

แผนการสอนประจำสัปดาห์ที่ 10

หัวข้อเนื้อหา โปรแกรมย่อยขั้นต้น (The initial Subprogram)

รายละเอียด

รูปแบบของโปรแกรมภาษาแอสเซมบลีที่เขียนในบทก่อน ๆ จะมีส่วนของโปรแกรมหลายส่วนที่ซ้ำซ้อนกัน สามารถที่จะแยกส่วนย่อยเหล่านั้นเป็นโปรแกรมย่อยที่มีความอิสระจากโปรแกรมหลักได้ การแยกโปรแกรมเป็นโปรแกรมย่อยนี้ ทำให้สามารถนำส่วนของโปรแกรมนั้นมาใช้ใหม่ได้สะดวก และการตรวจสอบและแก้ไขโปรแกรมยังสามารถกระทำได้ง่ายขึ้นด้วย

จำนวนชั่วโมงที่สอน 3 ชั่วโมง/สัปดาห์

กิจกรรมการเรียนการสอน

1. บรรยาย
2. สืบเสาะหาความรู้
3. ค้นคว้าเพิ่มเติม
4. ตอบคำถาม

สื่อการสอน

1. สื่ออิเล็กทรอนิกส์
2. เพาเวอร์พอยต์ 프리เซนเตชัน
3. บทเรียนออนไลน์
4. เอกสารอ้างอิงประกอบการค้นคว้า

แผนการประเมินผลการเรียนรู้

1. ผลการเรียนรู้

- 1.1 สังเกตจากงานที่กำหนดให้ไปทำมาส่ง
- 1.2 สังเกตจากการตอบคำถาม
- 1.3 สังเกตจากการนำความรู้ไปใช้

2. วิธีการประเมินผลการเรียนรู้

- 2.1 ตรวจผลงานภาคปฏิบัติ
- 2.2 ตรวจรายงาน
- 2.3 ตรวจแบบฝึกหัด

3. สัดส่วนของการประเมิน

- 3.1 ใบบงานที่นักศึกษาทำมาส่ง
- 3.2 คะแนนเก็บในชั้นเรียน
- 3.3 การเข้าชั้นเรียน

เนื้อหาที่สอน

ในสัปดาห์ที่ 10 การจัดการเรียนการสอน จะเกี่ยวข้องกับคำสั่งที่รองรับการเรียกโปรแกรมย่อย การประกาศโปรแกรมย่อย คำสั่งเก็บข้อมูล (PUSH) และดึงข้อมูล (POP) จากแอสตีกและตัวอย่างการใช้งานโปรแกรมย่อย ซึ่งรูปแบบของโปรแกรมภาษาแอสแซมบลีที่ได้ศึกษาก่อนหน้านี้จะมีส่วนของโปรแกรมหลายส่วนที่ซ้ำซ้อนกัน สามารถที่จะแยกส่วนย่อยเหล่านั้นเป็นโปรแกรมย่อยที่มีความอิสระจากโปรแกรมหลักได้การแยกโปรแกรมเป็นโปรแกรมย่อยนี้ ทำให้สามารถนำส่วนของโปรแกรมนั้นมาใช้ใหม่ได้สะดวก อีกทั้งการตรวจสอบและแก้ไขโปรแกรมยังสามารถกระทำได้ง่ายขึ้นด้วย

10.1 คำสั่งที่รองรับการเรียกโปรแกรมย่อย

การเรียกโปรแกรมย่อยมีความแตกต่างกับการกระโดดทั่วไป เนื่องจากภายหลังที่โปรแกรมย่อยทำงานเสร็จ หน่วยประมวลผลจะต้องสามารถกระโดดกลับมาทำงานในโปรแกรมหลักต่อไปได้ ดังนั้นการเรียกใช้โปรแกรมย่อยนั้นจะต้องมีการเก็บค่า ณ ตำแหน่งของคำสั่งที่ทำงานอยู่เดิมด้วยและเมื่อจบโปรแกรมย่อยโปรแกรมจะต้องกระโดดกลับมาทำงาน ณ

ตำแหน่งเดิมโดยใช้ข้อมูลที่เก็บไว้ คำสั่งใน 8086 ที่รองรับการใช้งานโปรแกรมย่อยคือ คำสั่ง CALL และ คำสั่ง RET เมื่อผู้ใช้เรียกคำสั่ง CALL พร้อมทั้งระบุตำแหน่งของโปรแกรมย่อย หน่วยประมวลผลจะเก็บตำแหน่งของคำสั่งถัดไปที่จะกลับมาทำงานลงในแอสติกและจะกระโดดไปทำงานที่โปรแกรมย่อย เมื่อโปรแกรมย่อยทำงานเสร็จ โปรแกรมย่อยจะเรียกใช้คำสั่ง RET เพื่อกระโดดกลับมาทำงานในโปรแกรมหลักต่อไป (ศัพท์บัญญัติ ราชบัณฑิตยสถาน, 2544)

10.2 การประกาศโปรแกรมย่อย

ผู้เขียนโปรแกรมสามารถสร้างโปรแกรมย่อยโดยการประกาศเลเบลที่จุดเริ่มต้นของโปรแกรมย่อยเท่านั้นก็ได้ แต่โดยทั่วไปแล้วจะประกาศโปรแกรมย่อยโดยใช้คู่กับคำสั่งเทียม PROC และ ENDP

