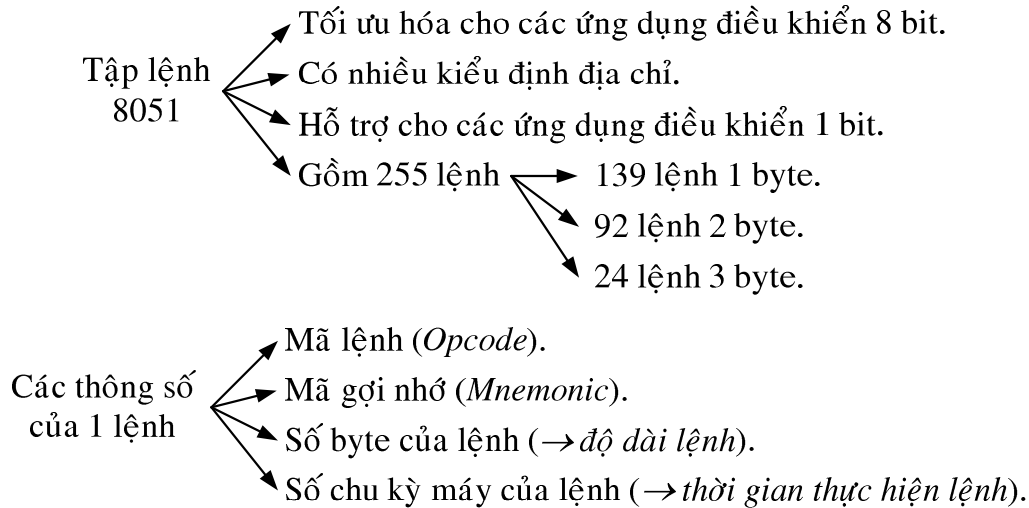


CHƯƠNG 3

TẬP LỆNH CỦA 8051

I. MỞ ĐẦU:

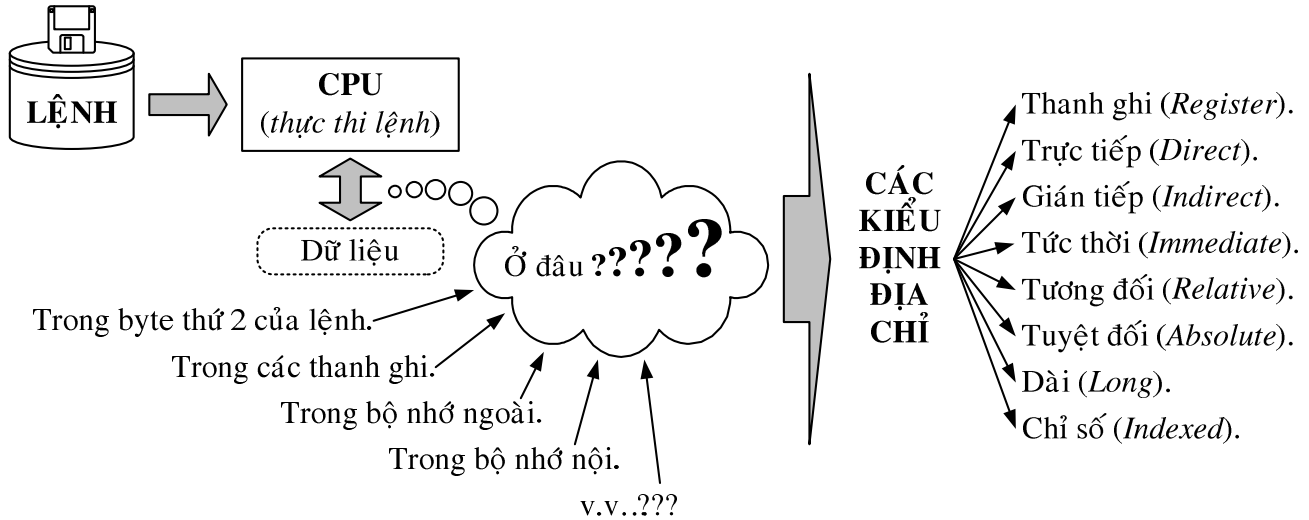


Khuông dạng tổng quát của một dòng lệnh:

[LABEL:] MNEMONIC [OPERAND][,OPERAND]... [;COMMENT]

- Nhãn (*Label*): biểu thị địa chỉ của dòng lệnh (hoặc dữ liệu) theo sau, được dùng trong trường toán hạng của lệnh nhảy, lệnh rẽ nhánh (*SJMP AAA; ACALL BBB; CJNE A, #35H, LOOP; JNB P3.1, TEST_1...*).
- ✓ **Lưu ý về nhãn:**
 - Do người lập trình tự đặt (không được trùng với từ khóa, mã gợi nhớ, chỉ dẫn, toán tử hoặc ký hiệu tiền định nghĩa).
 - Bắt đầu bằng ký tự chữ, dấu chấm hỏi (?), dấu gạch dưới (_).
 - Dài tối đa 31 ký tự.
 - Kết thúc bằng dấu hai chấm (:).
- Mã gợi nhớ (*Mnemonic*): biểu diễn các mã của lệnh hoặc các chỉ dẫn của chương trình dịch hợp ngữ (Mã gợi nhớ: *ADD, SUBB, INC, ...*; Chỉ dẫn: *ORG, EQU, DB, ...*).
- Toán hạng (*Operand*): chứa địa chỉ hoặc dữ liệu mà lệnh sẽ sử dụng. Số lượng toán hạng trong một dòng lệnh phụ thuộc vào từng dòng lệnh (*RET* – không toán hạng, *INC A* – một toán hạng, *ADD A, R0* – hai toán hạng, *CJNE A, #12H, ABC* – ba toán hạng).
- ✓ **Lưu ý về toán hạng:** trong các lệnh có 2 toán hạng thì toán hạng đầu tiên còn được gọi là **toán hạng đích** (*Destination*), toán hạng thứ hai còn được gọi là **toán hạng nguồn** (*Source*).
- Chú thích (*Comment*): làm cho rõ nghĩa cho chương trình. Các chú thích phải nằm trên cùng một dòng và bắt đầu bằng dấu chấm phẩy (;). Các chú thích nếu nằm trên nhiều dòng thì mỗi dòng cũng phải bắt đầu bằng dấu chấm phẩy (;).
- ✓ **Lưu ý:** Chi tiết về phần này xem thêm tại “Chương 7: Lập trình hợp ngữ” trong sách “Họ vi điều khiển – Tổng Văn On”.

II. CÁC KIỂU ĐỊNH ĐỊA CHỈ (ADDRESSING MODE):



1. Định địa chỉ thanh ghi (Register Addressing):

- Được dùng để truy xuất dữ liệu trong các thanh ghi từ **R0** đến **R7**.
- Số byte của lệnh: 1 byte.
- Cấu trúc lệnh:

Opcode	n	n	n
--------	---	---	---
- Ví dụ: **ADD A, R5** $\Rightarrow$ Lệnh cộng nội dung thanh ghi A với nội dung thanh ghi R5. (Giả sử: $(A)=05H$, $(R5)=9AH$).

$\Rightarrow$ Mã lệnh:

00101 101 B
← Mã lệnh cộng Thanh ghi R5

$\Rightarrow$ Mô tả lệnh:

A	05H	→	ADD A, R5
		→	05H + 9AH
R5	9AH	→	

- Ngoài ra, một số trường hợp đặc biệt kiểu định địa chỉ này cũng dùng để truy xuất dữ liệu trong các thanh ghi như: thanh ghi chứa **A**, thanh ghi con trỏ dữ liệu **DPTR**, thanh ghi bộ đếm chương trình **PC**, cờ nhớ **C** và cặp thanh ghi **AB**.
- Ví dụ:

INC A	$\Rightarrow$	Lệnh tăng nội dung thanh ghi A.
INC DPTR	$\Rightarrow$	Lệnh tăng nội dung thanh ghi DPTR.

2. Định địa chỉ trực tiếp (Direct Addressing):

- Được dùng để truy xuất dữ liệu trong các ô nhớ (**00H - FFH**) hay trong các thanh ghi (**A, B, P0-P3, DPH, DPL,...**) của bộ nhớ bên trong chip.
- Số byte của lệnh: 2 byte.
- Cấu trúc lệnh:

Opcode	Direct address
--------	----------------

- Ví dụ: **ADD A, P1** $\Leftrightarrow$ **ADD A, 90H** $\Rightarrow$ Lệnh cộng nội dung thanh ghi A với nội dung thanh ghi port 1 hay ô nhớ 90H. (Giả sử: $(A) = 05H$, $(P1) = (90H) = 9AH$).

$\Rightarrow$ Mã lệnh:

$\Rightarrow$ Mô tả lệnh:

3. Định địa chỉ gián tiếp (Indirect Addressing):

- Được dùng để truy xuất dữ liệu trong các ô nhớ “gián tiếp” của bộ nhớ bên trong chip. Các thanh ghi **R0** và **R1** được dùng để chứa địa chỉ của các ô nhớ gián tiếp ($00H - FFH$) trong chip. Lưu ý rằng, trước các thanh ghi R0, R1 cần phải có dấu “@”.

- Số byte của lệnh: 1 byte.

- Cấu trúc lệnh:

Opcode	i
--------	---

- Ví dụ: **ADD A, @R0** $\Rightarrow$ Lệnh cộng nội dung thanh ghi A với nội dung ô nhớ có địa chỉ chứa trong thanh ghi R0. (Giả sử: $(A) = 05H$, $(R0) = 3BH$, $(3BH) = 9AH$).

$\Rightarrow$ Mã lệnh:

$\Rightarrow$ Mô tả lệnh:

4. Định địa chỉ tức thời (Immediate Addressing):

- Được dùng để truy xuất một hằng số (*giá trị biết trước*) thay vì là một biến (*giá trị không biết trước*) như các kiểu định địa chỉ trên. Lưu ý rằng, trước dữ liệu tức thời cần phải có dấu “#”. Chế độ định địa chỉ tức thời có thể dùng để nạp dữ liệu vào mọi ô nhớ và thanh ghi bất kỳ (đối với thanh ghi 8 bit: $\#00H - \#0FFH$, đối với thanh ghi 16 bit: $\#0000H - \#0FFFFH$).

- Số byte của lệnh: 2 byte.

- Cấu trúc lệnh:

Opcode	Immediate data
--------	----------------

- Ví dụ: **ADD A, #9AH** $\Rightarrow$ Lệnh cộng nội dung thanh ghi A với giá trị 9AH. (Giả sử: $(A) = 05H$).

$\Rightarrow$ Mã lệnh:

$\Rightarrow$ Mô tả lệnh:

5. Định địa chỉ tương đối (Relative Addressing):

- Được sử dụng cho các lệnh nhảy.
- Địa chỉ tương đối (hay offset) là một giá trị **8 bit có dấu**.
- Tầm nhảy giới hạn là: **-128 byte ... 127 byte** từ vị trí của lệnh tiếp theo sau lệnh nhảy.
- Số byte của lệnh: 2 byte.
- Cấu trúc lệnh:

Opcode	Relative offset
--------	-----------------
- *Ví dụ 1: SJMP AAA* ⇒ Lệnh nhảy đến nhãn AAA (Giả sử: nhãn AAA đặt trước lệnh ở địa chỉ 0107H, lệnh SJMP nằm trong bộ nhớ tại địa chỉ 0100H và 0101H).

⇒ Mã lệnh:

10000000B	00000101B
-----------	-----------

Mã lệnh nhảy ← → Offset tương đối

⇒ Mô tả lệnh: xem hình 3.5.2.1.

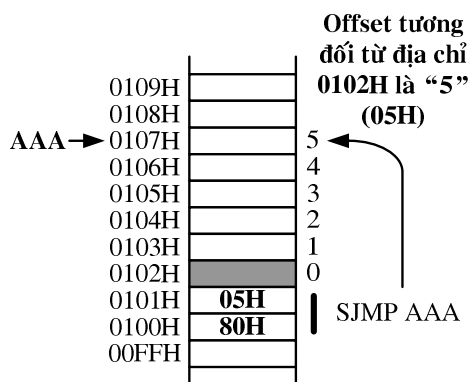
- *Ví dụ 2: SJMP AAA* ⇒ Lệnh nhảy đến nhãn AAA (Giả sử: nhãn AAA đặt trước lệnh ở địa chỉ 203BH, lệnh SJMP nằm trong bộ nhớ tại địa chỉ 2040H và 2041H).

⇒ Mã lệnh:

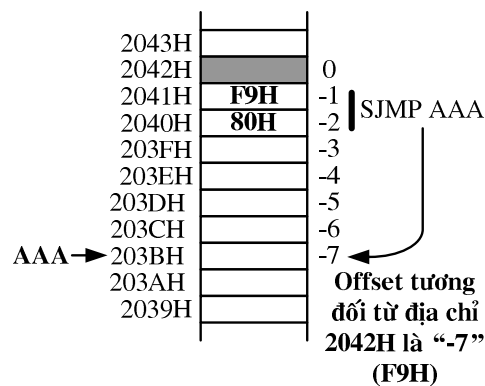
10000000B	11111001B
-----------	-----------

Mã lệnh nhảy ← → Offset tương đối

⇒ Mô tả lệnh: xem hình 3.2.5.2.



Hình 3.2.5.1



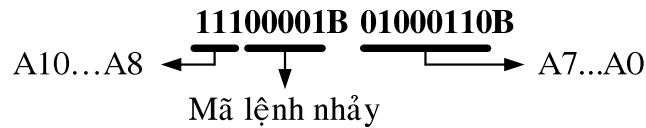
Hình 3.2.5.2

6. Định địa chỉ tuyệt đối (Absolute Addressing):

- Được sử dụng cho các lệnh ACALL và AJMP.
- Địa chỉ tuyệt đối là một giá trị **11 bit**.
- Tầm nhảy giới hạn là: **trong cùng trang 2K hiện hành** (trang 2K chứa lệnh nhảy).
- Số byte của lệnh: 2 byte.
- Cấu trúc lệnh:

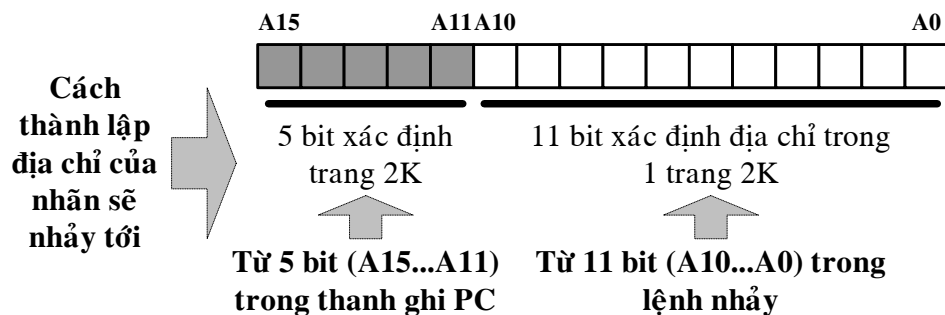
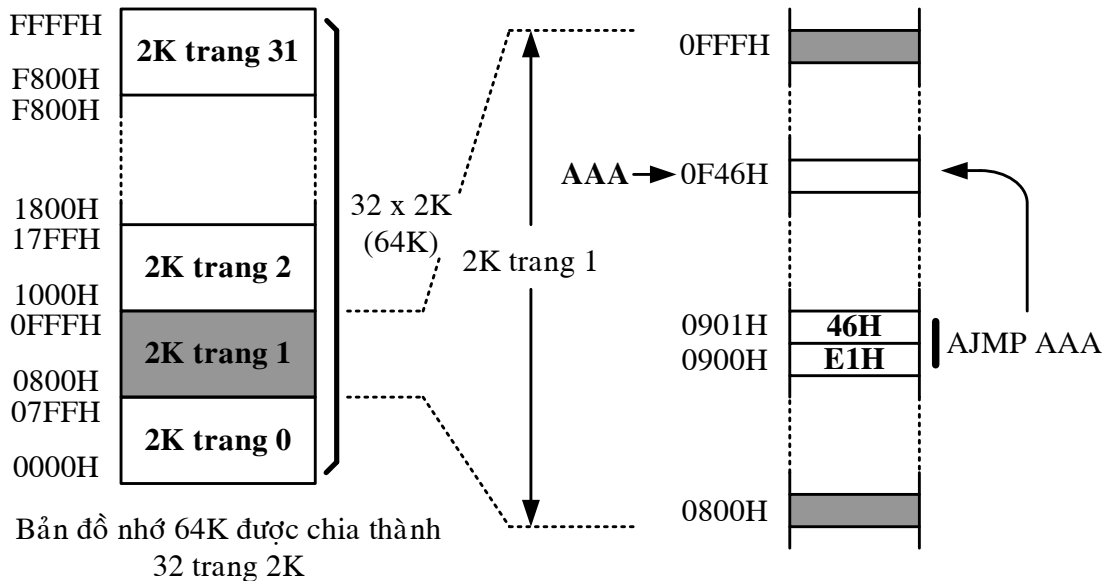
A10 ... A8	Opcode	A7 ... A0
------------	--------	-----------

- Ví dụ: **AJMP AAA** ⇒ Lệnh nhảy đến nhãn AAA (Giả sử: nhãn AAA đặt trước lệnh ở địa chỉ 0F46H, lệnh AJMP nằm trong bộ nhớ tại địa chỉ 0900H và 0901H).



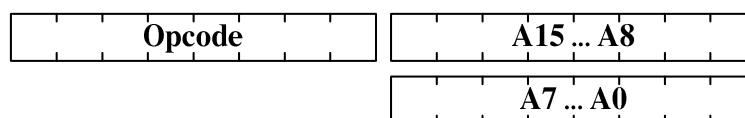
⇒ Mã lệnh:

⇒ Mô tả lệnh:



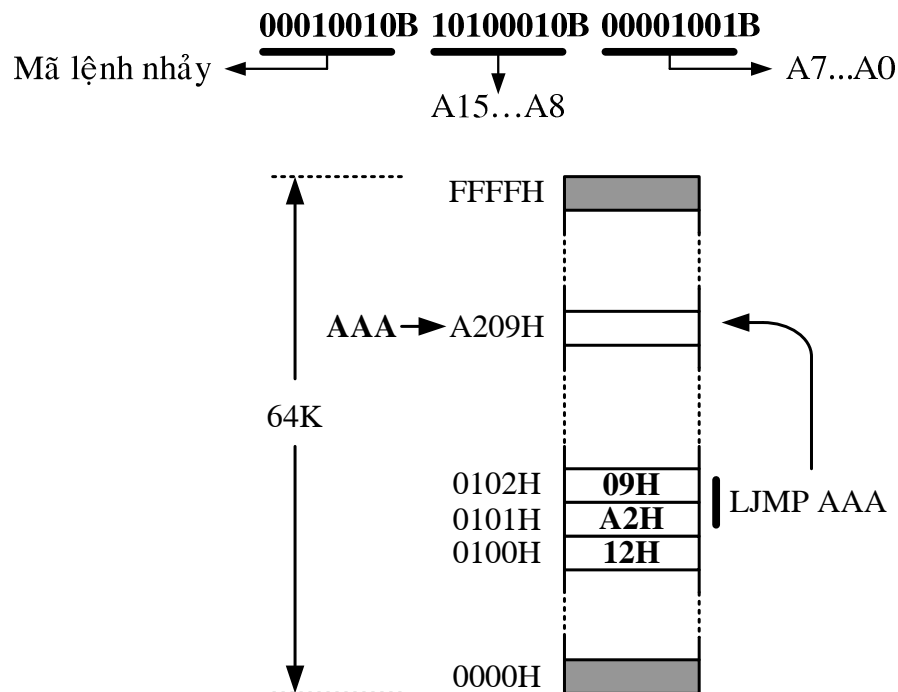
7. Định địa chỉ dài (Long Addressing):

- Được sử dụng cho các lệnh LCALL và LJMP.
- Địa chỉ dài là một giá trị **16 bit**.
- Tầm nhảy giới hạn là: **toàn bộ không gian nhớ 64K**.
- Số byte của lệnh: 3 byte.
- Cấu trúc lệnh:



- Ví dụ: **LJMP AAA** $\Rightarrow$ Lệnh nhảy đến nhãn AAA (Giả sử: nhãn AAA đặt trước lệnh ở địa chỉ A209H, lệnh LJMP nằm trong bộ nhớ tại địa chỉ 0100H, 0101H và 0102H).

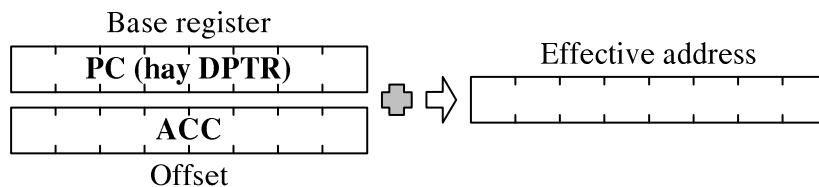
$\Rightarrow$ Mã lệnh:



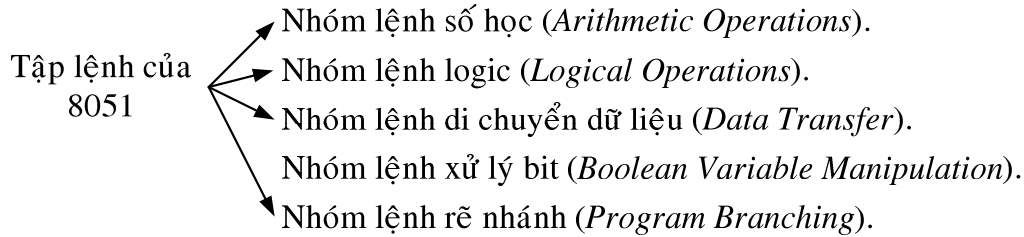
$\Rightarrow$ Mô tả lệnh:

8. Định địa chỉ chỉ số (*Indexed Addressing*):

- Được dùng trong các ứng dụng cần tạo các bảng nhảy hay các bảng tìm kiếm. Kiểu định địa chỉ này dùng một thanh ghi nền (**PC** hay **DPTR**) kết hợp với một *offset* (**A**) để tạo thành dạng địa chỉ hiệu dụng cho lệnh.
- Số byte của lệnh: 1 byte.
- Cấu trúc lệnh:



- Ví dụ: **JMP @A+DPTR** $\Rightarrow$ Lệnh nhảy gián tiếp.

III. TẬP LỆNH CỦA 8051 (8051 INSTRUCTION SET):**Một số ký hiệu dùng trong lệnh:**

Rn	Địa chỉ thanh ghi sử dụng ($R0 - R7$).
direct	Địa chỉ trực tiếp của một byte trong RAM nội ($00H-FFH$).
@Ri	Địa chỉ gián tiếp sử dụng ($R0$ hoặc $R1$).
source	Toán hạng nguồn (Rn , <i>direct</i> hoặc $@Ri$).
dest	Toán hạng đích (Rn , <i>direct</i> hoặc $@Ri$).
#data	Hằng số 8 bit ($\#00H - \#0FFH$).
#data16	Hằng số 16 bit ($\#0000H - \#0FFFFH$).
bit	Địa chỉ trực tiếp của một bit (<i>địa chỉ bit</i>).
rel	Offset 8 bit có dấu.
addr11	Địa chỉ 11 bit.
addr16	Địa chỉ 16 bit.
←	Được thay thế bởi ...
()	Nội dung của ...
(())	Nội dung được chứa bởi ...
rrr	Thanh ghi của dãy thanh ghi ($000 = R0$, $001 = R1$, ..., $111 = R7$).
i	Địa chỉ gián tiếp sử dụng $R0$ ($i = 0$) hoặc $R1$ ($i = 1$).
dddddddd	Các bit dữ liệu.
aaaaaaaa	Các bit địa chỉ.
eeeeeeee	Địa chỉ tương đối.

Một số lưu ý khi lập trình bộ vi điều khiển 8051:

- Để thông báo đó là một giá trị tức thời thì cần phải đặt thêm ký hiệu “#” vào trước giá trị đó. Nếu không có ký hiệu “#” thì giá trị đó được hiểu là địa chỉ của ô nhớ.

MOV A, #12H	;Nạp giá trị 12H vào thanh ghi A.
MOV A, 12H	;Sao chép nội dung của ô nhớ có địa ;chỉ 12H vào thanh ghi A.

Ở đây ta cũng nên lưu ý rằng nếu thiếu ký hiệu “#” thì lệnh trên cũng không gây ra lỗi trong quá trình biên dịch. Vì trình dịch hợp ngữ cho đó là một lệnh hợp lệ. Tuy nhiên, kết quả lập trình sẽ không đúng như ý muốn của người lập trình.

- Các giá trị tức thời nếu có thành phần chữ ($A, B, C, \dots, F$) đứng đầu thì cần phải **thêm số 0** vào trước thành phần chữ và sau ký hiệu “#”. Việc này để báo rằng thành phần chữ đó là một số HEX chứ không phải là một ký tự.

MOV A, #BH	;Thiếu số 0 → gây lỗi khi biên dịch.
MOV A, #0BH	;Thêm số 0 → đúng.
MOV A, #F9H	;Thiếu số 0 → gây lỗi khi biên dịch.
MOV A, #0F9H	;Thêm số 0 → đúng.

Ở đây ta cũng nên lưu ý rằng việc thiếu số 0 thêm vào này sẽ gây lỗi trong quá trình biên dịch đối với các chương trình biên dịch cũ. Ngày nay, một số phần mềm biên dịch đã hỗ trợ việc này. Điều này có nghĩa là ta có thể thêm hay không thêm số 0 vào thì đều không ảnh hưởng gì đến quá trình biên dịch (*không gây ra lỗi khi biên dịch*).

- Trong lệnh, các giá trị tức thời hay địa chỉ của ô nhớ có thể được biểu diễn dưới bất kỳ dạng nào BIN (*nhị phân*), DEC (*thập phân*) hay HEX (*thập lục phân*).

- o Địa chỉ ô nhớ: các câu lệnh sau đây là tương đương nhau:

MOV A, 64H ;Sao chép nội dung của ô nhớ có địa
;chỉ 64H vào thanh ghi A.

MOV A, 100 ;Sao chép nội dung của ô nhớ có địa
;chỉ 64H vào thanh ghi A.

MOV A, 01100100B ;Sao chép nội dung của ô nhớ có địa
;chỉ 64H vào thanh ghi A.

- o Giá trị tức thời: các câu lệnh sau đây là tương đương nhau:

MOV A, #0C9H ;Nạp giá trị C9H vào thanh ghi A.

MOV A, #201 ;Nạp giá trị C9H vào thanh ghi A.

MOV A, #11001001B ;Nạp giá trị C9H vào thanh ghi A.

Lưu ý các hậu tố đi kèm tương ứng cho từng dạng: **B** – dạng BIN (*nhị phân*), **H** – dạng HEX (*thập lục phân*), **D** hoặc **không có hậu tố** – dạng DEC (*thập phân*).

- Chuyển một giá trị tức thời hay địa chỉ của ô nhớ lớn hơn khả năng chứa của một thanh ghi thì sẽ gây ra lỗi (**00H-FFH**: cho thanh ghi hoặc ô nhớ 8 bit; **0000H-FFFFH**: cho thanh ghi 16 bit - DPTR).

MOV A, #123H ;Không hợp lệ vì 123H > FFH.

MOV A, #214 ;Hợp lệ vì 214 (D6H) < FFH (255).

MOV A, #0F2H ;Hợp lệ vì F2H < FFH.

MOV A, 123H ;Không hợp lệ vì 123H > FFH.

MOV A, 200 ;Hợp lệ vì 200 (C8H) < FFH (255).

MOV DPTR, #123H ;Hợp lệ vì 123H < FFFFH (16 bit).

1. Nhóm lệnh số học:

1.1. Lệnh ADD A, <src-byte>:

- Chức năng*: Cộng (*Add*).

- Mô tả*: ADD cộng nội dung của thanh ghi A (**A**) với nội dung của một byte có địa chỉ được chỉ ra trong lệnh (**src-byte**) và đặt kết quả vào thanh ghi A. *Các cờ bị ảnh hưởng*.

- o Cờ **CY** = 1 nếu có số nhớ từ bit 7. Ngược lại **CY** = 0.

- o Cờ **AC** = 1 nếu có số nhớ từ bit 3. Ngược lại **AC** = 0.

