

CHƯƠNG 3: TẬP LỆNH AVR

3.1 GIỚI THIỆU CHUNG

Chương này giới thiệu tập lệnh(instruction set) của họ AVR nói chung thông qua việc trình bày các phương pháp định địa chỉ (addressing mode) và các hoạt động thực hiện cho từng lệnh cụ thể. Với mỗi chip vi điều khiển khác nhau thuộc họ AVR sẽ có một chút khác biệt có thể là về phần cứng cũng như tập lệnh đi kèm, do đó khi sử dụng một chip cụ thể nào chúng ta cần phải tham khảo datasheet tương ứng. Trong giáo trình này sẽ đề cập đến chip ATmega324P, người đọc có thể tham khảo nhanh trong các phần phụ lục 2 và 3.

Phần phụ lục 2 trình bày bảng tóm tắt tập lệnh ATmega324P. Phụ lục 3 sẽ giải thích chi tiết cho từng lệnh một: mô tả hoạt động của lệnh, cú pháp, số byte lệnh, sự ảnh hưởng đến các cờ và số chu kỳ máy cần thiết để thực thi lệnh.

Các lệnh của vi điều khiển AVR được truy xuất theo từ(word), hầu hết có mã máy dài 2byte lệnh, một số lệnh dài 4byte. Thời gian thực thi cho mỗi lệnh đa số là 1,2 chu kỳ máy và một số lệnh 3,4 chu kỳ máy.

Một từ (2 byte) mã máy(opcode) được cắt trong bộ nhớ chương trình(Flash ROM) theo kiểu little endian, byte cao địa chỉ cao, byte thấp địa chỉ thấp.

Trong chương này có sử dụng một số ký hiệu theo quy định của trình biên dịch hợp ngữ (Assembler) cho họ AVR để minh họa các ví dụ cho hoàn chỉnh và dễ hiểu hơn. Trong chương 4 sẽ trình bày chi tiết các ký hiệu quy định cho trình hợp ngữ.

3.2. CÁC PHƯƠNG PHÁP ĐỊNH VỊ ĐỊA CHỈ

Các kiểu định địa chỉ là phần cần thiết cho toàn bộ tập lệnh của mỗi bộ vi xử lý, vi điều khiển. Nó cho phép ta xác định rõ nguồn và đích của dữ liệu theo nhiều phương pháp khác nhau phụ thuộc vào tình huống lập trình.

Tập lệnh của AVR hỗ trợ rất mạnh cho việc truy xuất dữ liệu từ bộ nhớ chương trình(Flash) cũng như bộ nhớ dữ liệu bao gồm RAM nội(SRAM), các thanh ghi đa dụng(GPRs), các thanh ghi giao tiếp ngoại vi hay thanh ghi I/O(IORs).

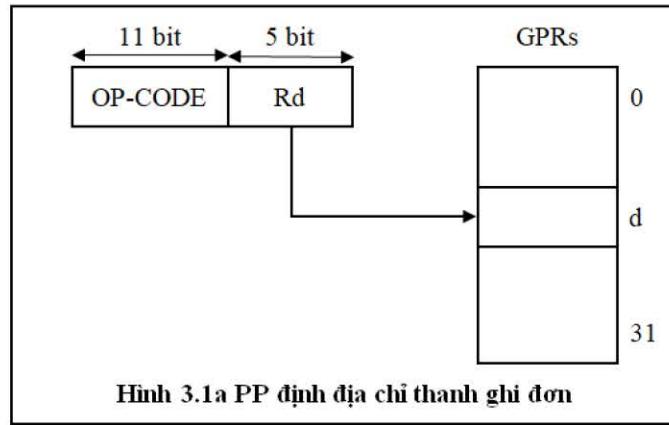
Tổng cộng có 15 phương pháp định địa chỉ khác nhau. Có thể chia thành 8 nhóm như sau:

- Thanh ghi đơn (tức thời).
- Thanh ghi.
- Dữ liệu trực tiếp.
- Dữ liệu gián tiếp.
- FLASH trực tiếp.
- FLASH gián tiếp.
- Định vị tương đối
- Định vị BIT

3.2.1 Định địa chỉ thanh ghi đơn(tức thời):

Phương pháp này gồm có 2 kiểu tùy theo các toán hạng được sử dụng trong lệnh: hoặc toán hạng chỉ là một thanh ghi đơn, hay 1 thanh ghi với 1 giá trị tức thời.

Đối với phương pháp có toán hạng chỉ là một thanh ghi đơn sẽ dùng 11bit cho OPCODE và 5 bit để mã hóa cho vị trí của Rd($0 \leq d \leq 31$). Hình 3.1a và 3.1b trình bày việc phân bổ số lượng bit để mã hóa cho một lệnh và trật tự sắp xếp các bit mã máy của một từ lệnh cho phương pháp này.



Ví dụ về một trong các câu lệnh sử dụng phương pháp định địa chỉ thanh ghi đơn là lệnh tăng nội dung của các thanh ghi R2 và R16 lên 1 như sau:

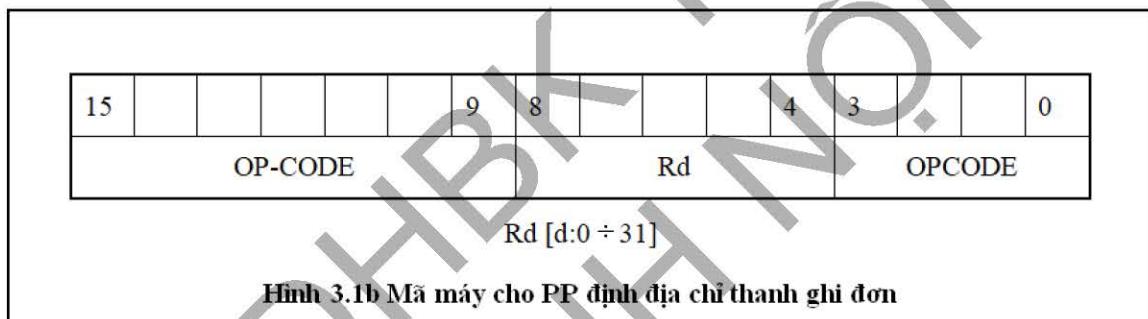
INC R2

INC R16

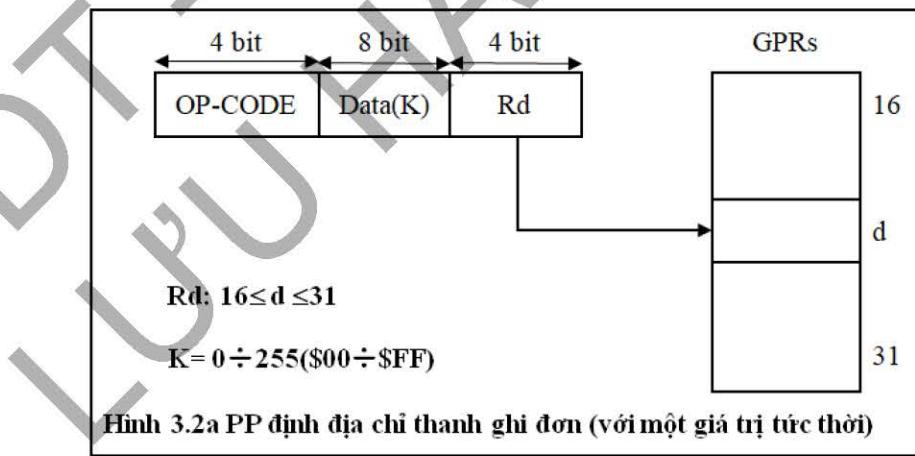
Lệnh này có OP-CODE là **1001 010x xxxx 0011B**. Năm bit còn lại để chỉ ra thanh ghi R2 hoặc R16 được truy xuất đến. Khi biên dịch ta có mã máy tương ứng của 2 lệnh trên lần lượt là:

0x9423 [0b**1001 0100 0010 0011**]

0x9503 [0b**1001 0101 0000 0011**]



Phương pháp định địa chỉ thanh ghi đơn với một giá trị tức thời sử dụng 4bit cho OPCODE, 4bit cho Rd và 8bit cho dữ liệu được minh họa ở hình 3.2a và 3.2b.



15						8	7			4	3			0	
OP-CODE				K7 ÷ K4				Rd				K3 ÷ K0			

Rd[d:16 ÷ 31], K[0 ÷ 255]

Hình 3.2b Mã máy cho PP định địa chỉ thanh ghi đơn với một giá trị tức thời K

Một ví dụ cho phương pháp định địa chỉ thanh ghi đơn với một giá trị tức thời là lệnh lưu các giá trị 200 và 0xAF lần lượt vào 2 thanh ghi R16 và R28 như sau:

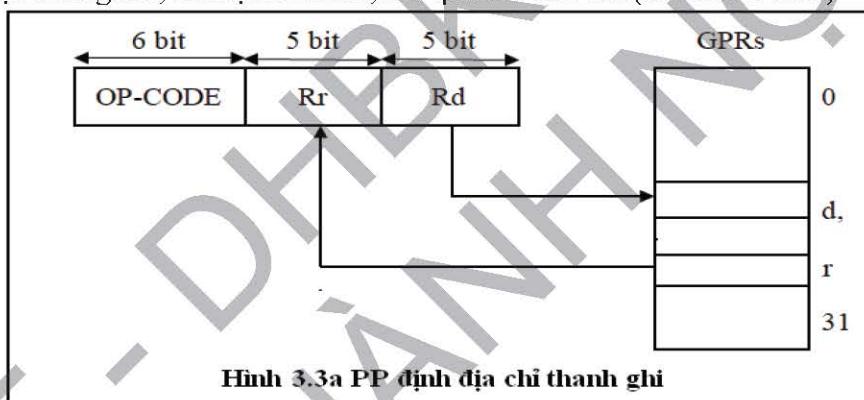
LDI R16,200

LDI R28,0xAF

Câu lệnh này có OP-CODE là 1110B, chỉ được phép sử dụng với các thanh ghi từ R16÷R31 được mã hóa bằng 4 bit nhị phân có giá trị tương ứng từ 0÷15. Ta có mã máy của 2 lệnh trên lần lượt là: 0xEC08 [0b1110 1100 0000 1000] 0xEACF[0b1110 1010 1100 1111]

3.2.2 Định địa chỉ thanh ghi:

Phương pháp định địa chỉ thanh ghi sử dụng trong câu lệnh có 2 toán hạng là các thanh ghi Rd và Rr(GPRs). Thực hiện các thao tác lệnh trên các dữ liệu được đặt trong 2 thanh ghi Rd và Rr, quy ước Rr là địa chỉ nguồn, Rd địa chỉ đích, kết quả cát vào Rd.(xem hình 3.3a)



Hình 3.3a PP định địa chỉ thanh ghi

Mã máy của lệnh được xác định như hình 3.3b, ký hiệu Rri và Rdi lần lượt là các bit trọng số i của Rr và Rd.

15					10	9	8	7			4	3			0
OP-CODE					r4	d4	d3	d2	d1	d0	r3	r2	r1	r0	
Rri					Rdi				Rri						

Rd[d:0 ÷ 31], Rr[r:0 ÷ 31]

Hình 3.3b Mã máy cho PP định địa chỉ thanh ghi

Quan sát 2 lệnh có phương pháp định địa chỉ thanh ghi sau:

MOV R2,R3

ADD R1,R16

Lệnh MOV có OP-CODE là 001011B, Rd=R2=00010B, Rr=R3=00011.

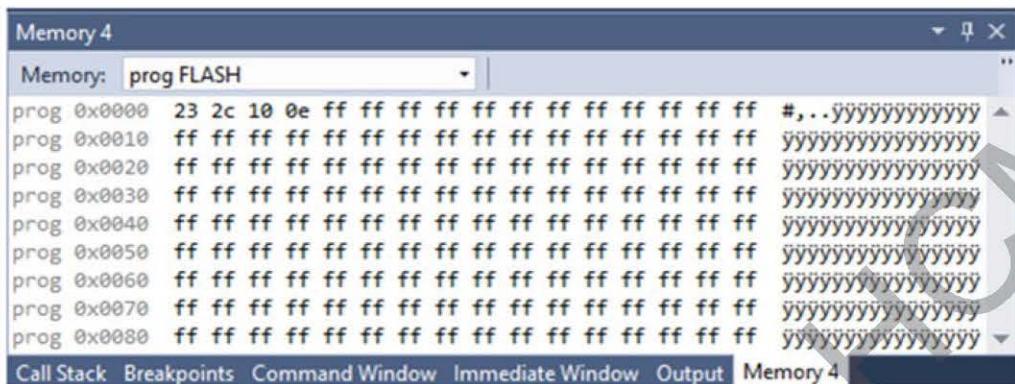
Lệnh ADD R1,R16 có OP-CODE là 000011B, Rd=R1=00001B, Rr=R16=10000B.

Theo nguyên tắc xác định mã máy ở trên, ta có mã máy tương ứng của 2 lệnh này lần lượt là:

0x2C23 [0b0010 1100 0010 0011]

0xE10 [0b0000 1110 0001 0000]

Có thể kiểm chứng trên phần mềm mô phỏng như ở hình 3.4. bằng cách quan sát bộ nhớ Code(Flash).

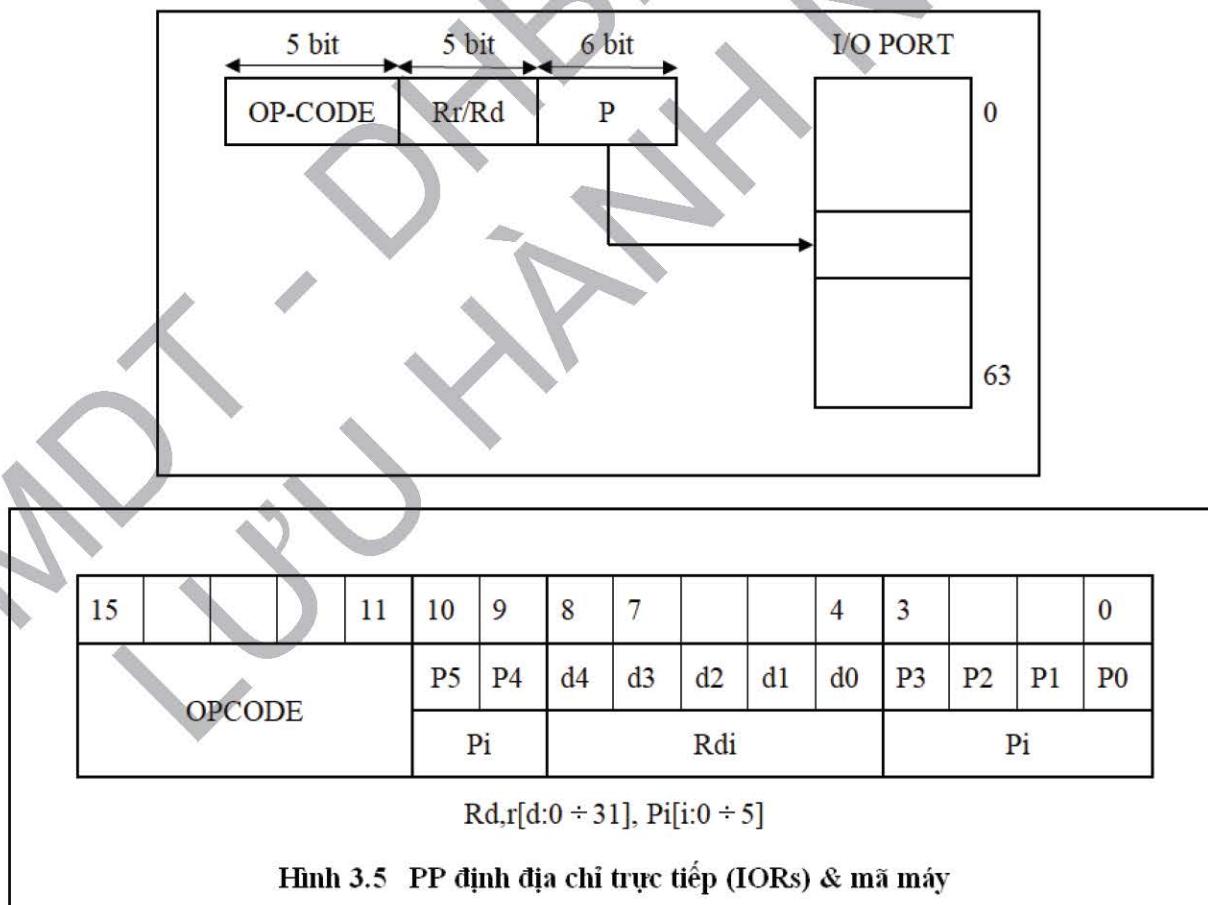


Hình 3.4 Mã máy của các lệnh MOV R0,R2 và ADD R1,R16

3.2.3 Định địa chỉ trực tiếp:

Kiểu định địa chỉ trực tiếp dùng để truy xuất đến các IORs(các thanh ghi I/O) và vùng SRAM(data memory).

Ký hiệu P trong tập lệnh thực hiện trên các toán hạng là các IORs, k là các ô nhớ thuộc vùng SRAM. Trong đó: $0 \leq P \leq 63$, và $0 \leq k \leq 65535$. Câu lệnh được mã hóa như hình 3.5 và 3.6.



Hình 3.5 PP định địa chỉ trực tiếp (IORs) & mã máy

Đối với các IORs, ta không nhất thiết phải biết địa chỉ I/O của các thanh ghi này, trình dịch hợp ngữ cho phép sử dụng mã gọi nhớ là tên tương ứng của port hoặc tên của thanh ghi chức năng để thực hiện lệnh. Ví dụ như dùng mã gọi nhớ là PORTA để truy xuất cho địa chỉ I/O của portA

hay TCNTn để làm việc với các thanh ghi điều khiển Timer tương ứng. Chẳng hạn như muốn lưu dữ liệu từ thanh ghi R1 vào portA ta có thể viết lệnh:

OUT PORTA,R1

thay cho lệnh:

OUT \$02,R1

vì địa chỉ I/O của Port A là \$02=0x02.