ตัวอย่าง โปรแกรมที่ 10.1 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:
; ...
; แต่ละส่วนอ่านการกดปุ่มและส่วนหาค่าของตัวเลขของแต่ละหลักไว้
; ให้ตัวเลขหลักหน้าเก็บใน bh และหลักหลังเก็บใน bl
; ...
                mov     al,bh
                call    printhexdigit
                mov     al,bl
                call    printhexdigit
                mov     ax,4c00h
                int     21h
end            start

```

โปรแกรมที่ 10.1 ส่วนของโปรแกรมพิมพ์ค่ารหัสแอสกีเป็นเลขฐานสิบหก

รูปแบบของการประกาศโปรแกรมย่อยมีลักษณะดังนี้

```

proc_name      PROC      NEAR
                actions
proc_name      ENDP

```

คำสั่งเทียม NEAR ที่ต่อจากคำสั่งเทียม PROC เป็นการระบุว่าโปรแกรมย่อยนี้เป็นโปรแกรมย่อยที่อยู่ในเซกเมนต์เดียวกับโปรแกรมหลัก และมีการเรียกใช้แบบใกล้ ซึ่งจะมีผลในขั้นตอนการเก็บตำแหน่งที่จะกระโดดกลับมาทำงานหลังการทำงานของโปรแกรมย่อย

การทำงานของโปรแกรมย่อยจากตัวอย่างมีการเปลี่ยนแปลงค่าของรีจิสเตอร์ AH และรีจิสเตอร์ DL ซึ่งการเปลี่ยนค่าของรีจิสเตอร์ทั้งสองตัวอาจทำให้เกิดผลกระทบข้างเคียงกับโปรแกรมหลักได้ถ้าโปรแกรมหลักมีการใช้งานรีจิสเตอร์ทั้งสองด้วยเช่นเดียวกัน ควรจะลดการเกิดผลข้างเคียงนี้โดยให้โปรแกรมย่อยเก็บค่าของรีจิสเตอร์ต่าง ๆ ที่โปรแกรมย่อยใช้และคืนค่าเดิมให้กับรีจิสเตอร์เหล่านั้นหลังการทำงาน นิยมใช้แอสติกในการเก็บค่าของรีจิสเตอร์เป็นการชั่วคราวเนื่องจากสามารถเก็บข้อมูลลงในแอสติกได้โดยไม่ต้องจองเนื้อที่ล่วงหน้าและการเก็บข้อมูลในลักษณะของแอสติกมีความสอดคล้องกับการทำงานแบบโปรแกรมย่อย

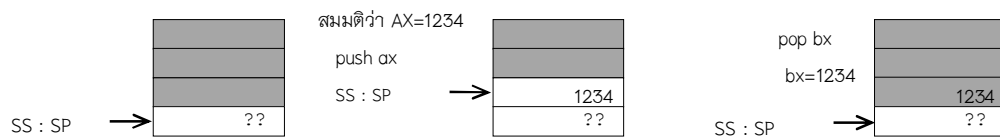
10.3 คำสั่งเก็บข้อมูลและดึงข้อมูลจากแอสตัก : คำสั่ง PUSH และคำสั่ง POP

สามารถใช้คำสั่ง PUSH ในการเก็บข้อมูลลงในแอสตัก และคำสั่ง POP สำหรับการเรียกข้อมูลออกมาจากแอสตัก คำสั่งทั้งสองมีรูปแบบดังนี้

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

ข้อมูลที่จะเก็บลงในแอสตักจะต้องมีขนาด 16 บิตเท่านั้น โดยการเก็บข้อมูลในแอสตักจะมีลักษณะเป็นแบบไล่ที่หลังดึงออกก่อน การเก็บข้อมูลลงในแอสตักจึงต้องระวังลำดับของการเก็บและการดึงข้อมูลออกไปด้วย

การทำงานของแอสตักจะมีรีจิสเตอร์ SS และ SP เป็นรีจิสเตอร์ที่ใช้เก็บตำแหน่งของข้อมูลที่เก็บลงไปอันล่าสุด โดย SS จะเก็บเซกเมนต์ของแอสตัก และ SP จะเก็บออฟเซตของข้อมูลล่าสุด เมื่อมีการเก็บข้อมูลเพิ่มลงในแอสตัก หรือมีการดึงข้อมูลออกไปก็จะมีการปรับค่าของรีจิสเตอร์ทั้งสองนี้ การทำงานของแอสตัก แสดงได้ดังภาพที่ 10.1



ภาพที่ 10.1 แสดงการทำงานของแอสตัก

โปรแกรมย่อยที่มีการเก็บค่าของรีจิสเตอร์ต่าง ๆ

สามารถแก้โปรแกรมย่อยในตัวอย่างให้มีการรักษาไว้ในรีจิสเตอร์ต่าง ๆ ได้ดังโปรแกรมที่ 10.2

```
;procedure PrintHexDigit
;input  dl : digit
;affect ah,dl
;
printhexdigitproc    near
                    push    ax
                    push    dx
```

```

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

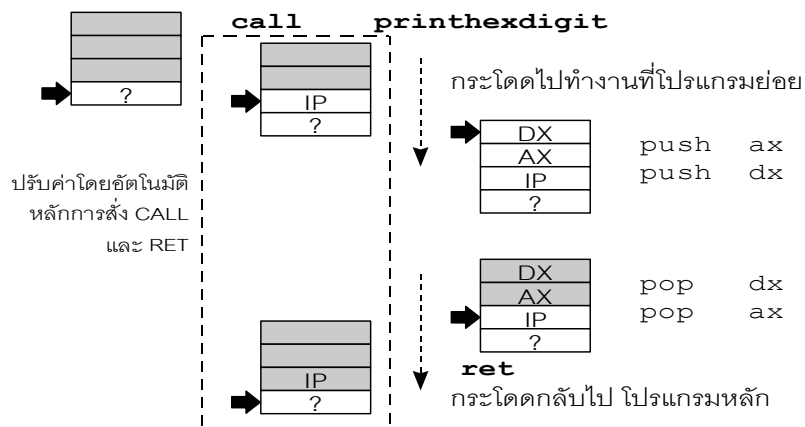
printit:
int    21h
pop   dx
pop   ax
ret