- o Cờ **OV** = 1 nếu có số nhớ từ bit 6 nhưng không có số nhớ từ bit 7 hoặc nếu có số nhớ từ bit 7 nhưng không có số nhớ từ bit 6. Ngược lại **OV** = 0.

- o Khi cộng hai số nguyên không dấu và có dấu:

- Số không dấu: **CY** = 1 ⇔ Phép toán có nhớ.

- Số có dấu: **CY** = 1 ⇔ Số dương = Số âm + Số âm.

- ⇔ Số âm = Số dương + Số dương.

• Các dạng lệnh:

ADD A, Rn

Số byte	1
Số chu kỳ	1
Mã đối tượng	00101rrr
Hoạt động	$(A) \leftarrow (A) + (Rn)$

ADD A, direct

Số byte	2
Số chu kỳ	1
Mã đối tượng	00100101 aaaaaaaa
Hoạt động	$(A) \leftarrow (A) + (\text{direct})$

ADD A, @Ri

Số byte	1
Số chu kỳ	1
Mã đối tượng	0010011i
Hoạt động	$(A) \leftarrow (A) + ((Ri))$

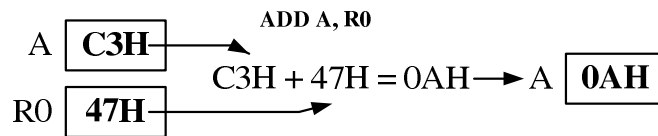
ADD A, #data

Số byte	2
Số chu kỳ	1
Mã đối tượng	00100100 dddddddd
Hoạt động	$(A) \leftarrow (A) + \#data$

- Ví dụ: Cho biết trước $(A)=C3H$, $(R0)=47H$, $(P1)=(90H)=AAH$, $(47H)=D2H$.

Sau khi thực thi lệnh **ADD A, R0** thì:

$(A)=0AH$, $CY=1$, $AC=0$, $OV=0$



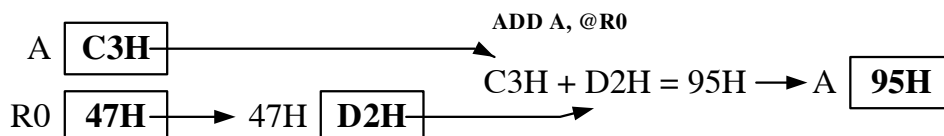
Sau khi thực thi lệnh **ADD A, 90H** hay **ADD A, P1** thì:

$(A)=6DH$, $CY=1$, $AC=0$, $OV=1$



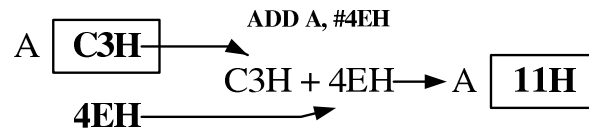
Sau khi thực thi lệnh **ADD A, @R0** thì:

$(A)=95H$, $CY=1$, $AC=0$, $OV=0$



Sau khi thực thi lệnh **ADD A, #4EH** thì:

(A)=11H, CY=1, AC=1, OV=0



1.2. ADDC A, <src-byte>

- **Chức năng:** Cộng có cờ nhớ (*Add with Carry*).
- **Mô tả:** ADDC cộng đồng thời nội dung của thanh ghi A (*A*) với nội dung của byte có địa chỉ được chỉ ra trong lệnh (*src-byte*) và cờ nhớ (*CY*), đặt kết quả vào thanh ghi A. Các cờ bị ảnh hưởng.
 - Cờ **CY** = 1 nếu có số nhớ từ bit 7. Ngược lại **CY** = 0.
 - Cờ **AC** = 1 nếu có số nhớ từ bit 3. Ngược lại **AC** = 0.
 - Cờ **OV** = 1 nếu có số nhớ từ bit 6 nhưng không có số nhớ từ bit 7 hoặc nếu có số nhớ từ bit 7 nhưng không có số nhớ từ bit 6. Ngược lại **OV** = 0.
 - Khi cộng hai số nguyên không dấu và có dấu:
 - Số không dấu: $CY = 1 \Leftrightarrow$ Phép toán có nhớ.
 - Số có dấu: $CY = 1 \Leftrightarrow$ Số dương = Số âm + Số âm.
 $\Leftrightarrow$ Số âm = Số dương + Số dương.
- **Các dạng lệnh:**

ADDC A, Rn

Số byte	1
Số chu kỳ	1
Mã đối tượng	00110rrr
Hoạt động	$(A) \leftarrow (A) + (C) + (Rn)$

ADDC A, direct

Số byte	2
Số chu kỳ	1
Mã đối tượng	00110101 aaaaaaaa
Hoạt động	$(A) \leftarrow (A) + (C) + (\text{direct})$

ADDC A, @Ri

Số byte	1
Số chu kỳ	1
Mã đối tượng	0011011i
Hoạt động	$(A) \leftarrow (A) + (C) + ((Ri))$

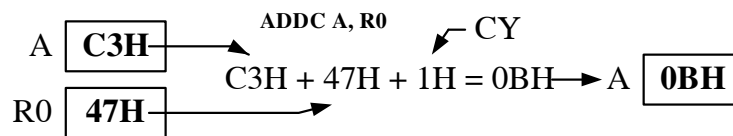
ADDC A, #data

Số byte	2
Số chu kỳ	1
Mã đối tượng	00110100 dddddddd
Hoạt động	$(A) \leftarrow (A) + (C) + \# \text{ data}$

- Ví dụ: Cho biết trước (A)=C3H, (R0)=47H, (P1)=(90H)=AAH, (47H)=D2H và cờ CY=1.

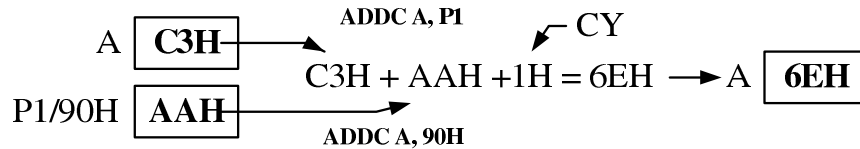
Sau khi thực thi lệnh **ADDC A, R0** thì:

(A)=0BH, CY=1, AC=0, OV=0



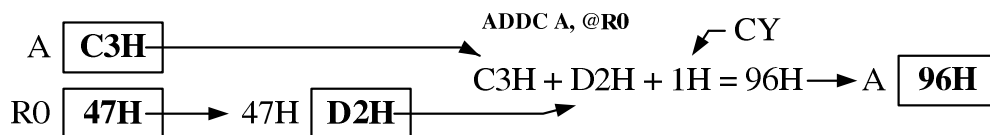
Sau khi thực thi lệnh **ADDC A, 90H** hay **ADDC A, P1** thì:

(A)=6DH, CY=1, AC=0, OV=1



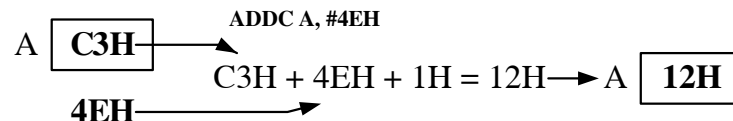
Sau khi thực thi lệnh **ADDC A, @R0** thì:

(A)=96H, CY=1, AC=0, OV=0



Sau khi thực thi lệnh **ADDC A, #4EH** thì:

(A)=11H, CY=1, AC=1, OV=0



1.3. SUBB A, <src-byte>

- Chức năng: Trừ có số mượn (*Subtract with Borrow*).
- Mô tả: SUBB trừ nội dung của thanh ghi A (A) với nội dung của byte có địa chỉ được chỉ ra trong lệnh (*src-byte*) cùng với cờ nhớ và cất kết quả vào thanh ghi A. Các cờ bị ảnh hưởng.
 - Cờ **CY** = 1 nếu có số mượn cho bit 7. Ngược lại **CY** = 0.
 - Cờ **AC** = 1 nếu có số mượn cho bit 3. Ngược lại **AC** = 0.
 - Cờ **OV** = 1 nếu có số mượn cho bit 6 nhưng không có số mượn cho bit 7 hoặc nếu có số mượn cho bit 7 nhưng không có số mượn cho bit 6. Ngược lại **OV** = 0.
 - Khi cộng hai số nguyên không dấu và có dấu:
 - Số không dấu: $CY = 1 \Leftrightarrow$ Phép toán có mượn.
 - Số có dấu: $CY = 1 \Leftrightarrow$ Số dương = Số âm - Số dương.
 $\Leftrightarrow$ Số âm = Số dương - Số âm.

- Các dạng lệnh:

SUBB A, Rn

Số byte	1
Số chu kỳ	1
Mã đối tượng	10011rrr
Hoạt động	$(A) \leftarrow (A) - (C) - (Rn)$

SUBB A, direct

Số byte	2
Số chu kỳ	1
Mã đối tượng	10010101 aaaaaaaa
Hoạt động	$(A) \leftarrow (A) - (C) - (\text{direct})$

SUBB A, @Ri

Số byte	1
Số chu kỳ	1
Mã đối tượng	1001011i
Hoạt động	$(A) \leftarrow (A) - (C) - ((Ri))$

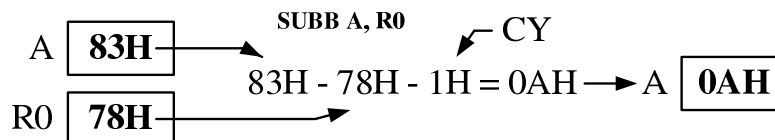
SUBB A, #data

Số byte	1
Số chu kỳ	1
Mã đối tượng	100110100 dddddddd
Hoạt động	$(A) \leftarrow (A) - (C) - \#data$

- Ví dụ: Cho biết trước $(A)=83H$, $(R0)=78H$, $(P1)=(90H)=AAH$, $(78H)=C5H$ và cờ $CY=1$.

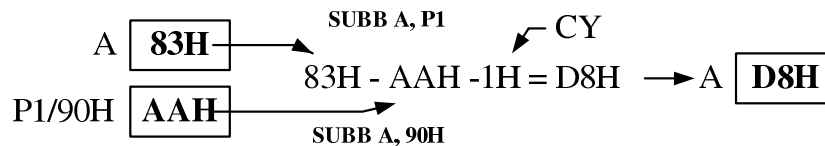
Sau khi thực thi lệnh **SUBB A, R0** thì:

$(A)=0AH$, $CY=0$, $AC=1$, $OV=1$



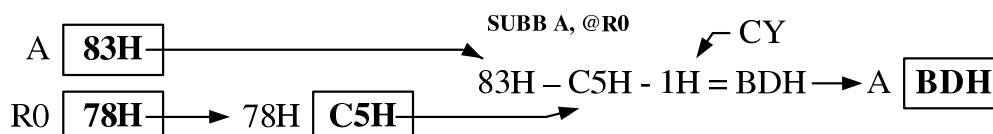
Sau khi thực thi lệnh **SUBB A, 90H** hay **SUBB A, P1** thì:

$(A)=D8H$, $CY=1$, $AC=1$, $OV=0$



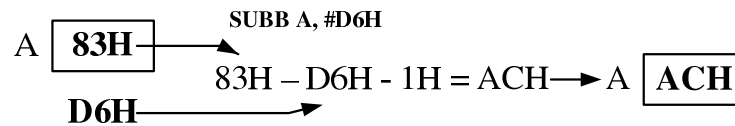
Sau khi thực thi lệnh **SUBB A, @R0** thì:

$(A)=BDH$, $CY=1$, $AC=1$, $OV=0$



Sau khi thực thi lệnh **SUBB A, #D6H** thì:

$$(A)=ACH, CY=1, AC=1, OV=0$$



1.4. INC byte

- **Chức năng:** Tăng thêm 1 (*Increment*).
- **Mô tả:** Tăng nội dung của byte có địa chỉ được chỉ ra trong lệnh (*byte*) thêm 1. Các cờ không bị ảnh hưởng.
- **Lưu ý:** Khi lệnh này được dùng để thay đổi giá trị của một port xuất thì giá trị được dùng làm dữ liệu ban đầu của port được lấy từ bộ chốt dữ liệu xuất, không phải được lấy từ các chân nhập.
- **Các dạng lệnh:**

INC A

Số byte	1
Số chu kỳ	1
Mã đối tượng	00000100
Hoạt động	$(A) \leftarrow (A) + 1$

INC Rn

Số byte	1
Số chu kỳ	1
Mã đối tượng	00001rrr
Hoạt động	$(Rn) \leftarrow (Rn) + 1$

INC direct

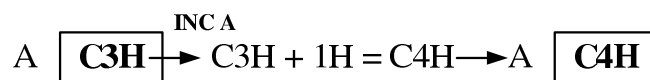
Số byte	2
Số chu kỳ	1
Mã đối tượng	00000101 aaaaaaaaa
Hoạt động	$(direct) \leftarrow (direct) + 1$

INC @Ri

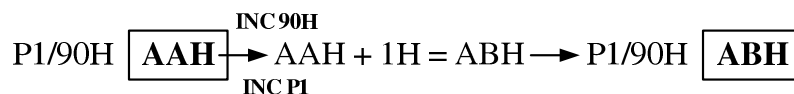
Số byte	1
Số chu kỳ	1
Mã đối tượng	0000011i
Hoạt động	$((Ri)) \leftarrow ((Ri)) + 1$

- **Ví dụ:** Cho biết trước $(A)=C3H$, $(R0)=69H$, $(P1)=(90H)=AAH$, $(69H)=7FH$.

Sau khi thực thi lệnh **INC A** thì: $(A)=C4H$



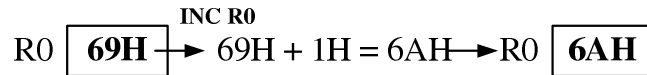
Sau khi thực thi lệnh **INC 90H** hay **INC P1** thì: $(P1)=(90H)=ABH$



Sau khi thực thi lệnh **INC @R0** thì: ($@R0$)=(69H)=80H



Sau khi thực thi lệnh **INC R0** thì: $R0=6AH$



1.5. INC DPTR

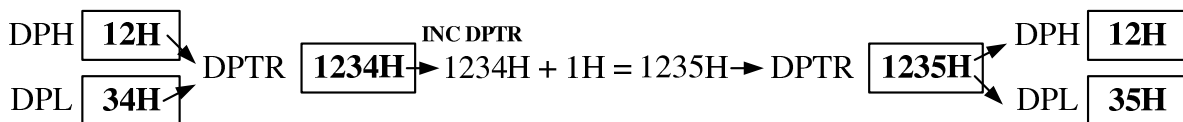
- Chức năng:** Tăng con trỏ dữ liệu (*Increment Data Pointer*).
- Mô tả:** Tăng nội dung của thanh ghi con trỏ dữ liệu 16-bit thêm 1. Các cờ không bị ảnh hưởng.

Số byte	1
Số chu kỳ	2
Mã đối tượng	10100011
Hoạt động	$(DPTR) \leftarrow (DPTR) + 1$

- Ví dụ 1:** Cho biết trước $(DPTR)=1234H$.

Sau khi thực thi lệnh **INC DPTR** thì:

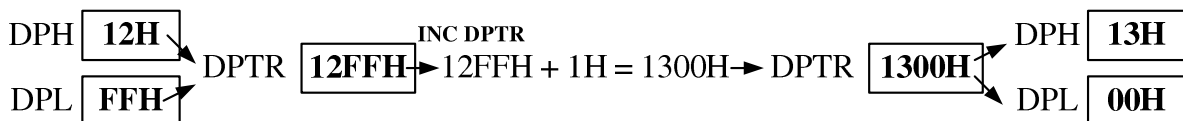
$(DPTR)=1235H$ với $(DPH)=12H$ và $(DPL)=35H$



- Ví dụ 2:** Cho biết trước $(DPH)=12H$ và $(DPL)=FFH$.

Sau khi thực thi lệnh **INC DPTR** thì:

$(DPTR)=1300H$ với $(DPH)=13H$ và $(DPL)=00H$



- Lưu ý:** Không có lệnh giảm nội dung của DPTR (**DEC DPTR**). Nếu muốn giảm nội dung của DPTR ta phải viết một đoạn chương trình con để thực hiện điều này. Chương trình con được minh họa như sau:

DEC_DPTR: PUSH ACC DEC DPL MOV A, DPL CJNE A, #0FFH, SKIP DEC DPH	;Chương trình con giảm DPTR. ;Cất tạm giá trị ACC. ;Giảm byte thấp của DPTR. ;So sánh byte thấp của DPTR ;với FFH. ;Giảm byte cao của DPTR.
SKIP: POP ACC RET	;Phục hồi giá trị ACC.

1.6. DEC byte

- **Chức năng:** Giảm bớt 1 (*Decrement*).
- **Mô tả:** Giảm nội dung của byte có địa chỉ được chỉ ra trong lệnh (**byte**) bớt 1. Các cờ không bị ảnh hưởng.
- **Lưu ý:** Khi lệnh này được dùng để thay đổi giá trị của một port xuất thì giá trị được dùng làm dữ liệu ban đầu của port được lấy từ bộ chốt dữ liệu xuất, không phải được lấy từ các chân nhập.
- **Các dạng lệnh:**

DEC A

Số byte	1
Số chu kỳ	1
Mã đối tượng	00010100
Hoạt động	$(A) \leftarrow (A) - 1$

DEC Rn

Số byte	1
Số chu kỳ	1
Mã đối tượng	00011rrr
Hoạt động	$(Rn) \leftarrow (Rn) - 1$

DEC direct

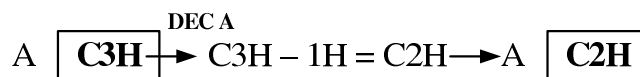
Số byte	2
Số chu kỳ	1
Mã đối tượng	00010101 aaaaaaaa
Hoạt động	$(\text{direct}) \leftarrow (\text{direct}) - 1$

DEC @Ri

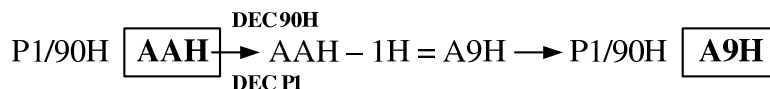
Số byte	1
Số chu kỳ	1
Mã đối tượng	0001011i
Hoạt động	$((Ri)) \leftarrow ((Ri)) - 1$

- **Ví dụ:** Cho biết trước $(A)=C3H$, $(R0)=60H$, $(P1)=(90H)=AAH$, $(60H)=7AH$.

Sau khi thực thi lệnh **DEC A** thì: $(A)=C2H$



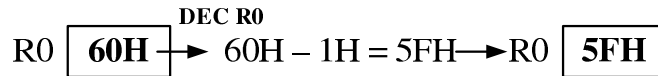
Sau khi thực thi lệnh **DEC 90H** hay **DEC P1** thì: $(P1)=(90H)=A9H$



Sau khi thực thi lệnh **DEC @R0** thì: $(@R0)=(60H)=79H$



Sau khi thực thi lệnh **DEC R0** thì: R0=5FH



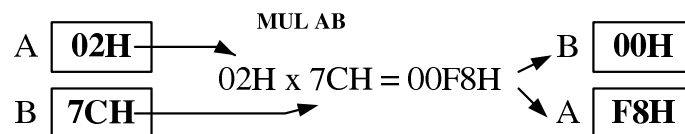
1.7. MUL AB

- **Chức năng:** Nhân (*Multiply*).
- **Mô tả:** MUL AB nhân các số nguyên không dấu 8-bit chứa trong thanh ghi A và thanh ghi B. Tích số là một giá trị 16 bit, **byte thấp** (8 bit thấp) được cất trong thanh ghi A còn **byte cao** (8 bit cao) được cất trong thanh ghi B.
Nếu tích số lớn hơn 255 (0FFH) thì cờ tràn OV=1. Cờ nhớ CY luôn luôn bị xóa.

Số byte	1
Số chu kỳ	4
Mã đối tượng	10100100
Hoạt động	(B) ← HIGH BYTE OF (A) × (B) (A) ← LOW BYTE OF (A) × (B)

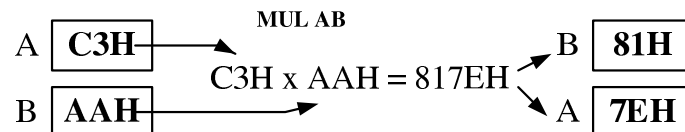
- **Ví dụ 1:** Cho biết trước (A)=02H, (B)=7CH.

Sau khi thực thi lệnh **MUL AB** thì: (B)= 00H, (A)= F8H, CY=0, OV=0.



- **Ví dụ 2:** Cho biết trước (A)=C3H, (B)=AAH.

Sau khi thực thi lệnh **MUL AB** thì: (B)= 81H, (A)= 7EH, CY=0, OV=1.



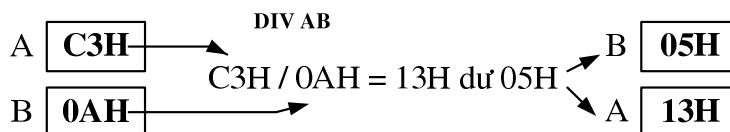
1.8. DIV AB

- **Chức năng:** Chia (*Divide*).
- **Mô tả:** DIV AB chia số nguyên không dấu 8-bit chứa trong thanh ghi A cho số nguyên không dấu 8-bit chứa trong thanh ghi B. **Thương số** được cất trong thanh ghi A còn **số dư** được cất trong thanh ghi B. Cờ CY và cờ OV bị xóa.
Nếu ban đầu B chứa 00H, giá trị trả về trong thanh ghi A và thanh ghi B không được xác định và cờ OV=1. Cờ CY được xóa trong mọi trường hợp.

Số byte	1
Số chu kỳ	4
Mã đối tượng	10000100
Hoạt động	(A) ← QUOTIENT OF (A) / (B) (B) ← REMAINDER OF (A) / (B)

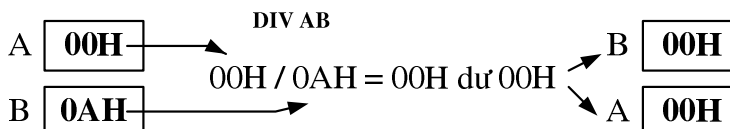
- Ví dụ 1: Cho biết trước (A)=C3H, (B)=0AH.

Sau khi thực thi lệnh **DIV AB** thì: (B)= 05H, (A)= 13H, CY=0, OV=0.



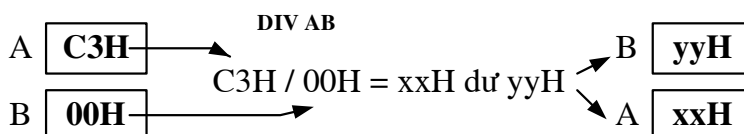
- Ví dụ 2: Cho biết trước (A)=00H, (B)=0AH.

Sau khi thực thi lệnh **DIV AB** thì: (B)= 00H, (A)= 00H, CY=0, OV=0.



- Ví dụ 3: Cho biết trước (A)=C3H, (B)=00H.

Sau khi thực thi lệnh **DIV AB** thì: (B)= yyH, (A)= xxH, CY=0, **OV=1**.



1.9. DA A

- Chức năng: Hiệu chỉnh thập phân nội dung của thanh ghi A đối với phép cộng (*Decimal-adjust Accumulator for Addition*)

- Mô tả: DA A hiệu chỉnh giá trị 8-bit trong thanh ghi A (giá trị này là kết quả phép cộng hai toán hạng có dạng BCD - gói trước đó) để tạo ra hai digit 4 bit. **Phép cộng được thực hiện bởi lệnh ADD hoặc ADDC, lệnh DA A không áp dụng cho phép trừ SUBB).**

Nếu cờ AC = 1 hoặc nếu 4 bit thấp của thanh ghi A có giá trị > “9” (xxxx1010 – xxxx1111), thì “6” được cộng với nội dung của thanh ghi A để tạo ra số BCD ở 4 bit thấp. Sau khi cộng, cờ CY = 1 nếu có số nhớ từ 4 bit thấp chuyển đến tất cả 4 bit cao.

Nếu cờ CY = 1 hoặc nếu 4 bit cao của thanh ghi A có giá trị > “9” (1010xxxx – 1111xxxx), thì “6” được cộng với 4 bit cao để tạo ra số BCD ở 4 bit cao. Sau khi cộng cờ CY = 1 nếu có số nhớ từ 4 bit cao nhưng cờ CY không bị xóa. Vậy thì cờ CY 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, 60H hay 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: DA A không thể đơn giản biến đổi số hex trong thanh ghi A thành số dạng BCD, DA A cũng không áp dụng cho phép trừ thập phân.

Số byte	1
Số chu kỳ	1
Mã đối tượng	11010100

Hoạt động

Giả sử nội dung của thanh ghi A là BCD

IF $[(A3 - A0) > 9] \text{ OR } [(AC) = 1]$

THEN $(A3 - A0) \leftarrow (A3 - A0) + 6$

AND

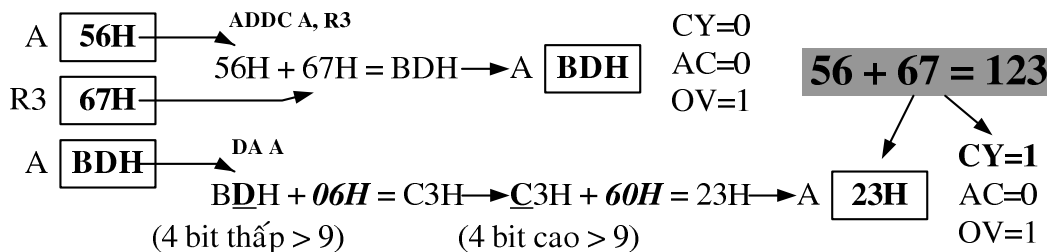
IF $[(A7 - A4) > 9] \text{ OR } [(C) = 1]$

THEN $(A7 - A4) \leftarrow (A7 - A4) + 6$

- Ví dụ 1: Cho biết trước (A)=56H $\rightarrow$ biểu diễn BCD của số 56
(R3)=67H $\rightarrow$ biểu diễn BCD của số 67

Sau khi thực thi chuỗi lệnh: **ADD A, R3**
DA A

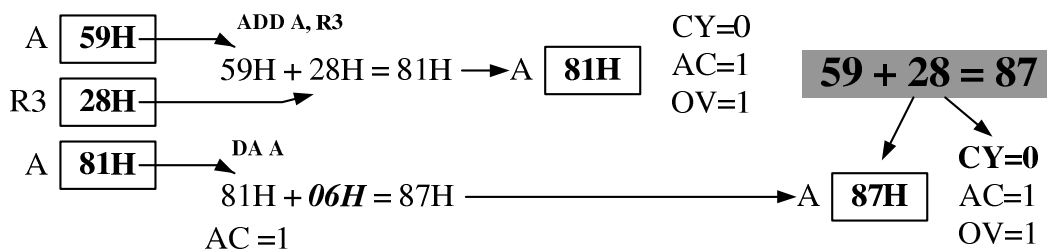
thì: cờ CY=1 và (A)=23H $\rightarrow$ biểu diễn BCD của số 123 (56+67)



- Ví dụ 2: Cho biết trước (A)=59H $\rightarrow$ biểu diễn BCD của số 59
(R3)=28H $\rightarrow$ biểu diễn BCD của số 28

Sau khi thực thi chuỗi lệnh: **ADD A, R3**
DA A

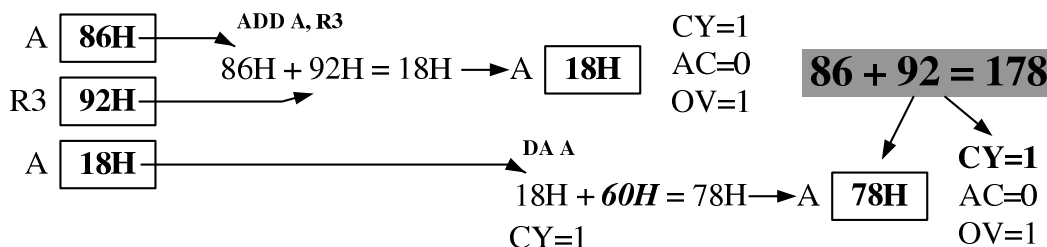
thì: cờ CY=0 và (A)=87H $\rightarrow$ biểu diễn BCD của số 87 (59+28)



- Ví dụ 3: Cho biết trước (A)=86H $\rightarrow$ biểu diễn BCD của số 86
(R3)=92H $\rightarrow$ biểu diễn BCD của số 92

Sau khi thực thi chuỗi lệnh: **ADD A, R3**
DA A

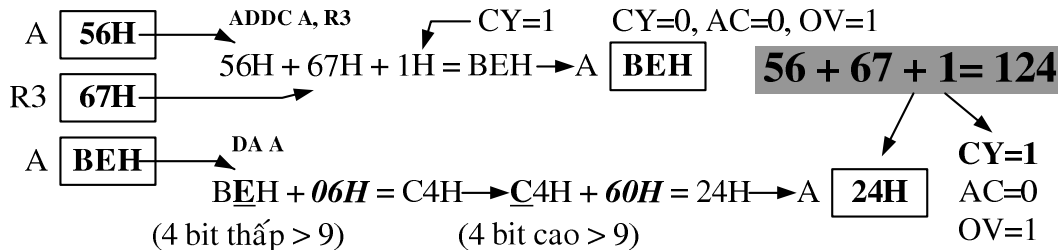
thì: cờ CY=1 và (A)=78H $\rightarrow$ biểu diễn BCD của số 178 (86+92)



- Ví dụ 4: Cho biết trước (A)=56H → biểu diễn BCD của số 56
(R3)=67H → biểu diễn BCD của số 67
cờ CY=1

Sau khi thực thi chuỗi lệnh: **ADDC A, R3**
DA A

thì: cờ CY=1 và (A)=24H → biểu diễn BCD của số 124 (56+67+1)



- Lưu ý: Các giá trị BCD có thể được tăng thêm 1 đơn vị hoặc giảm đi 1 đơn vị bằng cách **cộng với 01H (khi tăng)** hoặc **cộng với 99H (khi giảm)**.

