CHƯƠNG 3
Các lệnh nhảy, vòng lặp và lệnh gọi

Trong một chuỗi lệnh cần thực hiện thường có nhu cần cần chuyển điều khiển chương trình đến một vị trí khác. Có nhiều lệnh để thực hiện điều này trong 8051, ở chương này ta sẽ tìm hiểu các lệnh chuyển điều khiển có trong hợp ngữ của 8051 như các lệnh sử dụng cho vòng lặp, các lệnh nhảy có và không có điều khiển, lệnh gọi và cuối cùng là mô tả về một chương trình con giữ chậm thời gian.

3.1 Vòng lặp và các lệnh nhảy.

3.1.1 Tạo vòng lặp trong 8051.

Qúa trình lặp lại một chuỗi các lệnh với một số lần nhất định được gọi là vòng lặp. Vòng lặp là một trong những hoạt động được sử dụng rộng rãi nhất mà bất kỳ bộ vi sử lý nào đều thực hiện. Trong 8051 thì hoạt động vòng lặp được thực hiện bởi lệnh "DJNZ thanh ghi, nhãn". Trong lệnh này thanh ghi được giảm xuống, nếu nó không bằng không thì nó nhảy đến địa chỉ đích được tham chiếu bởi nhãn. Trước khi bắt đầu vòng lặp thì thanh ghi được nạp với bộ đếm cho số lần lặp lại. Lưu ý rằng, trong lệnh này việc giảm thanh ghi và quyết định để nhảy được kết hợp vào trong một lệnh đơn.

Ví dụ 3.1:

Viết một chương trình để: a) xoá ACC và sau đó b) cộng 3 vào ACC 10 lần.

Lời giải:

```
        MOV     A, #0       ; Xoá ACC,  A = 0
        MOV     R2, #10     ; Nạp bộ đếm R2 = 10
BACK:   ADD     A, #10      ; Cộng 03 vào ACC
        DJNZ    R2, AGAIN   ; Lặp lại cho đến khi R2 = 0 (10 lần)
        MOV     R5, A       ; Cắt A vào thanh ghi R5
```

Trong chương trình trên đây thanh ghi R2 được sử dụng như là bộ đếm. Bộ đếm lúc đầu được đặt bằng 10. Mỗi lần lặp lại lệnh DJNZ giảm R2 không bằng 0 thì nó nhảy đến địa chỉ đích gắn với nhãn "AGAIN". Hoạt động lặp lại này tiếp tục cho đến khi R2 trở về không. Sau khi R2 = 0 nó thoát khỏi vòng lặp và thực hiện đứng ngay dưới nó trong trường hợp này là lệnh "MOV R5, A".

Lưu ý rằng trong lệnh DJNZ thì các thanh ghi có thể là bất kỳ thanh ghi nào trong các thanh ghi R0 - R7. Bộ đếm cũng có thể là một ngăn nhớ trong RAM như ta sẽ thấy ở chương 5.

Ví dụ 3.2:

Số lần cực đại mà vòng lặp ở ví dụ 3.1 có thể lặp lại là bao nhiêu?

Lời giải:

Vì thanh ghi R2 chứa số đếm và nó là thanh ghi 8 bit nên nó có thể chứa được giá trị cực đại là FFH hay 155. Do vậy số lần lặp lại cực đại mà vòng lặp ở ví dụ 3.1 có thể thực hiện là 256.

3.2.1 Vòng lặp bền trong một vòng lặp.

Như trình bày ở ví dụ 3.2 số đếm cực đại là 256. Vậy điều gì xảy ra nếu ta muốn lặp một hành động nhiều hơn 256 lần? Để làm điều đó thì ta sử dụng một vòng lặp bên trong một vòng lặp được gọi là vòng lặp lồng (Nested Loop). Trong một vòng lặp lồng ta sử dụng 2 thanh ghi để giữ số đếm. Xét ví dụ 3.3 dưới đây.

Ví dụ 3.3:

Hãy viết một chương trình a) nạp thanh ghi ACC với giá trị 55H và b) bù ACC 700 lần.

Lời giải:

Vì 700 lớn hơn 256 (là số cực đại mà một thanh ghi vó thể chứa được) nên ta phải dùng hai thanh ghi để chứa số đếm. Đoạn mã dưới đây trình bày cách sử dụng hai thanh ghi R2 và R3 để chứa số đếm.

```
        MOV     A, #55H         ; Nạp A = 55H
        MOV     R3, #10         ; Nạp R3 = 10 số đếm vòng lặp ngoài
NEXT:   MOV     R2, #70         ; Nạp R2 = 70 số đếm vòng lặp trong
AGAIN: ` CPL    A               ; Bù thanh ghi A
        DJNZ    R2, AGAIN       ; Lặp lại 70 lần (vòng lặp trong)
        DJNZ    R3, NEXT
```

Trong chương trình này thanh ghi R2 được dùng để chứa số đếm vòng lặp trong. Trong lệnh "DJNZ R2, AGAIN" thì mỗi khi R2 = 0 nó đi thẳng xuống và lệnh "JNZ R3, NEXT" được thực hiện. Lệnh này ép CPU nạp R2 với số đếm 70 và vòng lặp trong khi bắt đầu lại quá trình này tiếp tục cho đến khi R3 trở về không và vòng lặp ngoài kết thúc.