printhexdigitendp

```

โปรแกรมที่ 10.2 ตัวอย่างโปรแกรมย่อยที่รักษาค่าของรีจิสเตอร์

สังเกตลำดับในการ push และ pop ของโปรแกรมย่อย ลำดับในการ push จะตรงกันข้ามกับลำดับในการ pop เนื่องจากการทำงานของแอสตักเป็นแบบข้อมูลทีใส่ทีหลังจะถูกดึงออกก่อน การเปลี่ยนแปลงของแอสตักในการทำงานดังกล่าว แสดงได้ดังภาพที่ 10.2



ภาพที่ 10.2 แสดงการเปลี่ยนแปลงของแอสตักในการเรียกใช้โปรแกรมย่อย

เมื่อมีการเรียกใช้โปรแกรมย่อยโดยคำสั่ง CALL ค่าของรีจิสเตอร์ IP ซึ่งเก็บตำแหน่งของคำสั่งถัดไปจะถูก PUSH ลงในแอสตักโดยอัตโนมัติ จากนั้นโปรแกรมจะกระโดดไปทำงานที่โปรแกรมย่อย ในโปรแกรมย่อยมีการเก็บค่าของรีจิสเตอร์ AX และรีจิสเตอร์ DX ลงในแอสตัก หลังการทำงานโปรแกรมย่อยได้คืนค่าของรีจิสเตอร์ทั้งสองโดยการดึงค่าจากแอสตัก และเรียกใช้คำสั่ง RET เพื่อกระโดดกลับไปทำงานยังโปรแกรมหลักต่อ สังเกตว่าถ้าเก็บและดึงค่าออกจากแอสตักได้ไม่ถูกต้องการกระโดดกลับไปทำงานยังโปรแกรมหลักอาจมีการผิดพลาดได้

10.4 ตัวอย่างการใช้งานโปรแกรมย่อย

โปรแกรมตัวอย่างต่อไปนี้จะสร้างโปรแกรมย่อยขึ้นมาสองโปรแกรมย่อย 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

การแยกโปรแกรมเป็นโปรแกรมย่อยนี้ ทำให้สามารถนำส่วนของโปรแกรมนั้นมาใช้ใหม่ได้สะดวกและการตรวจสอบและแก้ไขโปรแกรมนั้นยังสามารถกระทำได้ง่ายขึ้นด้วย ส่วนการเรียกโปรแกรมย่อยมีความแตกต่างกับการกระโดดทั่วไประยะเนื่องจากภายหลังที่โปรแกรมย่อยทำงานเสร็จ หน่วยประมวลผลจะต้องสามารถกระโดดกลับมาทำงานในโปรแกรมหลักต่อไปได้ ดังนั้นการเรียกใช้โปรแกรมย่อยนั้นจะต้องมีการเก็บตำแหน่งของคำสั่งที่ทำงานอยู่เดิมด้วย และเมื่อจบโปรแกรมย่อยโปรแกรมจะต้องกระโดดกลับมาทำงานที่เดิม โดยใช้ข้อมูลที่เก็บไว้ คำสั่งของ 8086 ที่รองรับการใช้งานโปรแกรมย่อยคือคำสั่ง CALL และคำสั่ง RET เมื่อผู้ใช้เรียกคำสั่ง CALL พร้อมทั้งระบุตำแหน่งของโปรแกรมย่อย หน่วยประมวลผลจะเก็บตำแหน่งของคำสั่งถัดไปที่จะกลับมาทำงานลงในแอสตัก และจะกระโดดไปทำงานที่โปรแกรมย่อย เมื่อโปรแกรมย่อยทำงานเสร็จ โปรแกรมย่อยจะเรียกใช้คำสั่ง RET เพื่อกระโดดกลับมาทำงานในโปรแกรมหลักต่อไป เมื่อหน่วยประมวลผลประมวลผลคำสั่ง RET หน่วยประมวลผลจะดึงค่าตำแหน่งที่โปรแกรมจะกระโดดกลับไปทำงานจากแอสตัก และกระโดดกลับไปทำงานต่อยังโปรแกรมหลักต่อไป

คำถามทบทวน

1. จงอธิบายการทำงานของคำสั่ง CALL และคำสั่ง RET
2. การประกาศโปรแกรมย่อยในภาษาแอสแซมบลีโดยทั่วไปมักจะใช้คู่ของคำสั่งเทียมอะไร
3. คำสั่งเก็บข้อมูลและดึงข้อมูลจากแอสตักคำสั่ง PUSH และคำสั่ง POP มีขั้นตอนการทำงานอย่างไร
4. การทำงานของแอสตักจะมีรีจิสเตอร์ที่เกี่ยวข้องอะไรบ้างและแต่ละรีจิสเตอร์มีหน้าที่และทำงานอย่างไร
5. จงอธิบายเหตุการณ์ที่เกิดขึ้นหลังจากมีการเรียกใช้โปรแกรมย่อยโดยคำสั่ง CALL

เอกสารอ้างอิง

ราชบัณฑิตยสถาน. (2544). *ศัพท์บัญญัติ ราชบัณฑิตยสถาน*. ค้นเมื่อ 23 พฤษภาคม 2557,
