TẬP LỆNH 8051/8031

Ý nghĩa các ký hiệu viết tắt trong tập lệnh

Ký hiệu	Ý nghĩa
Rn hay Rr	Định địa chỉ thanh ghi bằng các thanh ghi R0 đến R7
direct	Địa chỉ nội 8 bit (00H–FFH)
@Ri	Định địa chỉ gián tiếp dùng thanh ghi R0 hoặc R1
#data8	Hằng số 8 bit ở ngay trong lệnh. Chú ý với số Hex nếu bắt đầu bằng chữ thì phải thêm số 0 ở phía trước. Thí dụ: – Hằng số thập phân như #10, #-12 – Hằng số nhị phân như #01011010B – Hằng số thập lục phân (số Hex) như #1FH, #0A6H
#data16	Hằng số 16 bit ở ngay trong lệnh. Thí dụ: #0FF25H
bit	Địa chỉ trực tiếp (8 bit) của bit
byte	Một trong các kiểu: direct, @Ri, Rn hoặc #data8
rel	Offset 8 bit có dấu trong định địa chỉ tương đối
addr11	Địa chỉ tuyệt đối 11 bit trong trang 2K hiện hành
addr16	Địa chỉ 16 bit
src	Toán hạng nguồn (source) có thể là Rn, direct hoặc @Ri
dest	Toán hạng đích (destination) có thể là Rn, direct hoặc @Ri
(X)	Nội dung của X. Thí dụ: (A) nội dung thanh ghi A.
((X))	Nội dung ô nhớ có địa chỉ là nội dung của X.
DIR	Cách định địa chỉ trực tiếp (direct)
IN	Cách định địa chỉ gián tiếp (indirect)
REG	Cách định địa chỉ thanh ghi (register)
IMM	Cách định địa chỉ tức thời (Immediate)

Tham khảo tập lệnh

Table 1 – Data Transfer Instructions

Mnemonic	Instruction code								Hexa - decimal	Explanation	No. of MC
	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0			
MOV A, Rn	1	1	1	0	1	n_2	n_1	n_0	E8 ÷ EF	A ← Rn	1
MOV A, direct	1	1	1	0	0	1	0	1	E5	A ← (direct)	1
	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0	Byte 2		
MOV A, @Ri	1	1	1	0	0	1	1	i	E6 ÷ E7	A ← (Ri)	1
MOV A, #data	0	1	1	1	0	1	0	0	74	A ← data	1
	d_7	d_6	d_5	d_4	d_3	d_2	d_1	d_0	Byte 2		

Mnemonic: cú pháp lệnh
Instruction: định dạng mã máy
Hexa-Decimal: mã máy(số hex)
Explanation: Thao tác của lệnh
No. of MC: thời gian thực thi lệnh
Các thanh ghi SFR được hiểu như là các direct

I. CÁC PHƯƠNG PHÁP ĐỊNH ĐỊA CHỈ

- *Tức thời (Immediate)*
- *Định địa chỉ thanh ghi (Register)*
- *Trực tiếp (Direct)*
- *Gián tiếp (Indirect)*
- *Tương đối (Relative)*
- *Tuyệt đối (Absolute)*
- *Dài (Long)*
- *Chỉ số (Index)*

ĐỊNH ĐỊA CHỈ TỨC THỜI

- Toán hạng nguồn là một **hằng số**
- Sử dụng dấu **#**

MOV A,#12 — A: 0 0 0 0 1 1 0 0

MOV A,#0C4H — A: 1 1 0 0 0 1 0 0

MOV R1,#1110B — R1: 0 0 0 0 1 1 1 0

ADD A,#11110001B — CY: 1; A: 1 0 1 1 0 1 0 1

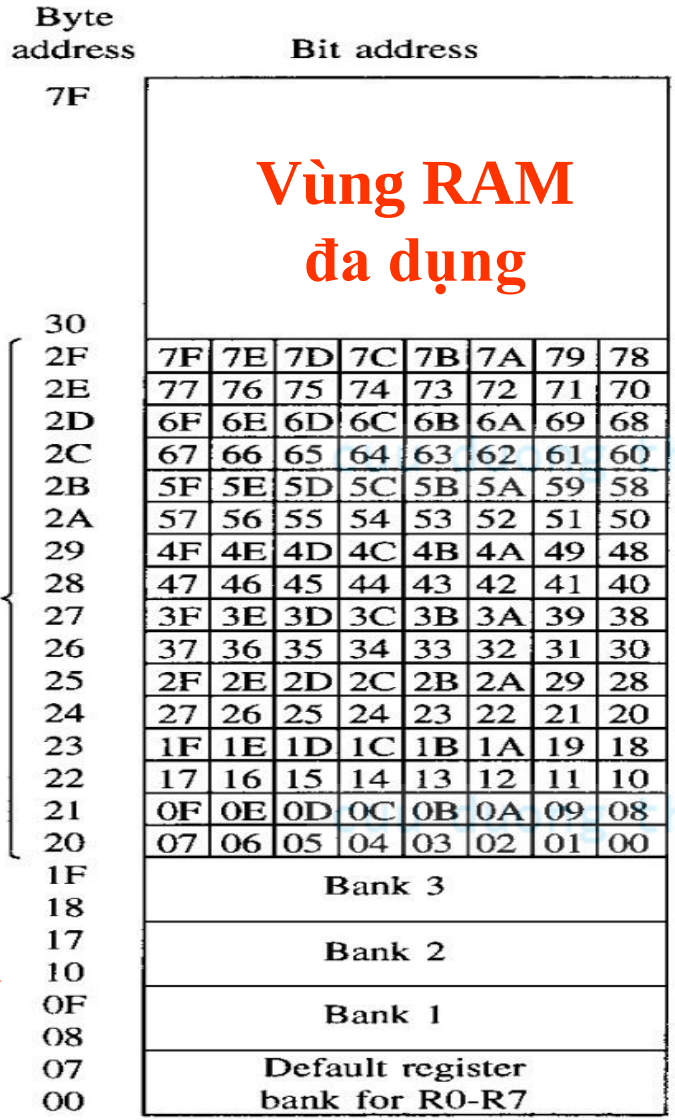
MOV DPTR,#2000H — DPH: 0 0 1 0 0 0 0 0; DPL: 0 0 0 0 0 0 0 0

ĐỊNH ĐỊA CHỈ THANH GHI

MOV	**R1,#36H**	**R1**	0 0 1 1 0 1 1 0
MOV	**A,R1**	**A**	0 0 1 1 0 1 1 0
MOV	**R7,#0FH**	**R7**	0 0 0 0 1 1 1 1
ANL	**A,R7**	**A**	0 0 0 0 0 1 1 0
INC	**A**	**A**	0 0 0 0 0 1 1 1
DEC	**A**	**A**	0 0 0 0 0 1 1 0

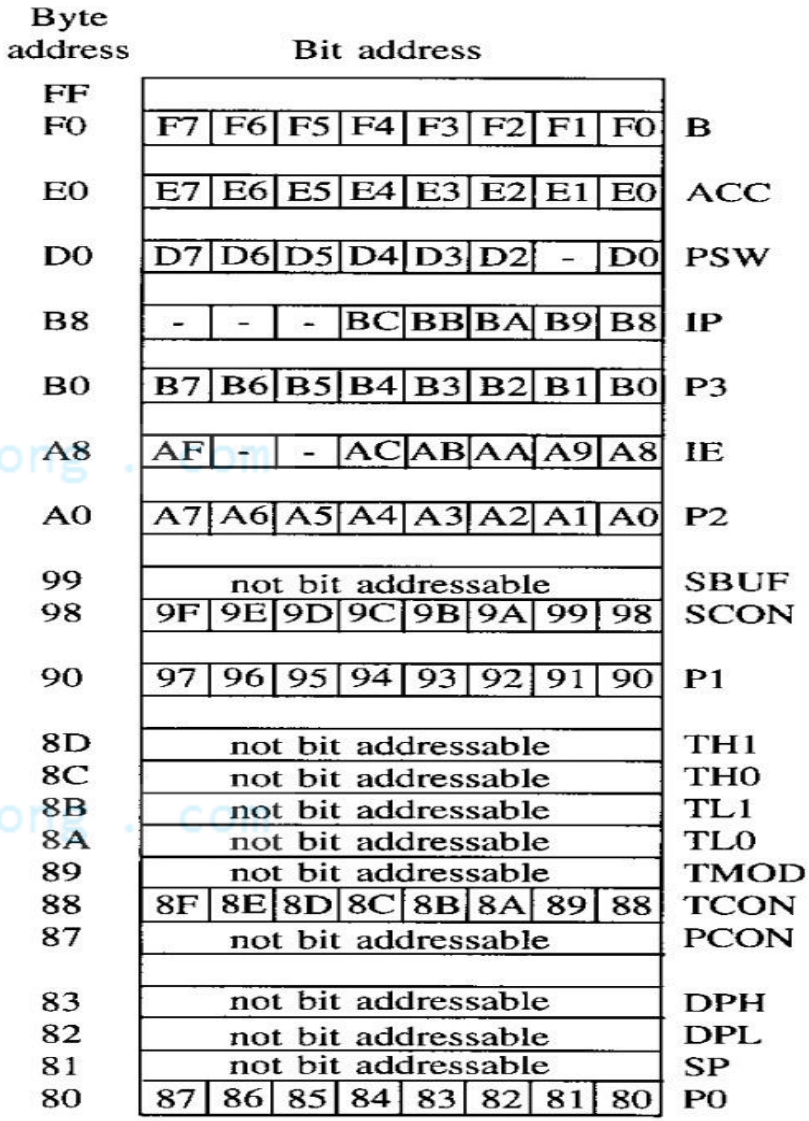
ĐỊNH ĐỊA CHỈ TRỰC TIẾP

Truy xuất đến RAM nội và các thanh ghi chức năng đặc biệt (A(E0H),PSW(D0H),P1(90H), P0(80H)…)



Vùng RAM định vị bit

Vùng bank thanh ghi



Các thanh ghi chức năng đặc biệt

ĐỊNH ĐỊA CHỈ TRỰC TIẾP

Ví dụ:

MOV A,70H

MOV R0,40H

MOV 56H,A

MOV 0D0H,A

MOV PSW,A

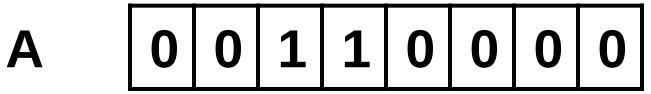
MOV P1,P3

SỰ KHÁC NHAU GIỮA PHƯƠNG PHÁP ĐỊNH ĐỊA CHỈ TRỰC TIẾP VÀ TỨC THỜI

MOV A,30H

Trực tiếp

MOV A,#30H



Tức thời

MOV A,04H ≡ MOV A,4 ≡ MOV A,R4

MOV A,07H ≡ MOV A,7 ≡ MOV A,R7

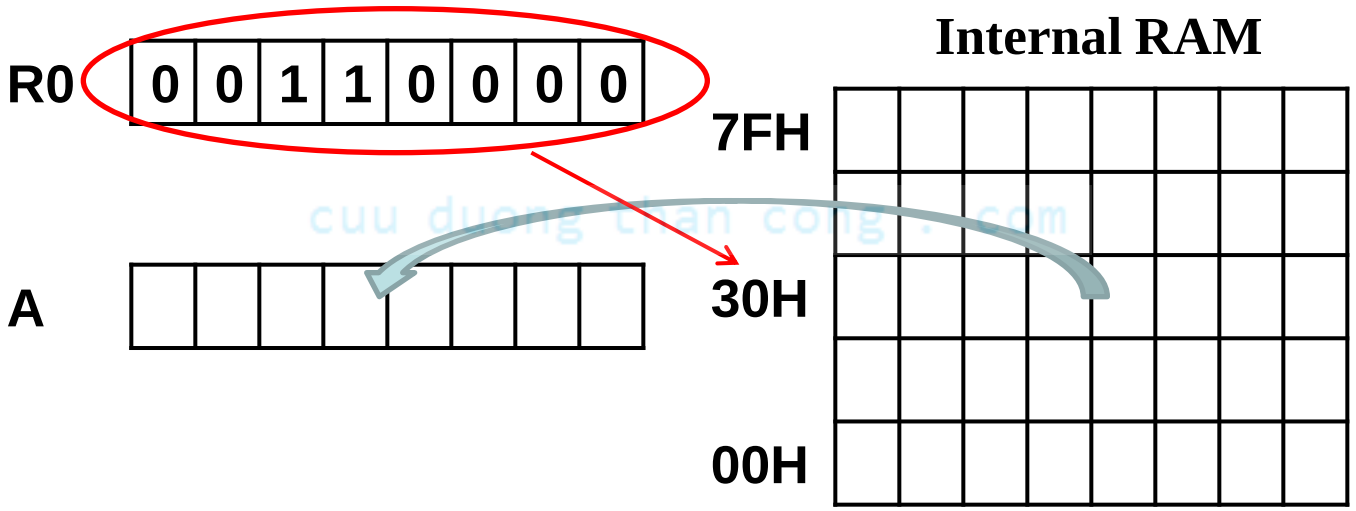
MOV 07H,06H ≡ MOV 7,6 ≠ ~~MOV R7,R6~~

ĐỊNH ĐỊA CHỈ GIÁN TIẾP

- Địa chỉ của nguồn hoặc đích tới được chỉ định trong các thanh ghi **Ri**.
- Chỉ dùng các thanh ghi **R0** or **R1** cho các **địa chỉ 8bit** (**RAM trong hoặc ngoài**), hoặc thanh ghi **DPTR** cho các **địa chỉ 16bit**(**RAM ngoài**).
- Sử dụng dấu **@** để truy xuất đến nội dung của các bộ nhớ: **@R0**, **@R1**

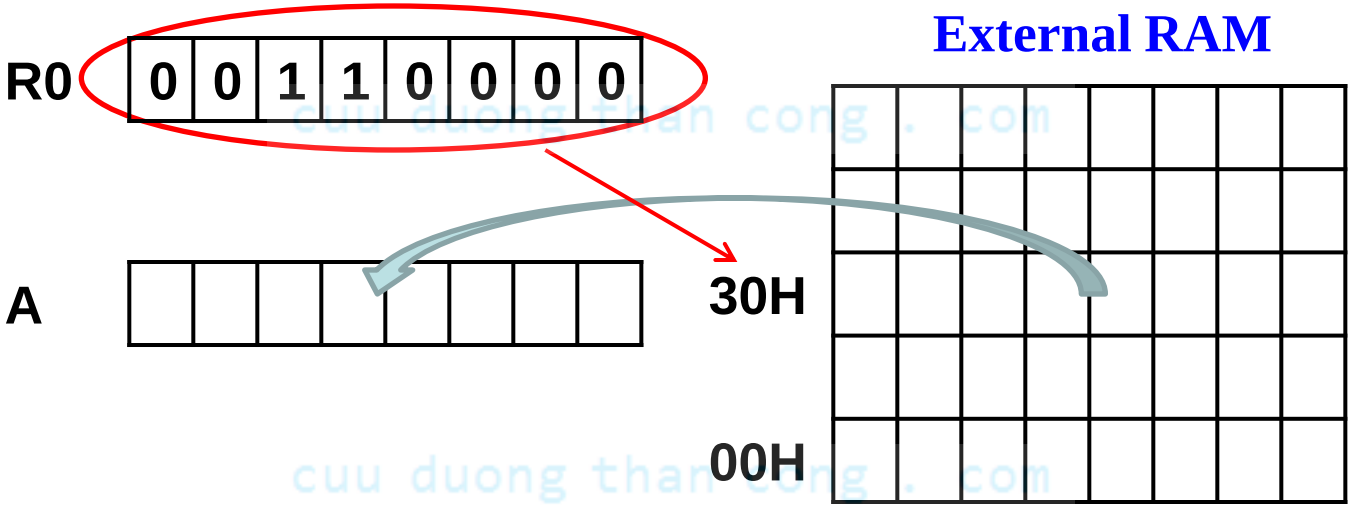
VD: Để truy xuất đến nội dung của ô nhớ 30H nằm ở **RAM trong**, ta thực hiện:

```
MOV   R0,#30H        ;R0 ← 30H
MOV   A,@R0          ;A←(R0)
```



