แผนการสอนประจำสัปดาห์ที่ 12

หัวข้อเนื้อหา การอ้างแอดเดรสและการขัดจังหวะ
(Addressing Mode and Interrupted)

รายละเอียด

8086 จะมองลักษณะของหน่วยความจำเป็นกลุ่ม ๆ เก็บในรูปแบบของเซกเมนต์ (64 Kbyte/ 1 Segment) ได้แก่ CS, DS, SS และ ES โดยแสดงแอดเดรสของหน่วยความจำที่จะติดต่อด้วยซึ่ง CS จะบรรจุค่าแสดงแอดเดรสเริ่มต้นของโปรแกรมส่วน DS จะเก็บค่าดาตาเซกเมนต์ขณะนั้น SS ก็จะเก็บค่าสแต็กเซกเมนต์ขณะนั้น และ ES จะกำหนดเซกเมนต์ของข้อมูลรวมที่เรียกว่า global data segment โดยการทำงานของคอมพิวเตอร์จะประกอบด้วยการทำงานร่วมกันของหน่วยประมวลผลกลางกับอุปกรณ์รอบข้าง ซึ่งจำเป็นต้องมีการส่งผ่านข้อมูลระหว่างหน่วยประมวลผลกลางและอุปกรณ์อื่น ๆ อยู่เสมอ ดังนั้นรูปแบบการเชื่อมต่อจึงต้องถูกออกแบบมาให้เหมาะสมกับการทำงานร่วมกันของอุปกรณ์ต่าง ๆ

จำนวนชั่วโมงที่สอน 3 ชั่วโมง/สัปดาห์

กิจกรรมการเรียนการสอน

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

สื่อการสอน

1. สื่ออิเล็กทรอนิกส์
2. เพาเวอร์พอยต์ พรีเซนเตชัน
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 12 การจัดการเรียนการสอน จะเกี่ยวข้องกับการอ้างแอดเดรสในเซกเมนต์ทั้งสี่ คือ CS, DS, SS และ ES การประยุกต์ใช้คำสั่งในการอ้างแอดเดรสแบบต่าง ๆ กระบวนการขัดจังหวะและการใช้คำสั่งติดต่อกับอุปกรณ์ใน

12.1 การอ้างแอดเดรสใน 8086

ใน 8086 จะมองลักษณะของหน่วยความจำ โดยแบ่งหน่วยความจำเป็นกลุ่มๆในรูปแบบของ เซกเมนต์ ในหนึ่งเซกเมนต์จะชี้ได้ถึง 64 กิโลไบต์ เซกเมนต์ทั้งสี่ได้แก่ CS, DS, SS และ ES จะแสดงแอดเดรสของหน่วยความจำที่จะติดต่อด้วย CS จะบรรจุค่าแสดงแอดเดรสเริ่มต้นของโปรแกรม DS จะเก็บค่าดาตาเซกเมนต์ขณะนั้น SS ก็จะเก็บค่าสแต็กเซกเมนต์ขณะนั้น และ ES จะกำหนดเซกเมนต์ของข้อมูลรวมที่เรียกว่า global data segment

โดยสิ่งที่สำคัญคือ เซกเมนต์จะแสดงตำแหน่งเหมือนกับเป็นพารากราฟ (paragraph) โดยจะเลื่อนไปทางซ้าย 4 บิต เพื่อที่จะกำหนดหรืออ้างแอดเดรสให้ครบ 20 เส้น โดยจุดเริ่มต้นของพารากราฟจะต้องมี 4 บิต หลังสุดเป็น 0 เช่น เป็น 00000H, 00010H,

00020H เป็นต้น เพื่อที่จะทำการติดต่อกับข้อมูลหนึ่งไบต์หรือหนึ่งเวิร์ดนั้น 8086 ได้เตรียมค่าออฟเซต เพื่อใช้อ้างตำแหน่งตั้งแต่จุดเริ่มต้นของเซกเมนต์แอดเดรส ตำแหน่งใดๆจะได้มาจากการบวกค่าเซกเมนต์รีจิสเตอร์กับค่าของออฟเซต 16 บิต เช่นถ้าเซกเมนต์มีค่า E89F และให้ออฟเซตมีค่า 0003H ทำการอ้างแอดเดรสไปที่ E89F3H (ศัพท์บัญญัติ ราชบัณฑิตยสถาน, 2544) ลักษณะในการอ้างแอดเดรส โดยระบุตำแหน่งด้วย เซกเมนต์และออฟเซต เช่น

```
mov     ax, [es:100h]
mov     bl, es:[bx]
```

การอ้างข้อมูลทั่วๆไปโดยไม่ระบุเซกเมนต์ ออฟเซตที่ระบุจะคิดเทียบกับ DS เช่น

```
mov     ax, B800h
mov     es, ax
```

12.2 แบบการอ้างแอดเดรส

อ้างแบบรีจิสเตอร์ (Register addressing) ข้อมูลอยู่ในรีจิสเตอร์ เช่น

```
mov     ax, bx
```

อ้างแบบค่าคงที่ (Immediate addressing) ข้อมูลอยู่ในคำสั่ง เช่น

```
mov     ax, 10
```

การอ้างตำแหน่งโดยตรง (Direct addressing) เป็นการอ้างแอดเดรสแบบที่ระบุค่าออฟเซตโดยตรง เช่น

```
mov     ax, [100h]
mov     [200h], cl
mov     cl, total
mov     sum, ax
```

การอ้างตำแหน่งทางอ้อมโดยใช้รีจิสเตอร์ (Register indirect addressing) เป็นการอ้างแอดเดรสโดยแอดเดรสของข้อมูลอยู่ในรีจิสเตอร์ เช่น

```
mov     bx, offset buffer
mov     al, [bx]
mov     di, offset total
add     [di] al
```

การอ้างตำแหน่งแบบดัชนีโดยตรง (Direct indexed addressing) เป็นการอ้างแอดเดรสโดยแอดเดรสของข้อมูลได้จากการนำค่ารีจิสเตอร์ดัชนี (SI หรือ DI) มาบวกกับเลขคิดเครื่องหมายขนาดแปดบิต หรือเลขไม่คิดเครื่องหมายขนาดสิบหกบิต

```
.data
balance         dw      10 dup(?)
credit      dw      10 dup(?)
debit       dw      10 dup(?)
            ...
            mov     cx, 10
            mov     si, 0
calloop:            mov     ax, balance[si]
            sub     ax, credit[si]
            add     ax, debit[si]
            mov     balance[si], ax
            inc     si
```