จาก :<http://rirs3.royin.go.th/coinages/>

โปรแกรมย่อย. (2557). *วิกิพีเดีย สารานุกรมเสรี*. ค้นเมื่อ 23 พฤษภาคม 2557, จาก :<http://th.wikipedia.org/wiki/>

ชูชัย ธนสารตั้งเจริญ, กำธร พาณิชปฐมพงษ์. *ภาษาเอสแซมบลี 80286/80386(PC)*. กรุงเทพฯ :สำนักพิมพ์ซีเอ็ดยูเคชั่น บมจ., 2536.

ธีรวัฒน์ ประกอบผล. *ระบบคอมพิวเตอร์และภาษาเอสแซมบลี*. กรุงเทพฯ :สำนักพิมพ์ส่งเสริมเทคโนโลยี (ไทย-ญี่ปุ่น), 2537.

แผนการสอนประจำสัปดาห์ที่ 11

หัวข้อเนื้อหา การกระทำระดับบิต (The Bit Level)

รายละเอียด

การจัดการกับข้อมูลที่ได้ศึกษาในบทก่อน ๆ นั้นมีหน่วยย่อยในการจัดการเป็นไบต์ โดยจะไม่สามารถจัดการข้อมูลที่มีขนาดย่อยกว่านั้นได้ แต่ในการทำงานจริงในบางครั้งรูปแบบของข้อมูลที่น่าไปใช้ จำเป็นต้องจัดการให้อยู่ในรูปของบิต ดังนั้นการใช้คำสั่งเกี่ยวกับการจัดการระดับบิตในการประมวลผลข้อมูลกลุ่ม ในบางกรณีคำสั่งกระทำระดับบิตสามารถช่วยให้การคำนวณต่าง ๆ ทำได้ง่ายและมีประสิทธิภาพมากขึ้นตามไปด้วย

จำนวนชั่วโมงที่สอน 3 ชั่วโมง/สัปดาห์

กิจกรรมการเรียนการสอน

1. บรรยาย
2. สืบเสาะหาความรู้
3. ค้นคว้าเพิ่มเติม
4. ตอบคำถาม

สื่อการสอน

1. สื่ออิเล็กทรอนิกส์
2. เพาเวอร์พอยต์ 프리เซนต์เซชัน
3. บทเรียนออนไลน์
4. เอกสารอ้างอิงประกอบการค้นคว้า

แผนการประเมินผลการเรียนรู้

1. ผลการเรียนรู้

- 1.1 สังเกตจากงานที่กำหนดให้ไปทำมาส่ง
- 1.2 สังเกตจากการตอบคำถาม
- 1.3 สังเกตจากการนำความรู้ไปใช้

2. วิธีการประเมินผลการเรียนรู้

- 2.1 ตรวจผลงานภาคปฏิบัติ
- 2.2 ตรวจรายงาน
- 2.3 ตรวจแบบฝึกหัด

3. สัดส่วนของการประเมิน

- 3.1 ใบงานที่นักศึกษาทำมาส่ง
- 3.2 คะแนนเก็บในชั้นเรียน
- 3.3 การเข้าชั้นเรียน

เนื้อหาที่สอน

ในสัปดาห์ที่ 11 การจัดการเรียนการสอน จะเกี่ยวข้องกับคำสั่งทางตรรกศาสตร์ การประยุกต์ใช้งานคำสั่งทางตรรกศาสตร์ การใช้คำสั่งเลื่อนบิต ตัวอย่างการใช้งานคำสั่งเลื่อนบิต การประยุกต์ใช้งานคำสั่งเลื่อนบิต คำสั่งหมุนบิต คำสั่งหมุนบิตที่ผ่านแฟล็กทด ตัวอย่างการใช้งานคำสั่งเกี่ยวกับการประมวลผลระดับบิต ซึ่งปัญหาของการจัดการกับข้อมูลจะกระทำในระดับไบต์ โดยจะไม่สามารถจัดการข้อมูลที่มีขนาดย่อยกว่านั้นได้ แต่ในการทำงานจริงในบางครั้งรูปแบบของข้อมูลที่น่าไปใช้ จำเป็นต้องจัดการให้อยู่ในรูปของบิต ดังนั้นการใช้คำสั่งเกี่ยวกับการจัดการระดับบิตในการประมวลผลข้อมูลกลุ่ม ในบางกรณีคำสั่งกระทำระดับบิตสามารถช่วยให้การคำนวณต่าง ๆ ทำได้ง่ายและมีประสิทธิภาพมากขึ้นตามไปด้วย

11.1 คำสั่งทางตรรกศาสตร์

คำสั่งในกลุ่มนี้เป็นคำสั่งประมวลผลข้อมูลระดับบิต โดยจะนำค่าในแต่ละบิตของข้อมูลมาประมวลผลทางตรรกศาสตร์ คำสั่งในกลุ่มนี้ได้แก่ คำสั่ง AND คำสั่ง OR คำสั่ง XOR และคำสั่ง NOT ซึ่งรูปแบบการใช้งานของกลุ่มคำสั่งดังกล่าวจะมีลักษณะเหมือนกัน คือจะมีการ