VD: Để truy xuất đến nội dung của ô nhớ 30H nằm ở **RAM ngoài (8bit)**, ta thực hiện:

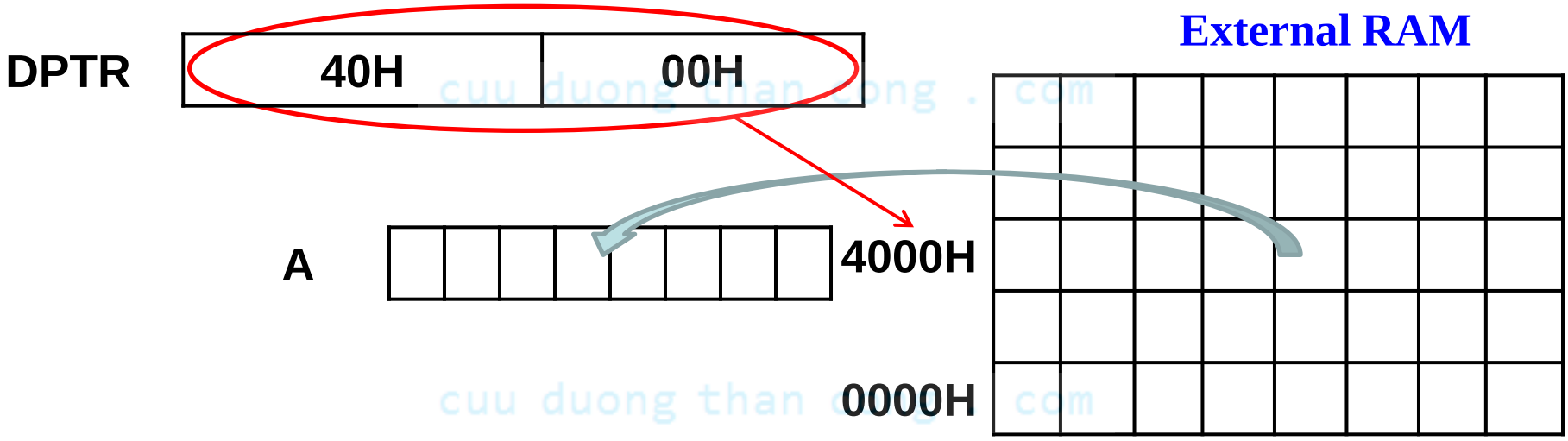
MOV **R0,#30H** ;R0 ← 30H
MOVX **A,@R0** ;A←(R0)



VD: Để truy xuất ***(đọc hoặc ghi dữ liệu)*** đến nội dung của ô nhớ 4000H nằm ở **RAM ngoài (16bit)**

- Đọc dữ liệu từ ô nhớ 4000H lưu vào thanh ghi A

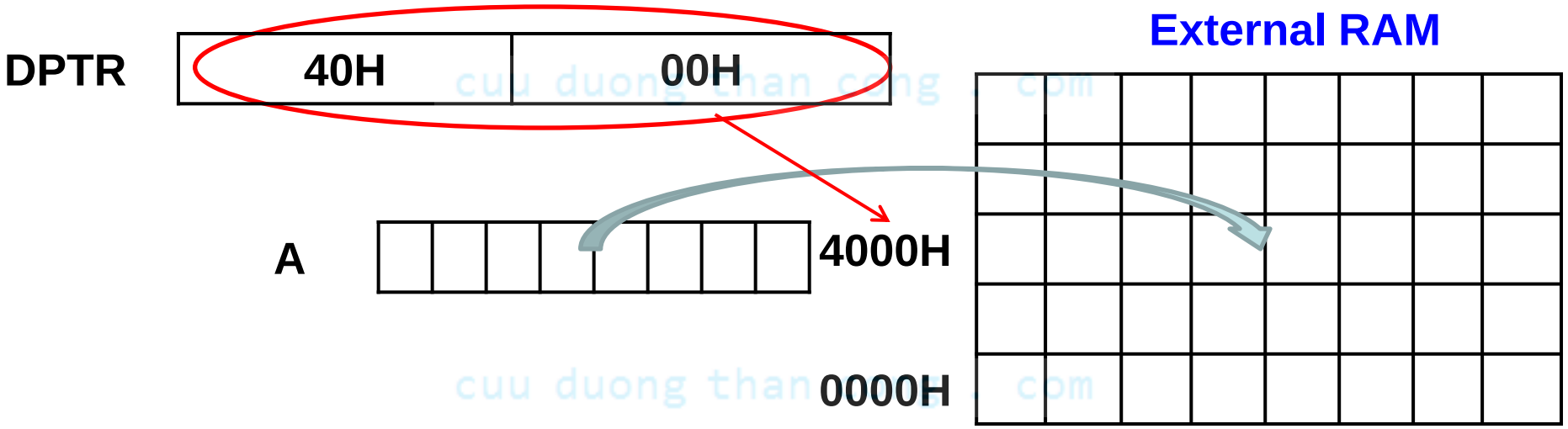
```
MOV     DPTR,#4000H        ;DPTR ← 4000H
MOVX    A,@DPTR            ;A←(4000H)
```



- Ghi dữ liệu vào ô nhớ 4000H (dữ liệu được chứa trong thanh ghi A)

MOV **DPTR,#4000H** ;DPTR ← 4000H

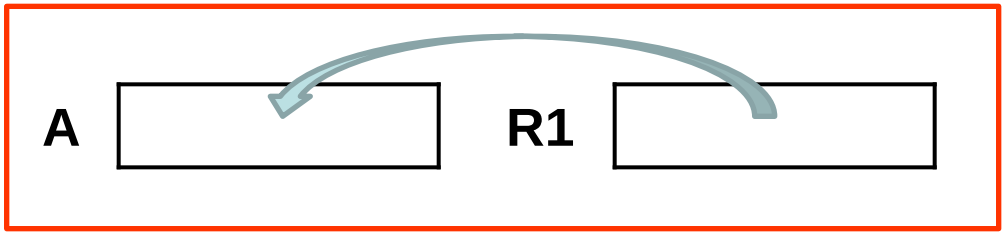
MOVX **@DPTR, A** ;4000H←A



SỰ KHÁC NHAU GIỮA PHƯƠNG PHÁP ĐỊNH ĐỊA CHỈ THANH GHI VÀ GIÁN TIẾP

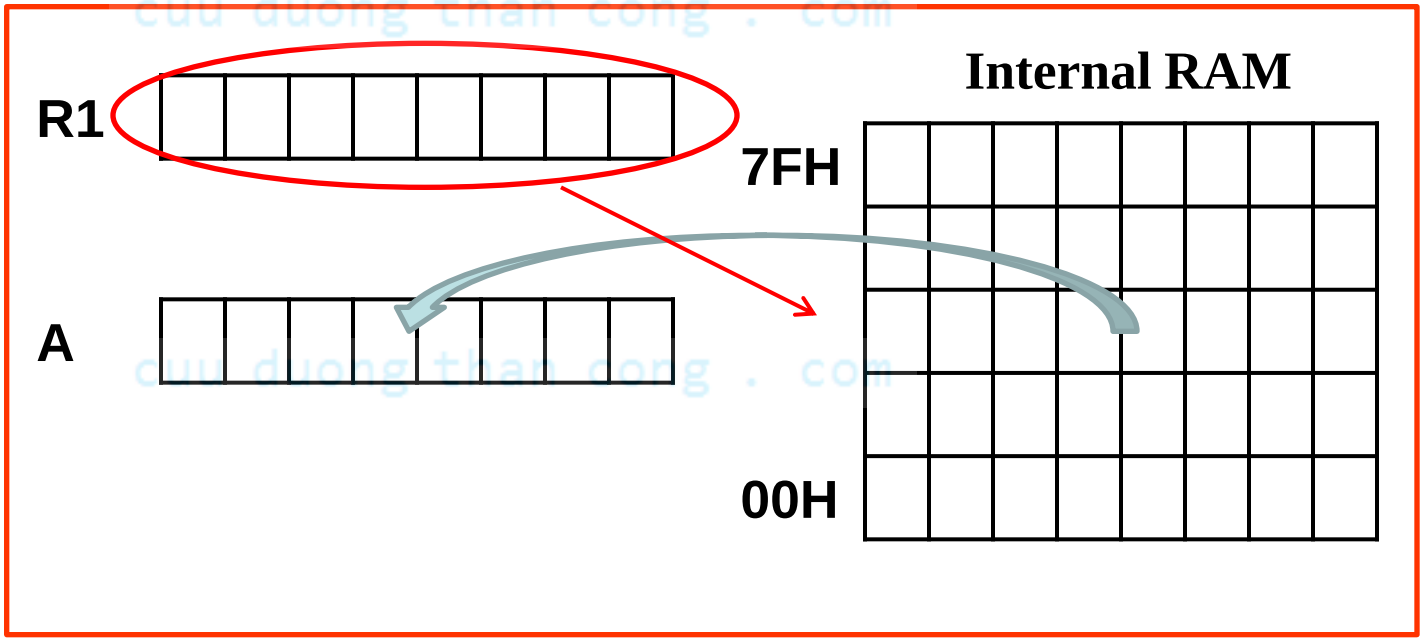
MOV A,R1

Thanh ghi



MOV A,@R1

Gián tiếp



ĐỊNH ĐỊA CHỈ TƯƠNG ĐỐI

- Được sử dụng trong các lệnh nhảy **SJMP**
- **Địa chỉ tương đối** (hay còn gọi là **offset: độ dời**) là 1 số **8-bit có dấu**. Giá trị này được cộng với PC để tạo ra địa chỉ của lệnh tiếp theo cần được thực thi.
- Phạm vi thực hiện : **-128 ~ +127**

VD: lệnh **SJMP LABEL1** được lưu trong bộ nhớ tại vị trí **0100H** và **0101H**.

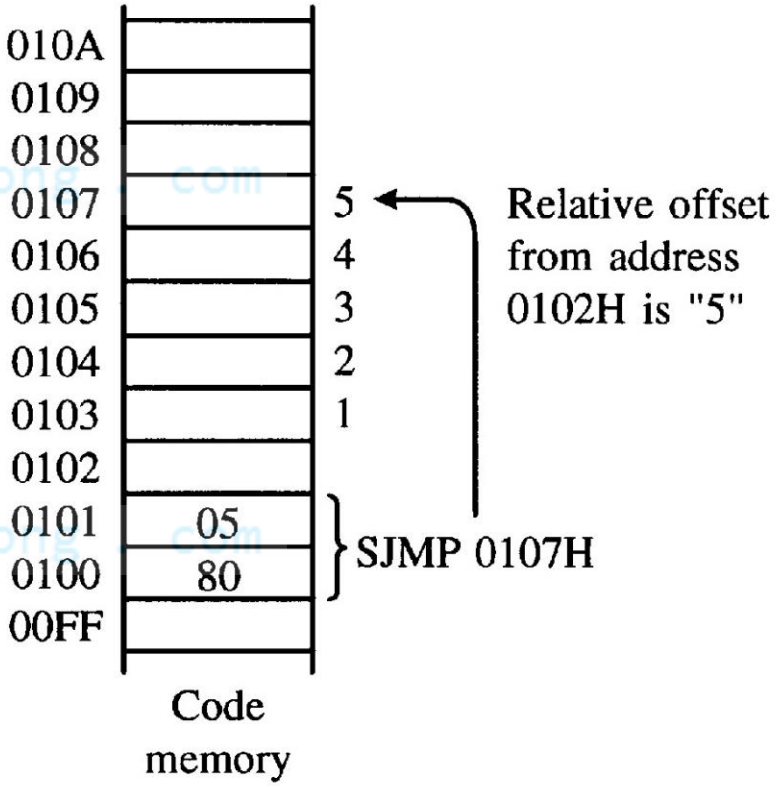
→ **PC** của lệnh kế: **0102H**

Nếu **LABEL1** như hình bên có địa chỉ là **0107H**.

→ **Offset** của lệnh này:

0107H – 0102H = 5

→ **Mã máy** của lệnh **SJMP LABEL1** là: **8005**



VD: lệnh **SJMP LABEL2** được lưu trong bộ nhớ tại vị trí 204**0H** và **2041H**.

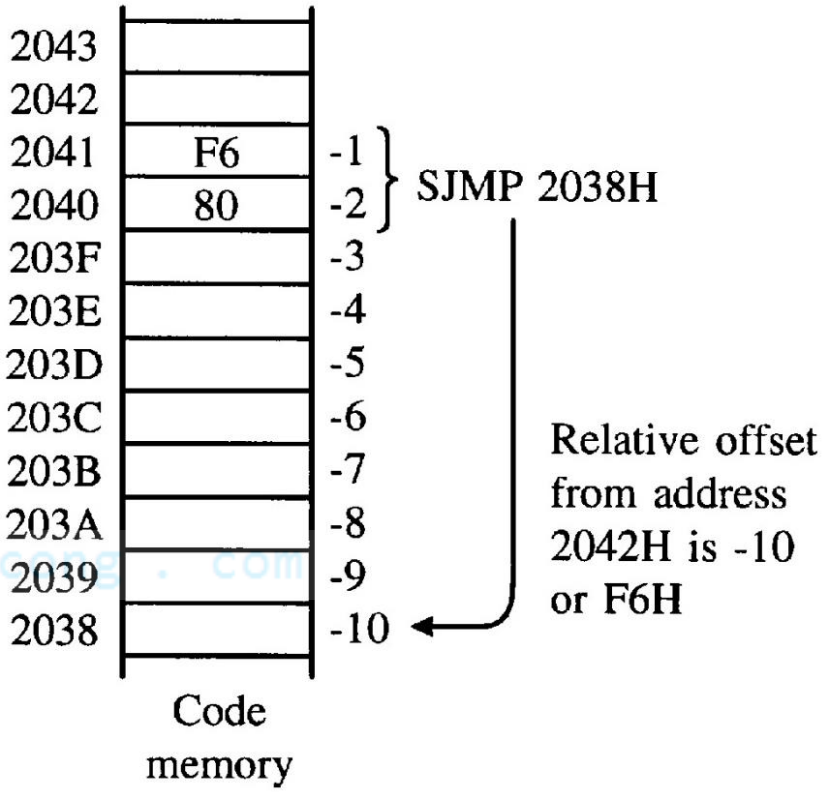
→ **PC** của lệnh kế: **2042H**

Nếu **LABEL2** như hình bên có địa chỉ là **2038H**.