การอ้างตำแหน่งแบบสัมพัทธ์กับฐาน (Base relative addressing) เป็นการอ้างแอดเดรสโดยแอดเดรสของข้อมูลได้จากการนำค่าคงที่ไปบวกกับค่าในรีจิสเตอร์ BX หรือ BP เช่น

```
.data
rec     dw      10 dup(4 dup(?))
        ...
        mov     cx, 10
        mov     bx, offset rec
updateloop:
        mov     ax, [bx]        ; x
        add     ax, [bx+2]      ; +y
        add     ax, [bx+4]      ; +z
        mov     [bx+6], ax      ; c = x+y+z
        add     bx, 8
```

การอ้างตำแหน่งแบบดัชนีกับฐาน (Base indexed addressing) เป็นการอ้างแอดเดรสโดยแอดเดรสของข้อมูลได้จากการนำค่าของรีจิสเตอร์ฐาน (BX หรือ BP) รวมกับค่าของรีจิสเตอร์ดัชนี (SI หรือ DI)

```
mov     ax, [bx+si+2]
mov     [bx+di], al
inc     byte ptr [bx+si]
mov     dx, [bp+si+2]
```

ถ้าคิดสัมพัทธ์กับ BP ออฟเซตที่ได้จะคิดเทียบกับรีจิสเตอร์ SS (Stack Segment)

ตัวอย่าง โปรแกรมแสดงตัวอักษรแบบใหญ่

การแสดงตัวอักษรใหญ่ๆนั้น ต้องมีการเก็บตัวอักษรที่จะพิมพ์ไว้ในหน่วยความจำ โดยการควรเก็บเป็นตารางจะง่ายต่อความเข้าใจ เช่น ตารางขนาด 8*8

```
.data
fontbuf  db     0, 0, 0, 1, 0, 0, 0, 0
         db     0, 0, 1, 0, 1, 0, 0, 0
         db     0, 1, 0, 0, 0, 1, 0, 0
         db     1, 0, 0, 0, 0, 0, 1, 0
         db     1, 1, 1, 1, 1, 1, 1, 0
         db     1, 0, 0, 0, 0, 0, 1, 0
         db     1, 0, 0, 0, 0, 0, 1, 0
         db     0, 0, 0, 0, 0, 0, 0, 0
```

ผู้เขียนโปรแกรมสามารถเปลี่ยนจากการเก็บ 0 และ 1 เป็น ช่องว่าง และ # แทนตามลำดับ

โปรแกรมย่อยสำหรับแสดงตัวอักษร หาตำแหน่งเริ่มต้นของตัวอักษร

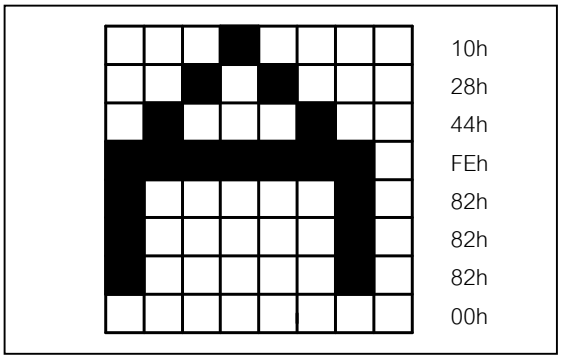
```
        mov     bx, offset fontbuf
        mov     dh, 0
        sub     dx, 'A'
        mov     cl, 6
        shl     dx, cl              ; dx = dx*16
```

โดยตัวอักษรตัวแรกเริ่มที่ 'A' เริ่มที่ offset fontbuf แต่ละตัวตำแหน่งเพิ่มขึ้นทีละ 64 ไบต์ซึ่งมาจากการเก็บตัวอักษรในรูปของตาราง 8*8 ที่แสดงข้างต้น

การพิมพ์ตัวอักษรออกมา

```
        mov     si, 0
        mov     dh, 0
printline:
        mov     cx, 8
printonechar:
        mov     dl, [bx+si]
        call    printchar
        inc     si
        loop    printonechar
        call    printnewline
        inc     dh
```

ผู้เขียนโปรแกรมสามารถลดขนาดการเก็บจากไบต์เป็นบิตได้โดย



ซึ่งจากวิธีนี้ทำให้ใช้ 8 ไบต์เท่านั้นในการเก็บสำหรับหนึ่งตัวอักษร

```
.data
fontbuf  db      10h, 28h, 44h, 0Feh
         db      82h, 82h, 82h, 00h
โดยที่ต้องสำหรับทุกตัวอักษร ( A – Z )
```

การแก้ไขส่วนของโปรแกรมแสดงผลใหม่ ดังนี้

```
         mov     si, 0
         mov     di, 0
printline:
         mov     dh, [bx+si]
         mov     cx, 8
printonechar:
         test    dh, 80h
         jz      printzero
         mov     dl, '#'
         jmp     printit
printzero:
         mov     dl, ' '
printit:
         call    printchar
         rol     dh, 1
         loop    printonechar
         call    printnewline
         inc     si
```

การทำงานของคอมพิวเตอร์จะประกอบด้วยการทำงานร่วมกันของหน่วยประมวลผลกลางกับอุปกรณ์รอบข้าง ซึ่งการทำงานร่วมกันนี้จำเป็นต้องมีการส่งผ่านข้อมูลระหว่างหน่วยประมวลผลกลางและอุปกรณ์อื่น ๆ อยู่เสมอ ดังนั้นรูปแบบการเชื่อมต่อระหว่างหน่วยประมวลผลกลางกับ

อุปกรณ์รอบข้างจึงต้องถูกออกแบบมาอย่างเหมาะสมกับการทำงานของอุปกรณ์ต่าง ๆ ซึ่งจะแบ่งรูปแบบการเชื่อมต่อระหว่างหน่วยประมวลผลกลางกับอุปกรณ์รอบข้างได้ดังนี้

1. Programmed I/O หน่วยประมวลผลมีคำสั่งโดยเฉพาะสำหรับการส่งรับข้อมูลกับอุปกรณ์

2. Memory-mapped I/O ซึ่งการติดต่อแบบนี้นิยมใช้ในอุปกรณ์ที่ไม่มีคำสั่งพิเศษในการติดต่อกับอุปกรณ์ I/Oการติดต่อจะเหมือนกับการเขียนและอ่านกับหน่วยความจำ

3. Interrupt I/O การที่ CPU ต้องตรวจสอบสถานะของ I/O ทำให้เสียเวลาจึงมีวิธีการติดต่อแบบนี้โดยให้ I/O รายงานแก่ CPU เองเมื่อมีการเปลี่ยนแปลงสถานะ เช่น มีข้อมูลเข้า ส่งข้อมูลเสร็จแล้ว มีลักษณะคือ I/O สร้างสัญญาณ Interrupt ให้กับ CPU เพื่อให้ CPU ประมวลผลเหตุการณ์ที่เกิดขึ้น

