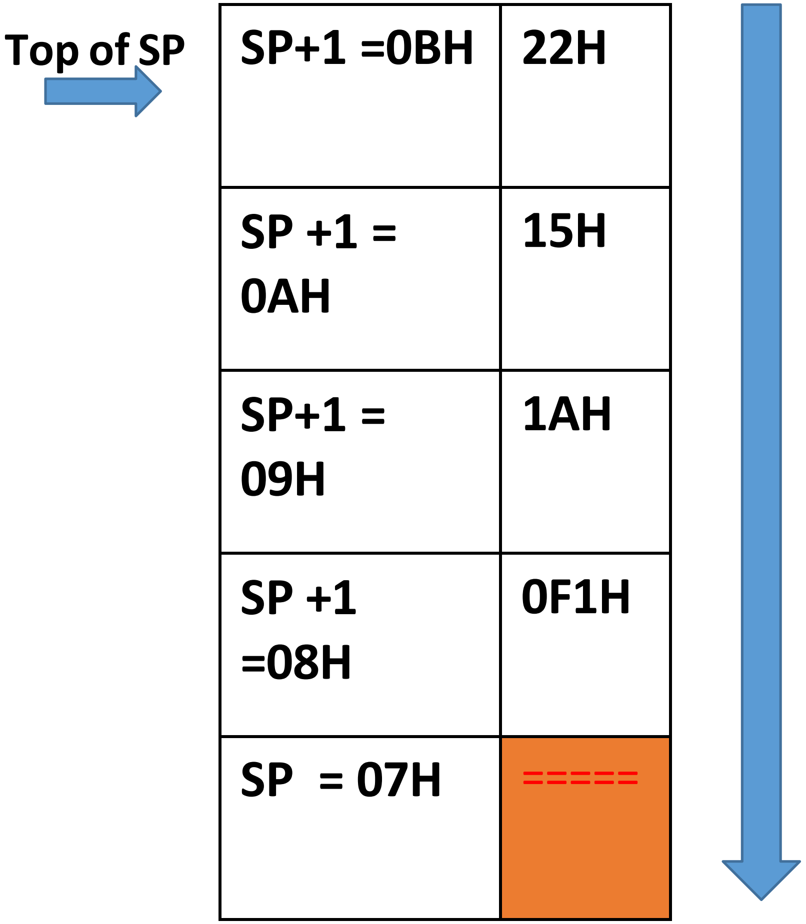
Mov r7, #22h

Push 2 → r2

Push 4 → r4

Push 5 → r5

Push 7 →r7



Top of SP

SP+1 =0BH	22H
SP +1 = 0AH	15H
SP+1 = 09H	1AH
SP +1 =08H	0F1H
SP = 07H	=====

Pop 7 → r7

Pop 5 → r5

Pop 4 → r4

Pop 2 →r2

If I am using different register bank then suppose RB3

RS1 = PSW.4 =1, RS0 = PSW.3 = 1 , SETB, CLR instruction is to be used.

SETB PSW.4

SETB PSW.3

Mov r2, #0f1h

Mov r4, #1ah

Mov r5, #15h

Mov r7, #22h

Push 2 → r2

Push 4 → r4

Push 5 → r5

Push 7 → r7

Top of SP

SP+1 =0BH	**22H**
SP +1 = 0AH	**15H**
SP+1 = 09H	**1AH**
SP +1 =08H	**0F1H**
SP = 07H	=====

Pop 7 → r7

Pop 5 → r5

Pop 4 → r4

Pop 2 → r2

If I am using different register bank then suppose RB1

RS1 = PSW.4 =0, RS0 = PSW.3 = 1 , SETB, CLR instruction is to be used.

CLR PSW.4

SETB PSW.3

Mov r2, #0f1h

Mov r4, #1ah

Mov r5, #15h

Mov r7, #22h

Push 2 → r2

Push 4 → r4

Push 5 → r5

Push 7 →r7

Top of SP

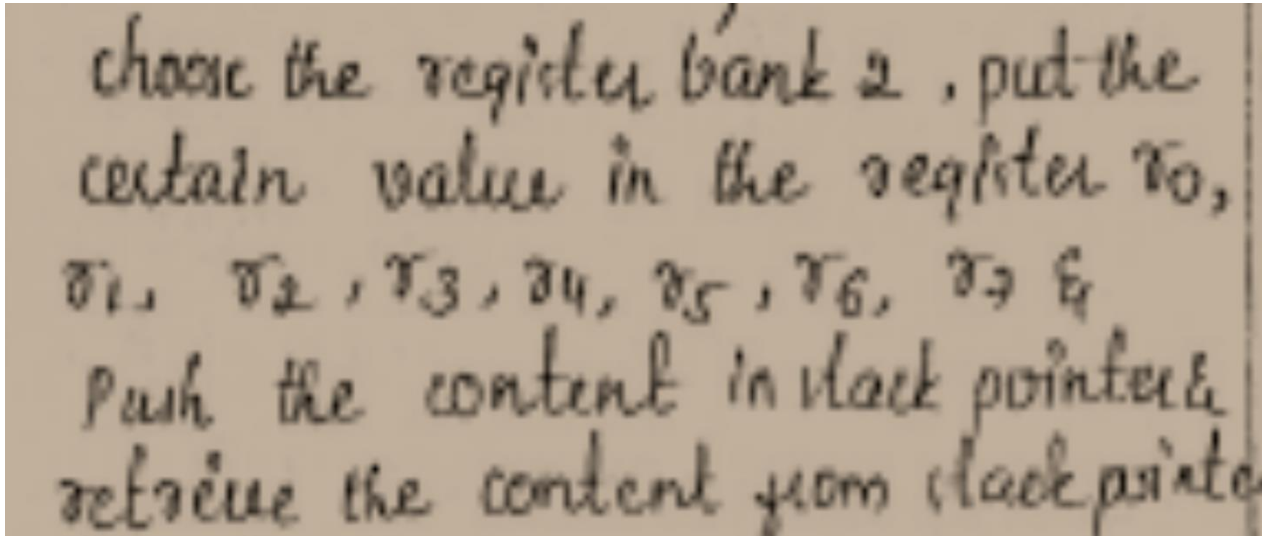
SP+1 =0BH	22H
SP +1 = 0AH	15H
SP+1 = 09H	1AH
SP +1 =08H	0F1H
SP = 07H	=====

Pop 7 → r7

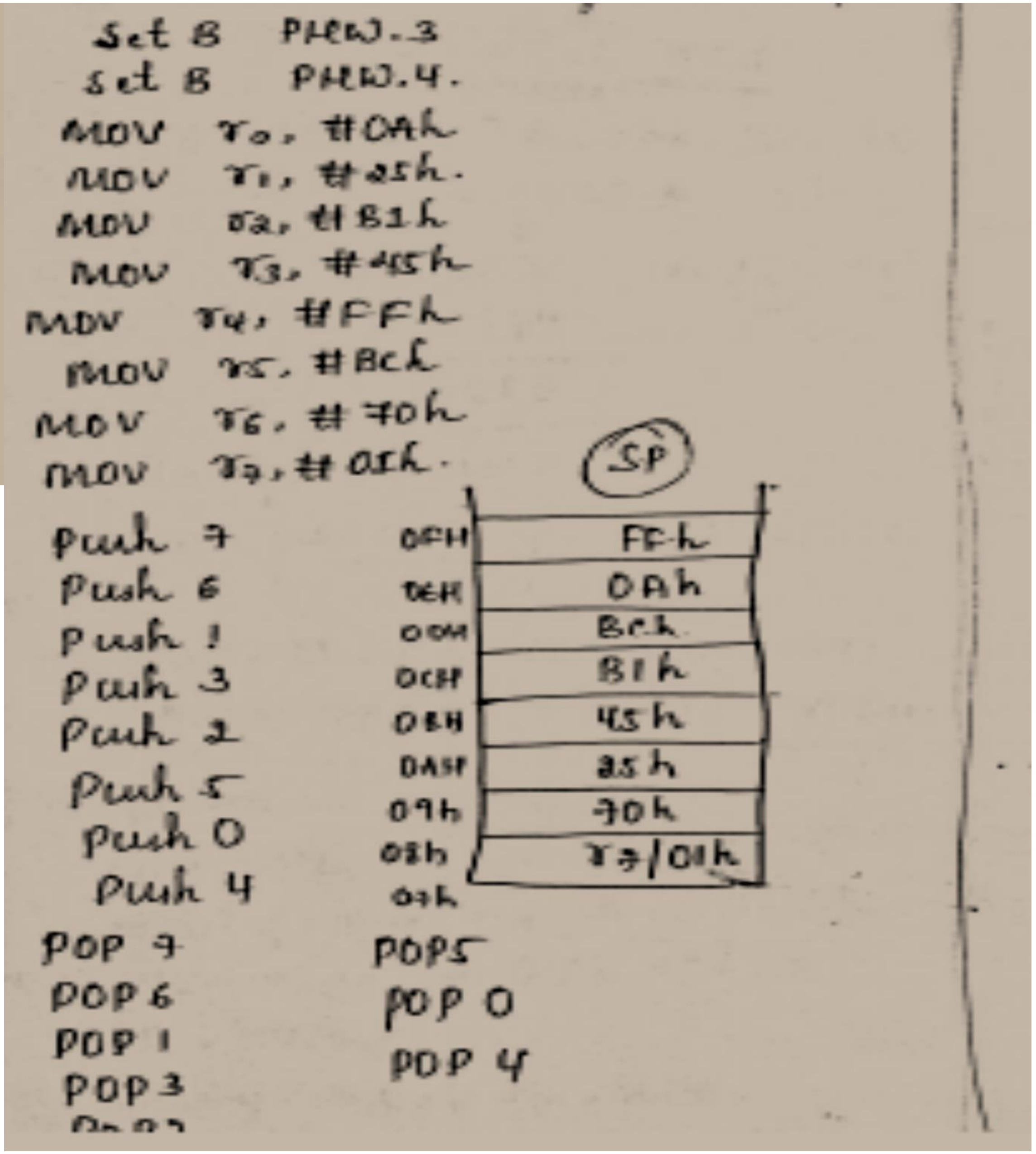
Pop 5 → r5

Pop 4 → r4

Pop 2 →r2



choose the register bank 2, put the certain value in the register r0, r1, r2, r3, r4, r5, r6, r7 & Push the content in stack pointer & retrieve the content from stackpointer



```
Set B  PSW.3
set B  PSW.4.
MOV  r0, #0Ah
MOV  r1, #25h.
MOV  r2, #81h
MOV  r3, #45h
MOV  r4, #FFh
MOV  r5, #BCh
MOV  r6, #70h
MOV  r7, #01h.

Push 7
Push 6
Push 1
Push 3
Push 2
Push 5
Push 0
Push 4

POP 7        POP5
POP 6        POP 0
POP 1        POP 4
POP 3
```

