แผนบริหารการสอนประจำบทที่ 9

หัวข้อเนื้อหา

- คำสั่งกระโดด
- คำสั่งวนรอบ

วัตถุประสงค์เชิงพฤติกรรม

- มีความรู้และความเข้าใจเกี่ยวกับคำสั่งกระโดดแบบไม่มีเงื่อนไข คำสั่งกระโดดที่พิจารณาค่าจากแฟล็ก และคำสั่งกระโดดที่พิจารณาค่าจากรีจิสเตอร์
- มีความรู้และความเข้าใจเกี่ยวกับกลุ่มคำสั่งวนรอบ เช่น LOOP LOOPZ และ LOOPNZ เป็นต้น

วิธีสอนและกิจกรรมการเรียนการสอน

- บรรยาย
- สืบเสาะหาความรู้
- ค้นคว้าเพิ่มเติม
- ตอบคำถาม

สื่อการเรียนการสอน

- สื่ออิเล็กทรอนิกส์
- ตอบคำถาม
- ภาพ
- เอกสารอ้างอิงประกอบการค้นคว้า

การวัดผลและประเมินผล

ใช้วิธีการสังเกตและจดบันทึกไว้เป็นระยะ

- สังเกตจากงานที่กำหนดให้ไปทำมาส่ง
- สังเกตจากการตอบคำถาม
- สังเกตจากการนำความรู้ไปใช้

การประเมินผล

วิธีตรวจผลงานต่างๆ ที่ให้ทำ

- ตรวจผลงานภาคปฏิบัติ
- ตรวจรายงาน
- ตรวจแบบฝึกหัด

ใช้วิธีการออกข้อสอบข้อเขียน

บทที่ 9 คำสั่งกระโดดและการกระทำซ้ำ (Jump and Iteration Loop)

ในบทนี้เราจะศึกษาเกี่ยวกับเรื่องคำสั่งกระโดด และคำสั่งเกี่ยวกับการทำซ้ำ เป้าหมายของบทนี้คือการนำคำสั่งเหล่านี้ไปใช้ในการสร้างโครงสร้างควบคุม (Control Structures) แบบต่าง ๆ ซึ่งเราจะศึกษาต่อไปในบทที่ 10

คำสั่งกระโดด

คำสั่งกระโดดเป็นคำสั่งที่สั่งให้หน่วยประมวลผลกระโดดไปทำงานที่ตำแหน่งอื่น รูปแบบทั่วไปของคำสั่งกระโดดคือ

Jxx label

คำสั่งกระโดดแบ่งได้เป็น 2 กลุ่ม คือ คำสั่งกระโดดแบบไม่มีเงื่อนไข และคำสั่งกระโดดแบบมีเงื่อนไข คำสั่งกระโดดแบบไม่มีเงื่อนไขคือคำสั่ง JMP ส่วนในกลุ่มของคำสั่งกระโดดแบบมีเงื่อนไขแบบคร่าว ๆ ออกเป็นสองกลุ่มคือกลุ่มซึ่งพิจารณาการกระโดดจากค่าในแฟล็ก และกลุ่มที่พิจารณาการกระโดดจากค่าในรีจิสเตอร์ ส่วนใหญ่คำสั่งกระโดดที่พิจารณาค่าในแฟล็กจะใช้ผลลัพธ์ที่ได้จากคำสั่ง CMP คำสั่งกระโดดต่าง ๆ สรุปได้ดังตารางที่ 9.1

คำสั่งกระโดด	ความหมาย	เงื่อนไข
<u>คำสั่งกระโดดแบบไม่มีเงื่อนไข</u>		
JMP	Jump	Always
<u>คำสั่งกระโดดที่พิจารณาค่าจากแฟล็ก</u>		
Zero flag		
JZ (JE)	Jump if Zero (Jump if Equal)	ZF = 1
JNZ (JNE)	Jump if Not Zero (Jump if Not Equal)	ZF = 0
Overflow flag		
JO	Jump if Overflow	OF = 1
JNO	Jump if Not Overflow	OF = 0
Parity flag		
JPO	Jump if Parity Odd	PF = 0
JPE	Jump if Parity Even	PF = 1
Signs flag		
JS	Jump if Sign	SF = 1
JNS	Jump if No Sign	SF = 0