4. Direct Memory Access I/O โดยทั่วไป I/O ทำงานช้ามากเมื่อเทียบกับ CPU ดังนั้นการที่ CPU ต้องจัดการโอนย้ายข้อมูลเองจะเป็นการเสียเวลา จึงทำให้เกิดวิธีการติดต่อโดยตรงระหว่าง memory และ I/O ซึ่งในขณะนั้น CPU สามารถทำงานอื่นที่ไม่ต้องติดต่อกับ memory ได้ ขั้นตอนโดยทั่วไปมีขั้นตอนดังนี้

4.1 หน่วยประมวลผลสั่งงาน I/O และระบุตำแหน่งในหน่วยความจำที่จะให้ I/O เขียนผลลัพธ์หรืออ่านข้อมูล จากนั้น CPU จะไปทำงานอื่น

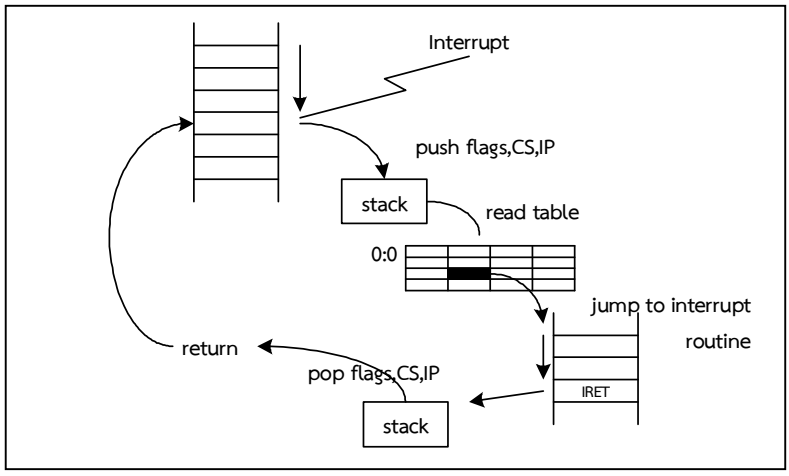
4.2 I/O จะเขียนผลลัพธ์ที่ได้หรืออ่านข้อมูลจากหน่วยความจำโดยตรงโดยไม่ผ่าน CPU

4.3 เมื่อ I/O ทำงานเสร็จจะแจ้ง CPU โดยการสร้างการขัดจังหวะ

12.3 กระบวนการขัดจังหวะใน 8086

กระบวนการขัดจังหวะหรือ interrupt เป็นกระบวนการหนึ่งที่อุปกรณ์รอบข้างจะเรียกร้องความสนใจจาก CPU เพื่อให้ CPU มาประมวลผลข้อมูลจากอุปกรณ์ที่ร้องขอนั้น ซึ่งกระบวนการนี้จะเริ่มจากอุปกรณ์รอบข้างส่งสัญญาณมาขอ interrupt จาก CPU เมื่อ CPU ได้รับสัญญาณ interrupt วงจรภายใน CPU จะพิจารณาว่าอุปกรณ์ใดที่ขอ interrupt โดยฮาร์ดแวร์ที่ควบคุมการ interrupt จะส่งค่าที่อยู่ระหว่าง 0 ถึง 255 ซึ่งเป็นหมายเลขที่ใช้แทนอุปกรณ์นั้นให้กับ CPU เมื่อ CPU ได้รับหมายเลขของอุปกรณ์ที่ส่งสัญญาณ Interrupt ก็ส่งผ่านการควบคุมไปยัง interrupt service routine โดย CPU ได้สำรองที่ 1024 ไบต์แรกในหน่วยความจำสำหรับ interrupt vector ข้อมูลที่อยู่ใน interrupt vector คือตัวชี้ที่บอกตำแหน่งที่อยู่ของ interrupt service routine ของอุปกรณ์แต่ละหมายเลขนั้น โดยที่ interrupt แต่ละหมายเลขจะเกี่ยวข้องกับ interrupt vector 4 ไบต์ เช่น interrupt vector เบอร์ 0 จะเกี่ยวข้องกับ vector ไบต์ 0 ถึง 3 โดยที่สองไบต์แรกจะชี้ไปยัง offset ของ interrupt service routine สองไบต์ต่อมาจะชี้ไปยัง segment ของ interrupt service routine

การ interrupt จะเก็บตำแหน่งที่อยู่ของคำสั่งถัดไป (return address) บน stack เนื่องจาก interrupt service routine อาจอยู่ในหน่วยความจำส่วนใดก็ได้ ดังนั้น CPU จึงต้องจัดการกับ interrupt คือจะต้องเก็บทั้ง offset และ segment ของโปรแกรมบน stack นอกจากนั้น ก็ยังเก็บค่าของ flag register ไว้บน stack ซึ่งใน flag register ก็มี interrupt enable flag (IF)บิตอยู่ด้วย การที่ CPU รับรู้ interrupt จากภายนอกได้หรือไม่ ขึ้นอยู่กับค่านี้ ถ้า IF มีค่าเป็น 0 CPU จะไม่ยอมรับสัญญาณจาก maskable interrupt ขณะที่กำลัง execute interrupt service routine อยู่เมื่อ CPU เก็บค่าของ flag register และ return address คือค่าใน register CS และ IP บน stack แล้ว ก็ใช้เบอร์ interrupt อ่านค่าตำแหน่งที่อยู่ของ interrupt service routine จาก interrupt vector แล้วย้ายการทำงานไปยัง interrupt service routine เมื่อทำมาถึงคำสั่ง IRET ซึ่งเป็นคำสั่ง return from interrupt ก็จะ pop ค่า 3 ค่าออกจาก stack และใส่ไว้ใน register IP, CS และ flag register ตามลำดับ แสดงได้ดังภาพ 12.1 (ศัพท์บัญญัติ ราชบัณฑิตยสถาน, 2544)



ภาพที่ 12.1 แสดงการเก็บตำแหน่งที่อยู่ของคำสั่งถัดไป (return address) บน stack

ที่มาของการขัดจังหวะ

การขัดจังหวะของ CPU มีที่มาจาก 2 ลักษณะได้แก่

1. การขัดจังหวะที่มาจากระบบฮาร์ดแวร์ อุปกรณ์ต่างๆที่ต้องการสร้างการขัดจังหวะจะส่งสัญญาณมาที่อุปกรณ์ควบคุมการขัดจังหวะ (Interrupt Controller) โดยสัญญาณที่เข้ามาที่อุปกรณ์ควบคุมนี้มีทั้งสิ้น 16 สัญญาณ (IRQ0 – IRQ16) จากนั้นอุปกรณ์ควบคุมการขัดจังหวะส่งสัญญาณมาที่ CPU พร้อมทั้งส่งหมายเลขของการขัดจังหวะมาพร้อมกันด้วย โดยถ้ามีการขัดจังหวะหลายอันที่ร้องขอพร้อมกันอุปกรณ์นี้จะจัดลำดับความสำคัญของอุปกรณ์และส่ง