→ **Offset** của lệnh này:

2038H – 2042H = F6 = -10

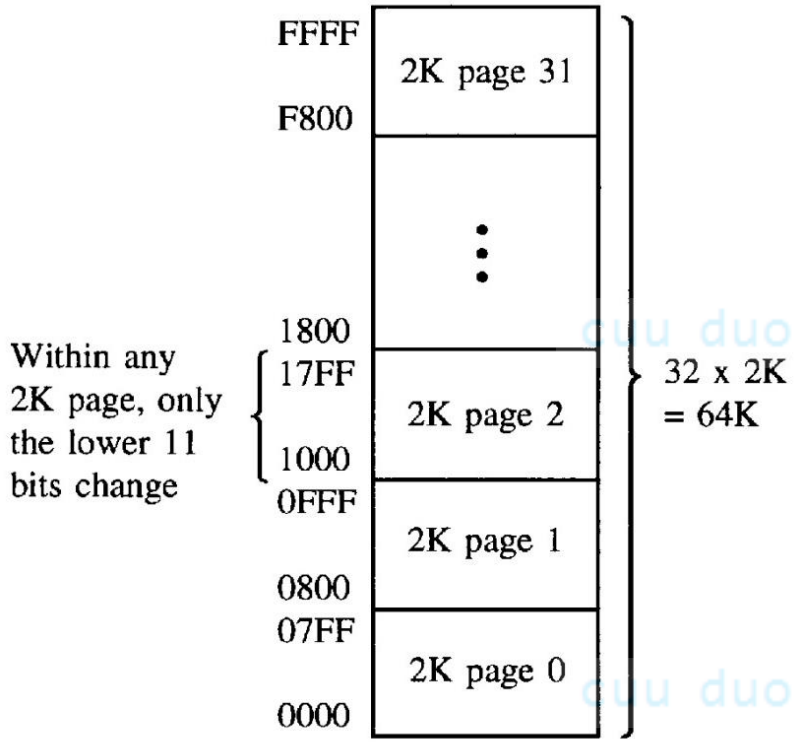
→ **Mã máy** của lệnh **SJMP LABEL2** là: **80F6**



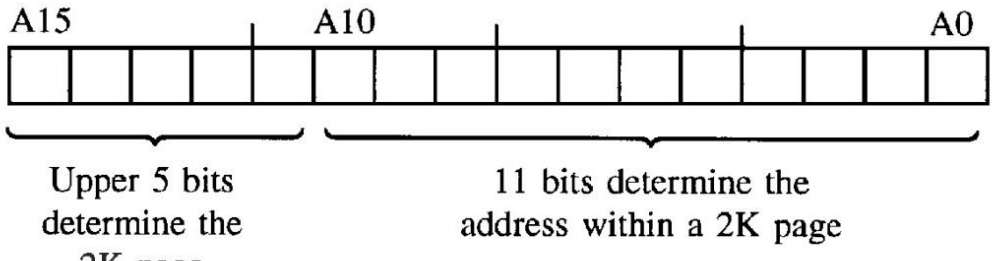
ĐỊNH ĐỊA CHỈ TUYỆT ĐỐI

-Được sử dụng trong **AJMP, ACALL**

-Phạm vi: mỗi trang 2K.



(a)



(b)

ĐỊNH ĐỊA CHỈ DÀI

-Được sử dụng trong **LJMP, LCALL**

-Phạm vi: 64K.

ĐỊNH ĐỊA CHỈ CHỈ SỐ

-Sử dụng một thanh ghi nền (PC, DPTR) và một offset(thanh ghi A) tạo thành địa chỉ hiệu dụng cho lệnh **MOVC** hoặc **JMP.**

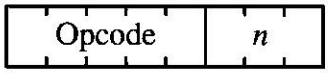
VD:

```
MOVC    A, @A+DPTR
MOVC    A, @A+PC
JMP     @A+DPTR
```

Ví dụ: Trong đoạn chương trình sau: nếu thanh ghi A có các giá trị chẵn từ 0÷6, chuỗi các lệnh sau sẽ thực hiện rẽ nhánh đến một trong 4 lệnh AJMP trong bảng nhảy có địa chỉ tại **JMP_TBL**.

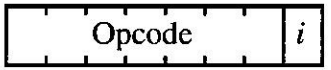
```
        MOV DPTR, # JMP_TBL
        JMP @A + DPTR
JMP_TBL: AJMP LABEL0
        AJMP LABEL1
        AJMP LABEL2
        AJMP LABEL3
```

ĐỊNH DẠNG MÃ LỆNH



(a) Register addressing (e.g., ADD A,R5)

(b) Direct addressing (e.g., ADD A,55H)

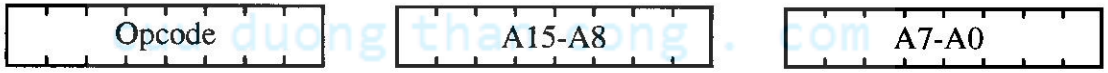


(c) Indirect addressing (e.g., ADD A,@R0)

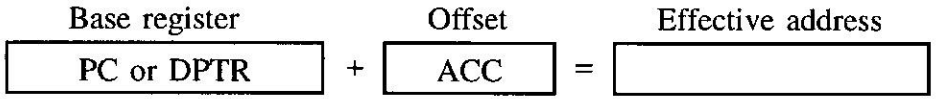
(d) Immediate addressing (e.g., ADD A,#44H)

(e) Relative addressing (e.g., SJMP AHEAD)

(f) Absolute addressing (e.g., AJMP BACK)



(g) Long addressing (e.g., LJMP FAR_AHEAD)



(h) Indexed addressing (e.g., MOVC A,@A+PC)

II. GIỚI THIỆU TẬP LỆNH

*** Gồm 255 lệnh:**

- 139 lệnh 1 byte
- 92 lệnh 2 byte
- 24 lệnh 3 byte

*** Gồm 5 nhóm lệnh:**

- Di chuyển dữ liệu
- Số học
- Logic
- Rẽ nhánh
- Xử lý bit

NHÓM LỆNH DI CHUYỂN DỮ LIỆU

MOV <DEST-BYTE>, <SOURCE-BYTE>

- **Lệnh này sẽ thực hiện di chuyển nội dung của toán hạng nguồn <SOURCE-BYTE> đến toán hạng đích <DEST-BYTE>**
- **Nội dung của byte được chỉ ra bởi toán hạng thứ hai được sao chép vào vị trí được xác định bởi toán hạng thứ nhất. Byte nguồn không bị ảnh hưởng. Các thanh ghi khác và các cờ không bị ảnh hưởng.**

Ví dụ:

Nội dung các ô nhớ trong RAM nội và port1 như sau:

(30H)=40H, (40H)=10H, P1=11001010B (0CAH)

Các lệnh:

```
MOV   R0,#30H
MOV   A,@R0
MOV   R1,A
MOV   B,@R1
MOV   @R1,P1
MOV   P2,P1
```

Làm nội dung các thanh ghi và ô nhớ có giá trị sau:

R0= 30H, B=10H, (40H)=P2=0CAH

Mnemonic	Instruction code								Hexa - decimal	Explanation	No. of MC
	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0			
MOV A, Rn	1	1	1	0	1	n_2	n_1	n_0	E8 ÷ EF	A ← Rn	1
MOV A, direct	1	1	1	0	0	1	0	1	E5	A ← (direct)	1
	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0	Byte 2		
MOV A, @Ri	1	1	1	0	0	1	1	i	E6 ÷ E7	A ← (Ri)	1
MOV A, #data	0	1	1	1	0	1	0	0	74	A ← data	1
	d_7	d_6	d_5	d_4	d_3	d_2	d_1	d_0	Byte 2		
MOV Rn, A	1	1	1	1	1	n_2	n_1	n_0	F8 ÷ FF	Rn ← A	1
MOV Rn, direct	1	0	1	0	1	n_2	n_1	n_0	A8 ÷ AF	Rn ← (direct)	2
	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0	Byte 2		
MOV Rn, #data	0	1	1	1	1	n_2	n_1	n_0	78 ÷ 7F	Rn ← data	1
	d_7	d_6	d_5	d_4	d_3	d_2	d_1	d_0	Byte 2		
MOV direct, A	1	1	1	1	0	1	0	1	F5	(direct) ← A	1
	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0	Byte 2		
MOV direct, Rn	1	0	0	0	1	n_2	n_1	n_0	88 ÷ 8F	(direct) ← Rn	2
	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0	Byte 2		
MOV direct1, direct2	1	0	0	0	0	1	0	1	85	(direct1) ← (direct2)	2
	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0	Byte 2	Source	
	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0	Byte 3	Destination	
MOV direct, @Ri	1	0	0	0	0	1	1	i	86 ÷ 87	(direct) ← (Ri)	2
	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0	Byte 2		
MOV direct, #data	0	1	1	1	0	1	0	1	75	(direct) ← data	2
	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0	Byte 2		
	d_7	d_6	d_5	d_4	d_3	d_2	d_1	d_0	Byte 3		
MOV @Ri, A	1	1	1	1	0	1	1	i	F6 ÷ F7	(Ri) ← A	1
MOV @Ri, direct	1	0	1	0	1	n_2	n_1	n_0	A6 ÷ A7	(Ri) ← (direct)	2
	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0	Byte 2		
MOV @Ri, #data	0	1	1	1	0	1	1	i	76 ÷ 77	(Ri) ← data	1
	d_7	d_6	d_5	d_4	d_3	d_2	d_1	d_0	Byte 2		

MOV DPTR, #data16

- **Lệnh này nạp hằng số 16bit cho con trỏ dữ liệu DPTR.**
- **Con trỏ dữ liệu được nạp bởi hằng số 16 bit chỉ ra trong lệnh. Hằng số 16bit được đặt ở byte2 và byte3 của lệnh. Byte2 là byte cao được nạp cho DPH còn byte3 là byte thấp được nạp cho DPL.**
- **Các cờ không bị ảnh hưởng.**

Ví dụ:

Lệnh :

MOV DPTR, #1234H

Nạp giá trị 1234 cho con trỏ dữ liệu. DPH chứa 12H còn DPL chứa 34H.

Mnemonic	Instruction code								Hexa - decimal	Explanation	No. of MC
	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0			
MOV DPTR, #data16	1	0	0	1	1	0	0	0	98	DPTR ← data16	2
	d_{15}	d_{14}	d_{13}	d_{12}	d_{11}	d_{10}	d_9	d_8	Byte 2		
	d_7	d_6	d_5	d_4	d_3	d_2	d_1	d_0	Byte 3		

MOVC A, @A+ <base-reg>

- **Lệnh này đọc dữ liệu của các bảng tra cứu được chứa trong bộ nhớ chương trình.**
- **Thanh ghi A sẽ chứa byte dữ liệu đọc từ bộ nhớ chương trình. Địa chỉ của byte được tìm nạp là tổng của giá trị 8bit không dấu ban đầu chứa trong thanh ghi A với nội dung của thanh ghi nền 16bit(DPTR hoặc PC).**
- **Lệnh này có thể truy cập đến một bảng có tối đa 256 phần tử có chỉ số từ 0÷255.**

Mnemonic	Instruction code								Hexa - decimal	Explanation	No. of MC
	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0			
MOVC A, @A + DPTR	1	0	0	1	0	0	1	1	93	A ← (A + DPTR)	2
MOVC A, @A + PC	1	0	0	0	0	0	1	1	83	A ← (A + PC)	2

Ví dụ:

Trong bộ nhớ chương trình chứa 1 bảng gồm 4byte dữ liệu liên tục sau: 66H,77H,88H,99H.

- Để truy xuất đến các byte dữ liệu trong bảng này với thanh ghi DPTR, ta thực hiện đoạn chương trình như sau:

```
        MOV   A,#n              ; n=0÷3
        MOV   DPTR,#Table
        MOVC  A, @A+DPTR
Table:  DB    66H,77H,88H,99H
```

Table có thể đặt ở bất cứ vị trí nào trong chương trình.

- **Để sử dụng thanh ghi PC truy xuất dữ liệu ta thực hiện thông qua chương trình con và bảng phải đặt ngay sau chương trình con.**

```
        MOV    A,#n              ; n=0÷3
        CALL   Read_Table
        ……
Read_Table:
        INC    A
        MOVC   A,@A+PC
        RET
Table:  DB     66H,77H,88H,99H
```

- **Ở đây cần thêm lệnh INC bởi vì PC chỉ đến lệnh RET khi lệnh MOVC thực thi (như vậy địa chỉ tại Table là địa chỉ tại RET cộng thêm 1.**
- **Việc tăng A thêm 1 như vậy làm cho địa chỉ thật sẽ chỉ đến địa chỉ có được bằng địa chỉ đầu bảng cộng với chỉ số trong thanh ghi A.**

MOVX <DEST-BYTE>, <SOURCE-BYTE>

- **Lệnh này di chuyển dữ liệu giữa thanh ghi A với nội dung của một byte của bộ nhớ dữ liệu ngoài.**
- **Các truy xuất đến bộ nhớ có địa chỉ 8bit sẽ được thực hiện gián tiếp qua Ri (i=0,1), và địa chỉ 16bit sẽ thực hiện qua thanh ghi DPTR.**

Mnemonic	Instruction code								Hexa - decimal	Explanation	No. of MC
	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0			
MOVX A, @Ri	1	1	1	0	0	0	1	i	E2 ÷ E3	A ← (Ri)	2
MOVX A, @DPTR	1	1	1	0	0	0	0	0	E0	A ← (DPTR)	2
MOVX @Ri, A	1	1	1	1	0	0	1	i	F2 ÷ F3	(Ri) ← A	2
MOVX @DPTR, A	1	1	1	1	0	0	0	0	F0	(DPTR) ← A	2

Ví dụ:

Cần chuyển nội dung của ô nhớ 30H vào thanh ghi A, các lệnh cần thực hiện:

```
MOV   R1, #30h
MOVX  A, @R1
```

Các truy xuất tới RAM ngoài hay ngoại vi có địa chỉ 16 bit

- **MOV** ***DPTR, #data16***
- **MOVX** ***A, @DPTR***

Cần chuyển nội dung của ô nhớ 2000H vào thanh ghi A, các lệnh cần thực hiện:

```
MOV   DPTR, #2000H
MOVX  A, @DPTR
```

Ngược lại muốn đưa data ra RAM ngoài hay ngoại vi, ta dùng lệnh

- **MOVX** ***@Ri, A***
- **MOVX** ***@DPTR, A***

PUSH DIRECT

- **Lệnh này sẽ cất nội dung của toán hạng được chỉ ra trong lệnh vào vùng stack.**
- **Các cờ không bị ảnh hưởng.**
- **Lệnh này chỉ thực hiện được đối với các toán hạng là địa chỉ trực tiếp.**

Ví dụ:

- **PUSH 50H**
- **PUSH DPL**
- **PUSH DPH**
- **PUSH ACC (Không dùng PUSH A)**
- **PUSH 01H (không dùng PUSH R1)**

POP DIRECT

- **Lệnh này sẽ lấy nội dung của ô nhớ được định địa chỉ bởi con trỏ SP và được chuyển đến ô nhớ được định địa chỉ trực tiếp chỉ ra trong lệnh.**
- **Các cờ không bị ảnh hưởng.**
- **Lệnh này cũng chỉ thực hiện đối với các toán hạng là địa chỉ trực tiếp.**

XCH A, <BYTE>

- Lệnh này sẽ thực hiện nạp cho thanh ghi A nội dung của byte được chỉ ra trong lệnh, đồng thời ghi nội dung ban đầu của thanh ghi A cho byte vừa nêu trên.
- Toán hạng nguồn đồng thời là toán hạng đích và ngược lại.
- Lệnh này có thể sử dụng các kiểu định địa chỉ thanh ghi, trực tiếp hoặc thanh ghi gián tiếp.