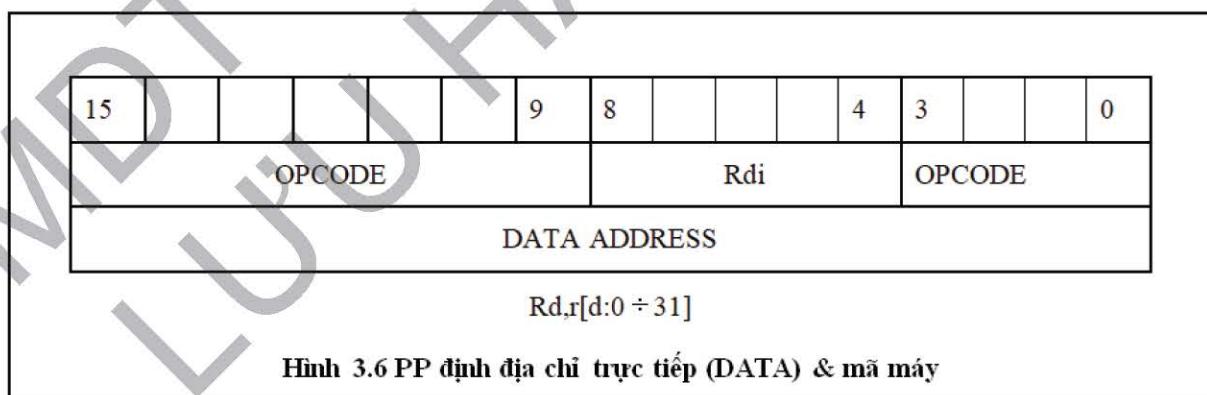
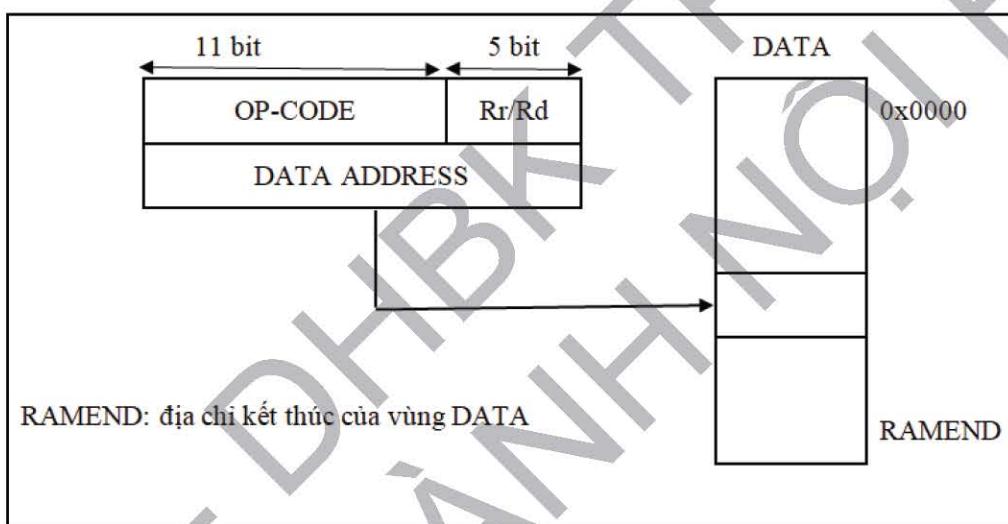
Mỗi một IORs trong không gian bộ nhớ dữ liệu sẽ có 2 địa chỉ tương ứng đó là địa chỉ I/O và địa chỉ dữ liệu. Do đó, khi làm việc với các IOxRs ta có thể dùng một trong 2 phương pháp định địa chỉ khác nhau để thực hiện lệnh. Với mỗi phương pháp khác nhau dẫn đến sự khác nhau về kích thước của lệnh. Một ví dụ để thấy rằng cũng đồng thời là việc thực hiện đọc dữ liệu từ PORTB, ta có thể viết 1 trong 2 lệnh sau sẽ cho một kết quả tương tự:

IN R3,PINB (hoặc IN R3, \$03)

LDS R3,\$23

Nhưng với lệnh thứ nhất, sử dụng phương pháp định địa chỉ là I/O sẽ cho mã máy có chiều dài 2 byte: 0xB033, còn phương pháp sau là: 0x9030 0023 – sẽ tương ứng với 4 byte lệnh.

- Lưu ý: Với họ AVR như ATmega324P có các thanh ghi I/O mở rộng(ExIORs) có địa chỉ dữ liệu(SRAM) từ 0X60 trở lên chỉ sử dụng kiểu định vị trực tiếp theo địa chỉ dữ liệu!(xem phần tập lệnh)



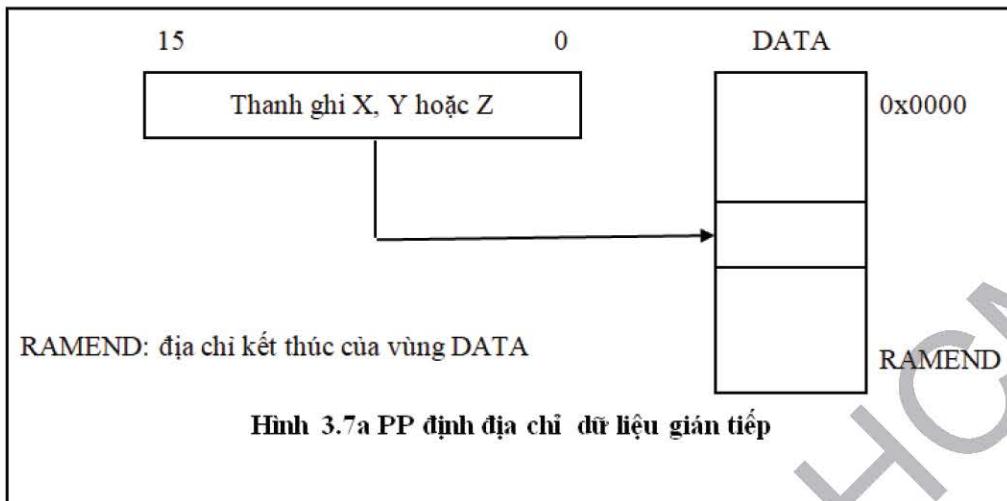
Hình 3.6 PP định địa chỉ trực tiếp (DATA) & mã máy

3.2.4 Định địa chỉ gián tiếp qua thanh ghi:

Phương pháp định địa chỉ gián tiếp qua thanh ghi thực hiện trên các lệnh dùng để truy xuất dữ liệu vùng SRAM với các thanh ghi GPRs gián tiếp qua các thanh ghi con trỏ X, Y hoặc Z.

Phương pháp này được chia ra làm 4 kiểu khác nhau: dữ liệu gián tiếp, dữ liệu gián tiếp với tiền tố trừ(-), dữ liệu gián tiếp với hậu tố cộng(+) và dữ liệu gián tiếp với một độ chuyển dời.

3.2.4.1 Dữ liệu gián tiếp.



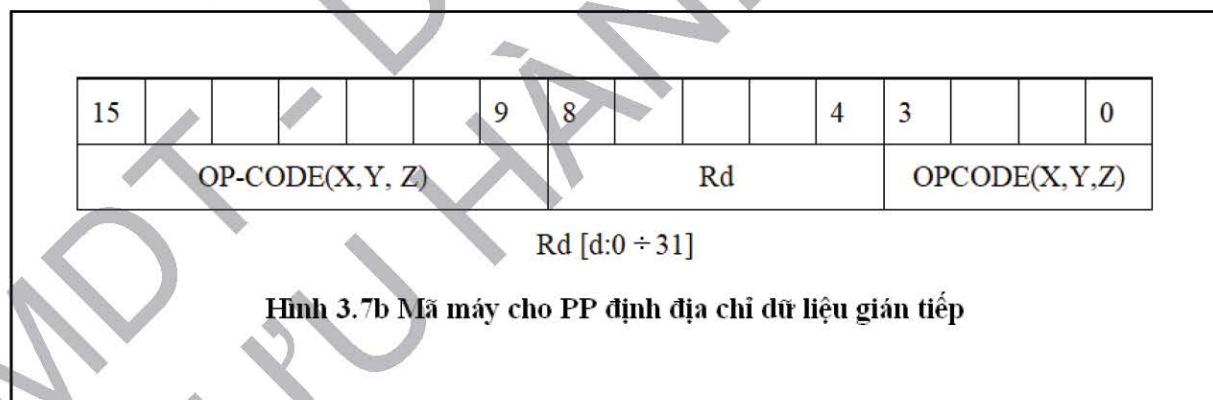
Hình 3.7a mô tả cho phương pháp định địa chỉ dữ liệu gián tiếp. Các thanh ghi X, Y và Z có thể hoạt động như các con trỏ(pointer) và nội dung của các thanh ghi này là địa chỉ của các ô nhớ nằm trong vùng SRAM, nơi mà dữ liệu sẽ được đọc hoặc là ghi. Mã máy của phương pháp này được trình bày ở hình 3.7b.

Để thực hiện việc ghi giá trị 0xCC vào địa chỉ 0x0800 cùng với việc chọn thanh ghi Y làm con trỏ trong phương pháp định địa chỉ này, ta thực hiện các lệnh sau:

```
LDI R16,0xCC ;lưu dữ liệu cần ghi vào R16
LDI R28,0x00 ;lưu địa chỉ ô nhớ vào Y
LDI R29,0X08 ;Y ≡ R29:R28
ST Y, R16 ;ghi dữ liệu
```

Và để đọc lại dữ liệu này rồi lưu vào thanh ghi R7 trong trường hợp chọn thanh ghi X làm con trỏ, ta cần thực hiện một số lệnh sau:

```
LDI R26,0x00 ;lưu địa chỉ ô nhớ vào thanh ghi X
LDI R27,0x08 ;X ≡ R27:R26
LD R7,X
```



Các lệnh ST Y,R16 và LD R7, X lần lượt có OP-CODE tương ứng là:

0b1000 001x xxxx 1000

0b1001 000x xxxx 1100

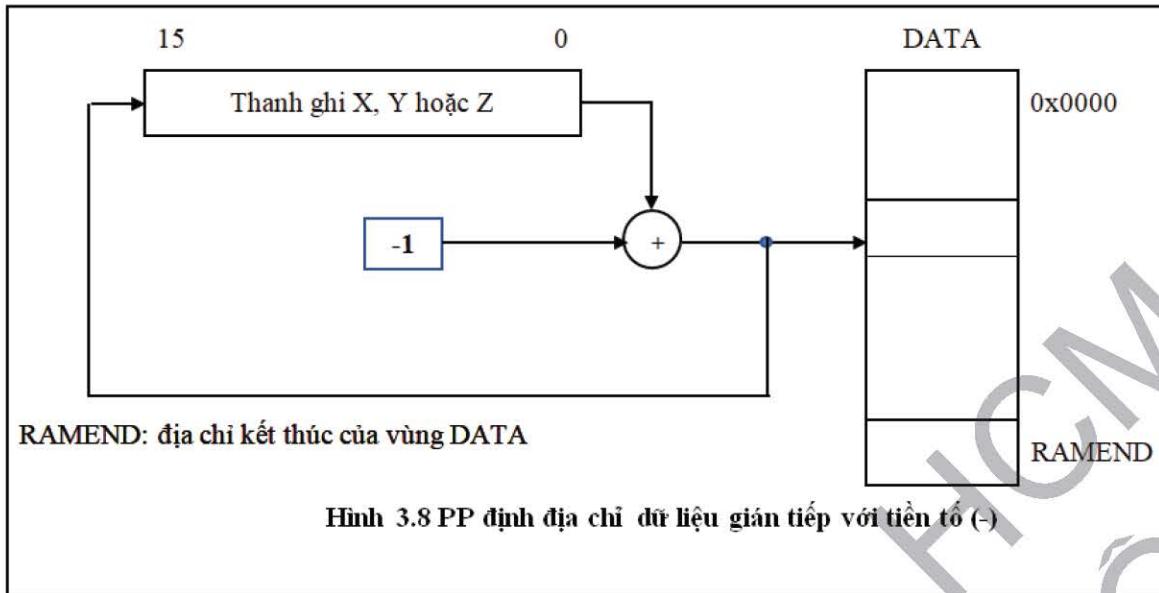
Vì vậy mã máy của 2 lệnh trên sẽ là:

0x8308 [0b1000 0011 0000 1000]

0x907C [0b1001 0000 0111 1100]

3.2.4.2 Dữ liệu gián tiếp với tiền tố trừ (-).

Phương pháp định địa chỉ kiểu dữ liệu gián tiếp với 1 tiền tố trừ (-) thực hiện trong câu lệnh có các toán hạng là các thanh ghi con trỏ với ký hiệu dấu trừ phía trước(-X, -Y, -Z).



Hình 3.8 PP định địa chỉ dữ liệu gián tiếp với tiền tố (-)

Địa chỉ của các toán hạng được xác định bằng cách lấy nội dung của các con trỏ trừ đi 1 trước khi thực hiện lệnh, và kết quả này sẽ giữ nguyên trong con trỏ sau khi thực hiện xong lệnh. Hình 3.8 trình bày các xác định địa chỉ của toán hạng. Việc xác định mã máy tương tự như phương pháp dữ liệu gián tiếp.

Bảng sau liệt kê mã máy của lệnh LD theo phương pháp gián tiếp với tiền tố (-) tương ứng với các thanh ghi X, Y và Z.

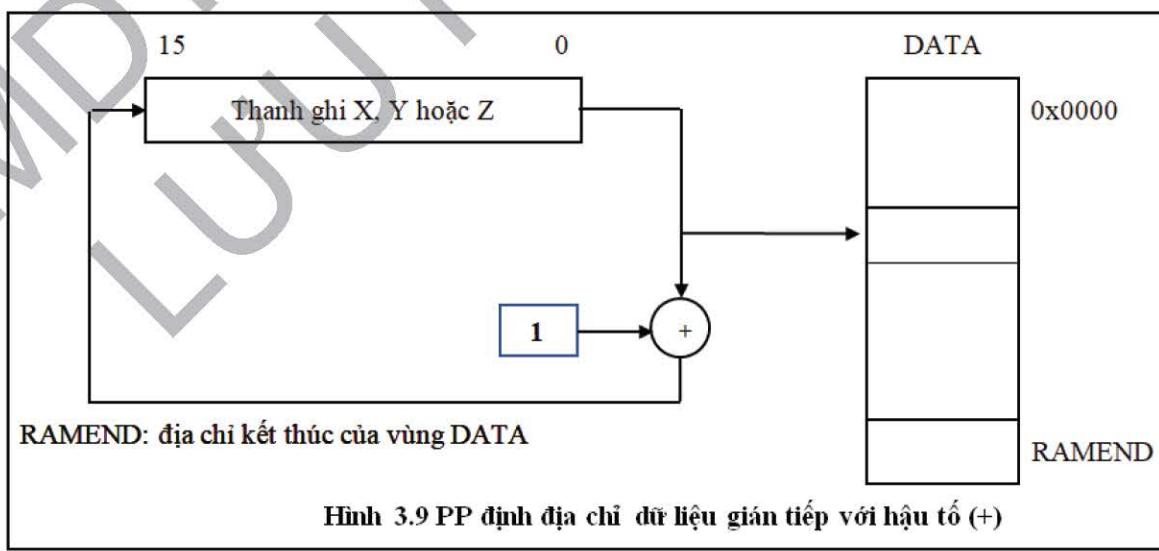
LỆNH	MÃ MÁY
LD Rd, -X	1001 000d ₄ d ₃ d ₂ d ₁ d ₀ 1110
LD Rd, -Y	1001 000d ₄ d ₃ d ₂ d ₁ d ₀ 1010
LD Rd, -Z	1001 000d ₄ d ₃ d ₂ d ₁ d ₀ 0010

Với d_i là giá trị nhị phân tương ứng với chỉ số d của thanh ghi được sử dụng trong lệnh.

Ví dụ lệnh LD R20,-X sẽ có mã máy là 1001 0001 0100 1110.

3.2.4.3 Dữ liệu gián tiếp với hậu tố cộng (+).

Phương pháp này thực hiện xong lệnh rồi tăng nội dung con trỏ lên 1. Ký hiệu được dùng trong tập lệnh là X+, Y+ hoặc Z+. Mã máy cũng tương tự như phương pháp dữ liệu gián tiếp.



Hình 3.9 PP định địa chỉ dữ liệu gián tiếp với hậu tố (+)

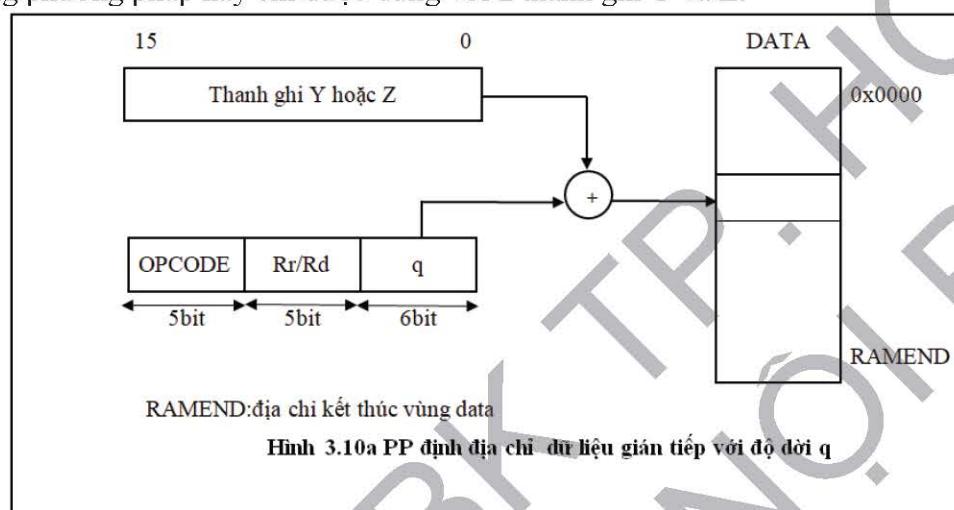
Một cách tương tự, chúng ta cũng có mã máy cho lệnh ST khi sử dụng phương pháp này với các thanh ghi X, Y Z như sau:

LỆNH	MÃ MÁY
ST X+, Rr	1001 001 r₄ r₃r₂r₁r₀ 1101
ST Y+,Rr	1001 001 r₄ r₃r₂r₁r₀ 1001
ST Z+, Rr	1001 001 r₄ r₃r₂r₁r₀ 0001

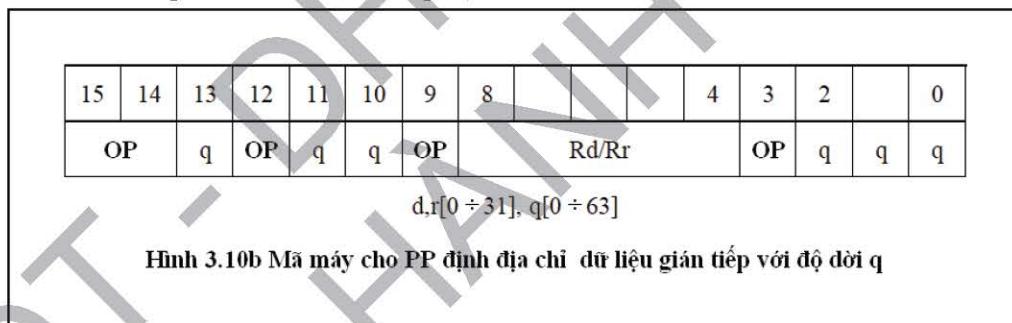
Với r_i là giá trị nhị phân tương ứng với chỉ số r của thanh ghi được sử dụng trong lệnh.
Ví dụ lệnh ST Y+, R5 sẽ có mã máy là 1001 0010 0101 1001.

3.2.4.4 Dữ liệu gián tiếp với một độ dời.

Việc xác định địa chỉ toán hạng trong phương pháp này bằng cách cộng nội dung của con trỏ với một độ dời q có giá trị nguyên 6 bit ($0 \leq q \leq 63$) rồi mới thực hiện lệnh.
Con trỏ trong phương pháp này chỉ được dùng với 2 thanh ghi Y và Z.



Việc xác định mã máy được mô tả trong hình 3.10b. Trong đó OP là mã lệnh tương ứng.
Rd/Rr là địa chỉ đích/nguồn của các thanh ghi GPRs.