สัญญาณที่มีความสำคัญสูงสุดก่อน อุปกรณ์ที่สร้างการขัดจังหวะกลุ่มนี้ได้แก่ นาฬิกาของระบบ แป้นพิมพ์ ฮาร์ดดิสก์ การ์ดเสียง ฯลฯ

2. การขัดจังหวะที่มาจากระบบซอฟต์แวร์ หรืออาจจะเรียกได้ว่าเป็นการขัดจังหวะที่เกิดขึ้นจากอุปกรณ์ภายในตัว CPU เองได้แก่ การเกิดความผิดพลาดในการทำงาน และการสั่งคำสั่ง INT โดยคำสั่ง INT มีรูปแบบดังนี้

```
INT       interrupt – type
โดยที่ interrupt – type คือเบอร์ของ interrupt ซึ่งมีค่าระหว่าง 0 ถึง 255
```

เมื่อคำสั่ง INT ถูก execute CPU จะดำเนินการดังต่อไปนี้คือ

1. Push ค่าของ flag register บน stack
2. Clear Trap Flag (TF) และ Interrupt Enable Flag (IF)
3. Push ค่าของ register CS บน stack
4. คำนวณตำแหน่งที่อยู่ใน interrupt vector ของ interrupt เบอร์นี้ โดยการคูณ interrupt–type ด้วย 4
5. Load ค่าใน word ที่สองใน interrupt vector สำหรับ interrupt เบอร์นี้ลงใน register CS ซึ่งก็คือค่าของ segment ของ interrupt service routine
6. Push ค่าใน register IP บน stack
7. Load ค่าใน word ที่หนึ่งใน interrupt vector เบอร์นี้ ใน register IP ซึ่งก็คือค่าของ offset ของ interrupt service routine
8. Execute interrupt service routine

ตัวอย่าง การรับตัวอักษรจากแป้นพิมพ์และแสดงผล

interrupt 21H เป็นการเรียกใช้งานฟังก์ชัน โดยใช้ register AH บรรจุหมายเลขฟังก์ชัน

```
;DEMONSTRATE FUNCTION 01H
;READ KEYBOARD AND DISPLAY
CODE            SEGMENT       PAGE
          ASSUME        CS:CODE,DS:CODE
;**************************
;* MACRO DEFINITION *
;**************************
```

```
READ_KBD_ECHO     MACRO
        MOV   AH, 01H              ;รอรับการกดตัวอักษรหนึ่งตัวจากแป้นพิมพ์
        INT   21H                  ;รหัสตัวอักษรที่กดถูกส่งไป register AL
        ENDM
DISP_CHAR     MACROCHARACTER
        MOV   DL, CHARACTER
        MOV   AH, 02H              ;แสดงผลตามค่าที่อยู่ใน register DL
        INT   21H
        ENDM
DISPLAY       MACROSTRING
        MOV   DX, OFFSET STRING
        MOV   AH, 09H ;แสดงสายตัวอักษร(string)บนจอภาพจนกว่าจะพบตัวอักษร
        INT   21H     ;'$' โดยregister DX จะบรรจุค่า offset (จาก segment DS)
        ENDM
TERMINATE     MACRO
        MOV   AH,4CH
        INT   21H
        ENDM
        ORG           0100H
START:  DISPLAY       SHOW              ;SHOW MESSAGE
        DISP_CHAR     10                ;SEND LINEFEED
FUNC_01H:             READ_KBD_ECHO     ;THIS FUNCTION
        CMP           AL,0DH            ;IS IT A CR-RETURN
        JNE           FUNC_01H          ;NO,LOOP BACK
        DISP_CHAR     10                ;
        TERMINATE
SHOW          DB      'TYPE ANY KEY !',13,10
        DB      'PRESS RETURN TO TERMINATE',13,10,'$'
CODE          ENDS
        END   START
```

12.4 คำสั่งติดต่อกับอุปกรณ์ใน 8086

หน่วยประมวลผลของ 8086 สามารถติดต่อกับอุปกรณ์ได้ทั้งแบบ Programmed I/O, Interrupt I/O, และ Direct Memory Access โดยคำสั่งเขียนอ่านมีรูปแบบดังนี้

```
IN      accumulator, DX
IN      accumulator, portnumber
OUT     DX, accumulator
OUT     portnumber, accumulator โดยที่
        accumulator คือ register AX หรือ AL ขึ้นกับว่าเป็นการติดต่อแบบ 8 บิต หรือ 16 บิต
        portnumber คือหมายเลขของอุปกรณ์ (หมายเลข I/O) เราสามารถระบุค่านี้โดยผ่าน
        ทาง register DX ได้ หมายเลขพอร์ตนี้มีค่าตั้งแต่ 0 ถึง 65535
```

ตัวอย่าง การติดต่อกับอุปกรณ์รอบข้าง : การติดต่อกับลำโพง

ภายในเครื่องคอมพิวเตอร์จะมีลำโพงเล็กๆโดยที่โปรแกรมสามารถควบคุมเสียงที่ออกจากลำโพงนี้ โดยควบคุมผ่านพอร์ต 61H ใช้ในการนำข้อมูลออกไปให้โปรแกรมและค่านี้จะคงอยู่อย่างนั้นจนกระทั่งโปรแกรมเปลี่ยนค่า แต่ละบิตของพอร์ต 61H มีความหมายดังนี้

```
บิต  0   timer 2 gates ( ควบคุมลำโพง )
     1   speaker direct control
     2   multiplex port 62H
     3   cassette motor control
     4   enable parity check on system board memory
     5   enable parity check on I/O board memory
     6   keyboard clock control
     7   keyboard clear/multiplex port 60H
```

การสั่งงานลำโพงอย่างง่ายๆสามารถทำได้โดยสั่งงานบิตที่ 1 โดยการกำหนดค่าลงในบิตนี้เปลี่ยนค่าไปมาระหว่าง 0 และ 1 ซึ่งจะทำให้ลำโพงเกิดการสั่นขึ้น การสั่งงานนั้นต้องสั่งงานโดยไม่ให้กระทบกับบิตอื่นๆ เช่น

```
in      AL, 61H
xor     AL, 02H
out     61H, AL
```

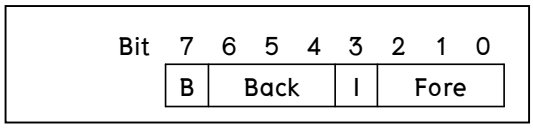
ในการสร้างเสียงต้องมีการหน่วงเวลาในการเปลี่ยนค่าบิตควบคุมลำโพง (บิตที่ 1) เพราะความถี่สูงเกินไปลำโพงจะไม่สามารถเคลื่อนที่ทันอีกทั้งให้ผู้ใช้สามารถได้ยินเสียงนั้นด้วย

