

แผนบริหารการสอนประจำบทที่ 11

หัวข้อเนื้อหา

- คำสั่งที่รองรับการเรียกโปรแกรมย่อย
- การประกาศโปรแกรมย่อย
- คำสั่งเก็บข้อมูล (PUSH) และดึงข้อมูล (POP) จากแอสตัก
- ตัวอย่างการใช้งานโปรแกรมย่อย

วัตถุประสงค์เชิงพฤติกรรม

- มีความรู้และความเข้าใจเกี่ยวกับคำสั่งที่รองรับการเรียกโปรแกรมย่อยและการประกาศโปรแกรมย่อย
- มีความรู้และความเข้าใจเกี่ยวกับกลุ่มคำสั่งเก็บข้อมูล (PUSH) และดึงข้อมูล (POP) จากแอสตัก

วิธีสอนและกิจกรรมการเรียนการสอน

- บรรยาย
- สืบเสาะหาความรู้
- ค้นคว้าเพิ่มเติม
- ตอบคำถาม

สื่อการเรียนการสอน

- สื่ออิเล็กทรอนิกส์
- ตอบคำถาม
- ภาพ
- เอกสารอ้างอิงประกอบการค้นคว้า

การวัดผลและประเมินผล

ใช้วิธีการสังเกตและจดบันทึกไว้เป็นระยะ

- สังเกตจากงานที่กำหนดให้ไปทำมาส่ง
- สังเกตจากการตอบคำถาม
- สังเกตจากการนำความรู้ไปใช้

การประเมินผล

วิธีตรวจผลงานต่างๆ ที่ให้ทำ

- ตรวจผลงานภาคปฏิบัติ
- ตรวจรายงาน
- ตรวจแบบฝึกหัด

ใช้วิธีการออกข้อสอบข้อเขียน

บทที่ 11 โปรแกรมย่อยขั้นต้น (The initial Subprogram)

รูปแบบของโปรแกรมภาษาแอสเซมบลีที่เราเขียนในบทก่อน ๆ จะมีส่วนของโปรแกรมหลายส่วนที่ซ้ำซ้อนกัน เราสามารถที่จะแยกส่วนย่อยเหล่านั้นเป็นโปรแกรมย่อยที่มีความอิสระจากโปรแกรมหลักได้ การแยกโปรแกรมเป็นโปรแกรมย่อยนี้ ทำให้เราสามารถนำส่วนของโปรแกรมนั้นมาใช้ใหม่ได้สะดวก และการตรวจสอบและแก้ไขโปรแกรมยังสามารถกระทำได้ง่ายขึ้นด้วย

คำสั่งที่รองรับการเรียกโปรแกรมย่อย

การเรียกโปรแกรมย่อยมีความแตกต่างกับการกระโดดทั่วไป เนื่องจากภายหลังที่โปรแกรมย่อยทำงานเสร็จ หน่วยประมวลผลจะต้องสามารถกระโดดกลับมาทำงานในโปรแกรมหลักต่อไปได้ ดังนั้นการเรียกใช้โปรแกรมย่อยนั้นจะต้องมีการเก็บตำแหน่งของคำสั่งที่ทำงานอยู่เดิมด้วย และเมื่อจบโปรแกรมย่อยโปรแกรมจะต้องกระโดดกลับมาทำงานที่เดิม โดยใช้ข้อมูลที่เก็บไว้

คำสั่งของ 8086 ที่รองรับการใช้งานโปรแกรมย่อยคือคำสั่ง **CALL** และ คำสั่ง **RET** เมื่อผู้ใช้เรียกคำสั่ง CALL พร้อมทั้งระบุตำแหน่งของโปรแกรมย่อย หน่วยประมวลผลจะเก็บตำแหน่งของคำสั่งถัดไปที่จะกลับมาทำงานลงในแอสตัก และจะกระโดดไปทำงานที่โปรแกรมย่อย เมื่อโปรแกรมย่อยทำงานเสร็จ โปรแกรมย่อยจะเรียกใช้คำสั่ง RET เพื่อกระโดดกลับมาทำงานในโปรแกรมหลักต่อไป เมื่อหน่วยประมวลผลประมวลผลคำสั่ง RET หน่วยประมวลผลจะดึงค่าตำแหน่งที่โปรแกรมจะกระโดดกลับไปทำงานจากแอสตัก และกระโดดกลับไปทำงานต่อยังโปรแกรมหลัก