```

mov    dl,25h
or     dl,0Fh      ; dl = 2Fh
mov    bl,55h
or     bl,80h      ; bl = 0D5h

```

คำสั่ง XOR

การทำงานของคำสั่ง XOR จะคล้ายกับคำสั่ง OR แต่ในกรณีที่ข้อมูลมีบิตที่เป็นหนึ่งทั้งคู่ ผลลัพธ์ที่ได้จะมีค่าเป็นศูนย์ (ตารางที่ 11.1) ลักษณะของการ XOR จะคล้ายกับการพิจารณาเหตุการณ์ที่เป็นไปได้ทั้งสองเหตุการณ์ แต่ไม่สามารถเป็นจริงพร้อมกันได้

ตัวอย่างคำสั่ง

```

mov    ax,1234h
xor    ax,2345h    ; ax = (0001 0010 0011 0100 and (0010 0011 0100 0101)
                  ; ax = (0011 0001 0111 0001) = 3171h

mov    dl,25h
xor    dl,0Fh      ; dl = 2Ah

mov    bl,15h
xor    bl,35h      ; bl = 20h

```

คำสั่ง NOT

คำสั่ง NOT จะสลับบิตของไอบีเปอร์แรนด์จาก 0 เป็น 1 และ 1 เป็น 0 แสดงได้ (ตารางที่ 11.1)

ตัวอย่างคำสั่ง

```

mov    ax,1234h
not    ax          ; ax = not(0001 0010 0011 0100)
                  ; ax = (1110 1101 1100 1011) = 0EDCBh

```

คำสั่ง TEST

คำสั่ง TEST จะทำงานเหมือนคำสั่ง AND ทุกประการ แต่ผลลัพธ์จากการ AND จะไม่เขียนค่าลงในไอบีเปอร์แรนด์ตัวแรก ผลจากการใช้คำสั่งนี้จะปรากฏในแฟล็ก นิยมใช้คำสั่งนี้ในการทดสอบว่าข้อมูลในบิตที่ต้องการมีค่าเป็นหนึ่งหรือไม่ โดยเราจะพิจารณาผลลัพธ์จากแฟล็กทด

ตัวอย่างคำสั่ง

```

mov  al,7Ah
test al,80h      ; test for the 7th bit
jz   biton       ; jump if 7th bit of al = 1
mov  bl,12h
test bl,1        ; test for the 0th bit

```

การประยุกต์ใช้งานคำสั่งทางตรรกศาสตร์

การนำคำสั่งทางตรรกศาสตร์มาใช้ในการประมวลผลข้อมูลระดับบิตได้ จากตารางที่ 11.1 สามารถนำมาสร้างตารางที่ 11.2 ซึ่งแสดงผลของการใช้คำสั่งทางตรรกศาสตร์กับข้อมูลได้

ตารางที่ 11.2 แสดงผลของการใช้คำสั่งทางตรรกศาสตร์กับข้อมูล

B	A and B	A or B	A xor B
0	0	A	A
1	A	1	~A

จากตารางเราจะพบว่าถ้าเราต้องการให้บิตใดของข้อมูลมีค่าเป็นหนึ่งในบิตอื่นมีค่าคงเดิม เราสามารถใช้คำสั่ง AND ได้ และถ้าเราต้องการจะทำให้บิตใดของข้อมูลมีค่าเป็นศูนย์โดยไม่มีผลกระทบต่อบิตอื่น ๆ เราสามารถใช้คำสั่ง OR สำหรับคำสั่ง XOR เราจะใช้ในกรณีที่ต้องการกลับบิตของข้อมูลจากศูนย์เป็นหนึ่งใน

ตัวอย่างการประยุกต์ใช้งานคำสั่งทางตรรกศาสตร์

โปรแกรมตัวอย่างต่อไปนี้จะเปลี่ยนบิตที่ 1 และ 2 ของ AL ให้มีค่าเป็นศูนย์ (การนับบิตจะนับบิตที่มีนัยสำคัญต่ำสุดเป็นบิตที่ 0) และเปลี่ยนบิตที่ 4 และบิตที่ 6 ให้มีค่าเท่ากับ 1 พร้อมทั้งกลับบิตที่ 3 ให้มีค่าตรงกันข้าม การทำงานควรจะมี แสดงได้ดังภาพที่ 11.1

```

AL      XXXX XXXX
and     1111 1001
or      0101 0000
xor     0000 1000
result  X1X1 X00X

```

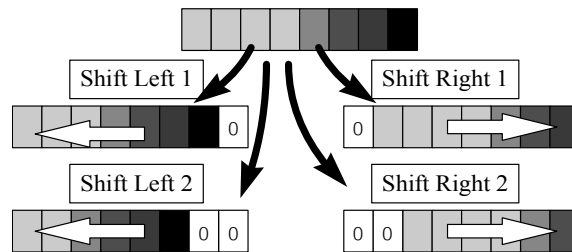
ภาพที่ 11.1 แสดงขั้นตอนการแปลงค่าของ AL

โปรแกรมจะมีลักษณะดังนี้

```
and    al,0F9h;clear bit 1&2
or     al,50h ;set bit 4&6
xor    al,08h ;switch bit 3
```

11.2 คำสั่งเลื่อนบิต

การประมวลผลอีกรูปแบบที่เราสามารถกระทำกับข้อมูลในระดับชั้นของบิตได้แก่ การเลื่อนบิต ลักษณะการเลื่อนบิต แสดงได้ดังภาพที่ 11.2 ในการเลื่อนบิตเราสามารถเลื่อนได้ทั้งทางซ้ายและทางขวา โดยคำสั่งสำหรับการเลื่อนบิตไปทางซ้ายได้แก่ คำสั่ง **SHL** (Shift Left) คำสั่งสำหรับการเลื่อนบิตไปทางขวาได้แก่ คำสั่ง **SHR** (Shift Right) เรานิยมใช้การเลื่อนบิตในการประมวลผลที่ต้องการประมวลผลข้อมูลทีละบิต และมีการประมวลผลเป็นแบบวงรอบ



ภาพที่ 11.2 แสดงลักษณะของการเลื่อนบิต

รูปแบบของคำสั่งเลื่อนบิตมีลักษณะดังนี้

SHR	regs,1	SHR	mem,1
SHR	regs,CL	SHR	mem,CL
SHR	regs,number	SHR	mem,number

โดยรูปแบบของคำสั่ง SHL จะมีลักษณะเหมือนคำสั่ง SHR รูปแบบที่สามจะใช้ได้กับหน่วยประมวลผล 80286 ขึ้นไปเท่านั้นโดยในการที่เราจะใช้รูปแบบของคำสั่งของ 80286 ในโปรแกรมเราจะต้องระบุ คำสั่งเทียม 286 ลงในโปรแกรมด้วย โดยใส่คำสั่งนี้ก่อนหน้าการใช้งานคำสั่งครั้งแรก (ศัพท์บัญญัติ ราชบัณฑิตยสถาน, 2544)

ตัวอย่างการใช้งานคำสั่งเลื่อนบิต

โปรแกรมตัวอย่างต่อไปนี้เป็นโปรแกรมนับจำนวนบิตที่มีค่าเป็นหนึ่งใน AX โดยจะให้ผลลัพธ์ใน BL