XCHD A, @Ri

- Lệnh này sẽ thực hiện tráo đổi nội dung nửa byte thấp của thanh ghi A(biểu diễn một digit số hex hoặc BCD) với nội dung nửa byte thấp của một byte trong RAM nội, byte này được định địa chỉ gián tiếp bởi thanh ghi chỉ ra trong lệnh.
- Nửa byte cao của các thanh ghi vừa nêu trên không bị ảnh hưởng và các cờ cũng không bị ảnh hưởng.

Ví dụ:

Nội dung của thanh ghi A và các thanh ghi như sau:

A = 0FH, R5 = 30H, R0 = 40H, (40H)=7EH

Sau khi thực hiện lệnh: XCH A,R5

⇒ nội dung của thanh ghi A và R5 là:

A = 30H, R5 = 0FH

Sau khi thực hiện lệnh: XCHD A,@R0

⇒ nội dung của thanh ghi A và ô nhớ được chỉ đến bởi R0 là:

A = 0E, (40H) = 7FH

NHÓM LỆNH SỐ HỌC

ADD A, <SOURCE-BYTE>

- **Lệnh này sẽ thực hiện phép cộng nội dung của 1 byte địa chỉ được chỉ ra ở <SOURCE-BYTE> với nội dung của thanh ghi A và kết quả sẽ chứa vào thanh ghi A.**
- **Nếu có số nhớ từ bit 7 hoặc bit 3, các cờ nhớ (C) và cờ nhớ phụ (AC) sẽ được set bằng 1, ngược lại các cờ này sẽ được xóa bằng 0. Khi cộng 2 số nguyên dương thì cờ nhớ C sẽ cho ta biết phép toán bị tràn khi bằng 1.**
- **Cờ tràn (OV) sẽ được set bằng 1 nếu có số nhớ từ bit 6 nhưng không có số nhớ từ bit 7 hoặc có số nhớ từ bit 7 nhưng không có số nhớ từ bit 6, ngược lại cờ này sẽ bị xóa. Khi cộng 2 số nguyên có dấu thì cờ OV cho ta biết kết quả âm được tạo ra từ tổng của 2 số nguyên dương hoặc kết quả dương được tạo ra từ tổng của 2 số nguyên âm.**
- **ADD có 4 kiểu định địa chỉ cho toán hạng nguồn:thanh ghi, trực tiếp, thanh ghi-gián tiếp hoặc tức thời.**

***Ví dụ*: thanh ghi A chứa nội dung là 0C3H, R0 lưu giữ giá trị 0AAH, lệnh: ADD A, R0 tạo ra kết quả là 6DH chứa trong thanh ghi A, cờ AC = 0, cờ C và OV đều bằng 1.**

- **ADD có 4 kiểu định địa chỉ cho toán hạng nguồn:thanh ghi, trực tiếp, thanh ghi-gián tiếp hoặc tức thời.**

Mnemonic	Instruction code								Hexa - decimal	Explanation	No. of MC
	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0			
ADD A, Rn	0	0	1	0	1	n_2	n_1	n_0	28 ÷ 2F	A ← A + Rn	1
ADD A, direct	0	0	1	0	0	1	0	1	25	A ← A + (direct)	1
	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0	Byte 2		
ADD A, @Ri	0	0	1	0	0	1	1	i	26 ÷ 27	A ← A + (Ri)	1
ADD A, #data	0	0	1	0	0	1	0	0	24	A ← A + data	1
	d_7	d_6	d_5	d_4	d_3	d_2	d_1	d_0	Byte 2		

ADDC A, <SOURCE-BYTE>

- **Lệnh này thực hiện và tác động lên các cờ tương tự như lệnh ADD, nhưng nội dung thanh ghi A sẽ cộng thêm giá trị cờ C trước đó.**
- **Nếu có số nhớ từ bit 7 hoặc bit 3, các cờ nhớ và cờ nhớ phụ được set bằng 1, ngược lại các cờ nêu trên được xóa. Khi cộng hai số nguyên không dấu, cờ nhớ cho ta biết phép toán bị tràn khi được set bằng 1.**
- **Cờ OV được set bằng 1 nếu có số nhớ từ bit 6 nhưng không có số nhớ từ bit 7 hoặc có số nhớ từ bit 7 nhưng không có số nhớ từ bit 6. Nghĩa là khi thực hiện phép cộng 2 số có dấu, cờ OV cho ta biết kết quả âm được tạo ra từ hai số nguyên dương hoặc kết quả dương được tạo ra từ hai số nguyên âm.**

***Ví dụ*: thanh ghi A chứa nội dung là 0C3H, R0 lưu giữ giá trị 0AAH, cờ C có giá trị 1, lệnh: ADDC A, R0 tạo ra kết quả là 6EH chứa trong thanh ghi A, cờ AC = 0, cờ C và OV đều bằng 1.**

- **ADDC có 4 kiểu định địa chỉ cho toán hạng nguồn:thanh ghi, trực tiếp, thanh ghi-gián tiếp hoặc tức thời.**

ADDC A, Rn	0	0	1	1	1	n_2	n_1	n_0	38 ÷ 3F	A ← A + Rn + CY	1
ADDC A, direct	0	0	1	1	0	1	0	1	35	A ← A + (direct) + CY	1
	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0	Byte 2		
ADDC A, @Ri	0	0	1	1	0	1	1	i	36 ÷ 37	A ← A + (Ri) + CY	1
ADDC A, #data	0	0	1	1	0	1	0	0	34	A ← A + data + CY	1
	d_7	d_6	d_5	d_4	d_3	d_2	d_1	d_0	Byte 2		

SUBB A, <SOURCE-BYTE>

- **Lệnh này sẽ thực hiện phép trừ bớt đi nội dung của thanh ghi A bởi nội dung của toán hạng được chỉ ra ở <SOURCE-BYTE> cùng với cờ nhớ và cất kết quả vào thanh ghi A.**
- **Cờ C được set bằng 1 nếu có số mượn được cần đến cho bit 7 và sẽ xóa cờ nhớ trong trường hợp ngược lại. Cờ AC được set bằng 1 nếu có số mượn được cần đến cho bit 3 và sẽ xóa cờ nhớ trong trường hợp ngược lại.**
- **Cờ OV sẽ được set bằng 1 nếu có số mượn được cần đến cho bit 6(không phải cho bit 7) hoặc cho bit 7(không phải cho bit 6), ngược lại cờ này sẽ bị xóa. Khi trừ các số nguyên có dấu thì cờ OV cho ta biết kết quả âm được tạo ra khi có 1 số dương trừ cho một số âm hoặc kết quả dương được tạo ra khi trừ 1 số âm cho 1 số dương.**

***Ví dụ*: thanh ghi A chứa nội dung là 0C9H, R2 lưu giữ giá trị 54H và cờ C=1, lệnh: SUBB A, R2 tạo ra kết quả là 74H chứa trong thanh ghi A, cờ AC và cờ C đều bằng 0, cờ OV bằng 1.**

- **SUBB có 4 kiểu định địa chỉ cho toán hạng nguồn:thanh ghi, trực tiếp, thanh ghi-gián tiếp hoặc tức thời.**

SUBB A, Rn	1	0	0	1	1	n_2	n_1	n_0	98 ÷ 9F	A ← A - Rn - CY	1
SUBB A, direct	1	0	0	1	0	1	0	1	95	A ← A - (direct) - CY	1
	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0	Byte 2		
SUBB A, @Ri	1	0	0	1	0	1	1	i	96 ÷ 97	A ← A - (Ri) - CY	1
SUBB A, #data	1	0	0	1	0	1	0	0	94	A ← A - data - CY	1
	d_7	d_6	d_5	d_4	d_3	d_2	d_1	d_0	Byte 2		

***Ví dụ*: thanh ghi A chứa nội dung là 0C9H, R2 lưu giữ giá trị 54H và cờ C=1, lệnh: SUBB A, R2 tạo ra kết quả là 74H chứa trong thanh ghi A, cờ AC và cờ C đều bằng 0, cờ OV bằng 1.**

	1	1	1		1			
A	1	1	0	0	1	0	0	1
-								
R2	0	1	0	1	0	1	0	0
	0	1	1	1	0	1	0	1
-								
Cờ C								1
A	0	1	1	1	0	1	0	0

Kết quả: ***thanh ghi A = 74, cờ C và AC = 0, cờ OV =1***

INC byte

- **Lệnh này sẽ thực hiện tăng 1 đơn vị của byte được chỉ ra trong lệnh.**
- **Các cờ không bị ảnh hưởng.**
- **Nếu giá trị ban đầu chứa trong byte là 0FFH thì sau lệnh này sẽ trở thành 00H.**
- **Có 3 kiểu định địa chỉ cho toán hạng nguồn:thanh ghi, trực tiếp, thanh ghi-gián tiếp.**

Mnemonic	Instruction code								Hexa - decimal	Explanation	No. of MC
	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0			
INC A	0	0	0	0	0	1	0	0	04	A ← A + 1	1
INC Rn	0	0	0	0	1	n_2	n_1	n_0	08 ÷ 0F	Rn ← Rn + 1	1
INC direct	0	0	0	0	0	1	0	1	05	(direct) ← (direct) + 1	1
	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0	Byte 2		
INC @Ri	0	0	0	0	0	1	1	i	06 ÷ 07	(Ri) ← (Ri) + 1	1
INC DPTR	1	0	1	0	0	0	1	1	A3	DPTR ← DPTR + 1	1

DEC byte

- Lệnh này sẽ thực hiện giảm 1 đơn vị của byte được chỉ ra trong lệnh.
- Các cờ không bị ảnh hưởng.
- Nếu giá trị ban đầu chứa trong byte là 00H thì sau lệnh này sẽ trở thành 0FFH.
- Lưu ý đối với thanh ghi DPTR không có lệnh này. Để làm được việc tương đương với lệnh giảm DPTR 1 đơn vị ta sẽ thực hiện bằng chuỗi lệnh sau:

```
DEC   DPL
MOV   R7, DPL
CJNE  R7, #0FFH, SKIP
DEC   DPH
SKIP: (tiếp tục)
```

- Có 3 kiểu định địa chỉ cho toán hạng nguồn:thanh ghi, trực tiếp, thanh ghi-gián tiếp.

DEC A	0	0	0	1	0	1	0	0	14	A ← A - 1	1
DEC Rn	0	0	0	1	1	n_2	n_1	n_0	18 ÷ 1F	Rn ← Rn - 1	1
DEC direct	0	0	0	1	0	1	0	1	15	(direct) ← (direct) - 1	1
	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0	Byte 2		
DEC @Ri	0	0	0	1	0	1	1	i	16 ÷ 17	(Ri) ← (Ri) - 1	1

MUL AB

- **Lệnh này sẽ thực hiện phép nhân các số nguyên không dấu 8bit chứa trong các thanh ghi A và B. Kết quả là một giá trị 16bit, byte thấp được cất trong thanh ghi A và byte cao cất trong thanh ghi B.**
- **Nếu tích số lớn hơn 255 (0FFH) thì cờ tràn sẽ được set bằng 1, ngược lại cờ này bị xóa. Cờ nhớ luôn bị xóa.**

DIV AB

- **Lệnh này sẽ thực hiện phép chia số nguyên không dấu 8bit chứa trong thanh ghi A cho số nguyên không dấu 8bit chứa trong thanh ghi B. Kết quả thương số cất trong thanh ghi A và dư số cất trong thanh ghi B.**
- **Các cờ nhớ và cờ tràn đều bị xóa.**
- **Ngoại lệ: Nếu ban đầu B chứa 00H, giá trị trả về trong thanh ghi A và thanh ghi B không được xác định và cờ tràn được set bằng 1. Cờ nhớ được xóa trong mọi trường hợp.**

DA A

- **Lệnh này thực hiện hiệu chỉnh thập phân nội dung của thanh ghi A đối với phép cộng.**
- **Lệnh trên sẽ hiệu chỉnh giá trị 8bit trong thanh ghi A(là kết quả của phép cộng hai toán hạng có dạng BCD nén trước đó, *phép cộng này được thực hiện bởi các lệnh ADD hoặc ADDC*) để tạo ra hai digit 4 bit.**
- **Nếu các bit từ 0 đến 3 của thanh ghi A có giá trị lớn hơn 9, hoặc nếu cờ nhớ phụ AC được set bằng 1 thì nội dung của A sẽ được cộng với 6 để tạo ra digit BCD ở nữa byte thấp. Phép cộng này sẽ set cờ nhớ nếu số nhớ từ 4 bit thấp chuyển đến tất cả 4bit cao, ngược lại sẽ xóa cờ nhớ.**
- **Nếu cờ nhớ được set bằng 1, hoặc nếu 4 bit cao của thanh ghi A vượt quá 9, các bit cao này sẽ được cộng thêm 6 để tạo ra kết quả là digit BCD ở nữa byte cao và cờ nhớ sẽ không bị xóa. Và cờ nhớ chỉ ra rằng tổng của 2 toán hạng BCD ban đầu lớn hơn 99. Cờ OV không bị ảnh hưởng.**
- **Tất cả sự kiện trên chỉ xảy ra trong một chu kỳ máy. Lệnh này thực hiện phép biến đổi thập phân bằng cách cộng 00H, 06H hoặc 66H với nội dung của thanh ghi A tùy thuộc vào nội dung ban đầu của thanh ghi A và các điều kiện của từ trạng thái chương trình PSW.**
- ***Lưu ý rằng lệnh này không thể đơn giản biến đổi số hex trong thanh ghi A thành số dạng BCD, và cũng không áp dụng cho phép trừ thập phân.***

NHÓM LỆNH LOGIC

ANL <DEST-BYTE>, <SOURCE-BYTE>

- **Lệnh này sẽ thực hiện phép toán AND từng bit giữa hai toán hạng được chỉ ra trong lệnh và lưu kết quả vào toán hạng đích.**
- **Các cờ không bị ảnh hưởng.**
- **Khi toán hạng đích là thanh ghi A, toán hạng nguồn có thể dùng 4 kiểu định địa chỉ. Khi toán hạng đích là địa chỉ trực tiếp thì toán hạng nguồn có thể dùng 2 kiểu định địa chỉ.**

Mnemonic	Instruction code								Hexa - decimal	Explanation	No. of MC
	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0			
ANL A, Rn	0	1	0	1	1	n_2	n_1	n_0	58 ÷ 5F	A ← A AND Rn	1
ANL A, direct	0	1	0	1	0	1	0	1	55	A ← A AND (direct)	1
	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0	Byte 2		
ANL A, @Ri	0	1	0	1	0	1	1	i	56 ÷ 57	A ← A AND (Ri)	1
ANL A, #data	0	1	0	1	0	1	0	0	54	A ← A AND data	1
	d_7	d_6	d_5	d_4	d_3	d_2	d_1	d_0	Byte 2		
ANL direct, A	0	1	0	1	0	0	1	0	52	(direct) ← (direct) AND A	1
	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0	Byte 2		
ANL direct, #data	0	1	0	1	0	0	1	1	53	(direct) ←	1
	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0	Byte 2	(direct) AND data	
	d_7	d_6	d_5	d_4	d_3	d_2	d_1	d_0	Byte 3		