- Ví dụ 1: Giả sử cho (A)=39H → biểu diễn BCD của số 39.

Sau khi thực thi chuỗi lệnh: **ADD A, #01H**
DA A

thì: cờ CY=0 và (A)=40H → biểu diễn BCD của số 40.

- Ví dụ 2: Giả sử cho (A)=30H → biểu diễn BCD của số 30.

Sau khi thực thi chuỗi lệnh: **ADD A, #99H**
DA A

thì: cờ CY=1 và (A)=29H → biểu diễn BCD của số 29.

2. Nhóm lệnh logic:

Bảng trạng thái của các phép toán logic
AND – OR – XOR – CPL

A	B	A AND B	A OR B	A XOR B	CPL A
0	0	0	0	0	1
0	1	0	1	1	1
1	0	0	1	1	0
1	1	1	1	0	0

2.1. ANL <dest-byte>, <src-byte>

- Chức năng: AND hai toán hạng (*Logical-AND*).
- Mô tả: ANL thực hiện phép toán AND từng bit giữa hai toán hạng được chỉ ra trong lệnh và lưu kết quả vào toán hạng đích (**dest-byte**). Các cờ không bị ảnh hưởng.
- Lưu ý: Khi lệnh này được dùng để sửa đổi một port xuất, giá trị được dùng làm dữ liệu ban đầu của port được đọc từ bộ chốt dữ liệu xuất, không phải từ các chân port.

- Các dạng lệnh:

ANL A, Rn

Số byte	1
Số chu kỳ	1
Mã đối tượng	01011rrr
Hoạt động	$(A) \leftarrow (A) \text{ AND } (Rn)$

ANL A, direct

Số byte	2
Số chu kỳ	1
Mã đối tượng	01010101 aaaaaaaa
Hoạt động	$(A) \leftarrow (A) \text{ AND } (\text{direct})$

ANL A, @Ri

Số byte	1
Số chu kỳ	1
Mã đối tượng	0101011i
Hoạt động	$(A) \leftarrow (A) \text{ AND } ((Ri))$

ANL A, #data

Số byte	2
Số chu kỳ	1
Mã đối tượng	01010100 dddddddd
Hoạt động	$(A) \leftarrow (A) \text{ AND } \#data$

ANL direct, A

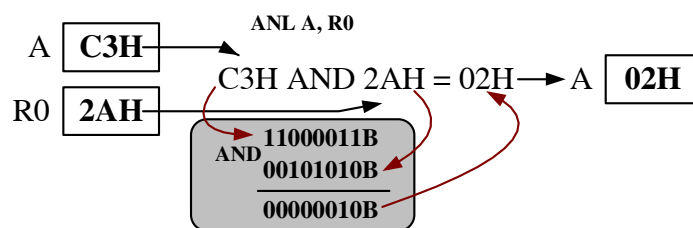
Số byte	2
Số chu kỳ	1
Mã đối tượng	01010010 aaaaaaaa
Hoạt động	$(\text{direct}) \leftarrow (\text{direct}) \text{ AND } (A)$

ANL direct, #data

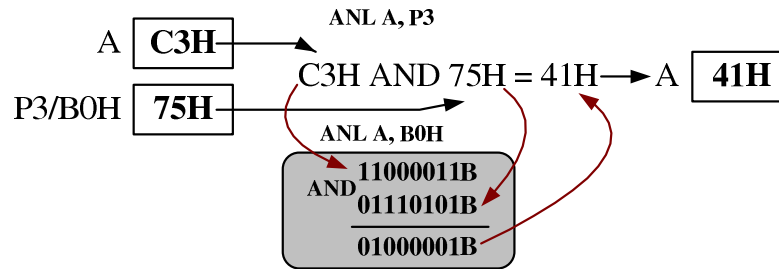
Số byte	3
Số chu kỳ	2
Mã đối tượng	01010011 aaaaaaaa dddddddd
Hoạt động	$(\text{direct}) \leftarrow (\text{direct}) \text{ AND } \#data$

- Ví dụ: Cho biết trước $(A)=C3H$, $(R0)=2AH$, $(P3)=(B0H)=75H$, $(2AH)=55H$.

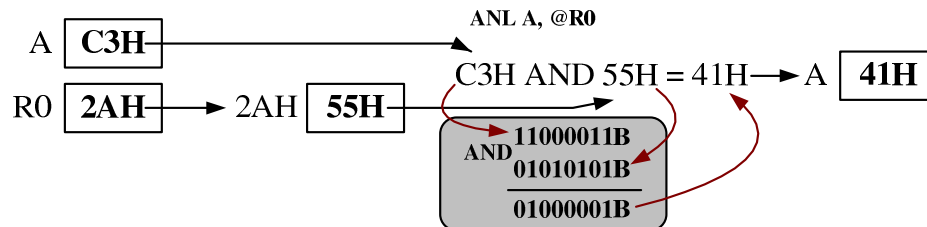
Sau khi thực thi lệnh **ANL A, R0** thì: $(A)=02H$



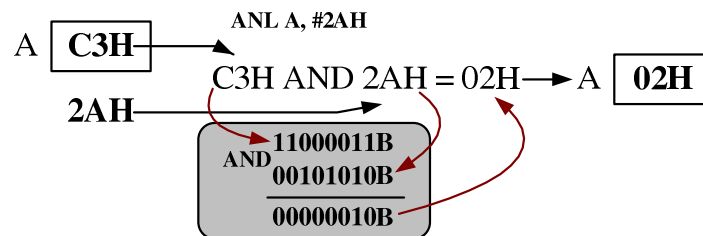
Sau khi thực thi lệnh **ANL A, B0H** hay **ANL A, P3** thì: (A)=41H



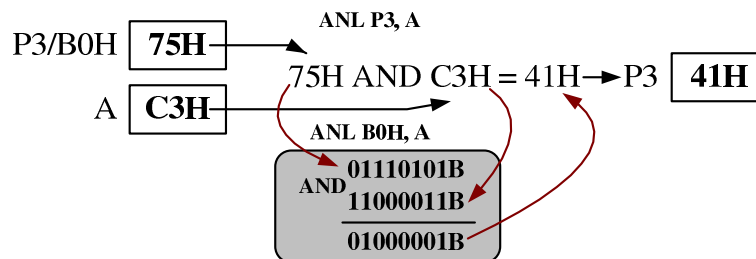
Sau khi thực thi lệnh **ANL A, @R0** thì: (A)=41H



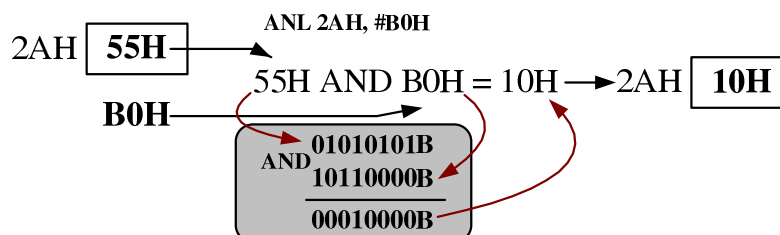
Sau khi thực thi lệnh **ANL A, #2AH** thì: (A)=02H



Sau khi thực thi lệnh **ANL B0H, A** hay **ANL P3, A** thì: (P3)=41H



Sau khi thực thi lệnh **ANL 2AH, #B0H** thì: (2AH)=10H



2.2. ORL <dest-byte>, <src-byte>

- *Chức năng:* OR logic hai toán hạng (*Logical-OR*).
- *Mô tả:* ORL thực hiện phép toán OR từng bit giữa hai toán hạng được chỉ ra trong lệnh và lưu kết quả vào toán hạng đích (***dest-byte***). *Các cờ không bị ảnh hưởng.*
- *Lưu ý:* Khi lệnh này được dùng để sửa đổi một port xuất, giá trị được dùng làm dữ liệu ban đầu của port được đọc từ bộ chốt dữ liệu xuất, không phải từ các chân port.
- *Các dạng lệnh:*

ORL A, Rn

Số byte	1
Số chu kỳ	1
Mã đối tượng	01001rrr
Hoạt động	$(A) \leftarrow (A) \text{ OR } (Rn)$

ORL A, direct

Số byte	2
Số chu kỳ	1
Mã đối tượng	01000101 aaaaaaaaa
Hoạt động	$(A) \leftarrow (A) \text{ OR } (\text{direct})$

ORL A, @Ri

Số byte	1
Số chu kỳ	1
Mã đối tượng	0100011i
Hoạt động	$(A) \leftarrow (A) \text{ OR } ((Ri))$

ORL A, #data

Số byte	2
Số chu kỳ	1
Mã đối tượng	01000100 dddddddd
Hoạt động	$(A) \leftarrow (A) \text{ OR } \#data$

ORL direct, A

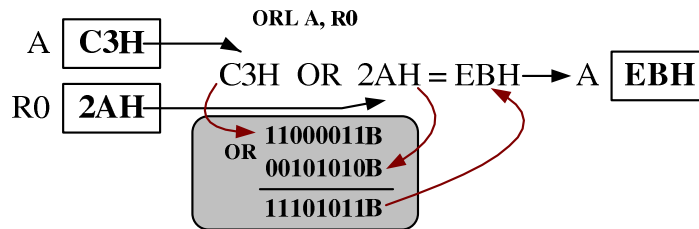
Số byte	2
Số chu kỳ	1
Mã đối tượng	01000010 aaaaaaaaa
Hoạt động	$(\text{direct}) \leftarrow (\text{direct}) \text{ OR } (A)$

ORL direct, #data

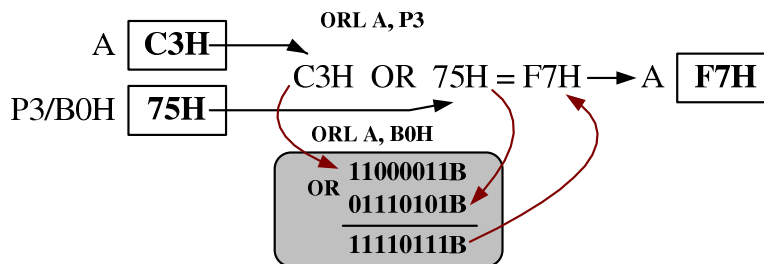
Số byte	3
Số chu kỳ	2
Mã đối tượng	01000011 aaaaaaaaa dddddddd
Hoạt động	$(\text{direct}) \leftarrow (\text{direct}) \text{ OR } \#data$

- Ví dụ: Cho biết trước (A)=C3H, (R0)=2AH, (P3)=(B0)=75H, (2AH)=55H.

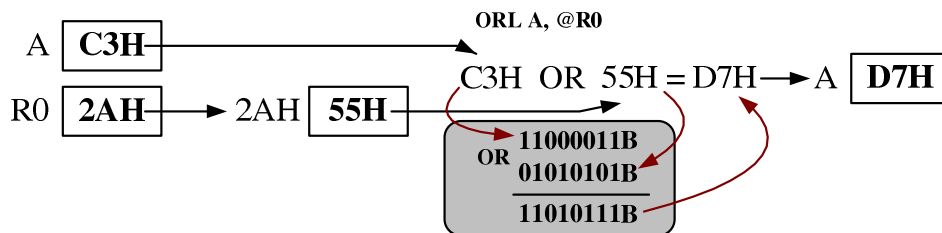
Sau khi thực thi lệnh **ORL A, R0** thì: (A)=EBH



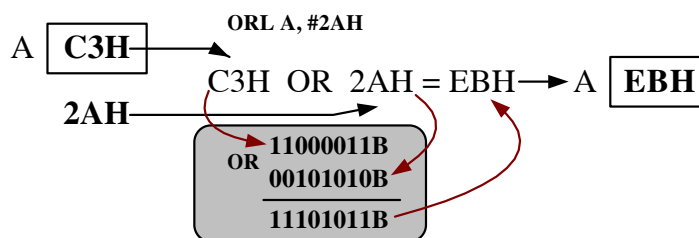
Sau khi thực thi lệnh **ORL A, B0H** hay **ORL A, P3** thì: (A)=F7H



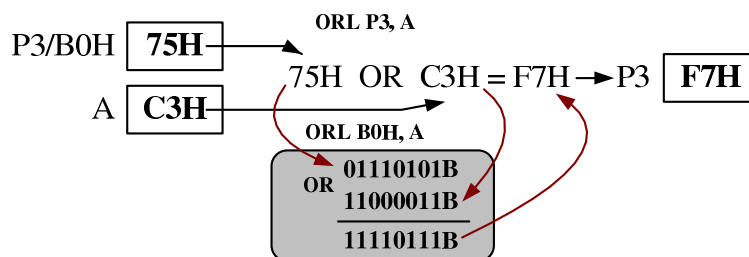
Sau khi thực thi lệnh **ORL A, @R0** thì: (A)=D7H



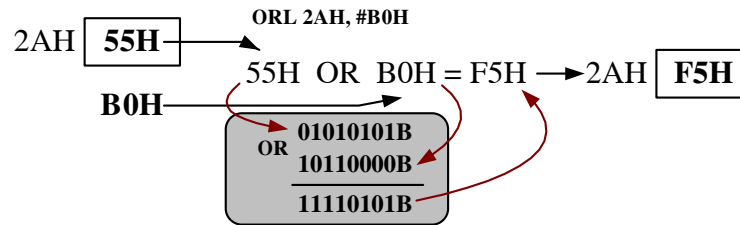
Sau khi thực thi lệnh **ORL A, #2AH** thì: (A)=EBH



Sau khi thực thi lệnh **ORL B0H, A** hay **ORL P3, A** thì: (P3)=F7H



Sau khi thực thi lệnh **ORL 2AH, #B0H** thì: (2AH)=F5H



2.3. XRL <dest-byte>, <src-byte>

- **Chức năng:** XOR logic hai toán hạng (*Logical Exclusive-OR*).
- **Mô tả:** XRL thực hiện phép toán XOR từng bit giữa hai toán hạng được chỉ ra trong lệnh và lưu kết quả vào toán hạng đích (*dest-byte*). Các cờ không bị ảnh hưởng.
- **Lưu ý:** Khi lệnh này được dùng để sửa đổi một port xuất, giá trị được dùng làm dữ liệu ban đầu của port được đọc từ bộ chốt dữ liệu xuất, không phải từ các chân port.
- **Các dạng lệnh:**

XRL A, Rn

Số byte	1
Số chu kỳ	1
Mã đối tượng	01101rrr
Hoạt động	$(A) \leftarrow (A) \oplus (Rn)$

XRL A, direct

Số byte	2
Số chu kỳ	1
Mã đối tượng	01100101 aaaaaaaa
Hoạt động	$(A) \leftarrow (A) \oplus (\text{direct})$

XRL A, @Ri

Số byte	1
Số chu kỳ	1
Mã đối tượng	0110011i
Hoạt động	$(A) \leftarrow (A) \oplus ((Ri))$

XRL A, #data

Số byte	2
Số chu kỳ	1
Mã đối tượng	01100100 dddddddd
Hoạt động	$(A) \leftarrow (A) \oplus \#data$

XRL direct, A

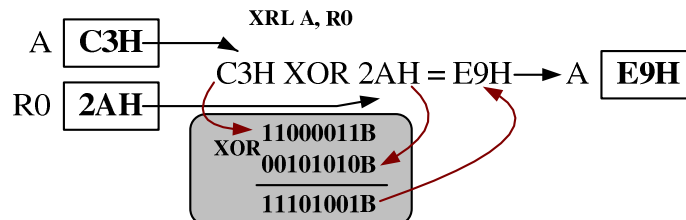
Số byte	2
Số chu kỳ	1
Mã đối tượng	01100010 aaaaaaaa
Hoạt động	$(\text{direct}) \leftarrow (\text{direct}) \oplus (A)$

XRL direct, #data

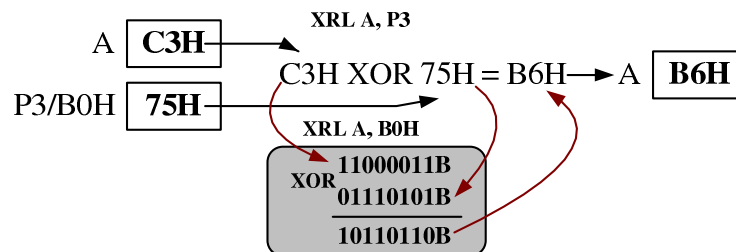
Số byte 3
 Số chu kỳ 2
 Mã đối tượng 01100011 aaaaaaaa dddddddd
 Hoạt động (direct) $\leftarrow$ (direct) $\oplus$ #data

- Ví dụ: Cho biết trước (A)=C3H, (R0)=2AH, (P3)=(B0)=75H, (2AH)=55H.

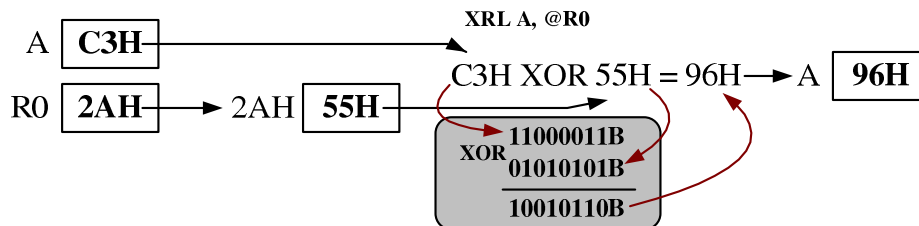
Sau khi thực thi lệnh **XRL A, R0** thì: (A)=E9H



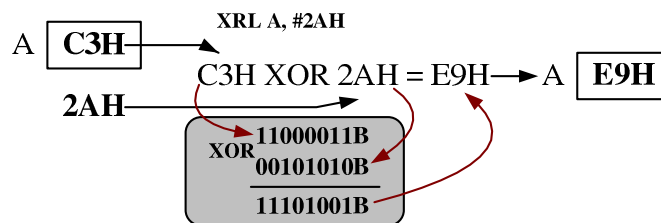
Sau khi thực thi lệnh **XRL A, B0H** hay **XRL A, P3** thì: (A)=B6H



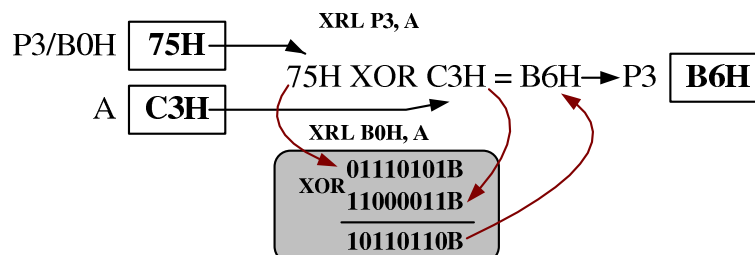
Sau khi thực thi lệnh **XRL A, @R0** thì: (A)=96H



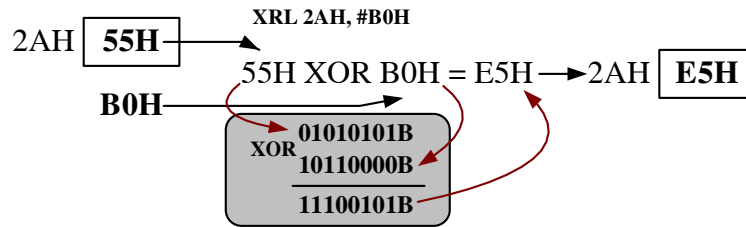
Sau khi thực thi lệnh **XRL A, #2AH** thì: (A)=E9H



Sau khi thực thi lệnh **XRL B0H, A** hay **XRL P3, A** thì: (P3)=B6H



Sau khi thực thi lệnh **XRL 2AH, #B0H** thì: (2AH)=E5H



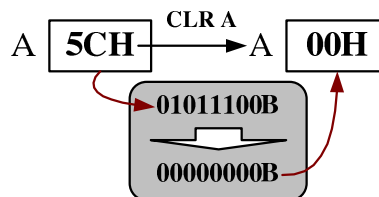
2.4. CLR A

- **Chức năng:** Xóa thanh ghi A (Clear Acc).
- **Mô tả:** Thanh ghi A bị xóa (tất cả các bit đều bằng 0). Các cờ không bị ảnh hưởng.
- **Các dạng lệnh:**

Số byte	1
Số chu kỳ	1
Mã đối tượng	11100100
Hoạt động	(A) ← 0

- **Ví dụ:** Cho biết trước (A)=5CH.

Sau khi thực thi lệnh **CLR A** thì: (A)=00H



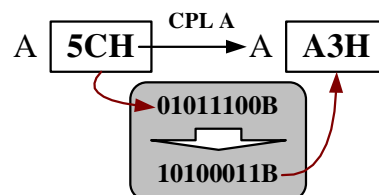
2.5. CPL A

- **Chức năng:** Lấy bù nội dung thanh ghi A (Complement Acc).
- **Mô tả:** Mỗi một bit của thanh ghi A được lấy bù logic (bù 1: các bit 1 được thì đổi thành bit 0 và các bit 0 được thì đổi thành bit 1). Các cờ không bị ảnh hưởng.

Số byte	1
Số chu kỳ	1
Mã đối tượng	11110100
Hoạt động	(A) ← NOT(A)

- **Ví dụ:** Cho biết trước (A)=5CH.

Sau khi thực thi lệnh **CPL A** thì: (A)=A3H



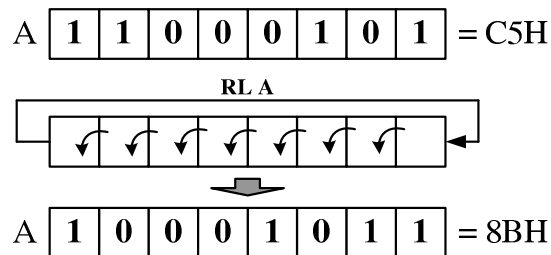
2.6. RL A

- *Chức năng:* Quay trái thanh ghi A (*Rotate Acc Left*).
- *Mô tả:* 8 bit trong thanh ghi A được quay trái 1 bit. Bit 7 được quay đến vị trí của bit 0. Các cờ không bị ảnh hưởng.

Số byte	1
Số chu kỳ	1
Mã đối tượng	00100011
Hoạt động	$(A_{n+1}) \leftarrow (A_n), n = 0 - 6$ $(A_0) \leftarrow A_7$

- *Ví dụ:* Cho biết trước (A)=C5H.

Sau khi thực thi lệnh **RL A** thì: (A)=8BH



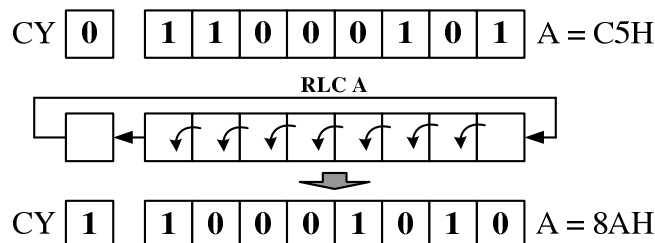
2.7. RLC A

- *Chức năng:* Quay trái thanh ghi A cùng với cờ nhớ.
- *Mô tả:* 8 bit trong thanh ghi A và cờ nhớ cùng được quay trái 1 bit. Bit 7 được di chuyển đến cờ CY và trạng thái ban đầu của cờ CY được đưa đến vị trí của bit 0. Các cờ khác không bị ảnh hưởng.

Số byte	1
Số chu kỳ	1
Mã đối tượng	00110011
Hoạt động	$(A_{n+1}) \leftarrow (A_n), n = 0 - 6$ $(A_0) \leftarrow (C), (C) \leftarrow A_7$

- *Ví dụ:* Cho biết trước (A)=C5H và cờ CY=0.

Sau khi thực thi lệnh **RLC A** thì: (A)=8AH và cờ CY=1



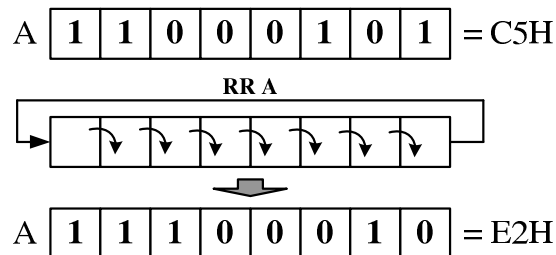
2.8. RR A

- *Chức năng:* Quay phải thanh ghi A (*Rotate Acc Right*).
- *Mô tả:* 8 bit trong thanh ghi A được quay phải 1 bit. Bit 0 được quay đến vị trí của bit 7. Các cờ không bị ảnh hưởng.

Số byte	1
Số chu kỳ	1
Mã đối tượng	00000011
Hoạt động	$(A_n) \leftarrow (A_{n+1}), n = 0 - 6$ $(A_7) \leftarrow A_0$

- *Ví dụ:* Cho biết trước (A)=C5H.

Sau khi thực thi lệnh **RR A** thì: (A)=E2H

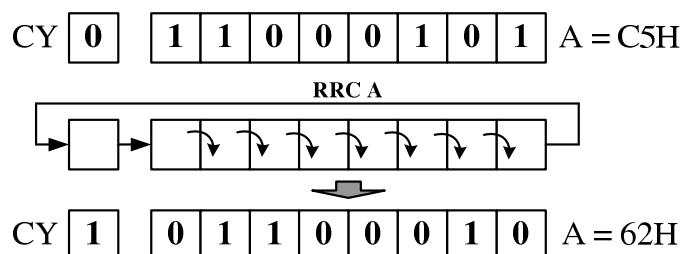
**2.9. RRC A**

- *Chức năng:* Quay phải thanh ghi A cùng với cờ nhớ.
- *Mô tả:* 8 bit trong thanh ghi A và cờ nhớ cùng 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. Các cờ khác không bị ảnh hưởng.

Số byte	1
Số chu kỳ	1
Mã đối tượng	00010011
Hoạt động	$(A_n) \leftarrow (A_{n+1}), n = 0 - 6$ $(A_7) \leftarrow (C)$ $(C) \leftarrow A_0$

- *Ví dụ:* Cho biết trước (A)=C5H và cờ CY=0.

Sau khi thực thi lệnh **RRC A** thì: (A)=62H và cờ CY=1



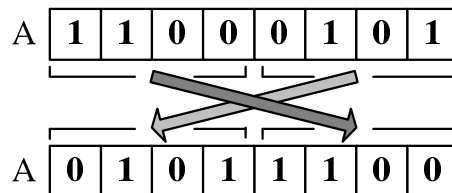
2.10. SWAP A

- **Chức năng:** Trao đổi nội dung hai nửa thấp và cao của thanh ghi A (*Swap nibble*).
- **Mô tả:** SWAP A trao đổi nội dung hai nửa thấp và cao của thanh ghi A (*các bit từ 3 đến 0 $\Leftrightarrow$ các bit từ 7 đến 4*). Thao tác này còn có thể được hiểu như là quay thanh ghi A đi 4 bit. *Các cờ không bị ảnh hưởng.*

Số byte	1
Số chu kỳ	1
Mã đối tượng	11000100
Hoạt động	$(A3 - A0) \leftrightarrow (A7 - A4)$

- **Ví dụ:** Cho biết trước (A)=C5H.