3.1.3 Các lệnh nhảy có điều kiện.

Các lệnh nhảy có điều kiện đối với 8051 được tổng hợp trong bảng 3.1. Các chi tiết về mỗi lệnh được cho trong phụ lục AppendixA. Trong bảng 3.1 lưu ý rằng một số lệnh như JZ (nhảy nếu A = 0) và JC (nhảy nếu có nhớ) chỉ nhảy nếu một điều kiện nhất định được thoả mãn. Kế tiếp ta xét một số lệnh nhảy có điều kiện với các

Ví dụ minh hoạ sau.

a- Lệnh JZ (nhảy nếu A = 0). Trong lệnh này nội dung của thanh ghi A được kiểm tra. Nếu nó bằng không thì nó nhảy đến địa chỉ đích. Ví dụ xét đoạn mã sau:

```
MOV     A, R0       ; Nạp giá trị của R0 vào A
JZ      OVER        ; Nhảy đến OVER nếu A = 0
MOV     A, R1       ; Nạp giá trị của R1 vào A
JZ      OVER        ; Nhảy đến OVER nếu A = 0
OVER ...
```

Trong chương trình này nếu R0 hoặc R1 có giá trị bằng 0 thì nó nhảy đến địa chỉ có nhãn OVER. Lưu ý rằng lệnh JZ chỉ có thể được sử dụng đối với thanh ghi A. Nó chỉ có thể kiểm tra xem thanh ghi A có bằng không không và nó không áp dụng cho bất kỳ thanh ghi nào khác. Quan trọng hơn là ta không phải thực hiện một lệnh số học nào như đếm giảm để sử dụng lệnh JNZ như ở ví dụ 3.4 dưới đây.

Ví dụ 3.4:

Viết một chương trình để xác định xem R5 có chứa giá trị 0 không? Nếu nạp thì nó cho giá trị 55H.

Lời giải:

```
MOV     A, R5       ; Sao nội dung R5 vào A
JNZ     NEXT        ; Nhảy đến NEXT nếu A không bằng 0
MOV     R5, #55H    ;
```

```
NEXT:          ...
```

b- Lệnh JNC (nhảy nếu không có nhớ, cờ CY = 0).

Trong lệnh này thì bit cờ nhớ trong thanh ghi cờ PSW được dùng để thực hiện quyết định nhảy. Khi thực hiện lệnh "JNC nhãn" thì bộ xử lý kiểm tra cờ nhớ xem nó có được bật không (CY = 1). Nếu nó không bật thì CPU bắt đầu nạp và thực hiện các lệnh từ địa chỉ của nhãn. Nếu cờ CY = 1 thì nó sẽ không nhảy và thực hiện lệnh kế tiếp dưới JNC.

Cần phải lưu ý rằng cũng có lệnh "JC nhãn". Trong lệnh JC thì nếu CY = 1 nó nhảy đến địa chỉ đích là nhãn. Ta sẽ xét các ví dụ về các lệnh này trong các ứng dụng ở các chương sau.

Ngoài ra còn có lệnh JB (nhảy nếu bit có mức cao) và JNB (nhảy nếu bit có mức thấp). Các lệnh này được trình bày ở chương 4 và 8 khi nói về thao tác bit.

Bảng 3.1: Các lệnh nhảy có điều kiện.

Lệnh	Hoạt động
JZ	Nhảy nếu A = 0
JNZ	Nhảy nếu A 0
DJNZ	Giảm và nhảy nếu A = 0
CJNE A, byte	Nhảy nếu A byte
CJNE re, # data	Nhảy nếu Byte data
JC	Nhảy nếu CY = 1
JNC	Nhảy nếu CY = 0
JB	Nhảy nếu bit = 1
JNB	Nhảy nếu bit = 0
JBC	Nhảy nếu bit = 1 và xoá nó

Ví dụ 3.5:

Hãy tìm tổng của các giá trị 79H, F5H và E2H. Đặt vào trong các thanh ghi R0 (byte thấp) và R5 (byte cao).

Lời giải:

```
        MOV   A, #0          ; Xoá thanh ghi A = 0
        MOV   R5, A          ; Xoá R5
        ADD   A #79H         ; Cộng 79H vào A (A = 0 + 79H = 79H)
        JNC   N-1            ; Nếu không có nhớ cộng kế tiếp
        INC   R5             ; Nếu CY = 1, tăng R5

N-1:    ADD   A, #0F5H       ; Cộng F5H vào A (A = 79H + F5H = 6EH) và CY = 1
        JNC   N-2            ; Nhảy nếu CY = 0
        INC   R5             ; Nếu CY = 1 tăng R5 (R5 = 1)
N-2:    ADD   A, #0E2H       ; Cộng E2H vào A (A = GE + E2 = 50) và CY = 1
        JNC   OVER           ; Nhảy nếu CY = 0
        INC   R5             ; Nếu CY = 1 tăng R5
OVER:   MOV   R0, A          ; Bây giờ R0 = 50H và R5 = 02
```

c- Tất cả các lệnh nhảy có điều kiện đều là những phép nhảy ngắn.