ORL <DEST-BYTE>, <SOURCE-BYTE>

- **Lệnh này sẽ thực hiện phép toán OR từng bit giữa hai toán hạng được chỉ ra trong lệnh và lưu kết quả vào toán hạng đích.**
- **Các cờ không bị ảnh hưởng.**
- **Hai toán hạng cho phép 6 tổ hợp các kiểu định địa chỉ. Khi toán hạng đích là thanh ghi A, toán hạng nguồn có thể dùng 4 kiểu định địa chỉ. Khi toán hạng đích là địa chỉ trực tiếp thì toán hạng nguồn có thể dùng 2 kiểu định địa chỉ.**

ORL A, Rn	0	1	0	0	1	n_2	n_1	n_0	48 ÷ 4F	A ← A OR Rn	1
ORL A, direct	0	1	0	0	0	1	0	1	45	A ← A OR (direct)	1
	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0	Byte 2		
ORL A, @Ri	0	1	0	0	0	1	1	i	46 ÷ 47	A ← A OR (Ri)	1
ORL A, #data	0	1	0	0	0	1	0	0	44	A ← A OR data	1
	d_7	d_6	d_5	d_4	d_3	d_2	d_1	d_0	Byte 2		
ORL direct, A	0	1	0	0	0	0	1	0	42	(direct) ← (direct) OR A	1
	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0	Byte 2		
ORL direct, #data	0	1	0	0	0	0	1	1	43	(direct) ← (direct) OR data	2
	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0	Byte 2		
	d_7	d_6	d_5	d_4	d_3	d_2	d_1	d_0	Byte 3		

XRL <DEST-BYTE>, <SOURCE-BYTE>

- **Lệnh này sẽ thực hiện phép toán OR từng bit giữa hai toán hạng được chỉ ra trong lệnh và lưu kết quả vào toán hạng đích.**
- **Các cờ không bị ảnh hưởng.**
- **Hai toán hạng cho phép 6 tổ hợp các kiểu định địa chỉ. Khi toán hạng đích là thanh ghi A, toán hạng nguồn có thể dùng 4 kiểu định địa chỉ. Khi toán hạng đích là địa chỉ trực tiếp thì toán hạng nguồn có thể dùng 2 kiểu định địa chỉ.**

XRL A, Rn	0	1	1	0	1	n_2	n_1	n_0	68 ÷ 6F	A ← A XOR Rn	1
XRL A, direct	0	1	1	0	0	1	0	1	65	A ← A XOR (direct)	1
	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0	Byte 2		
XRL A, @Ri	0	1	1	0	0	1	1	i	66 ÷ 67	A ← A XOR (Ri)	1
XRL A, #data	0	1	1	0	0	1	0	0	64	A ← A XOR data	1
	d_7	d_6	d_5	d_4	d_3	d_2	d_1	d_0	Byte 2		
XRL direct, A	0	1	1	0	0	0	1	0	62	(direct) ← (direct) XOR A	1
	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0	Byte 2		
XRL direct, #data	0	1	1	0	0	0	1	1	63	(direct) ← (direct) XOR data	2
	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0	Byte 2		
	d_7	d_6	d_5	d_4	d_3	d_2	d_1	d_0	Byte 3		

CLR A

- Lệnh này sẽ thực hiện xóa thanh ghi A(tất cả các bit đều bằng 0).
- Lệnh này không ảnh hưởng đến các cờ.

CPL A

- Lệnh này sẽ thực hiện lấy bù nội dung thanh ghi A. (Các bit 1 được đổi thành 0 và ngược lại.
- Lệnh này không ảnh hưởng đến các cờ.

SWAP A

- Lệnh này thực hiện hoán đổi nội dung hai nữa thấp và cao của thanh ghi A.
- Lệnh này không ảnh hưởng đến các cờ.

RL A

- **Lệnh này sẽ thực hiện quay trái thanh ghi A.**
- **8 bit trong thanh ghi A được quay trái 1 bit. Bit 7 được quay đến vị trí của bit 0.**
- **Lệnh này không ảnh hưởng đến các cờ.**

RLC A

- **Lệnh này sẽ thực hiện quay trái thanh ghi A cùng với cờ nhớ.**
- **8 bit trong thanh ghi A và cờ nhớ được quay trái 1 bit. Bit 7 được di chuyển đến cờ nhớ và trạng thái ban đầu của cờ nhớ được đưa đến vị trí của bit 0.**
- **Lệnh này không ảnh hưởng đến các cờ.**

RR A

- **Lệnh này sẽ thực hiện quay phải thanh ghi A.**
- **8 bit trong thanh ghi A được quay phải 1 bit. Bit 0 được quay đến vị trí của bit 7.**
- **Lệnh này không ảnh hưởng đến các cờ.**

RRC A

- **Lệnh này sẽ thực hiện quay phải thanh ghi A cùng với cờ nhớ.**
- **8 bit trong thanh ghi A và cờ nhớ được quay phải 1 bit. Bit 0 được di chuyển đến cờ nhớ và trạng thái ban đầu của cờ nhớ được đưa đến vị trí của bit 7.**
- **Lệnh này không ảnh hưởng đến các cờ.**

NHÓM LỆNH RẼ NHÁNH

ACALL addr11

- Lệnh này sẽ thực hiện gọi không điều kiện một chương trình con đặt tại địa chỉ được chỉ ra trong lệnh.
- Lệnh này tăng nội dung của PC lên 2 để PC chứa địa chỉ của lệnh kế lệnh ACALL, sau đó cất nội dung 16bit của PC vào stack SP (byte thấp cất trước) và tăng SP lên 2.
- Địa chỉ đích nhận được bằng cách nối liên tiếp: 5 bit cao của thanh ghi PC (sau khi được tăng), các bit từ 5 đến 7 của opcode và byte thứ hai của lệnh. Do đó, chương trình con được gọi phải được bắt đầu trong cùng khối 2K của bộ nhớ chương trình với byte đầu tiên của lệnh theo sau lệnh ACALL.
- Lệnh này không ảnh hưởng đến các cờ.

Ví dụ: Khởi động SP chứa 07H, PC ở vị trí 0123H, nhãn SUBRTNadd11 ở vị trí 0345H của bộ nhớ chương trình. Sau khi thực hiện lệnh: ACALL SUBRTNadd11

- Con trỏ SP có giá trị 09H.
- Ở vị trí 08H và 09H của RAM nội chứa 25H và 01H.
- Thanh ghi PC chứa giá trị 0345H.

LCALL addr16

- Lệnh này sẽ thực hiện gọi một chương trình con đặt tại địa chỉ được chỉ ra trong lệnh.
- Lệnh này thực hiện cộng 3 với nội dung của PC để tạo ra địa chỉ của lệnh kế và sau đó cất địa chỉ kết quả 16bit vào stack, tăng con trỏ SP lên 2. Các byte cao và byte thấp của PC sau đó được nạp bởi byte thứ 2 và thứ 3 trong lệnh LCALL. Việc thực thi chương trình được tiếp tục với lệnh ở địa chỉ này. Như vậy chương trình con có thể bắt đầu ở bất cứ nơi nào trong không gian bộ nhớ chương trình 64KB.
- Lệnh này không ảnh hưởng đến các cờ.

Ví dụ: Khởi động SP chứa 07H, PC ở vị trí 0123H, nhãn SUBRTNadd16 ở vị trí 1234H của bộ nhớ chương trình. Sau khi thực hiện lệnh: LCALL SUBRTNadd16

- Con trỏ SP có giá trị 09H.
- Ở vị trí 08H và 09H của RAM nội chứa 26H và 01H.
- Thanh ghi PC chứa giá trị 1234H.

RET

- **Lệnh trở về từ chương trình con.**
- **Lệnh này sẽ thực hiện lấy lại các byte cao và byte thấp của PC từ stack, giảm SP đi 2. Việc thực thi chương trình tiếp tục với lệnh ở địa chỉ chứa trong PC, trong trường hợp tổng quát là lệnh ngay sau lệnh ACALL hoặc LCALL.**
- **Lệnh này không ảnh hưởng đến các cờ.**

RETI

- **Lệnh trở về từ chương trình phục vụ ngắt.**
- **Lệnh này sẽ thực hiện lấy lại các byte cao và byte thấp của PC từ stack, phục hồi logic ngắt để có thể nhận các ngắt khác có cùng ưu tiên ngắt với ngắt vừa được xử lý. Con trỏ SP giảm đi 2. Việc thực thi chương trình tiếp tục với lệnh ở địa chỉ chứa trong PC, trong trường hợp tổng quát là lệnh ngay sau điểm mà yêu cầu ngắt được phát hiện.**
- **Không có thanh ghi nào khác bị ảnh hưởng. PSW không được tự động phục hồi trở lại trạng thái trước khi xử lý ngắt.**

AJMP addr11

- Lệnh này chuyển việc thực thi chương trình đến địa chỉ được chỉ ra trong lệnh, địa chỉ này được tạo thành bằng cách kết hợp 5bit cao của PC (sau khi tăng PC lên 2), các bit từ 5 đến 7 của opcode và byte thứ 2 của lệnh. Do vậy đích nhảy đến phải ở trong cùng khối 2K của bộ nhớ chương trình với byte đầu tiên của lệnh theo sau lệnh AJMP.

LJMP addr16

- Lệnh này tạo một rẽ nhánh không điều kiện đến địa chỉ được chỉ ra trong lệnh bằng cách nạp các byte cao và byte thấp của PC với các byte thứ hai và byte thứ ba của lệnh. Do vậy địa chỉ đích có thể ở bất cứ nơi nào tyrong không gian địa chỉ của bộ nhớ chương trình 64KB.
- Lệnh này không ảnh hưởng đến các cờ.

SJMP rel

- Lệnh này điều khiển chương trình rẽ nhánh không điều kiện đến địa chỉ được chỉ ra trong lệnh. Địa chỉ đích được tính bằng cách cộng độ dời có dấu trong byte 2 của lệnh với PC(sau khi tắng nội dung PC lên 2). Do vậy, tầm nhảy cho phép là ±128 byte trước và sau lệnh.

JMP @A+DPTR

- Thực hiện nhảy gián tiếp.
- Cộng giá trị không dấu chứa trong thanh ghi A với con trỏ 16bit và nạp kết quả tổng cho bộ đếm chương trình PC. Đây chính là địa chỉ của lệnh kế tiếp được tìm nạp.
- Cả 2 thanh ghi A và DPTR đều không bị thay đổi.
- Lệnh này không ảnh hưởng đến các cờ.

VD:

Một số chẵn từ 0 đến 6 chứa trong thanh ghi A. Các lệnh sau rẽ nhánh đến 1 trong 4 lệnh AJMP trong bảng nhảy bắt đầu ở JMP_TBL:

```
        MOV   DPTR,#JMP_TBL
        JMP   @A+DPTR
JMP_TBL:
        AJMP  LABEL0
        AJMP  LABEL1
        AJMP  LABEL2
        AJMP  LABEL3
```

Nếu thanh ghi A chứa giá trị là 04H khi bắt đầu chuỗi lệnh trên, việc thực thi rẽ nhánh đến LABEL2.

JZ rel

- **Thực hiện nhảy đến địa chỉ được cho trong lệnh nếu nội dung thanh ghi A bằng 0, ngược lại tiếp tục với lệnh tiếp theo.**
- **Địa chỉ của đích nhảy được tính bằng cách cộng độ dời tương đối (có dấu) ở byte thứ hai của lệnh với nội dung của PC (sau khi được tăng bởi 2)**
- **Các cờ không bị ảnh hưởng.**

JNZ rel

- **Thực hiện nhảy đến địa chỉ được cho trong lệnh nếu nội dung thanh ghi A khác 0, ngược lại tiếp tục với lệnh tiếp theo.**
- **Địa chỉ của đích nhảy được tính bằng cách cộng độ dời tương đối (có dấu) ở byte thứ hai của lệnh với nội dung của PC (sau khi được tăng bởi 2)**
- **Các cờ không bị ảnh hưởng.**

JC rel

- **Thực hiện nhảy đến địa chỉ được cho trong lệnh cờ nhớ bằng 1, ngược lại tiếp tục với lệnh tiếp theo.**
- **Địa chỉ của đích nhảy được tính bằng cách cộng độ dời tương đối (có dấu) ở byte thứ hai của lệnh với nội dung của PC (sau khi được tăng bởi 2)**
- **Các cờ không bị ảnh hưởng.**

JNC rel

- **Thực hiện nhảy đến địa chỉ được cho trong lệnh cờ nhớ bằng 0, ngược lại tiếp tục với lệnh tiếp theo.**
- **Địa chỉ của đích nhảy được tính bằng cách cộng độ dời tương đối (có dấu) ở byte thứ hai của lệnh với nội dung của PC (sau khi được tăng bởi 2)**
- **Các cờ không bị ảnh hưởng.**

JB bit,rel

- Thực hiện nhảy đến địa chỉ được cho trong lệnh nếu bit được chỉ ra trong lệnh được set bằng 1, ngược lại tiếp tục với lệnh tiếp theo.
- Địa chỉ của đích nhảy được tính bằng cách cộng độ dời tương đối (có dấu) ở byte thứ ba của lệnh với nội dung của PC (sau khi được tăng đến địa chỉ của byte đầu tiên của lệnh kế). Bit được kiểm tra sẽ không bị thay đổi.
- Các cờ không bị ảnh hưởng.

JNB bit,rel

- Thực hiện tương tự như lệnh JB bit,rel trong trường hợp bit được kiểm tra bằng 0.

JBC bit, rel

- Nếu bit được chỉ ra trong lệnh bằng 1, thì thực hiện xóa bit và rẽ nhánh đến địa chỉ cho trong lệnh, ngược lại tiếp tục với lệnh tiếp theo.
- Địa chỉ của đích nhảy được tính bằng cách cộng độ dời tương đối (có dấu) ở byte thứ ba của lệnh với nội dung của PC (sau khi được tăng đến địa chỉ của byte đầu tiên của lệnh kế).