ตัวอย่าง โปรแกรมย่อยสำหรับสร้างเสียงอย่างง่าย

```
gensoundproc    near
; CX    delay   [for freq]
; DX    duration
        push    ax
        push    dx
gensignal:
        in      al, 61h
        xor     al, 02h
        out     61h, al
        push    cx
delayloop:
        nop     ;no operation
        loop    delayloop
        pop     cx
        dec     dx
        or      dx, dx
        jnz     gensignal
        pop     dx
        pop     ax
        ret
gensoundendp
```

ตัวอย่าง การติดต่อกับอุปกรณ์รอบข้าง : ระบบการแสดงผลแบบตัวอักษร

การแสดงแบบตัวอักษรใน IBMPC นั้น ในแต่ละตัวอักษรที่แสดงจะมีค่าที่เกี่ยวข้อง 2 ค่าคือ รหัสแอสกีของตัวอักษรนั้น และแสดงค่า display attribute ของตัวอักษร ซึ่งเป็นค่าที่บอกสีของตัวอักษร และลักษณะการแสดงผล โดยบิตที่ 0–3 แสดงสีของตัวอักษร และบิตที่ 4–7 แสดงสีของพื้นหลัง



ผู้เขียนโปรแกรมสามารถเขียนโปรแกรมจัดการการแสดงผลที่หน้าจอได้สองวิธี

1. ใช้บริการของ Video BIOS [Interrupt 10H]

ตัวอย่าง แสดงรูปบนจอโดยใช้ BIOS interrupt เบอร์ 10H

```
; PROGRAM DRAW A DIAGONAL LINE OF SMILE FACES ON SCREEN
CODE_SEG        SEGMENT
        ASSUME  CS:CODE-SEG
DIAG    PROC    FAR
        PUSH    DS              ;SAVE RETURN ADDRESS
        MOV     AX, 0
        PUSH    AX
        PUSH    AX              ;SAVE REGISTERS AX
        PUSH    BX              ;SAVE REGISTERS BX
        PUSH    CX              ;SAVE REGISTERS CX
        PUSH    DX              ;SAVE REGISTERS DX
        STI                     ;ENABLE INTERRUPT
        MOV     AH, 15          ;SET BH TO ACTIVE PAGE
        INT     10H
        MOV     CX, 1
        MOV     DX, 0           ;START AT ROW=0 COLUMN=0
CRSR:   MOV     AH, 2           ;MOVE CURSOR TO NEXT POSITION
        INT     10H
```

```
        MOV   AL, 2              ;CHARACTER DISPLAY
        MOV   AH, 10             ;WRITE CHARACTER TO SCREEN
```

```
INT     10H
        INC   DH                 ;POINT TO NEXT ROW
        INC   DL                 ;POINT TO NEXT COLUMN
        CMP   DH, 25             ;BOTTOM OF SCREEN
        JNE   CRSR
        POP   DX
        POP   CX                 ;RESTORE REGISTERS
        POP   BX
        POP   AX
        RET
DIAG    ENDP
CODE_SEG        ENDS
END
```

ในหน่วยแสดงผลแบบ VGA หน่วยความจำตั้งแต่ตำแหน่งที่ 0B800h: 0000h จะเป็นหน่วยความจำที่เก็บข้อมูลของหน้าจอ โดยการเก็บขอมูลจะมีลักษณะเก็บทีละตัวอักษรเรียงกันไปตามลำดับ ซึ่งการเก็บข้อมูลจะเก็บตัวอักษรละ 2 ไบต์ ไบต์แรก (ไบต์ต่ำ) จะเก็บค่ารหัสแอสกีไบต์ที่สองจะเก็บค่า display attribute การเก็บจะเก็บเรียงทีละบรรทัด บรรทัดละ 80 ตัวอักษร

ตัวอย่าง การคำนวณข้อมูลของตัวอักษรที่บรรทัดที่ 10 ตัวที่ 3 อยู่ที่ offset 10*(80*2)+3*2 = 1606

ตัวอย่างโปรแกรมแสดงตัวอักษรทั้งหมดบนจอภาพ

```
CODE_SEG            SEGMENT
START               PROC        FAR
        ASSUME      CS:CODE_SEG
;SAVE RETURN ADDRESS
        PUSH        DS
        MOV         AX, 0
        PUSH        AX
```

```
;CLEAR THE DISPLAY
        MOV     AX, 0B0000H
        MOV     EX, AX          ;ES POINTS TO BASE OF ADAPTE
        MOV     DI, 0
        MOV     AL, ' '         ;BLANK CHARACTER
        MOV     AH, 07H         ;NORMAL VIDEO
```

```
                MOV     CX, 2000
                REP     STOSW   ;BLANK OUT VIDEO DISPLAY
;FILL THE DISPLAY WITH 256 CHARACTERS WITH A BLANK COLUMN
;AND LINE BETWEEN ALL CHARACTERS
                MOV     AL, 0           ;CHARACTER TO DISPLAY
                MOV     AH, 0           ;COLUMN NUMBER
                MOV     DI, 160         ;POINT TO FIRST COLUMN IN ROW 2
  AGAIN:        MOV     ES:[DI], AL     ;PUT CHARACTER IN MEMORY
                ADD     DI, 4           ;POINTS TO NEXT POSITION
                ADD     AH, 2
                CMP     AH, 80          ;FINISHED THIS ROW ?
                JB      SAME_LINE
;IF COLUMN NUMBER IS GREATER THAN 80 THEN SKIP A LINE
                ADD     DI, 160
                MOV     AH, 0
SAME_LINE:      CMP     AL, 255
                JE      FINISHED        ;256 CHARACTERS YET
                INC     AL              ;UPDATE TO NEXT CHARACTER
                JMP     AGAIN
;AFTER DISPLAYING ALL THE CHARACTERS WAIT FOR A KEY TO BE
;PRESSED THEN BLANK THE DISPLAY AND RETURN TO DOS
                MOV     AH, 0           ;GET KEYBOARD INPUT
                INT     16H
```

```
                MOV     DI, 0           ;INITIAL OFFSET ADDRESS
                MOV     AL, ' '         ;BLANK CHARACTER
                MOV     AH, 07H         ;NORMAL DISPLAY ATTRIBUTE
                MOV     CX, 2000
                REP     STOSW           ;BLANK ADAPTER MEMORY
                RET
START           ENDP
CODE_SEG        ENDS
END             START
```