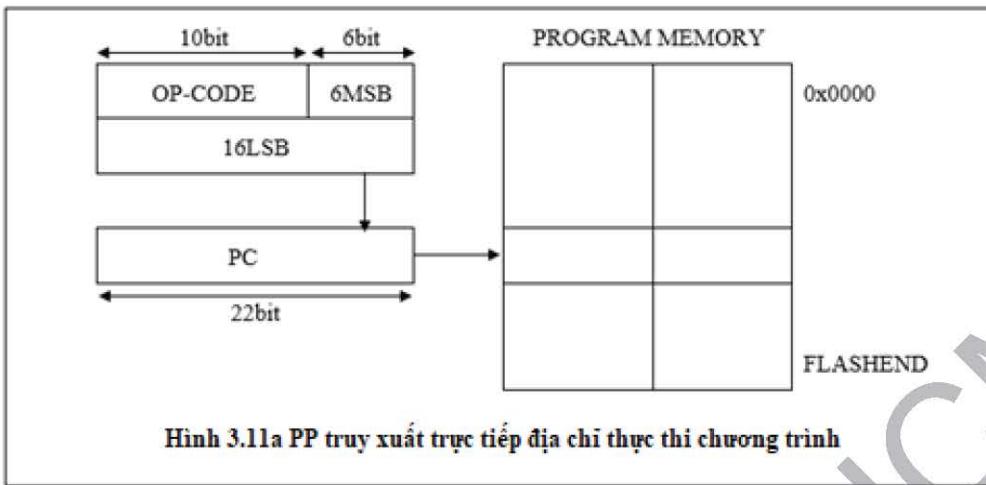


Ví dụ lệnh STD Z+5, R2, lệnh này có OP-CODE là 10010 tương ứng với mã máy sau:

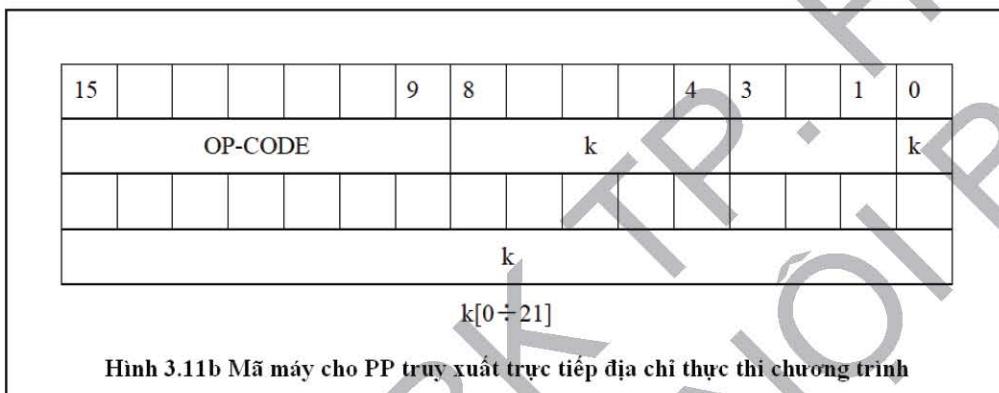
1000 0010 0010 0101
 R2
 q

3.2.5 Định địa chỉ truy xuất trực tiếp vùng bộ nhớ chương trình - FLASH:

Được dùng trong các câu lệnh JMP và CALL. Chương trình sẽ thực hiện lệnh tại một địa chỉ tức thời đặt trong lệnh.



Hình 3.11a PP truy xuất trực tiếp địa chỉ thực thi chương trình



Hình 3.11b Mã máy cho PP truy xuất trực tiếp địa chỉ thực thi chương trình

Ví dụ lệnh **JMP LABEL**, giả sử nhãn LABEL đặt tại địa chỉ 2000H và OP-CODE tương ứng của lệnh JMP là **1001 0101 10** thì mã máy của lệnh sẽ là 4 byte tương ứng:

1001 0100 0000 1100
0010 0000 0000 0000

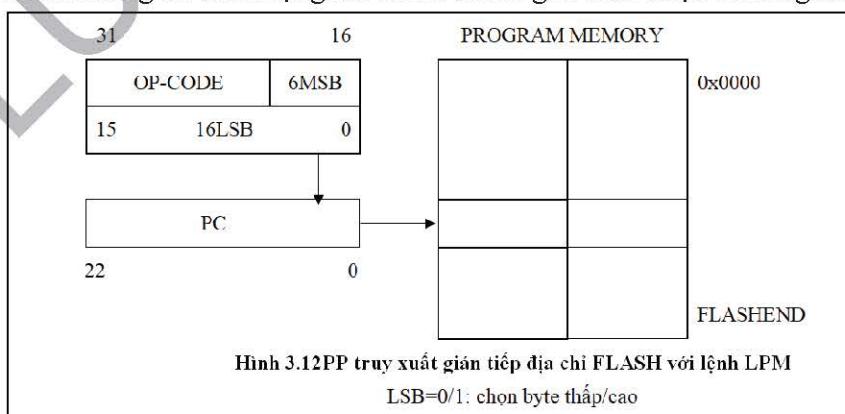
3.2.6 Định địa chỉ truy xuất gián tiếp vùng FLASH:

3.2.6.1 Truy xuất dữ liệu gián tiếp từ bộ nhớ chương trình:

Một địa chỉ trong bộ nhớ chương trình (FLASH) cần được truy xuất sẽ đặt trực tiếp vào thanh ghi Z tại các vị trí MSBs. Vì tổ chức của bộ nhớ chương trình là một từ 16 bit, nên nội dung cần truy xuất sẽ chọn dữ liệu là byte thấp hoặc byte cao của địa chỉ cần truy xuất, việc này được xác định bởi bit LSB của thanh ghi Z.

Phương pháp này được sử dụng trong các câu lệnh: LPM; LPM Rd,Z; LPM Rd,Z+ hay SPM. Tuy nhiên, lệnh SPM không hoàn toàn có sẵn trong tất cả các chip của họ AVR.

Khi lệnh LPM không có toán hạng thì R0 là thanh ghi đích được hiểu ngầm trong lệnh.



Hình 3.12 PP truy xuất gián tiếp địa chỉ FLASH với lệnh LPM
LSB=0/1: chọn byte thấp/cao

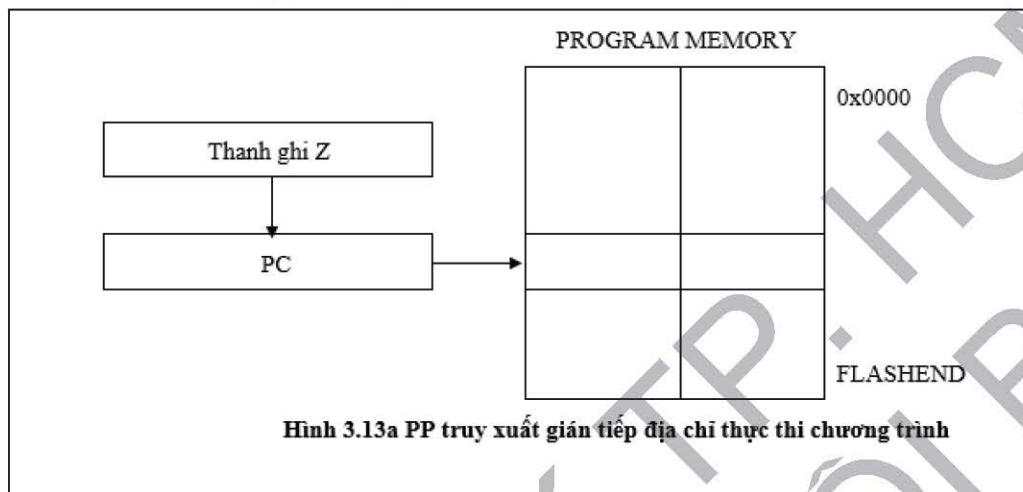
Lệnh LPM có mã máy cố định là: **0x95C8**.

Hai lệnh còn lại có cách tính mã máy hoàn toàn như phương pháp định địa chỉ gián tiếp qua thanh ghi tương ứng.

3.2.6.2 Truy xuất gián tiếp địa chỉ bộ nhớ chương trình

Trong phương pháp định địa chỉ này thanh ghi PC được nạp địa chỉ thực thi gián tiếp qua thanh ghi Z,

Phương pháp định địa chỉ gián tiếp qua thanh ghi Z sử dụng trong các lệnh LJMP và ICALL, không có toán hạng và mã máy là 1 giá trị cố định. Định vị địa chỉ này còn gọi là định vị địa chỉ tuyệt đối(Absolute Adressing), mã máy 2 byte-3MC

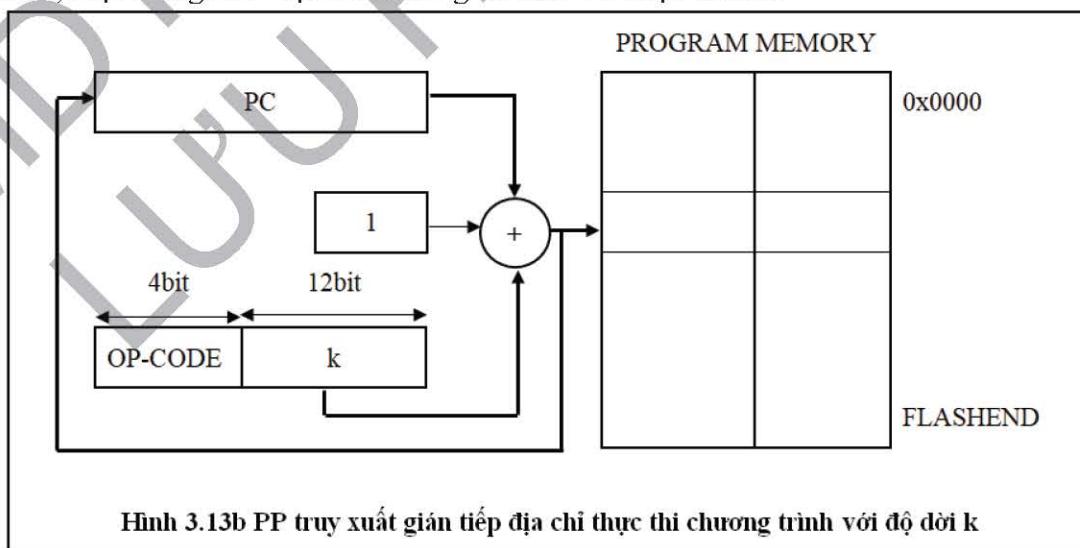


3.2.7 Định vị địa chỉ tương đối

3.2.7.1 Địa chỉ tương đối không điều kiện

Phương pháp này được sử dụng trong các lệnh RJMP và RCALL. Toán hạng trong lệnh là một nhãn hoặc một địa chỉ nằm trong phạm vi một độ dời tương đối k, với k là 1 số có dấu 12 bit so với PC của lệnh kế, có giá trị từ -2048 đến 2047. Mã máy của lệnh trên được trình bày ở hình 3.13c. Riêng với lệnh RCALL tùy thuộc vào chip sử dụng tương ứng với độ rộng của thanh ghi PC(16/22bit), mã máy sẽ là 2 hoặc 3 byte lệnh.

Trong mọi trường hợp người lập trình xác định địa chỉ đích cho trình dịch hợp ngữ theo cách thông thường như là 1 nhãn hoặc 1 hằng số. Trình dịch hợp ngữ sẽ đặt địa chỉ đích theo đúng định dạng của từng lệnh. Nếu định dạng yêu cầu bởi lệnh vượt ra ngoài khoảng cách dùng để xác định địa chỉ đích, một thông báo “địa chỉ đích ngoài tầm” sẽ được đưa ra.



Lệnh	OP-CODE	OFFSET
RJMP K	1100	k
RCALL K	1101	k
WORD	15 12 11 0

K=nhãn địa chỉ đích
K=PC+1+k

k[-2048 ÷ 2047]

Hình 3.13c Mã máy cho lệnh RJMP và RCALL

Ví dụ tính mã máy cho lệnh **RCALL DELAY**, giả sử rằng lệnh RCALL đặt tại địa chỉ 2000H, và nhãn DELAY đặt tại 200BH. Trong trường hợp này: K=200BH suy ra k= +10(offset). Ta có mã máy tương ứng của lệnh là **1101 0000 0000 1010**.

3.2.7.2 Địa chỉ tương đối có điều kiện

Phương pháp này áp dụng cho các lệnh nhảy có điều kiện xét theo các bit cờ trạng thái thanh ghi SREG hoặc so sánh giữa 2 thanh ghi. Độ lệch k(offset) là số có dấu 7 bit nên $k=-64 \div 63$, do đó địa chỉ đích giới hạn trong vùng từ -64 đến +63 địa chỉ so với địa chỉ kế tiếp của lệnh hiện hành (PC+1)

Hình 3.13d là mã lệnh của các lệnh nhảy có điều kiện

15				10	9		OFFSET		3	2	1	0
OP-CODE				k6		k0	OP_CODE				

Hình 3.13d. Mã máy cho các lệnh nhảy có điều kiện

Ví dụ lệnh **BRVS LOOP**, lệnh này sẽ rẽ nhánh đến nhãn LOOP khi cờ tràn V trong thanh ghi SREG bằng 1. Giả sử lệnh được đặt tại địa chỉ 50H và nhãn LOOP đặt tại 40H, lệnh này có OP-CODE là **111100**. Tương tự với cách tính offset ở trên, ta có k=-17(1101111), cờ V có vị trí ở bit 3 của thanh ghi SREG nên ta sẽ có mã máy tương ứng của lệnh là: **1111 0011 0111 1011**.

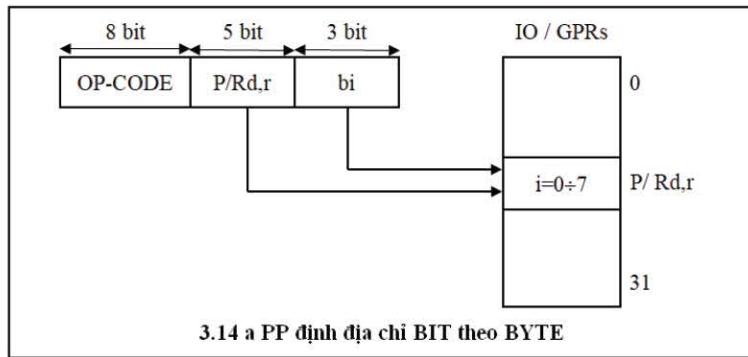
3.2.8 Định vị địa chỉ bit

3.2.8.1 Định địa chỉ BIT theo BYTE:

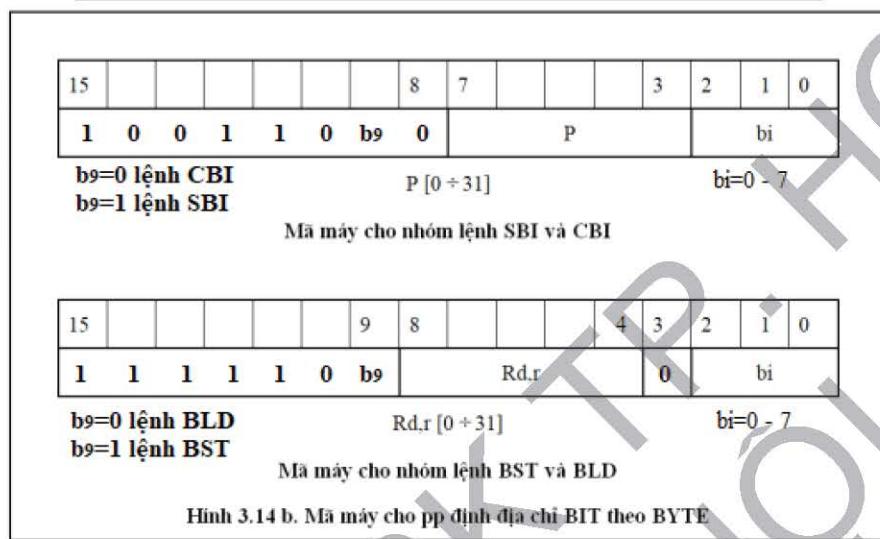
Phương pháp định địa chỉ này được sử dụng trong các lệnh xử lý bit đối với các bit có địa chỉ bit thuộc các thanh ghi IO có địa chỉ từ 00÷1FH, hoặc bit của nhóm thanh ghi GPRs. Địa chỉ của toán hạng được xác định chính là địa chỉ bit và địa chỉ thanh ghi tương ứng trong toán hạng của lệnh.

Phương pháp này được sử dụng cho 2 nhóm lệnh: nhóm lệnh chỉ tác động trên các bit của các thanh ghi IO là các lệnh SBI và CBI, nhóm còn lại là 2 lệnh BST và BLD, liên quan đến bit T của thanh ghi SREG và các bit của các thanh ghi GPRs, trong trường hợp này địa chỉ của bit T được hiểu ngầm trong mã lệnh.

Phương pháp định địa chỉ và mã máy của các nhóm lệnh này được trình bày ở hình 3.14a và b.



3.14 a PP định địa chỉ BIT theo BYTE



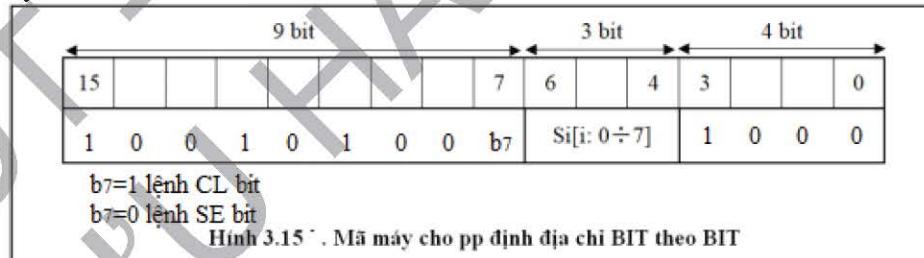
Hình 3.14 b. Mã máy cho pp định địa chỉ BIT theo BYTE

Ví dụ lệnh **SBI PORTA,6** sẽ đặt bit 6(110) của thanh ghi PORTA lên 1 (địa chỉ IO tương ứng của PORTA là 00010). Ta có mã máy của lệnh là: 1001 1010 0001 0110.

Hoặc lệnh **BLD R3,7** thực hiện lấy nội dung của bit T trong thanh ghi SREG lưu vào bit 111) của thanh ghi R3. Ta có mã máy tương ứng của lệnh là: 1111 1000 0011 0111.

3.2.8b Định địa chỉ BIT theo BIT:

Phương pháp định địa chỉ BIT theo BIT chỉ được thực hiện trên các bit của thanh ghi SREG, thực hiện trên các lệnh đặt/xóa cờ **BSET/BCLR** hoặc **SE/CL**. Hình 3.15 trình bày mã máy cho phương pháp này.



Hình 3.15⁷. Mã máy cho pp định địa chỉ BIT theo BIT

Ví dụ lệnh **BCLR 0** hoặc **CLC** cả 2 lệnh đều thực hiện việc xóa cờ Carry trong thanh ghi SREG, có mã máy là 1001 0100 **1000** 1000.

❖ Câu hỏi ôn tập

1. Người lập trình có thể tự ý tạo ra các phương pháp định địa chỉ mới cho một chip AVR không?
 2. Chúng ta có thể truy xuất các thanh ghi I/O mở rộng bằng phương pháp định địa chỉ vùng I/O không?
 3. Có bao nhiêu phương pháp định địa chỉ để truy xuất cho các thanh ghi I/O?
 4. Có bao nhiêu thanh ghi được sử dụng như con trỏ trong tập lệnh của ATmega324P?
 5. Liệt kê tên các thanh ghi con trỏ của ATmega324P, và kích thước của mỗi thanh ghi?

3.4 CÁC NHÓM LỆNH CỦA AVR

Các lệnh của AVR được chia làm 6 nhóm lệnh:

- Nhóm lệnh chuyển dời.
- Nhóm lệnh số học.
- Nhóm lệnh logic.
- Nhóm lệnh rẽ nhánh.
- Nhóm lệnh xử lý bit.
- Nhóm lệnh điều khiển MCU.

Phần phụ lục 2 giúp ta tham chiếu nhanh tất cả các lệnh của ATmega324P đã được phân nhóm.

3.4.1 NHÓM LỆNH CHUYÊN DỜI

Các lệnh di chuyển dữ liệu của AVR sử dụng nhóm các thanh ghi GPRs làm trung gian để lưu giữ các dữ liệu tạm thời, tùy theo việc truy xuất dữ liệu giữa các không gian nào với GPRs mà sẽ sử dụng những lệnh tương ứng cho việc đọc hoặc ghi. Sau đây chúng ta sẽ khảo sát cụ thể từng vùng của các bộ nhớ dữ liệu cũng như bộ nhớ chương trình của AVR.