CJNE <dest-byte>,<src-byte>,rel

- **Lệnh này sẽ so sánh giá trị của 2 toán hạng đầu tiên và rẽ nhánh nếu các giá trị của 2 toán hạng không bằng nhau. Địa chỉ của đích rẽ nhánh được tính bằng cách cộng độ dời tương đối (có dấu) trong byte sau cùng của lệnh với nội dung của PC (sau khi nội dung của PC được tăng đến địa chỉ bắt đầu của lệnh kế tiếp lệnh CJNE).**
- **Hai toán hạng đầu tiên cho phép có 4 tổ hợp các kiểu định địa chỉ:thanh ghi lưu trữ A có thể được so sánh với một byte được định địa chỉ trực tiếp hoặc byte dữ liệu tức thời; và một byte trong RAM được định địa chỉ kiểu gián tiếp hoặc nội dung của một thanh ghi có thể được so sánh với một hằng số tức thời.**
- **Cờ C được set bằng 1 nếu giá trị nguyên không dấu của <dest-byte> nhỏ hơn giá trị nguyên không dấu của <src-byte>, ngược lại (lớn hơn hay bằng) cờ C bị xóa.**

Mnemonic	Instruction code								Hexa - decimal	Explanation	No. of MC
	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0			

CJNE A, direct, rel	1	0	1	1	0	1	0	1	B5	PC ← PC + 3
										IF A < (direct) THEN
										PC ← PC + rel AND CY ← 1
										OR
										IF A > (direct) THEN
										PC ← PC + rel AND CY ← 0
											2

	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0	Byte 2		
	r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0	Byte 3		

CJNE A, #data, rel	1	0	1	1	0	1	0	0	B4	PC ← PC + 3
										IF A < data THEN
										PC ← PC + rel AND CY ← 1
										OR
										IF A > data THEN
										PC ← PC + rel AND CY ← 0
											2

	d_7	d_6	d_5	d_4	d_3	d_2	d_1	d_0	Byte 2		
	r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0	Byte 3		
CJNE Rn, #data, rel	1	0	1	1	1	n_2	n_1	n_0	B8 ÷ BF	PC ← PC + 3 IF Rn < data THEN PC ← PC + rel AND CY ← 1 OR IF Rn > data THEN PC ← PC + rel AND CY ← 0	2
	d_7	d_6	d_5	d_4	d_3	d_2	d_1	d_0	Byte 2		
	r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0	Byte 3		
CJNE @Ri, #data, rel	1	0	1	1	0	1	1	i	B6 ÷ B7	PC ← PC + 3 IF (Ri) < data THEN PC ← PC + rel AND CY ← 1 OR IF (Ri) > data THEN PC ← PC + rel AND CY ← 0	2
	d_7	d_6	d_5	d_4	d_3	d_2	d_1	d_0	Byte 2		
	r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0	Byte 3		

DJNZ <BYTE>, <REL-ADDR>

- Lệnh này thực hiện giảm byte chỉ ra bởi toán hạng đầu trong lệnh đi 1 và rẽ nhánh đến địa chỉ được chỉ ra bởi toán hạng thứ hai trong lệnh nếu kết quả sau khi giảm khác 0.
- Địa chỉ đích của rẽ nhánh được tính bằng cách cộng độ dời tương đối (có dấu) trong byte cuối cùng của lệnh với nội dung của PC (sau khi PC được tằng đến địa chỉ của byte đầu tiên của lệnh kế tiếp).
- Byte được giảm có thể là nội dung của một thanh ghi hoặc được định địa chỉ trực tiếp.
- Lệnh này không ảnh hưởng đến các cờ.

DJNZ Rn, rel	1	1	0	1	1	n_2	n_1	n_0	D8 ÷ DF	PC ← PC + 2 Rn ← Rn - 1 IF Rn ≠ 0 THEN PC ← PC + rel	2
	r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0	Byte 2		
DJNZ direct, rel	1	1	0	1	0	1	0	1	D5	PC ← PC + 2 (direct) ← (direct) - 1 IF (direct) ≠ 0 THEN PC ← PC + rel	2
	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0	Byte 2		
	r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0	Byte 3		

NOP

- Lệnh này không làm gì cả. Việc thực thi chương trình tiếp tục với lệnh tiếp theo. Không có thanh ghi hay cờ nào bị ảnh hưởng.
- Thường dùng trong việc tạo xung.

NHÓM LỆNH XỬ LÝ BIT

CLR <BIT>

- **Thực hiện xóa bit. Lệnh này có thể thao tác trên cờ nhớ hoặc trên một bit bất kỳ được định địa chỉ bit.**
- **Các cờ không bị ảnh hưởng.**

SETB <BIT>

- **Thực hiện đặt bit được chỉ ra trong lệnh bằng 1. Lệnh này có thể thao tác trên cờ nhớ hoặc trên một bit bất kỳ được định địa chỉ bit.**
- **Các cờ không bị ảnh hưởng.**

CPL <BIT>

- **Thực hiện lấy bù bit được chỉ ra trong lệnh. Lệnh này có thể thao tác trên cờ nhớ hoặc trên một bit bất kỳ được định địa chỉ bit.**
- **Các cờ không bị ảnh hưởng.**

ANL C,<SRC-BIT>

- Thực hiện phép AND của bit nguồn với cờ nhớ C. Nếu có dấu / đặt trước bit nguồn thì bit nguồn sẽ được lấy bù trước khi AND với cờ nhớ nhưng giá trị bit nguồn không bị thay đổi bởi thao tác lấy bù.
- Không có cờ nào bị ảnh hưởng.
- Toán hạng nguồn chỉ được sử dụng kiểu định địa chỉ trực tiếp.

ANL C, bit	1 b_7	0 b_6	0 b_5	0 b_4	0 b_3	0 b_2	1 b_1	0 b_0	82 Byte 2	CY ← CY AND (bit)	2
ANL C, /bit	1 b_7	0 b_6	1 b_5	1 b_4	0 b_3	0 b_2	0 b_1	0 b_0	B0 Byte 2	CY ← CY AND $\overline{(bit)}$	2

ORL C,<SRC-BIT>

- Thực hiện phép OR của bit nguồn với cờ nhớ C. Nếu có dấu / đặt trước bit nguồn thì bit nguồn sẽ được lấy bù trước khi AND với cờ nhớ nhưng giá trị bit nguồn không bị thay đổi bởi thao tác lấy bù.
- Không có cờ nào bị ảnh hưởng.
- Toán hạng nguồn chỉ được sử dụng kiểu định địa chỉ trực tiếp.

MOV <DEST-BIT>, <SRC-BIT>,

- **Nội dung của bit được chỉ ra bởi toán hạng thứ hai được sao chép vào vị trí bit được xác định bởi toán hạng thứ nhất. Một trong 2 toán hạng phải là cờ nhớ, toán hạng còn lại có thể là bit bất kỳ được định địa chỉ bit.**
- **Bit nguồn không bị ảnh hưởng.**
- **Các thanh ghi khác và các cờ không bị ảnh hưởng.**

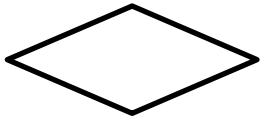
III. CẤU TRÚC CHƯƠNG TRÌNH

Lập trình có cấu trúc là một kỹ thuật dùng để tổ chức và viết chương trình sao cho giảm được độ phức tạp, cải thiện tính rõ ràng của chương trình đồng thời dễ dàng gỡ rối, sửa chữa cũng như thay đổi.

Mọi vấn đề lập trình đều có thể thực hiện được bằng cách dùng 3 cấu trúc:

- Các phát biểu (statement): cung cấp cho ta cơ chế cơ bản để thực hiện một điều gì đó, ví dụ như gán một giá trị cho một biến hoặc gọi một chương trình con chẳng hạn…Bất cứ nơi nào mà một phát biểu đơn được sử dụng, ta đều có thể sử dụng một nhóm các phát biểu hay khối phát biểu.
- Các vòng lặp (loop): được dùng để lặp lại việc thực hiện một thao tác.
- Sự lựa chọn (choice): là sự rẽ nhánh trên đường của người lập trình.

Các ký hiệu thường dùng nhất cho việc lập lưu đồ



Khối quyết định (Decision block)

Mũi tên chỉ đường đi của chương trình (Program flow arrow)

Hộp xử lý (Process box)

Khối xuất nhập (Input/Output block)

Kết thúc chương trình (Program terminator)

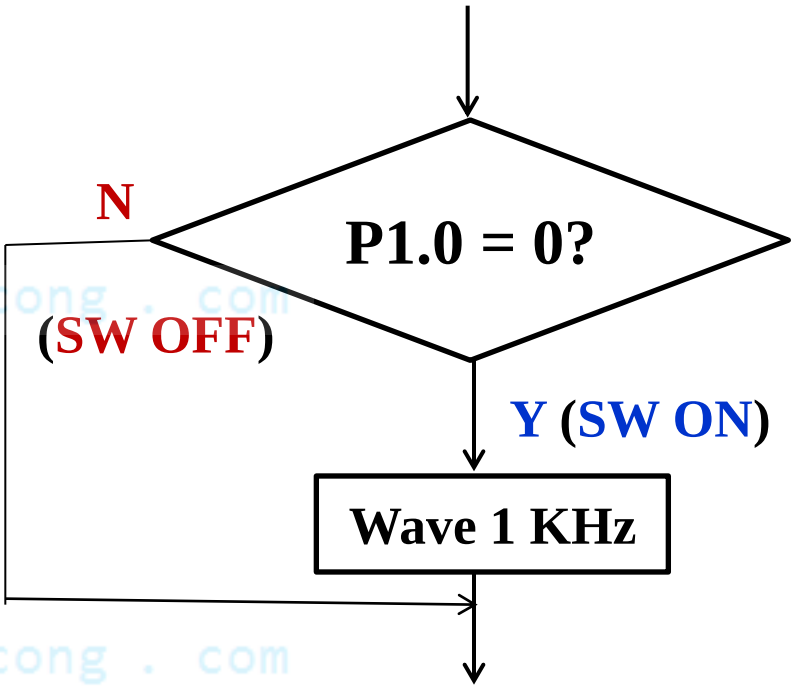
Kết nối qua trang (Page off connector)

Chương trình con (Subroutine)

Cấu trúc IF … THEN …

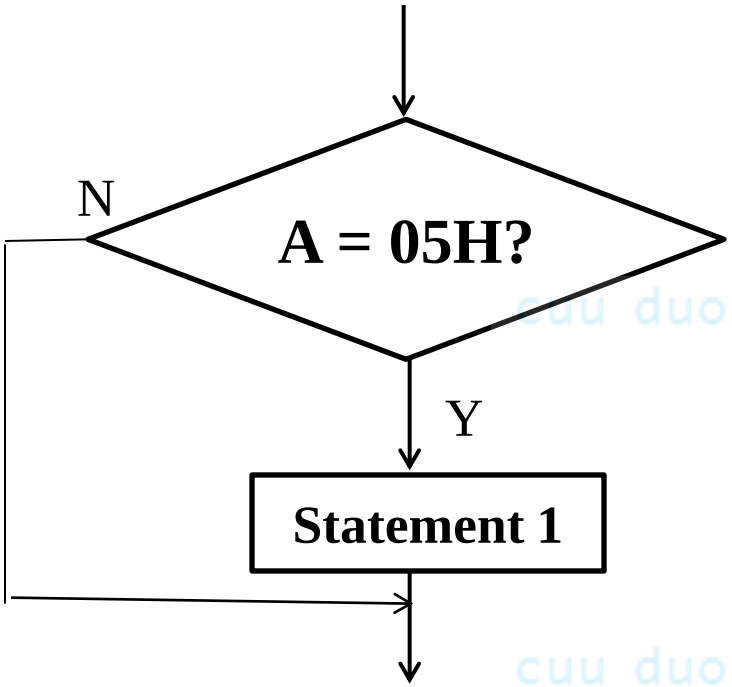
Ví dụ: ***Khi SW tác động đến chân P1.0 = 0 (SW ON) thì tạo 1 sóng vuông 1KHz.***

```
        ORG 0000H

LOOP:   JB P1.0,SW_OFF
SW_ON:  ACALL WAVE1K
SW_OFF: SJMP LOOP
WAVE1K:
        …
        RET

        END
```



Kiểm tra điều kiện bằng/ không bằng - CJNE

Ví dụ: ***So sánh nội dung của thanh ghi A với giá trị 05H.***



CJNE A,#05H,Skip

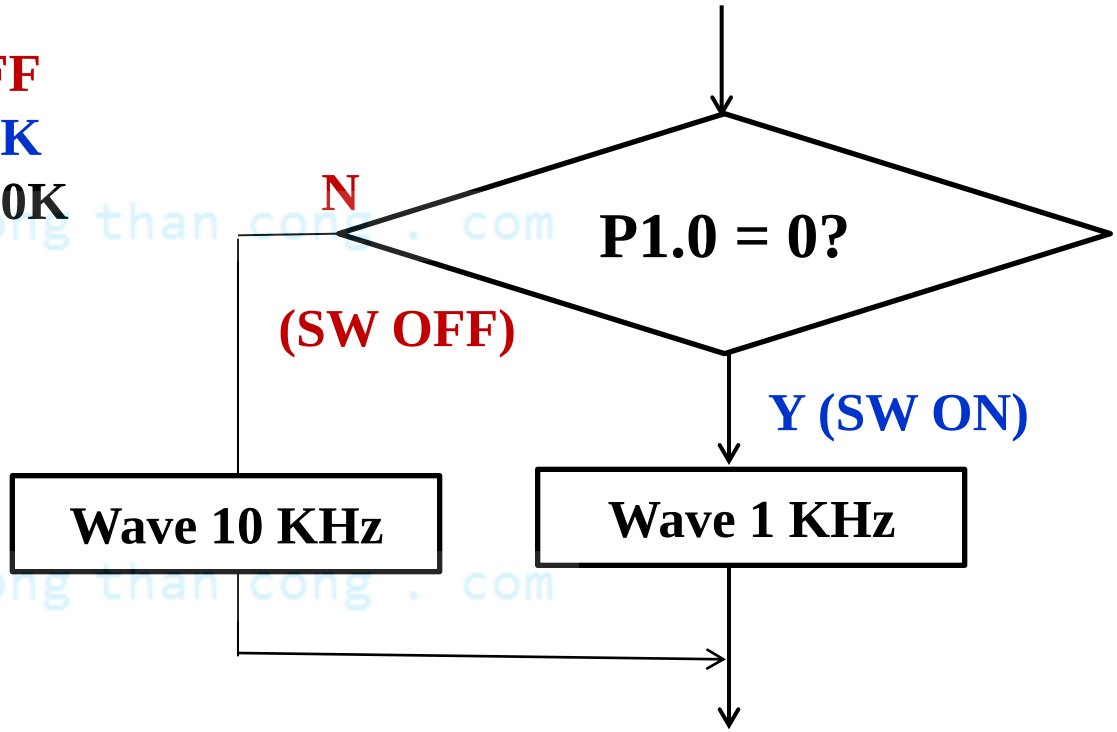
(Statement 1)

Skip: **(Continue)**

Cấu trúc IF … THEN … ELSE …

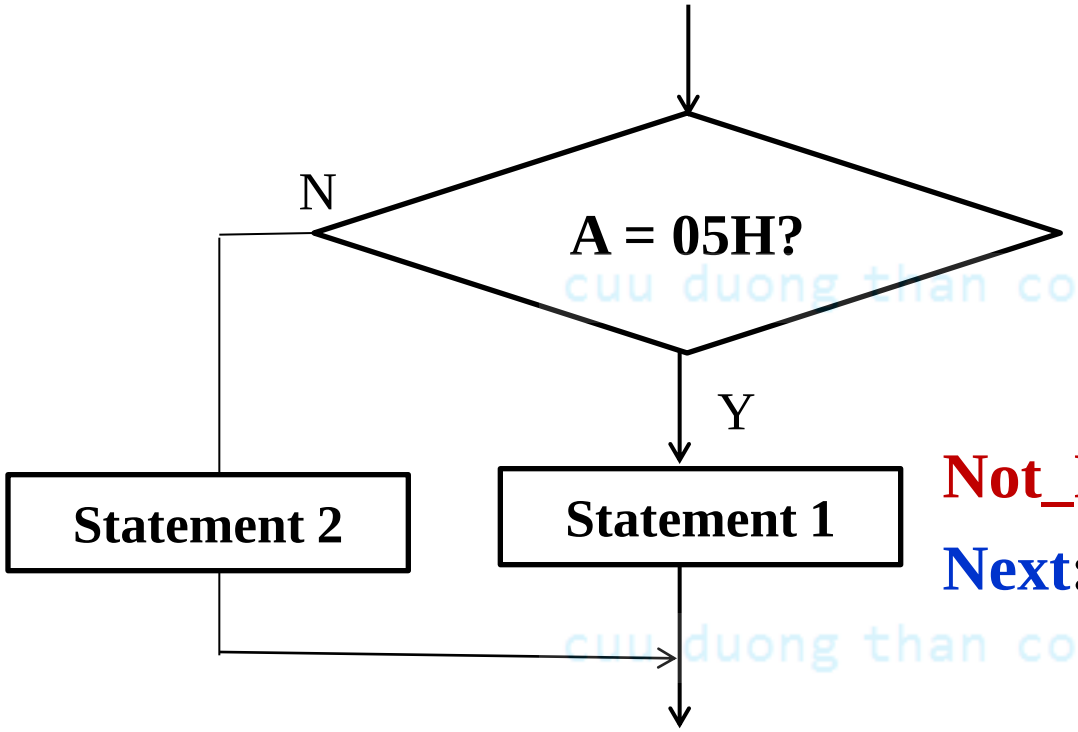
Ví dụ: ***Khi SW tác động đến chân P1.0 = 0 (SW ON) thì tạo 1 sóng vuông 1KHz, ngược lại (SW OFF) tạo sóng 10KHz.***

```
                ORG 0000H
LOOP:           JB P1.0, SW_OFF
                ACALL WAVE1K
SW_OFF:         ACALL WAVE10K
                SJMP LOOP
WAVE1K:
                …
                RET

WAVE10K:
                …
                RET

                END
```