Cần phải lưu ý rằng tất cả các lệnh nhảy có điều kiện đều là các phép nhảy ngắn, có nghĩa là địa chỉ của đích đều phải nằm trong khoảng -127 đến +127 byte của nội dung bộ đếm chương trình PC.

3.1.4 Các lệnh nhảy không điều kiện.

Lệnh nhảy không điều kiện là một phép nhảy trong đó điều khiển được truyền không điều kiện đến địa chỉ đích. Trong 8051 có hai lệnh nhảy không điều kiện đó là: LJMP - nhảy xa và SJMP - nhảy gần.

a- Nhảy xa LJMP:

Nhảy xa LJMP là một lệnh 3 byte trong đó byte đầu tiên là mã lệnh còn hai byte còn lại là địa chỉ 16 bit của đích. Địa chỉ đích 02 byte có phép một phép nhảy đến bất kỳ vị trí nhớ nào trong khoảng 0000 - FFFFH.

Hãy nhớ rằng, mặc dù bộ đếm chương trình trong 8051 là 16 bit, do vậy cho không gian địa chỉ là 64k byte, nhưng bộ nhớ chương trình ROM trên chíp lớn như vậy. 8051 đầu tiên chỉ có 4k byte ROM trên chíp cho không gian chương trình, do vậy mỗi byte đều rất quý giá. Vì lý do đó mà có cả lệnh nhảy gần SJMP chỉ có 2 byte so với lệnh nhảy xa LZ0MP dài 3 byte. Điều này có thể tiết kiệm được một số byte bộ nhớ trong rất nhiều ứng dụng mà không gian bộ nhớ có hạn hẹp.

b- Lệnh nhảy gồm SJMP.

Trong 2 byte này thì byte đầu tiên là mã lệnh và byte thứ hai là chỉ tương đối của địa chỉ đích. Đích chỉ tương đối trong phạm vi 00 - FFH được chia thành các lệnh nhảy tới và nhảy lùi: Nghĩa là -128 đến +127 byte của bộ nhớ tương đối so với địa chỉ hiện thời của bộ đếm chương trình. Nếu là lệnh nhảy tới thì địa chỉ đích có thể nằm trong khoảng 127 byte từ giá trị hiện thời của bộ đếm chương trình. Nếu địa chỉ đích ở phía sau thì nó có thể nằm trong khoảng -128 byte từ giá trị hiện hành của PC.

3.1.5 Tính toán địa chỉ lệnh nhảy gần.

Ngoài lệnh nhảy gần SJMP thì tất cả mọi lệnh nhảy có điều kiện như JNC, JZ và DJNZ đều là các lệnh nhảy gần bởi một thực tế là chúng đều lệnh 2 byte. Trong những lệnh này thì byte thứ nhất đều là mã lệnh, còn byte thứ hai là địa chỉ tương đối. Địa chỉ đích là tương đối so với giá trị của bộ đếm chương trình. Để tính toán địa chỉ đích byte thứ hai được cộng vào thanh ghi PC của lệnh đứng ngay sau lệnh nhảy. Để hiểu điều này hãy xét ví dụ 3.6 dưới đây.

Ví dụ 3.6:

Sử dụng tệp tin liệt kê dưới đây hãy kiểm tra việc tín toán địa chỉ nhảy về trước.

```
01    0000                        ORG     0000
02    0000    7800                MOV     R0, #0
03    0002    7455                MOV     A, #55H
04    0004    6003                JZ      NEXT
05    0006    08                  NIC     R0
06    0007    04      AGAIN: INC          A
07    0008    04                  INC     A
08    0009    2477    NEXT:       ADD     A, #77h
09    000B    5005                JNC     OVER
10    000D    E4                  CLR     A
11    000E    F8                  MOV     R0, A
12    000F    F9                  MOV     R1, A
```

```
13      0010        FA                      MOV         R2, A
14      0011        FB                      MOV         R3, A
15      0012        2B          OVER:       ADD         A, R3
16      0013        50F2                    JNC         AGAIN
17      0015        80FE        HERE:       SJMP        SHERE
18      0017                                END
```

Lời giải:

Trước hết lưu ý rằng các lệnh JZ và JNC đều là lệnh nhảy về trước. Địa chỉ đích đối với lệnh nhảy về trước được tính toán bằng cách cộng giá trị PC của lệnh đi ngay sau đó vào byte thứ hai của lệnh nhảy gần được gọi là địa chỉ tương đối. Ở dòng 04 lệnh "JZ NEXT" có mã lệnh 60 và toán hạng 03 tại địa chỉ 0004 và 0005. Ở đây 03 là địa chỉ tương đối, tương đối so với địa chỉ của lệnh kế tiếp là: "INC R0" và đó là 0006. Bằng việc cộng 0006 vào 3 thì địa chỉ đích của nhãn NEXT là 0009 được tạo ra. Bằng cách tương tự như vậy đối với dòng 9 thì lệnh "JNC OVER" có mã lệnh và toán hạng là 50 và 05 trong đó 50 là mã lệnh và 05 là địa chỉ tương đối. Do vậy, 05 được cộng vào OD là địa chỉ của lệnh "CLA A" đứng ngay sau lệnh "JNC OVER" và cho giá trị 12H chính là địa chỉ của nhãn OVER.