SP

0FH	FFh
0EH	0Ah
0DH	BCh
0CH	81h
0BH	45h
0AH	25h
09h	70h
08h	r7/01h
07h	

DELAY CALCULATION

Microseconds = 2 X 255 X t ➔ … µsec

Milliseconds = 2 X 255 X t X unknown number ➔ … msec

Seconds = 255 X 255 X t X unknown number ➔ … sec

22MHz, 200msec

T = 1/[22MHz / 12] = 0.5454 µsec

2 x 255 x t x unknown value = …. msec

0.274 msec x 720 == 200.2msec

22MHz, 10sec

T = 1/[22MHz / 12] = 0.5454 µsec

255 x 255 x 0.5454µsec x 282 = 9.993 sec = 10sec

22MHz, 100msec

T = 1/[22MHz / 12] = 0.5454 µsec

2 x 255 x t x unknown value = …. msec

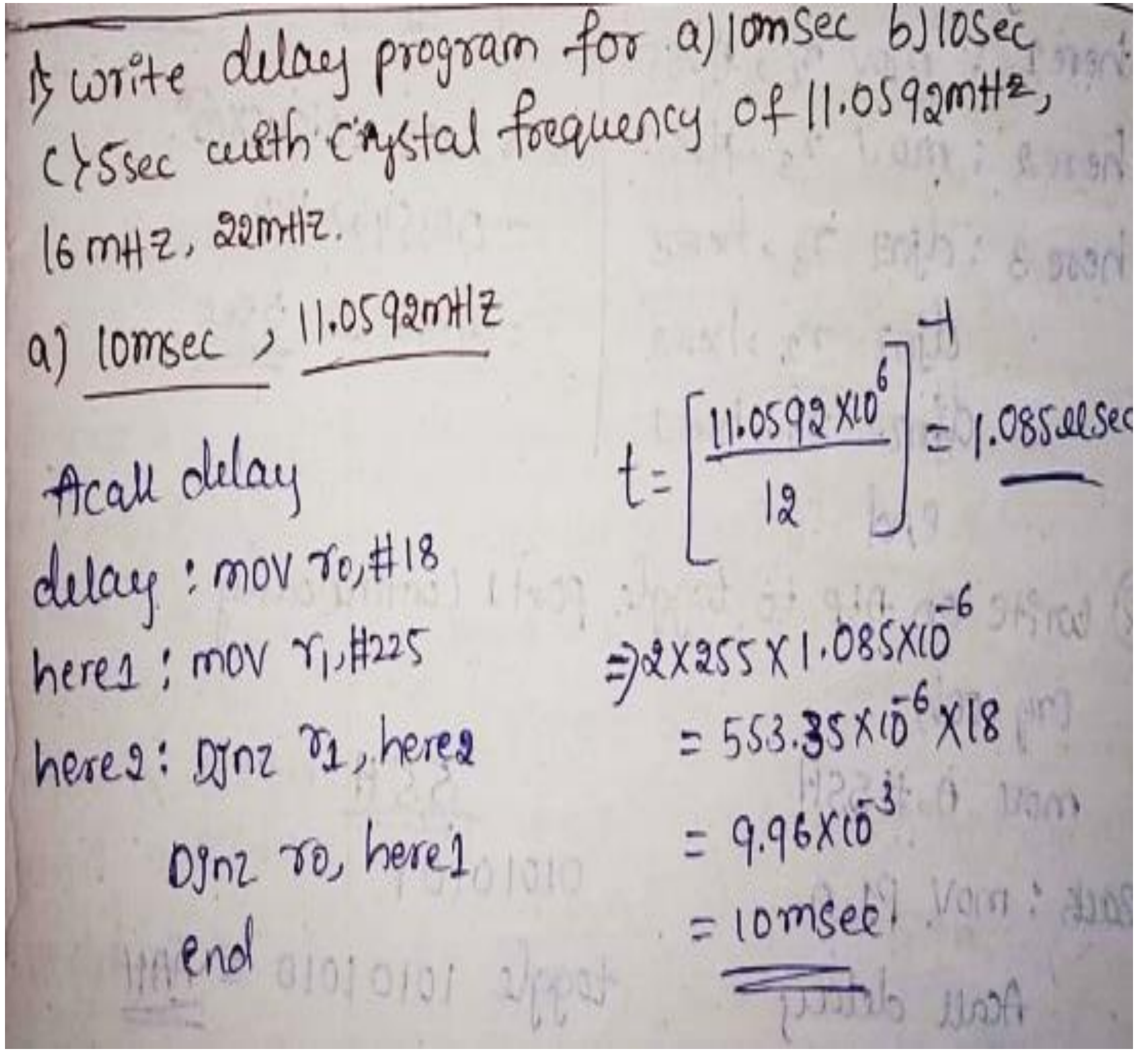
0.274 msec x 360 == 100msec

22MHz, 20sec

T = 1/[22MHz / 12] = 0.5454 µsec

255 x 255 x t x unknown value = …. msec

0.0355 sec x 564 == 19.99 = 20sec



1) write delay program for a) 10msec b) 10sec c) 5sec with crystal frequency of 11.0592mHz, 16mHz, 22mHz.

a) 10msec , 11.0592mHz

```
Acall delay
delay : mov r0,#18
here1 ; mov r1,#225
here2: Djnz r1, here2
       Djnz r0, here1
       end
```

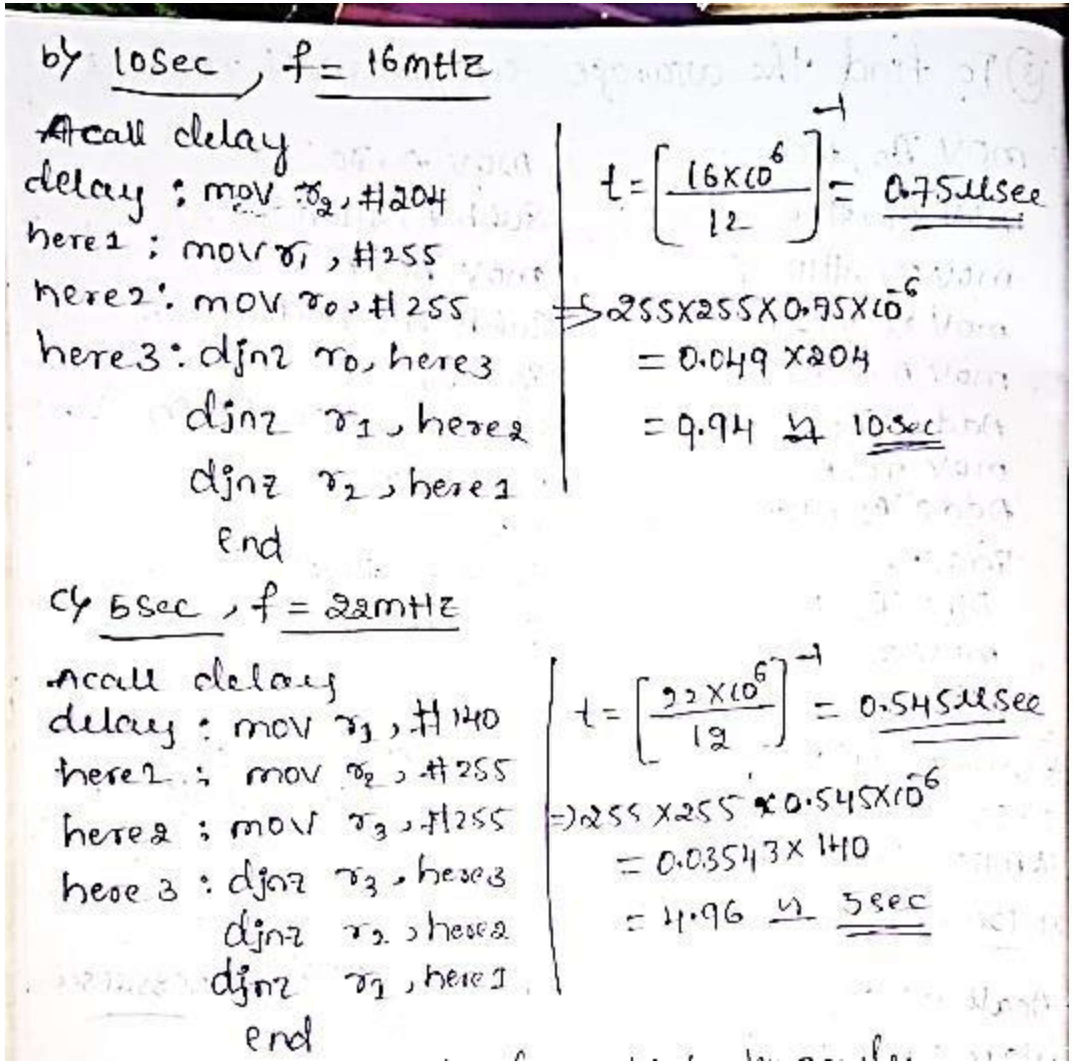
$$t = \left[\frac{11.0592 \times 10^{6}}{12}\right]^{-1} = 1.085 \mu sec$$

$$\Rightarrow 2 \times 255 \times 1.085 \times 10^{-6}$$

$$= 553.35 \times 10^{-6} \times 18$$

$$= 9.96 \times 10^{-3}$$

$$= 10 msec$$



b) 10sec , f = 16mHz

```
Acall delay
delay : mov r2, #204
here1 : mov r1 , #255
here2 : mov r0 , #255
here3 : djnz r0, here3
        djnz r1 , here2
        djnz r2 , here1
        end
```