ตัวอย่าง โปรแกรมย่อยเขียนตัวอักษรลงบนหน้าจอโดยตรง

```
WRITEONECHAR    PROC    NEAR
                ;( DH, DL ) : ( ROW, COL )
                ;AL: CHAR       AH: ATTRIBUTE
                PUSH    DS
                PUSH    AX
                PUSH    BX
                PUSH    CX
                PUSH    DX
                MOV     AX, 0B800H
                MOV     DS, AX
                MOV     CL, DH
                PUSH    AX
                MOV     AL, 160
                MUL     CL              ; AX=ROW*160
                MOV     BX, AX
  POP           AX
                MOV     DH, 0
                SHL     DX, 1
                ADD     BX, DX
                MOV     [BX], AX
```

สรุป

8086 จะมองลักษณะของหน่วยความจำ โดยแบ่งหน่วยความจำเป็นกลุ่มๆในรูปแบบของเซกเมนต์ ในหนึ่งเซกเมนต์จะชี้ได้ถึง 64 กิโลไบต์ เซกเมนต์ทั้งสี่ได้แก่ CS, DS, SS และ ES จะแสดงแอดเดรสของหน่วยความจำที่จะติดต่อด้วย CS จะบรรจุค่าแสดงแอดเดรสเริ่มต้นของโปรแกรม DS จะเก็บค่าดาตาเซกเมนต์ขณะนั้น SS ก็จะเก็บค่าสแต็กเซกเมนต์ขณะนั้น และ ES จะกำหนดเซกเมนต์ของข้อมูลรวมที่เรียกว่า global data segment

กระบวนการขัดจังหวะใน 8086 เป็นกระบวนการหนึ่งที่อุปกรณ์รอบข้างจะเรียกร้องความสนใจจาก CPU เพื่อให้ CPU มาประมวลผลข้อมูลจากอุปกรณ์ที่ร้องขอนั้น ซึ่งกระบวนการนี้จะเริ่มจากอุปกรณ์รอบข้างส่งสัญญาณมาขอ interrupt จาก CPU เมื่อ CPU ได้รับสัญญาณ interrupt วงจรภายใน CPU จะพิจารณาว่าอุปกรณ์ใดที่ขอ interrupt โดยฮาร์ดแวร์ที่ควบคุมการ interrupt จะส่งค่าที่อยู่ระหว่าง 0 ถึง 255 ซึ่งเป็นหมายเลขที่ใช้แทนอุปกรณ์นั้นให้กับ CPU เมื่อ CPU ได้รับหมายเลขของอุปกรณ์ที่ส่งสัญญาณ Interrupt ก็ส่งผ่านการควบคุมไปยัง interrupt service routine โดย CPU ได้สำรองที่ 1024 ไบต์แรกในหน่วยความจำสำหรับ interrupt vector ข้อมูลที่อยู่ใน interrupt vector คือตัวชี้ที่บอกตำแหน่งที่อยู่ของ interrupt service routine ของอุปกรณ์แต่ละหมายเลขนั้น โดยที่ interrupt แต่ละหมายเลขจะเกี่ยวข้องกับ interrupt vector 4 ไบต์ เช่น interrupt vector เบอร์ 0 จะเกี่ยวข้องกับ vector ไบต์ 0 ถึง 3 โดยที่สองไบต์แรกจะชี้ไปยัง offset ของ interrupt service routine สองไบต์ต่อมาจะชี้ไปยัง segment ของ interrupt service routine

คำถามทบทวน

1. ลักษณะการอ้างอิงแอดเดรสมีกี่แบบอะไรบ้าง
2. จงเขียนคำสั่งเพื่ออ้างอิงแอดเดรสต่อไปนี้
   2.1 อ้างแบบรีจิสเตอร์ (Register addressing)
   2.2 อ้างแบบค่าคงที่ (Immediate addressing)
   2.3 อ้างโดยตรง (Direct addressing)
   2.4 อ้างทางอ้อมโดยใช้รีจิสเตอร์ (Register indirect addressing)
   2.5 อ้างแบบดัชนีโดยตรง (Direct indexed addressing)
   2.6 อ้างแบบสัมพัทธ์กับฐาน (Base relative addressing)
   2.7 อ้างแบบดัชนีกับฐาน (Base indexed addressing)
3. จงเขียนโปรแกรมแสดงตัวอักษร B และมีการเก็บตัวอักษรที่จะพิมพ์ไว้ในหน่วยความจำโดยเก็บเป็นตารางขนาด 8*8
4. จงอธิบายรูปแบบการเชื่อมต่อระหว่างหน่วยประมวลผลกลางกับอุปกรณ์รอบข้างต่อไปนี้
   4.1 Programmed I/O
   4.2 Memory-mapped I/O
   4.3 Interrupt I/O
   4.4 Direct Memory Access I/O
5. การขัดจังหวะของ CPU มีกี่ลักษณะอะไรบ้าง
6. เมื่อคำสั่ง INT ถูก execute CPU จะดำเนินการโดยมีขั้นตอนการทำงานอย่างไร

7. จงอธิบายหน่วยประมวลผลของ 8086 สามารถติดต่อกับอุปกรณ์ในแบบต่างๆ ต่อไปนี้ได้อย่างไร
   7.1 Programmed I/O
   7.2 Interrupt I/O
   7.3 Direct Memory Access
8. จงเขียนโปรแกรมแสดงตัวอักษรที่มีคำว่า “Computer Science Suan Dusit” บนจอภาพ

เอกสารอ้างอิง

ราชบัณฑิตยสถาน. (2544). *ศัพท์บัญญัติ ราชบัณฑิตยสถาน.* ค้นเมื่อ 18 มิถุนายน 2557,จาก :http://rirs3.royin.go.th/coinages/

การอ้างอิงแอดเดรสของหน่วยความจำ (2557). *วิกิพีเดีย สารานุกรมเสรี.* ค้นเมื่อ 18 มิถุนายน 2557, จาก:http://th.wikipedia.org/wiki/

การขัดจังหวะ (2557). *วิกิพีเดีย สารานุกรมเสรี.* ค้นเมื่อ 18 มิถุนายน 2557, จาก: http://th.wikipedia.org/wiki/

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

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

แผนการสอนประจำสัปดาห์ที่ 13

หัวข้อเนื้อหา คำสั่งจัดการกับสายข้อมูล (Character Instruction)