Ví dụ 3.7:

Hãy kiểm tra tính toán địa chỉ của các lệnh nhảy lùi trong ví dụ 3.6.

Lời giải:

Trong danh sách liệt kê chương trình đó thì lệnh "JNC AGAIN" có mã lệnh là 50 và địa chỉ tương đối là F2H. Khi địa chỉ tương đối của F2H được cộng vào 15H là địa chỉ của lệnh đứng dưới lệnh nhảy ta có 15H + F2H = 07 (và phần nhớ được bỏ đi). Để ý rằng 07 là địa chỉ nhãn AGAIN. Và hãy cũng xét lệnh "SJMP HERE" có mã lệnh 80 và địa chỉ tương đối FE giá trị PC của lệnh kế tiếp là 0017H được cộng vào địa chỉ tương đối FEH ta nhận được 0015H chính là địa chỉ nhãn HERE (17H + FEH = 15H) phần nhớ được bỏ đi). Lưu ý rằng FEH là -2 và 17h + (-2) = 15H. Về phép cộng số âm sẽ được bàn ở chương 6.

3.1.6 Tính toán địa chỉ đích nhảy lùi.

Trong khi ở trường hợp nhảy tới thì giá trị thay thế là một số dương trong khoảng từ 0 đến 127 (00 đến 7F ở dạng Hex) thì đối với lệnh nhảy lùi giá trị thay thế là một số âm nằm trong khoảng từ 0 đến -128 như được giải thích ở ví dụ 3.7.

Cần phải nhấn mạnh rằng, bất luận SJMP nhảy tới hay nhảy lùi thì đối với một lệnh nhảy bất kỳ địa chỉ của địa chỉ đích không bao giờ có thể lớn hơn 0 -128 đến +127 byte so với địa chỉ gắn liền với lệnh đứng ngay sau lệnh SJMP. Nếu có một sự nỗ lực nào vi phạm luật này thì hợp ngữ sẽ tạo ra một lỗi báo rằng lệnh nhảy ngoài phạm vi.

3.2 Các lệnh gọi CALL.

Một lệnh chuyển điều khiển khác là lệnh CALL được dùng để gọi một chương trình con. Các chương trình con thường được sử dụng để thực thi các công việc cần phải được thực hiện thường xuyên. Điều này làm cho chương trình trở nên có cấu trúc hơn ngoài việc tiết kiệm được thêm không gian bộ nhớ. Trong 8051 có 2 lệnh để gọi đó là: Gọi xa CALL và gọi tuyệt đối ACALL mà quyết định sử dụng lệnh nào đó phụ thuộc vào địa chỉ đích.

3.2.1 Lệnh gọi xa LCALL.

Trong lệnh 3 byte này thì byte đầu tiên là mã lệnh, còn hai byte sau được dùng cho địa chỉ của chương trình con đích. Do vậy LCALL có thể được dùng để gọi các chương trình con ở bất kỳ vị trí nào trong phạm vi 64k byte, không gian địa chỉ của 8051. Để đảm bảo rằng sau khi thực hiện một chương trình được gọi để 8051 biết được chỗ quay trở về thì nó tự động cất vào ngăn xếp địa chỉ của lệnh đứng ngay sau lệnh gọi LCALL. Khi một chương trình con được gọi, điều khiển được chuyển đến chương trình con đó và bộ xử lý cất bộ đếm chương trình PC vào ngăn xếp và bắt đầu nạp lệnh vào vị trí mới. Sau khi kết thúc thực hiện chương trình con thì lệnh trở về RET chuyển điều khiển về cho nguồn gọi. Mỗi chương trình con cần lệnh RET như là lệnh cuối cùng (xem ví dụ 3.8).

Các điểm sau đây cần phải được lưu ý từ ví dụ 3.8.

1. Lưu ý đến chương trình con DELAY khi thực hiện lệnh "LCALL DELAY" đầu tiên thì địa chỉ của lệnh ngay kế nó là "MOV A, #0AAH" được đẩy vào ngăn xếp và 8051 bắt đầu thực hiện các lệnh ở địa chỉ 300H.
2. Trong chương trình con DELAY, lúc đầu bộ đếm R5 được đặt về giá trị 255 (R5 = FFH). Do vậy, vòng lặp được lặp lại 256 lần. Khi R5 trở về 0 điều khiển rơi xuống lệnh quay trở về RET mà nó kéo địa chỉ từ ngăn xếp vào bộ đếm chương trình và tiếp tục thực hiện lệnh sau lệnh gọi CALL.