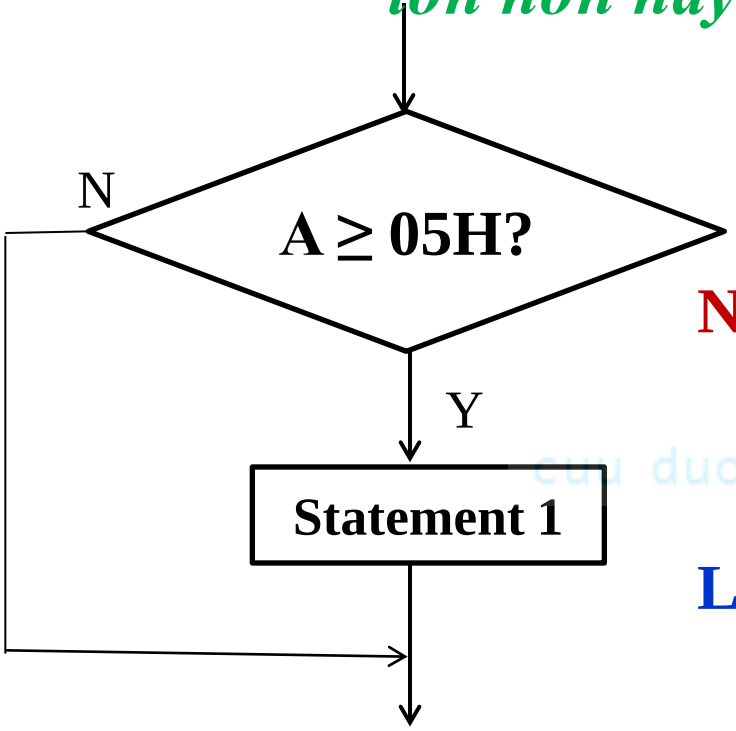
Kiểm tra điều kiện bằng/ không bằng - CJNE

Ví dụ: ***So sánh nội dung của thanh ghi A với giá trị 05H.***




```
CJNE A,#05H,Not_Eq
(Statement 1)
SJMP Next
Not_Eq: (Statement 2)
Next:   (Continue)
```

Kiểm tra điều kiện lớn hơn hay bằng/ nhỏ hơn - CJNE

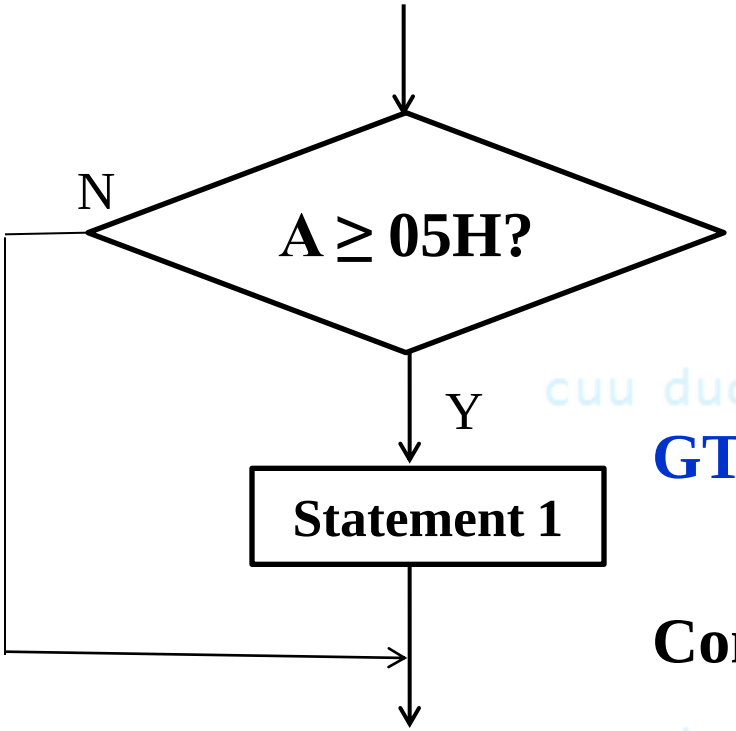



```
CJNE A,#05H,Next
Next:
        JC LessThan
        (Statement 1)
LessThan:
        (Continue)
```



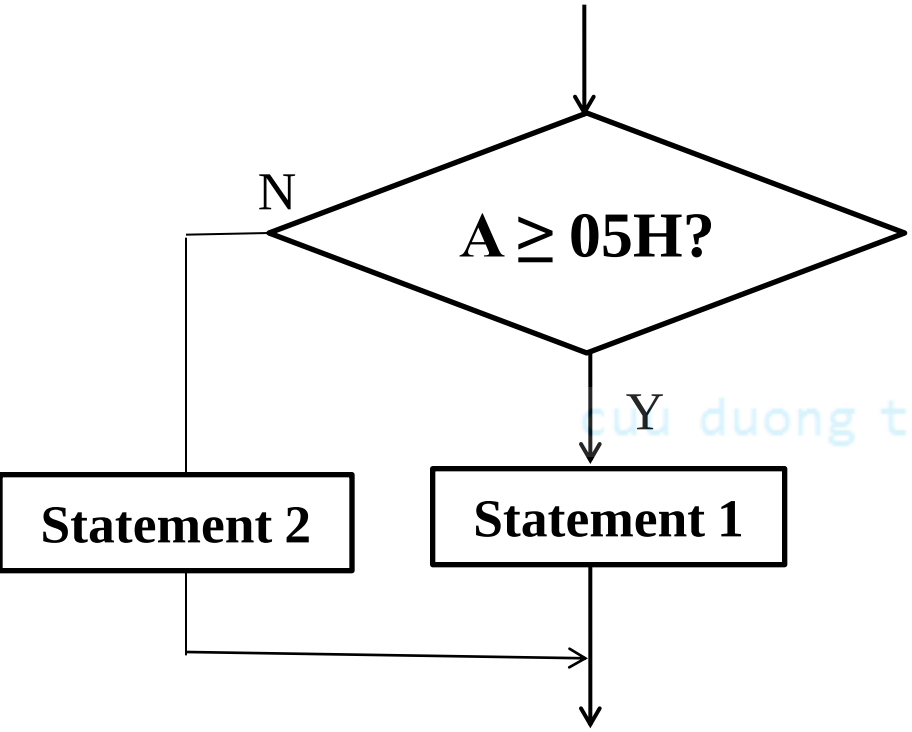
```
CJNE A,#05H,Next   ≡   CJNE A,#05H,$+3
Next: …                …
```

Kiểm tra điều kiện lớn hơn hay bằng/ nhỏ hơn - CJNE




```
CJNE A,#05H,$+3
JNC GT_Eq
SJMP Continue
GT_Eq:
        (Statement 1)
Continue:
        (Continue)
```

Kiểm tra điều kiện lớn hơn hay bằng/ nhỏ hơn - CJNE



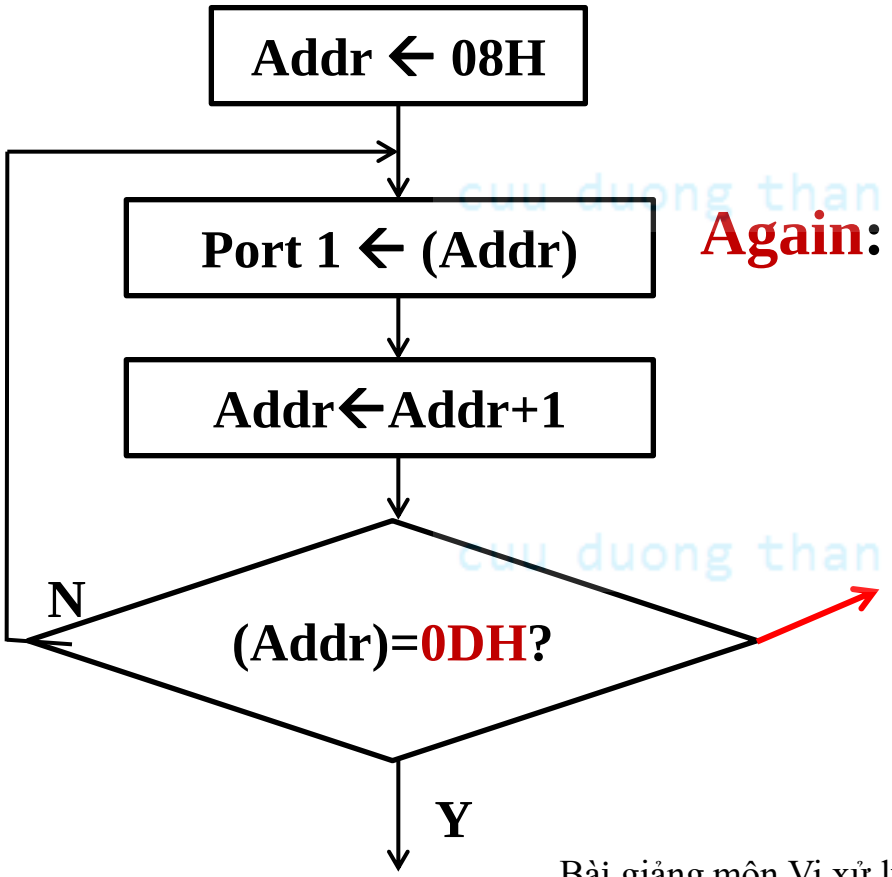

```
CJNE A,#05H,$+3
JNC   GT_Eq
(Statement 2)
SJMP Next
GT_Eq:
(Statement 1)
Next:
(Continue)
```

Ví dụ: ***Kiểm tra nội dung của thanh ghi A, nếu $5 \leq A \leq 10$ thì xuất nội dung của A ra P1, ngược lại xuất ra P2.***

```
        ORG 0
        CJNE A,#5,$+3
        JC PORT2
        CJNE A,#11,$+3
        JNC PORT2
        MOV P1,A
        SJMP DONE
PORT2:  MOV P2,A
DONE:   NOP
        END
```

Cấu trúc vòng lặp REPEAT … UNTIL

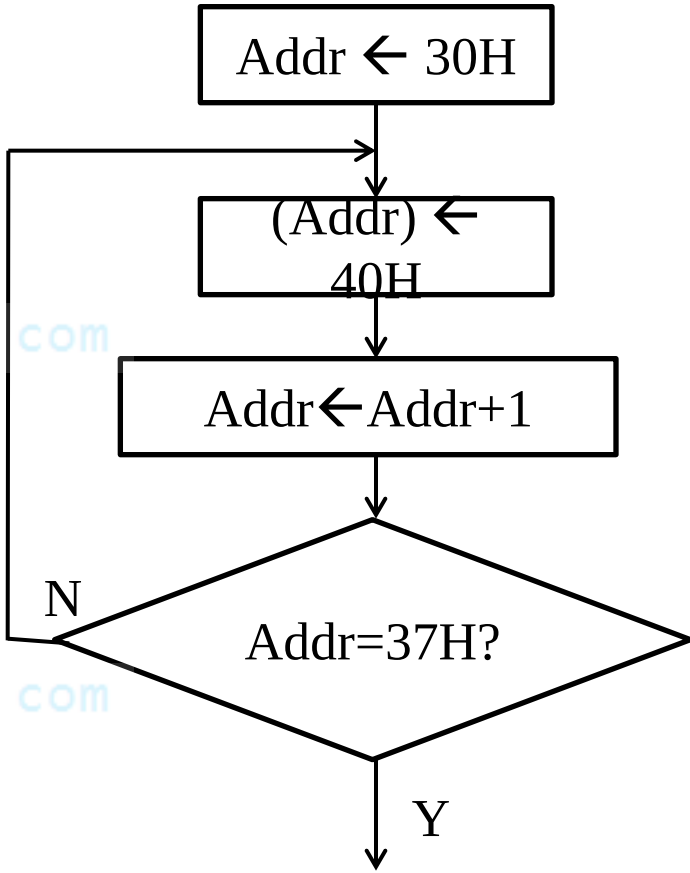
Ví dụ: *Giả sử có một chuỗi mã ASCII được chứa ở RAM nội bắt đầu từ địa chỉ 08H. Chuỗi này sẽ kết thúc bởi mã 0DH. Viết chương trình thực hiện gửi chuỗi này ra Port 1.*



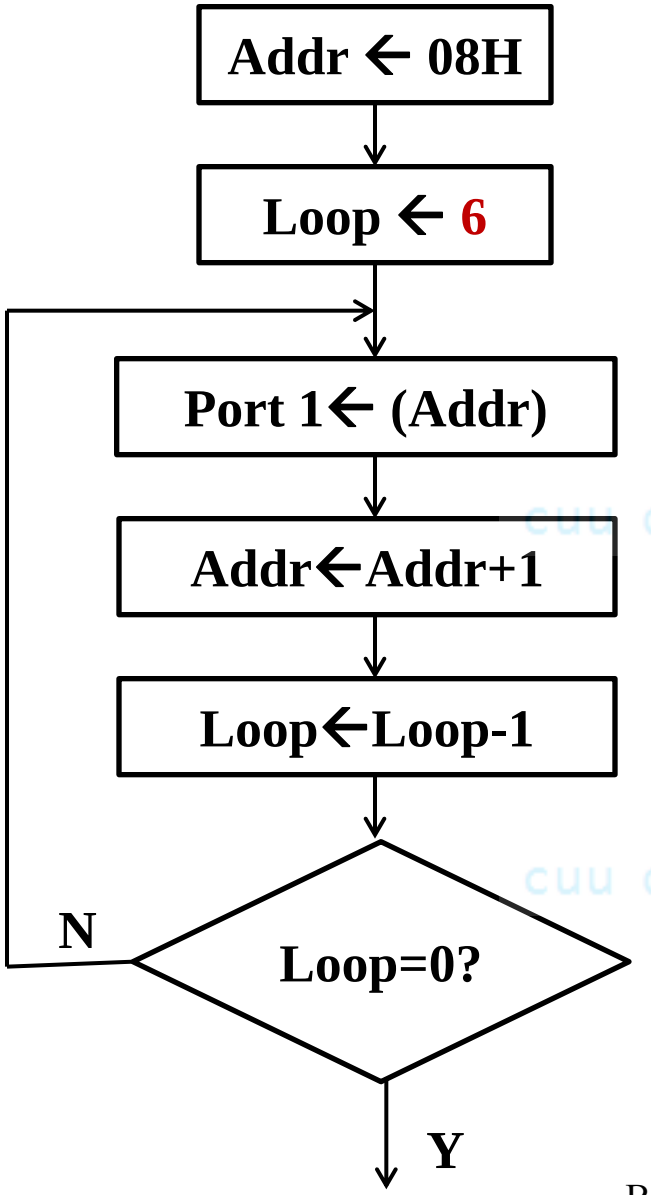

```
ORG 0000H
MOV R0,#08H
Again:
MOV P1,@R0
INC R0
CJNE R0,#0DH,Again
END
```

Ví dụ: *Viết chương trình ghi giá trị 40H vào RAM ngoài bắt đầu từ địa chỉ 30H đến 36H.*

```
        ORG 0000H
        MOV R1,#30H  ;Address=30H
Again:
        MOVX @R1,#40H
        INC R1
        CJNE R1,#37H,Again
        END
```



Cấu trúc vòng lặp FOR



Giả sử có 1 chuỗi mã ASCII chứa trong RAM nội bắt đầu từ địa chỉ 08H đến 0DH. Việt chương trình thực hiện gửi chuỗi này ra Port 1.