รายละเอียด

คำสั่งจัดการกับสายข้อมูล เป็นคำสั่งที่ใช้จัดการกับข้อมูลเป็นที่เป็นชุดเรียงต่อกัน คำสั่งในกลุ่มนี้จะระบุตำแหน่งของข้อมูลด้วย Index register คือ SI (Source Index) และ DI (Destination Index) ประกอบกับ DS และ EX โดย ข้อมูลต้นทางจะอยู่ที่ตำแหน่ง DS:SI และข้อมูลปลายทางจะอยู่ที่ ES:DI การทำงานคำสั่งในกลุ่มนี้จะมีการปรับค่าของรีจิสเตอร์ที่เก็บตำแหน่งของข้อมูลให้โดยอัตโนมัติ เพื่อว่าเราจะสามารถใช้คำสั่งนั้นกระทำกับข้อมูลที่ตำแหน่งติดกันถัดไปได้โดยไม่ต้องปรับค่าของรีจิสเตอร์เอง ทิศทางในการปรับจะขึ้นกับค่าที่กำหนดในแฟล็กทิศทาง (DF : Direction flag)

จำนวนชั่วโมงที่สอน 3 ชั่วโมง/สัปดาห์

กิจกรรมการเรียนการสอน

1. บรรยาย
2. สืบเสาะหาความรู้

3. ค้นคว้าเพิ่มเติม

4. ตอบคำถาม

สื่อการสอน

1. สื่ออิเล็กทรอนิกส์

2. เพาเวอร์พอยต์ พรีเซนเตชัน

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

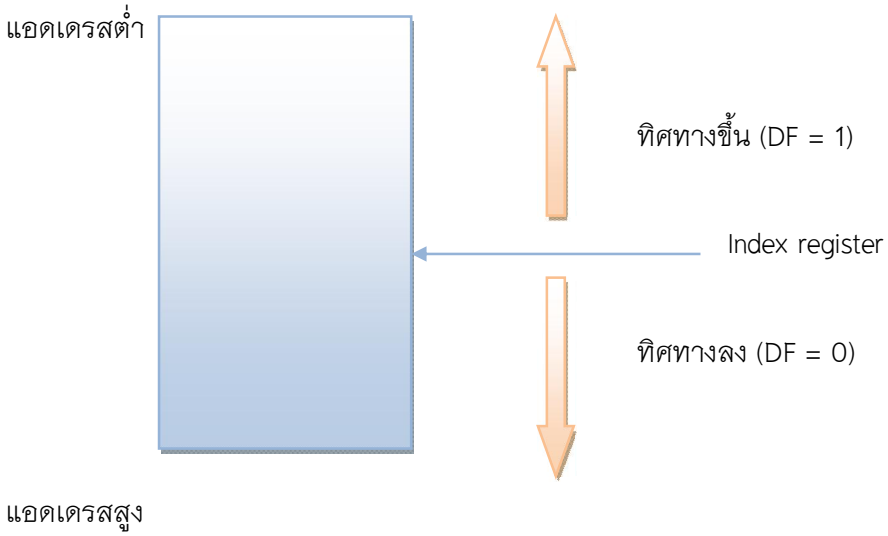
เนื้อหาที่สอน

ในสัปดาห์ที่ 13 การจัดการเรียนการสอน จะเกี่ยวข้องกับคำสั่งจัดการสายข้อมูลต่างๆ เช่น คำสั่ง MOVS, คำสั่ง REP, คำสั่ง STOS, คำสั่ง LODS, คำสั่ง REPZ, คำสั่ง CMPS, คำสั่ง REPNZ และคำสั่ง SCAS เป็นต้น

คำสั่งจัดการกับสายข้อมูล เป็นคำสั่งที่ใช้จัดการกับข้อมูลเป็นที่เป็นชุดเรียงต่อกัน

คำสั่งในกลุ่มนี้จะระบุตำแหน่งของข้อมูลด้วย Index register คือ SI (Source Index) และ DI (Destination Index) ประกอบกับ DS และ EX โดย ข้อมูลต้นทางจะอยู่ที่ตำแหน่ง DS:SI และข้อมูลปลายทางจะอยู่ที่ ES:DI การทำงานคำสั่งในกลุ่มนี้จะมีการปรับค่าของรีจิสเตอร์ที่เก็บตำแหน่งของข้อมูลให้โดยอัตโนมัติ เพื่อว่าเราจะสามารถใช้คำสั่งนั้นกระทำกับข้อมูลที่ตำแหน่งติดกันถัดไปได้โดยไม่ต้องปรับค่าของรีจิสเตอร์และทิศทางในการปรับค่าจะขึ้นอยู่กับค่าที่กำหนดในแฟล็กทิศทาง (DF : Direction flag) (ศัพท์บัญญัติ ราชบัณฑิตยสถาน, 2544)

ถ้า DF เป็น 0 แสดงว่ามีการเพิ่มตำแหน่งของการทำงาน โดยจะชี้ไปที่ข้อมูลตัวถัดไป คือ ค่าของรีจิสเตอร์ดัชนีจะเพิ่มขึ้น แต่ถ้า DF เป็น 1 แสดงว่าการทำงานจะเป็นการลดตำแหน่ง โดยจะชี้ไปที่ข้อมูลที่อยู่ก่อนหน้านี้ คือรีจิสเตอร์ดัชนีจะมีค่าลดลง



การกำหนดค่าของ DF สามารถทำได้ด้วยคำสั่ง STD (set direction flag) ซึ่งจะทำให้ค่าใน DF เป็น 1 และ CLD (clear direction flag) ซึ่งจะทำให้ค่าใน DF เป็น 0

คำสั่งต่างๆ ในกลุ่มนี้ ประกอบไปด้วย

MOVSx	Move string
LODSx	Load string
STOSx	Store string
SCASx	Scan string

CMPSx	Compare string

หมายเหตุ ตัวแปร x แทนค่าของ B (byte) หรือ W (word) หรือ D (double) ซึ่งหมายถึงขนาดของข้อมูลที่ถูกกระทำด้วยคำสั่งนั้นๆ

13.1 คำสั่ง MOVS

คำสั่ง MOVS เป็นคำสั่งที่ใช้สำหรับการคัดลอกข้อมูลจากแหล่งข้อมูลต้นทางไปยังแหล่งข้อมูลปลายทาง โดยมีรูปแบบดังนี้

```
MOVSB
MOVSW
MOVSD
```

คำสั่ง MOVS จะทำการคัดลอกข้อมูลจากตำแหน่ง DS:SI ไปยังตำแหน่ง ES:DI พร้อมทั้งปรับค่าในรีจิสเตอร์ DI และ SI คำสั่ง MOVSB ใช้ในการคัดลอกค่าขนาดไบต์ คำสั่ง MOVSW ใช้คัดลอกค่าขนาดเวิร์ด และคำสั่ง MOVSD ใช้คัดลอกค่าขนาดดับเบิลเวิร์ด