Sau khi thực thi lệnh **SWAP A** thì: (A)=5CH



- **Lưu ý:** Lệnh này có thể được dùng để chuyển đổi giá trị nhị phân trong thanh ghi A (*giá trị này nhỏ hơn 100*) thành số BCD như sau:

MOV	B, #10	;Chia giá trị cho 10 để tách ra
DIV	AB	; (A)=digit chục – (B)=digit đơn vị.
SWAP	A	;Đưa digit chục lên nửa cao của ACC.
ADD	A, B	;Thêm digit đơn vị vào nửa thấp.

3. Nhóm lệnh di chuyển dữ liệu:**3.1. MOV <dest-byte>, <src-byte>**

- **Chức năng:** Di chuyển nội dung của toán hạng nguồn (*src-byte*) đến toán hạng đích (*dest-byte*).
- **Mô tả:** 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.*
- **Các dạng lệnh:**

MOV A, Rn

Số byte	1
Số chu kỳ	1
Mã đối tượng	11101rrr
Hoạt động	$(A) \leftarrow (Rn)$

MOV A, direct

Số byte	2
Số chu kỳ	1
Mã đối tượng	11100101 aaaaaaaa
Hoạt động	$(A) \leftarrow (\text{direct})$

Lưu ý: MOV A, ACC là lệnh không hợp lệ.

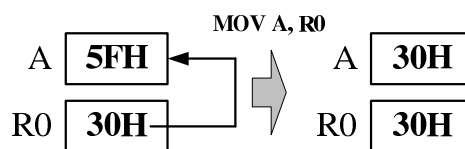
MOV A, @Ri		
Số byte	1	
Số chu kỳ	1	
Mã đối tượng	1110011i	
Hoạt động	$(A) \leftarrow ((Ri))$	
MOV A, #data		
Số byte	2	
Số chu kỳ	1	
Mã đối tượng	01110100	dddddddd
Hoạt động	$(A) \leftarrow \#data$	
MOV Rn, A		
Số byte	1	
Số chu kỳ	1	
Mã đối tượng	11111rrr	
Hoạt động	$(Rn) \leftarrow (A)$	
MOV Rn, direct		
Số byte	2	
Số chu kỳ	2	
Mã đối tượng	10101rrr	aaaaaaaa
Hoạt động	$(Rn) \leftarrow (direct)$	
MOV Rn, #data		
Số byte	2	
Số chu kỳ	1	
Mã đối tượng	01111rrr	dddddddd
Hoạt động	$(Rn) \leftarrow \#data$	
MOV direct, A		
Số byte	2	
Số chu kỳ	1	
Mã đối tượng	11110101	aaaaaaaa
Hoạt động	$(direct) \leftarrow (A)$	
MOV direct, Rn		
Số byte	2	
Số chu kỳ	2	
Mã đối tượng	10001rrr	aaaaaaaa
Hoạt động	$(direct) \leftarrow (Rn)$	
MOV direct, direct		
Số byte	3	
Số chu kỳ	2	
Mã đối tượng	10000101	aaaaaaaa aaaaaaaa
Hoạt động	$(direct) \leftarrow (direct)$	

Lưu ý: byte 2 chứa địa chỉ nguồn, byte 3 chứa địa chỉ đích.

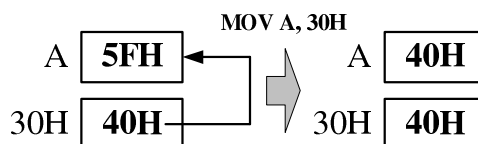
MOV direct, @Ri			
Số byte	2		
Số chu kỳ	2		
Mã đối tượng	1000011i		aaaaaaaa
Hoạt động	(direct) ← ((Ri))		
MOV direct, #data			
Số byte	3		
Số chu kỳ	2		
Mã đối tượng	01110101	aaaaaaaa	dddddddd
Hoạt động	(direct) ← #data		
MOV @Ri, A			
Số byte	1		
Số chu kỳ	1		
Mã đối tượng	1111011i		
Hoạt động	((Ri)) ← (A)		
MOV @Ri, direct			
Số byte	2		
Số chu kỳ	2		
Mã đối tượng	1010011i	aaaaaaaa	
Hoạt động	((Ri)) ← (direct)		
MOV @Ri, #data			
Số byte	2		
Số chu kỳ	1		
Mã đối tượng	0111011i	dddddddd	
Hoạt động	((Ri)) ← #data		

- Ví dụ: Cho biết trước (A)=5FH, (R0)=30H, (30H)=40H, (P1)=CAH.

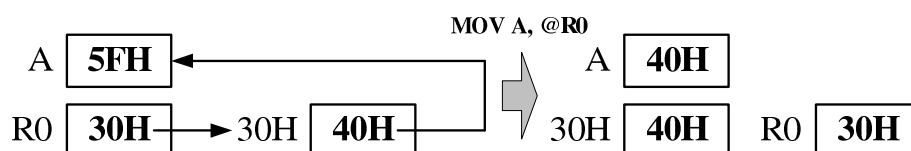
Sau khi thực thi lệnh **MOV A, R0** thì: (A)=30H, (R0)=30H



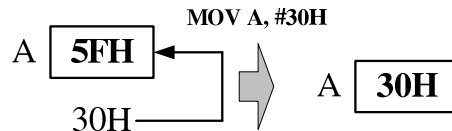
Sau khi thực thi lệnh **MOV A, 30H** thì: (A)=40H, (30H)=40H



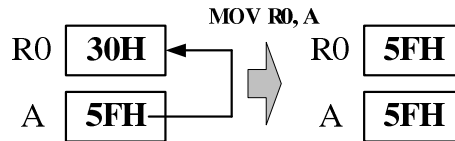
Sau khi thực thi lệnh **MOV A, @R0** thì: (A)=40H, (R0)=30H, (30H)=40H



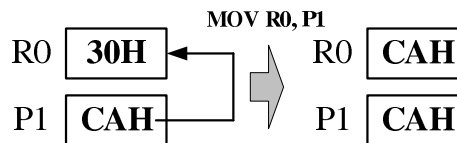
Sau khi thực thi lệnh **MOV A, #30H** thì: (A)=30H



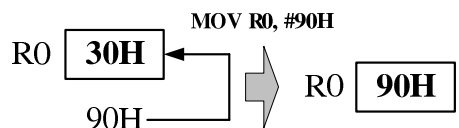
Sau khi thực thi lệnh **MOV R0, A** thì: (A)=5FH, (R0)=5FH



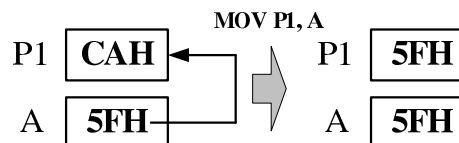
Sau khi thực thi lệnh **MOV R0, P1** thì: (R0)=CAH, (P1)=CAH



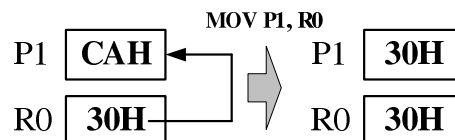
Sau khi thực thi lệnh **MOV R0, #90H** thì: (R0)=90H



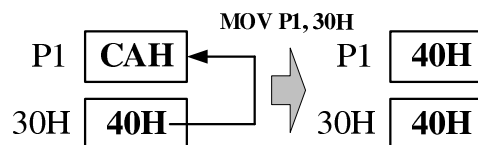
Sau khi thực thi lệnh **MOV P1, A** thì: (A)=5FH, (P1)=5FH



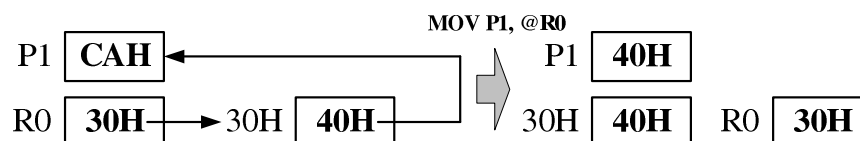
Sau khi thực thi lệnh **MOV P1, R0** thì: (R0)=30H, (P1)=30H



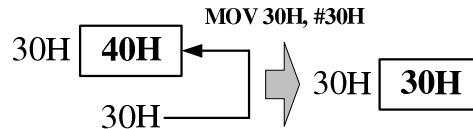
Sau khi thực thi lệnh **MOV P1, 30H** thì: (30H)=40H, (P1)=40H



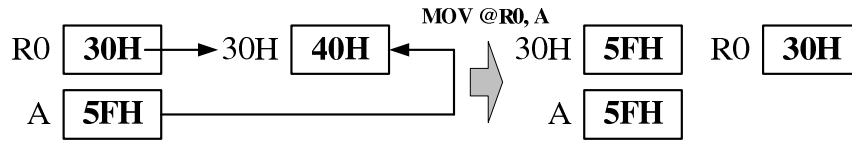
Sau khi thực thi lệnh **MOV P1, @R0** thì: (R0)=30H, (30H)=40H, (P1)=40H



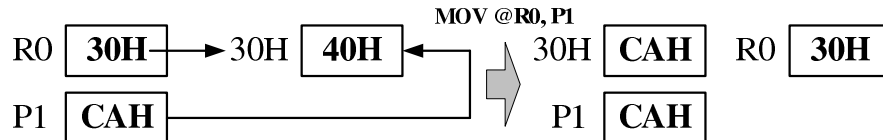
Sau khi thực thi lệnh **MOV 30H, #30H** thì: (30H)=30H



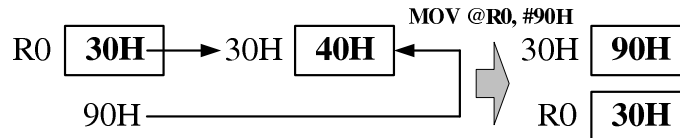
Sau khi thực thi lệnh **MOV @R0, A** thì: (A)=5FH, (30H)=5FH, (R0)=30H



Sau khi thực thi lệnh **MOV @R0, P1**: (30H)=CAH, (P1)=CAH, (R0)=30H



Sau khi thực thi lệnh **MOV @R0, #90H** thì: (30H)=90H, (R0)=30H



3.2. MOVC A, @A+ <base-reg>

- **Chức năng:** Di chuyển byte mã hoặc byte hằng số.
- **Mô tả:** MOVC nạp cho thanh ghi A byte mã hoặc byte hằng số từ bộ nhớ chương trình. Địa chỉ của byte được tìm nạp là tổng của giá trị 8 bit không dấu ban đầu chứa trong thanh ghi A với nội dung của thanh ghi nền 16 bit (*thanh ghi nền có thể là con trỏ dữ liệu hoặc PC*). Trong trường hợp sau, thanh ghi PC được tăng để chỉ đến địa chỉ của lệnh tiếp theo trước khi được cộng với nội dung của thanh ghi A, các thanh ghi nền không bị thay đổi. Phép cộng bit thứ 16 do số nhớ từ 8 bit thấp có thể truyền qua các bit cao. *Các cờ không bị ảnh hưởng.*
- **Các dạng lệnh:**

MOVC A, @A+DPTR

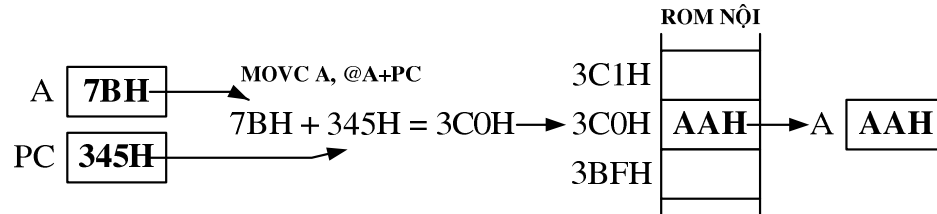
Số byte	1
Số chu kỳ	2
Mã đối tượng	10010011
Hoạt động	$(A) \leftarrow ((A)+(DPTR))$

MOVC A, @A+PC

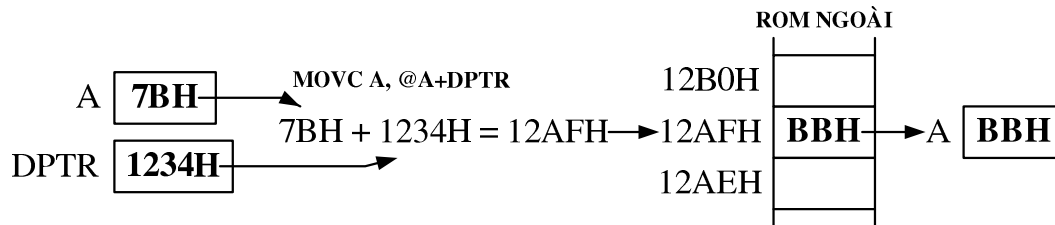
Số byte	1
Số chu kỳ	2
Mã đối tượng	10000011
Hoạt động	$(A) \leftarrow ((A)+(PC))$

- Ví dụ 1: Cho biết trước (A)=7BH, (PC)=345H, (DPTR)=1234H. Ô nhớ ROM nội (3C0H)=AAH. Ô nhớ ROM ngoài (12AFH)=BBH.

Sau khi thực thi lệnh **MOVC A, @A+PC** thì: (A)=AAH



Sau khi thực thi lệnh **MOVC A, @A+DPTR** thì: (A)=BBH



- Ví dụ 2: Cho chuỗi lệnh sau:

MOVC A, @A+PC ; (@A+PC)=(A+[PC])
SJMP \$; Độ dài lệnh là 2 byte.

DULIEU:

DB 66H,77H,88H
DB 99H,0AAH
DB 0BBH

Sau khi thực thi chuỗi lệnh thì:

- ⇒ (A) = 66H nếu trước khi thực thi lệnh ta có (A) = 02H.
- ⇒ (A) = 77H nếu trước khi thực thi lệnh ta có (A) = 03H.
- ⇒ (A) = 88H nếu trước khi thực thi lệnh ta có (A) = 04H.
- ⇒ (A) = 99H nếu trước khi thực thi lệnh ta có (A) = 05H.
- ⇒ (A) = AAH nếu trước khi thực thi lệnh ta có (A) = 06H.
- ⇒ (A) = BBH nếu trước khi thực thi lệnh ta có (A) = 07H.

- Ví dụ 3: Cho chuỗi lệnh sau:

MOV DPTR, #CODEDISP
MOVC A, @A+DPTR ; (@A+DPTR)=(A+[DPTR])
SJMP \$; Độ dài lệnh là 2 byte.

CODEDISP:

DB 48H,5AH,6BH,0A9H,0F5H,90H

Sau khi thực thi chuỗi lệnh thì:

- ⇒ (A) = 48H nếu trước khi thực thi lệnh ta có (A) = 00H.
- ⇒ (A) = 5AH nếu trước khi thực thi lệnh ta có (A) = 01H.
- ⇒ (A) = 6BH nếu trước khi thực thi lệnh ta có (A) = 02H.
- ⇒ (A) = A9H nếu trước khi thực thi lệnh ta có (A) = 03H.
- ⇒ (A) = F5H nếu trước khi thực thi lệnh ta có (A) = 04H.
- ⇒ (A) = 90H nếu trước khi thực thi lệnh ta có (A) = 05H.

3.3. MOVX <dest-byte>, <src-byte>

- **Chức năng:** Di chuyển ở bộ nhớ ngoài (*Move External*).
- **Mô tả:** MOVX chuyển dữ liệu giữa thanh ghi A với nội dung của một byte trong bộ nhớ dữ liệu ngoài (*ta dùng ký hiệu X nói tiếp với MOV*). Các lệnh này được chia làm 2 loại, hai loại này khác nhau ở chỗ chúng cung cấp **địa chỉ gián tiếp 8 bit hay 16 bit cho bộ nhớ dữ liệu ngoài** có dung lượng **256 byte hay 64 KB**.

Với **loại thứ nhất**, sử dụng thanh ghi R0 hoặc R1 để lưu giữ địa chỉ của dữ liệu cần truy xuất thuộc RAM ngoài. Loại này dùng trong trường hợp bộ nhớ RAM có dung lượng nhỏ (*tối đa là 256 byte*). **Port 1 & Port 2 là port xuất/nhập dữ liệu.**

Với **loại thứ hai**, sử dụng thanh ghi DPTR để lưu giữ địa chỉ của dữ liệu cần truy xuất thuộc RAM ngoài. Loại này dùng trong trường hợp bộ nhớ RAM có dung lượng lớn (*tối đa là 64 KB*). **Port 1 là port xuất/nhập dữ liệu.**

Trong nhiều tình huống ta có thể trộn hai loại trên của lệnh MOVX. Một dãy RAM lớn với các đường địa chỉ cao được điều khiển bởi P2 có thể được định địa chỉ thông qua con trỏ dữ liệu hoặc với mã để xuất ra các bit địa chỉ cao đến P2 được tiếp theo bởi một lệnh MOVX sử dụng R0 hoặc R1.

- **Các dạng lệnh:**

MOVX A, @Ri

Số byte	1
Số chu kỳ	2
Mã đối tượng	11100011i
Hoạt động	(A) ← ((Ri))

MOVX A, @DPTR

Số byte	1
Số chu kỳ	2
Mã đối tượng	11100000
Hoạt động	(A) ← ((DPTR))

MOVX @Ri, A

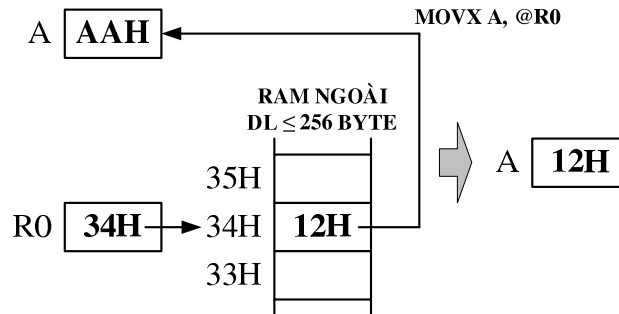
Số byte	1
Số chu kỳ	2
Mã đối tượng	11110011
Hoạt động	((Ri)) ← (A)

MOVX @DPTR, A

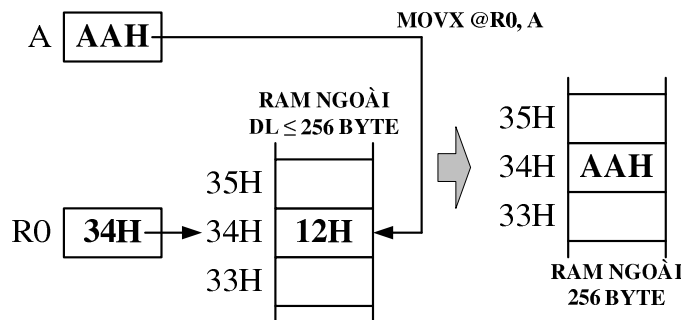
Số byte	1
Số chu kỳ	2
Mã đối tượng	11110000
Hoạt động	((DPTR)) ← (A)

- Ví dụ: Cho biết trước (A)=AAH, (DPTR)=1234H, (R0)=34H. Ô nhớ RAM ngoài 256 byte: (34H)=12H & 64 KB: (1234H)=7FH

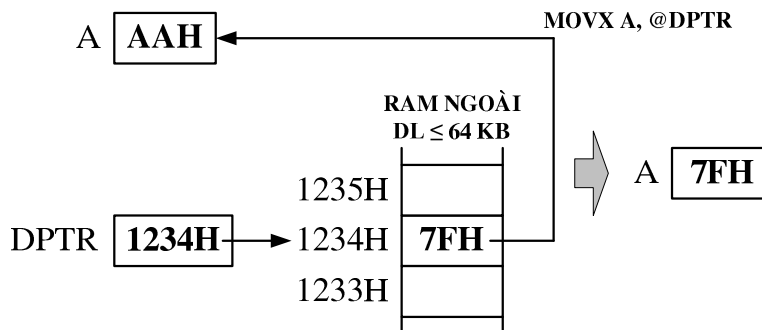
Sau khi thực thi lệnh **MOVX A, @R0** thì: (A)=12H



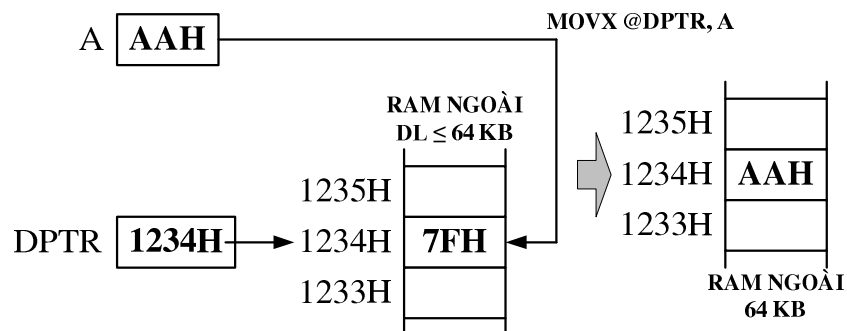
Sau khi thực thi lệnh **MOVX @R0, A** thì: ô nhớ RAM ngoài (34H)=AAH



Sau khi thực thi lệnh **MOVX A, @DPTR** thì: (A)=7FH



Sau khi thực thi lệnh **MOVX @DPTR, A** thì: (1234H)=AAH



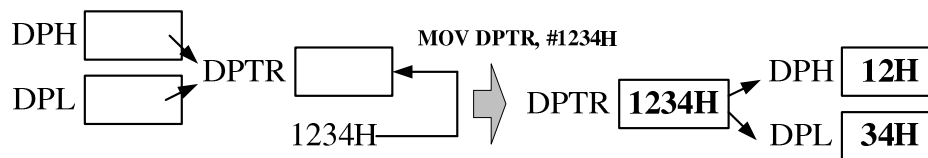
3.4 MOV DPTR, #data16

- *Chức năng:* Nạp hằng số 16-bit cho con trỏ dữ liệu DPTR.
- *Mô tả:* Con trỏ dữ liệu được nạp bởi hằng số 16-bit chỉ ra trong lệnh. Hằng số 16-bit được đặt ở byte 2 và byte 3 của lệnh. Byte 2 là byte cao được nạp cho DPH còn byte 3 là byte thấp được nạp cho DPL. Các cờ *không bị ảnh hưởng*.

Số byte	3
Số chu kỳ	2
Mã đối tượng	10010000 dddddddd dddddddd
Hoạt động	(DPTR) ← #data16

- *Ví dụ:*

Sau khi thực thi lệnh **MOV DPTR, #1234H** thì:

**3.5. PUSH direct**

- *Chức năng:* Cất vào ngăn xếp (*stack*).
- *Mô tả:* Con trỏ *stack* được tăng bởi 1. Nội dung của toán hạng được chỉ ra trong lệnh sau đó được sao chép vào RAM nội tại địa chỉ được trỏ đến bởi con trỏ stack. Các cờ *không bị ảnh hưởng*.

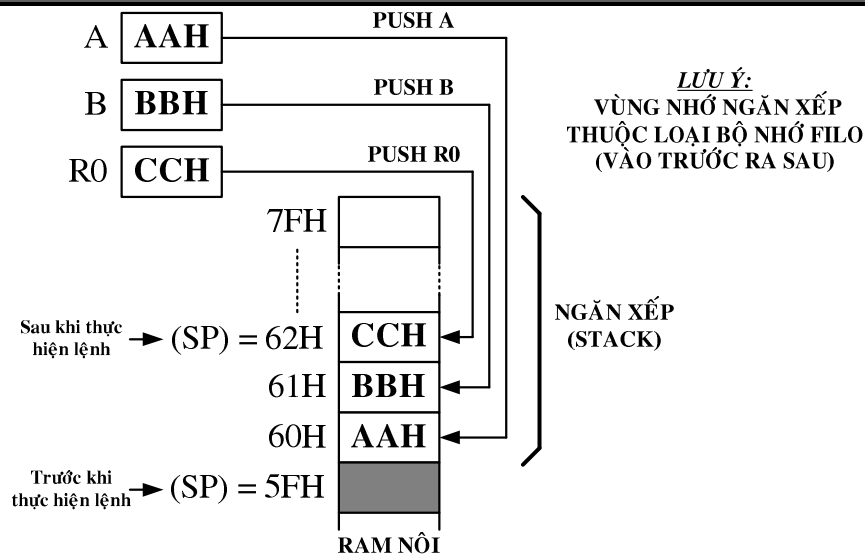
Số byte	2
Số chu kỳ	2
Mã đối tượng	11000000 aaaaaaaa
Hoạt động	(SP) ← (SP) + 1 ((SP)) ← (direct)

- *Ví dụ:* Cất tạm thời nội dung của thanh ghi A, B và R0 vào ngăn xếp. Cho biết trước (A)=AAH, (B)=BBH, (R0)=CCH, (SP)=5FH.

Sau khi thực thi chuỗi lệnh:

PUSH A
PUSH B
PUSH 00H

Thì: (SP)=62H, (60H)=AAH, (61H)=BBH, (62H)=CCH



3.6. POP direct

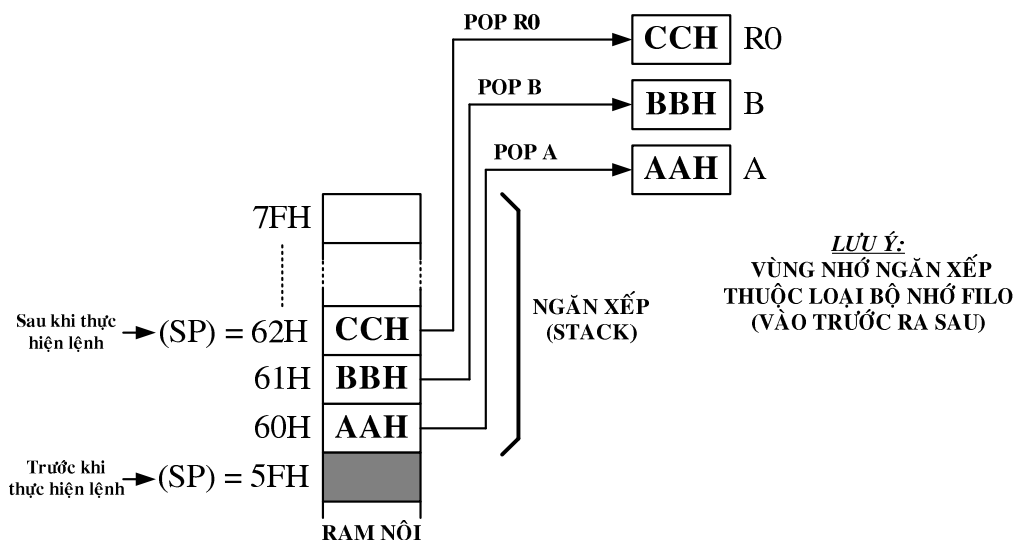
- Chức năng:** Lấy ra từ ngăn xếp (*stack*).
- Mô tả:** Nội dung của vùng RAM nội được định địa chỉ bởi con trỏ stack SP được đọc và nội dung con trỏ stack được giảm bởi 1. Giá trị đọc được sau đó được chuyển đến byte được định địa chỉ trực tiếp chỉ ra trong lệnh. Các cờ không bị ảnh hưởng.

Số byte	2
Số chu kỳ	2
Mã đối tượng	11010000 aaaaaaaa
Hoạt động	(direct) ← ((SP)) (SP) ← (SP) - 1

- Ví dụ:** Lấy lại nội dung của thanh ghi A, B và R0 đã cất vào ngăn xếp lúc đầu (ví dụ trên).