Ví dụ 3.8:

Hãy viết một chương trình để chốt tất cả các bit của cổng P1 bằng cách gửi đến nó giá trị 55H và AAH liên tục. Hãy đặt một độ trễ thời gian giữa mỗi lần xuất dữ liệu tới cổng P1. Chương trình này sẽ được sử dụng để kiểm tra các cổng của 8051 trong chương tiếp theo.

Lời giải:

```
                ORG     0000
BACK:           MOV     A, #55H         ; Nạp A với giá trị 55H
                MOV     P1, A           ; Gửi 55H đến cổng P1
                LCALL   DELAY           ; Tạo trễ thời gian
                MOV     A, #0AAH        ; Nạp A với giá trị  AAH
                MOV     P1, A           ; Gửi AAH đến cổng P1
                LCALL   DELAY           ; Giữ chậm
                SJMP    BACK            ; Lặp lại vô tận
; ------------------ - Đây là chương trình con tạo độ trễ thời gian
                ORG     300H            ; Đặt chương trình con trễ thời gian ở địa chỉ 300H
DELAY:          MOV     R5, #00H        ; Nạp bộ đếm R5 = 255H (hay FFH)
AGAIN:          DJNZ    R5, AGAIN       ; Tiếp tục cho đến khi R5 về không
                RET                     ; Trả điều khiển về nguồn gọi (khi R5 = 0)
                END                     ; Kết thúc tệp tin của hợp ngữ
```

Lượng thời gian trễ trong ví dụ 8.3 phục thuộc vào tần số của 8051. Cách tính chính xác thời gian sẽ được giải thích ở chương 4. Tuy nhiên ta có thể tăng thời gian độ trễ bằng cách sử dụng vòng lặp lồng như chỉ ra dưới đây.

```
DELAY:                                  ; Vòng lặp lồng  giữ chậm
                MOV     R4, #255        ; Nạp R4 = 255 (FFH dạng hex)
NEXT:           MOV     R5, #255        ; Nạp R5  = 255 (FFH dạng hex)
AGAIN:          DJNZ    R5, AGAIN       ; Lặp lại cho đến khi RT = 0
                DJNZ    R4, NEXT        ; Giảm R4
                                        ;Tiếp tục nạp R5 cho đến khi R4 = 0
```

```
        RET                     ; Trở về (khi R4 =  0)
```

3.2.2 Lệnh gọi CALL và vai trò của ngăn xếp.

Ngăn xếp và con trỏ ngăn xếp ta sẽ nghiên cứu ở chương cuối. Để hiểu được tầm quan trọng của ngăn xếp trong các bộ vi điều khiển bây giờ khảo sát nội dung của ngăn xếp và con trỏ ngăn xếp đối với ví dụ 8.3. Điều này được trình bày ở ví dụ 3.9 dưới đây.

Ví dụ 3.9:

Hãy phân tích nội dung của ngăn xếp sau khi thực hiện lệnh LCALL đầu tiên dưới đây.

```
001     0000                    OR6
002     0000    7455    BACK:   MOV   A, #55H      ; Nạp A với giá trị 55H
003     0002    F590            MOV   P1, A        ; Gửi 55H tới cổng P1
004     0004    120300          LCALL DELAY        ; Tạo trễ thời gian
005     0007    74AA            MOV   A, #0AAH     ; Nạp A với giá trị AAH
006     0009    F590            MOV   P1, A        ; Gửi AAH tới cổng      P1
007     000B    120300          LCALL DELAY        ; Tạo trễ thời gian
008     000E    80F0            SJMP  BACK         ; Tiếp tục thực hiện
009     0010
010     0010                    ; ..................... Đây là chương trình con giữ chậm
011     0300                    MOV   300H
012     0300            DELAY:
013     0300    7DFF            MOV   R5, #FFH     ; Nạp R5 = 255
014     0302    DDFE    AGAIN:  DJNZ  R5, AGAIN    ; Dừng ở đây
015     0304    22              RET                ; Trở về nguồn gọi
016     0305                    END                ; Kết thúc nạp tin hợp ngữ
```

Lời giải:

Khi lệnh LCALL đầu tiên được thực hiện thì địa chỉ của lệnh "MOV A, #0AAH" được cất vào ngăn xếp. Lưu ý rằng byte thấp vào trước và byte cao vào sau. Lệnh cuối cùng của chương trình con được gọi phải là lệnh trở về RET để chuyển CPU kéo (POP) các byte trên đỉnh của ngăn xếp vào bộ đếm chương trình PC và tiếp tục thực hiện lệnh tại địa chỉ 07. Sơ đồ bên chỉ ra khung của ngăn xếp sau lần gọi LCALL đầu tiên.