$$t = \left[\frac{16 \times 10^{6}}{12}\right]^{-1} = 0.75 \mu sec$$

$$\Rightarrow 255 \times 255 \times 0.75 \times 10^{-6}$$

$$= 0.049 \times 204$$

$$= 9.94 \simeq 10 sec$$

c) 5sec , f = 22mHz

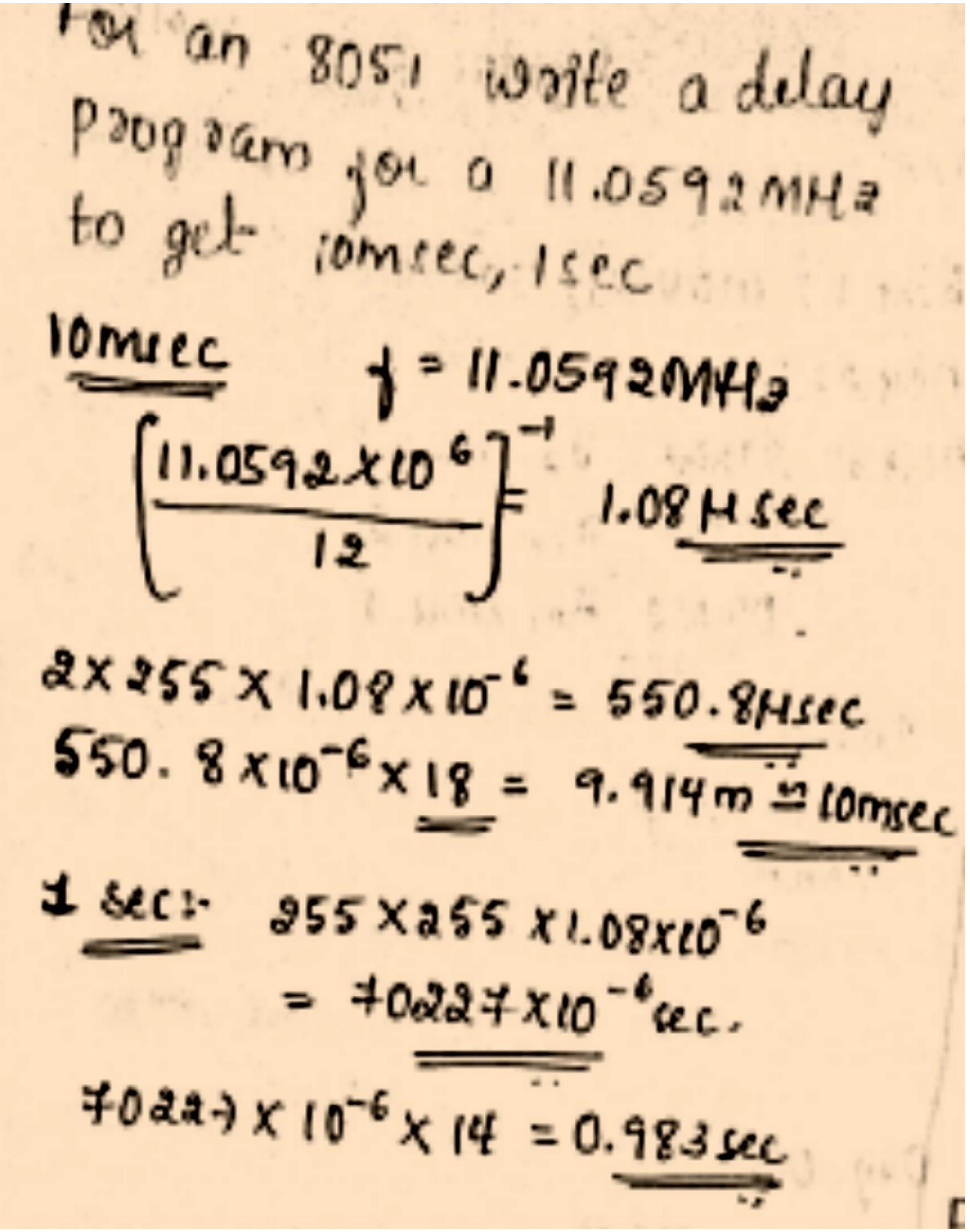
```
Acall delay
delay : mov r1 , #140
here1 : mov r2 , #255
here2 : mov r3 , #255
here3 : djnz r3 , here3
        djnz r2 , here2
        djnz r1 , here1
        end
```

$$t = \left[\frac{22 \times 10^{6}}{12}\right]^{-1} = 0.545 \mu sec$$

$$\Rightarrow 255 \times 255 \times 0.545 \times 10^{-6}$$

$$= 0.03543 \times 140$$

$$= 4.96 \simeq 5 sec$$



For an 8051 write a delay program for a 11.0592MHz to get 10msec, 1sec

10msec

$f = 11.0592\text{MHz}$

$$\left[\frac{11.0592\times10^{6}}{12}\right]^{-1} = 1.08\,\mu\text{sec}$$

$2\times255\times1.08\times10^{-6} = 550.8\,\mu\text{sec}$

$550.8\times10^{-6}\times18 = 9.914\text{m} \simeq 10\text{msec}$

1 sec:- $255\times255\times1.08\times10^{-6}$

$= 70227\times10^{-6}\,\text{sec.}$

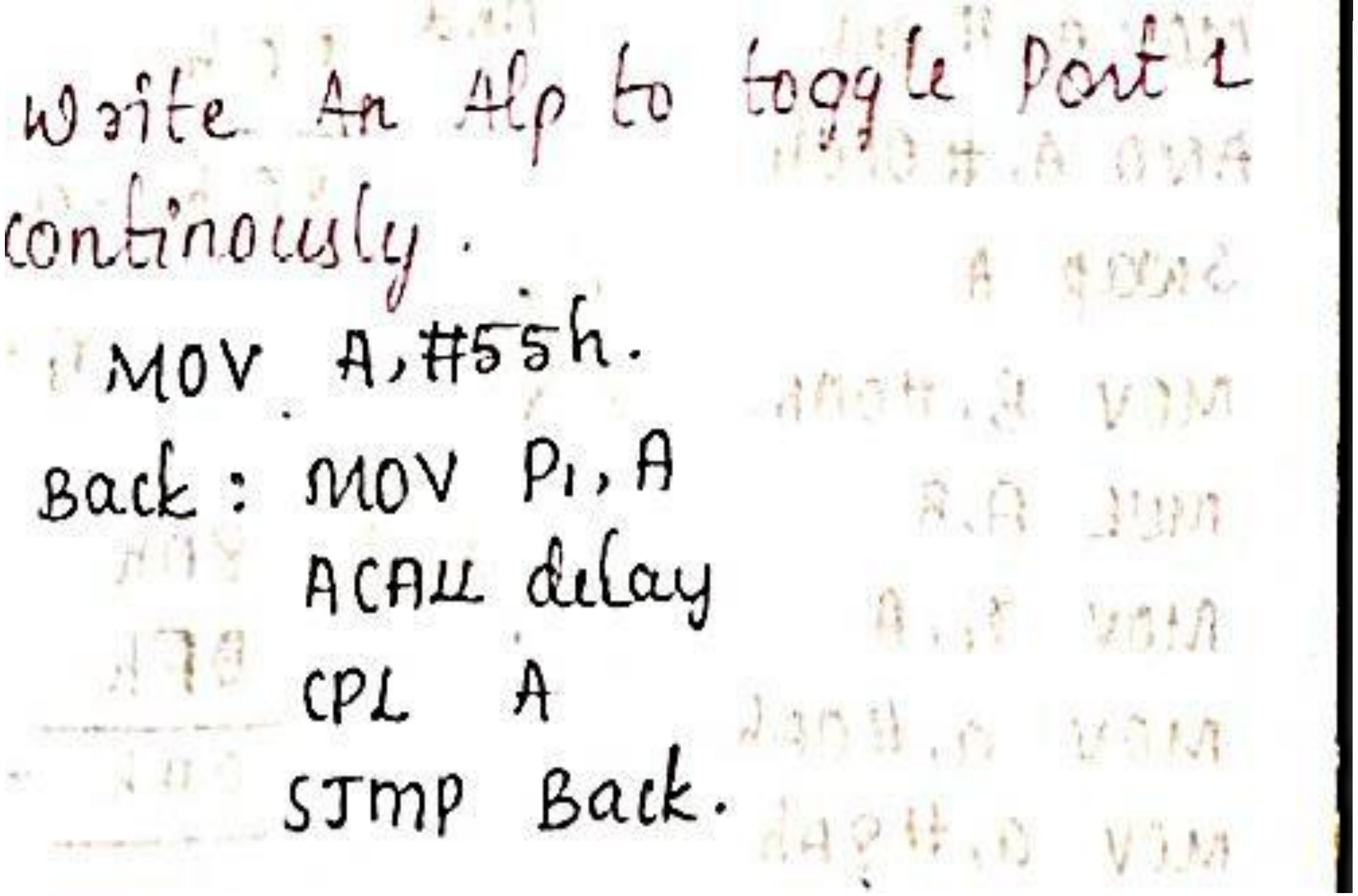
$70227\times10^{-6}\times14 = 0.983\,\text{sec}$

Program: **10msec**

```
 ACALL delay.
delay: MOV r0, #18
here1: MOV r1, #255.
here2: DJNZ r1, here2
       DJNZ r0, here1
        RET
```

1 sec:-

```
 ACALL delay-
 delay: MOV r0, #14.
 here1: MOV r1, #255
 here2: MOV r2, #255.
here3:  DJNZ r2, here3.
        DJNZ r1, here2.
        DJNZ r0, here1
         RET
```



Write An Alp to toggle Port 1 continously.

```
 MOV A,#55h.
Back: MOV P1, A
      ACALL delay
      CPL   A
      SJMP  Back.
```