```

mov    bl,0

mov    cx,16      ; 16 bits

procbits:
test   ax,1      ; test for last bit
jz     doprocbit ; if last bit=0 jump
inc    bl

doprocbit:
shr    ax,1      ; next bit
loop  procbits

```

ความหมายทางคณิตศาสตร์ของการเลื่อนบิต

ตารางที่ 11.3 แสดงผลลัพธ์ของการเลื่อนบิตของข้อมูลต่าง ๆ

จากตารางจะสังเกตเห็นได้ว่านอกจากการเลื่อนบิตจะมีความหมายโดยตรงคือการเลื่อนบิตไปทางซ้ายหรือทางขวาแล้ว การเลื่อนบิตยังมีความหมายทางคณิตศาสตร์อีกด้วย

ข้อมูล (ค่า)	เลื่อนบิตไปทาง	จำนวน (บิต)	ผลลัพธ์ (ค่า)
0010 1110 (46)	ซ้าย	1	0101 1100 (92)
0010 1110 (46)	ซ้าย	2	1011 1000 (184)
0010 1110 (46)	ซ้าย	3	0111 0000 (112)
0110 0100 (100)	ขวา	1	0011 0010 (50)
0110 0100 (100)	ขวา	2	0001 1001 (25)
0110 0100 (100)	ขวา	3	0000 1100 (12)

สังเกตว่าการเลื่อนบิตไปทางซ้ายจะมีผลลัพธ์เหมือนกับการคูณด้วยกำลังของสอง ยกตัวอย่างเช่น การเลื่อนบิตไปทางซ้าย 1 บิตจะเหมือนกับการคูณด้วยสอง ข้อสังเกตคือจะต้องพิจารณากรณีที่ข้อมูลอยู่ในขอบเขตด้วย เช่น กรณีของการเลื่อน 0010 1110 ไปทางซ้าย 3 บิต (คูณด้วย 8) ผลลัพธ์ที่ได้จะมีความผิดพลาด การเลื่อนบิตไปทางขวาจะให้ผลลัพธ์ตรงกันข้ามกับการเลื่อนบิตไปทางขวา นั่นคือจะเสมือนการหารด้วยกำลังสอง (สังเกตว่าผลลัพธ์ที่ได้จะมีการตัดเศษเนื่องจากบิตที่เลื่อนจะหายไป เช่นในตัวอย่างที่เลื่อนบิตทางขวา 3 บิต)

คำสั่งเลื่อนบิตแบบคิดเครื่องหมาย : คำสั่ง SAL และคำสั่ง SAR

ถ้าใช้การเลื่อนบิตแทนการคูณหรือหารด้วยกำลังของสองกับตัวเลขแบบคิดเครื่องหมาย เราจะพบว่า การเลื่อนบิตไปทางซ้ายที่แสดงถึงการคูณนั้นยังสามารถใช้กับตัวเลขแบบคิดเครื่องหมายได้ เนื่องจากหลักที่เลื่อนเข้ามาแทนนั้นยังคงเป็นเลขศูนย์เหมือนในกรณีของเลขไม่คิดเครื่องหมาย แต่ในกรณีของการเลื่อนบิตไปทางขวาที่ใช้สำหรับการหารด้วยกำลังของสองนั้น บิตที่เลื่อนเข้ามาแทนอาจมีค่าเป็น 0 หรือ 1 ก็ได้ขึ้นกับเครื่องหมายของตัวเลขนั้น จึงมีคำสั่งเลื่อนบิตที่ใช้สำหรับเลขที่มองเป็นเลขคิดเครื่องหมาย คือ คำสั่ง SAL (Shift Arithmetic Left) และ คำสั่ง SAR (Shift Arithmetic Right) คำสั่ง SAL จะทำงานเหมือนคำสั่ง SHL ทุกประการ ตัวอย่างการใช้งานคำสั่งเป็นดังตารางที่ 11.4

ตารางที่ 11.4 แสดงตัวอย่างผลลัพธ์ของการเลื่อนบิตแบบคิดเครื่องหมาย

ข้อมูล (ค่า)	เลื่อนบิตไปทาง	จำนวน (บิต)	ผลลัพธ์ (ค่า)
0010 1110 (46)	ซ้าย	1	0101 1100 (92)
1110 1000 (-24)	ซ้าย	2	1101 0000 (-48)
0110 0100 (100)	ขวา	1	0011 0010 (50)
1010 1100 (-84)	ขวา	2	1110 1011 (-21)
1010 1100 (-84)	ขวา	3	1111 0101 (-11)

การประยุกต์ใช้งานคำสั่งเลื่อนบิต

การนำคำสั่งเลื่อนบิตไปใช้ในการคูณและหารข้อมูลได้ โดยการใช้คำสั่งเลื่อนบิตแทนการใช้คำสั่ง MUL ทำให้การคูณทำงานได้เร็วขึ้นและในบางกรณีทำให้เขียนโปรแกรมได้ง่ายขึ้นด้วย

ตัวอย่างคำสั่ง

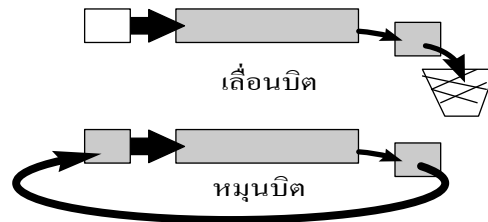
```