Vùng GPRs

Có thể di chuyển các byte/word dữ liệu giữa các thanh ghi/cặp thanh ghi với nhau.

Cú pháp của lệnh như sau:

- Chuyển dời byte: **MOV Rd,Rr**

Rr: chứa dữ liệu nguồn, Rd: chứa dữ liệu đích
 $r = 0 \div 31, d = 0 \div 31$

Chuyển nội dung Rr vào Rd

- Chuyển dời word: **MOVW Rd+1:Rd,Rr+1:Rr**

Hay **MOVW Rd,Rr**

$r = 0, 2, 4, \dots, 30 ; d = 0, 2, 4, \dots, 30$

Chuyển nội dung Rr+1:Rr vào Rd+1:Rr

Hoặc có thể nạp một hằng số vào một thanh ghi theo phương pháp thanh ghi tức thời bởi lệnh:

LDI Rd,K

➤ Lưu ý: : lệnh trên chỉ được phép thực hiện đối với các thanh ghi từ R16÷R31, và K là một hằng số nguyên không dấu 8 bit: $0 \leq K \leq 255$.

Ví dụ 3.1: Thực hiện nạp giá trị 0xAA và 0xBB lần lượt vào 2 thanh ghi R1 và R0.

Giải:

Do không có lệnh nạp trực tiếp một giá trị tức thời vào các thanh ghi R0÷R15 nên chúng ta phải thực hiện trung gian qua một trong các thanh ghi R16÷R31, và R0,R1 là 2 thanh ghi liên tiếp nhau nên có thể dùng lệnh di chuyển dữ liệu theo WORD giữa các cặp thanh ghi thay vì BYTE.

LDI R16,0xAA

LDI R17,0xBB

MOVW R0,R16

Vùng SRAM

Việc đọc hoặc ghi dữ liệu giữa vùng SRAM và GPRs có thể được thực hiện theo nhiều cách khác nhau tùy theo việc chọn phương pháp định địa chỉ để thao tác lệnh.

PHƯƠNG PHÁP định địa chỉ	LỆNH ĐỌC	LỆNH GHI
PHƯƠNG PHÁP trực tiếp	LDS Rd,k	STS k,Rr
PHƯƠNG PHÁP gián tiếp	LD Rd,I_reg	ST I_reg,Rr
	LD Rd,-I_reg	ST -I_reg,Rr
PHƯƠNG PHÁP gián tiếp	LD Rd,I_reg+ I-reg:X,Y,Z	ST I_reg+,Rr I_reg:X,Y,Z
	LDD Rd,I_reg+q I_reg:Y,Z	STD I_reg+q,Rr I_reg:Y,Z

Ví dụ 3.2: Thực hiện sao chép giá trị 0xAA vào các ô nhớ có địa chỉ từ \$100 đến \$104 theo một trong các yêu cầu sau:

- Sử dụng phương pháp định địa chỉ trực tiếp.
- Sử dụng phương pháp định địa chỉ gián tiếp qua thanh ghi mà không lặp vòng.
- Sử dụng phương pháp định địa chỉ gián tiếp qua thanh ghi và thực hiện lặp vòng.

Giải:

- Sử dụng phương pháp định địa chỉ trực tiếp.

```
LDI R17,0xAA ;nạp giá trị 0xAA vào R17  
STS 0x100,R17 ;lưu nội dung của R17 vào ô nhớ $100  
STS 0x101,R17 ;lưu nội dung của R17 vào ô nhớ $101  
STS 0x102,R17 ;lưu nội dung của R17 vào ô nhớ $102  
STS 0x103,R17 ;lưu nội dung của R17 vào ô nhớ $103  
STS 0x104,R17 ;lưu nội dung của R17 vào ô nhớ $104
```

- Sử dụng phương pháp định địa chỉ gián tiếp qua thanh ghi mà không lặp vòng.

```
LDI R16,0xAA ;nạp giá trị 0xAA vào R16  
LDI YL,0x00 ;nạp giá trị 0x00 vào R28 (byte thấp của địa chỉ)  
LDI YH,0x1 ;nạp giá trị 0x1 vào R29 (byte cao của địa chỉ)  
ST Y,R16 ;lưu nội dung của R16 vào địa chỉ 0x100  
INC YL ;tăng địa chỉ lên 1  
ST Y,R16 ;lưu nội dung của R16 vào địa chỉ 0x101  
INC YL ;tăng địa chỉ lên 1  
ST Y,R16 ;lưu nội dung của R16 vào địa chỉ 0x102  
INC YL ;tăng địa chỉ lên 1  
ST Y,R16 ;lưu nội dung của R16 vào địa chỉ 0x103  
INC YL ;tăng địa chỉ lên 1  
ST Y,R16 ;lưu nội dung của R16 vào địa chỉ 0x104
```

Hoặc gọn hơn:

```
LDI R16,0xAA ;nạp giá trị 0xAA vào R16  
LDI YL,0x00;nạp giá trị 0x00 vào R28 (byte thấp của địa chỉ)  
LDI YH,0x1 ;nạp giá trị 0x1 vào R29 (byte cao của địa chỉ)  
ST Y+,R16 ;lưu nội dung của R16 vào địa chỉ 0x100,tăng địa chỉ lên 1  
ST Y+,R16 ;lưu nội dung của R16 vào địa chỉ 0x101,tăng địa chỉ lên 1  
ST Y+,R16 ;lưu nội dung của R16 vào địa chỉ 0x102,tăng địa chỉ lên 1  
ST Y+,R16 ;lưu nội dung của R16 vào địa chỉ 0x103,tăng địa chỉ lên 1  
ST Y+,R16 ;lưu nội dung của R16 vào địa chỉ 0x104,tăng địa chỉ lên 1
```

- Sử dụng phương pháp định địa chỉ gián tiếp qua thanh ghi và thực hiện lặp vòng.

```
LDI R16,0x5 ;R16 = 5 (R16 là biến đếm)  
LDI R20,0xAA ;nạp giá trị 0xAA vào R20(giá trị cần sao chép)  
LDI YL,0x00 ;nạp byte thấp của địa chỉ vào YL  
LDI YH,0x1 ;nạp byte cao của địa chỉ vào YH
```

LOOP:

```
ST Y+,R20 ;lưu nội dung của R20 vào ô nhớ có địa chỉ là nội dung của Y, tăng Y lên 1  
DEC R16 ;giảm biến đếm  
BRNE LOOP ;thực hiện lặp lại lệnh tại nhãn LOOP khi biến đếm chưa bằng 0
```

Ví dụ 3.3: Giả sử có chuỗi số có giá trị từ 0 ÷ 9 được lưu trong SRAM từ địa chỉ \$100. Hãy chuyển chuỗi dữ liệu trên trong SRAM theo các yêu cầu sau:

- Sử dụng phương pháp định địa chỉ gián tiếp với hậu tố (+) chuyển chuỗi dữ liệu trên đến vị trí bắt đầu từ địa chỉ \$200 và giữ nguyên thứ tự như ban đầu.

b. Sử dụng phương pháp định địa chỉ gián tiếp với tiền tố (-) chuyển chuỗi dữ liệu trên đến vị trí bắt đầu từ địa chỉ \$300 và đảo ngược thứ tự so với ban đầu.

Giải:

a. Sử dụng phương pháp định địa chỉ gián tiếp với hậu tố (+)

```
LDI XH,0X01 ;nạp byte địa chỉ cao của vùng dữ liệu cần chuyển vào thanh ghi X  
LDI XL,0X00 ;nạp byte địa chỉ thấp vào thanh ghi X  
LDI R20,10 ;biến đếm=10  
LDI YH,0X02 ;nạp địa chỉ đầu của vị trí cần chuyển đến.  
LDI YL,0X00 ;Y=$200
```

L0:

```
LD R17,X+ ;R17=($100)  
ST Y+,R17 ; Y=R17, Y=Y+1  
DEC R20 ;giảm biến đếm  
BRNE L0 ;lặp lại đến khi chuyển xong dữ liệu
```

b. Sử dụng phương pháp định địa chỉ gián tiếp với tiền tố (-)

```
LDI XH,0X01 ; nạp byte địa chỉ cao của vùng dữ liệu cần chuyển vào thanh ghi X  
LDI XL,0X00 ;nạp byte địa chỉ thấp vào thanh ghi X  
LDI R20,10 ;biến đếm=10  
LDI YH,0X03 ;nạp địa chỉ cuối của vị trí cần chuyển đến cộng thêm 1.  
LDI YL,0X0A ;Y=$30A
```

L0:

```
LD R17,X+ ;R17=($100)  
ST -Y,R17 ;Y=Y-1, Y=R17  
DEC R20 ;giảm biến đếm  
BRNE L0 ;lặp lại đến khi chuyển xong dữ liệu
```

Ví dụ 3.4: Viết chương trình thực hiện cộng dữ liệu được đặt trong 3 ô nhớ trong SRAM bắt đầu từ địa chỉ \$200, kết quả cất lại vào địa chỉ đầu tiên.

Giải:

```
LDI R16,HIGH(RAMEND) ;khởi tạo giá trị cho SP  
OUT SPH,R16  
LDI R16,LOW(RAMEND)  
OUT SPL,R16  
LDI ZL,0x00 ;Z=$200  
LDI ZH,2 ;  
LDI R21,0 ;R21 = 0  
LD R20,Z ;R20 = (Z)  
LDD R16,Z+1 ;R16 = (Z+1)  
ADD R20,R16 ;R20 = R20 + R16  
BRCC L1 ;rẽ nhánh đến L1 nếu cờ C=0  
INC R21 ;tăng R21
```

L1:

```
LDD R16,Z+2 ;R16 =(Z+2)  
ADD R20,R16 ;R20 = R20 + R16  
BRCC L2 ; rẽ nhánh đến L2 nếu cờ C=0  
INC R21 ;tăng R21
```

L2:

```
ST Z,R20 ;(Z)=R20  
STD Z+1,R21 ;(Z+1)=R21  
 ;Tổng=R21(byte cao)_R20(byte thấp)
```

Vùng I/O REGs(SFRs)

Để thực hiện việc đọc dữ liệu từ IORs vào GPRs ta sẽ dùng lệnh:

IN Rd,P ;với P là một ô nhớ thuộc vùng I/O reg

Ví dụ 3.5: Viết đoạn chương trình liên tục nhận dữ liệu từ thanh ghi PINB và gửi ra thanh ghi PORTC.

Giải:

```
AGAIN:IN      R16,PINB    ;đọc dữ liệu từ PortB vào R16  
          OUT      PORTC,R16  ;gửi ra PortC  
          JMP      AGAIN     ;thực hiện một cách liên tục
```

Mỗi một vị trí trong không gian bộ nhớ dữ liệu chỉ có duy nhất 1 địa chỉ được gọi là địa chỉ bộ nhớ dữ liệu(MEM), còn gọi là vùng SRAM. Ngoài ra mỗi một thanh ghi I/O chuẩn cũng có 1 địa chỉ tương đối so với vị trí bắt đầu của vùng I/O gọi là địa chỉ I/O, vì vậy các IORs sẽ có 2 địa chỉ như hình 3.17

Như vậy, ngoài việc đọc dữ liệu trong vùng I/O bởi lệnh IN, chúng ta cũng có thể truy cập như một ô nhớ của vùng dữ liệu bằng lệnh LDS với địa chỉ MEM tương ứng.

Nhưng so với lệnh LDS thì lệnh IN sẽ có những ưu điểm vượt trội sau:

- Lệnh IN sẽ được thực thi nhanh hơn so với lệnh LDS, vì lệnh này chỉ mất 1 chu kỳ máy trong khi lệnh LDS phải thực hiện trong 2 chu kỳ máy.
- Xét về số byte lệnh thì lệnh IN là 2 byte và lệnh LDS là 4 byte, do vậy sẽ chiếm nhiều không gian bộ nhớ chương trình hơn.
- Lệnh IN luôn có sẵn trong tất cả các chip AVR, còn lệnh LDS thì không thực hiện được trong một vài chip của họ AVR.

Và chú ý rằng lệnh IN chỉ được thực hiện trong vùng I/O, còn lệnh LDS thì có thể truy xuất toàn bộ những vị trí nằm trong vùng dữ liệu(MEM) của AVR.

Việc ghi dữ liệu vào IORs được thực hiện bằng lệnh:

OUT P,Rr ;với P là một ô nhớ thuộc vùng I/O reg

Lệnh này cũng chỉ được thực hiện đối với các địa chỉ thuộc vùng I/O. Cũng tương tự với lệnh IN, nếu xem mỗi IORs là một địa chỉ của vùng MEM thì chúng ta cũng có thể ghi dữ liệu vào vùng này bằng lệnh STS với địa chỉ tương ứng trong vùng MEM của nó.

Như đã đề cập ở phần phương pháp định địa chỉ trực tiếp, mỗi IORs đều có thể truy xuất theo tên chức năng trong vùng I/O đã được định nghĩa để dễ gọi nhớ. Ví dụ như muốn ghi dữ liệu là nội dung của R5 vào PORTA chúng ta có thể dùng một trong các lệnh sau để thực hiện:

OUT PORTA, R5

OUT \$02,R5 ; \$02 là địa chỉ của PORTA trong vùng I/O

STS \$22,R5 ; \$22 là địa chỉ của PORTA trong vùng MEM

Nên nhớ rằng chúng ta sẽ không thể thực hiện lệnh nạp trực tiếp một giá trị tức thời vào các IORs cũng như các ô nhớ trong SRAM, mà điều này chỉ có thể làm được thông qua các thanh ghi GPRs.

➤ **Lưu ý:**

- Các địa chỉ từ \$20(0x20) đến \$5F(0x5F) trong vùng MEM chỉ dùng để định nghĩa 64 thanh ghi I/O chuẩn trong tất cả các chip AVR có địa chỉ tương ứng trong vùng I/O là từ \$00÷\$3F.
- Một số chip AVR có số thanh ghi I/O nhỏ hơn 64 thì những vị trí còn lại trong vùng I/O sẽ dùng vào mục đích khác và người lập trình không thể can thiệp đến những địa chỉ này.
- Các chip AVR có vùng I/O mở rộng(ExIORs) từ \$60÷\$FF như ATmega324P phải truy xuất các ExIORs trong vùng này như truy xuất vùng MEM(bằng lệnh LD hoặc STS), xem hình 3.18
- Đối với các chip AVR khác nhau, thì với cùng 1 địa chỉ có thể gán cho các thanh ghi I/O khác nhau. Ví dụ như địa chỉ \$2 được gán cho thanh ghi TWAR của chip ATmega32, trong khi đó địa chỉ này lại được gán cho thanh ghi DDRE của chip ATmega128. Vì vậy, những lệnh giống nhau với cùng 1 địa chỉ trong vùng I/O sẽ có thể khác ý nghĩa trong những chip AVR khác nhau. Do đó, một chương trình được viết cho một chip AVR này sẽ có thể chạy không đúng trên chip AVR khác, và cách tốt nhất là nên sử dụng tên của các thanh ghi I/O để lập trình thay cho việc sử dụng địa chỉ của chúng.

ADDRESS		NAME	ADDRESS		NAME
MEM	I/O		MEM	I/O	
\$20	\$00	PINA	\$42	\$22	EEARH
\$21	\$01	DDRA	\$43	\$23	GTCCR
\$22	\$02	PORTA	\$44	\$24	TCCR0A
\$23	\$03	PINB	\$45	\$25	TCCR0B
\$24	\$04	DDRB	\$46	\$26	TCNT0
\$25	\$05	PORTB	\$47	\$27	PORCOA
\$26	\$06	PINC	\$48	\$28	OCR0B
\$27	\$07	DDRC	\$4A	\$2A	GPIOR1
\$28	\$08	PORTC	\$4B	\$2B	GPIOR2
\$29	\$09	PIND	\$4C	\$2C	SPCR
\$2A	\$0A	DDRD	\$4D	\$2D	SPSR
\$2B	\$0B	PORTD	\$4E	\$2E	SPDR
\$35	\$15	TIFR0	\$50	\$30	ACSR
\$36	\$16	TIFR1	\$51	\$31	OCDR
\$37	\$17	TIFR2	\$53	\$33	SMCR
\$3B	\$1B	PCIFR	\$54	\$34	MCUSR
\$3C	\$1C	EIFR	\$55	\$35	MCUCR
\$3D	\$1D	EIMSK	\$57	\$37	SPMCSR
\$3E	\$1E	GPIOR	\$5B	\$3B	RAMPZ
\$3F	\$1F	EECR	\$5D	\$3D	SPL
\$40	\$20	EEDR	\$5E	\$3E	SPH
\$41	\$21	EEARL	\$5F	\$3F	SREG

Hình 3.17 Bảng địa chỉ I/O REGs và địa chỉ bộ nhớ dữ liệu tương ứng

Ví dụ 3.6: Cho nhận xét về kết quả biên dịch và thực thi đoạn lệnh sau,với MCU ATmega324P:

```

LDI    R16,$02      ;R16=$02
OUT   TCCR1A,R16   ;TCCR1A=$02
LDI    R16,$06      ;R16=$06
OUT   TCCR1B,R16   ;TCCR1B=$06
LDI    R17,$00      ;R17=$00
STS   TCCR0A,R17   ;TCCR0A=$00
LDI    R17,$09      ;R17=$09
STS   TCCR0B,R17   ;TCCR0B=$09

```

Đoạn lệnh trên thực hiện khởi động Timer1 và Timer0 làm việc theo mode đặt.

Giải:

Khi biên dịch sẽ báo lỗi dòng 2 và dòng 4, vì các IORs TCCR1A, TCCR1B là các ExIORs.

Khi thực thi chương trình sẽ chạy không đúng ý tưởng, các thanh ghi TCCR0A, TCCR0B không được nạp các giá trị tương ứng, vì sử dụng các lệnh truy xuất vùng MEM, không phải truy xuất IORs. Các ô nhớ vùng MEM tương ứng ký hiệu TCCR0A=0x24 và ký hiệu TCCR0B=0x25 sẽ được nạp các giá trị \$00 và \$09. Nếu quy chiếu về địa chỉ vùng I/O(IORs) 0x24(MEM)=DDRB và 0x25(MEM)=PORTB!

Đoạn lệnh trên phải sửa lại như sau:

LDI	R16,\$02	;R16=\$02
STS	TCCR1A,R16	:TCCR1A=\$02
LDI	R16,\$06	;R16=\$06
STS	TCCR1B,R16	:TCCR1B=\$06
LDI	R17,\$00	;R17=\$00
OUT	TCCR0A,R17	:TCCR0A=\$00
LDI	R17,\$09	;R17=\$09
OUT	TCCR0B,R17	:TCCR0B=\$09

MEM	NAME	MEM	NAME	MEM	NAME
\$60	WDTCSR	\$7E	DIDR0	\$B4	OCR2B
\$61	CLKPR	\$7F	DIDR1	\$B6	ASSR
\$64	PRR	\$80	TCCR1A	\$B8	TWBR
\$66	OSCCAL	\$81	TCCR1B	\$B9	TWSR
\$68	PCICR	\$84	TCNT1L	\$BA	TWAR
\$69	EICRA	\$85	TCNT1H	\$BB	TWDR
\$6B	PCMSK0	\$86	ICR1L	\$BC	TWCR
\$6C	PCMSK1	\$87	ICR1H	\$BD	TWAMR
\$6D	PCMSK2	\$88	OCR1AL	\$C0	UCSR0A
\$6E	TIMSK0	\$89	OCR1AH	\$C1	UCSR0B
\$6F	TIMSK1	\$8A	OCR1BL	\$C2	UCSR0C
\$70	TIMSK2	\$8B	OCR1BH	\$C4	UBRR0L
\$73	PCMSK3	\$B0	TCCR2A	\$C5	UBRR0H
\$78	ADCL	\$B1	TCCR2B	\$C6	UDR0
\$79	ADCH	\$B2	TCNT2	\$C8	UCSR1A
\$7A	ADCSRA	\$B3	OCR2A	\$C9	UCSR1B
\$7B	ADCSRB	\$B4	TCNT1L	\$CA	UCRR1C
\$7C	ADMUX	\$B5	TCNT1H	\$CC	UDR1

Hình 3.18 CÁC I/O REG CÓ ĐỊA CHỈ ≥ \$60

Vùng FLASH

FLASH là không gian bộ nhớ dành cho chương trình có thể lên đến 8MB tùy thuộc vào chip AVR. Chúng ta có thể sử dụng phần không gian này để lưu một mảng dữ liệu cố định bằng chỉ dẫn .DB, và định dạng của các dữ liệu này có thể biểu diễn theo dạng nhị phân, thập phân, HEX hay mã ASCII đều được.