Program: 10msec

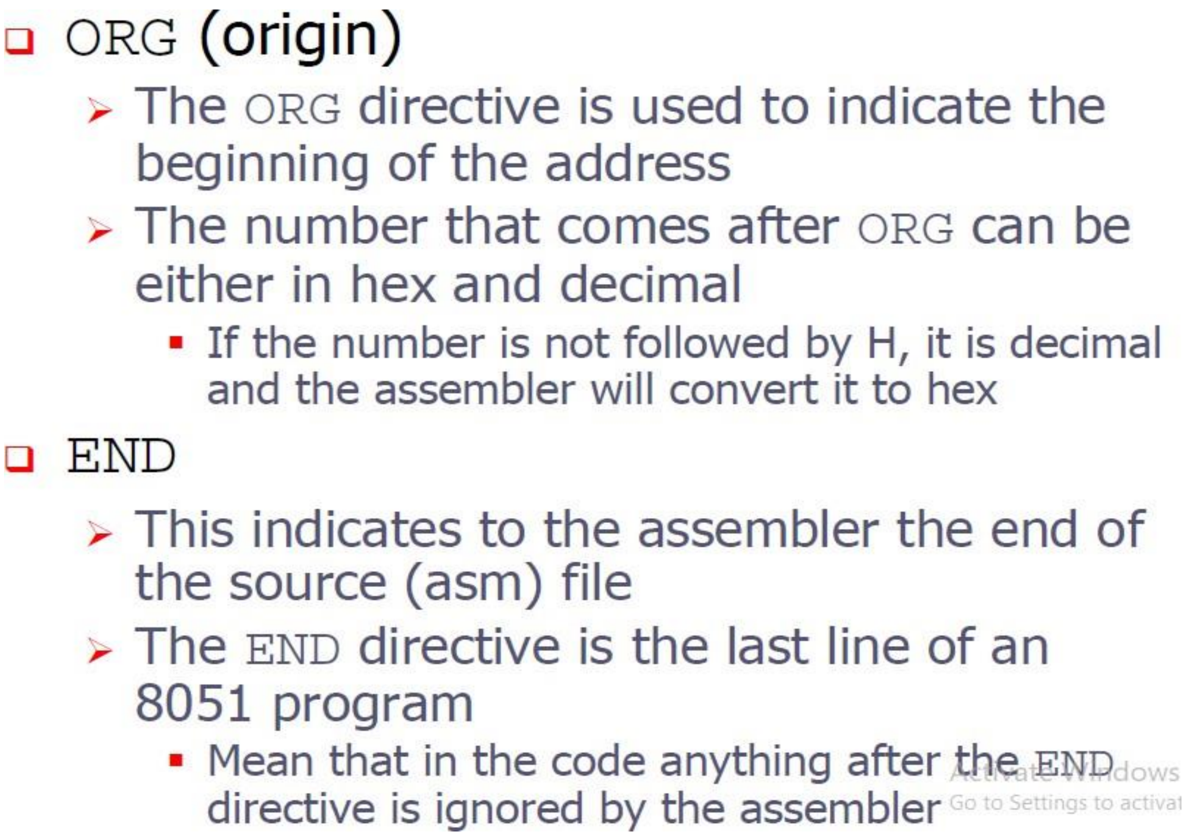
```
 ACALL delay.
delay: MOV r0, #18
here1: MOV r1, #255.
here2:  DJNZ r1, here2
        DJNZ r0, here1
         RET
```

1 sec:-

```
 ACall delay-
 delay: MOV r0, #14.
 here1: MOV r1, #255
 here2: MOV r2,#255.
here3:  DJNZ r2, here3.
        DJNZ r1, here2.
        DJNZ r0, here1
         RET
```

Module 2 Theory part end

ASSEMBLER DIRECTIVES OF 8051



- ORG (origin)
  - The ORG directive is used to indicate the beginning of the address
  - The number that comes after ORG can be either in hex and decimal
    - If the number is not followed by H, it is decimal and the assembler will convert it to hex
- END
  - This indicates to the assembler the end of the source (asm) file
  - The END directive is the last line of an 8051 program
    - Mean that in the code anything after the END directive is ignored by the assembler

ORG 00H

ORG 30H → STARTS AT 30H Address line

ORG 1000H → 16 bit address line

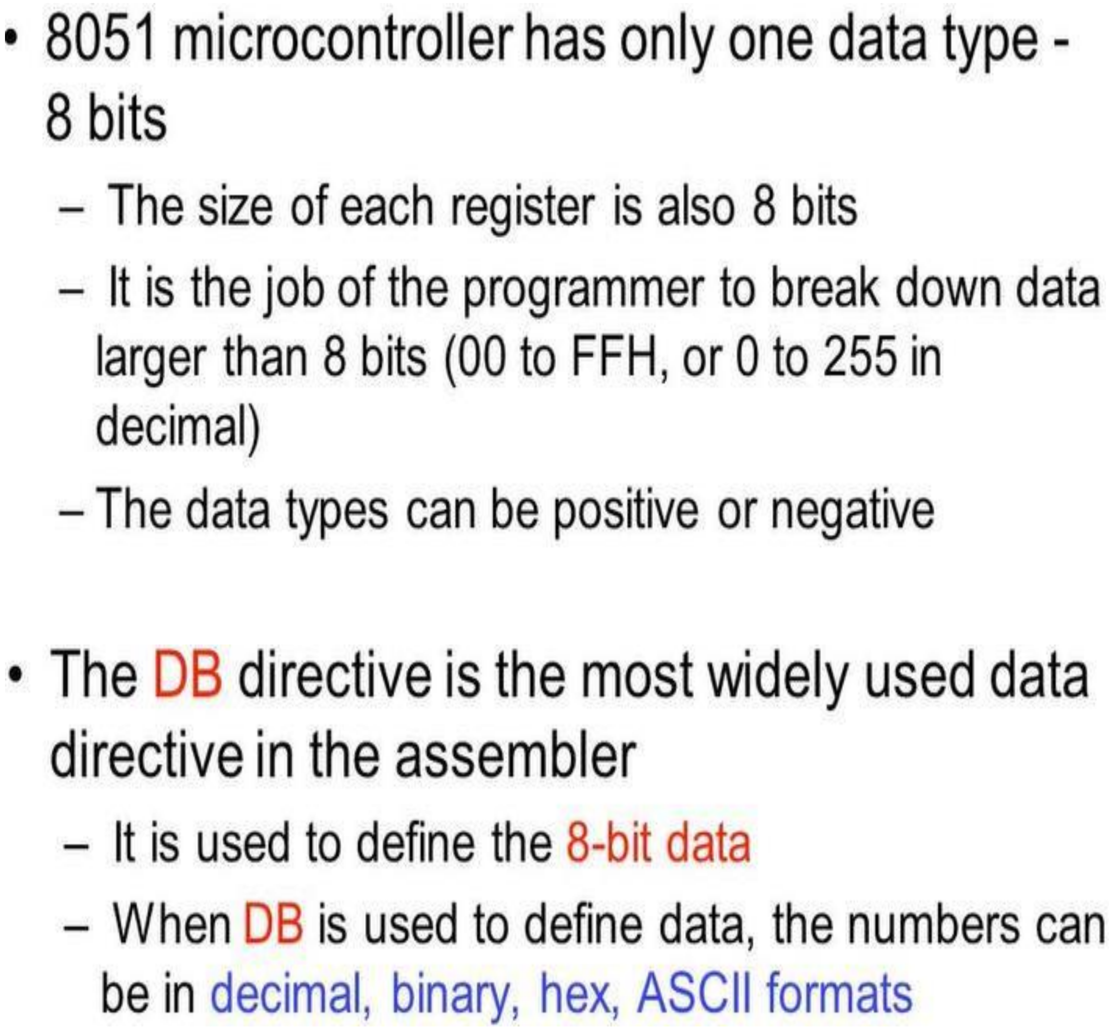
ORG 30 → Here no ‘H’ → 30 is decimal numbered address

- EQU
  - Used to create symbols that can be used to represent registers, numbers, and addresses

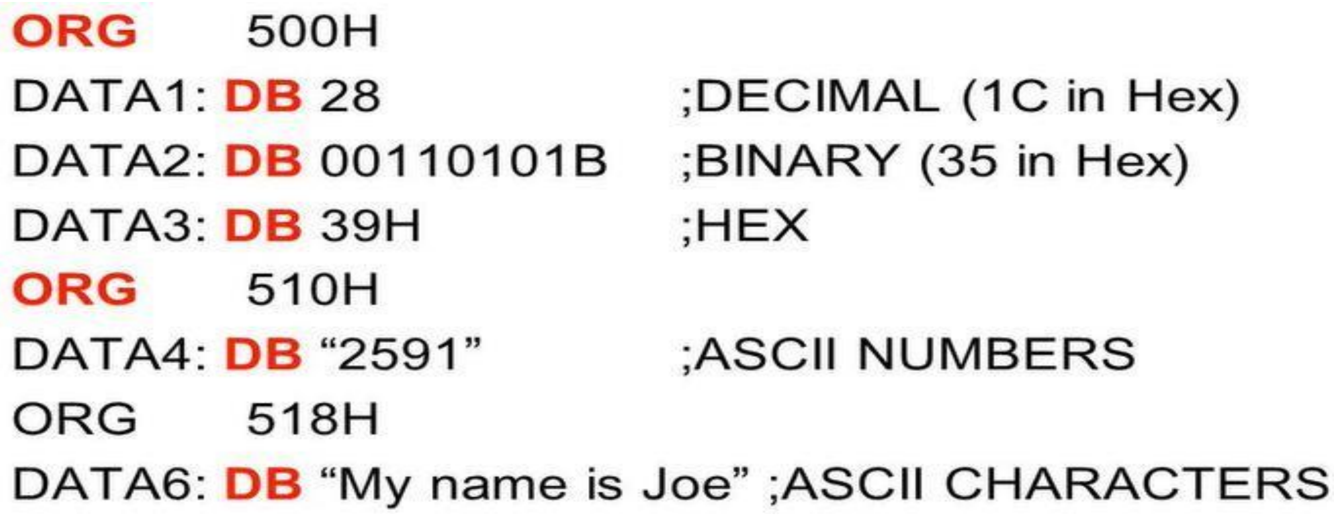
  - This is used to define a constant without occupying a memory location

```
COUNT      EQU      25
... ....
MOV        R3, #COUNT    /// R3 = 25H
```

```
MOTOR  EQU 12H
....
....
....
MOV P1, MOTOR  /// P1 = 12H
```



- 8051 microcontroller has only one data type - 8 bits
  - The size of each register is also 8 bits
  - It is the job of the programmer to break down data larger than 8 bits (00 to FFH, or 0 to 255 in decimal)
  - The data types can be positive or negative
- The DB directive is the most widely used data directive in the assembler
  - It is used to define the 8-bit data
  - When DB is used to define data, the numbers can be in decimal, binary, hex, ASCII formats



```
ORG      500H
DATA1: DB 28              ;DECIMAL (1C in Hex)
DATA2: DB 00110101B       ;BINARY (35 in Hex)
DATA3: DB 39H             ;HEX
ORG      510H
DATA4: DB "2591"          ;ASCII NUMBERS
ORG      518H
DATA6: DB "My name is Joe" ;ASCII CHARACTERS
```

DB → Represent decimal, binary, hexadecimal numbers.
Also ASCII numbers or characters
" ….. " == each character will have 8 bit
Example: My == 8 + 8 =16 bit
"25" == 2 = 8bit, 5 = 8bit
Space → 8bit
Syntax: Datax: DB ---
x = 1, 2, ….