0A		
09		00
08		07
SP	=	09

3.2.3 Sử dụng lệnh PUSH và POP trong các chương trình con.

Khi gọi một chương trình con thì ngăn xếp phải bám được vị trí mà CPU cần trở về. Sau khi kết thúc chương trình con vì lý do này chúng ta phải cẩn thận mỗi khi thao tác với các nội dung của ngăn xếp. Nguyên tắc là số lần đẩy vào (PUSH) và kéo ra (POP) luôn phải phù hợp trong bất kỳ chương trình con được gọi vào. Hay nói cách khác đối với mỗi lệnh PUSH thì phải có một lệnh POP. Xem ví dụ 3.10.

3.2.4 Gọi các chương trình con.

Trong lập trình hợp ngữ thường có một chương trình chính và rất nhiều chương trình con mà chúng được gọi từ chương trình chính. Điều này cho phép ta tạo mới chương trình con trong một mô-đun riêng biệt. Mỗi mô-đun có thể được kiểm tra tách biệt và sau đó được kết hợp với nhau cùng với chương trình chính. Quan trọng hơn là trong một chương trình lớn thì các mô-đun có thể được phân cho các lập trình viên khác nhau nhằm rút ngắn thời gian phát triển.

Ví dụ 3.10:

Phân tích ngăn xếp đối với lệnh LCALL đầu tiên trong đoạn mã.

```
01  0000                        ORG       0
02  0000  7455        BACK:     MOV       A, #55H       ; Nạp A với giá trị 55H
03  0002  F590                  MOV       P1, A         ; Gửi 55H ra cổng P1
04  0004  7C99                  MOV       R4, #99H
05  0006  7D67                  MOV       R5, #67H
06  0008  120300                LCALL     DELAY         ; Tạo giữ chậm thời gian
07  000B  74AA                  MOV       A, #0AAH      ; Nạp A với AAH
08  000D  F590                  MOV       P1, A         ; Gửi AAH ra cổng P1
09  000F  120300                LCALL     DELAY
10  0012  80EC                  SJMP      BACK          ; Tiếp tục thực hiện
11  0014                        ; ..... Đây là chương trình con DELAY
12  0300                        ORG       300H
13  0300  C004        DELAY     PUSH      4             ; Đẩy R4 vào ngăn xếp
14  0302  C005                  PUSH      5             ; Đẩy R5 vào ngăn xếp
15  0304  7CFF                  MOV       R4, 00FH      ; Gán R4 = FFH
16  0306  7DFF        NEXT:     MOV       R5,  #00FH    ; Gán R5 = 255
17  0308  DDFE        AGAIN:    DJNZ      R5, AGAIN
18  030A  DCFA                  DJNZ      R4, NEXT
19  030C  D005                  POP       5             ; Kéo đỉnh ngăn xếp vào R5
20  030E  D004                  POP       4             ; Kéo đỉnh ngăn xếp vào R4
21  0310  22                    RET                     ; Trở về nguồn gọi
22        0311                  END                     ; Kết thúc tệp tin hợp ngữ
```

Lời giải:

Trước hết lưu ý rằng đối với các lệnh PUSH và POP ta phải xác định địa chỉ trực tiếp của thanh ghi được đẩy vào, kéo ra từ ngăn xếp. Dưới đây là sơ đồ khung của ngăn xếp.

Sau lệnh LCALL thứ nhất			**Sau lệnh PUSH 4**			**Sau lệnh POSH 5**		
0B			0B			0B	67	R5
0A			0A	99	R4	0A	09	R4
09	00	PCH	09	00	PCH	09	00	PCL
08	0B	PCL	0B	0B	PCL	08	0B	PCL

Cần phải nhấn mạnh rằng trong việc sử dụng LCALL thì địa chỉ đích của các chương trình con có thể ở đâu đó trong phạm vi 64k byte không gian bộ nhớ của 8051. Điều này không áp dụng cho tất cả mọi lệnh gọi CALL chẳng hạn như đối với ACALL dưới đây:

```
; MAIN program calling subroutines
                ORG             0
MAIN:           LCALL           SUBR-1
                LCALL           SUBR-2
                LCALL           SUBR-3

HERE:           SJMP            MAIN
;---------------- end of MAIN
;
SUBR-1l         ...
                ...
                RET
; ---------------- end of subroutinel  1
; SUBR-1l       ...
                ...
                RET
; ---------------- end of subroutinel  2
 ; SUBR-1l      ...
                ...
                RET
; ---------------- end of subroutinel  3
                END                     ; end of the asm    file
```

Hình 3.1: Chương trình chính hợp ngữ của 8051 có gọi các chương trình con.