คำสั่งกระโดด	ความหมาย	เงื่อนไข
Carry flag		
JC	Jump if Carry	CF = 1
JNC	Jump if No Carry	CF = 0
Comparing unsigned numbers		
JA (JNBE)	Jump if Above (Not Below or Equal)	(CF and ZF) = 0
JB (JNAE)		CF = 1
JAE (JNB)	Jump if Below (Not Above or Equal)	CF = 0
JBE (JNA)		(CF or ZF) = 1
	Jump if Above or Equal (Not Below)	
	Jump if Below or Equal (Not Above)	
Comparing signed numbers		
JG (JNLE)	Jump if Greater (Not Less or Equal)	ZF = 0 and SF = OF
JL (JNGE)		SF <> OF
JGE (JNL)	Jump if Less (Not Greater or Equal)	SF = OF
JLE (JNG)		(ZF = 1) of (SF <> OF)
	Jump if Greater or Equal (Not Less)	
	Jump if Less or Equal (Not Greater)	

ตารางที่ 6.1 คำสั่งกระโดดต่าง ๆ

คำสั่งกระโดด	ความหมาย	เงื่อนไข
<u>คำสั่งกระโดดที่พิจารณาค่าจากรีจิสเตอร์</u>		
Testing for CX		
JCXZ	Jump if CX is equal to zero	CX = 0

คำสั่งต่าง ๆ เหล่านี้จะใช้ในการสร้างโครงสร้างควบคุมการทำงานของโปรแกรม โดยจะใช้ประกอบกับคำสั่ง CMP ดังที่ได้กล่าวมาแล้ว ยกเว้นคำสั่ง JCXZ จะนิยมใช้ประกอบกับกลุ่มคำสั่งประเภทการทำซ้ำ (LOOP) คำสั่งที่ใช้ควบคุมการกระโดดแบบมีเงื่อนไขที่ใช้การเปรียบเทียบระหว่างโอเปอร์แรนด์สองตัวของคำสั่ง CMP มีสองกลุ่มคือ กลุ่มที่คิดการเปรียบเทียบเป็นการเปรียบเทียบของเลขไม่คิดเครื่องหมาย (JA JB JAE JBE)

และกลุ่มที่คิดเป็นเลขคิดเครื่องหมาย (JG JL JGE JLE) ในการใช้คำสั่งกระโดดทั้งสองกลุ่มนี้เราจะต้องพิจารณาข้อมูลที่เปรียบเทียบกันด้วย.

ตัวอย่าง

```
(1)             cmp     ah,10           ; เปรียบเทียบ ah กับ 10
                jz      lab1            ; ถ้าเท่ากันให้กระโดดไปที่ lab1
                mov     bx,2
        lab1:   add     cx,10

(2)             cmp     ah,10           ; เปรียบเทียบ ah กับ 10
                jge     tenup           ; ถ้ามากกว่าหรือเท่ากับให้กระโดดไปที่ tenup
                add     dl,'0'
                jmp     endif           ; กระโดดไปที่ endif
        tenup:
                add     dl,'A'
        endif:

(3)     getonechar:
                mov     ah,1            ; ใช้บริการหมายเลข 1 : อ่านอักขระ
                int     21h
                cmp     al,'Q'          ; เปรียบเทียบ al กับ 'Q'
                jne     getonechar      ; ถ้าไม่เท่ากันให้กระโดดไปที่ getonechar (กลับไปรับตัวอักษรใหม่)

(4)             mov     ah,02           ; บริการหมายเลข 2 : พิมพ์อักขระ
                mov     dl,32           ; เริ่มที่ ช่องว่าง ASCII = 32 (' ')
        printloop:
                cmp     dl,128          ; เปรียบเทียบ dl กับ 128 (ASCII สุดท้าย)
                ja      finish          ; ถ้ามากกว่ากระโดดไปที่ finish
                int     21h             ; พิมพ์อักขระที่มี ASCII = dl
                inc     dl              ; เพิ่ม dl
                jmp     printloop
        finish:
```

คำสั่งวนรอบ

คำสั่งที่ใช้กระทำซ้ำใน 8086 ยังมีอีกกลุ่มหนึ่งที่ใช้ค่าในรีจิสเตอร์ CX (Counter Register) ในการนับจำนวนครั้งของการทำงาน คำสั่งกลุ่มนี้คือ LOOP LOOPZ และ LOOPNZ

คำสั่ง LOOP

คำสั่ง LOOP จะลดค่าของรีจิสเตอร์ CX ลงหนึ่ง. ถ้า CX มีค่าไม่เท่ากับศูนย์คำสั่ง LOOP จะกระโดดไปทำงานที่เลเบลที่ระบุ รูปแบบของคำสั่ง LOOP เป็นดังนี้

```
LOOP  label
```