1. Write a assemble language program to transfer a block of bytes of data from one location to another location both internally

```
Org 00h
Mov r3, #04h
Mov r0, #40h
Mov r1, #50h

Next: Mov a, @r0
Mov @r1, a
Inc r0
Inc r1

Djnz r3, next

Here:sjmp here
end
```

Data memory address	Contents
D:0x040	01h
D:0x041	02h
D:0x042	03h
D:0x043	04h

Data memory address	Contents
D:0x050	00h
D:0x051	00h
D:0x052	00h
D:0x053	00h

Data memory address	Contents
D:0x050	01h
D:0x051	02h
D:0x052	03h
D:0x053	04h

1. Write a assemble language program to transfer a bytes of data from one location to another location both externally

```
Org 00h
Mov r3, #04h
Mov r0, #00h ///dpl = source
Mov r1, #00h ///dpl = destination
Back: Mov dph, #12h
Mov dpl, r0 /// dpl = source == 00h
Movx a, @dptr /// 1200h
Inc r0
Mov dph, #15h
Mov dpl, #r1 ////// mov dptr = 1501h
Movx @dptr, a //// 1500h
Inc r1
Djnz r3, back
Here:sjmp here
end
```

Data memory address	Contents
x:0x1200	01h
x:0x1201	02h
x:0x1202	03h
x:0x1203	04h

Data memory address	Contents
x:0x1500	00h
x:0x1501	00h
x:0x1502	00h
x:0x1503	00h

Data memory address	Contents
x:0x1500	01h
x:0x1501	02h
x:0x1502	03h
x:0x1503	04h

3. Write a assemble language program to exchange a block of bytes of data between two memory locations

```
org 00h
Mov r3, #04h
Mov r0, #40h
Mov r1, #50h

Next: Mov a, @r0
Xch a, @r1
Mov @r0, a

Inc r0
Inc r1

Djnz r3, next

Here:sjmp here
end
```

Data memory address	Contents
D:0x040	01h
D:0x041	02h
D:0x042	03h
D:0x043	04h

Data memory address	Contents
D:0x050	10h
D:0x051	20h
D:0x052	30h
D:0x053	40h

Data memory address	Contents
D:0x040	10h
D:0x041	20h
D:0x042	30h
D:0x043	40h

Data memory address	Contents
D:0x050	01h
D:0x051	02h
D:0x052	03h
D:0x053	04h

1. Eight bit numbers X, NUM1 and NUM2 are stored in internal data RAM locations 20h, 21h and 22H respectively. Write an assembly language program to compute the following:

IF X=0; then NUM1 (AND) NUM2,
IF X=1; then NUM1 (OR) NUM2,
IF X=2; then NUM1 (XOR) NUM2,
ELSE RES =00, RES is 23H RAM location.

2. Write a assemble language program to toggle all the bits of Port 2 for every 200ms. Assume crystal is 11.0592MHz. Show all the calculations needed.

3. Write an assembly language program to find the average of 10 students marks stored in external RAM memory address 8000H. Load the average value in internal RAM memory 30H.

4. Write an assembly language program to find the factorial of a number. Use Subroutine programming.

5. Write an ALP to convert a packed BCD number into two ASCII numbers. Store the result in R5 and R6 respectively.

6. Write an ALP to convert a Binary number to packed BCD number (hexadecimal to decimal). The binary number is stored at 40h location. Store the converted packed BCD number at 50h and 51h internal RAM location.

7. Write an assembly language program to sort an array of n= 5 bytes of data in ascending order stored from location 30h. (Use bubble sort algorithm)

8. Write an assembly language program to count the number of 1’s and 0’s in an 8-bit data received from port P1. Store the count of 1’s and 0’s in 30h and 31h.

9. Assume a push button switch is connected to port pin P1.2, Write an assembly language program to monitor the switch and turn ON the LED’s connected to port P2 as long as the switch is pushed.

+ ve and – ve number program

45h, a3h,11h, 07h, f1h

```
Org 00h
Mov dptr, #2000h
Mov r3, #05h
Back: Movx a, @dptr  /// 45h
rlc a
Jnc Positive
inc 30h
Inc dpl
djnz r3, back
here:sjmp here
Positive: inc 20h
Inc dpl
djnz r3, back
here:sjmp here
end
```

Concept: +ve or –ve no. Indicated by 8th bit of any hexadecimal number

Ex: 45h = 0100 0101

8th bit = 0 = + ve ➔ 30h

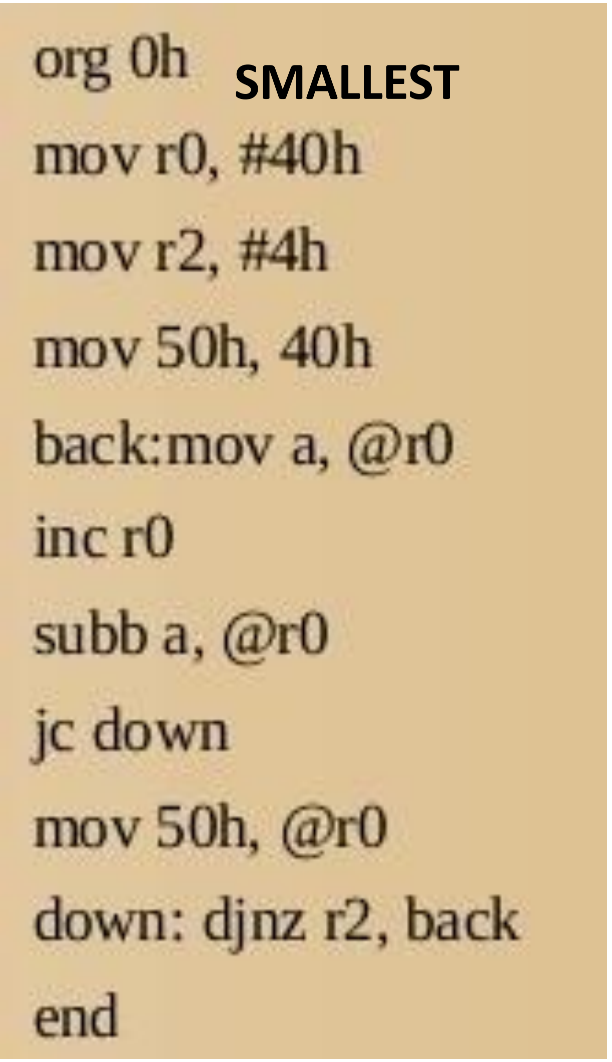
A5h = 1010 0101

8th bit = 1 = - ve ➔ 20h

Find the largest and smallest number in an array of numbers which are stored in memory locations.

F1h, 12h, 33h, F8h,

- ➔ F8h = largest
- ➔ 12h = smallest



SMALLEST

```
org 0h
mov r0, #40h
mov r2, #4h
mov 50h, 40h
back:mov a, @r0
inc r0
subb a, @r0
jc down
mov 50h, @r0
down: djnz r2, back
end
```

LARGEST

```
org 0h
mov r0, #40h
mov r2, #4h
mov 50h, 40h
back:mov a, @r0
inc r0
subb a, @r0
jnc down
mov 50h, @r0
down: djnz r2, back
end
```

Write an ALP for Decimal UP-Counter

```
ORG 00H
MOV R3, #64H

BACK1: MOV A, #00H
MOV P1, #00H

BACK2: MOV P1, A
ACALL VIJAY
ADD A, #01H
DA A
DJNZ R3, BACK2
SJMP BACK1

VIJAY:MOV R0, #255
HERE1:MOV R1, #255
HERE2:MOV R2, #255
HERE3:DJNZ R2, HERE3
DJNZ R1, HERE2
DJNZ R0, HERE1
RET
END
```

EXPECTED RESULTS:

NOTE THAT 64H = 99 in decimal

PORT1: P1: 00 TO 99 displayed

Write an ALP for Decimal DOWN-Counter.

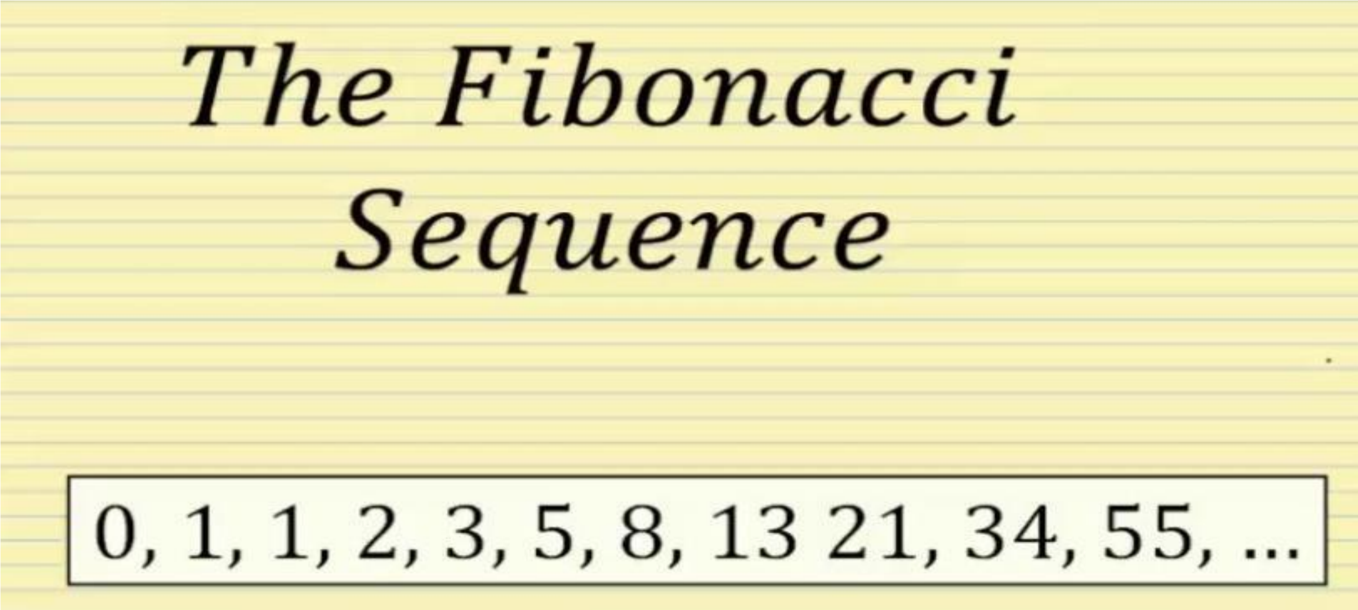
```
ORG 00H
MOV R3, #64H

BACK1: MOV A, #99H
MOV P1, #00H

BACK2: MOV P1, A
ACALL VIJAY
ADD A, #99H
DA A
DJNZ R3, BACK2
SJMP BACK1

VIJAY:MOV R0, #255
HERE1:MOV R1, #255
HERE2:MOV R2, #255
HERE3:DJNZ R2, HERE3
DJNZ R1, HERE2
DJNZ R0, HERE1
RET
END
```

EXPECTED RESULTS:

NOTE THAT 64H = 99 in decimal

PORT1: P1: 99 TO 00 displayed

```
Org 00h
Mov r3, #0ah
Mov r1, #20h
Mov a, #00h
Mov @r1, a  /// 0
add a,#01h
inc r1
Mov @r1, a  ///1
Back: dec r1             //// to 20h
add a, @r1    //// 0 + 1 ,, 20h + 21h = 01
da a
inc r1
inc r1
Mov @r1, a
djnz r3, back
end
```



The Fibonacci Sequence

0, 1, 1, 2, 3, 5, 8, 13 21, 34, 55, ...