3.2.5 Lệnh gọi tuyệt đối ACALL (Absolute call).

Lệnh ACALL là lệnh 2 byte khác với lệnh LCALL dài 3 byte. Do ACALL chỉ có 2 byte nên địa chỉ đích của chương trình con phải nằm trong khoảng 2k byte địa chỉ vì chỉ có 11bit của 2 byte được sử dụng cho địa chỉ. Không có sự khác biệt nào giữa ACALL và LCALL trong khái niệm cất bộ đếm chương trình vào ngăn xếp hay trong chức năng của lệnh trở về RET. Sự khác nhau duy nhất là địa chỉ đích của lệnh LCALL có thể nằm bất cứ đâu trong phạm vi 64k byte không gian địa chỉ của 8051, còn trong khi đó địa chỉ của lệnh ACALL phải nằm trong khoảng 2 byte. Trong nhiều biến thế của 8051 do các hãng cung cấp thì ROM trên chíp chỉ có 1k byte.. Trong những trường hợp như vậy thì việc sử dụng ACALL thay cho LCALL có thể tiết kiệm được một số byte bộ nhớ của không gian ROM chương trình.

Ví dụ 3.11:

Một nhà phát triển sử dụng chíp vi điều khiển Atmel AT89C1051 cho một sản phẩm. Chíp này chỉ có 1k byte ROM Flash trên chíp. Hỏi trong khi lệnh LCALL và ACALL thì lệnh nào hữu ích nhất trong lập trình cho chíp này.

Lời giải:

Lệnh ACALL là hữu ích hơn vì nó là lệnh 2 byte. Nó tiết kiệm một byte mỗi lần gọi được sử dụng.

Tất nhiên, việc sử dụng các lệnh gọn nhẹ, chúng ta có thể lập trình hiệu quả bằng cách có một hiểu biết chi tiết về tất cả các lệnh được hỗ trợ bởi bộ vi xử lý đã cho và sử dụng chúng một cách khôn ngoan. Xét ví dụ 3.12 dưới đây.

Ví dụ 3.12:

Hãy viết lại chương trình ở ví dụ 3.8 một cách hiệu quả mà bạn có thể:

Lời giải:

```
        ORG         0
        MOV         A, #55H         ; Nạp Avới giá trị 55H
BACK:   MOV         P1, A           ; Xuất giá trị trong A ra cổng P1
        ACALL       DELAY           ; Giữ chậm
        CPL         A               ; Bù thành ghi A
        SJMP        BACK            ; Tiếp tục thực hiện vô hạn
; -------- Đây là chương trình con giữ chậm DELAY
DELAY:
        MOV         R5,  #0FFH      ; Nạp R5 = 255 (hay FFH) làm cho bộ đếm
AGAIN:  DJNZ        R5, AGAIN       ; Dừng ở đây cho đến khi R5  = 0
        RET                         ; Trở về
        END                         ; Kết thúc
```

3.3 Tạo và tính toán thời gian giữ chậm.

3.3.1 Chu kỳ máy:

Đối với CPU để thực hiện một lệnh thì mất một chu kỳ đồng hồ này được coi như các chu kỳ máy. Phụ lục AppendixA.2 cung cấp danh sách liệt kê các lệnh 8051 và các chu kỳ máy của chúng. Để tính toán một độ trễ thời gian, ta sử dụng danh sách liệt kê này. Trong họ 8051 thì độ dài của chu kỳ máy phụ thuộc vào tần số của bộ dao động thạch anh được nối vào hệ thống 8051. Bộ dao động thạch anh cùng với mạch điện trên chip cung cấp xung đồng hồ cho CPU của 8051 (xem chương 4). Tần số của tinh thể thạch anh được nối tới họ 8051 dao động trong khoảng 4MHz đến 30 MHz phụ thuộc vào tốc độ chíp và nhà sản xuất. Thường xuyên nhất là bộ dao động thạch anh tần số 10.0592MHz được sử dụng để làm cho hệ 8051 tương thích với cổng nối tiếp của PC IBM (xem chương 10). Trong 8051, một chu kỳ máy kéo dài 12 chu kỳ dao động. Do vậy, để tính toán chu kỳ máy ta lấy 1/12 của tần số tinh thể thạch anh, sau đó lấy giá trị nghịch đảo như chỉ ra trong ví dụ 3.13.

Ví dụ 3.13:

Đoạn mã dưới đây trình bày tần số thạch anh cho 3 hệ thống dựa trên 8051 khác nhau. Hãy tìm chu kỳ máy của mỗi trường hợp: a) 11.0592MHz b) 16MHz và c) 20MHz.

Lời giải:

a) 11.0592/12 = 921.6kHz; Chu kỳ máy là 1/921.6kHz = 1.085μs (micro giây)
b) 16MHz/12 = 1.333MHz; Chu kỳ máy MC = 1/1.333MHz = 0.75μs
c) 20MHz/12 = 1.66MHz $\Rightarrow$ MC = 1/1.66MHz = 0.60μs

Ví dụ 3.14:

Đối với một hệ thống 8051 có 11.0592MHz hãy tìm thời gian cần thiết để thực hiện các lệnh sau đây.

```
a) MOV  R3, #55     b) DEC  R3      c) DJNZ R2 đích
d) LJMP e) SJMP     f) NOP  g) MUL  AB
```

Lời giải:

Chu kỳ máy cho hệ thống 8051 có tần số đồng hồ là 11.0592MHz Là 1.085μs như đã tính ở ví dụ 3.13. Bảng A-1 trong phụ lục Appendix A trình bày số chu kỳ máy đối với các lệnh trên. Vậy ta có:

Lệnh	Chu kỳ máy	Thời gian thực hiện
(a) MOV R3, #55	1	1 × 1.085 μs = 1.085 μs
(b) DEC R3	1	1 × 1.085 μs = 1.085 μs
(c) DJNZ R2, target	2	2 × 1.085 μs = 2.17 μs
(d) LJMP	2	2 × 1.085 μs = 2.17 μs
(e) SJMP	2	2 × 1.085 μs = 2.17 μs
(f) NOP	1	1 × 1.085 μs = 1.085 μs
(g) MUL AB	4	4 × 1.085 μs = 4.34 μs

3.3.2 Tính toán độ trễ.

Như đã trình bày ở trên đây, một chương trình con giữ chậm gồm có hai phần: (1) thiết lập bộ đếm và (2) một vòng lặp. Hầu hết thời gian giữ chậm được thực hiện bởi thân vòng lặp như trình bày ở ví dụ 3.15.

Ví dụ 3.15:

Hãy tìm kích thước của thời gian giữ chậm trong chương trình sau, nếu tần số giao động thach anh là 11.0592MHz.

```
                MOV     A, #55H
AGAIN: MOV              P1, A
                ACALL   DELAY
                CPL     A
                SJMP    AGAIN
; --------  Time delay
DELAY: MOV              R3, #200
HERE  :         DJNZ    R3, HERE
                RET
```

Lời giải:

Từ bảng A-1 của phụ lục Appendix A ta có các chu kỳ máy sao cho các lệnh của chương trình con giữ chậm là:

```
DELAY:      MOV     R3, #200        1
HERE  :     DJNZ    R3, HERE        2
            RET                     1
```

Do vậy tổng thời gian giữ chậm là $[(200 \times 2) + 1 + 1] \times 1.085 = 436.17\mu s$.

Thông thường ta tính thời gian giữ chậm dựa trên các lệnh bên trong vòng lặp và bỏ qua các chu kỳ đồng hồ liên quan với các lệnh ở ngoài vòng lặp.

Trong ví dụ 3.15 giá trị lớn nhất mà R3 có thể chứa là 255, do vậy một cách tăng độ trễ là sử dụng lệnh UOP (không làm gì) trong vòng lặp để tiêu tốn thời gian một cách đơn giản. Điều này được chỉ ra trong ví dụ 3.16 dưới đây.

Ví dụ 3.16:

Hãy tìm độ trễ thời gian cho chương trình con sau. Giả thiết tần số dao động thạch anh là 11.0592MHz.

```
                                        Số chu kỳ máy
DELAY:    MOV       R3, #250            1

HERE :    NOP                           1
          NOP                           1
          NOP                           1
          NOP                           1
          DJNZ      R3, HERE            2

          RET                           1
```

Lời giải:

Thời gian trễ bên trong vòng lặp HERE là [250 (1 + 1 + 1 + 1 + 1 + 2)] × 1.0851μs = 1627.5μs. Cộng thêm hai lệnh ngoài vòng lặp ta có 1627.5μs × 1.085μs = 1629.67μs.

3.3.3 Độ trễ thời gian của vòng lặp trong vòng lặp.

Một cách khác để nhận được giá trị từ độ trễ lớn là sử dụng một vòng lặp bên trong vòng lặp và cũng được gọi là vòng lặp lồng nhau. Xem ví dụ 3.17 dưới đây.

Ví dụ 3.17:

Đối với một chu kỳ máy 1.085μs hãy tính thời gian giữ chậm trong chương trình con sau:

```
DELAY:                                  chu kỳ máy
          MOV       R2, #200            1
AGAIN:    MOV       R3, #250            1
HERE:     NOP                           1
          NOP                           1
          DJNZ      R3, HERE            2
          DJNZ      R2, AGAIN           2
          RET                           1
```

Lời giải:

Đối với vòng lặp HERE ta có (4 × 250) × 1.085μs = 1085μs. Vòng lặp AGAIN lặp vòng lặp HERE 200 lần, do vậy thời gian trễ là 200 × 1085μs 217000μs, nên ta không tính tổng phí. Tuy nhiên, các lệnh "MOV R3, #250" và "DJNZ R2, AGAIN" ở đầu và cuối vòng lặp AGAIN cộng (3 × 200 × 1.085μs) = 651μs vào thời gian trễ và kết quả ta có 217000 + 651 = 217651μs = 217.651 miligiây cho tổng thời gian trễ liên quan đến chương trình con giữ chậm DELAY nói trên. Lưu ý rằng, trong trường hợp vòng lặp lồng nhau cũng như trong mọi vòng lặp giữ chậm khác thời gian xấp xỉ gần dúng vì ta bỏ qua các lệnh đầu và cuối trong chương trình con.