คำสั่ง LOOP จะลดค่าของรีจิสเตอร์ CX โดย**ไม่กระทบกับแฟล็ก**. นั่นคือคำสั่ง LOOP มีผลเหมือนคำสั่ง

```
DEC   CX
JNZ   label
```

แต่จะไม่มีผลกระทบกับแฟล็ก

ตัวอย่างการใช้คำสั่ง LOOP

โปรแกรมตัวอย่างนี้คำนวณผลรวมของเลขตั้งแต่ 1 ถึง 20

```
        mov     cx,20               ; ทำซ้ำ 20 ครั้ง
        mov     bl,1                ; เริ่มจาก 1
        mov     dx,0                ; กำหนดค่าเริ่มต้นให้กับผลรวม
addonenumber:
        add     dl,bl               ; บวก 8 บิตล่าง
        adc     dh,0                ; รวมตัวทด
        inc     bl                  ; ตัวถัดไป
        loop    addonenumber        ; ถ้า CX ลดลงแล้วไม่เท่ากับ 0  ทำซ้ำต่อไป
```

คำสั่ง JCXZ

ในกรณีที่ CX มีค่าเท่ากับศูนย์ก่อนการกระทำซ้ำโดยใช้คำสั่ง LOOP ผลลัพธ์จากการลดค่าจะมีค่าเท่ากับ 0FFFFh ทำให้จำนวนรอบของการทำงานไม่ถูกต้อง เรานิยมใช้คำสั่ง JCXZ ในการป้องกันความผิดพลาดในกรณีที่ค่าของรีจิสเตอร์ CX มีค่าเท่ากับ 0 โดยปกติถ้า CX มีค่าเท่ากับศูนย์ เราจะ สั่งให้โปรแกรมกระโดดไปที่จุดสิ้นสุดการกระทำซ้ำ ดังตัวอย่าง

```
        initialization
        jcxz    endloop             ; CX =0 ?
label1:
        actions
        loop    label1              ; loop
endloop:
```

คำสั่ง LOOPZ และ LOOPNZ

คำสั่ง LOOPZ และ LOOPNZ มีลักษณะทำงานเหมือนคำสั่ง LOOP แต่จะนำค่าของแฟล็กมาใช้ในการพิจารณาการกระโดดด้วย. คำสั่ง LOOPZ จะลดค่าของรีจิสเตอร์ CX โดยไม่กระทบแฟล็กและจะกระโดดไปที่เลเบลที่ระบุเมื่อ CX มีค่าไม่เท่ากับศูนย์ และ แฟล็กศูนย์มีค่าเป็น 1 (ผลลัพธ์ของคำสั่งก่อนหน้ามีค่าเท่ากับศูนย์) สังเกตว่า คำสั่ง LOOPZ จะทำงานเหมือนคำสั่ง LOOP แต่แฟล็กศูนย์จะต้องมีค่าเป็นหนึ่งด้วย คำสั่งนี้ถึงจะกระโดดไปที่เลเบลที่กำหนด ในทำนองกลับกันคำสั่ง LOOPNZ จะกระโดดไปทำงานเมื่อ CX มีค่าไม่เท่ากับศูนย์และแฟล็กศูนย์มีค่าเป็น 0 คำสั่งทั้งสองนิยมใช้ในการทำซ้ำที่ทราบจำนวนครั้งแต่มีเงื่อนไขในการทำซ้ำ

ตัวอย่างคำสั่ง LOOPZ และ LOOPNZ

ตัวอย่างนี้แสดงการใช้คำสั่ง LOOPNZ ในการค้นหาข้อมูลค่าหนึ่งจากชุดของข้อมูล. ในตัวอย่างนี้จำนวนข้อมูลคือ 100 ค่า และข้อมูลที่ต้องการค้นหาเก็บที่รีจิสเตอร์ DX

```
.data
datalistdw      100 dup (?)

.code

        ...                         ;
```

```
        mov     bx,offset datalist      ; ให้ BX เก็บค่าตำแหน่งของ datalist
        mov     cx,100                  ; ทำซ้ำ 100 ครั้ง
        dec     bx,2                    ; ลดค่าของ BX เพราะใน loop มีการเพิ่มค่า
checkdata:
        inc     bx,2                    ; BX ชี้ไปยังข้อมูลตัวถัดไป
        cmp     dx,[bx]                 ; เปรียบเทียบ
        loopnz  checkdata               ; ทำซ้ำถ้ายังไม่พบข้อมูลและยังไม่ครบข้อมูล
        jz      found                   ; ค้นเจอข้อมูลกระโดดไปที่ found

        ; not found                     ; ไม่พบข้อมูล

        ...

found:
        ; found                        ; พบข้อมูล

        ...
```