Tổ chức bộ nhớ chương trình trong AVR là 2 byte dữ liệu được lưu tại một địa chỉ. Về nguyên tắc MCU sẽ thực hiện lưu byte đầu tiên vào vị trí byte thấp của địa chỉ đầu tiên cần lưu, byte thứ hai lưu vào vị trí byte cao địa chỉ đầu tiên, byte thứ ba lưu vào byte thấp của địa chỉ tiếp theo và cứ như thế đến hết mảng dữ liệu muốn tạo trong bộ nhớ FLASH.

- Lưu ý: trường hợp mảng dữ liệu có số byte lẻ, trình biên dịch sẽ lưu byte lẻ cuối cùng vào byte thấp địa chỉ ô nhớ cuối, bỏ qua byte cao và đưa ra dòng cảnh báo:

“.cseg .db misalignment-padding zero byte”

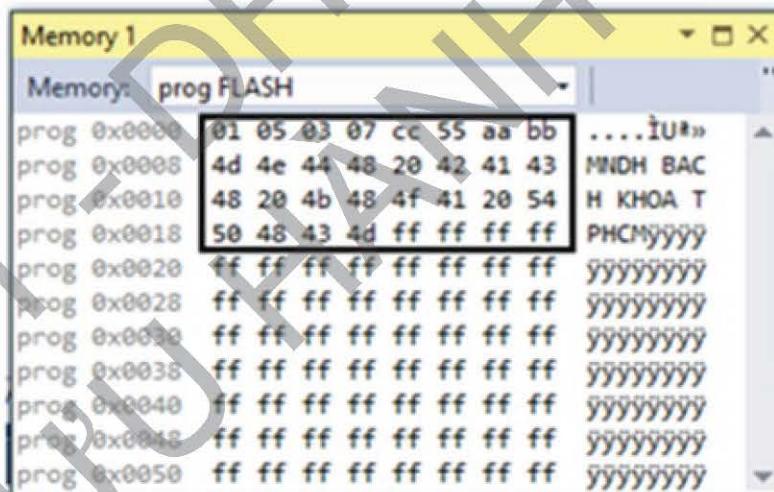
Ví dụ sau minh họa cho việc tạo một số mảng dữ liệu cố định trong vùng FLASH với dữ liệu được định dạng trong các hệ thống số nhị phân, thập phân, HEX, hoặc mã ASCII, lưu ý là sử dụng dấu nháy đơn (‘) cho các ký tự đơn, và dấu nháy kép(“) cho chuỗi ký tự.

Ví dụ 3.7: Viết các lệnh khai báo dữ liệu là các hằng số dạng thập phân, nhị phân, HEX, ASCII đặt trong bộ nhớ chương trình.

Giải:

```
.ORG 0 ;dữ liệu sẽ được lưu từ địa chỉ 0x0000  
DATA1: .DB 1,5,3,7 ;DECIMAL  
DATA2: .DB 0B11001100, 0B01010101 ;BINARY  
DATA3: .DB 0xAA,0xBB ;HEX  
DATA4: .DB 'M','N' ;ASCII  
DATA5: .DB "DH BACH KHOA TPHCM" ;ASCII
```

Sau khi thực hiện đoạn chương trình trên, ta có thể quan sát kết quả ở của số FLASH trong chương trình mô phỏng như hình 3.19



Hình 3.19 Mảng dữ liệu được tạo ra trong vùng FLASH

- Lưu ý: Hình 3.19 minh họa địa chỉ truy xuất vùng Flash theo byte(theo trình mô phỏng ATmel Studio 7), thực tế dữ liệu byte truy xuất theo địa chỉ word như ví dụ trên như hình 3.20.

Quy ước mặc định địa chỉ truy xuất dữ liệu trong vùng Flash đặt trong thanh ghi Z.

Các lệnh đọc dữ liệu trong vùng Flash có định dạng như sau:

LPM ;nạp dữ liệu vùng Flash địa chỉ trả bởi Z vào R0
LPM Rd,Z ;nạp dữ liệu vùng Flash địa chỉ trả bởi Z vào Rd,d=(0÷31)
LPM Rd,Z+ ;nạp dữ liệu vùng Flash địa chỉ trả bởi Z vào Rd,d=(0÷31),tăng địa chỉ thêm 1

LOW	HIGH	ADDRESS
\$0000	\$0001	\$0000
\$0002	\$0003	\$0001
\$0004	\$0005	\$0002
\$0006	\$0007	\$0003
\$0008	\$0009	\$0004
\$000A	\$000B	\$0005
...	...	
\$FFF0	\$FFF1	\$7FFE
\$FFF2	\$FFF3	\$7FFF

Hình 3.20 Giá trị của Z để truy xuất vùng FLASH

Với mảng dữ liệu đã tạo được mô phỏng trên hình 3.19, ví dụ như ta muốn lấy dữ liệu là byte thấp tại địa chỉ 0x0002 trong vùng FLASH thì giá trị cần nạp cho Z là 0x0004 bằng các lệnh sau:

```
LDI R30,0x04 ;ZL=0x04
LDI R31,0x00 ;Zh=0x00
LPM ;R0=CC
```

Hình 3.20 minh họa cho giá trị của Z khi lấy dữ liệu từ các địa chỉ tương ứng của vùng FLASH.

Hình 3.21 cho thấy kết quả quan sát khi chạy mô phỏng đoạn chương trình cùng với mảng dữ liệu đã tạo ở trên. Cửa sổ FLASH là nội dung của phần CODE chương trình, bắt đầu từ địa chỉ 0x0000, sau khi thực hiện lệnh LPM ta có nội dung byte thấp của ô nhớ có địa chỉ 0x0002(tương ứng giá trị 0x0004 nạp vào thanh ghi Z) là 0xCC được lưu trong thanh ghi R0.

Hình 3.21 Kết quả mô phỏng của lệnh LPM

Khi cần truy xuất đến các dữ liệu trong vùng FLASH theo phương pháp trực tiếp, thì bit LSB của thanh ghi Z sẽ xác định cho việc lấy byte thấp hay cao của ô nhớ đó, các bit MSB còn lại là giá trị địa chỉ trong vùng FLASH cần truy xuất.

Trình biên dịch cho phép sử dụng các hàm HIGH() và LOW() để lấy byte cao và byte thấp của một giá trị 16 bit như sau:

```
LDI ZH,HIGH(0x2010) ;ZH=0x20
LDI ZL,LOW(0x2010) ;ZL=0x10
```

Việc truy cập byte thấp của một ô nhớ trong vùng FLASH đồng nghĩa với việc dịch trái địa chỉ cần truy xuất 1 bit rồi đặt vào thanh ghi Z, để lấy nội dung của byte cao thì chỉ việc đặt bit LSB của Z có địa chỉ đã được dịch trái lên 1. Chẳng hạn như muốn đọc dữ liệu là byte thấp của ô nhớ có địa chỉ 0b00000101 ta đặt vào Z nội dung là 0b00001010, và muốn đọc byte cao thì đặt LSB lên 1 thành 0b00001011.

Có thể dùng ký hiệu <<1 cho toán tử dịch trái một bit và thực hiện các lệnh sau để đọc byte thấp của ô nhớ có địa chỉ \$100:

LDI ZH, HIGH(\$100<<1)	;nạp byte cao của địa chỉ vào ZH.
LDI ZL, LOW (\$100<<1)	;nạp byte thấp của địa chỉ vào ZL.
LPM R16, Z	;đọc nội dung của địa chỉ ô nhớ vào R16.

Một cách tương tự, có thể sử dụng ký hiệu | dành cho toán tử OR với bit 1 để đọc byte cao:

LDI ZH, HIGH((\\$100<<1) 1)	;nạp byte cao của địa chỉ với LSB=1 vào ZH.
LDI ZL, LOW ((\\$100<<1) 1)	;nạp byte thấp của địa chỉ với SB=1 vào ZL.
LPM R16, Z	;đọc nội dung của địa chỉ ô nhớ vào R16.

Lệnh LMP Rd,Z⁺ là 1 cái tiến trong việc truy xuất các dữ liệu có địa chỉ liên tiếp nhau trong vùng Flash mà không cần phải sử dụng thêm lệnh INC để trả đến địa chỉ kế tiếp cần truy xuất.

Ta có thể truy xuất 2 byte liên tiếp từ ô nhớ Flash địa chỉ \$100 như sau:

LDI ZH, HIGH(\$100<<1)	;nạp byte cao của địa chỉ vào ZH.
LDI ZL, LOW (\$100<<1)	;nạp byte thấp của địa chỉ vào ZL.
LPM R16, Z +	;đọc nội dung byte thấp của địa chỉ ô nhớ vào R16.
LPM R17, Z	;đọc nội dung byte cao của địa chỉ ô nhớ vào R17.

Ví dụ 3.8: Giả sử chuỗi ký tự “WORLD PEACE” được đặt trong bộ nhớ chương trình tại địa chỉ \$100. Viết chương trình gửi chuỗi ký tự này ra PortB sử dụng một trong các phương pháp sau:

- Biến đếm.
- Ký tự null(0) để nhận biết việc kết thúc chuỗi.
- Phương pháp truy xuất trực tiếp FLASH với hậu tố +.

Giải:

- a. Sử dụng biến đếm.

.INCLUDE "M324PDEF.INC"	;khai báo sử dụng file định nghĩa các ký hiệu MCU 324P
.ORG 0	
LDI R16,11	;R16 là biến đếm chứa số ký tự của chuỗi
LDI R20,0xFF	
OUT DDRB,R20	;đặt PortB là ngõ xuất
LDI ZH,HIGH(DATA<<1)	;ZH = byte cao của địa chỉ.
LDI ZL,LOW(DATA<<1)	;ZL = byte thấp của địa chỉ.
L1: LPM R20,Z	;lấy ký tự đặt vào R20
OUT PORTB,R20	;gửi ra Port B
INC ZL	;trở đến byte kế
DEC R16	;giảm biến đếm
BRNE L1	;lặp vòng về L1 nếu R16≠0
HERE: RJMP HERE	;dừng tại chỗ
.ORG 0x100	;tạo chuỗi tại địa chỉ \$100
DATA: .DB "WORLD PEACE"	

- b. Sử dụng ký tự null để nhận biết kết thúc chuỗi.

.INCLUDE "M324PDEF.INC"	
.ORG 0	
LDI R20,0xFF	
OUT DDRB,R20	;đặt PortB là ngõ xuất
LDI ZH,HIGH(DATA<<1)	;ZH = byte cao của địa chỉ.
LDI ZL,LOW(DATA<<1)	;ZL = byte thấp của địa chỉ.
L1: LPM R20,Z	;lấy ký tự đặt vào R20
CPI R20,0	;so sánh R20 với 0
BREQ HERE	;nhảy đến nhãn HERE nếu bằng
OUT PORTB,R20	;gửi ra Port B
INC ZL	;trở đến byte kế
RJMP L1	;lặp lại
HERE: RJMP HERE	;dừng tại chỗ
.ORG 0x100	;tạo chuỗi tại địa chỉ \$100
DATA: .DB "WORLD PEACE",0	;

c. Sử dụng phương pháp truy xuất trực tiếp FLASH với hậu tố +

```
.include "M324PDEF.INC"
.ORG 0
LDI    R20,0xFF
OUT   DDRB,R20      ;đặt PortB là ngõ xuất
LDI    ZH,HIGH(DATA<<1) ;ZH = byte cao của địa chỉ.
LDI    ZL,LOW(DATA<<1) ;ZL = byte thấp của địa chỉ.
L1:   LPM   R20,Z+    ;lấy ký tự đặt vào R20 và tăng địa chỉ lên 1
      CPI   R20,0     ;so sánh R20 với 0
      BREQ  HERE      ;nhảy đến nhãn HERE nếu bằng
      OUT   PORTB,R20  ;gửi ra Port B
      RJMP  L1        ;lặp lại
HERE: RJMP  HERE      ;dừng tại chỗ
.ORG 0x100
DATA: .DB "WORLD PEACE",0 ;
```

Vùng STACK

Vùng STACK(ngăn xếp) là một phần RAM được CPU sử dụng để lưu trữ thông tin tạm thời. Thông tin này có thể là dữ liệu hoặc địa chỉ.

Thanh ghi dùng để truy xuất vùng Stack được gọi là thanh ghi SP(Stack Pointer) hay con trỏ vùng ngăn xếp. SP được hiểu là một thanh ghi 16 bit, gồm 2 thanh ghi 8 bit thuộc vùng I/O có tên là SPL và SPH, trong đó SPL và SPH lần lượt là byte thấp và byte cao của thanh ghi SP. Nội dung của thanh ghi SP chính là địa chỉ truy xuất thông tin lưu giữ trong vùng stack.

Thanh ghi SP phải có kích thước đủ lớn để truy xuất được tất cả các địa chỉ thuộc vùng RAM. Vì vậy, đối với các chip AVR có dung lượng vùng RAM lớn hơn 256 byte thanh ghi SP sẽ là 2 thanh ghi SPH và SPL, còn những chip AVR khác có dung lượng vùng RAM nhỏ hơn 256 byte thì SP chỉ sử dụng 1 thanh ghi là SPL.

Lệnh PUSH lưu thông tin vào ô nhớ có địa chỉ trả bởi SP và sau đó SP sẽ giảm đi 1. Để lấy lại thông tin từ vùng stack trước tiên SP sẽ tăng lên 1 và nội dung của địa chỉ trả bởi SP sẽ được lấy ra bởi lệnh POP. Toán hạng trong các lệnh này chỉ được dùng cho các GPRs. Hoạt động của vùng stack thực hiện theo nguyên tắc LIFO (Last In First Out), nghĩa là cất vào sau lấy ra trước.

Định dạng các lệnh PUSH, POP như sau:

PUSH Rr ;cất nội dung trong Rr vào vùng stack địa chỉ trả bởi SP,SP giảm 1,r=(0÷31)
POPRd ;SP tăng 1,lấy nội dung có địa chỉ trả bởi SP trong vùng stack nạp vào Rd
;d=(0÷31)

Khi mới khởi động cấp nguồn, tùy theo chip AVR, nội dung của SP được khởi động bằng giá trị định trước. Với MCU ATmega324P SP=0x08FF=địa chỉ cao nhất của SRAM, MCU ATmega32 SP=0x0000- chính là địa chỉ của thanh ghi R0... Do đó, một cách tổng quát, ta cần phải khởi tạo và dự trữ vùng stack khi mới khởi động chương trình, bằng cách nạp SP địa chỉ đỉnh của stack(Top of Stack) chính là địa chỉ cao nhất của SRAM.

Dung lượng RAM trên mỗi chip AVR khác nhau sẽ không giống nhau, trình biên dịch cho AVR có sử dụng định nghĩa RAMEND là địa chỉ cao nhất của vùng RAM tương ứng với chip AVR được sử dụng, vì vậy để khởi tạo cho SP chúng ta chỉ cần nạp địa chỉ RAMEND vào SPH và SPL bằng các lệnh sau:

```
LDI  R16,HIGH(RAMEND) ;lấy byte cao của RAMEND
OUT SPH,R16           ;lưu vào SPH
LDI  R16,LOW(RAMEND)  ;lấy byte thấp
OUT SPL,R16           ;lưu vào SPL
```

Ví dụ 3.9: Ví dụ sau minh họa cho hoạt động của vùng Stack và nội dung của các thanh ghi sau khi thực hiện các lệnh tương ứng của đoạn chương trình sau:

```
.ORG 0
LDI  R16, HIGH(RAMEND)
OUT SPH, R16
LDI  R16, LOW(RAMEND)
```

OUT	SPL, R16
LDI	R31, 0
LDI	R20, 0x21
LDI	R22, 0x66
PUSH	R20
PUSH	R22
LDI	R20, 0
LDI	R22, 0
POP	R22
POP	R31

SAU KHI THỰC HIỆN LỆNH	NỘI DUNG THANH GHI				STACK			
	R20	R22	R31	SP				
OUT SPL,R16	0	0	0	0x08FF	<table border="1"> <tr><td>8FD</td></tr> <tr><td>8FE</td></tr> <tr><td>8FF</td></tr> </table> ← SP	8FD	8FE	8FF
8FD								
8FE								
8FF								
LDI R22, 0x66	0x21	0x66	0	0x08FF	<table border="1"> <tr><td>8FD</td></tr> <tr><td>8FE</td></tr> <tr><td>8FF</td></tr> </table> ← SP	8FD	8FE	8FF
8FD								
8FE								
8FF								
PUSH R20	0x21	0x66	0	0x08FE	<table border="1"> <tr><td>8FD</td></tr> <tr><td>8FE</td></tr> <tr><td>8FF</td></tr> </table> 0x21 ← SP	8FD	8FE	8FF
8FD								
8FE								
8FF								
PUSH R22	0x21	0x66	0	0x08FD	<table border="1"> <tr><td>8FD</td></tr> <tr><td>8FE</td></tr> <tr><td>8FF</td></tr> </table> 0x66 0x21 ← SP	8FD	8FE	8FF
8FD								
8FE								
8FF								
LDI R22, 0	0	0	0	0x08FD	<table border="1"> <tr><td>8FD</td></tr> <tr><td>8FE</td></tr> <tr><td>8FF</td></tr> </table> 0x66 0x21 ← SP	8FD	8FE	8FF
8FD								
8FE								
8FF								
POP R22	0	0x66	0	0x08FE	<table border="1"> <tr><td>8FD</td></tr> <tr><td>8FE</td></tr> <tr><td>8FF</td></tr> </table> 0x66 0x21 ← SP	8FD	8FE	8FF
8FD								
8FE								
8FF								
POP R31	0	0x66	0x21	0x08FF	<table border="1"> <tr><td>8FD</td></tr> <tr><td>8FE</td></tr> <tr><td>8FF</td></tr> </table> 0x66 0x21 ← SP	8FD	8FE	8FF
8FD								
8FE								
8FF								

Hình 3.22 Minh họa hoạt động của vùng STACK trong ví dụ 3.9

PHƯƠNG PHÁP BẢNG TRA (LOOK-UP TABLE)

Phương pháp bảng tra là một khái niệm được sử dụng rộng rãi trong lập trình vi điều khiển. Nó cho phép truy xuất các phần tử của bảng khi được sử dụng thường xuyên trong các tính toán tối thiểu. Ví dụ chúng ta cần tính giá trị của biểu thức $4+x^2$, trong đó x là 1 số có giá trị từ 0-9. Chúng ta có thể sử dụng một bảng tra thay cho việc tính toán giá trị của biểu thức nhiều lần với cùng 1 giá trị của x. Để lấy được các phần tử của bảng tra trong AVR, chúng ta cần cộng thêm chỉ số tương ứng với giá trị của tham số cần tra vào địa chỉ của bảng tra.