การประกาศโปรแกรมย่อย

เราสามารถสร้างโปรแกรมย่อยโดยการประกาศเลเบลที่จุดเริ่มต้นของโปรแกรมย่อยเท่านั้นก็ได้ แต่โดยทั่วไปแล้วเราจะประกาศโปรแกรมย่อยโดยใช้คู่ของคำสั่งเทียม PROC และ ENDP ดังตัวอย่างโปรแกรมที่ 11.1 โปรแกรมตัวอย่างนี้เป็นโปรแกรมที่สร้างโปรแกรมย่อยสำหรับพิมพ์เลขฐานสิบหกหนึ่งหลักชื่อ printhexdigit และใช้โปรแกรมย่อยนี้ในการเขียนโปรแกรมพิมพ์ค่ารหัสแอสกีของปุ่มที่รับค่าจากผู้ใช้ (ในตัวอย่างได้ละโปรแกรมบางส่วนไว้)

```
.model small
.dosseg

.stack 100h

.code

;procedure PrintHexDigit
;input al : digit
;affect ah,dl
```

```

;
printhexdigit    proc    near
    mov    ah,2
    mov    dl,al
    add    dl,'0'
    cmp    al,10
    jb    printit
    add    dl,'A'-'0'-10
printit:
    int    21h
    ret
printhexdigit    endp

start:

; ...
; ละส่วนอ่านการกดปุ่มและส่วนหาค่าของตัวเลขของแต่ละหลักไว้
; ให้ตัวเลขหลักหน้าเก็บใน bh และหลักหลังเก็บใน bl
; ...

    mov    al,bh
    call   printhexdigit

    mov    al,bl
    call   printhexdigit

    mov    ax,4c00h
    int    21h
end    start

```

โปรแกรมที่ 11.1 ส่วนของโปรแกรมพิมพ์ค่ารหัสแอสกีเป็นเลขฐานสิบหก

รูปแบบของการประกาศโปรแกรมน้อยมีลักษณะดังนี้

```

proc_name  PROC      NEAR
           actions
proc_name  ENDP

```

คำสั่งเทียม NEAR ที่ต่อจากคำสั่งเทียม PROC เป็นการระบุว่าโปรแกรมย่อยนี้เป็นโปรแกรมย่อยที่อยู่ในเซกเมนต์เดียวกันกับโปรแกรมหลัก และมีการเรียกใช้แบบใกล้ ซึ่งจะมีผลในขั้นตอนการเก็บตำแหน่งที่จะกระโดดกลับมาทำงานหลังการทำงานของโปรแกรมย่อย

การทำงานของโปรแกรมย่อยจากตัวอย่างมีการเปลี่ยนแปลงค่าของรีจิสเตอร์ AH และรีจิสเตอร์ DL การเปลี่ยนค่าของรีจิสเตอร์ทั้งสองอาจทำให้เกิดผลข้างเคียงกับโปรแกรมหลักได้ถ้าโปรแกรมหลักมีการใช้งานรีจิสเตอร์ทั้งสองด้วยเช่นเดียวกัน เราควรจะลดการเกิดผลข้างเคียงนี้โดยให้โปรแกรมย่อยเก็บค่าของรีจิสเตอร์ต่าง ๆ ที่โปรแกรมย่อยใช้และคืนค่าเดิมให้กับรีจิสเตอร์เหล่านั้นหลังการทำงาน เรานิยมใช้แอสตึกในการเก็บค่าของรีจิสเตอร์เป็นการชั่วคราวเนื่องจากเราสามารถเก็บข้อมูลลงในแอสตึกได้โดยไม่ต้องจองเนื้อที่ล่วงหน้า และการเก็บข้อมูลในลักษณะของแอสตึก มีความสอดคล้องกับการทำงานแบบโปรแกรมย่อย

คำสั่งเก็บข้อมูลและดึงข้อมูลจากแอสตึก : คำสั่ง PUSH และคำสั่ง POP

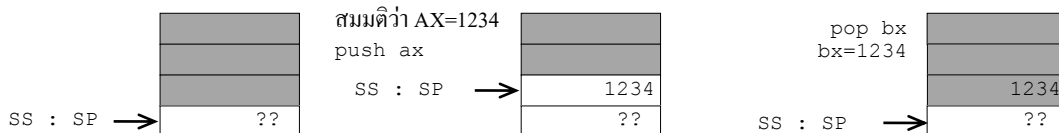
เราสามารถใช้คำสั่ง PUSH ในการเก็บข้อมูลลงในแอสตึก และคำสั่ง POP สำหรับการเรียกข้อมูลออกมาจากแอสตึก. คำสั่งทั้งสองมีรูปแบบดังนี้

```
push regs16      push mem
pop  regs16      pop  mem
```

ข้อมูลที่จะเก็บลงในแอสตึกจะต้องมีขนาด 16 บิตเท่านั้น โดยการเก็บข้อมูลในแอสตึกจะมีลักษณะเป็นแบบใส่ที่หลังดึงออกก่อน การเก็บข้อมูลลงในแอสตึกจึงต้องระวังลำดับของการเก็บและการดึงข้อมูลออกไปด้วย