ในตัวอย่างแรกนี้คำสั่ง DEC BX,2 ในโปรแกรมก่อนที่จะถึงส่วนที่ทำงานซ้ำเป็นการปรับค่าของ BX ให้สอดคล้องกับการปรับค่าในส่วนที่ทำงานซ้ำ. การลดค่าของ BX ในกรณีนี้ทำให้การเปรียบเทียบครอบคลุมถึงค่าแรกของข้อมูลด้วย. การใช้เทคนิคเช่นนี้ในโปรแกรมอาจทำให้โปรแกรมทำงานได้เร็วขึ้น แต่อาจทำให้ผู้อื่นที่มาอ่านโปรแกรมของเราเข้าใจผิดได้. ดังนั้นถ้าเราใช้เทคนิคต่าง ๆ ในโปรแกรม เราควรใส่หมายเหตุให้ชัดเจน และโดยปกติเรายังสามารถใช้วิธีอื่นในการจัดการกับข้อมูลตัวแรกได้ ดังตัวอย่างถัดไป.

ตัวอย่างนี้เป็นการค้นหาอักษรตัวแรกของข้อความที่ไม่ใช่ช่องว่าง โดยข้อความนี้มีความยาว 100 ตัวอักษร ในตัวอย่าง ตำแหน่งเริ่มต้นของข้อความเก็บอยู่ที่ BX

```
        cmp     byte ptr [bx],' '       ; พิจารณาอักษรตัวแรก
        jnz     found                   ; อักษรตัวแรกไม่ใช่ช่องว่าง
        mov     cx,100                  ; ทำซ้ำ 100 ครั้ง
findnotspace:
        inc     bx                      ; BX ชี้ไปยังอักษรตัวถัดไป
        cmp     byte ptr [bx],' '       ; เปรียบเทียบ
        loopz   findnotspace            ; ทำซ้ำถ้ายังพบช่องว่างและยังไม่ครบข้อมูล
        jnz     found                   ; ค้นเจออักษรตัวแรก

        ; not found                     ; ข้อความมีแต่ช่องว่าง
        ...

found:
        ; found                        ; พบอักษรตัวแรก
        ...
```

ตัวอย่างโปรแกรม

ตัวอย่างที่ 1

โปรแกรมตัวอย่างต่อไปนี้เป็นโปรแกรมที่พิมพ์ค่าของรหัสแอสกีที่รับมาเป็นเลขฐานสิบหก. การแสดงตัวเลขเป็นเลขฐานสิบหกนั้นมีความยุ่งยากเพราะอักษร ‘0’ ถึง ‘F’ ที่จะใช้ในการแสดงค่านั้นมีรหัสแอสกีที่แยกออกเป็นสองช่วง. ช่วงแรกเป็นช่วยของตัวเลขเริ่มที่รหัส 48 ของเลขศูนย์ อีกกลุ่มหนึ่งคือช่วงของตัวอักษรเริ่มที่รหัส 65 ของตัว ‘A’. ดังนั้นในการแสดงผลเราจะต้องตรวจสอบตัวเลขในแต่ละหลักว่าอยู่ในช่วงใดโดยใช้การเปรียบเทียบ

```
;
; display ASCII in HEX
;
.model small
.dosseg

.data
newline         db      10,13,'$'

.stack  100h

.code
start:
        mov     ax,@data                ;set DS
        mov     ds,ax

        mov     ah,01h          ;Function 1 : Read char
        int     21h                     ;AL=ASCII

        mov     ah,0
        mov     bl,16                   ;div AL by 16
        div     bl
        mov     cl,al                   ;first digit
        mov     bl,ah                   ;second digit

        mov     dx,offset newline       ;display newline
        mov     ah,09h
```

```
        int     21h

        mov     dl,cl           ;print first digit
        cmp     cl,9
        ja      overnine1       ;above 9
        add     dl,'0'
        jmp     print1
overnine1:
        add     dl,'A'-10
print1:
        mov     ah,2
        int     21h

        mov     dl,bl           ;print second digit
        cmp     bl,9
        ja      overnine2
        add     dl,'0'
        jmp     print2
overnine2:
        add     dl,'A'-10
print2:
        mov     ah,2
        int     21h

        mov     ax,4C00h
        int     21h
end     start
```