mov    bl,al
shl    al,1
add    bl,al        ;bl = al*3
mov    cl,2
shl    ax,cl
mov    bx,ax
shl    ax,1
add    bx,ax        ;bx=(ax*4)+(ax*8) = ax*12

```

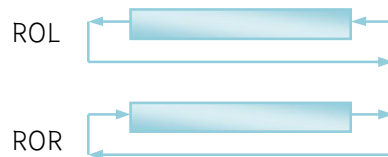
11.3 คำสั่งหมุนบิต

คำสั่งหมุนบิตมีความแตกต่างกับคำสั่งเลื่อนบิตในจุดที่ว่า บิตที่เลื่อนไปแล้วไม่ได้ถูกทิ้งหายไป แต่จะถูกนำมาใส่แทนบิตที่เลื่อนไป แสดงได้ดังภาพที่ 11.3



ภาพที่ 11.3 แสดงลักษณะของการเลื่อนบิต และ การหมุนบิต

เช่นเดียวกับคำสั่งเลื่อนบิต คำสั่งหมุนบิตมีลักษณะการหมุนสองแบบคือ หมุนไปทางซ้าย (คำสั่ง ROL : Rotate Left) และ หมุนไปทางขวา (คำสั่ง ROR : Rotate Right) รูปแบบของคำสั่งทั้งสองจะมีลักษณะเหมือนคำสั่งเลื่อนบิต การทำงานของคำสั่งทั้งสอง แสดงได้ดังภาพที่ 11.4



ภาพที่ 11.4 แสดงลักษณะการทำงานของคำสั่งหมุนบิต

นิยมใช้คำสั่งหมุนบิตแทนคำสั่งเลื่อนบิตในกรณีที่ต้องการให้ค่าของข้อมูลกลับเหมือนเดิมหลังประมวลผลครบรอบ

ตัวอย่างคำสั่ง

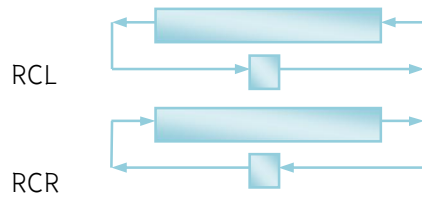
```

mov  al,4Ah
mov  cl,4
ror  al,cl      ;al = 0A4h
mov  bx,92EAh  ;bx=1001 0010 1110 1010
rol  bx,1      ;bx=0010 0101 1101 0101=35D5h
mov  cx,8
mov  dx,0
loophere:
      xor  dx,ax
      rol  ax,1
      loop loophere

```

คำสั่งหมุนบิตที่ผ่านแฟล็กทด

คำสั่งหมุนบิตอีกกลุ่มหนึ่งจะเป็นการหมุนโดยนำบิตไปผ่านแฟล็กทด ลักษณะการทำงาน แสดงได้ดังภาพที่ 11.5 ซึ่งจะเห็นว่าบิตที่เข้ามาแทนบิตที่หมุนไปจะนำมาจากแฟล็กทด และบิตที่ถูกหมุนออกไปจะเข้าไปแทนค่าในแฟล็กทด โดยคำสั่งหมุนบิตผ่านแฟล็กทดคือคำสั่ง RCL (Rotate Carry Left) และคำสั่ง RCR (Rotate Carry Right)



ภาพที่ 11.5 แสดงลักษณะการทำงานของคำสั่งหมุนบิตที่ผ่านแฟล็กทด

สังเกตว่าบิตที่ล้นออกมาจะถูกนำไปพักที่แฟล็กทด ก่อนที่จะนำมาแทนที่ในข้อมูล นิยมใช้คำสั่งหมุนบิตผ่านแฟล็กทดในการเลื่อนบิตข้อมูลที่เกี่ยวข้องอยู่ในหลายรีจิสเตอร์ ในการใช้งานคำสั่งนี้เราจะต้องกำหนดค่าให้กับแฟล็กทดเสียก่อน โดยใช้คำสั่ง STC และคำสั่ง CLC

ตัวอย่างการใช้งานคำสั่งหมุนบิตที่ผ่านแฟล็กทด

แสดงการเขียนคำสั่งแสดงการเลื่อนบิตของข้อมูลขนาด 32 บิตที่อยู่ในรีจิสเตอร์ DX,AX ไปทางซ้าย 1 บิต

```

clc
rcl ax,1
rcl dx,1
    
```

<p>ตัวอย่างคำสั่ง การคูณข้อมูลขนาด 48 บิตที่เก็บในรีจิสเตอร์ BX,DX และ AX ต่อเนื่องกัน ด้วย 4</p> <pre> clc rcl ax,1 rcl dx,1 rcl bx,1 clc rcl ax,1 rcl dx,1 rcl bx,1 </pre>	<p>ตัวอย่างคำสั่ง การหารข้อมูลใน DX,AX ด้วย 2</p> <pre> clc rcr dx,1 rcr ax,1 </pre>
---	---

ตัวอย่างการใช้งานคำสั่งเกี่ยวกับการประมวลผลระดับบิต

โปรแกรมตัวอย่างต่อไปนี้เป็นโปรแกรมที่แสดงค่ารหัสแอสกีของปุ่มที่รับจากผู้ใช้เป็นเลขฐานสอง โปรแกรมย่อยที่แสดงข้อมูลเป็นรหัสเลขฐานสองใช้การเลื่อนบิตในการประมวลผล