Sau khi thực thi chuỗi lệnh: **POP 00H**
POP B
POP A

Thì: (SP)=5FH, (R0)=CCH, (B)=BBH, (A)=AAH



3.7. XCH A, <byte>

- *Chức năng:* Trao đổi nội dung của thanh ghi A với nội dung của một byte
- *Mô tả:* XCH nạp cho thanh ghi A nội dung của byte 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.
- *Các dạng lệnh:*

XCH A, Rn

Số byte	1
Số chu kỳ	1
Mã đối tượng	11001rrr
Hoạt động	(A) $\leftrightarrow$ (Rn)

XCH A, direct

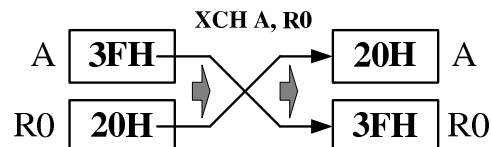
Số byte	2
Số chu kỳ	1
Mã đối tượng	11000101 aaaaaaaa
Hoạt động	(A) $\leftrightarrow$ (direct)

XCH A, @Ri

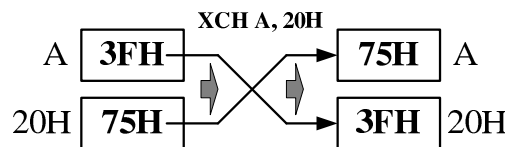
Số byte	1
Số chu kỳ	1
Mã đối tượng	1100011i
Hoạt động	(A) $\leftrightarrow$ ((Ri))

- *Ví dụ:* Cho biết trước (A)=3FH, (R0)=20H, (20H)=75H.

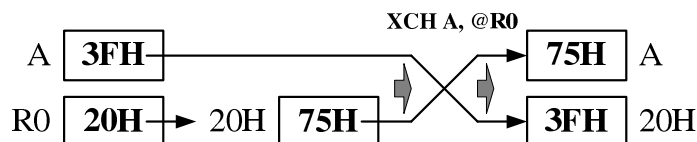
Sau khi thực thi lệnh **XCH A, R0** thì: (A)=20H, (R0)=3FH



Sau khi thực thi lệnh **XCH A, 20H** thì: (A)=75H, (20H)=3FH



Sau khi thực thi lệnh **XCH A, @R0** thì: (A)=75H, (20H)=3FH, (R0)=20H



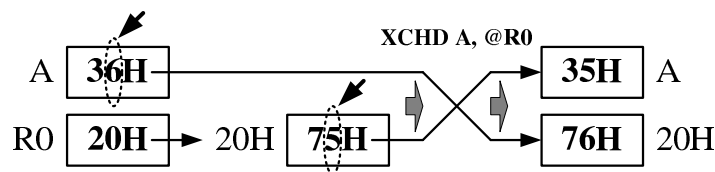
3.8. XCHD A, @Ri

- **Chức năng:** Trao đổi digit (*Exchange Digit*).
- **Mô tả:** XCHD trao đổi nội dung nửa thấp của thanh ghi A (biểu diễn một digit số hex hoặc BCD) với nội dung nửa thấp của một byte trong RAM nội, byte này được định địa chỉ gián tiếp bởi thanh ghi chỉ ra trong lệnh. Nửa cao của các thanh ghi vừa nêu trên không bị ảnh hưởng và các cờ cũng không bị ảnh hưởng.

Số byte	1
Số chu kỳ	1
Mã đối tượng	110101i
Hoạt động	$(A3 - A0) \leftrightarrow (Ri3 - Ri0)$

- **Ví dụ:** Cho biết trước $(A)=36H$, $(R0)=20H$, $(20H)=75H$.

Sau khi thực thi lệnh **XCHD A, @R0** thì: $(A)=35H$, $(20H)=76H$

**4. Nhóm lệnh xử lý bit:****4.1. CLR bit**

- **Chức năng:** Xóa bit.
- **Mô tả:** Bit được chỉ ra trong lệnh được xóa. Các cờ không bị ảnh hưởng. CLR có thể thao tác trên cờ nhớ và trên một bit bất kỳ được định địa chỉ bit.
- **Các dạng lệnh:**

CLR C

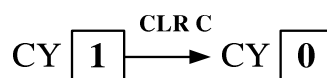
Số byte	1
Số chu kỳ	1
Mã đối tượng	11000011
Hoạt động	$(C) \leftarrow 0$

CLR bit

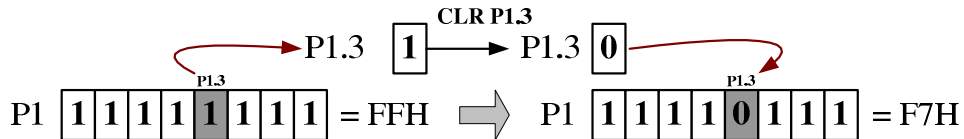
Số byte	2
Số chu kỳ	1
Mã đối tượng	11000010 bbbbbbbb
Hoạt động	$(bit) \leftarrow 0$

- **Ví dụ:** Cho biết trước cờ $CY=1$, $(P1)=FFH$.

Sau khi thực thi lệnh **CLR C** thì: cờ $CY=0$



Sau khi thực thi lệnh **CLR P1.3** thì: P1.3=0 tức làm (P1)=F7H



4.2. CPL bit

- **Chức năng:** Lấy bù bit (Complex).
- **Mô tả:** Bit được chỉ ra trong lệnh được lấy bù. Một bit có giá trị 1 được đổi thành 0 và ngược lại. Các cờ không bị ảnh hưởng. CPL có thể thao tác trên cờ nhớ và trên một bit bất kỳ được định địa chỉ bit.
- **Lưu ý:** Khi lệnh này được dùng để làm thay đổi giá trị của một chân xuất (bất kỳ chân nào của một port) thì giá trị được dùng làm dữ liệu ban đầu của chân được lấy từ bộ chốt dữ liệu xuất, không phải được lấy từ chân nhập.
- **Các dạng lệnh:**

CPL C

Số byte	1
Số chu kỳ	1
Mã đối tượng	10110011
Hoạt động	(C) ← NOT(C)

CPL bit

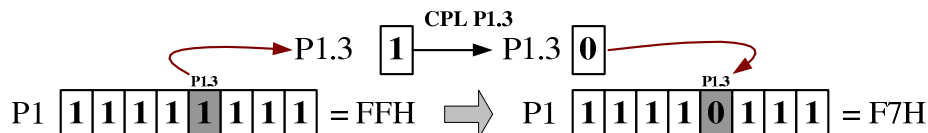
Số byte	2
Số chu kỳ	1
Mã đối tượng	10110010 bbbbbbbb
Hoạt động	(bit) ← NOT(bit)

- **Ví dụ:** Cho biết trước cờ CY=1, (P1)=FFH.

Sau khi thực thi lệnh **CPL C** thì: cờ CY=0



Sau khi thực thi lệnh **CPL P1.3** thì: P1.3=0 tức làm (P1)=F7H



4.3. SETB <bit>

- *Chức năng:* Set bit bằng 1 (Set Bit).
- *Mô tả:* SETB set bit được chỉ ra trong lệnh bằng 1. SETB có thể thao tác trên cờ nhớ hoặc các bit bất kỳ được định địa chỉ bit. Các cờ không bị ảnh hưởng.
- *Các dạng lệnh:*

SETB C

Số byte	1
Số chu kỳ	1
Mã đối tượng	11010011
Hoạt động	(C) ← 1

SETB bit

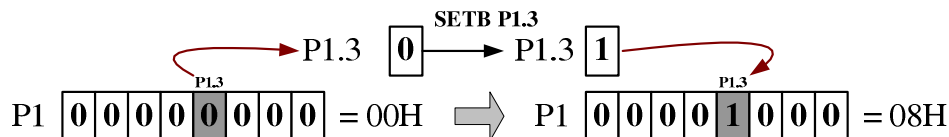
Số byte	2
Số chu kỳ	1
Mã đối tượng	11010010 bbbbbbbb
Hoạt động	(bit) ← 1

- *Ví dụ:* Cho biết trước cờ CY=0, (P1)=00H.

Sau khi thực thi lệnh **SETB C** thì: cờ CY=1



Sau khi thực thi lệnh **SETB P1.3** thì: P1.3=1 tức là (P1)=08H

**4.4. ANL C, <src-bit>**

- *Chức năng:* AND logic hai bit.
- *Mô tả:* Nếu giá trị của bit nguồn là 0 thì lệnh này sẽ xóa CY và ngược lại nếu giá trị của bit nguồn là 1 thì lệnh này giữ nguyên giá trị hiện hành của cờ CY. Dấu gạch chéo (/) đặt trước toán hạng trong chương trình hợp ngữ chỉ ra rằng bit nguồn được lấy bù trước khi AND với CY nhưng *giá trị của bit nguồn không bị thay đổi* bởi thao tác lấy bù này. Các cờ không bị ảnh hưởng.
- *Các dạng lệnh:*

ANL C, bit

Số byte	2
Số chu kỳ	2
Mã đối tượng	10000010 bbbbbbbb
Hoạt động	(C) ← (C) AND (bit)

ANL C, /bit

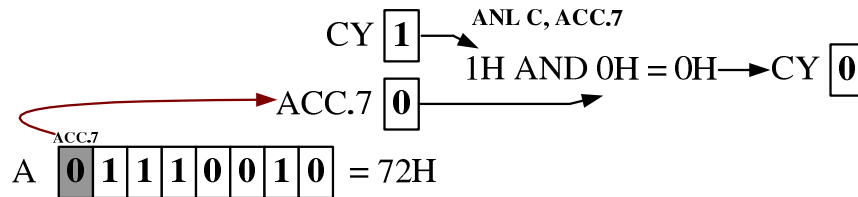
Số byte	2
Số chu kỳ	2
Mã đối tượng	10110000 bbbbbbbb

Hoạt động

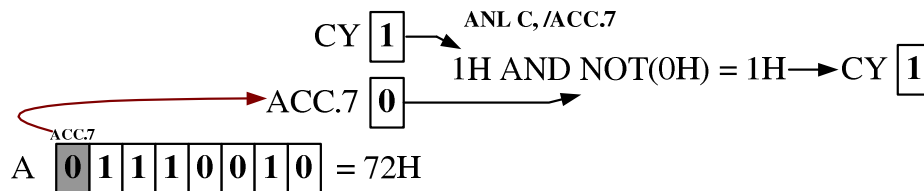
$(C) \leftarrow (C) \text{ AND NOT}(\text{bit})$

- Ví dụ: Cho biết trước (A)=72H, cờ CY=1.

Sau khi thực thi lệnh **ANL C, ACC.7** thì: cờ CY=0, (A)=72H



Sau khi thực thi lệnh **ANL C, /ACC.7** thì: cờ CY=1, (A)=72H



4.5. ORL C, <src-bit>

- Chức năng: OR logic hai bit.
- Mô tả: Nếu giá trị của bit nguồn là 1 thì phép toán sẽ *set* cờ CY=1 và ngược lại nếu giá trị của bit nguồn là 0 thì phép toán giữ nguyên giá trị hiện hành của cờ CY. Dấu gạch chéo / đặt trước toán hạng trong chương trình hợp ngữ chỉ ra rằng bit nguồn được lấy bù trước khi OR logic với cờ nhớ nhưng **giá trị của bit nguồn không bị thay đổi** bởi thao tác lấy bù. Các cờ không bị ảnh hưởng.
- Các dạng lệnh:

ORL C, bit

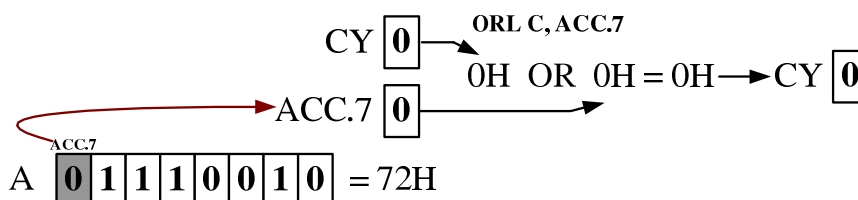
Số byte	2
Số chu kỳ	2
Mã đối tượng	01110010 bbbbbbbb
Hoạt động	$(C) \leftarrow (C) \text{ OR } (\text{bit})$

ORL C, /bit

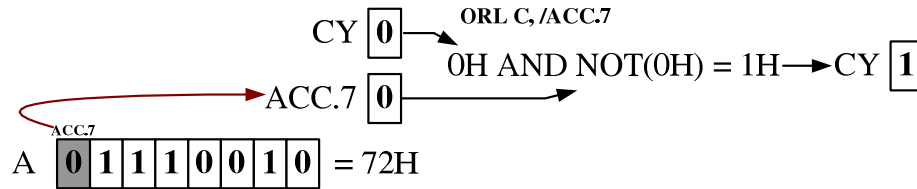
Số byte	2
Số chu kỳ	2
Mã đối tượng	10100000 bbbbbbbb
Hoạt động	$(C) \leftarrow (C) \text{ OR NOT}(\text{bit})$

- Ví dụ: Cho biết trước(A)=72H, cờ CY=0.

Sau khi thực thi lệnh **ORL C, ACC.7** thì: cờ CY=0, (A)=72H



Sau khi thực thi lệnh **ORL C, /ACC.7** thì: cờ CY=1, (A)=72H



- Lưu ý:** Các lệnh trên bao gồm lệnh **ANL** và **ORL** nhưng không tồn tại lệnh **XRL**. Cho nên nếu ta cần **XOR** hai bit, **BIT1** và **BIT2**, và kết quả được cất vào trong cờ nhớ (**CY**) thì ta sử dụng đoạn lệnh sau đây:

```

.....
MOV    C, BIT1                ;Nạp BIT1 vào cờ nhớ.
JNB    BIT2, SKIP             ;BIT2=0 thì C = BIT1.
CPL    C                      ;BIT2=1 thì C\ = BIT1\
SKIP:
.....                        ;BIT1 XOR BIT2
    
```

4.6. MOV <dest-bit>, <src-bit>

- Chức năng:** Di chuyển bit nguồn (**src-bit**) đến bit đích (**dest-bit**).
- Mô tả:** Nội dung của bit được chỉ ra bởi toán hạng thứ hai được sao chép vào vị trí được xác định bởi toán hạng thứ nhất. Một trong hai toán hạng phải là cờ nhớ và 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.*
- Các dạng lệnh:**

MOV C, bit

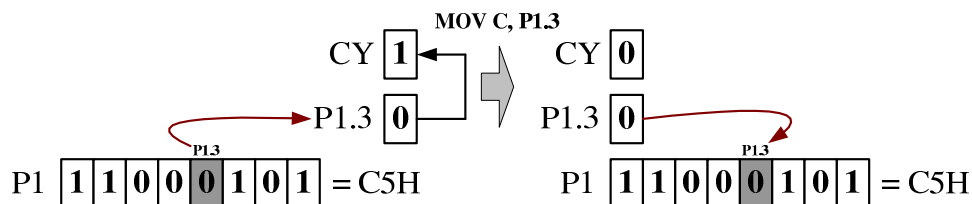
Số byte	2	
Số chu kỳ	1	
Mã đối tượng	10100010	bbbbbbbb
Hoạt động	(C) ← (bit)	

MOV bit, C

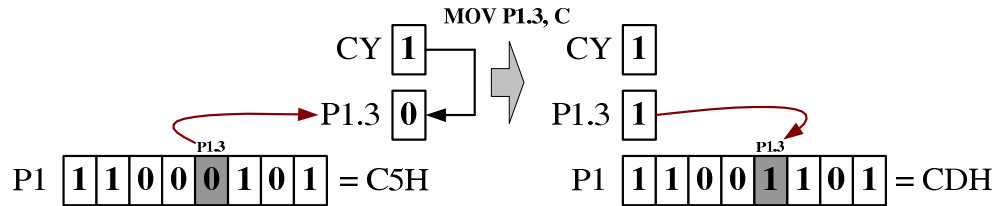
Số byte	2	
Số chu kỳ	2	
Mã đối tượng	10010010	bbbbbbbb
Hoạt động	(bit) ← (C)	

- Ví dụ:** Cho biết trước cờ CY=1, (P1)=C5H.

Sau khi thực thi lệnh **MOV C, P1.3** thì: cờ CY=0, (P1)=C5H



Sau khi thực thi lệnh **MOV P1.3, C** thì: cờ CY=1, (P1)=CDH

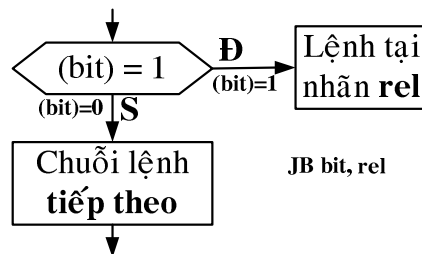


4.7. JB bit, rel

- **Chức năng:** Nhảy nếu bit bằng 1.
- **Mô tả:** Nếu bit chỉ ra trong lệnh bằng 1 thì nhảy đến địa chỉ được chỉ ra trong lệnh còn ngược lại thì tiếp tục với lệnh tiếp theo. Các cờ không bị ảnh hưởng, bit được kiểm tra sẽ không bị thay đổi.

Số byte	3
Số chu kỳ	2
Mã đối tượng	00100000 bbbbbbbb eeeeeeee
Hoạt động	(PC) ← (PC) + 3 IF (bit) = 1 THEN (PC) ← (PC) + byte_2

- **Lưu đồ:**



- **Lưu ý:** Tầm nhảy của lệnh **JB bit, rel** bị giới hạn ở khoảng cách nhảy từ -128 byte (*nhảy lui*) đến +127 byte (*nhảy tới*) kể từ lệnh kế tiếp theo sau lệnh nhảy có điều kiện này.

- **Ví dụ 1:** Cho biết trước (A)=56H, (P1)=CAH.

Sau khi thực thi chuỗi lệnh:

JB P1.2, AAA
JB ACC.2, BBB

thì chương trình được tiếp tục với lệnh tại nhãn BBB, (P1)=CAH và (A)=56H.

- **Ví dụ 2:** Cho chuỗi lệnh:

```
MOV    A, #0FFH
MOV    P0, #9CH          ; 9CH = 10011100B
JB      P0.7, AAA
MOV    A, #00H
```

AAA:

.....

Sau khi thực thi chuỗi lệnh thì (A)=FFH, (P0)=9CH (*chương trình thực hiện lệnh nhảy JB P0.7, AAA*).

- Ví dụ 3: Cho chuỗi lệnh:

```

MOV    A, #0FFH
MOV    P0, #9CH          ; 9CH = 10011100B
JB     P0.5, AAA
MOV    A, #00H
AAA:
.....

```

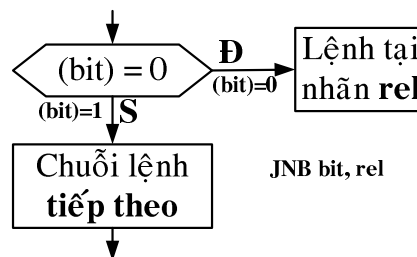
Sau khi thực thi chuỗi lệnh thì (A)=00H, (P0)=9CH (chương trình không thực hiện lệnh nhảy **JB P0.5, AAA**).

4.8. JNB bit, rel

- Chức năng: Nhảy nếu bit bằng 0.
- Mô tả: Nếu bit chỉ ra trong lệnh bằng 0 thì nhảy đến địa chỉ được chỉ ra trong lệnh còn ngược lại thì tiếp tục với lệnh tiếp theo. Các cờ không bị ảnh hưởng.

Số byte	3
Số chu kỳ	2
Mã đối tượng	00110000 bbbbbbbb eeeeeeee
Hoạt động	(PC) ← (PC) + 3 IF (bit) = 0 THEN (PC) ← (PC) + byte_2

- Lưu đồ:



- Lưu ý: Tầm nhảy của lệnh **JNB bit, rel** bị giới hạn ở khoảng cách nhảy từ -128 byte (nhảy lui) đến +127 byte (nhảy tới) kể từ lệnh kế tiếp theo sau lệnh nhảy có điều kiện này.
- Ví dụ 1: Cho biết trước (A)=56H, (P1)=CAH.

Sau khi thực thi chuỗi lệnh:

```

JNB P1.3, AAA
JNB ACC.3, BBB

```

thì chương trình được tiếp tục với lệnh tại nhãn BBB, (A)=56H và (P1)=CAH.

- Ví dụ 2: Cho chuỗi lệnh:

```

MOV    P1, #10H
MOV    A, #6BH           ; 6BH = 01101011B
JNB    ACC.5, AAA
MOV    P1, #01H
AAA:
.....

```

Sau khi thực thi chuỗi lệnh thì (A)=6BH, (P1)=01H (chương trình không thực hiện lệnh nhảy **JNB ACC.5, AAA**).

- Ví dụ 3: Cho chuỗi lệnh:

```

MOV    P1, #10H
MOV    E0H, #6BH        ; 6BH = 01101011B
JNB    E2H, AAA
MOV    P1, #01H
AAA:
.....

```

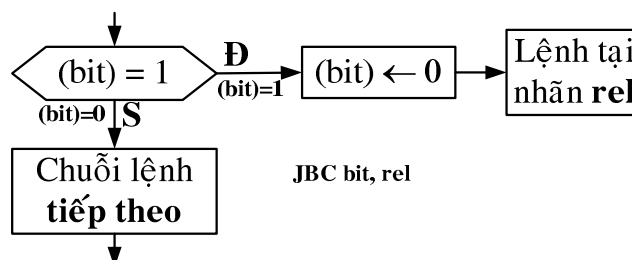
Sau khi thực thi chuỗi lệnh thì (A)=(E0H)=6BH, (P1)=10H (chương trình thực hiện lệnh nhảy **JNB E2H, AAA**).

4.9. JBC bit, rel

- Chức năng: Nhảy nếu bit bằng 1 và xóa bit (làm cho bit = 0).
- Mô tả: Nếu bit được chỉ ra trong lệnh bằng 1 thì xóa bit này và rẽ nhánh đến địa chỉ cho trong lệnh còn ngược lại thì tiếp tục với lệnh tiếp theo. Bit sẽ không được xóa nếu bit này đã là 0. Các cờ không bị ảnh hưởng.

Số byte	3
Số chu kỳ	2
Mã đối tượng	00010000 bbbbbbbb eeeeeeee
Hoạt động	$(PC) \leftarrow (PC) + 3$ IF (bit) = 1 THEN (bit) $\leftarrow$ 0 (PC) $\leftarrow$ (PC) + byte_2

- Lưu đồ:



- *Lưu ý:* Tầm nhảy của lệnh **JBC bit, rel** bị giới hạn ở khoảng cách nhảy từ -128 byte (*nhảy lui*) đến +127 byte (*nhảy tới*) kể từ lệnh kế tiếp theo sau lệnh nhảy có điều kiện này.
Khi lệnh này được dùng để làm thay đổi giá trị của một port xuất thì giá trị được dùng làm dữ liệu ban đầu của port được lấy từ bộ chốt dữ liệu xuất, không phải được lấy từ các chân nhập.

- *Ví dụ 1:* Cho biết trước (A)=56H.

Sau khi thực thi chuỗi lệnh:

JBC ACC.3, AAA

JBC ACC.2, BBB

thì chương trình được tiếp tục với lệnh tại nhãn BBB và (A)=52H.

- *Ví dụ 2:* Cho chuỗi lệnh:

MOV A, #76H

MOV P3, #9CH ; 9CH = 10011100B

JBC P3.2, AAA

MOV A, #67H

AAA:

.....

Sau khi thực thi chuỗi lệnh thì (A)=76H, (P3)=98H (*chương trình thực hiện lệnh nhảy JBC P3.2, AAA*).

- *Ví dụ 3:* Cho chuỗi lệnh:

MOV A, #76H

MOV B0H, #9CH ; 9CH = 10011100B

JBC B1H, AAA

MOV A, #67H

AAA:

.....

Sau khi thực thi chuỗi lệnh thì (A)=67H, (P3)=(B0H)=9CH (*chương trình không thực hiện lệnh nhảy JBC B1H, AAA*).

4.10. JC rel

- *Chức năng:* Nhảy nếu cờ CY = 1.
- *Mô tả:* Nếu cờ CY = 1 thì nhảy đến địa chỉ cho trong lệnh còn ngược lại thì tiếp tục với lệnh tiếp theo. Các cờ không bị ảnh hưởng.

Số byte 2

Số chu kỳ 2

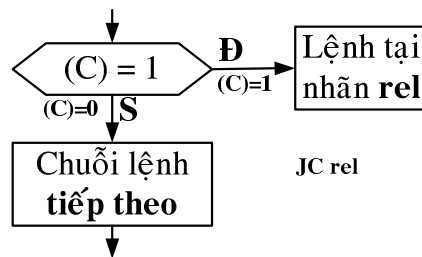
Mã đối tượng 01000000 eeeeeeee

Hoạt động (PC) ← (PC) + 2

IF (C) = 1 THEN

(PC) ← (PC) + byte_2

- Lưu đồ:



- Lưu ý: Tầm nhảy của lệnh **JC rel** bị giới hạn ở khoảng cách nhảy từ -128 byte (*nhảy lui*) đến +127 byte (*nhảy tới*) kể từ lệnh kế tiếp theo sau lệnh nhảy có điều kiện này.
- Ví dụ 1: Cho biết trước (A)=7FH, cờ CY=0.

Sau khi thực thi chuỗi lệnh:

```

JC   AAA
ADD  A,#0F7H
JC   BBB
    
```

thì chương trình được tiếp tục với lệnh tại nhãn BBB, (A)=76H, cờ CY=1.

- Ví dụ 2: Cho chuỗi lệnh:

```

MOV    A, #9AH
MOV    R0, #76H
ADD    A, R0
JC     AAA
MOV    R0, #67H
AAA:
.....
    
```

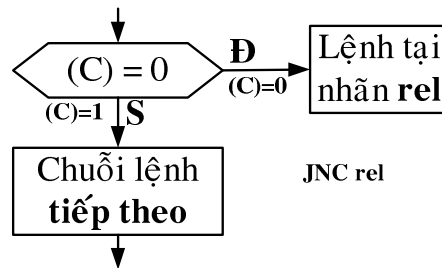
Sau khi thực thi chuỗi lệnh thì (A)=10H, (R0)=76H và cờ CY=1 (*chương trình thực hiện lệnh nhảy JC AAA*).

4.11. JNC rel

- Chức năng: Nhảy nếu cờ CY = 0.
- Mô tả: Nếu cờ CY = 0 thì nhảy đến địa chỉ cho trong lệnh còn ngược lại thì tiếp tục với lệnh tiếp theo. Các cờ không bị ảnh hưởng.

Số byte	2
Số chu kỳ	2
Mã đối tượng	01010000 eeeeeeee
Hoạt động	(PC) ← (PC) + 2 IF (C) = 0 THEN (PC) ← (PC) + byte_2

- Lưu đồ:



- Lưu ý: Tầm nhảy của lệnh **JNC rel** bị giới hạn ở khoảng cách nhảy từ -128 byte (*nhảy lui*) đến +127 byte (*nhảy tới*) kể từ lệnh kế tiếp theo sau lệnh nhảy có điều kiện này.
- Ví dụ 1: Cho biết trước (A)=7FH, cờ CY=1.

Sau khi thực thi chuỗi lệnh:

```

JNC AAA
ADD A,#26H
JNC BBB
    
```

thì chương trình được tiếp tục với lệnh tại nhãn BBB, (A)=A5H, cờ CY=0.

- Ví dụ 2: Cho chuỗi lệnh:

```

MOV    A, #0B9H
MOV    R1, #52H
ADD    A, R1
JNC    AAA
MOV    R1, #25H
AAA:
.....
    
```

Sau khi thực thi chuỗi lệnh thì (A)=0BH, (R1)=25H và cờ CY=1 (*chương trình không thực hiện lệnh nhảy JNC AAA*).