1. Eight bit numbers X, NUM1 and NUM2 are stored in internal data RAM locations 20h, 21h and 22H respectively. Write an assembly language program to compute the following:

IF X=0; then NUM1 (AND) NUM2,
IF X=1; then NUM1 (OR) NUM2,
IF X=2; then NUM1 (XOR) NUM2,
ELSE RES =00, RES is 23H RAM location.

Concept of Problem:

20H ➔ X, 21H➔NUM1, 22H➔NUM2, 23H➔RES
20h = 00h, 21h and 22 h,, Results stored in Accumulator ➔ A
20h = 01h, 21h or 22h,, Results stored in Accumulator ➔ A
20h = 02h, 21h XOR 22h, Results stored in Accumulator ➔
20h = other than 00h, 01h, 02h, 23 h = 00h.

```
org 00h
Mov r0, 21h
Mov r1, 22h
mov a, 20h
cjne a,#00h,vijay1
Mov a, 21h
Anl a, 22h
Here: sjmp Here
mov a,20h
vijay1: cjne a,#01h,vijay2
Mov a, 21h
Orl a, 22h
Here1:sjmp Here1
mov a,20h
vijay2:cjne a,#02h,vijay3
Mov a, 21h
Xrl a, 22h
Here2:sjmp Here2
Vijay3: mov 23h, #0ffh
Here3:sjmp here3
end
```

Write an assembly language program to find the average of 10 students marks stored in external RAM memory address 8000H. Load the average value in internal RAM memory 30H.

```
org 00h
mov dptr, #8000h
mov r1, #0ah
mov r2, #00h
loop: movx a,@dptr
add a, r2
mov r2,a        /// Finally r2 contains total marks
inc dpl
djnz r1, loop
mov b, #0ah       /// b = 0ah
mov a,r2          //// a = r2
div ab
mov 30h,a            //// quotient value
mov 31,b           //// remainder value
end
```

Data memory address	Contents
x:0x8000	01h
x:0x8001	02h
x:0x8002	03h
x:0x8003	04h
x:0x8004	05h
x:0x8005	06h
x:0x8006	07h
x:0x8007	08h
x:0x8008	09h
x:0x8009	0Ah

Write an assembly language program to find the factorial of a number. Use Subroutine programming.

Input is 04h ➔

Output is ➔ 04h X 03h X 02h X 01h

We can show the results in terms of decimal Value by using da a instruction

Note this program is only for smaller values

```
org 00h
mov r0,#04h
mov a,r0
acall fact
fact:dec r0
cjne r0,#01,vijay
sjmp stop
vijay:mov b,r0
mul ab
da a
acall fact
stop:sjmp stop
end
```

Write an assembly language program to count the number of 1's and 0's in an 8-bit data received from port P1. Store the count of 1's and 0's in 30h and 31h.

P1 => 45h ==> **0100 0101**

No. of one's (1's) = 03h => stored in 30h

No. of zero's (0's) = 05h => stored in 31h

Cy flag <= 0100 0101

With initial carry = 0 using clr c

RLC => cy data bits
0 1000 1010 JC LOOP inc 31h
RLC => 1 0001 0100 JC LOOP
LOOP:inc 30h

8 TIMES RLC NEEDED == R3 COUNTER 8 TIMES

```
Org 00h
Clr c
Mov a, 90h /// 90h = Port 1
Mov r3, #08h
Back: rlc a
jc loop
Inc 31h
Djnz r3, back
Here:sjmp here
loop:inc 30h
Djnz r3, back
Here:sjmp Here
end
```

```
Org 00h
Clr c
Mov a, 90h /// 90h = Port 1
Mov r3, #08h
Back: rlc a
jnc loop
Inc 30h
Djnz r3, back
Here:sjmp here
loop:inc 31h
Djnz r3, back
Here:sjmp Here
end
```

Ascending and Descending Order

```
Org 00h
Mov r5, #05h
Backexternal: mov r4, #05h
Mov r0, #20h
Mov r1, #21h
Backinternal: mov a, @r0
Mov 50h, a
Mov b, @r1
Subb a, b
Jnc VIJAY /// note that Jnc for Ascending and Jc for Descending
Sjmp last
mov a, 50h
Xch a, @r1
Xch a, @r0
VIJAY: inc r0
Inc r1
djnz r4, backinternal
djnz r5, backexternal
end
```

Concept: 05h, F1h, 23h, FFh, 01h

Ascending: 01h, 05h,23h, F1h, FFh

R0=>R1=>R2

05h, 23h, F1h, 01h,FFh = 1st step

05h, 23h, 01h, F1h, FFh = 2nd step

05h, 01h, 23h, F1h, FFh = 3rd step

01h, 05h, 23h, F1h, FFh = 4th step

Square and Cube of Number

Square of a given number

1500h == 1A h == input

50h, 51h == output

```
org 00h
mov dptr, #1500h
movx a, @dptr
mov b, a  or mov b ,#1Ah
mul ab   //// a = A4h, b=02h
mov 50h, b
mov 51h, a
end
```

```
  1Ah X 1Ah
---------------------
   02  A4 h
  50h  51h
```

Input value :1Ah X 1Ah X 1Ah

```
    1Ah X 1Ah
---------------------
    02 A4 h

  02 A4 h X 1Ah
---------------------
     10 A8 h
  00  34
-----------------------
  00 44  A8 h
```

Cube of a given number

1500h == 1A h == Input

50h, 51h == Output

```
org 00h
Mov dptr, #1500h
Movx a, @dptr
Mov a, b
mul ab /// a = A4h, b = 02h
mov r4, a   //// r4 = A4h
mov r5, b   //// r5 = 02h
mov a, #1ah
mov b, r4
mul ab ////// a= A8h, b = 10h
mov r1, a   /// r1 = A8h
Mov r2, b  /// r2 = 10h
```

```
Mov a, #1Ah
Mov b, #r5
Mul ab // a = 34h, b = 00h
Mov 32h, r1
Add a, r2 /// a = 44h
Mov 31h, a

Mov a, #00h
Mov r1, b
Addc a, r1

Addc a, b
Mov 31h, a
Mov 30h, b
end
```

Input value :1Ah X 1Ah X 1Ah

```
   1Ah X 1Ah
---------------------
    02 A4 h

  02 A4 h X 1Ah
---------------------
    10 A8 h   /// A4 h X 1Ah
 00  34       /// 02 h X 1Ah
------------------------
00 44 A8 h → 30h, 31h, 32h
```

Find Odd or Even of a given number , put the output FFh in Port2 for odd and 00h in port 2 for even

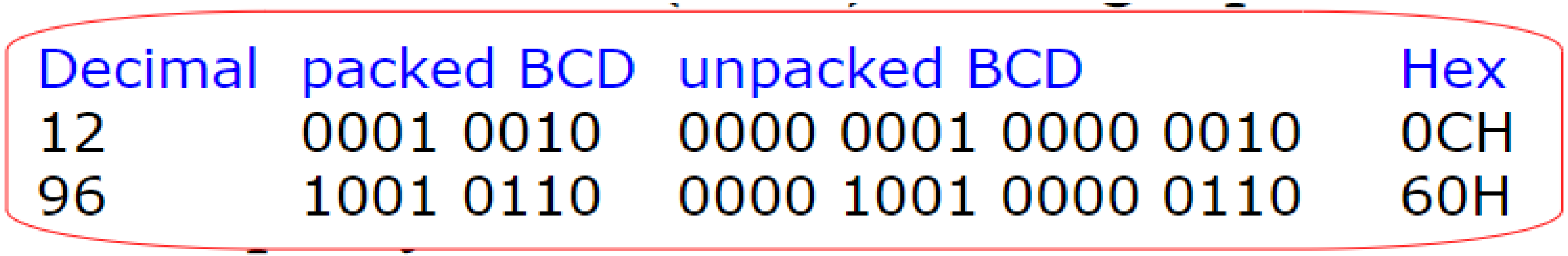
Example: 23h => 0010 0011 = odd
4Ah => 0100 1010 = even

0 = 0000h => even
1 = 0001h => odd
2 = 0010h => even
3 = 0011h => odd
4 = 0100h => even
5 = 0101h => odd
6 = 0110h => even
7 = 0111h => odd
8 = 1000h => even
9 = 1001h => odd
A = 1010h => even
B = 1011h => odd
C = 1100h => even
D = 1101h => odd
E = 1110h => even
F = 1111h => odd

```
org 00h
Mov P2, #00h
clr c
Mov dptr, #1500h
Movx a, @dptr   /// a = 23h = input
rrc a      ///if cy = 1 jump to next loop otherwise next instruction is executed
jc next
mov P2, #00h
here1:sjmp here1
next: mov P2, #0FFh
here: sjmp here
end
```