```
.model small
.dosseg
.code
; Display Binary (input : al)
dispbin proc near
    push ax
    push cx
    push dx
    mov cx,8
printloop:
    test al,80h      ;test for 1
    jz printzero
    mov dl,'1'
    jmp printit
printzero:
    mov dl,'0'
printit: mov ah,2
    int 21h
    shl al,1
    loop printloop
    pop dx
    pop cx
    pop ax
    ret
dispbin endp
start:
    mov ah,1
    int 21h
    call dispbin
    mov ax,4C00h
    int 21h
end start
```

สรุป

คำสั่งทางตรรกศาสตร์เป็นคำสั่งประมวลผลข้อมูลระดับบิต โดยจะนำค่าในแต่ละบิตของข้อมูลมาประมวลผลทางตรรกศาสตร์ คำสั่งในกลุ่มนี้ได้แก่ คำสั่ง AND คำสั่ง OR คำสั่ง XOR และคำสั่ง NOT รูปแบบการใช้งานของคำสั่ง AND คำสั่ง OR และคำสั่ง XOR จะมีลักษณะเหมือนกัน คือจะรับโอเปอร์แรนด์สองตัว และจะนำข้อมูลในโอเปอร์แรนด์ตัวแรกมากระทำกับข้อมูลตัวที่สอง และจะเก็บผลลัพธ์ของการกระทำนั้นในโอเปอร์แรนด์ตัวแรก ส่วนในกรณีของคำสั่ง NOT จะรับโอเปอร์แรนด์ตัวเดียว และจะทำการกลับค่าในบิตแล้วเก็บผลลัพธ์ลงในโอเปอร์แรนด์ตัวนั้นเลย ส่วนการประมวลผลอีกรูปแบบที่เราสามารถกระทำกับข้อมูลในระดับชั้นของบิตได้แก่การเลื่อนบิต ในการเลื่อนบิตเราสามารถเลื่อนได้ทั้งทางซ้ายและทางขวา โดยคำสั่งสำหรับการเลื่อนบิตไปทางซ้ายได้แก่ คำสั่ง SHL (Shift Left) คำสั่งสำหรับการเลื่อนบิตไปทางขวาได้แก่ คำสั่ง SHR (Shift Right) เรานิยมใช้การเลื่อนบิตในการประมวลผลที่ต้องการประมวลผลข้อมูลที่ละบิต และมีการประมวลผลเป็นแบบวงรอบ

คำถามทบทวน

1. จงแสดงผลลัพธ์ของคำสั่งจากคำสั่งข้างล่างที่ให้มาเมื่อเปิดของข้อมูลตัวตั้งทั้งสองตัวมีค่าเป็น 1

1.1 mov ax,2345h
 and ax,3456h ; ax = ? and ?
 ; ax = ?

1.2 mov ax,2345h
 or ax,3456h ; ax = ? and ?
 ; ax = ?

1.3 mov ax,2345h
 xor ax,3456h ; ax = ? and ?
 ; ax = ?

2. จงแสดงผลลัพธ์ของการเลื่อนบิตของข้อมูลต่าง ๆ จากตารางที่ให้มา

ข้อมูล (ค่า)	เลื่อนบิตไปทาง	จำนวน(บิต)	ผลลัพธ์ (ค่า)
0011 1110 (63)	ซ้าย	1	?
0011 1110 (63)	ซ้าย	2	?
0011 1110 (63)	ซ้าย	3	?
0111 0100 (116)	ขวา	1	?
0111 0100 (116)	ขวา	2	?
0111 0100 (116)	ขวา	3	?

3. จงแสดงผลลัพธ์ของการเลื่อนบิตแบบคิดเครื่องหมายของข้อมูลต่าง ๆ จากตารางที่ให้มา

ข้อมูล (ค่า)	เลื่อนบิตไปทาง	จำนวน(บิต)	ผลลัพธ์ (ค่า)
1100 0010 (-63)	ซ้าย	1	?
1101 1000 (-40)	ซ้าย	2	?
1100 0010 (-63)	ขวา	1	?
1000 1100 (-116)	ขวา	2	?
1000 1100 (-116)	ขวา	3	?

4. จงอธิบายพร้อมยกตัวอย่างคำสั่งหมุนบิตผ่านแฟล็กทดต่อไปนี้

4.1 คำสั่ง RCL (Rotate Carry Left)

4.2 คำสั่ง RCR (Rotate Carry Right)

เอกสารอ้างอิง

ราชบัณฑิตยสถาน. (2544). *ศัพท์บัญญัติ ราชบัณฑิตยสถาน*. ค้นเมื่อ 15 มิถุนายน 2557, จาก
: <http://rirs3.royin.go.th/coinages/>

การเลื่อนปิดของข้อมูล (2557). *วิกิพีเดีย สารานุกรมเสรี*. ค้นเมื่อ มิถุนายน 2557, จาก:
<http://th.wikipedia.org/wiki/>

ชูชัย ธนสารตั้งเจริญ, กำธร พาณิชปฐมพงษ์. *ภาษาเอสแซมบลี 80286/80386(PC)*. กรุงเทพฯ
: สำนักพิมพ์ซีเอ็ดยูเคชั่น บมจ., 2536.

ธีรวัฒน์ ประกอบผล. *ระบบคอมพิวเตอร์และภาษาเอสแซมบลี*. กรุงเทพฯ : สำนักพิมพ์ส่งเสริม
เทคโนโลยี (ไทย-ญี่ปุ่น), 2537.