5. Nhóm lệnh rẽ nhánh:

5.1. ACALL addr11

- **Chức năng:** Gọi đến địa chỉ tuyệt đối (*Absolute Call*)
- **Mô tả:** ACALL gọi không điều kiện một chương trình con đặt tại địa chỉ được chỉ ra trong lệnh (xem thêm giải thích về chương trình con ở phần 5.3). **Chú ý rằng, 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. Các cờ không bị ảnh hưởng.**

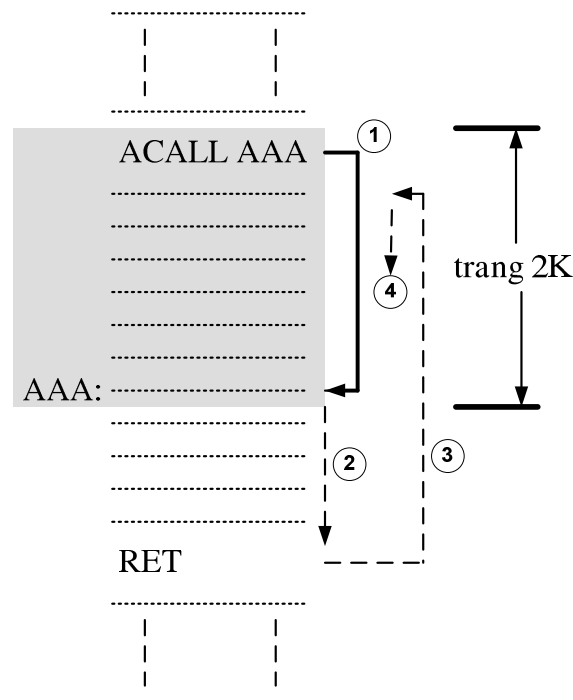
Số byte	2	
Số chu kỳ	2	
Mã đối tượng	aaa1001	aaaaaaaa

Lưu ý: aaa = A10–A8 và aaaaaaaa = A7–A0

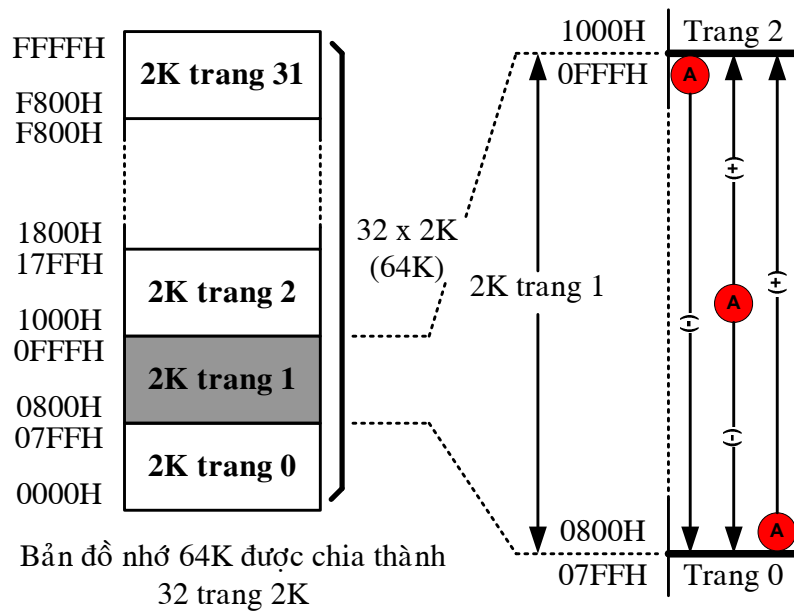
Hoạt động

$$\begin{aligned} (PC) &\leftarrow (PC) + 2 \\ (SP) &\leftarrow (SP) + 1 \\ ((SP)) &\leftarrow (PC7-PC0) \\ (SP) &\leftarrow (SP) + 1 \\ ((SP)) &\leftarrow (PC15-PC8) \\ (PC10-PC0) &\leftarrow \text{địa chỉ trang} \end{aligned}$$

- **Mô tả:**



- *Lưu ý:* Do không gian bộ nhớ chương trình tối đa là 64KB cho nên ta có 32 trang (*khối*) và mỗi trang bắt đầu ở địa chỉ là biên của 2KB (*như là: 0000H, 0800H, 1000H, 1800H, ..., F800H*).



- *Ví dụ 1:* Cho biết trước (SP)=07H, lệnh ACALL ở vị trí 0123H và nhãn AAA ở vị trí 0345H của bộ nhớ chương trình. Lệnh kế tiếp lệnh ACALL ở vị trí 0125H.

Sau khi thực thi lệnh **ACALL AAA** thì: (SP)=09H, (PC)=0345H và 2 ô nhớ của RAM nội (08H)=23H, (09H)=01H.

- *Ví dụ 2:* Cho biết trước (SP)=07H, lệnh ACALL ở vị trí 0800H và nhãn AAA ở vị trí 0700H của bộ nhớ chương trình.

Không thể thực thi lệnh **ACALL AAA** (lỗi lập trình) vì lệnh ACALL và nhãn AAA không nằm trong cùng một trang (lệnh ACALL thuộc trang 1, nhãn AAA thuộc trang 0).

5.2. LCALL addr16

- **Chức năng:** Gọi một chương trình con (Long Call).
- **Mô tả:** LCALL gọi một chương trình con với địa chỉ bắt đầu chương trình con được chỉ ra trong lệnh (xem thêm giải thích về chương trình con ở phần 5.3). Chú ý rằng, **chương trình con có thể bắt đầu ở bất cứ nơi nào trong không gian bộ nhớ chương trình 64KB. Các cờ không bị ảnh hưởng.**

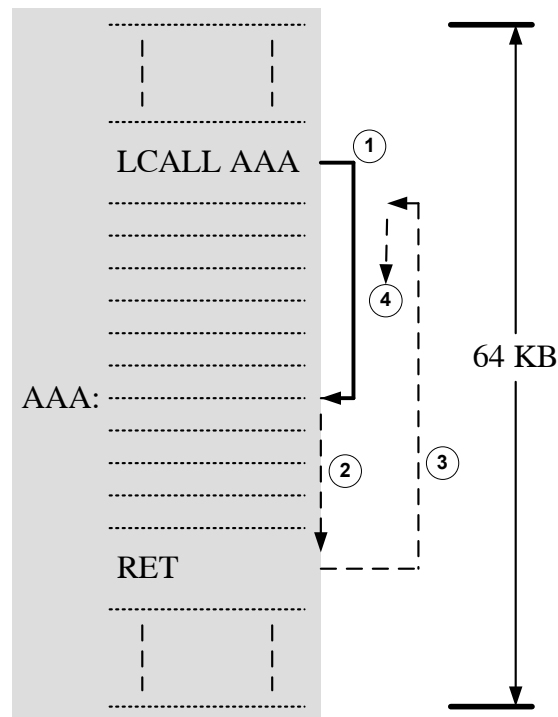
Số byte	3
Số chu kỳ	2
Mã đối tượng	00010010 aaaaaaaa aaaaaaaa

Lưu ý: byte 2 chứa các bit địa chỉ từ A15 – A8
byte 3 chứa các bit địa chỉ từ A7 – A0.

Hoạt động

$$\begin{aligned} (PC) &\leftarrow (PC) + 3 \\ (SP) &\leftarrow (SP) + 1 \\ (SP) &\leftarrow (PC7 - PC0) \\ (SP) &\leftarrow (SP) + 1 \\ (SP) &\leftarrow (PC15 - PC8) \\ (PC) &\leftarrow \text{addr15} - \text{addr0} \end{aligned}$$

- **Mô tả:**



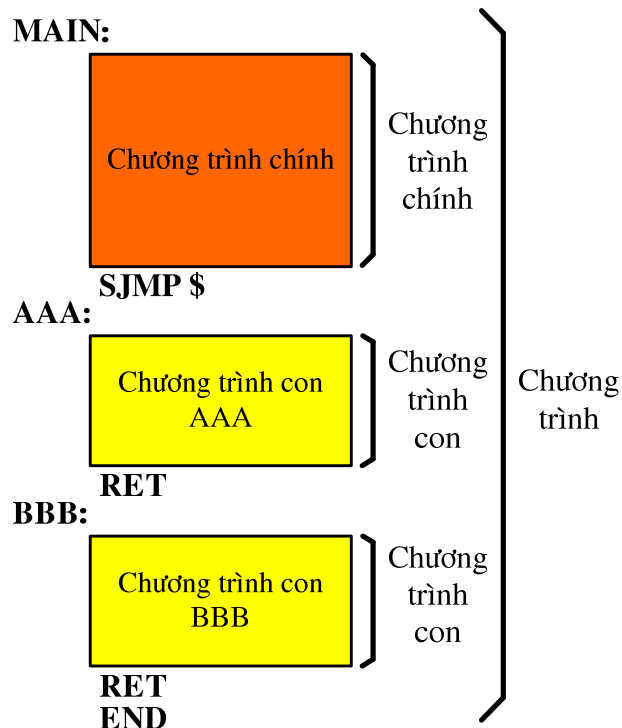
- **Ví dụ:** Cho biết trước (SP)=07H, lệnh LCALL ở vị trí 0123H và nhãn AAA ở vị trí 1234H của bộ nhớ chương trình. Lệnh kế tiếp lệnh LCALL ở vị trí 0126H.

Sau khi thực thi lệnh **LCALL AAA** thì: (SP)=09H, (PC)=1234H và 2 ô nhớ của RAM nội (08H)=26H, (09H)=01H.

5.3. RET

- **Chức năng:** Trở về từ chương trình con (*Return*).
- **Mô tả:** RET lấy lại các byte cao và byte thấp của PC từ *stack*, giảm con trỏ *stack* bởi 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. *Các cờ không bị ảnh hưởng.*

Số byte	1
Số chu kỳ	2
Mã đối tượng	00100010
Hoạt động	(PC15 – PC8) ← ((SP)) (SP) ← (SP) – 1 (PC7 – PC0) ← ((SP)) (SP) ← (SP) – 1
- **Lưu ý:** Chương trình con có một số qui định sau:
 - Là một chuỗi gồm nhiều lệnh được kết hợp lại với nhau để thực hiện một công việc nào đó.
 - Bắt đầu bằng một **NHÃN**, do người lập trình tự đặt ra (*nhãn này chính là tên của chương trình con*).
 - Kết thúc bằng lệnh **RET**.
 - Được đặt ở cuối chương trình, phía trên chỉ dẫn kết thúc chương trình **END**.
 - Giữa chương trình con và chương trình chính phải được cách ly bằng lệnh:
 - ✓ **SJMP \$**.
 - ✓ **SJMP MAIN** (*MAIN: là nhãn bắt kỳ thuộc chương trình chính*).
 - Chương trình con có thể được gọi ra nhiều lần tại bất kỳ thời điểm nào, tùy thuộc vào người lập trình yêu cầu (*thông qua các lệnh gọi ACALL, LCALL và các tín hiệu ngắt*).
 - Trong một chương trình có thể có một hoặc nhiều chương trình con.



- *Ví dụ 1:* Dựa vào *Ví dụ 1* của phần 5.1. Cho biết trước (SP)=09H; ô nhớ RAM nội (08H)=25H và (09H)=01H.

Sau khi thực thi lệnh **RET** thì: (SP)=07H và chương trình được tiếp tục với lệnh tại địa chỉ 0125H (vị trí của lệnh kế tiếp lệnh **ACALL**).

- *Ví dụ 2:* Dựa vào *Ví dụ* của phần 5.2. Cho biết trước (SP)=09H; ô nhớ RAM nội (08H)=26H và (09H)=01H.

Sau khi thực thi lệnh **RET** thì: (SP)=07H và chương trình được tiếp tục với lệnh tại địa chỉ 0126H (vị trí của lệnh kế tiếp lệnh **LCALL**).

5.4. RETI

- *Chức năng:* Trở về từ chương trình con phục vụ ngắt.
- *Mô tả:* RETI 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 xử lý. Con trỏ *stack* được giảm bởi 2. 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*. 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. *Nếu có một ngắt có ưu tiên ngắt thấp hơn hoặc cùng ưu tiên ngắt được treo khi lệnh RETI được thực thi, một lệnh được thực thi trước khi ngắt đang treo được xử lý.*

<i>Số byte</i>	1
<i>Số chu kỳ</i>	2
<i>Mã đối tượng</i>	00110010
<i>Hoạt động</i>	(PC15 – PC8) ← ((SP)) (SP) ← (SP) – 1 (PC7 – PC0) ← ((SP)) (SP) ← (SP) – 1

- *Lưu ý:* Lệnh **RETI** trả điều khiển về chương trình gọi từ một ISR (*ISR: Interrupt Service Routine: chương trình con phục vụ ngắt*). Điểm khác nhau giữa RETI và RET là RETI có báo hiệu cho hệ thống điều khiển ngắt rằng quá trình xử lý ngắt đã xong. Nếu trường hợp không có một ngắt nào được duy trì trong thời gian RETI thực thi thì lệnh RETI sẽ hoạt động như lệnh RET.
- *Ví dụ:* Cho biết trước (SP)=0BH; ô nhớ RAM nội (0AH)=23H và (0BH)=01H. một tín hiệu ngắt được phát hiện trong lệnh ở địa chỉ 0123H đang thực thi.

Sau khi thực thi lệnh **RETI** thì: (SP)=09H và chương trình được tiếp tục với lệnh tại địa chỉ 0123H.

5.5. AJMP addr11

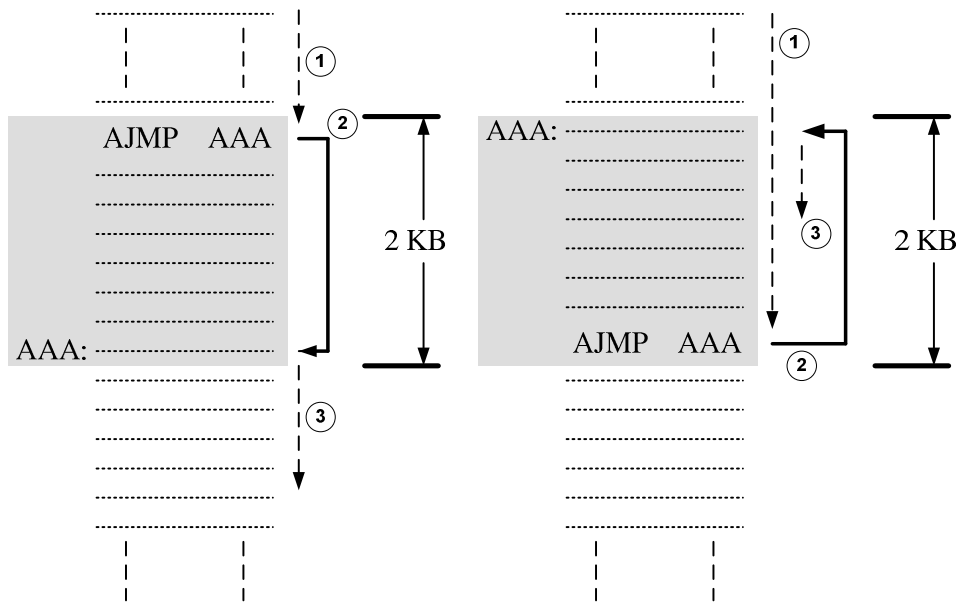
- *Chức năng:* Nhảy đến địa chỉ tuyệt đối (*Absolute Jump*).
- *Mô tả:* AJMP chuyển việc thực thi chương trình đến địa chỉ được chỉ ra trong lệnh. Chú ý rằng, **đích nhảy đến phải ở trong cùng khối 2K của bộ nhớ chương trình với byte đầu tiên của lệnh theo sau lệnh AJMP.**

Số byte 2
 Số chu kỳ 2
 Mã đối tượng aaa00001 aaaaaaaa

Ghi chú: aaa = A10 – A8 và aaaaaaaa = A7 – A0

Hoạt động (PC) ← (PC) + 2
 (PC10 – PC0) ← địa chỉ trang

- *Mô tả:*



- *Ví dụ:* Cho biết trước lệnh AJMP ở vị trí 0345H và nhãn AAA ở vị trí 0123H của bộ nhớ chương trình.

Sau khi thực thi lệnh **AJMP AAA** thì: (PC)=0123H.

5.6. LJMP addr16

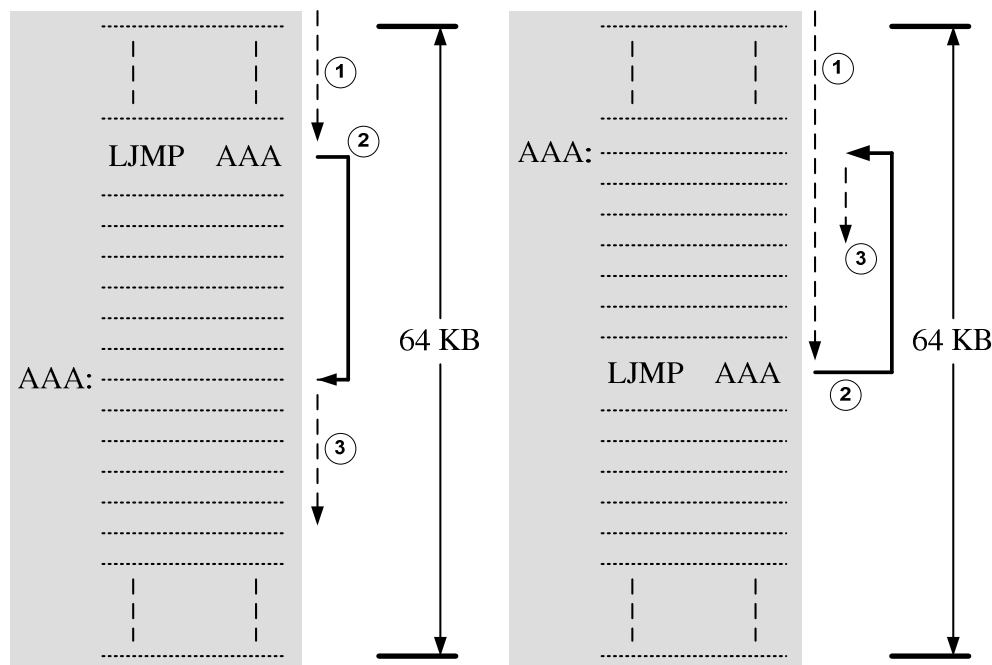
- *Chức năng:* Nhảy dài (Long Jump).
- *Mô tả:* LJMP tạo ra một rẽ nhánh không điều kiện đến địa chỉ được chỉ ra trong lệnh. Chú ý rằng, **địa chỉ đích có thể ở bất cứ nơi nào trong không gian địa chỉ của bộ nhớ chương trình 64KB**. Các cờ không bị ảnh hưởng.

Số byte	3
Số chu kỳ	2
Mã đối tượng	00010010 aaaaaaaa aaaaaaaa

Lưu ý: byte 2 chứa các bit địa chỉ từ A15–A8
byte 3 chứa các bit địa chỉ từ A7–A0

Hoạt động (PC) ← addr15 – addr0

- *Mô tả:*



- *Ví dụ:* Cho biết trước lệnh LJMP ở vị trí 0123H và nhãn AAA ở vị trí 1234H của bộ nhớ chương trình.

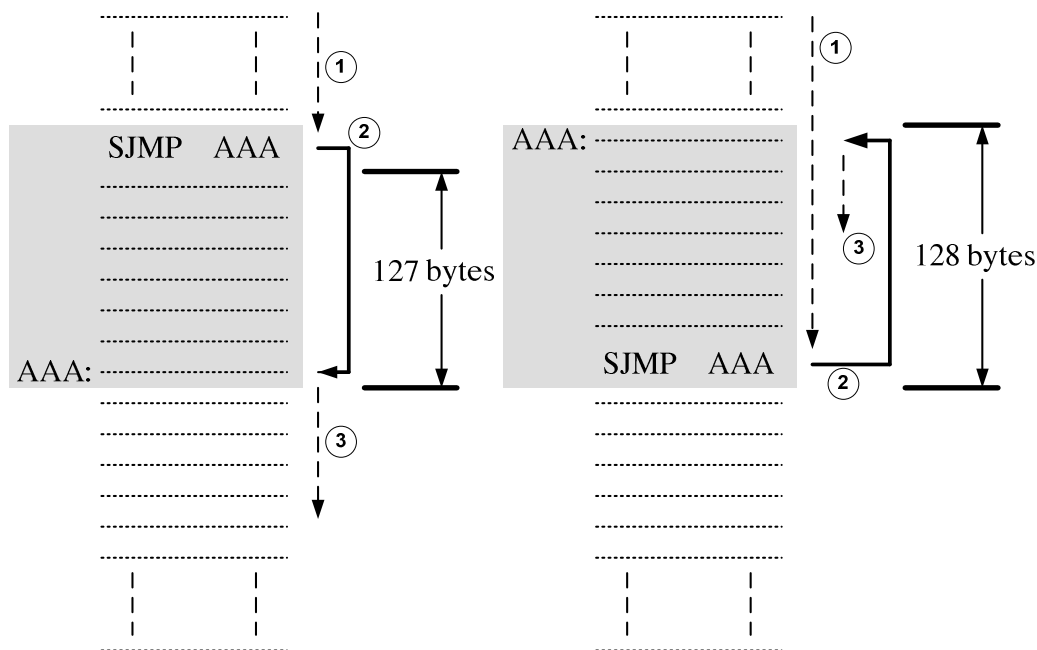
Sau khi thực thi lệnh **LJMP AAA** thì: (PC)=1234H.

5.7. SJMP rel

- **Chức năng:** Nhảy ngắn (*Short Jump*).
- **Mô tả:** Điều khiển chương trình rẽ nhánh không điều kiện đến địa chỉ được chỉ ra trong lệnh. Chú ý rằng, **tầm nhảy cho phép là 128 byte trước lệnh và 127 byte sau lệnh**.

Số byte	2
Số chu kỳ	2
Mã đối tượng	10000000 eeeeeeee
Hoạt động	$(PC) \leftarrow (PC) + 2$ $(PC) \leftarrow (PC) + \text{byte_2}$

- **Mô tả:**



- **Lưu ý:** Lệnh **SJMP \$** là một lệnh vòng lặp vô tận (lệnh nhảy tại chỗ), thường được sử dụng khi cần kết thúc một chương trình điều khiển của chip 8051 (*lưu ý rằng các tín hiệu ngắt vẫn hoạt động bình thường, vì thế đây chính là phương pháp duy nhất để thoát khỏi vòng lặp vô tận này*).
- **Ví dụ:** Cho biết trước lệnh SJMP nằm tại địa chỉ 0100H và nhãn AAA nằm tại địa chỉ 0123H trong bộ nhớ chương trình.

Sau khi thực thi lệnh **SJMP AAA** thì: $(PC)=0123H$.

5.8. JMP @A+DPTR

- *Chức năng:* Nhảy gián tiếp.
- *Mô tả:* Cộng giá trị 8-bit không dấu chứa trong thanh ghi A với con trỏ 16-bit và nạp kết quả tổng cho bộ đếm chương trình PC. Đây chính là địa chỉ của lệnh kế tiếp được tìm nạp. Cả hai, thanh ghi A và con trỏ dữ liệu đều không bị thay đổi. Các cờ không bị ảnh hưởng.

Số byte	1
Số chu kỳ	2
Mã đối tượng	01110011
Hoạt động	$(PC) \leftarrow (PC) + (A) + (DPTR)$

- *Ví dụ:* Cho biết trước (A)=00H, 02H, 04H, 06H.

Sau khi thực thi chuỗi lệnh:

```
MOV DPTR, #JMP_TBL
JMP @A+DPTR
```

JMP_TBL:

```
AJMP AAA ;Độ dài lệnh là 2 byte.
AJMP BBB ;Độ dài lệnh là 2 byte.
AJMP CCC ;Độ dài lệnh là 2 byte.
AJMP DDD ;Độ dài lệnh là 2 byte.
```

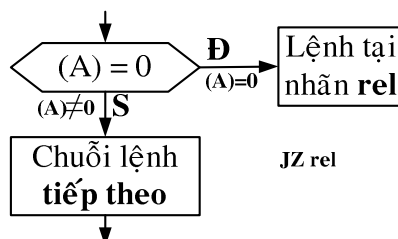
- thì: nếu (A)=00H, chương trình được tiếp tục với lệnh tại nhãn AAA.
 nếu (A)=02H, chương trình được tiếp tục với lệnh tại nhãn BBB.
 nếu (A)=04H, chương trình được tiếp tục với lệnh tại nhãn CCC.
 nếu (A)=06H, chương trình được tiếp tục với lệnh tại nhãn DDD.

5.9. JZ rel

- *Chức năng:* Nhảy nếu nội dung thanh ghi A bằng 0.
- *Mô tả:* Nếu tất cả các bit của thanh ghi A đều bằng 0 thì nhảy đến địa chỉ cho trong lệnh còn ngược lại tiếp tục với lệnh tiếp theo. Các cờ không bị ảnh hưởng. Nội dung thanh ghi A không bị thay đổi.

Số byte	2
Số chu kỳ	2
Mã đối tượng	01100000 eeeeeeee
Hoạt động	$(PC) \leftarrow (PC) + 2$ IF (A) = 0 THEN $(PC) \leftarrow (PC) + \text{byte_2}$

- *Lưu đồ:*



- *Lưu ý:* Tầm nhảy của lệnh **JZ rel** bị giới hạn ở khoảng cách nhảy từ -128 byte (*nhảy lui*) đến +127 byte (*nhảy tới*) kể từ lệnh kế tiếp theo sau lệnh nhảy có điều kiện này.
- *Ví dụ:* Cho biết trước (A)=01H.

Sau khi thực thi chuỗi lệnh:

JZ AAA
DEC A
JZ BBB

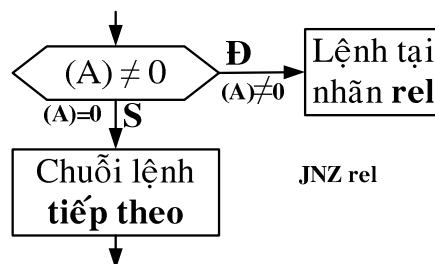
thì chương trình được tiếp tục với lệnh tại nhãn BBB.

5.10. JNZ rel

- *Chức năng:* Nhảy nếu nội dung thanh ghi A khác 0.
- *Mô tả:* Nếu thanh ghi A có một bit bất kỳ bằng 1 thì nhảy đến địa chỉ cho trong lệnh còn ngược lại tiếp tục với lệnh tiếp theo. *Các cờ không bị ảnh hưởng. Nội dung thanh ghi A không bị thay đổi.*

Số byte	2
Số chu kỳ	2
Mã đối tượng	01110000 eeeeeeee
Hoạt động	(PC) ← (PC) + 2 IF (A) <> 0 THEN (PC) ← (PC) + byte_2

- *Lưu đồ:*



- *Lưu ý:* Tầm nhảy của lệnh **JNZ rel** bị giới hạn ở khoảng cách nhảy từ -128 byte (*nhảy lui*) đến +127 byte (*nhảy tới*) kể từ lệnh kế tiếp theo sau lệnh nhảy có điều kiện này.
- *Ví dụ:* Cho biết trước (A)=00H.

Sau khi thực thi chuỗi lệnh:

JNZ AAA
INC A
JNZ BBB

thì chương trình được tiếp tục với lệnh tại nhãn BBB.