การทำงานของแอสตึกจะมีรีจิสเตอร์ SS และ SP เป็นรีจิสเตอร์ที่ใช้เก็บตำแหน่งของข้อมูลที่เก็บลงไปอันล่าสุด โดย SS จะเก็บเซกเมนต์ของแอสตึก และ SP จะเก็บออฟเซตของข้อมูลล่าสุด เมื่อมีการเก็บข้อมูลเพิ่มลงในแอสตึก หรือมีการดึงข้อมูลออกไปก็จะมีการปรับค่าของรีจิสเตอร์ทั้งสองนี้ การทำงานของแอสตึกมีลักษณะดังรูปที่

11.1



รูปที่ 11.1 การทำงานของแอสตึก

โปรแกรมย่อยที่มีการเก็บค่าของรีจิสเตอร์ต่าง ๆ

เราสามารถแก้โปรแกรมย่อยในตัวอย่างให้มีการรักษาค่าในรีจิสเตอร์ต่าง ๆ ได้ดังโปรแกรมที่ 1.2

```
;procedure PrintHexDigit
;input  al : digit
;affect ah,dl
;
printhexdigit      proc      near
```

```

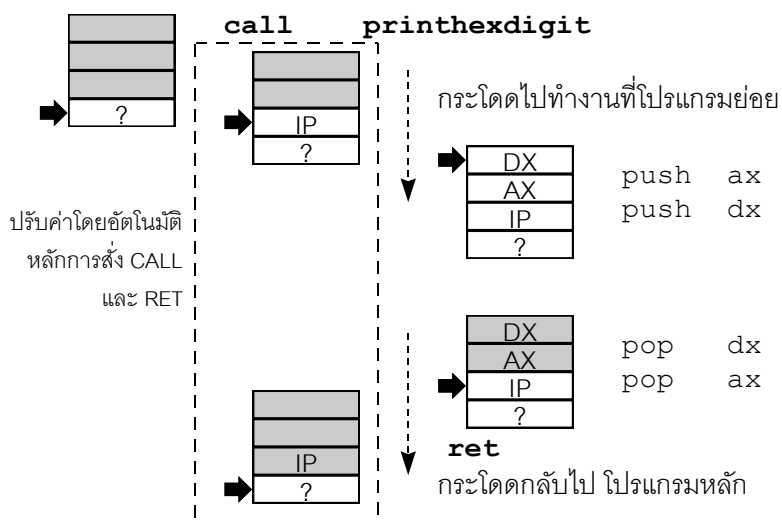
push    ax
push    dx
mov     ah,2
mov     dl,al
add     dl,'0'
cmp     al,10
jb     printit
add     dl,'A'-'0'-10

printit:
int     21h
pop     dx
pop     ax
ret

printhexdigit    endp
    
```

โปรแกรมที่ 1.2 ตัวอย่างโปรแกรมย่อยที่รักษาค่าของรีจิสเตอร์

สังเกตลำดับในการ push และ pop ของโปรแกรมย่อย ลำดับในการ push จะตรงกันข้ามกับลำดับในการ pop เนื่องจากการทำงานของแอสตักเป็นแบบข้อมูลที่ใส่ทีหลังจะถูกดึงออกก่อน การเปลี่ยนแปลงของแอสตักในการทำงานดังกล่าวแสดงได้โดยรูปที่ 11.2



รูปที่ 11.2 แสดงการเปลี่ยนแปลงของแอสตักในการเรียกใช้โปรแกรมย่อย

เมื่อมีการเรียกใช้โปรแกรมย่อยโดยคำสั่ง CALL ค่าของรีจิสเตอร์ IP ซึ่งเก็บตำแหน่งของคำสั่งถัดไปจะถูก PUSH ลงใน แอสตักโดยอัตโนมัติ จากนั้นโปรแกรมจะกระโดดไปทำงานที่โปรแกรมย่อย ในโปรแกรมย่อยมีการ

เก็บค่าของรีจิสเตอร์ AX และรีจิสเตอร์ DX ลงในแอสต์ก หลังการทำงานโปรแกรมย่อยได้คืนค่าของรีจิสเตอร์ทั้งสองโดยการดึงค่าจากแอสต์ก และเรียกใช้คำสั่ง RET เพื่อกระโดดกลับไปทำงานยังโปรแกรมหลักต่อ สังเกตว่าถ้าเราเก็บและดึงค่าออกจากแอสต์กได้ไม่ถูกต้องการกระโดดกลับไปทำงานยังโปรแกรมหลักอาจมีการผิดพลาดได้