Ví dụ 3.10: Giả sử 3 bit thấp của portC được kết nối đến 3 công tắc. Viết chương trình gửi các ký tự ASCII đến portD tương ứng với các giá trị nhận được từ các công tắc như sau:

PC2-PC0	Mã ASCII
000	'0'
001	'1'
010	'2'
011	'3'
100	'4'
101	'5'
110	'6'
111	'7'

Giải:

.ORG 0

```

LDI    R16,0x0
OUT   DDRC,R16          ;DDRC = 0x00 (port C: input)
LDI    R16,0xFF
OUT   PORTC,R16          ;Port C có điện trở kéo lên
OUT   DDRD,R16          ;DDRD = 0xFF (port D : output)
LDI    ZH,0               ;ZH = byte cao của địa chỉ.
                           ;đọc trạng thái của các công tắc vào R16
BEGIN: IN    R16,PINC
ANDI   R16,0b00000111      ;che 3 bit thấp
LDI    ZL,LOW(ASCII TABLE<<1) ;ZL = byte thấp của địa chỉ.
ADD    ZL,R16              ;cộng chỉ số tương ứng giá trị của công tắc
LPM    R17,Z               ;lấy mã ASCII tương ứng từ bảng tra
OUT   PORTD,R17            ;xuất ra portD
RJMP   BEGIN              ;
                           ;;

.ORG $20
ASCII TABLE:
.DB '0','1','2','3','4','5','6','7'

```

❖ Ôn tập

- Thực hiện các câu lệnh để nạp giá trị 0b11001010 vào thanh ghi SPL?
- Viết đoạn chương trình thực hiện xuất giá trị \$55 ra PORTD bằng cách sử dụng toán hạng trong câu lệnh đối với PORTD theo các phương pháp định địa chỉ sau:
 - Tên thanh ghi.
 - Địa chỉ I/O.
 - Địa chỉ MEM.
- Viết 1 chương trình thực hiện xóa 16 ô nhớ trong vùng SRAM bắt đầu từ địa chỉ \$100, sử dụng các phương pháp sau:
 - Sử dụng lệnh INC cho việc tăng địa chỉ lên 1.
 - Địa chỉ tự động tăng trong lệnh.
- Sử dụng phương pháp bảng tra thực hiện tính giá trị của biểu thức $Y=x^2+2x+3$ và xuất ra PortC, biết rằng x là số BCD nhận được từ 4 bit thấp của PortB

3.4.2 NHÓM LỆNH SỐ HỌC

Các phép toán số học cũng như logic trong họ AVR đều phải được thực hiện trên các thanh ghi GPRs. Phần này sẽ trình bày các thao tác lệnh trên các toán hạng là các số không dấu cũng như có dấu và sự tác động(nếu có) đến các cờ trong thanh ghi trạng thái SREG sau khi lệnh được thực thi.

Khái niệm về số có dấu trong máy tính

Trong đời sống hàng ngày, các dữ liệu được sử dụng là các số dương và âm. Ví dụ khi nói về nhiệt độ thì nhiệt độ âm hoặc dương là một giá trị so sánh với ngưỡng 0°C (zero độ), dưới zero là nhiệt độ âm và ngược lại nhiệt độ dương là ở trên mức zero. Vì vậy, trong máy tính cũng phải có khả năng xử lý những con số như vậy. Bit MSB của dữ liệu được dùng để định nghĩa cho số có dấu, với số dương MSB được qui ước bằng 0 và ngược lại. Khi đề cập đến số có dấu, người sử dụng cần quan tâm đến việc biểu diễn và tầm giá trị có thể biểu diễn được tương ứng với số bit biểu diễn.(Xem chi tiết ở chương 0).

Lệnh cộng

Trong AVR, phép cộng chỉ được thực hiện trên các toán hạng được định địa chỉ kiểu thanh ghi. Một trong các câu lệnh của phép cộng như sau:

ADD Rd,Rr ; Rd=Rd+Rr,r=(0÷31),d=(0÷31)

Lệnh trên thực hiện cộng các dữ liệu đặt trong 2 toán hạng Rd và Rr, kết quả được cất vào Rd.

Phép cộng có thể tác động đến các cờ Z, C, N, V, H hoặc S của thanh ghi SREG phụ thuộc vào giá trị của các toán hạng tham gia vào.

Ví dụ 3.11: Xác định sự tác động đến các cờ trong thanh ghi SREG sau khi thực hiện các lệnh sau:

LDI R21,0xF5 ;R21 = F5H

LDI R22,0x0B ;R22 = 0BH

ADD R21,R22 ;R21 = R21+R22

Giải:

GPRs	Hex	Nhị phân									
			D7	D6	D5	D4	D3	D2	D1	D0	
SỐ NHỎ	1	1	1	1	1	1	1	1	1		
R21	F5H		1	1	1	1	0	1	0	1	
R22	0BH		0	0	0	0	1	0	1	1	
R22	100H		0	0	0	0	0	0	0	0	

Kết quả: R22 = 00, các cờ của thanh ghi SREG bị tác động như sau:

C=1 vì có số nhớ từ vị trí D7.

Z=1 vì kết quả trong R22 là zero.

H=1 vì có số nhớ từ vị trí D3 sang D4.

Tràn bit dấu xảy ra trong phép toán(Overflow)

Quan sát đoạn lệnh sau để hiểu thao tác thực hiện lệnh của vi điều khiển và các tác động đến các cờ trong thanh ghi SREG khi có hiện tượng tràn bit dấu xảy ra.

LDI R20,0X60 ;R20 = 0110 0000 (+96)

LDI R21,0x46 ;R21 = 0100 0110 (+70)

ADD R20,R21 ;R20 = (+96) + (+70) = 1010 0110
;R20 = A6H = -90 (kết quả sai!!!)

Giải thích:

Sau khi thực hiện lệnh cộng, kết quả trong thanh ghi R20 = 10100110B (-90) là một số âm, D7=1 nên sẽ tác động đến cờ N=1. Đây là 1 kết quả sai, do đó CPU sẽ đặt cờ tràn V lên 1 để chỉ ra phép toán vượt quá phạm vi biểu diễn.

Vì vậy khi thực hiện các phép toán trên số có dấu, người lập trình cần quan sát giá trị của cờ V để xử lý tình huống cho phù hợp.

➤ Lưu ý:

Phép cộng trong AVR không hỗ trợ cho phương pháp định địa chỉ trực tiếp trong vùng SRAM. Vì vậy để thực hiện phép cộng các ô nhớ trong vùng SRAM trước hết phải chuyển các giá trị cần cộng vào các thanh ghi GPRs.

Ví dụ 3.12: Viết đoạn chương trình thực hiện cộng giá trị 0x55 là nội dung của ô nhớ \$200 trong vùng SRAM với giá trị 0x33, kết quả lưu vào R20.

Giải:

```
LDS    R2,0x200 ;R2 = 55H
LDI    R20,0x33 ;R20 = 33H
ADD    R20,R2    ;R20 = R20 + R2 = 33H + 55H = 88H, C = 0
```

Lệnh cộng có nhớ

Khi thực hiện phép cộng trên số không dấu 16 bit, chúng ta cần quan tâm đến cờ nhớ được tạo ra từ byte thấp sang byte cao của phép toán. Lệnh ADC sẽ được sử dụng trong trường hợp này theo định dạng sau:

ADC Rd,Rr ; Rd=Rd+Rr+C,r=(0÷31),d=(0÷31)

Quan sát phép cộng với 2 toán hạng là các số 16 bit 3CE7H và 3B8DH.

Bài toán sẽ tuần tự thực hiện các bước sau:

- Đầu tiên thực hiện cộng 2 byte thấp: E7 + 8D = 74 và C = 1.
- Tiếp theo cộng 2 byte cao và số nhớ tạo ra từ kết quả của phép cộng 2 byte thấp. Ví dụ 3.9 minh họa cho bài toán này.

$$\begin{array}{r}
 & & 1 \\
 & 3C & E7 \\
 + & 3B & 8D \\
 \hline
 78 & 74
 \end{array}$$

Ví dụ 3.13:

Viết chương trình cộng 2 số 16 bit: 0x3CE7 và 0x3B8D. Giả sử rằng R1 = 8D, R2 = 3B, R3 = E7 và R4 = 3C. Kết quả lưu vào 2 thanh ghi R3 và R4, với R3 là byte thấp của tổng.

Giải:

```
ADD R3,R1 ;R3 = R3 + R1 = E7 + 8D = 74 & C = 1
ADC R4,R2 ;R4 = R4 + R2 + C = 3C + 3B + 1 = 78H
```

Lệnh ADIW thực hiện phép cộng một giá trị tức thời 6 bit(hằng số) với nội dung của một cặp thanh ghi. Lệnh này chỉ được thực hiện đối với 4 cặp thanh ghi cao nhất của GPRs(R24÷R31) và phù hợp cho các thao tác trên những thanh ghi con trỏ.

Lệnh trừ

Đối với AVR chúng ta có tất cả 5 lệnh cho phép trừ như sau:

SUB	Rd,Rr	;Rd=Rd-Rr
SBC	Rd,Rr	;Rd=Rd-Rr-C
SUBI	Rd,K	;Rd=Rd-K
SBCI	Rd,K	;Rd=Rd-K-C
SBIW	Rd+1:Rd,K	;Rd+1:Rd=Rd+1:Rd-K

Đa số các họ vi điều khiển thường có 2 lệnh khác nhau dành cho phép trừ là **SUB** và **SUBB**(phép trừ với số mượn - SUBtract with Borrow). Tập lệnh AVR sử dụng cờ nhớ(Carry) thay cho ký hiệu số mượn, vì vậy 2 lệnh SBC và SBCI được thực hiện cho phép trừ với số mượn.

Lệnh **SUB Rd,Rr** trong AVR thực hiện bằng cách chuyển thành phép cộng với số bù 2.

Phản ứng của CPU sẽ thực hiện lệnh này theo các bước sau:

- Lấy bù 2 của số trừ (toán hạng bên tay phải - Rr).
- Cộng với số bị trừ (toán hạng bên tay trái - Rd) và lưu kết quả ở Rd.
- Đảo bit cờ nhớ

Ví dụ 3.14: Trình bày các bước thực hiện của đoạn lệnh sau:

```
LDI R20, 0x23 ;R20=23H
LDI R21, 0x3F ;R21=3FH
SUB R21, R20 ;R21=R21-R20
```

Giải:

$$\begin{array}{r}
 R21 = 3F \quad 0011 \quad 1111 \\
 - R20 = 23 \quad 0010 \quad 0011 \quad \rightarrow + \quad 1101 \quad 1101 \quad (\text{Lấy bù 2 số trừ}) \\
 \hline
 \underline{1C} \qquad \qquad \qquad 1 \quad \underline{0001 \quad 1100} \quad (\text{Cộng với số trừ})
 \end{array}$$

C=0, D7=N=0 (Đảo bit cờ nhớ:C=0, kết quả là số dương)

Các cờ bị tác động sau lệnh SUB là C và N, nếu N=0(hoặc C=0) kết quả là số dương và ngược lại. Nếu là số âm thì cần thực hiện thêm bước lấy bù 2 bằng lệnh NEG để có được giá trị của phép toán.

Các lệnh **SUBI** và **SBIW** thực hiện phép trừ với một giá trị tức thời với một thanh ghi hoặc một cặp thanh ghi. Giá trị tức thời trong lệnh **SUBI** là một hằng số không dấu 8 bit, còn lệnh **SBIW** chỉ là 6 bit. Cũng giống như phép cộng, các cặp thanh ghi trong lệnh **SBIW** chỉ được dùng với 4 cặp thanh ghi có chỉ số cao nhất.

Ví dụ 3.15: Viết chương trình thực hiện phép toán 29H-18H và lưu kết quả vào thanh ghi R21 bằng 2 cách:

- Không dùng lệnh **SUBI**.
- Dùng lệnh **SUBI**

Giải:

- Không dùng lệnh **SUBI**.

```
LDI R21,0x29 ;R21 = 29H  
LDI R22,0x18 ;R22 = 18H  
SUB R21,R22 ;R21 = R21 - R22 = 29 - 18 = 11 H
```

- Dùng lệnh **SUBI**

```
LDI R21,0x29 ;R21 = 29H  
SUBI R21,0x18 ;R21 = R21 - 18 = 29 - 18 = 11 H
```

Ví dụ 3.16: Viết chương trình thực hiện phép toán 2917H-18H và lưu kết quả vào các thanh ghi R25 và R24.

Giải:

```
LDI R25,0x29 ;lưu byte cao vào R25 (R25 = 29H)  
LDI R24,0x17 ;lưu byte thấp vào R24 (R24 = 17H)  
SBIW R25:R24,0x18 ;R25:R24 = R25:R24 - 0x18 = 2917 - 18 = 28FF
```

Lệnh tăng, giảm 1 đơn vị

Các lệnh **INC** và **DEC** dùng để tăng hoặc giảm nội dung của thanh ghi đi 1.

```
INC Rd ;Rd=Rd+1(d=0÷31)  
DEC Rd ;Rd=Rd-1(d=0÷31)
```

Lệnh nhân các số nguyên

Tập lệnh AVR cho phép thực hiện phép nhân trên các số nguyên có dấu và/hoặc không dấu 8 bit. Kết quả là 1 số 16 bit được cắt vào 2 thanh ghi R1(byte cao) và R0(byte thấp). Các toán hạng trong lệnh chỉ được sử dụng với các thanh ghi từ R16÷R23.

Lệnh **MUL** và **MULS** lần lượt thực hiện cho các toán hạng là những số không dấu và có dấu.

Lệnh **MULSU** Rd,Rr thực hiện phép toán giữa số có dấu và không dấu. Trong đó, Rd là toán hạng có dấu và Rr là toán hạng không dấu.

Đoạn lệnh sau thực hiện phép toán 25H x 65H:

```
LDI R23,0x25 ;R23 = 25H  
LDI R24,0x65 ;R24 = 65H  
MUL R23,R24 ;25H * 65H = E99H, KQ: R1=0EH và R0=99H
```

Lệnh nhân các số lẻ

Lệnh **FMUL** Rd,Rr thực hiện phép tính nhân 2 toán hạng là các số lẻ không dấu. Trong đó Rd là số bị nhân và Rr là số nhân, dấu chấm cơ số được hiểu ngầm giữa 2 bit D7 và D6. Kết quả là một số lẻ 16 bit đặt trong 2 thanh ghi R1(byte cao) và R0(byte thấp), dấu chấm cơ số được hiểu ngầm giữa 2 bit D15 và D14.

Đoạn lệnh sau thực hiện phép toán 2 số lẻ 40H(0.5) x 20H(0.25):

```
LDI R23,0x40 ;R23 = 40H  
LDI R24,0x20 ;R24 = 20H  
FMUL R23,R24 ;40H * 20H = 1000H(0.125), KQ: R1=10H và R0=00H
```

Ngoài ra, còn có lệnh **FMULS** thực hiện phép toán nhân giữa 2 số lẻ có dấu, lệnh **FMULSU** là phép nhân giữa 2 số lẻ có dấu và không dấu.

Phép chia

Trong AVR không có lệnh chia, vì vậy phép chia được thực hiện thông qua giải thuật trừ nhiều lần. Để thực hiện phép chia từng byte, chúng ta đặt số bị chia vào một thanh ghi và liên tục trừ cho số chia đến khi không còn trừ được(còn nhớ bằng 1). Thương số chính là số lần thực hiện phép trừ, và phần còn lại của số bị chia chính là số dư của phép toán.

Chương trình sau thực hiện phép chia là nội dung thanh ghi R20(NUM) cho số có giá trị đặt trong R21(DENOMINATOR). Kết quả của phép chia là nội dung thanh ghi R22(QUOTIENT) phần dư lưu lại trong thanh ghi R20.

```
.DEF NUM = R20          ;định nghĩa ký hiệu NUM
.DEF DENOMINATOR = R21   ;định nghĩa ký hiệu DENOMINATOR
.DEF QUOTIENT = R22     ;định nghĩa ký hiệu QUOTIENT
    LDI    NUM,48          ;số bị chia = 48
    LDI    DENOMINATOR,5   ;số chia = 5
    CLR    QUOTIENT        ;thương số = 0
L1:   INC    QUOTIENT
      SUB   NUM, DENOMINATOR
      BRCC  L1              ;lặp lại nếu cờ C=0
      DEC   QUOTIENT         ;C=1 bỏ qua lần trừ
      ADD   NUM, DENOMINATOR ;trả lại số dư vào R20
      HERE: JMP   HERE        ;dừng tại chỗ
```

Ví dụ 3.17: Giả sử có một số HEX là FDH được lưu trong RAM tại địa chỉ \$315. Viết chương trình đổi số này sang hệ thập phân. Kết quả lưu vào các ô nhớ 0x322, 0x323, và 0x324. Với digit thấp nhất nằm ở ô nhớ 0x322.

Giải:

```
.EQU HEX_NUM = 0x315
.EQU RMND_L = 0x322
.EQU RMND_M = 0x323
.EQU RMND_H = 0x324
.DEF NUM = R20
.DEF DENOMINATOR = R21
.DEF QUOTIENT = R22
    LDI    R16,0xFD          ;$FD = 253D
    STS    HEX_NUM,R16        ;lưu $FD vào địa chỉ 0x315
    LDS    NUM, HEX_NUM
    LDI    DENOMINATOR,10      ;số chia = 10
L1:   INC    QUOTIENT
      SUB   NUM, DENOMINATOR
      BRCC  L1              ;lặp lại nếu C = 0
      DEC   QUOTIENT         ;
      ADD   NUM, DENOMINATOR ;
      STS    RMND_L, NUM       ;lưu kết quả vào hàng đơn vị
      MOV    NUM, QUOTIENT
      LDI    QUOTIENT,0
      INC    QUOTIENT
      SUB   NUM, DENOMINATOR
      BRCC  L2              ;
      DEC   QUOTIENT         ;
      ADD   NUM, DENOMINATOR ;
      STS    RMND_M, NUM       ;lưu vào hàng chục
      STS    RMND_H, QUOTIENT  ;lưu vào hàng trăm
      HERE: JMP   HERE        ;dừng tại chỗ
```

❖ Ôn tập

1. Cho biết lệnh ADD R20,R21 lưu kết quả vào đâu ?
2. Giải thích vì sao lệnh ADD R1,0x04 bị báo lỗi? Hãy viết lại cho đúng với mục đích thực hiện mà không bị lỗi.
3. Cho biết lệnh SUB R1,R2 lưu kết quả vào đâu ?
4. Xác định giá trị của cờ C sau mỗi đoạn lệnh sau:
 - (a) LDI R21 0x4F
 - (b) LDI R21, 0x9C
 - LDI R22 0xB1
 - LDI R22, 0x63
 - ADD R21, R22
 - ADD R21, R22
5. Trình bày hoạt động của CPU khi thực hiện phép toán 43H – 05H?
6. Giả sử cờ C = 1, R1 = 95H, và R2 = 4FH trước khi thực hiện lệnh “SBC R1,R2”, hãy xác định nội dung của R1 và C sau lệnh trên?
7. Phép nhân các toán hạng không dấu sẽ đặt kết quả vào các thanh ghi nào?
8. Cho biết lệnh **MUL R2,0x10** là lệnh có hợp lệ không? Giải thích?
9. Giá trị lớn nhất của 2 toán hạng được thực hiện trong phép nhân là bao nhiêu?
10. Có phải lệnh **MUL** chỉ được thao tác trên 2 thanh ghi R0 và R1?

3.4.3 NHÓM LỆNH LOGIC

Nhóm lệnh logic của AVR thực hiện các phép toán logic AND, OR, XOR, NOT(bù 1) trên các byte dữ liệu đặt trong 2 toán hạng là các thanh ghi Rd và Rr. Phép toán được thực hiện trên từng bit có cùng giá trị vị trí trọng số. Kết quả cất vào Rd.