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

- *Chức năng:* So sánh và nhảy nếu không bằng.
- *Mô tả:* CJNE so sánh giá trị của 2 toán hạng (*src-byte*) và (*dest-byte*) rồi rẽ nhánh đến địa chỉ được chỉ ra trong lệnh nếu các giá trị của 2 toán hạng này không bằng nhau. Còn $CY = 1$ nếu giá trị nguyên không dấu của (*dest-byte*) nhỏ hơn giá trị nguyên không dấu của (*src-byte*) và ngược lại $CY = 0$. Không có toán hạng nào trong 2 toán hạng bị ảnh hưởng.
- *Các dạng lệnh:*

CJNE A, direct, rel

Số byte 3
 Số chu kỳ 2
 Mã đối tượng 10110101 aaaaaaaa eeeeeeee
 Hoạt động (PC) $\leftarrow$ (PC) + 3
 IF (A) $\neq$ (direct) THEN
 (PC) $\leftarrow$ (PC) + địa chỉ tương đối
 IF (A) < (direct) THEN
 (C) $\leftarrow$ 1
 ELSE
 (C) $\leftarrow$ 0

CJNE A, #data, rel

Số byte 3
 Số chu kỳ 2
 Mã đối tượng 10110100 dddddddd eeeeeeee
 Hoạt động (PC) $\leftarrow$ (PC) + 3
 IF (A) $\neq$ #data THEN
 (PC) $\leftarrow$ (PC) + địa chỉ tương đối
 IF (A) < #data THEN
 (C) $\leftarrow$ 1
 ELSE
 (C) $\leftarrow$ 0

CJNE Rn, #data, rel

Số byte 3
 Số chu kỳ 2
 Mã đối tượng 10111rrr dddddddd eeeeeeee
 Hoạt động (PC) $\leftarrow$ (PC) + 3
 IF (Rn) $\neq$ #data THEN
 (PC) $\leftarrow$ (PC) + địa chỉ tương đối
 IF (Rn) < #data THEN
 (C) $\leftarrow$ 1
 ELSE
 (C) $\leftarrow$ 0

CJNE @Ri, #data, rel

Số byte 3

Số chu kỳ 2

Mã đối tượng 1011011i dddddddd eeeeeeee

Hoạt động (PC) ← (PC) + 3

IF ((Ri)) <> #data THEN

(PC) ← (PC) + địa chỉ tương đối

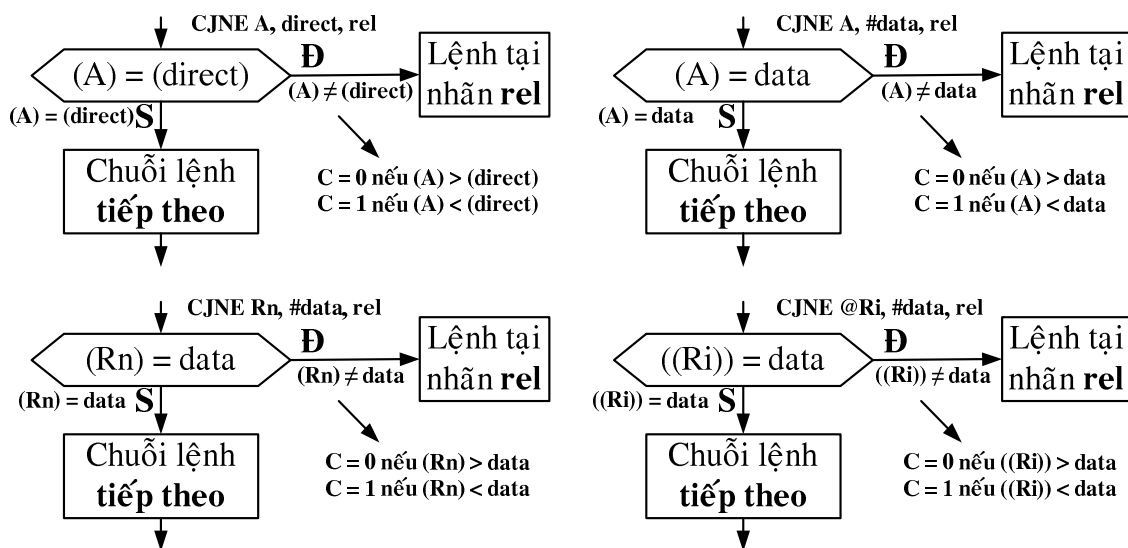
IF ((Ri)) < #data THEN

(C) ← 1

ELSE

(C) ← 0

- Lưu đồ:



- Ví dụ 1: Cho biết trước (A)=34H, (01H)=(R1)=34H, (34H)=B9H. Sau khi thực thi chuỗi lệnh:

CJNE A, 01H, AAA

CJNE @R1, #9BH, BBB

thì chương trình được tiếp tục với lệnh tại nhãn BBB và cờ C=0.

- Ví dụ 2: Cho chuỗi lệnh:

CLR C

MOV A, #40H

MOV 40H, #0B1H

CJNE A, 40H, AAA

MOV 40H, #1BH

AAA:

ADDC A, 40H

Sau khi thực thi chuỗi lệnh thì (A)=F2H, (40H)=B1H, CY=0.

- Ví dụ 3: Ứng dụng cho phép so sánh lớn hơn hay nhỏ hơn. Giả sử:
 - Khi ta muốn nhảy đến nhãn BIG nếu $(A) \geq \#20H$, thì các lệnh sau được sử dụng:

```

.....
CJNE    A, #20H, $+3      ;So sánh (A) với con số 20H.
JNC     BIG               ;Nhảy đến BIG nếu  $(A) \geq \#20H$ .
.....

```

- Khi ta muốn nhảy đến nhãn SMALL nếu $(A) < \#20H$, thì các lệnh sau được sử dụng:

```

.....
CJNE    A, #20H, $+3      ;So sánh (A) với con số 20H.
JC      SMALL            ;Nhảy đến SMALL nếu  $(A) < \#20H$ .
.....

```

- Lưu ý: Ký hiệu \$ là một ký hiệu của trình dịch hợp ngữ, biểu thị địa chỉ của lệnh hiện hành (vì CJNE có độ dài lệnh là 3 byte nên \$+3 sẽ là địa chỉ của lệnh tiếp theo CJNE, tức là lệnh JC/JNC).

5.12. DJNZ <byte>, <rel-addr>

- Chức năng: Giảm và nhảy nếu byte khác 0.
- Mô tả: DJNZ giảm byte chỉ ra bởi toán hạng đầu trong lệnh và rẽ nhánh đến địa chỉ được chỉ ra bởi toán hạng thứ hai trong lệnh nếu kết quả sau khi giảm khác 0. Nếu giá trị ban đầu của byte là 00H ta sẽ có tràn sang 0FFH. Các cờ không bị ảnh hưởng.
- Các dạng lệnh:

DJNZ Rn, rel

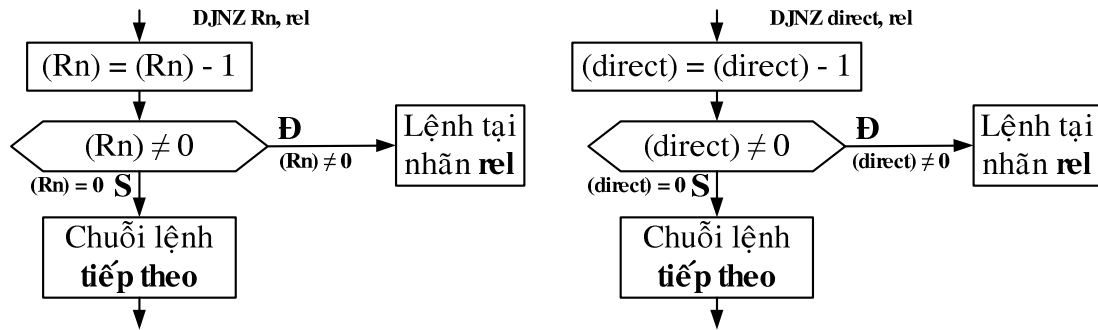
Số byte	2
Số chu kỳ	2
Mã đối tượng	11011rrr eeeeeeee
Hoạt động	$(PC) \leftarrow (PC) + 2$ $(Rn) \leftarrow (Rn) - 1$ IF $(Rn) < > 0$ THEN $(PC) \leftarrow (PC) + \text{byte_2}$

DJNZ direct, rel

Số byte	3
Số chu kỳ	2
Mã đối tượng	11010101 aaaaaaaaa eeeeeeee
Hoạt động	$(PC) \leftarrow (PC) + 2$ $(direct) \leftarrow (direct) - 1$ IF $(direct) < > 0$ THEN $(PC) \leftarrow (PC) + \text{byte_2}$

- Lưu ý: Khi lệnh này được dùng để làm thay đổi giá trị của một port xuất thì giá trị được dùng làm dữ liệu ban đầu của port được lấy từ bộ chốt dữ liệu xuất, không phải được lấy từ các chân nhập.

- Lưu đồ:



- Lưu ý: Khi lệnh DJNZ được thực hiện thì giá trị trong thanh ghi Rn được giảm (-1) trước khi đem ra so sánh với giá trị “0”.
- Ví dụ 1: Cho chuỗi lệnh sau:

```

MOV A, #50
MOV R2, #8
AAA: ADD A, #1
DJNZ R2, AAA
  
```

Sau khi thực thi chuỗi lệnh thì (A)=58, (R2)=0.

- Ví dụ 2: Cho chuỗi lệnh sau:

```

MOV A, #50
MOV R3, #56
AAA: ADD A, #1
DJNZ R3, AAA
  
```

- Sau khi thực thi chuỗi lệnh thì (A)=106, (R1)=0.

- Ví dụ 3: Cho chuỗi lệnh sau:

```

MOV A, #50
MOV R1, #0
AAA: ADD A, #1
DJNZ R1, AAA
  
```

Sau khi thực thi chuỗi lệnh thì (A)=50, (R1)=0.

- Lưu ý rằng: Qua ba ví dụ về lệnh DJNZ ở trên cho ta một nhận xét như sau:
 - Nếu (Rn)=8 ⇒ Lệnh ADD được thực hiện **8 lần**.
 - Nếu (Rn)=56 ⇒ Lệnh ADD được thực hiện **56 lần**.
 - Nếu (Rn)=0 ⇒ Lệnh ADD được thực hiện **256 lần**.

5.13. NOP

- Chức năng: Không làm gì (No Operation).
- Mô tả: Việc thực thi chương trình tiếp tục với lệnh tiếp theo. Không có thanh ghi hay cờ nào bị ảnh hưởng.

Số byte	1
Số chu kỳ	1
Mã đối tượng	00000000
Hoạt động	(PC) ← (PC) + 1

Danh sách các lệnh làm ảnh hưởng tới các cờ CY, AC và OV

Instruction	Flag			Instruction	Flag		
	C	OV	AC		C	OV	AC
ADD	X	X	X	CLR C	O		
ADDC	X	X	X	CPL C	X		
SUBB	X	X	X	ANL C,bit	X		
MUL	O	X		ANL C,/bit	X		
DIV	O	X		ORL C,bit	X		
DA	X			ORL C,/bit	X		
RRC	X			MOV C,bit	X		
RLC	X			CJNE	X		
SETB C	1						

Ghi chú: X là cờ tương ứng bị ảnh hưởng.
 0 là cờ tương ứng bằng 0. 1 là cờ tương ứng bằng 1.

IV. CÁC VÍ DỤ ỨNG DỤNG VỀ TẬP LỆNH:

- Ví dụ 1: Viết đoạn lệnh để xóa thanh ghi A và sau đó cộng 9 vào thanh ghi A 10 lần. Sau khi hoàn tất thì cất giá trị trong thanh ghi A vào thanh ghi R7.

Giải

```

MOV    A, #0           ;Xóa ACC, A = 0.
MOV    R0, #10          ;Nạp số lần lặp, R0 = 10.
BACK:
ADD     A, #9           ;Cộng thêm 9 vào ACC.
DJNZ   R0, BACK         ;Kiểm tra số lần lặp lại, 10 lần.
MOV    R7, A            ;Cất ACC vào thanh ghi R7.
```

- Ví dụ 2: Viết đoạn lệnh để nạp vào thanh ghi A với giá trị FFH và sau đó lấy bù thanh ghi A 500 lần.

Giải

```

MOV    A, #0FFH        ;Nạp A = FFH.
MOV    R0, #10          ;Nạp số lần lặp 1, R0 = 10.
LOOP:
MOV    R1, #50          ;Nạp số lần lặp 2, R1 = 50.
BACK:
CPL    A                ;Lấy bù ACC.
DJNZ   R1, BACK         ;Kiểm tra số lần lặp 2 (vòng trong), 50 lần.
DJNZ   R0, LOOP         ;Kiểm tra số lần lặp 1(vòng ngoài), 10 lần.
```

- Ví dụ 3: Viết đoạn lệnh để xác định xem R0 có chứa giá trị 0 hay không? Nếu không thì nạp vào R0 giá trị FFH.

Giải

```
MOV    A, R0           ;Chuyển nội dung R0 vào A.
JNZ    NEXT           ;Nhảy đến NEXT nếu A ≠ 0 (thoát ra).
MOV    R0, #0FFH      ;Nạp R0 = FFH nếu A = 0.
```

NEXT:

- Ví dụ 4: Viết đoạn lệnh để tìm tổng của 79H, F5H và E2H. Ghi byte thấp của tổng vào R0 và byte cao của tổng vào R1.

Giải

```
MOV    A, #0           ;Xoá ACC, A = 0.
MOV    R1, #0           ;Xoá R1, R1 = 0.
ADD    A, #79H          ;Cộng thêm 79H (A = 79H, CY = 0).
JNC    NO_CY1           ;Nhảy nếu CY = 0.
INC    R1               ;Nếu CY = 1 thì tăng R1.
NO_CY1:
ADD    A, #0F5H         ;Cộng thêm F5H (A = 6EH, CY = 1).
JNC    NO_CY2           ;Nhảy nếu CY = 0.
INC    R1               ;Nếu CY = 1 thì tăng R1.
NO_CY2:
ADD    A, #0E2H         ;Cộng thêm E2H (A = 50H, CY = 1).
JNC    NO_CY3           ;Nhảy nếu CY = 0.
INC    R1               ;Nếu CY = 1 thì tăng R1.
NO_CY3:
MOV    R0, A            ;R0 = 50H (byte thấp), R1 = 02H (byte cao).
```

- Ví dụ 5: Hệ thống sử dụng 8051 có tần số dao động của thạch anh là 11,0592MHz. Hãy xác định thời gian cần thiết để thực hiện các lệnh sau:

```
MOV R3, #55H      DEC R3      DJNZ R2, AAA
LJMP AAA          SJMP AAA     NOP
MUL AB
```

Giải

Chu kỳ máy:

$$T_{Machine} = \frac{12}{f_{OSC}} = \frac{12}{11,0592MHz} = 1,085(\mu s)$$

<u>Lệnh:</u>	<u>Số chu kỳ máy:</u>	<u>Thời gian thực hiện:</u>
MOV R3, #55H	1	t = 1 x 1,085 (μs) = 1,085 (μs)
DEC R3	1	t = 1 x 1,085 (μs) = 1,085 (μs)
DJNZ R2, AAA	2	t = 2 x 1,085 (μs) = 2,17 (μs)
LJMP AAA	2	t = 2 x 1,085 (μs) = 2,17 (μs)
SJMP AAA	2	t = 2 x 1,085 (μs) = 2,17 (μs)
NOP	1	t = 1 x 1,085 (μs) = 1,085 (μs)
MUL AB	4	t = 4 x 1,085 (μs) = 4,34 (μs)

- Ví dụ 6: Hệ thống sử dụng 8051 có tần số dao động của thạch anh là 11,0592MHz. Hãy xác định thời gian cần thiết để thực hiện hoàn tất đoạn lệnh sau:

```
MOV A, #55H
MOV P1, A
CPL A
END
```

Giải

Chu kỳ máy:

$$T_{Machine} = \frac{12}{f_{OSC}} = \frac{12}{11,0592MHz} = 1,085(\mu s)$$

Đoạn lệnh:

```
MOV A, #55H
MOV P1, A
CPL A
END
```

Số chu kỳ máy:

```
1
1
1
```

Tổng thời gian thực hiện đoạn lệnh trên là:

$$t = (1 + 1 + 1) \times 1,085 (\mu s) = 3,255 (\mu s)$$

- Ví dụ 7: Hệ thống sử dụng 8051 có tần số dao động của thạch anh là 11,0592MHz. Hãy xác định thời gian cần thiết để thực hiện hoàn tất đoạn lệnh sau:

DELAY:

```
MOV R3, #200
DJNZ R3, $
RET
```

Giải

Chu kỳ máy:

$$T_{Machine} = \frac{12}{f_{OSC}} = \frac{12}{11,0592MHz} = 1,085(\mu s)$$

Đoạn lệnh:**DELAY:**

```
MOV R3, #200
DJNZ R3, $
RET
```

Số chu kỳ máy:

```
1
2
1
```

Tổng thời gian thực hiện đoạn lệnh trên là:

$$t = [1 + (2 \times 200) + 1] \times 1,085 (\mu s) = 436,17 (\mu s)$$

- Ví dụ 8: Hệ thống sử dụng 8051 có tần số dao động của thạch anh là 11,0592 MHz. Hãy xác định thời gian cần thiết để thực hiện hoàn tất đoạn lệnh sau:

Giải

Chu kỳ máy:

$$T_{Machine} = \frac{12}{f_{OSC}} = \frac{12}{11,0592MHz} = 1,085(\mu s)$$

Đoạn lệnh:Số chu kỳ máy:

DELAY:

MOV R3, #250

1

HERE:

NOP

1

NOP

1

NOP

1

NOP

1

DJNZ R3, HERE

2

RET

1

Tổng thời gian thực hiện đoạn lệnh trên là:

$$t = [1 + ((1 + 1 + 1 + 1 + 2) \times 250) + 1] \times 1,085 (\mu s) = 1629,67 (\mu s)$$

- Ví dụ 9: Viết đoạn lệnh để chuyển giá trị của thanh ghi R5, R6 và A vào ngăn xếp. Sau đó lấy ra và cho lần lượt vào các thanh ghi R2, R3 và B tương ứng.

Giải

PUSH	05H	;Cất R5 vào ngăn xếp.
PUSH	06H	;Cất R6 vào ngăn xếp.
PUSH	0E0H	;Cất ACC vào ngăn xếp.
POP	0F0H	;Lấy từ ngăn xếp cho vào B, (B) = (A).
POP	03H	;Lấy từ ngăn xếp cho vào R3, (R3) = (R6).
POP	02H	;Lấy từ ngăn xếp cho vào R2, (R2) = (R5).

- Ví dụ 10: Viết đoạn lệnh để chuyển giá trị trong ô nhớ có địa chỉ 55H vào các ô nhớ RAM tại địa chỉ từ 40H – 44H, sử dụng:

- Chế độ định địa chỉ trực tiếp.
- Chế độ định địa chỉ gián tiếp (không dùng vòng lặp).
- Chế độ định địa chỉ gián tiếp (dùng vòng lặp).

Giải Chế độ định địa chỉ trực tiếp:

MOV	A, #55H	;Nạp giá trị 55H vào thanh ghi A.
MOV	40H, A	;Sao nội dung của A vào ô nhớ 40H.
MOV	41H, A	;Sao nội dung của A vào ô nhớ 41H.
MOV	42H, A	;Sao nội dung của A vào ô nhớ 42H.
MOV	43H, A	;Sao nội dung của A vào ô nhớ 43H.
MOV	44H, A	;Sao nội dung của A vào ô nhớ 44H.

✚ Chế độ định địa chỉ gián tiếp (không dùng vòng lặp):

MOV	A, #55H	;Nạp giá trị 55H vào thanh ghi A.
MOV	R0, #40H	;Nạp địa chỉ bắt đầu, R0 = 40H.
MOV	@R0, A	;Sao nội dung A vào ô nhớ do R0 trỏ đến.
INC	R0	;Tăng con trỏ, R0 = 41H.
MOV	@R0, A	;Sao nội dung A vào ô nhớ do R0 trỏ đến.
INC	R0	;Tăng con trỏ, R0 = 42H.
MOV	@R0, A	;Sao nội dung A vào ô nhớ do R0 trỏ đến.
INC	R0	;Tăng con trỏ, R0 = 43H.
MOV	@R0, A	;Sao nội dung A vào ô nhớ do R0 trỏ đến.
INC	R0	;Tăng con trỏ, R0 = 44H.
MOV	@R0, A	;Sao nội dung A vào ô nhớ do R0 trỏ đến.

✚ Chế độ định địa chỉ gián tiếp (dùng vòng lặp):

MOV	A, #55H	;Nạp giá trị 55H vào thanh ghi A.
MOV	R0, #40H	;Nạp địa chỉ bắt đầu, R0 = 40H.
MOV	R1, #5	;Nạp số lần lặp lại, R1 = 5.
LOOP:		
MOV	@R0, A	;Sao nội dung A vào ô nhớ do R0 trỏ đến.
INC	R0	;Tăng con trỏ.
DJNZ	R1, LOOP	;Kiểm tra số lần lặp cho đến khi số lần = 0.

- Ví dụ 11: Viết đoạn lệnh để xóa 16 ô nhớ RAM nội có địa chỉ bắt đầu từ 60H.

Giải

CLR	A	;Xóa ACC, A = 0.
MOV	R0, #60H	;Nạp địa chỉ bắt đầu, R0 = 60H.
MOV	R1, #16	;Nạp số lần lặp lại, R1 = 16.
LOOP:		
MOV	@R0, A	;Sao nội dung A vào ô nhớ do R0 trỏ đến.
INC	R0	;Tăng con trỏ.
DJNZ	R1, LOOP	;Kiểm tra số lần lặp cho đến khi số lần = 0.

- Ví dụ 12: Viết đoạn lệnh để chuyển một khối dữ liệu gồm 10 byte từ vị trí ô nhớ RAM nội bắt đầu tại địa chỉ 35H đến các vị trí ô nhớ RAM nội bắt đầu tại địa chỉ 60H.

Giải

MOV	R0, #35H	;Nạp địa chỉ bắt đầu (nguồn), R0 = 35H.
MOV	R1, #60H	;Nạp địa chỉ bắt đầu (đích), R1 = 60H.
MOV	R2, #10	;Nạp số lần lặp lại, R2 = 5.
LOOP:		
MOV	A, @R0	;Sao nội dung ô nhớ do R0 trỏ đến vào A.
MOV	@R1, A	;Sao nội dung A vào ô nhớ do R1 trỏ đến.
INC	R0	;Tăng con trỏ (nguồn).
INC	R1	;Tăng con trỏ (đích).
DJNZ	R2, LOOP	;Kiểm tra số lần lặp cho đến khi số lần = 0.

- Ví dụ 13: Giả sử chữ “DHCN” được lưu trong ROM nội tại vùng nhớ có địa chỉ bắt đầu là 200H. Hãy phân tích cách chương trình sau đây hoạt động và xác định xem chữ “DHCN” sẽ được lưu vào đâu sau khi chương trình hoàn tất?

ORG	0000H	;Địa chỉ lưu chương trình trong ROM.
MOV	DPTR, #200H	;Nạp con trỏ vùng dữ liệu, DPTR=200H.
CLR	A	;Xoá ACC, A = 0.
MOVC	A, @A+DPTR	;Lấy dữ liệu tại ô nhớ ROM do (A+DPTR) = 200H trở đến đưa vào A.
MOV	R0, A	;Cất dữ liệu đó vào R0.
INC	DPTR	;Tăng con trỏ, DPTR=201H.
CLR	A	;Xoá ACC, A = 0.
MOVC	A, @A+DPTR	;Lấy dữ liệu tại ô nhớ ROM do (A+DPTR) = 201H trở đến đưa vào A.
MOV	R1, A	;Cất dữ liệu đó vào R1.
INC	DPTR	;Tăng con trỏ, DPTR=202H.
CLR	A	;Xoá ACC, A = 0.
MOVC	A, @A+DPTR	;Lấy dữ liệu tại ô nhớ ROM do (A+DPTR) = 202H trở đến đưa vào A.
MOV	R2, A	;Cất dữ liệu đó vào R2.
INC	DPTR	;Tăng con trỏ, DPTR=203H.
CLR	A	;Xoá ACC, A = 0.
MOVC	A, @A+DPTR	;Lấy dữ liệu tại ô nhớ ROM do (A+DPTR) = 203H trở đến đưa vào A.
MOV	R3, A	;Cất dữ liệu đó vào R3.
SJMP	\$	;Dừng chương trình.
ORG	200H	;Địa chỉ lưu chương trình trong ROM.
MYDATA:		
DB	“DHCN”	;Khai báo dữ liệu.
END		;Kết thúc chương trình.

Giải

Theo chương trình trên, các ô nhớ của bộ nhớ chương trình (ROM) có địa chỉ 200H - 203H chứa các nội dung sau: **(200H) = 44H = ‘D’, (201H) = 48H = ‘H’, (202H) = 43H = ‘C’, (203H) = 4EH = ‘N’**.

Đầu tiên với (DPTR) = 200H và (A) = 0. Lệnh **MOVC A, @A+DPTR** chuyển nội dung của ô nhớ có địa chỉ 200H (A + DPTR = 0 + 200H) trong ROM vào A. Thanh ghi A lúc này sẽ chứa giá trị 44H là mã ASCII của ký tự “D”. Ký tự này được cất vào thanh ghi R0.

Tiếp theo, DPTR được tăng lên (DPTR = 201H) và A được xoá (A = 0) để lấy nội dung của vị trí nhớ kế tiếp trong ROM có địa chỉ là 201H (A + DPTR = 0 + 200H) vào A. Thanh ghi A lúc này sẽ chứa giá trị 48H là mã ASCII của ký tự “H”. Ký tự này được cất vào thanh ghi R1.

Quá trình diễn ra tương tự như vậy, sau khi hoàn tất chương trình ta có (R0) = 44H, (R1) = 48H, (R2) = 43H, (R3) = 4EH là mã ASCII của các ký tự “D”, “H”, “C” và “N”.

- Ví dụ 14: Giả sử chữ “TP.HCM” được lưu trong ROM nội tại vùng nhớ có địa chỉ bắt đầu là 200H. Hãy viết chương trình để chuyển các byte dữ liệu này vào các ô nhớ RAM nội bắt đầu từ địa chỉ 40H.

Giải

✚ Phương pháp sử dụng bộ đếm dữ liệu:

```

ORG    0000H           ;Địa chỉ lưu chương trình trong ROM.
MOV    DPTR, #MYDATA   ;Nạp con trỏ vùng dữ liệu.
MOV    R0, #40H         ;Nạp địa chỉ bắt đầu chứa trong RAM.
MOV    R1, #6           ;Nạp giá trị bộ đếm (số lượng ký tự).

LOOP:
CLR    A               ;Xoá ACC, A = 0
MOVC   A, @A+DPTR      ;Lấy dữ liệu tại ô nhớ ROM do
                        ;(A+DPTR) trỏ đến đưa vào A.
MOV    @R0, A           ;Cất vào ô nhớ RAM do R0 trỏ đến.
INC    DPTR             ;Tăng con trỏ dữ liệu.
INC    R0               ;Tăng địa chỉ vùng RAM.
DJNZ   R1, LOOP        ;Lặp lại cho đến khi bộ đếm = 0.
SJMP   $               ;Dừng chương trình.
ORG    200H            ;Địa chỉ lưu chương trình trong ROM.

MYDATA:
DB     "TP.HCM"         ;Khai báo dữ liệu.
END                    ;Kết thúc chương trình.

```

✚ Phương pháp sử dụng ký tự NULL để kết thúc chuỗi:

```

ORG    0000H           ;Địa chỉ lưu chương trình trong ROM.
MOV    DPTR, #MYDATA   ;Nạp con trỏ vùng dữ liệu.
MOV    R0, #40H         ;Nạp địa chỉ bắt đầu chứa trong RAM.

LOOP:
CLR    A               ;Xoá ACC, A = 0
MOVC   A, @A+DPTR      ;Lấy dữ liệu tại ô nhớ ROM do
                        ;(A+DPTR) trỏ đến đưa vào A.
JZ     EXIT            ;Thoát ra nếu có ký tự NULL.
MOV    @R0, A           ;Cất vào ô nhớ RAM do R0 trỏ đến.
INC    DPTR             ;Tăng con trỏ dữ liệu.
INC    R0               ;Tăng địa chỉ vùng RAM.
SJMP   LOOP            ;Lặp lại.

EXIT:
SJMP   $               ;Dừng chương trình.
ORG    200H            ;Địa chỉ lưu chương trình trong ROM.

MYDATA:
DB     "TP.HCM", 0      ;Khai báo dữ liệu, có ký tự NULL.
END                    ;Kết thúc chương trình.

```

- Ví dụ 15: Viết chương trình để lấy giá trị x từ P1 và liên tục gửi giá trị x^2 đến P2.

Giải

```

ORG    0000H           ;Địa chỉ lưu chương trình trong ROM.
MOV     DPTR, #MYDATA   ;Nạp con trỏ vùng dữ liệu.
MOV     P1, #0FFH       ;Cấu hình P1 là port nhập.

LOOP:   MOV     A, P1     ;Lấy số liệu x từ P1.
        MOVC    A, @A+DPTR ;Lấy giá trị  $x^2$  từ vùng dữ liệu.
        MOV     P2, A     ;Xuất giá trị  $x^2$  ra P2.
        SJMP    LOOP      ;Lặp lại.
ORG     200H           ;Địa chỉ lưu chương trình trong ROM.
MYDATA: ;Vùng dữ liệu tương ứng từ  $0^2 - 9^2$ .
        DB      0,1,4,9,16,25,36,49,64,81
        END

```

- Ví dụ 16: Viết đoạn lệnh tính tổng các giá trị của các ô nhớ RAM nội có địa chỉ 40H – 44H. Kết quả byte thấp cất vào thanh ghi A và byte cao cất vào thanh ghi B.

Giải

```

MOV     R0, #40H       ;Nạp địa chỉ bắt đầu chứa trong RAM.
MOV     R1, #5          ;Nạp giá trị bộ đếm (số lượng ô nhớ).
CLR     A               ;Xoá ACC, A = 0.
MOV     B, A            ;Xoá thanh ghi B, B = 0

LOOP:   ADD     A, @R0    ;Cộng nội dung ô nhớ do R0 trỏ đến vào A.
        JNC     NO_CY     ;Nhảy nếu không có nhớ, CY = 0.
        INC     B         ;Tăng thanh ghi B nếu có nhớ, CY = 1.

NO_CY:  INC     R0        ;Tăng con trỏ đến ô nhớ kế tiếp.
        DJNZ    R1, LOOP  ;Lặp lại cho đến khi bộ đếm = 0.