Dr. Vijaya Kumar H R, Associate. Prof, Dept. of ECE, AIT, Tumkuru

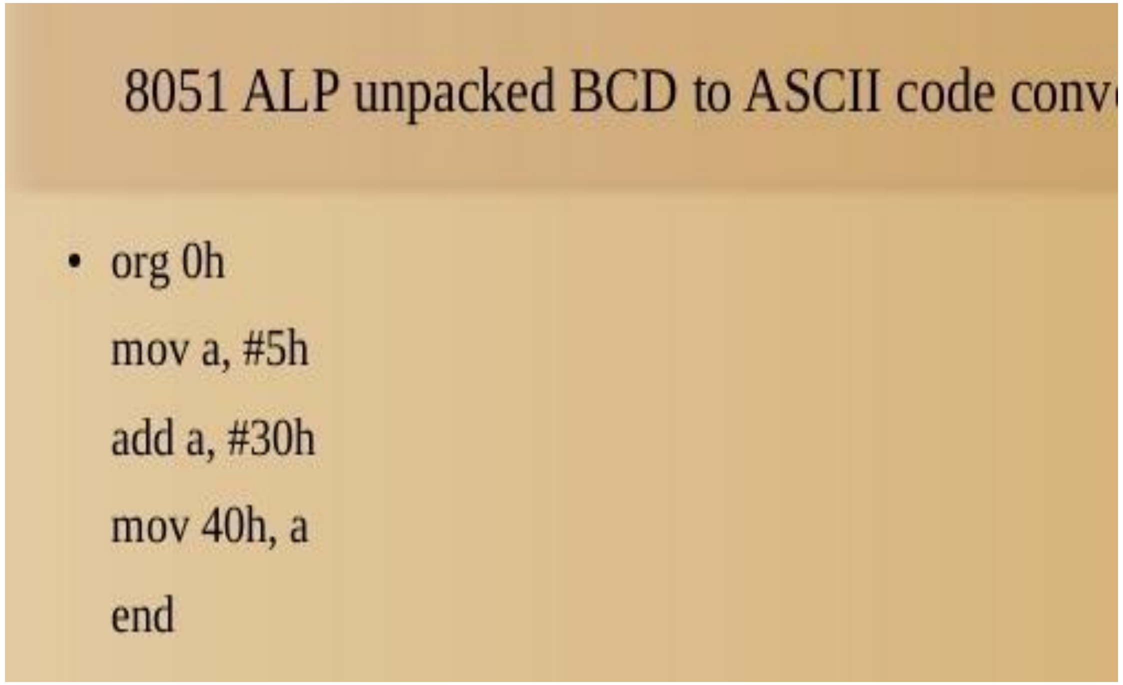
We need to observe first binary bit of any given number to indicate odd or even, 0 = even, 1= odd



Decimal	packed BCD	unpacked BCD	Hex
12	0001 0010	0000 0001 0000 0010	0CH
96	1001 0110	0000 1001 0000 0110	60H

KEY	ASCII(HEX)	BINARY	BCD(UNPACKED)
0	30	011 0000	0000 0000
1	31	011 0001	0000 0001
2	32	011 0010	0000 0010
3	33	011 0011	0000 0011
4	34	011 0100	0000 0100
5	35	011 0101	0000 0101
6	36	011 0110	0000 0110
7	37	011 0111	0000 0111
8	38	011 1000	0000 1000
9	39	011 1001	0000 1001

Input: Unpacked BCD number = 05h
Output: ASCII number = 35



8051 ALP unpacked BCD to ASCII code conv

```
• org 0h
  mov a, #5h
  add a, #30h
  mov 40h, a
  end
```

Input: ASCII number = 35
Output: Unpacked BCD number = 05h

8051 ALP ASCII to unpacked BCD code conve

```
• org 0h
  mov a, #35h
  subb a, #30h
  mov 40h, a
  end
```

Packed BCD to ASCII number program

Input: Packed BCD number = 12h= 0001 0010

Output: ACSCII: 31 32 = 0011 0001 0011 0020

```
org 00h
mov a, #12h
anl a, #0f0h /// a = 10h
swap a  /// a = 01h
add a, #30h /// a = 31h
mov r1, a /// r1 = 31h
mov a, #12h
anl a, #0fh ///a = 02h
add a, #30h /// a = 32h
Mov r2, a
end
```

Store the final answer in r1,r2 register

ASCII number to packed BCD number program

Input: ACSCII: 31 32 = 0011 0001 0011 0020

Output: Packed BCD number = 12h= 0001 0010

Take the input from 2000h = 31h and 2001h =32h

Store the final answer in 20h location

```
org 00h
mov dptr, #2000h
movx a, @dptr /// a = 31h
subb a, #30h //// a = 01h
mov r1, a  ///// r1 = 01h
Inc dpl //// 2000h becomes → 2001h
Movx a,@dptr ///a =32h
subb a, #30 /// a = 02h
mov r2, a /// r2 ==02h   since r1 =01h, r2 = 02h, but answer 12h => 20h
mov a, r1 //// a = 01h
swap a  //// a = 10h
orl a, r2 //// a = 10h or 02h → 0001 0000 0r 0000 0010  = > 0001 0010 →12h
mov 20h, a
end
```

8051 ALP HEX to DECIMAL code conversion

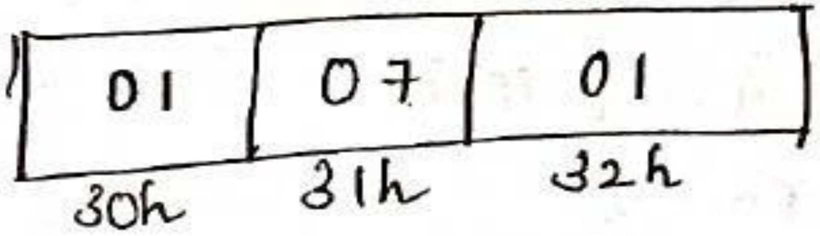
```
Org 00h
MOV A, #ABh
MOV B, #0Ah
div AB
MOV 32h, B
MOV B, #0Ah.
div AB.
MOV 31h, B
MOV 30h, A
End.
```

i/p = AB

o/p = 171

0A)AB(11
AA
01

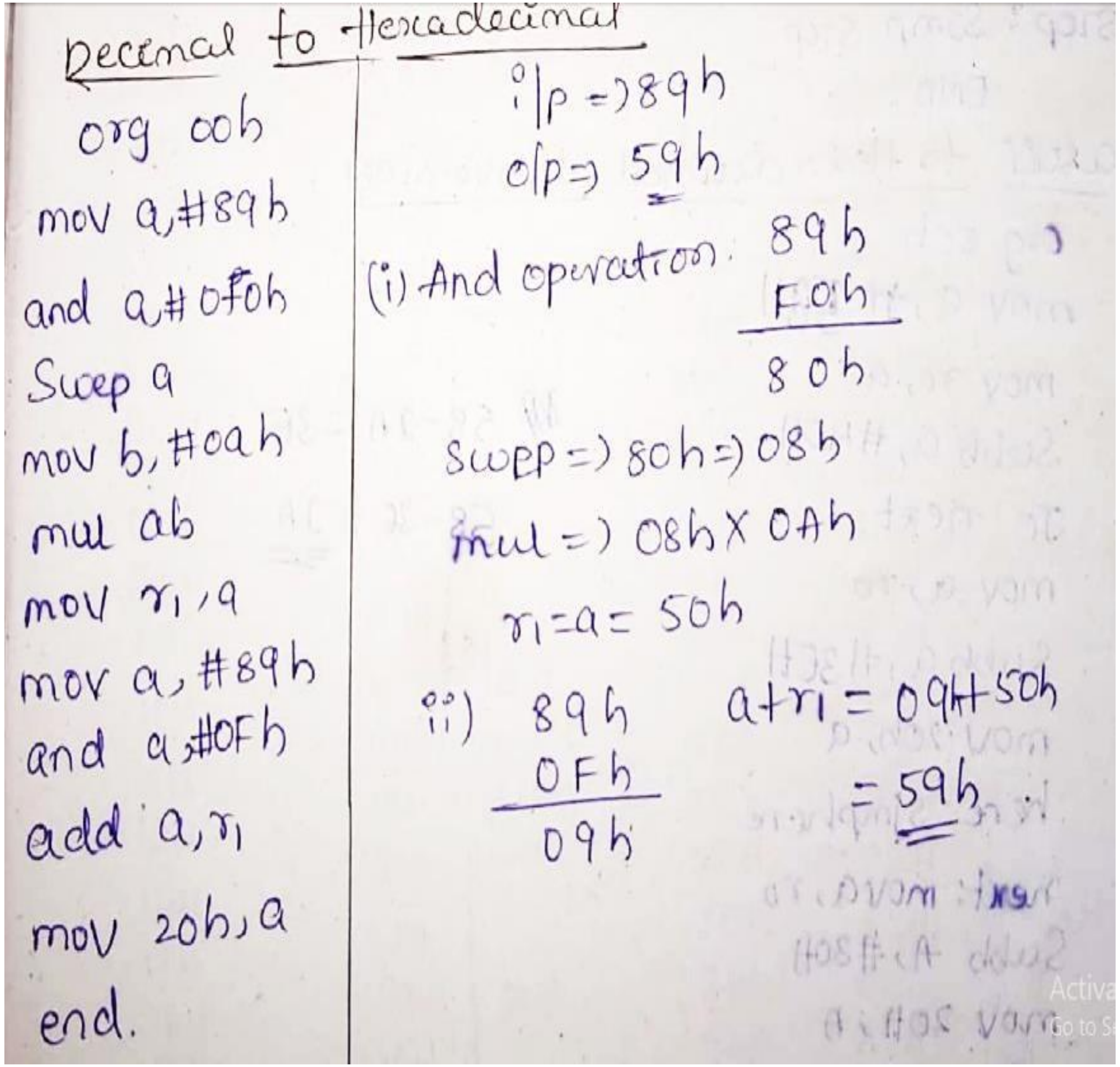
0A)11(1
0A
07



0A × 11 = AA , AB − AA = 01.

0A × 1 = 0A , 11 − 0A = 07.