0DH – 08H +1 = **6**

→ Loop **6 times**


```
        ORG 0000H
        MOV R0,#08H
        MOV R7,#6
Again:  MOV P1,@R0
        INC R0
        DJNZ R7,Again
        END
```

Cấu trúc vòng lặp FOR

Loop 250 times

```
        MOV R7,#250
LABEL1:
        ...
        ...
        DJNZ R7,LABEL1
```

Loop 500,000 times

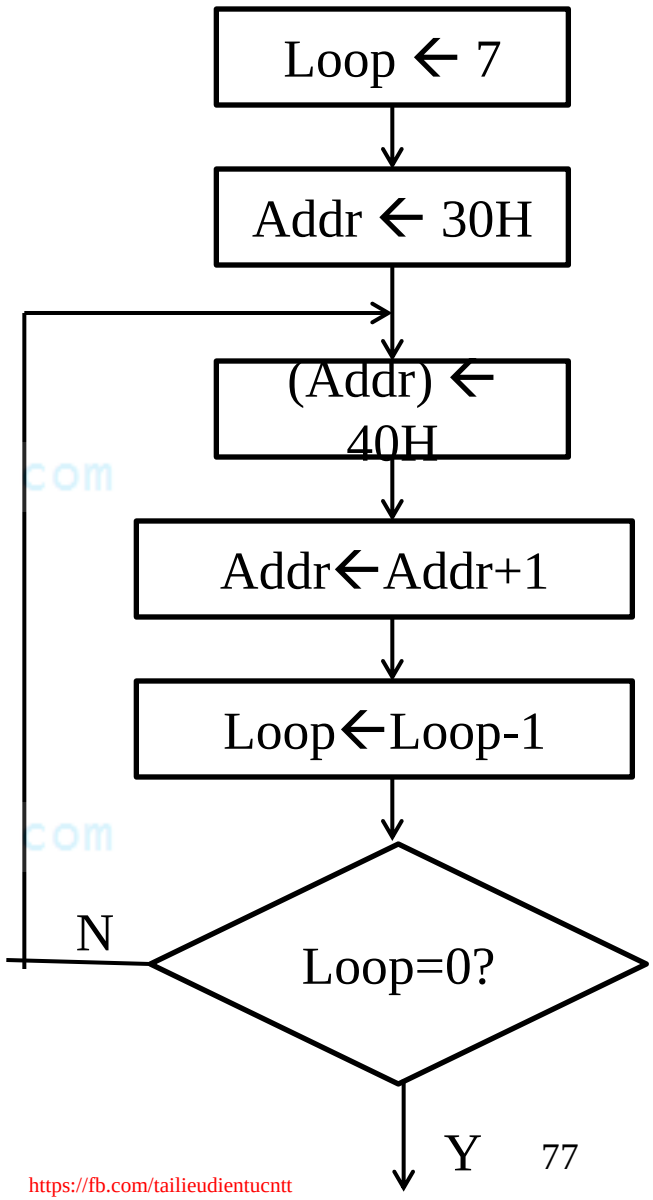
```
        MOV R5,#10
LABEL3:MOV R6,#200
LABEL2:MOV R7,#250
LABEL1:...
        ...
        DJNZ R7,LABEL1
        DJNZ R6,LABEL2
        DJNX R5,LABEL2
```

Loop 50,000 times

```
        MOV R6,#200
LABEL2:
        MOV R7,#250
LABEL1:...
        ...
        DJNZ R7,LABEL1
        DJNZ R6,LABEL2
```

Ví dụ: *Viết chương trình ghi giá trị 40H vào RAM ngoài bắt đầu từ địa chỉ 30H đến 36H.*

```
        ORG 0000H
        MOV R5,#7          ;Loop=7
        MOV R1,#30H        ;Address=30H
Again:
        MOV @R1,#40H
        INC R1
        DJNZ R5,Again
        END
```



Source: Slide T. Lê Chí Thông

Tra bảng

Entry number	Output data
1	data1
2	data2
3	data3
…	…

Entry number

```
        MOV     A,#ENTRY_NUMBER
        MOV     DPTR,#TABLE
        MOVC    A,@A+DPTR
TABLE:  DB      data1, data2, data3, …
```

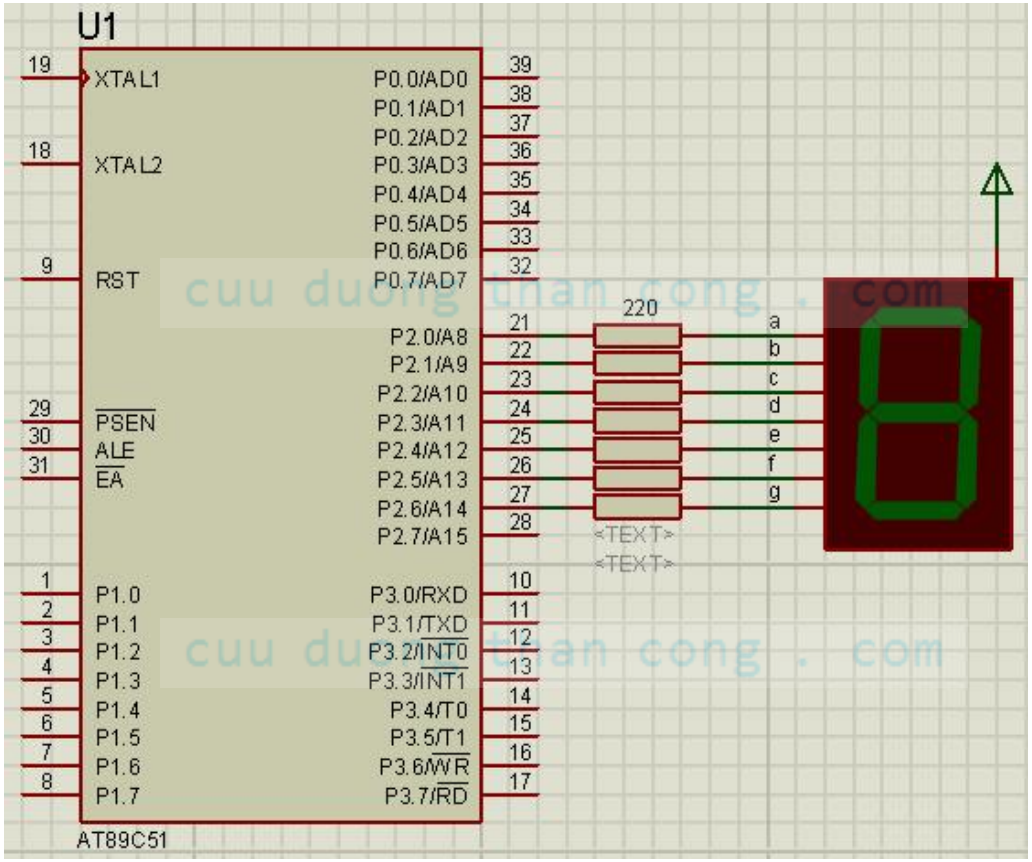
Output data

Ví dụ: *Cho 1 số BCD nén được cất trong RAM nội tại địa chỉ 33H. Viết chương trình tính bình phương của decade cao của số này. Kết quả lưu vào RAM nội tại địa chỉ 34H.*

BCD	Square
0	0
1	1
2	4
3	9
4	16
…	…

```
ORG 0
MOV A,33H
SWAP A
ANL A,#0FH
MOV DPTR,#TABLE
MOVC A,@A+DPTR
MOV 34H,A
TABLE:
DB 0,1,4,9,16,25,36,49,64,81
END
```

Ví dụ: *Cho 1 số BCD nén được cất trong RAM nội tại địa chỉ 34H. Viết chương trình hiển thị decade thấp của số này lên led 7 đoạn anode chung được kết nối đến Port 2 như hình vẽ.*



Source: Slide T. Lê Chí Thông

	D7	D6 g	D5 f	D4 e	D3 d	D2 c	D1 b	D0 a	
0	0	1	0	0	0	0	0	0	40H
1	0	1	1	1	1	0	0	1	79H
2	0	0	1	0	0	1	0	0	24H
3	0	0	1	1	0	0	0	0	30H
4	0	0	0	1	1	0	0	1	19H
5	0	0	0	1	0	0	1	0	12H
6	0	0	0	0	0	0	1	0	02H
7	0	1	1	1	1	0	0	0	78H
8	0	0	0	0	0	0	0	0	00H
9	0	0	0	1	0	0	0	0	10H

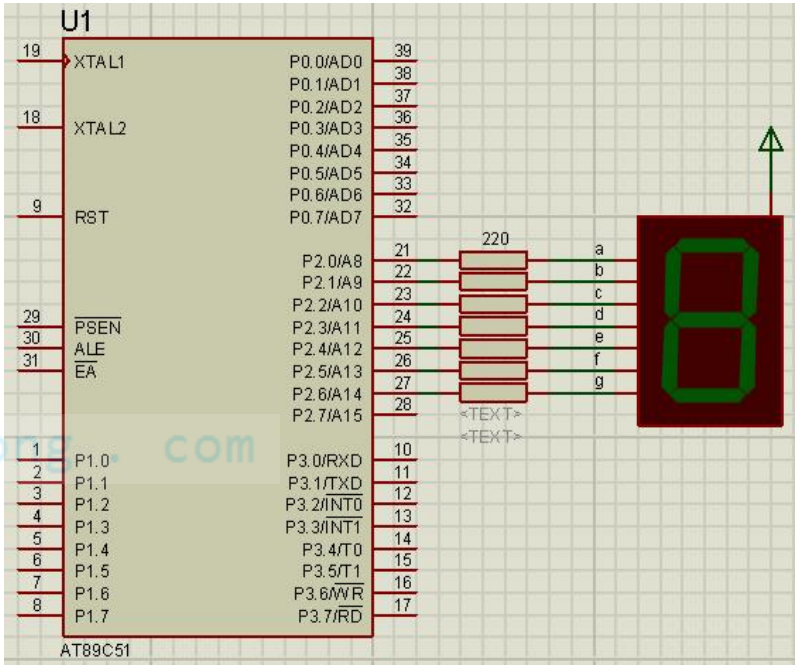
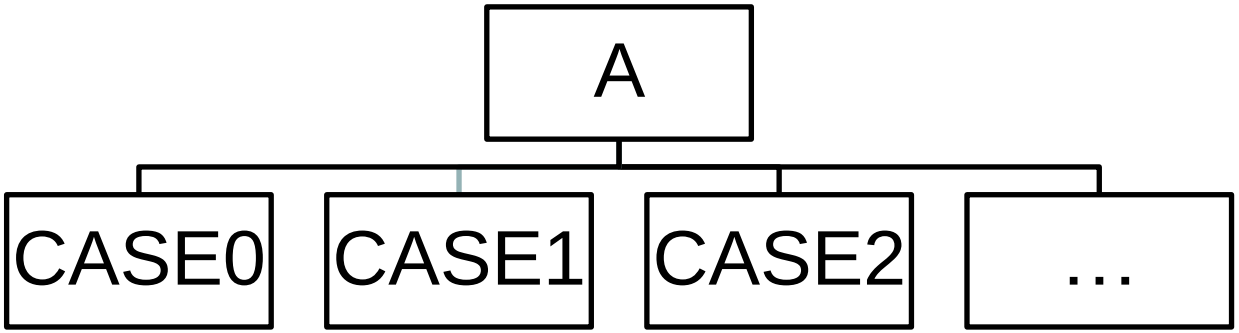


TABLE: DB 40h,79h,24h,30h,19h, 12h,02h,78h,00h,10h

```
        ORG 0000H
        MOV A,34H
        ANL A,#0FH
        MOV DPTR,#TABLE
        MOVC A,@A+DPTR
        MOV P2,A
        SJMP DONE
TABLE:  DB 40h,79h,24h,30h,19h
        DB 12h,02h,78h,00h,10h
DONE:   NOP
        END
```

Cấu trúc – CASE




```
        MOV   DPTR,#JUMP_TABLE
        MOV   A,INDEX_NUMBER
        RL    A
        JMP   @A+DPTR
        …
JUMP_TABLE:
        AJMP  CASE0
        AJMP  CASE1
        AJMP  CASE2
        …
```

Ví dụ: *Giả sử có 2 SW được kết nối đến 2 chân của port 1(tích cực thấp) P1.0 và P1.1. Viết chương trình đọc trạng thái của các SW liên tục mỗi 1 phút và thực hiện các phép toán theo bảng sau:*

P1.1	P1.0	Phép toán thực hiện
0	0	Xoay phải P2 một bit
0	1	Xoay trái P2 một bit
1	0	Lấy bù P2
1	1	Hoán chuyển 2 nữa byte của P2

Cách 1:

```
        ORG 0000H
        MOV DPTR,#JMP_TABLE
LOOP:   MOV A,P1
        ANL A,#00000011B
        RL A
        JMP @A+DPTR
JMP_TABLE:
        AJMP CASE0
        AJMP CASE1
        AJMP CASE2
        AJMP CASE3

CASE0:  MOV A,P2
        RR A
        SJMP CONT

CASE1:  MOV A,P2
        RL A
        SJMP CONT

CASE2:  MOV A,P2
        CPL A
        SJMP CONT

CASE3:  SWAP A

CONT:   MOV P2,A
        ACALL DELAY1M
        SJMP LOOP
DELAY1M:
        …
        RET
        END
```

Cách 2:

```
        ORG 0000H
LOOP:   MOV A,P1
        ANL A,#00000011B
        MOV R6,A
        MOV A,P2
        CJNE R6,#0,NE0
CASE0:  RR A
        SJMP DONE
NE0:    CJNE R6,#1,NE1
CASE1:  RL A
        SJMP DONE
NE1:    CJNE R6,#2,NE2
CASE2:  CPL A
        SJMP DONE
NE1:    CJNE R0,#2,NE2
CASE2:  CPL A
        SJMP DONE
NE2:    SWAP A
DONE:   MOV P2,A
        ACALL DELAY1M
        SJMP LOOP
DELAY1M:
        …
        RET
        END
```