Riêng phép AND và OR có thể dùng với toán hạng là một giá trị tức thời(hàng số 8 bit).

Đoạn lệnh sau minh họa cho một số phép toán logic:

LDI	R16,0xAA	;R16	=	AAH	=	1010 1010B
LDI	R17,0x0F	;R17	=	0FH	=	0000 1111B
AND	R16,R17	;R16 = AA & FF	=	0AH	=	0000 1010B
OR	R16,R17	;R16 = 0A ∨ 0F	=	0FH	=	0000 1111B
ORI	R16,0xF0	;R16 = 0F ∨ F0	=	FFH	=	1111 1111B
ANDI	R16,0xFF	;R16 = FF & FF	=	FFH	=	1111 1111B
EOR	R16,R17	;R16 = FF ⊕ 0F	=	F0H	=	1111 0000B
COM	R17	;R17 = /(0F)	=	F0H	=	1111 0000B

Lệnh lấy bù 2 của một thanh ghi:

NEG Rd ; d=0÷31

Ngoài ra, có thể đặt(set) hoặc xóa(clear) một hoặc nhiều bit trong một thanh ghi bởi các lệnh sau:

SBR Rd,K;16≤d≤31, 0≤K≤255

CBR Rd,K;

Với K là 1 hàng số 8 bit, bit cần đặt/xóa thì vị trí bit tương ứng trong k bằng 1.

Hoặc có thể xóa hoặc đặt toàn bộ các bit trong một thanh ghi bằng lệnh **CLR** hoặc **SER**.

Ví dụ cần đặt vị trí bit 0 và 2 trong thanh ghi R16 lên 1, và xóa các bit ở vị trí 5 và 7 trong thanh ghi R20 về 0, ta dùng các lệnh tương ứng sau:

SBR R16,0B00000101

CBR R20,0B10100000

Lệnh **TST** dùng kiểm nội dung của thanh ghi bằng 0 hay âm, tác động lên các bit cờ Z(zero) và N(số âm). Sau lệnh này cờ Z=1 nếu nội dung của thanh ghi cần kiểm tra bằng 0 và ngược lại. Cờ N=1 nếu bit MSB của thanh ghi bằng 1 và ngược lại.

❖ Ôn tập

Xác định nội dung của thanh ghi R0 sau khi thực hiện các đoạn lệnh sau:

- | | | | |
|-----|---------------|-----|---------------|
| a) | LDI R20,0x65 | (b) | LDI R20, 0x70 |
| | LDI R21,0x76 | | LDI R21, 0x6B |
| | AND R20,R21 | | OR R20, R21 |
| (c) | LDI R20,0x95 | (d) | LDI R20, 0x5D |
| | LDI R21,0xAA | | LDI R21, 0x75 |
| | EOR R20,R21 | | AND R20, R21 |
| (e) | LDI R20,0x0C5 | (f) | LDI R20, 0x6A |
| | LDI R21,0x12 | | LDI R21, 0x6E |
| | OR R20,R21 | | EOR R20, R21 |
| (g) | LDI R20,0x37 | | |
| | LDI R21,0x26 | | |
| | OR R20,R21 | | |

3.4.4 NHÓM LỆNH RẼ NHÁNH

Tập lệnh AVR có nhiều lệnh để điều khiển luồng chương trình, bao gồm các lệnh gọi và quay về từ một chương trình con, lệnh rẽ nhánh có điều kiện hoặc không có điều kiện.

Lệnh nhảy không điều kiện

Có 3 biến thể của lệnh nhảy không điều kiện là JMP, RJMP và IJMP sử dụng kiểu định địa chỉ trực tiếp hoặc gián tiếp vùng FLASH.

Lệnh **JMP** có đích đến là một địa chỉ bất kỳ trong không gian bộ nhớ chương trình tối đa là 4M(word), tùy thuộc vào vi điều khiển được sử dụng. Câu lệnh sau thực hiện việc nhảy tại chỗ nếu thực thi xong đoạn chương trình

HERE:JMP HERE

Lệnh **RJMP** là một lệnh nhảy tương đối. Một địa chỉ tương đối(hay còn gọi là offset) là 1 giá trị 12bit có dấu. Giá trị này được cộng với bộ đếm chương trình (PC hiện hành) để tạo ra địa chỉ tiếp theo cần được thực thi. Tầm nhảy được giới hạn là từ -2048 byte đến 2047 byte so với địa chỉ PC hiện hành. Thông thường các địa chỉ nhảy đến được xác định bởi các nhãn và trình dịch hợp ngữ sẽ xác định offset tương đối tương ứng. So với lệnh JMP có 4 byte lệnh thì lệnh này chỉ là lệnh 2 byte nhưng ngược lại thì đích nhảy đến bị giới hạn trong tầm cho phép như đã nói ở trên.

LABEL1:

....
RJMP LABEL1 ;nhảy đến nhãn
.... ;nhãn có thể đặt sau(LABEL2) hoặc trước(LABEL1) lệnh RJMP
LABEL2: ;nhưng không quá phạm vi 2K so với PC hiện hành.
.... ;không làm gì

Lệnh **IJMP** nhảy đến một địa chỉ gián tiếp được đặt trong thanh ghi Z. Khi lệnh này được thực thi, giá trị của PC sẽ được nạp là nội dung thanh ghi Z. Vì Z là một thanh ghi 2 byte nên địa chỉ mà lệnh IJMP có thể nhảy đến trong phạm vi không gian bộ nhớ chương trình tối đa là 64Kword.

Chương trình con và ngắt

Các lệnh gọi chương trình con hoạt động một cách tương tự với các lệnh nhảy không điều kiện, gồm có 3 lệnh là CALL, RCALL và ICALL.

Khi một chương trình con được gọi bởi lệnh CALL, trước tiên CPU sẽ lưu địa chỉ của lệnh tiếp theo sau lệnh CALL vào vùng stack và sau đó chuyển điều khiển đến chương trình con. Đây chính là cách để CPU biết địa chỉ tiếp theo cần thực hiện là ở đâu sau khi trở về từ chương trình con.

Đối với các họ AVR có độ dài của PC không quá 16bit(ATmega128/32/324P) thì giá trị của PC được chia thành 2 byte, byte cao sẽ được cắt vào vùng Stack trước, sau đó đến byte thấp.

Còn những họ AVR có PC lớn hơn 16 bit nhưng không quá 24 bit thì giá trị của PC được chia thành 3 byte, byte cao nhất sẽ được cắt trước, đến byte tiếp theo và sau cùng là byte thấp.

Trong cả 2 trường hợp trên thì các byte cao hơn được cắt vào stack trước tiên.

Các lệnh RCALL và ICALL cũng có các hạn chế trên địa chỉ đích như các lệnh RJMP và IJMP.

Các chương trình con cần được kết thúc bởi lệnh RET, lệnh này trả việc thực thi chương trình trở về lệnh theo sau lệnh CALL. Lệnh RET sẽ lấy lại 2 byte sau cùng ra khỏi vùng stack và nạp chúng cho bộ đếm chương trình.

Lệnh RETI trả điều khiển về chương trình gọi từ một trình phục vụ ngắt(ISR: Interrupt Service Routine). Điểm khác nhau giữa RET và RETI là RETI báo hiệu cho hệ thống điều khiển ngắt biết rằng quá trình xử lý ngắt đã xong, phục hồi lại các trạng thái của chương trình trước khi phục vụ ngắt, bao gồm cả công việc giống như lệnh RET. Các ngắt và lệnh RETI sẽ được trình bày chi tiết trong chương 10.

Ví dụ 3.18: Viết chương trình thực hiện xuất liên tục các dữ liệu \$55 và \$AA ra PORTB, giữa mỗi lần xuất tạo một khoảng thời gian trễ bằng cách gọi chương trình con DELAY. Quan sát sự tác động đến vùng Stack khi thực hiện lệnh gọi chương trình con.

Giai:

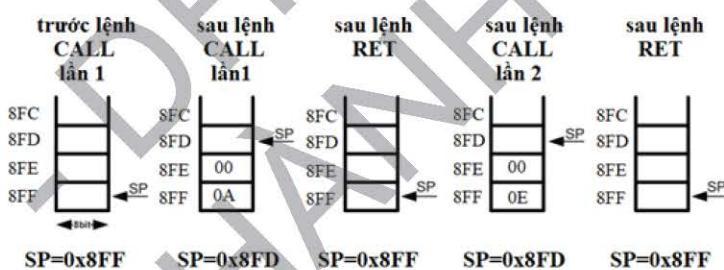
.ORG 0

0000	LDI	R16,HIGH(RAMEND) ;	khởi tạo vùng stack
0001	OUT	SPH,R16	
0002	LDI	R16,LOW(RAMEND) ;	
0003	OUT	SPL,R16	
0004	LDI	R16,0xFF	
0005	OUT	DDRB,R16	;đặt PORTB: xuất
BACK:			
0006	LDI	R16,0x55	; R16= 0x55
0007	OUT	PORTB,R16	;55H → port B
0008	CALL	DELAY	;tạo trễ
000a	LDI	R16,0xAA	; R16= 0xAA
000b	OUT	PORTB,R16	; AAH → port B
000c	CALL	DELAY	;làm trễ
000e	RJMP	BACK	;lặp lại liên tục

.ORG 0x300 ;đặt chương trình con DELAY tại địa chỉ \$300
DELAY:

0300	LDI	R20,0xFF	;R20 = 255
AGAIN:			
0301	NOP	,không làm gì nhưng tốn 1 chu kỳ máy	
0302	NOP		
0303	DEC R20	;giảm R20	
0304	BRNE AGAIN	;lặp lại đến khi R20 bằng 0	

PHÂN TÍCH TÁC ĐỘNG ĐẾN VÙNG STACK KHI THỰC HIỆN LỆNH CALL



Lệnh nhảy có điều kiện

Tập lệnh AVR cung cấp nhiều kiểu lệnh nhảy có điều kiện. Khi các điều kiện được xét là trạng thái của các cờ trong thanh ghi SREG, địa chỉ đích được xác định trong giới hạn ở khoảng cách nhảy từ -64 đến +63 byte kể từ lệnh theo sau lệnh nhảy có điều kiện. Ngoài ra điều kiện được xét cũng có thể là các bit trong tập các thanh ghi GPRs hoặc các thanh ghi I/O, nếu điều kiện xét được thỏa thì đích đến được xác định bằng cách bỏ qua lệnh theo ngay sau lệnh nhảy có điều kiện.

Các lệnh rẽ nhánh có điều kiện khác liên quan đến bit được trình bày trong phần nhóm lệnh xử lý bit. Một số lệnh do tác động của các cờ trong thanh ghi trạng thái(SREG) có định dạng như sau:

- Cờ Zero:

- BREQ k ; rẽ nhánh nếu cờ zero=1

Ví dụ:

CP R1,R0 ; so sánh 2 thanh ghi R1 và R0
BREQ EQUAL ;rẽ nhánh đến nhãn EQUAL nếu R1=R0(Z=1)

...

EQUAL: ... ;thực hiện lệnh gì đó..

- BRNE k ; ngược với lệnh trên

Ví dụ:

EOR R27,R27 ; R27=0

LOOP:

INC R27 ; tăng R27

...

CPI R27,5 ; so sánh R27 với 5

BRNE LOOP ; rẽ nhánh đến LOOP nếu $R27 \neq 5$ ($Z=0$)

NOP ;

- *Cờ nhớ C(carry):*

- BRCS k ; rẽ nhánh nếu cờ C=1

Ví dụ:

CPI R26,\$56 ; so sánh R26 với \$56

BRCS CARRY ; rẽ nhánh nếu cờ C=1

...

CARRY:....

- BRCC k ; rẽ nhánh nếu cờ C=0

Ví dụ:

ADD R22,R23 ; $R22 \leftarrow R22 + R23$

BRCC NOCARRY ; rẽ nhánh nếu C=0

...

NOCARRY:.... ;

- BRSH k ; rẽ nhánh nếu lớn hơn hay bằng

Ví dụ:

SUBI R19,4 ; $R19 \leftarrow R19 - 4$

BRSH LOOP ; rẽ nhánh nếu $R19 \geq 4$ (số không dấu)

...

LOOP: ... ;

- BRLO k ; rẽ nhánh nếu nhỏ hơn

Ví dụ:

EOR R19,R19 ; $R19 = 0$

LOOP:

INC R19 ; tăng R19

...

CPI R19,\$10 ; so sánh R19 với \$10

BRLO LOOP ; rẽ nhánh nếu $R19 < \$10$ (số không dấu)

...

- *Cờ âm N(negative):*

- BRMI k ; rẽ nhánh nếu là số âm ($N=1$)

Ví dụ:

SUBI R18,4 ; $R18 \leftarrow R18 - 4$

BRMI SO_AM ; rẽ nhánh nếu kết quả là số âm

...

SO_AM:....

- BRPL k ; rẽ nhánh nếu là số dương ($N=0$)

Ví dụ:

SUBI R26,\$50 ; $R26 \leftarrow R26 - \$50$

BRPL SO_DUONG ; rẽ nhánh nếu là số dương

...

SO_DUONG: ...

- Cờ tràn bù 2 V(Over):

- BRVS k ; (V = 1)

Ví dụ:

ADD R3,R4 ;R3←R3+R4

BRVS TRAN ;rẽ nhánh nếu phép toán bị tràn

...

TRAN: ...

- BRVC k ; (V = 0)

Ví dụ:

ADD R3,R4 ;R3←R3+R4

BRVC LOOP ; rẽ nhánh nếu phép toán không bị tràn

...

LOOP: ...

- Cờ dấu S(Signed): (S=N ⊕ V)

- BRGE k ; (S = 0)

Ví dụ:

CP R11,R12 ;so sánh R11 và R12

BRGE LOOP ;rẽ nhánh nếu $R11 \geq R12$ (có dấu)

...

LOOP: ...

- BRLT k ; (S= 1)

Ví dụ:

CP R16,R1 ;so sánh R16 và R1

BRLT LOOP ;rẽ nhánh nếu $R16 < R1$ (có dấu)

...

LOOP: ...

- Cờ nhớ phân nửa H(half carry):

- BRHS k ; (H = 1)

Ví dụ:

BRHS LOOP ;rẽ nhánh nếu cờ H=1

...

LOOP: ...

- BRHC k ; (H = 0)

Ví dụ:

BRHC LOOP ;rẽ nhánh nếu cờ H=0

...

LOOP: ...

- Cờ sao chép T(bit copy):

- BRTS k ; (T = 1)

Ví dụ:

BST R3,5 ;lưu bit 5 của R3 vào cờ T

BRTS LOOP ;rẽ nhánh nếu bit này bằng 1

...

LOOP: ...

- BRTC k ; (T = 0)

Ví dụ:

BST R3,5 ; lưu bit 5 của R3 vào cờ T
 BRTC LOOP ; rẽ nhánh nếu bit này bằng 0
 ...
 LOOP: ...

- Cờ ngắt I(Global Interrupt enable):

- BRIE k ; (I = 1)

Ví dụ:

BRIE INT_EN ;rẽ nhánh nếu ngắt được cho phép

...

INT_EN: ...

- BRID k ; (I = 0)

Ví dụ:

BRIE INT_DIS ;rẽ nhánh nếu ngắt không được cho phép

...

INT_DIS: ...

Một trong những ứng dụng phổ biến của lệnh nhảy có điều kiện là thực hiện các chương trình tạo trễ(delay), việc tính toán khoảng thời gian tạo trễ phụ thuộc vào tần số hoạt động của hệ thống và các lệnh được sử dụng trong chương trình. Để xác định số chu kỳ máy tương ứng của 1 lệnh người lập trình cần tham khảo tập lệnh của vi điều khiển thực tế được sử dụng.

Ví dụ 3.19: Một hệ vi điều khiển AVR hoạt động với tần số Fosc=1MHz. Hãy xác định thời gian thực hiện cho các lệnh sau:

- | | | |
|----------|----------|----------|
| (a) LDI | (b) DEC | (c) LD |
| (d) ADD | (e) NOP | (f) JMP |
| (g) CALL | (h) BRNE | (i) .DEF |

Giải:

Từ tập lệnh sẽ tra được số chu kỳ máy(MC) tương ứng của các lệnh để xác định thời gian thực thi như sau:

STT	LỆNH	THỜI GIAN THỰC THI	Chú thích
(a)	LDI	1MC x 1μs	
(b)	DEC	1MC x 1μs	
(c)	OUT	1MC x 1μs	
(d)	ADD	1MC x 1μs	
(e)	NOP	1MC x 1μs	
(f)	JMP	3MC x 1μs	
(g)	CALL	4MC x 1μs	
(h)	BRNE	2/1MC x 1μs	MC là 2 nếu điều kiện đúng và ngược lại
(i)	.DEF	0	Các chỉ dẫn không tồn chu kỳ máy

Với họ AVR 1 chu kỳ máy(MC) bằng 1 chu kỳ dao động. Vì vậy với tần số 1MHz sẽ có chu kỳ máy là 1μs.

Ví dụ 3.20: Xác định thời gian trễ được tạo ra bởi chương trình con sau, biết rằng tần số hoạt động của hệ thống là 10MHz.

.DEF COUNT = R20

...
 DELAY: LDI COUNT, 0xFF ;1MC
 AGAIN: NOP ;1
 NOP ;1
 DEC COUNT ;1
 BRNE AGAIN ;2/1
 RET ;4

Giải:

Thời gian trễ: $[1 + (1 + 1 + 1) \times 255 + (2 \times 254 + 1) + 4] \times 0.1 \mu\text{s} = 127.9 \mu\text{s}$.

Lệnh BRNE sẽ thực hiện trong 2 chu kỳ máy nếu điều kiện đúng, nghĩa là sẽ quay lại thực hiện lệnh tại nhãn AGAIN, ngược lại lệnh thực hiện trong 1 chu kỳ máy, sau đó thực hiện lệnh RET và kết thúc chương trình con DELAY. Tính một cách chính xác thì chương trình trên thực hiện trong thời gian $127.9 \mu\text{s}$.

❖ Ôn tập

1. Trong AVR, có thể chuyển điều khiển đến bất cứ vị trí nào trong không gian bộ nhớ chương trình 4M bằng cách sử dụng lệnh CALL?(Đúng/Sai)
2. Cũng hỏi như câu 1 với lệnh RCALL?
3. Khi mới bật nguồn, nội dung thanh ghi SP sẽ có giá trị là bao nhiêu với MCU ATmega324P?
4. Nội dung của SP sẽ tăng hay giảm tương ứng với các lệnh CALL và RET?
5. Cho biết kích thước của vùng Stack trong AVR?

3.4.5 NHÓM LỆNH XỬ LÝ BIT

Tập lệnh AVR cung cấp các lệnh thực hiện thao tác truy xuất bit theo kiểu dữ liệu trực tiếp trên bit hoặc byte của toán hạng tùy theo vùng dữ liệu được thao tác.

Mọi hoạt động truy xuất bit đều sử dụng kiểu định địa chỉ trực tiếp, không có lệnh tác động trên bit đối với không gian bộ nhớ chương trình.

Thao tác bit trên các thanh ghi GPRs

Các thanh ghi GPRs đều không được định vị bit, nên thao tác xử lý bit đối với các bit của thanh ghi đều được thực hiện trên cả byte dữ liệu của toán hạng. Một số lệnh xử lý bit chỉ cho phép sử dụng trên các thanh ghi từ R16 đến R31.