ตัวอย่างที่ 2

ตัวอย่างนี้เป็นโปรแกรมที่รับข้อความจากผู้ใช้ และรับตัวอักษรที่ผู้ใช้ต้องการตรวจสอบว่ามีในข้อความหรือไม่. โปรแกรมจะแสดงคำตอบว่า YES หรือ NO โปรแกรมนี้ค้นหาตัวอักษรโดยใช้คำสั่ง LOOPNZ

```
; Find a character in a string
.model   small
.dosseg

.data
maxlen   db       30
strlen   db       ?
str      db       30 dup (?)

msg1     db       'Enter string :$'
msg2     db       10,13,'Enter character to find :$'
yesmsg   db       10,13,'YES',10,13,'$'
nomsg    db       10,13,'NO',10,13,'$'

.stack   100h

.code
start:
         mov      ax,@data
         mov      ds,ax

         mov      dx,offset msg1          ;disp msg1
         mov      ah,09h                  ;func 9
         int      21h

         mov      dx,offset maxlen
         mov      ah,0Ah                  ;read string
         int      21h

         mov      dx,offset msg2          ;disp msg2
```

```
                mov     ah,09h
                int     21h

                mov     ah,01h          ;read char
                int     21h
                mov     dl,al           ;dl=al=char

                mov     bx,offset str
                mov     cl,strlen
                mov     ch,0
                jcxz    notfound
                cmp     [bx],al         ;first char
                jz      found
                dec     cx              ;first char was compared
                jcxz    notfound        ;so we dec cx
compareloop:
                inc     bx              ;next char
                cmp     [bx],al
                loopnz  compareloop     ;loop
                jz      found           ;found?
notfound:
                mov     dx,offset nomsg ;set dx
                jmp     print
found:
                mov     dx,offset yesmsg ;set dx
print:
                mov     ah,09h          ;display msg
                int     21h             ;using func 09h
                mov     ax,4C00h
                int     21h
end     start
```

สังเกตว่าโปรแกรมนี้เราใช้คำสั่ง JCXZ ในการป้องกันกรณีที่ผู้ใช้ป้อนอักษรเพียงตัวเดียวหรือไม่ป้อนเลย. เราเลือกที่จะทดสอบตัวอักษรตัวแรก แทนที่จะใช้วิธีลดค่าของ BX ในการจัดการเกี่ยวกับข้อมูลตัวแรก

สรุป

คำสั่งกระโดดแบ่งได้เป็น 2 กลุ่ม คือ คำสั่งกระโดดแบบไม่มีเงื่อนไข และคำสั่งกระโดดแบบมีเงื่อนไข คำสั่งกระโดดแบบไม่มีเงื่อนไขคือคำสั่ง JMP ส่วนในกลุ่มของคำสั่งกระโดดแบบมีเงื่อนไขแบบคร่าว ๆ ออกเป็นสองกลุ่มคือกลุ่มซึ่งพิจารณาการกระโดดจากค่าในแฟล็ก และกลุ่มที่พิจารณาการกระโดดจากค่าในรีจิสเตอร์ ส่วนใหญ่คำสั่งกระโดดที่พิจารณาค่าในแฟล็กจะใช้ผลลัพธ์ที่ได้จากคำสั่ง CMP

คำถามทบทวน

1.จงอธิบายพร้อมยกตัวอย่างคำสั่งกระโดดแบบไม่มีเงื่อนไขและคำสั่งกระโดดแบบมีเงื่อนไขว่ามีอะไรบ้าง

2.จงอธิบายพร้อมยกตัวอย่างคำสั่ง CMP กลุ่มที่คิดการเปรียบเทียบเป็นการเปรียบเทียบของเลขไม่คิดเครื่องหมายว่ามีอะไรบ้าง

3.จงอธิบายพร้อมยกตัวอย่างคำสั่ง CMP กลุ่มที่กลุ่มที่คิดเป็นเลขคิดเครื่องหมายว่ามีอะไรบ้าง

4. จงอธิบายการโปรแกรมตัวอย่างนี้คำนวณผลรวมของเลขตั้งแต่ 25 ถึง 100 แต่ละบรรทัดสำหรับการเรียกใช้ฟังก์ชั่นจากส่วนของโปรแกรมที่แสดงอยู่ข้างล่างนี้

```
        mov     cx,100
        mov     bl,25
        mov     dx,0
addonenumber:
        add     dl,bl
        adc     dh,0
        inc     bl
        loop    addonenumber
```