```

- Ví dụ 17: Viết đoạn lệnh tính tổng hai số 16 bit là 3CE7H và 3B8DH. Cất kết quả vào R7 (byte cao) và R6 (byte thấp).

Giải

```

CLR     C               ;Xoá cờ nhớ, CY = 0.
MOV     A, #0E7H        ;Nạp byte thấp 1 vào A, A = E7H.
ADD     A, #8DH          ;Cộng byte thấp 2 vào A, A = 74H, CY = 1.
MOV     R6, A            ;Lưu byte thấp của tổng vào R6.
MOV     A, #3CH          ;Nạp byte cao 1 vào A, A = 3CH.
ADDC    A, #3BH          ;Cộng có nhớ byte cao 2 vào A, A = 78H.
MOV     R7, A            ;Lưu byte cao của tổng vào R7.

```

- Ví dụ 18: Viết đoạn lệnh tính tổng của 5 dữ liệu BCD được lưu trong RAM nội tại địa chỉ bắt đầu từ 40H như sau: (40H) = 71H, (41H) = 11H, (42H) = 65H, (43H) = 59H, (44H) = 37H. Kết quả được lưu vào thanh ghi R7 (byte cao) và thanh ghi A (byte thấp) dưới dạng số BCD.

Giải

```

MOV    R0, #40H    ;Nạp địa chỉ bắt đầu chứa trong RAM.
MOV    R1, #5       ;Nạp giá trị bộ đếm (số lượng ô nhớ).
CLR    A            ;Xoá ACC, A = 0.
MOV    R7, A        ;Xoá thanh ghi R7, R7 = 0
LOOP:  ADD    A, @R0    ;Cộng nội dung ô nhớ do R0 trỏ đến vào A.
        DA    A        ;Hiệu chỉnh thành số BCD.
        JNC    NO_CY    ;Nhảy nếu không có nhớ, CY = 0.
        INC    R7      ;Tăng thanh ghi R7 nếu có nhớ, CY = 1.
NO_CY: INC    R0        ;Tăng con trỏ đến ô nhớ kế tiếp.
        DJNZ   R1, LOOP ;Lặp lại cho đến khi bộ đếm = 0.

```

- Ví dụ 19: Viết đoạn lệnh để nhận dữ liệu dưới dạng số HEX trong phạm vi 00H – FFH từ Port 1 và chuyển đổi về dạng thập phân. Lưu các số vào các thanh ghi R7 (LSB), R6 và R5 (MSB).

Giải

```

MOV    P1, #0FFH    ;Cấu hình P1 là port nhập.
MOV    A, P1        ;Đọc dữ liệu từ P1.
MOV    B, #10       ;Nạp B = 10.
DIV    AB           ;Chia cho 10, tách lấy số cao/số thấp.
MOV    R7, B        ;Lưu giá trị hàng đơn vị vào R7.
MOV    B, #10       ;Nạp B = 10.
DIV    AB           ;Chia cho 10, tách lấy số cao/số thấp.
MOV    R6, B        ;Lưu giá trị hàng chục vào R6.
MOV    R5, A        ;Lưu giá trị hàng trăm vào R5.

```

- Ví dụ 20: Viết đoạn lệnh đọc và kiểm tra Port 1 xem có chứa giá trị 45H hay không? Nếu (P1) = 45H thì xuất giá trị 99H ra Port 2, ngược lại thì thoát khỏi đoạn lệnh.

Giải

```

MOV    P2, #00H    ;Xoá P2, P2 = 0.
MOV    P1, #0FFH    ;Cấu hình P1 là port nhập.
MOV    R0, #45H    ;Nạp R0 = 45H, giá trị cần kiểm tra.
MOV    A, P1        ;Đọc dữ liệu từ P1.
XRL    A, R0        ;Kiểm tra dữ liệu có bằng 45H, nếu bằng thì
JNZ    EXIT        ;A = 0, không bằng thì A ≠ 0.
MOV    P2, #99H    ;Nạp P2 = 99H nếu P1 = 45H (A = 0).
EXIT:

```

- Ví dụ 21: Viết đoạn lệnh để lấy bù 2 giá trị chứa trong thanh ghi R0.

Giải

```

MOV    A, R0        ;Nạp dữ liệu cần lấy bù vào A.
CPL    A            ;Lấy bù 1.
ADD    A, #1        ;Cộng thêm 1 để được bù 2.

```

- Ví dụ 22: Viết đoạn lệnh kiểm tra thanh ghi A xem có chứa giá trị 99H hay không? Nếu (A) = 99H thì nạp giá trị FFH vào thanh ghi R1, ngược lại thì nạp giá trị 00H vào thanh ghi R1.

Giải

```

MOV    R1, #0           ;Nạp R0 = 00H.
CJNE   A, #99H, NEXT    ;So sánh A với giá trị 99H, nếu không
                        ;bằng thì nhảy đến NEXT.
MOV    R1, #0FFH        ;Nạp R0 = FFH nếu A = 99H.

```

NEXT:

- Ví dụ 23: Giả sử một cảm biến nhiệt được nối tới ngõ vào P1. Hãy viết đoạn lệnh đọc nhiệt độ và so sánh với giá trị 75. Dựa vào kết quả kiểm tra để đặt giá trị nhiệt độ vào các thanh ghi như sau:

Nếu $t = 75^{\circ}\text{C}$ thì (A) = 75.Nếu $t < 75^{\circ}\text{C}$ thì (R1) = t.Nếu $t > 75^{\circ}\text{C}$ thì (R2) = t.Giải

```

MOV    P1, #0FFH        ;Cấu hình P1 là port nhập.
MOV    A, P1             ;Đọc dữ liệu (t) từ P1.
CJNE   A, #75, OVER      ;So sánh dữ liệu (t) với giá trị 75, nếu
                        ;không bằng thì nhảy đến OVER.
SJMP   EXIT              ;Nếu A = 75 thì thoát chương trình.
OVER:   ;Trường hợp khi dữ liệu (t) khác 75.
JNC    NEXT              ;Nhảy đến NEXT nếu dữ liệu (t) > 75, C=0.
MOV    R1, A              ;Nếu dữ liệu (t) < 75 thì nạp dữ liệu vào
SJMP   EXIT              ;R1 (R1 = A = t) và thoát chương trình.
NEXT:   ;Trường hợp khi dữ liệu (t) > 75.
MOV    R2, A              ;Nạp dữ liệu vào R2 (R2 = A = t).
EXIT:

```

- Ví dụ 24: Viết đoạn lệnh liên tục kiểm tra giá trị tại Port 1 nếu giá trị này khác 63H. Nếu (P1) = 63H thì thoát đoạn lệnh không kiểm tra nữa.

Giải

```

MOV    P1, #0FFH        ;Cấu hình P1 là port nhập.
HERE:   MOV    A, P1      ;Đọc dữ liệu từ P1.
        CJNE   A, #63H, HERE ;Duy trì kiểm tra đến khi P1 = 63H.

```

- Ví dụ 25: Giả sử các ô nhớ trong RAM nội có địa chỉ từ 40H – 44H chứa nhiệt độ của các ngày được chỉ ra dưới đây. Hãy viết đoạn lệnh kiểm tra xem có giá trị nào bằng 65 không? Nếu có thì đặt địa chỉ của ô nhớ đó vào R4, ngược lại thì đặt R4 = 0.

(40H) = 76, (41H) = 79, (42H) = 69, (43H) = 65, (44H) = 64

Giải

```

MOV    R4, #0           ;Xoá R4 = 0.
MOV    R0, #40H         ;Nạp địa chỉ bắt đầu chứa trong RAM.
MOV    R1, #5            ;Nạp giá trị bộ đếm (số lượng ô nhớ).
MOV    A, #65           ;Nạp giá trị cần tìm vào A, A = 65.
BACK:
CJNE   A, @R0, NEXT     ;So sánh dữ liệu do R0 trỏ đến với 65.
MOV    R4, R0           ;Nếu bằng thì lưu địa chỉ ô nhớ đó vào R4.
SJMP   EXIT            ;Thoát chương trình.
NEXT:
        ;Trường hợp dữ liệu do R0 trỏ đến khác 65.
INC    R0              ;Tăng con trỏ đến ô nhớ kế tiếp.
DJNZ   R2, BACK        ;Lặp lại cho đến khi bộ đếm bằng 0.
EXIT:

```

- Ví dụ 26: Viết đoạn lệnh tìm tổng các chữ số 1 trong thanh ghi R0.

Giải

```

MOV    R1, #0           ;Xoá R1 = 0, lưu số chữ số 1.
MOV    R2, #8           ;Nạp giá trị bộ đếm (số bit kiểm tra).
MOV    A, R0            ;Nạp dữ liệu cần kiểm tra từ R0 vào A.
LOOP:
RLC    A               ;Xoay trái, đưa bit cần kiểm tra vào cờ CY.
JNC    NEXT           ;Kiểm tra cờ CY.
INC    R1             ;Tăng giá trị của R1 nếu cờ CY = 1.
NEXT:
DJNZ   R2, LOOP       ;Lặp lại cho đến khi bộ đếm bằng 0.

```

- Ví dụ 27: Viết đoạn lệnh để chuyển đổi số BCD nén chứa trong thanh ghi A thành hai số ASCII và chứa hai số này trong thanh ghi R2 và R3.

Giải

```

MOV    A, #29H         ;Nạp A = 29H, mã BCD nén của số 29.
MOV    R2, A           ;Lưu lại số BCD cần chuyển đổi trong R2.
ANL    A, #0FH         ;Xoá (che) 4 bit cao của số BCD (A = 09H).
ORL    A, #30H         ;Chuyển thành mã ASCII (A = 39H).
MOV    R3, A           ;Cất vào R3 (R3 = 39H – mã ASCII của 9).
MOV    A, R2           ;Lấy lại số BCD lúc ban đầu (A = 29H).
ANL    A, #0F0H        ;Xoá (che) 4 bit thấp của số BCD (A=20H).
SWAP   A               ;Hoán chuyển vị trí 4 bit cao vấp bit thấp.
ORL    A, #30H         ;Chuyển thành mã ASCII (A = 32H).
MOV    R2, A           ;Cất vào R2 (R2 = 32H – mã ASCII của 2).

```

• Ví dụ 28: Giả sử bit P2.3 là một đầu vào và biểu diễn nhiệt độ của một lò sấy. Nếu P2.3 = 1 có nghĩa là lò quá nóng. Hãy liên tục kiểm tra bit này, nếu nhiệt độ quá cao thì hãy gửi một xung mức cao đến P1.5 để bật còi báo hiệu.

Giải

JNB	P2.3, \$	;Duy trì kiểm tra đầu vào.
SETB	P1.5	;Khi P2.3 = 1, tạo mức cao cho P1.5 rồi
CLR	P1.5	;tạo mức thấp cho P1.5, phát một xung.

• Ví dụ 29: Viết đoạn lệnh kiểm tra xem thanh ghi A có chứa một số chẵn hay không? Nếu có thì chia cho 2, nếu không thì tăng lên 1 để chẵn hoá số đó rồi chia nó cho 2.

Giải

MOV	B, #2	;Gán số chia, B = 2.
JNB	ACC.0, OK	;Kiểm tra nếu là số chẵn thì nhảy đến OK.
INC	A	;Chẵn hoá nếu là số lẻ.
OK:		
DIV	AB	;Chia A cho 2.

V. PHẦN BÀI TẬP:

• **Truy xuất RAM nội (theo 2 cách: định địa chỉ ô nhớ trực tiếp và gián tiếp):**

Bài 1: Viết đoạn lệnh ghi (*chuyển*) giá trị 40H vào ô nhớ 30H của RAM nội.

Bài 2: Viết đoạn lệnh xóa nội dung ô nhớ 31H của RAM nội.

Bài 3: Viết đoạn lệnh ghi (*chuyển*) nội dung thanh ghi A vào ô nhớ 32H của RAM nội.

Bài 4: Viết đoạn lệnh ghi (*chuyển*) nội dung ô nhớ 33H của RAM nội vào thanh ghi A.

Bài 5: Viết đoạn lệnh ghi (*chuyển*) nội dung ô nhớ 34H của RAM nội vào ô nhớ 35H của RAM nội.

Bài 6: Viết đoạn lệnh ghi (*chuyển*) nội dung thanh ghi R4 vào ô nhớ 36H của RAM nội.

Bài 7: Viết đoạn lệnh ghi (*chuyển*) nội dung ô nhớ 37H của RAM nội vào thanh ghi R5.

Bài 8: Viết đoạn lệnh ghi (*chuyển*) nội dung thanh ghi A vào thanh ghi R1.

Bài 9: Viết đoạn lệnh ghi (*chuyển*) nội dung thanh ghi R2 vào thanh ghi A.

Bài 10: Viết đoạn lệnh ghi (*chuyển*) giá trị ABH vào thanh ghi A.

Bài 11: Viết đoạn lệnh ghi (*chuyển*) giá trị CDH vào thanh ghi R3.

• **Truy xuất RAM ngoài:**

Bài 1: Viết đoạn lệnh ghi (*chuyển*) giá trị 40H vào ô nhớ 30H của RAM ngoài (RAM ngoài có dung lượng ≤ 256 byte).

Bài 2: Viết đoạn lệnh xóa ô nhớ 31H của RAM ngoài (RAM ngoài có dung lượng ≤ 256 byte).

Bài 3: Viết đoạn lệnh ghi (*chuyển*) nội dung ô nhớ 32H của RAM ngoài vào thanh ghi A (RAM ngoài có dung lượng ≤ 256 byte).

Bài 4: Viết đoạn lệnh ghi (*chuyển*) nội dung thanh ghi A vào ô nhớ 33H của RAM ngoài (RAM ngoài có dung lượng ≤ 256 byte).

Bài 5: Viết đoạn lệnh chuyển dữ liệu ô nhớ 34H của RAM ngoài vào ô nhớ 35H của RAM ngoài (RAM ngoài có dung lượng ≤ 256 byte).

Bài 6: Viết đoạn lệnh ghi (*chuyển*) giá trị 40H vào ô nhớ 1230H của RAM ngoài (*RAM ngoài có dung lượng > 256 byte*).

Bài 7: Viết đoạn lệnh xóa ô nhớ 1231H của RAM ngoài (*RAM ngoài có dung lượng > 256 byte*).

Bài 8: Viết đoạn lệnh ghi (*chuyển*) nội dung ô nhớ 1232H của RAM ngoài vào thanh ghi A (*RAM ngoài có dung lượng > 256 byte*).

Bài 9: Viết đoạn lệnh ghi (*chuyển*) nội dung thanh ghi A vào ô nhớ 1233H của RAM ngoài (*RAM ngoài có dung lượng > 256 byte*).

Bài 10: Viết đoạn lệnh chuyển dữ liệu ô nhớ 1234H của RAM ngoài vào ô nhớ 1235H của RAM ngoài (*RAM ngoài có dung lượng > 256 byte*).

- **Truy xuất Port:**

Bài 1: Viết đoạn lệnh xuất (*ghi*) giá trị 0FH ra Port 1.

Bài 2: Viết đoạn lệnh xuất (*ghi*) giá trị F0H ra Port 2.

Bài 3: Viết đoạn lệnh xuất (*ghi*) nội dung thanh ghi A ra Port 1.

Bài 4: Viết đoạn lệnh nhập (*đọc*) từ Port 1 vào thanh ghi A.

Bài 5: Viết đoạn lệnh nhập (*đọc*) từ Port 1 và xuất ra Port 2.

Bài 6: Viết đoạn lệnh xuất (*ghi*) nội dung ô nhớ 37H của RAM nội ra Port 3.

Bài 7: Viết đoạn lệnh nhập (*đọc*) từ Port 2 vào ô nhớ 38H của RAM nội.

Bài 8: Viết đoạn lệnh xuất mức 1 (*mức logic cao*) ra chân P1.0

Bài 9: Viết đoạn lệnh xuất mức 0 (*mức logic thấp*) ra chân P1.1

- **Truy xuất RAM nội, RAM ngoài và Port:**

Bài 1: Viết đoạn lệnh chuyển ô nhớ 40H (*RAM nội*) vào ô nhớ 2000H (*RAM ngoài*).

Bài 2: Viết đoạn lệnh chuyển nội dung ô nhớ 2001H (*RAM ngoài*) vào ô nhớ 41H (*RAM nội*).

Bài 3: Viết đoạn lệnh nhập (*đọc*) từ Port 1 vào ô nhớ 42H (*RAM nội*).

Bài 4: Viết đoạn lệnh nhập (*đọc*) từ Port 1 vào ô nhớ 2002H (*RAM ngoài*).

Bài 5: Viết đoạn lệnh xuất (*ghi*) nội dung ô nhớ 43H (*RAM nội*) ra Port 1.

Bài 6: Viết đoạn lệnh xuất (*ghi*) nội dung ô nhớ 2003H (*RAM ngoài*) ra Port 1.

- **Sử dụng vòng lặp:**

Bài 1: Viết đoạn lệnh xóa 20 ô nhớ RAM nội có địa chỉ bắt đầu là 30H.

Bài 2: Viết đoạn lệnh xóa các ô nhớ RAM nội từ địa chỉ 20H đến 7FH.

Bài 3: Viết đoạn lệnh xóa 250 ô nhớ RAM ngoài có địa chỉ bắt đầu là 4000H.

Bài 4: Viết đoạn lệnh xóa 2500 ô nhớ RAM ngoài có địa chỉ bắt đầu là 4000H.

Bài 5: Viết đoạn lệnh xóa các ô nhớ RAM ngoài từ địa chỉ 2000H đến 205FH.

Bài 6: Viết đoạn lệnh xóa các ô nhớ RAM ngoài từ địa chỉ 2000H đến 3FFFH.

Bài 7: Viết đoạn lệnh xóa toàn bộ RAM ngoài có dung lượng 8KB, biết rằng địa chỉ đầu là 2000H.

Bài 8: Viết đoạn lệnh chuyển một chuỗi dữ liệu gồm 10 byte trong RAM nội có địa chỉ đầu là 30H đến vùng RAM nội có địa chỉ đầu là 40H.

Bài 9: Viết đoạn lệnh chuyển một chuỗi dữ liệu gồm 100 byte trong RAM ngoài có địa chỉ đầu là 2000H đến vùng RAM ngoài có địa chỉ đầu là 4000H.

Bài 10: Viết đoạn lệnh chuyển một chuỗi dữ liệu gồm 1000 byte trong RAM ngoài có địa chỉ đầu là 2000H đến vùng RAM ngoài có địa chỉ đầu là 4000H.

Bài 11: Viết đoạn lệnh chuyển một chuỗi dữ liệu gồm 10 byte trong RAM nội có địa chỉ đầu là 30H đến vùng RAM ngoài có địa chỉ đầu là 4000H.

Bài 12: Viết đoạn lệnh chuyển một chuỗi dữ liệu gồm 10 byte trong RAM ngoài có địa chỉ đầu là 5F00H đến vùng RAM nội có địa chỉ đầu là 40H.

Bài 13: Cho một chuỗi dữ liệu gồm 20 byte liên tiếp trong RAM nội, bắt đầu từ địa chỉ 20H. Hãy viết đoạn lệnh lần lượt xuất các dữ liệu này ra Port 1.

Bài 14: Giả sử Port 1 được nối đến một thiết bị phát dữ liệu (ví dụ như 8 nút nhấn). Hãy viết đoạn lệnh nhận liên tiếp 10 byte dữ liệu từ thiết bị phát này và ghi vào 10 ô nhớ (RAM nội) liên tiếp bắt đầu từ ô nhớ 50H.

• **Tạo trễ:**

Bài 1: Viết chương trình con delay 100s, biết rằng f_{OSC} dùng trong hệ thống là:

- a. 6 MHz.
- b. 11,0592 MHz.
- c. 12 MHz.
- d. 24 MHz

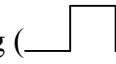
Bài 2: Viết chương trình con delay 100ms, biết rằng f_{OSC} dùng trong hệ thống là:

- a. 6 MHz.
- b. 11,0592 MHz.
- c. 12 MHz.
- d. 24 MHz

Bài 3: Viết chương trình con delay 1s, biết rằng f_{OSC} dùng trong hệ thống là:

- a. 6 MHz.
- b. 11,0592 MHz.
- c. 12 MHz.
- d. 24 MHz

• **Tạo xung:**

Bài 1: Viết đoạn lệnh tạo một xung dương () tại chân P1.0 với độ rộng xung 1ms, biết rằng $f_{OSC} = 12 \text{ MHz}$.

Bài 2: Viết đoạn lệnh tạo chuỗi xung vuông có $f = 100 \text{ KHz}$ tại chân P1.1 ($f_{OSC} = 12 \text{ MHz}$).

Bài 3: Viết đoạn lệnh tạo chuỗi xung vuông có $f = 100 \text{ KHz}$ và có chu kỳ làm việc $D = 40\%$ tại chân P1.2 ($f_{OSC} = 12 \text{ MHz}$).

Bài 4: Viết đoạn lệnh tạo chuỗi xung vuông có $f = 10 \text{ KHz}$ tại chân P1.3 ($f_{OSC} = 24 \text{ MHz}$).

Bài 5: Viết đoạn lệnh tạo chuỗi xung vuông có $f = 10 \text{ KHz}$ và có chu kỳ làm việc $D = 30\%$ tại chân P1.3 ($f_{OSC} = 24 \text{ MHz}$).

Bài 6: Viết đoạn lệnh tạo chuỗi xung vuông có $f = 10 \text{ Hz}$ tại chân P1.4 ($f_{OSC} = 12 \text{ MHz}$).

Bài 7: Viết đoạn lệnh tạo chuỗi xung vuông có $f = 10 \text{ Hz}$ và có chu kỳ làm việc $D = 25\%$ tại chân P1.5 ($f_{OSC} = 11,0592 \text{ MHz}$).

• **Các phép toán:**

Bài 1: Cho một chuỗi số 8 bit không dấu trong RAM nội gồm 10 số bắt đầu từ ô nhớ 30H. Hãy viết chương trình con cộng chuỗi số này và ghi kết quả vào ô nhớ 2FH trong RAM nội (giả sử kết quả nhỏ hơn hoặc bằng 255).

Bài 2: Cho một chuỗi số 8 bit không dấu trong RAM nội gồm 10 số bắt đầu từ ô nhớ 30H. Hãy viết chương trình con cộng chuỗi số này và ghi kết quả vào hai ô nhớ 2EH:2FH trong RAM nội (ô nhớ 2EH chứa byte cao của kết quả và ô nhớ 2FH chứa byte thấp của kết quả).

Bài 3: Cho một chuỗi số 16 bit không dấu trong RAM nội gồm 10 số bắt đầu từ ô nhớ 30H theo nguyên tắc ô nhớ có địa chỉ nhỏ hơn chứa byte cao và ô nhớ có địa chỉ lớn hơn chứa byte thấp. (Ví dụ: byte cao của số 16 bit đầu tiên được cất tại ô nhớ 30H và byte thấp của số 16 bit đầu tiên được cất tại ô nhớ 31H; byte cao của số 16 bit thứ hai được cất tại ô nhớ 32H và byte thấp của số 16 bit thứ hai được cất tại ô nhớ 33H). Hãy viết chương trình con cộng chuỗi số này và cất kết quả vào hai ô nhớ 2EH:2FH trong RAM nội.

Bài 4: Tương tự như các bài 1, 2, 3 nhưng thực hiện đối với phép trừ.

Bài 5: Viết chương trình con lấy bù 2 số 16 bit chứa trong hai thanh ghi R2:R3.

• **So sánh:**

Bài 1: Cho hai số 8 bit, số thứ nhất chứa trong ô nhớ 30H, số thứ hai chứa trong ô nhớ 31H. Viết chương trình con so sánh hai số này. Nếu số thứ nhất lớn hơn hoặc bằng số thứ hai thì set cờ F0, nếu ngược lại thì xóa cờ F0.

Bài 2: Cho hai số 16 bit, số thứ nhất chứa trong hai ô nhớ 30H:31H, số thứ hai chứa trong hai ô nhớ 32H:33H. Viết chương trình con so sánh hai số này. Nếu số thứ nhất lớn hơn hoặc bằng số thứ hai thì set cờ F0, nếu ngược lại thì xóa cờ F0.

Bài 3: Cho một chuỗi ký tự dưới dạng mã ASCII trong RAM nội, dài 20 byte, bắt đầu từ địa chỉ 50H. Viết đoạn lệnh xuất các ký tự in hoa có trong chuỗi này ra Port 1. Biết rằng mã ASCII của ký tự in hoa là từ 65H (chữ A) đến 90H (chữ Z).

Bài 4: Viết đoạn lệnh nhập một chuỗi ký tự từ Port 1 dưới dạng mã ASCII và ghi vào RAM ngoài, bắt đầu từ địa chỉ 0000H. Biết rằng chuỗi này kết thúc bằng ký tự CR (có mã ASCII là 0DH) và ghi cả ký tự này vào RAM.

Bài 5: Viết đoạn lệnh nhập một chuỗi ký tự từ Port 1 dưới dạng mã ASCII và ghi vào RAM ngoài, bắt đầu từ địa chỉ 0000H. Biết rằng chuỗi này kết thúc bằng ký tự CR (có mã ASCII là 0DH) và không ghi ký tự này vào RAM.

Bài 6: Viết đoạn lệnh nhập một chuỗi ký tự từ Port 1 dưới dạng mã ASCII và ghi vào RAM ngoài, bắt đầu từ địa chỉ 0000H. Biết rằng chuỗi này kết thúc bằng ký tự CR (có mã ASCII là 0DH) và không ghi ký tự này vào RAM mà thay bằng ký tự NULL (có mã ASCII là 00H).

Bài 7: Cho một chuỗi ký tự dưới dạng mã ASCII trong RAM nội, dài 20 byte, bắt đầu từ địa chỉ 50H. Viết đoạn lệnh đổi các ký tự in hoa có trong chuỗi này thành ký tự thường. Biết rằng mã ASCII của ký tự thường bằng mã ASCII của ký tự in hoa cộng thêm 32H.

Bài 8: Cho một chuỗi ký tự số dưới dạng mã ASCII trong RAM nội, dài 20 byte, bắt đầu từ địa chỉ 50H. Viết đoạn lệnh đổi các ký tự số này thành mã BCD. Biết rằng mã ASCII của các ký tự số là từ 30H (số 0) đến 39H (số 9).

• **Sử dụng lệnh nhảy có điều kiện:**

Bài 1: Cho một chuỗi dữ liệu dưới dạng số có dấu trong RAM ngoài, dài 100 byte, bắt đầu từ địa chỉ 0100H. Viết đoạn lệnh lần lượt xuất các dữ liệu trong chuỗi ra Port 1 nếu là số dương (xem số 0 là dương) và xuất ra Port 2 nếu là số âm.

Bài 2: Cho một chuỗi dữ liệu dưới dạng số có dấu trong RAM ngoài, bắt đầu từ địa chỉ 0100H và kết thúc bằng số 0. Viết đoạn lệnh lần lượt xuất các dữ liệu trong chuỗi ra Port 1 nếu là số dương và xuất ra Port 2 nếu là số âm.

Bài 3: Cho một chuỗi dữ liệu dưới dạng số không dấu trong RAM ngoài, bắt đầu từ địa chỉ 0100H và độ dài chuỗi là nội dung ô nhớ 00FFH. Viết đoạn lệnh đếm số số chẵn (*chia hết cho 2*) có trong chuỗi và cất vào ô nhớ 00FEH.

Bài 4: Cho một chuỗi dữ liệu dưới dạng số không dấu trong RAM ngoài, bắt đầu từ địa chỉ 0100H và độ dài chuỗi là nội dung ô nhớ 00FFH. Viết đoạn lệnh ghi các số chẵn (*xem số 0 là số chẵn*) có trong chuỗi vào RAM nội bắt đầu từ địa chỉ 30H cho đến khi gặp số lẻ thì dừng.

Bài 5: Viết chương trình con có nhiệm vụ lấy một byte từ một chuỗi data gồm 20 byte cất trong RAM ngoài bắt đầu từ địa chỉ 2000H và xuất ra Port1. Mỗi lần gọi chương trình con chỉ xuất một byte, lần gọi kế thì xuất byte kế tiếp, lần gọi thứ 21 thì lại xuất byte đầu, ...