Decimal to Hexadecimal Number



Decimal to Hexadecimal

```
org 00h
mov a,#89h
and a,#0f0h
Swap a
mov b,#0ah
mul ab
mov r1,a
mov a,#89h
and a,#0Fh
add a,r1
mov 20h,a
end.
```

i/p =) 89h

o/p =) 59h

(i) And operation.

$$\begin{array}{r} 89h \\ F0h \\ \hline 80h \end{array}$$

Swap =) 80h =) 08h

mul =) 08h X 0Ah

r1=a= 50h

(ii)

$$\begin{array}{r} 89h \\ 0Fh \\ \hline 09h \end{array}$$

a+r1 = 09h+50h

= 59h

```
org 00h
mov a,#17
mov b,#16
div ab
mov r1, b
add a, #09h
mov r5, a
mov r2,#01
mov a, r1
swap a
orl a, r2
mov r3, a
anl a, #0f0h
swap a
mov b, #0ah
mul ab
mov r1, a
mov a,r3
anl a, #0fh
add a, r1
mov r6, a
mov a, r5
end
```

16)171(10
160

11
First division A = 10 = 0Ah, B = 11 = 0Bh
Answer => AB h

16)165(10
160

05
First division A = 10 = 0Ah, B = 05h
Answer => A5 h

16)212(13
208

04
First division A = 13 = 0Dh, B = 04 = 04h
Answer => D4 h

Write an alp to convert Hexadecimal number to ASCII number ?? Home work program

- ➔ First we need to convert Hexadecimal number to Decimal number (Packed)
- ➔ Second we need to convert Decimal number to ASCII number

0C h Hexadecimal number

➔ 12h Decimal number

➔ 31h &32h ASCII number

Write an alp to convert ASCII number to Hexadecimal number ?? Home work problem

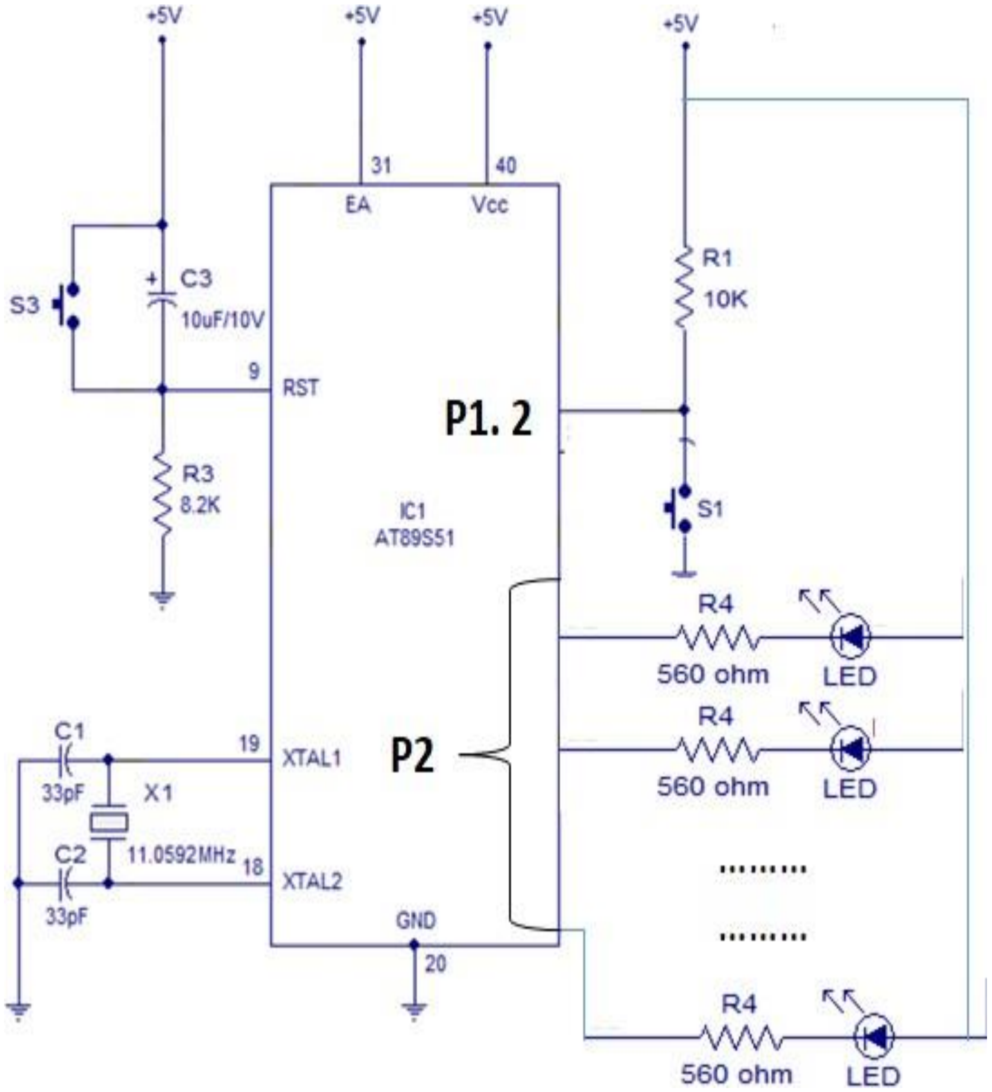
- ➔ First we need to convert ASCII number to Decimal number (Packed)
- ➔ Second we need to convert Decimal number to Hexadecimal number

35h and 37h ASCII number

➔ 57h Decimal number (Packed)

➔ 87h is Hexadecimal final answer

Assume a push button switch is connected to port pin P1.2, Write an assembly language program to monitor the switch and turn ON the LED's connected to port P2 as long as the switch is pushed



Switch Connect to => P1.2
8 LED'S Connected to => P2
If P1.2 switch press or on => 8 LED => on => FF h
If P1.2 switch not press or off => 8 LED => off => 00h
P1.2 on or off => Setb, clr instruction used
SetB = P1.2 => on status , clr p1.2=> off status

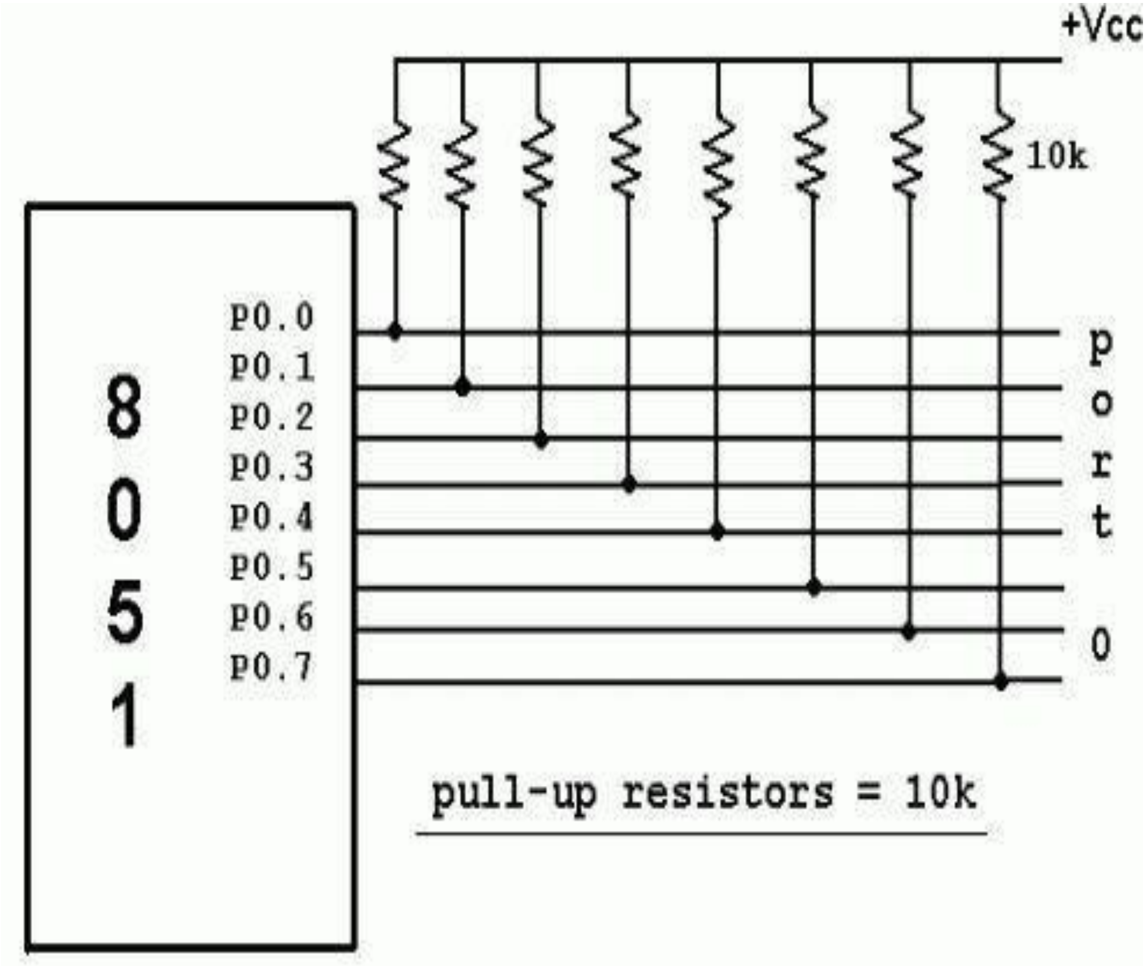
```
org 00h
mov p1, #00h
mov p2, #00h /// 8 LED PINS Connected
clr p1.2 /// p1.2 = Switch is on status
again: jnb p1.2, ledoff
mov p2, #0ffh
sjmp again
ledoff: mov p2, #00h
sjmp again
end
```

Assume a push button switch is connected to port pin P2.3, Write an assembly language program to monitor the switch and to run the motor connected to port P3 as long as the switch is pushed – Home work

Assume a push button switch is connected to port pin P1.0, Write an assembly language program to monitor the switch and to run the RED LED at P2.4 for OFF and Green LED at P2.5 for ON – Home work

Assume a push button switch are connected to port pin P1.0 and P1.1, Write an assembly language program to monitor the switches and to run the RED LED1 at P2.4 for OFF by P1.and Green LED1 at P2.5 for ON and RED LED2 at P2.4 for OFF and Green LED2 at P2.5 for ON – Home work

Why pull-up resistors are connected in 8051 Microcontroller



- The reason for not having pull-ups for port 0 internally is because this port uses for multiplexing of address and data that’s why external pull ups are used and it depends on what purpose we are going to use this port. When we use external pull-ups it becomes general purpose port.

- Again, port 0 is open drain hence it requires pull-ups and all other ports are not having open drain its not left to user. If port 0 has to be used as i/o port its must to connect pull up resistor.

Find the Execution time for the following 12 MHZ for 1. DJNZ R1, BACK, 2. MUL AB, 3. movc a, @a+pc, 4. xchd a, @r1, 5. 5. addc a, r5,   6. div ab

1.DJNZ R1, BACK:

- **The execution time is 2 machine cycles if the jump is taken and 1 machine cycle if not taken.**
- **2 machine cycles = 2 / 12 µs = 0.167 µs**
- **1 machine cycle = 1 / 12 µs = 0.083 µs**

2. MUL AB:

- **The execution time is 4 machine cycles.**
- **4 machine cycles = 4 / 12 µs = 0.333 µs**

3. MOVX A, @A+PC: 2 machine cycles = 2 / 12 µs = 0.167 µs
4. XCHD A, @R1: 1 machine cycle = 1 / 12 µs = 0.083 µs
5. ADDC A, R5: 1 machine cycle = 1 / 12 µs = 0.083 µs
6. DIV AB: 4 machine cycles = 4 / 12 µs = 0.333 µs