Lệnh đặt(Set) hoặc xóa(clear) bit

Lệnh SBR(Set Bits in Register) thực hiện đặt các vị trí bit được chỉ định trong thanh ghi lên 1 theo định dạng sau:

SBR Rd,k ;d=(16÷31),k=(0÷255)

Trong đó: Rd: có giá trị từ R16÷R31, k là một giá trị 8 bit: 00-FFH.

Ví dụ như muốn đặt vị trí của bit 0 và 5 trong thanh ghi R16 lên 1 thì giá trị bit tương ứng trong toán hạng K tại các vị trí 0 và 5 được xác định là 1, các vị trí còn lại bằng 0. Sau lệnh này, các bit ở vị trí 0 và 5 trong thanh ghi R16 sẽ được đặt lên 1, không quan tâm giá trị trước đó của các bit này là bao nhiêu.

LDI R16,0b01011001 ;R16 = 0x59

SBR R16,0b00100001 ;R16=0x79

Kết quả thực hiện của lệnh này hoàn toàn giống với lệnh ORI Rd,k. Có thể nói lệnh SBR là 1 tên khác của lệnh ORI.

Lệnh CBR (Clear Bits in Register) thực hiện một cách tương tự lệnh SBR thay vì đặt các vị trí bit được chỉ định trong các toán hạng lên 1 thì ngược lại sẽ xóa các bit mà không quan tâm đến giá trị trước đó của chúng trong thanh ghi được chỉ định. Lệnh này cũng giống với lệnh ANDI Rd,k.

Định dạng lệnh CBR như sau:

CBR Rd,k ;d=(16÷31),k=(0÷255)

Lệnh sao chép bit

Để sao chép từng bit dữ liệu giữa các thanh ghi GPRs với nhau, tập lệnh AVR sử dụng bit(còn) T(Temporary) trong thanh ghi trạng thái SREG làm trung gian cho việc thực hiện này bởi các lệnh BST và BLD theo định dạng như sau:

BST Rd,b ;lưu bit b(b=0÷7) thanh ghi Rd(d=0÷31) vào bit T

BLD Rd,b ;chép bit T vào bit b(b=0÷7) thanh ghi Rd(d=0÷31)

Đoạn lệnh sau chuyển bit 3 của thanh ghi R16 vào bit 5 thanh ghi R20:

BST R16,3 ;lưu bit 3 trong R16 vào bit T

BLD R20,5 ;chép giá trị của bit T vào bit 5 trong R20

Ví dụ 3.21:

Viết chương trình thực hiện đọc trạng thái của một công tắc được kết nối đến chân PB4 và lưu vào bit D0(LSB) của một ô nhớ RAM có địa chỉ 0x200.

Giải:

```
.EQU MYREG = 0x200 ;định nghĩa vị trí MYREG là địa chỉ ô nhớ 0x200
CBI DDRB,4 ;xác định ngõ nhập cho PB4
SBI PORTB,4 ;khai báo điện trở kéo lên PB4
IN R17,PINB ;đọc portB vào R17
BST R17,4 ;lưu cờ T = PINB.4
LDI R16,0x00 ;R16 = 0
BLD R16,0 ;sao chép bit T vào bit D0 của R16
STS MYREG,R16 ;lưu giá trị R16 vào ô nhớ $200
HERE: JMP HERE
```

Lệnh kiểm tra bit

Để xác định trạng thái của một bit trong thanh ghi GPRs có giá trị là 1 hay 0, chúng ta sử dụng lệnh SBRS và SBRC theo định dạng sau:

SBRS Rd,b ;bỏ qua lệnh kế tiếp nếu bit b=1(b=0÷7),b thuộc Rd(d=16÷31)

SBRC Rd,b ;bỏ qua lệnh kế tiếp nếu bit b=0(b=0÷7),b thuộc Rd(d=16÷31)

Lệnh SBRS thực hiện kiểm tra một bit của thanh ghi và bỏ qua lệnh tiếp theo sau nếu bit là 1 và ngược lại với lệnh SBRC.

Đoạn chương trình sau sẽ không thực hiện câu lệnh “LDI R20,0x55”

```
LDI    R17,0b0001010 ;R17=0b0001010
SBRS   R17,3          ;kiểm tra bit ở vị trí 3 trong thanh ghi R17
LDI    R20,0x55       ;lệnh này được bỏ qua vì bit được kiểm tra bằng 1
LDI    R30,0x33
```

Ví dụ 3.22:

Bằng cách sử dụng lệnh SBRS, viết chương trình thực hiện kiểm tra trạng thái của 1 công tắc SW được kết nối đến chân PC7 để thực hiện các yêu cầu sau:

- Nếu SW=0: gửi ký tự ‘N’ ra portD.
- Nếu SW=1: gửi ký tự ‘Y’ ra portD.

Giải:

```
CBI  DDRC,7      ;PC7: ngõ nhập
SBI  PORTC,7     ;khai báo điện trở kéo lên PC7
LDI  R16,0xFF
OUT DDRD,R16    ;Port D: port xuất
```

AGAIN:

```
IN   R20,PINC    ;R20 = PIN
SBRS R20,7        ;bỏ qua lệnh kế(RJMP) nếu PC7=1 và thực hiện lệnh LDI R16,'Y'
RJMP OVER         ;ngược lại, nhảy đến nhãn OVER
LDI  R16,'Y'      ;R16 = 'Y'
OUT PORTD,R16    ;gửi 'Y' ra PortD
RJMP AGAIN        ;lặp vòng lại
```

OVER:

```
LDI  R16,'N'      ;R16 = 'N'
OUT PORTD,R16    ;gửi 'N' ra PORTD
RJMP AGAIN        ;lặp vòng lại
```

Thao tác bit trên các thanh ghi I/O

Đối với 32 thanh ghi I/O đầu tiên (có địa chỉ từ 0 đến 31 trong vùng I/O), việc đặt hoặc xóa các bit trong các thanh ghi này được thực hiện bằng các lệnh **SBI** và **CBI** định dạng như sau:

SBI A,b ;đặt bit b=1(b=0÷7) thuộc thanh ghi I/O A((A=0÷31))

CBI A,b ;xóa bit b=0(b=0÷7) thuộc thanh ghi I/O A((A=0÷31))

Ví dụ sau sẽ thực hiện đặt bit 4 của PORTA và xóa bit 5 của PORTB:

```
SBI PORTA,4
CBI PORTB,5
```

Chúng ta cũng có thể kiểm tra trạng thái của các bit trong các thanh ghi I/O bằng các lệnh SBIS và SBIC một cách tương tự với các lệnh kiểm tra bit trong các thanh ghi GPRs. Lệnh SBIS và CBIS có định dạng như sau:

SBIS A,b ;bỏ qua lệnh kế tiếp nếu bit b=1(b=0÷7),b thuộc thanh ghi I/O A((A=0÷31))

SBICA,b ;bỏ qua lệnh kế tiếp nếu bit b=0(b=0÷7),b thuộc thanh ghi I/O A((A=0÷31))

Ví dụ 3.22 có thể viết lại bằng lệnh SBIS như sau:

```
CBI  DDRC,7      ; PC7: ngõ nhập
SBI  PORTC,7     ;khai báo điện trở kéo lên PC7
LDI  R16,0xff
OUT DDRD,R16    ; Port D: port xuất
```

AGAIN:

SBIS	PINC,7	;bỏ qua lệnh kế nếu PC7=1
RJMP	OVER	;PC7=0 nhảy đến nhãn OVER
LDI	R16,'Y'	;R16 = 'Y'
OUT	PORTD,R16	;gửi 'Y' ra PortD
RJMP	AGAIN	;

OVER:

LDI	R16,'N'	;R16 = 'N'
OUT	PORTD,R16	;gửi 'N' ra PORTD
RJMP	AGAIN	;

Thao tác bit trên thanh ghi trạng thái SREG

Như chúng ta đã biết, thanh ghi SREG gồm có 8 cờ trạng thái: C(Carry flag), Z(Zero flag), N(Negative flag), V(Overflow flag), S(Sign flag), H(Half carry flag), T(Temporary flag) và I(Interrupt flag), tương ứng với các vị trí bit như sau:

Bit	D7	D6	D5	D4	D3	D2	D1	D0
SREG	I	T	H	S	V	N	Z	C

Tất cả các bit cờ đều có thể được đặt và xóa bằng các lệnh **BSET** và **BCLR** theo định dạng sau:

BSET s

BCLR s

Trong đó s là vị trí bit cờ tương ứng trong thanh ghi SREG (s = 0÷7). Ví dụ muốn đặt bit cờ C lên 1 và xóa cờ H bằng 0, ta thực hiện các lệnh sau:

BSET 0

BCLR 5

Hoặc đơn giản hơn là sử dụng trực tiếp tên của cờ tương ứng trong lệnh mà không cần toán hạng như bảng sau:

Lệnh ĐẶT bit cờ tương ứng lên 1		Lệnh XÓA bit cờ tương ứng lên 1		
Lệnh	Thao tác lệnh và kết quả	Lệnh	Thao tác lệnh và kết quả	
SEC	Đặt cờ nhớ - Carry	C=1	CLC	Xóa cờ nhớ - Carry
SEZ	Đặt cờ không - Zero	Z=1	CLZ	Xóa cờ không - Zero
SEN	Đặt cờ âm - Negative	N=1	CLN	Xóa cờ âm - Negative
SEV	Đặt cờ tràn – OverFlow	O=1	CLV	Xóa cờ tràn – OverFlow
SES	Đặt cờ dấu – Sign	S=1	CLS	Xóa cờ dấu – Sign
SEH	Đặt cờ nhớ phân nửa - Half	H=1	CLH	Xóa cờ nhớ phân nửa - Half
SET	Đặt cờ tạm - Temporary	T=1	CLT	Xóa cờ tạm - Temporary
SEI	Đặt cờ ngắt - Interrupt	I=1	CLI	Xóa cờ ngắt - Interrupt

➤ Lưu ý:

Vùng RAM nội của AVR không được định vị bit, tập lệnh AVR cũng không cung cấp các lệnh xử lý bit ở vùng này, nên việc thao tác lệnh trên các bit của vùng RAM nội phải được xử lý trung gian qua các thanh ghi GPRs.

Ví dụ 3.23: Viết chương trình kiểm tra giá trị của ô nhớ có địa chỉ \$195 ở RAM nội, nếu là số chẵn thì gửi ra PORTB, ngược lại thì đổi thành số chẵn và tiếp tục gửi ra PORTB.

Giải:

.EQU	MYREG = 0x195	;định nghĩa ô nhớ \$195
LDI	R16,0xFF	
OUT	DDRB,R16	; Port B: ngõ xuất

AGAIN:

```
LDS R16,MYREG  
SBRS R16,0 ;kiểm tra bit D0  
RJMP OVER ;bỏ qua lệnh này nếu D0=1  
CBR R16,0b00000001 ;đổi thành số chẵn (LSB = 0)
```

OVER:

```
OUT PORTB,R16 ;gửi ra Port B  
JMP AGAIN ;
```

Cách khác:

```
.EQU MYREG = 0x195 ;  
LDI R16,0xFF  
OUT DDRB,R16 ;
```

AGAIN:

```
LDS R16,MYREG  
CBR R16,0b00000001 ;
```

OVER:

```
OUT PORTB,R16 ;  
JMP AGAIN ;có thể thay bằng lệnh RJMP.
```

Các lệnh xoay ROR và ROL

Trong nhiều ứng dụng cần phải thực hiện xoay từng bit của một toán hạng. Tập lệnh AVR cung cấp các lệnh ROR và ROL thực hiện xoay phải hoặc xoay trái nội dung của một thanh ghi qua cờ nhớ C.

Lệnh ROR xoay tất cả các bit của thanh ghi từ trái sang phải, cờ nhớ C đưa vào vị trí MSB và LSB của thanh ghi sẽ đưa vào cờ nhớ C. Định dạng lệnh ROR như sau:

ROR Rd ;C → b₇ → b₆ → b₅ → b₄ → b₃ → b₂ → b₁ → b₀ → C, d = (0 ÷ 31)

Đoạn chương trình sau và kết quả thực thi minh họa cho lệnh ROR

```
CLC ;C = 0  
LDI R20,0x26 ;R20 = 0010 0110  
ROR R20 ;R20 = 0001 0011 C = 0  
ROR R20 ;R20 = 0000 1001 C = 1  
ROR R20 ;R20 = 1000 0100 C = 1
```

Lệnh ROL thực hiện tương tự, thực hiện xoay toàn bộ bit trong thanh ghi từ phải sang trái. Đầu tiên cờ C đưa vào LSB, cuối cùng là vị trí MSB đưa vào cờ C. Định dạng lệnh ROL như sau:

ROL Rd ;C ← b₇ ← b₆ ← b₅ ← b₄ ← b₃ ← b₂ ← b₁ ← b₀ ← C, d = (0 ÷ 31)

Ví dụ 3.24: Viết chương trình gửi tuần tự một byte dữ liệu là 41H(mỗi thời điểm truyền 1bit) ra ngõ PB1, cần truyền bit 1 ở thời điểm trước và sau khi kết thúc việc gửi byte dữ liệu, vị trí LSB được truyền trước.

Giải:

```
ORG 0  
SBI DDRB,1 ;PB1: ngõ xuất  
LDI R20,0x41 ;R20 = 41H  
CLC ;xóa cờ C  
LDI R16,8 ;R16 = 8: biến đếm số bit cần truyền  
SBI PORTB,1 ;trước khi truyền PB1= 1
```

AGAIN:

```
ROR R20 ; LSB → C  
BRCS ONE ;C = 1 nhảy đến nhãn ONE  
CBI PORTB,1 ;ngược lại gửi bit 0 ra PB1  
JMP NEXT ;nhảy đến nhãn NEXT
```

ONE:

SBI PORTB,1 ;gửi bit 1 ra PB1

NEXT:

```

DEC R16 ;giảm biến đếm
BRNE AGAIN ;nếu biến đếm khác không, lặp lại việc truyền dữ liệu
SBI PORTB, 1 ;biến đếm =0 :việc truyền kết thúc, đặt PB1=1
HERE:
JMP HERE ;dừng tại chỗ

```

Các lệnh dịch LSL, LSR và ASR

Lệnh LSL và LSR thực hiện phép dịch logic các bit trong thanh ghi sang trái hoặc sang phải 1 bit.

Lệnh LSL dịch bit 0 vào LSB, đưa MSB vào cờ C. Lệnh này có thể xem như phép nhân nội dung của thanh ghi lên 2 với điều kiện sau khi thực hiện phép dịch giá trị của cờ C = 0. Định dạng lệnh LSL như sau:

```
LSL Rd ;C ← b7 ← b6 ← b5 ← b4 ← b3 ← b2 ← b1 ← b0 ← 0, d = (0 ÷ 31)
```

Xem đoạn chương trình sau và kết quả thực hiện sẽ minh họa cho nhận xét ở trên.

```

CLC ; C = 0
LDI R20,0x26 ;R20 = 0010 0110(38), C = 0
LSL R20 ;R20 = 0100 1100(76), C = 0
LSL R20 ;R20 = 1001 1000(152), C = 0
LSL R20 ;R20 = 0011 0000(48), C = 1
; sau phép dịch C = 1 nên nội dung R20 không phải là kết quả
; của phép nhân với 2 của nội dung R20 trước đó.

```

Lệnh LSR thực hiện ngược lại với lệnh LSL, trước tiên dịch bit 0 vào MSB, vị trí LSB đưa vào cờ C. Lệnh này cũng được xem như việc thực hiện phép chia nội dung của thanh ghi cho 2, nội dung của C chính là phần dư của phép chia. Định dạng lệnh LSR như sau:

```
LSR Rd ;0 → b7 → b6 → b5 → b4 → b3 → b2 → b1 → b0 → C, d = (0 ÷ 31)
```

Xem đoạn chương trình sau và kết quả thực hiện theo từng dòng.

```

LDI R20,0x26 ;R20 = 0010 0110 (38)
LSR R20 ;R20 = 0001 0011 (19), C = 0
LSR R20 ;R20 = 0000 1001 (9), C = 1
LSR R20 ;R20 = 0000 0100 (4), C = 1

```

Lệnh ASR thực hiện phép dịch phải toán học. Lệnh thực hiện giống với phép dịch phải logic với 7 bit thấp của thanh ghi, vị trí MSB được giữ nguyên. Đây cũng có thể xem như phép chia một số có dấu cho 2. Định dạng lệnh ASR như sau:

```
ASR Rd ; b7, b6 → b5 → b4 → b3 → b2 → b1 → b0 → C, d = (0 ÷ 31)
```

Xem đoạn chương trình sau và kết quả theo từng dòng.

```

LDI R20,0xD0 ;R20 = 1101 0000(-48), C = 0
ASR R20 ;R20 = 1110 1000(-24), C = 0
ASR R20 ;R20 = 1111 0100(-12), C = 0
ASR R20 ;R20 = 1111 1010(-6), C = 0
ASR R20 ;R20 = 1111 1101(-3), C = 0
ASR R20 ;R20 = 1111 1110(-1), C = 1

```

Lệnh SWAP thực hiện hoán đổi 2 nửa byte thấp và cao của một thanh ghi. Định dạng lệnh như sau:

```
SWAP Rd ;Rd(7:4)↔Rd(3:0), d=(0÷31)
```

Đoạn lệnh sau hoán vị 4 bit thấp và cao thanh ghi R20:

```

LDI R20, 0x72 ;R20 = 0x72
SWAP R20 ;R20 = 0x27

```

❖ Ôn tập

1. Có phải tất cả các thanh ghi của AVR đều được định vị bit?
2. Thanh ghi SREG có phải là thanh ghi được định vị bit?
3. Trong các thanh ghi sau, thanh ghi nào được định vị bit?
 - (a). Port A (b). Port B (c). R19 (d). Thanh ghi trạng thái (e) Thanh ghi PC
4. Làm thế nào để kiểm tra nội dung bit D1 của thanh ghi R23 là 0 hay 1?
5. Liệt kê các câu lệnh có thể để xóa cờ nhớ Carry?
6. Cho biết kết quả được tác động sau khi thực hiện xong các lệnh sau:
 - (a) SBR R16,0x1 (b) CBR R30,0x7 (c) BST R19,2
 - (d) SBI PORTB,4 (e) CBI SREG,1 (f) CLI
7. Lệnh LSR có thể xem là phép chia của 1 số có dấu trong thanh ghi cho 2 không? Tại sao?
8. Viết đoạn lệnh thực hiện phép hoán đổi 2 nữa byte thấp và cao của một thanh ghi mà không sử dụng lệnh SWAP?

3.4.5 NHÓM LỆNH ĐIỀU KHIỂN MCU

Nhóm này gồm các lệnh NOP, BREAK, SLEEP và WDR.

Lệnh **NOP** không thực hiện thao tác gì cả nhưng tốn mất một chu kỳ máy. Lệnh này thường được sử dụng trong các chương trình tạo trễ.

Lệnh **BREAK** được sử dụng bởi hệ thống gỡ lỗi (Debug) trên chip, và thường không được sử dụng trong phần mềm ứng dụng. Khi lệnh này được thực thi, CPU của AVR được đặt ở trạng thái dừng, trình gỡ lỗi có thể truy cập các tài nguyên bên trong.

Lệnh **SLEEP** đặt MCU vào trạng thái ngủ. Trước khi thực hiện lệnh này cần phải điều khiển bit SE(Sleep Enable) trong thanh ghi MCUCR lên 1, và chọn một trong các chế độ ngủ bởi các bit SM2,SM1 và SM0 đã được đề cập ở chương 2.

Lệnh **WDR**(WatchDog Reset) khởi động lại bộ định thời Watchdog. Lệnh này được thực thi trong một khoảng thời gian được giới hạn. Các khoảng thời gian được thiết lập bởi các bit trong thanh ghi **WDTCR** và đã được trình bày chi tiết trong chương 2.