ตัวอย่างการใช้งานโปรแกรมย่อย

โปรแกรมตัวอย่างต่อไปนี้สร้างโปรแกรมย่อยขึ้นมาสองโปรแกรมย่อย โปรแกรมย่อยแรกเป็นโปรแกรมย่อยที่พิมพ์เลขฐานสิบหกหนึ่งหลัก ส่วนโปรแกรมย่อยที่สองเป็นโปรแกรมย่อยที่พิมพ์เลขฐานสิบหกขนาด 1 ไบต์ โดยเรียกใช้งานโปรแกรมย่อยโปรแกรมแรก

```
.model small
.dosseg

.stack 100h

.code
;procedure PrintHexDigit
;display one hex digit
;input : al = the digit
printhexdigit    proc    near
    push    ax
    push    dx
    mov     ah,2
    mov     dl,al
    add     dl,'0'           ;make ascii from the number
    cmp     al,10
    jb     printit         ;if above 9 adjust the ascii value
    add     dl,'A'-'0'-10
printit:
    int     21h           ;print the number
    pop     dx
    pop     ax
    ret
printhexdigit    endp

;procedure PrintHexNumber
;display one hex number (1 byte)
```

```

;input : al = the number
printhexnumber    proc
    push    ax
    push    bx
    mov     bl,16          ;div by 16 to get 2 digits
    mov     ah,0
    div     bl             ;al=high digit , ah=low digit
    call    printhexdigit ;print the high digit
    mov     al,ah
    call    printhexdigit ;print the next digit
    pop     bx
    pop     ax
    ret
printhexnumber    endp
start:
    mov     ah,1
    int     21h
    call    printhexnumber
    mov     ax,4c00h
    int     21h
end    start

```

สรุป

การแยกโปรแกรมเป็นโปรแกรมน้อยๆนี้ ทำให้เราสามารถนำส่วนของโปรแกรมนั้นมาใช้ใหม่ได้สะดวก และการตรวจสอบและแก้ไขโปรแกรมยังสามารถกระทำได้ง่ายขึ้นด้วย ส่วนการเรียกโปรแกรมน้อยมีความแตกต่างกับการกระโดดทั่วไป เนื่องจากภายหลังที่โปรแกรมน้อยทำงานเสร็จ หน่วยประมวลผลจะต้องสามารถกระโดดกลับมาทำงานในโปรแกรมหลักต่อไปได้ ดังนั้นการเรียกใช้โปรแกรมน้อยนั้นจะต้องมีการเก็บตำแหน่งของคำสั่งที่ทำงานอยู่เดิมด้วย และเมื่อจบโปรแกรมน้อยโปรแกรมจะต้องกระโดดกลับมาทำงานที่เดิม โดยใช้ข้อมูลที่เก็บไว้

คำสั่งของ 8086 ที่รองรับการใช้งานโปรแกรมน้อยคือคำสั่ง CALL และ คำสั่ง RET เมื่อผู้ใช้เรียกคำสั่ง CALL พร้อมทั้งระบุตำแหน่งของโปรแกรมน้อย หน่วยประมวลผลจะเก็บตำแหน่งของคำสั่งถัดไปที่จะกลับมาทำงานลงในแอสตีก และจะกระโดดไปทำงานที่โปรแกรมน้อย เมื่อโปรแกรมน้อยทำงานเสร็จ โปรแกรมย่อยจะเรียกใช้คำสั่ง RET เพื่อกระโดดกลับมาทำงานในโปรแกรมหลักต่อไป เมื่อหน่วยประมวลผลประมวลผลคำสั่ง RET หน่วยประมวลผลจะดึงค่าตำแหน่งที่โปรแกรมจะกระโดดกลับไปทำงานจากแอสตีก และกระโดดกลับไปทำงานต่อยังโปรแกรมหลักต่อไป

คำถามทบทวน

1. จงอธิบายการทำงานของคำสั่ง CALL และคำสั่ง RET
2. การประกาศโปรแกรมย่อยในภาษาแอสเซมบลีโดยทั่วไปมักจะใช้คู่ของคำสั่งเทียมอะไร
3. คำสั่งเก็บข้อมูลและดึงข้อมูลจากแอสต์กคำสั่ง PUSH และคำสั่ง POP มีขั้นตอนการทำงานอย่างไร
4. การทำงานของแอสต์กจะมีรีจิสเตอร์ที่เกี่ยวข้องอะไรบ้างและแต่ละรีจิสเตอร์มีหน้าที่และทำงานอย่างไร
5. จงอธิบายเหตุการณ์ที่เกิดขึ้นหลังจากมีการเรียกใช้โปรแกรมย่อยโดยคำสั่ง CALL