ตัวอย่าง โปรแกรมคัดลอกข้อมูลจำนวน 100 ตัวที่เริ่มต้นที่เลเบล d1 ไปยังเลเบล d2

```
.data
d1      dw      100 dup (N)
d2      dw      100 dup (?)

.code
  mov   ax,@data
  mov   ds,ax
  mov     es,ax
  mov   si,offset d1        ; source address
  mov   di,offset d2        ; destination address
```

```
 cld                          ; move in forward direction
 mov    cs,100                ; 100 byte to be moved
copy:   movsw
 loop copy
 ...
```

13.2 คำสั่ง REP

คำสั่งการกระทำกับสายข้อมูลนั้นมักจะต้องทำงานซ้ำเป็นวงรอบเพราะในการเรียกคำสั่งแต่ละครั้งนั้นจะเป็นการกระทำกับข้อมูลทีละ 1 ตัวเท่านั้น ดังเช่นในตัวอย่างข้างต้นเป็นการใช้คำสั่ง MOVSW ร่วมกับคำสั่ง LOOP แต่เราก็อาจจะนำคำสั่ง REP มาใช้แทนได้

คำสั่ง REP ใช้นำหน้าคำสั่งอื่นๆ เพื่อให้ทำคำสั่งนั้นซ้ำ โดยในการกระทำคำสั่งแต่ละครั้งนั้นจะมีการลดค่าของรีจิสเตอร์ CX ลงครั้งละ 1 จนกว่าค่าของรีจิสเตอร์ CX จะเท่ากับ 0 ซึ่งก็คล้ายกับการทำงานของคำสั่ง LOOP นั่นเอง

ตัวอย่างคำสั่ง

```
        mov     cx,100
copy:   movsw
        loop    copy
```

ผู้เขียนโปรแกรมสามารถแทนบรรทัดข้างต้นทั้ง 3 ได้ด้วย

```
mov     cx,500
rep     movsw
```

13.3 คำสั่ง STOS

คำสั่ง STOS เป็นคำสั่งที่ใช้สำหรับการคัดลอกข้อมูลขนาดไบท์จากรีจิสเตอร์ AL หรือ ขนาดเวิร์ดจากรีจิสเตอร์ AX หรือ ขนาดดับเบิลเวิร์ดจากรีจิสเตอร์ EAX ไปยังหน่วยความจำตำแหน่ง ES:DI พร้อมทั้งปรับค่ารีจิสเตอร์ DI โดยมีรูปแบบดังนี้

```
STOSB
STOSW
STOSD
```

ตัวอย่าง โปรแกรมกำหนดค่าเริ่มต้นเท่ากับ 0 ให้กับข้อมูล 100 ตัวที่เริ่มต้นที่หน่วยความจำตำแหน่ง d1

```
.data
d1      dw      100 dup (?)
.code
  mov   ax,@data
  mov   ds,ax
  mov   es,ax
  mov   ax,0
  mov   di,offset d1          ; destination address
  cld                         ; fill them forwards
  mov   cx,100                ; number of locations to be filled
rep     stosw
  ...
```

13.4 คำสั่ง LODS

คำสั่ง LODS เป็นคำสั่งที่ใช้สำหรับการคัดลอกข้อมูลขนาดไบท์จากหน่วยความจำตำแหน่ง DS:SI ไปยังรีจิสเตอร์ AL หรือ ขนาดเวิร์ดไปยังรีจิสเตอร์ AX หรือ ขนาดดับเบิลเวิร์ดไปยังรีจิสเตอร์ EAX ไปยัง พร้อมทั้งปรับค่ารีจิสเตอร์ SI โดยมีรูปแบบดังนี้

```
LODSB
LODSW
LODSD
```

ตัวอย่าง โปรแกรมหาค่าผลรวมของข้อมูลแบบไบต์ 100 ตัวที่เริ่มต้นที่ d1 และเก็บผลรวมที่ได้ไว้ใน DX

```
.data
d1      db      100 dup (?)
.code
```

```
        mov     ax,@data
        mov     ds,ax
        ...
        mov     dx,0
        mov     si,offset d1            ; destination address
        cld                              ; forward direction
        mov     cx,100                   ; number of bytes to be moved
calsum: lodsb                           ;move the next byte into AL
        add     dl,al                    ;add AL to DX
        adc     dh,0
        loop    calsum                  ;if more to do, go back and repeat
        ...
```

โดยปกติจะไม่พบการใช้คำสั่ง REP ร่วมกับคำสั่ง LODS

ตัวอย่างคำสั่ง

```
        .data
d1      dw      100 dup (?)
.code
        mov     ax,@data
        mov     ds,ax
        ...
        mov     di,offset d1
        cld
        mov     cx,100
        rep     lodsw
        ...
```

จากโปรแกรมข้างต้นเป็นตัวอย่างการใช้คำสั่ง LODS อ่านค่าจากหน่วยความจำตำแหน่ง d1 ทีละเวิร์ดไปเก็บยังรีจิสเตอร์ AX ซึ่งจะเห็นได้ว่าค่าในรีจิสเตอร์ AX จะถูกเขียนทับไปเรื่อยๆ โดยในที่สุดก็มีเพียงค่าสุดท้ายเท่านั้น ไม่มีประโยชน์อันใด

ตัวอย่าง โปรแกรมคำนวณค่าจำนวน 100 ค่าเพื่อใส่ลงในหน่วยความจำที่เริ่มต้นที่ตำแหน่ง d2 โดยมีค่าเท่ากับค่าในหน่วยความจำที่เริ่มต้นที่ตำแหน่ง d1 ที่มีลำดับเท่ากันคูณด้วย 2

```
.data
d1      db      100 dup (?)
d2      db      100 dup (?)

.code
  mov   ax,@data
  mov   ds,ax
  ...
  mov   ax,ds
  mov   es,ax
  mov   si,offset d1          ;source address
  mov   di,offset d2          ;destination address
  cld                         ;forward direction
  mov   cx,100                ;number of bytes to be moved
calsum: lodsb                 ;move next byte to AL
  shl   al,1                   ;multiply AL by 2
  stosb                       ;store it in the new location
  loop  calsum                ;if more to do ,go back and repeat
  ...
```

13.5 คำสั่ง REPZ

คำสั่ง REPZ มีการทำงานคล้ายกับคำสั่ง REP แต่คำสั่ง REPZ จะกระทำซ้ำเมื่อ zero flag มีค่าเป็น 1 และ CX มีค่าไม่เท่ากับ 0

13.6 คำสั่ง CMPS

คำสั่ง CMPS จะนำค่าในหน่วยความจำตำแหน่ง ES:DI มาลบออกจากค่าในหน่วยความจำตำแหน่ง DS:SI แล้วปรับค่าแฟล็กต่างๆ ที่เกี่ยวข้องโดยค่าของข้อมูลในหน่วยความจำยังคงเดิม และปรับค่าของรีจิสเตอร์ DI และ SI โดยอัตโนมัติ คำสั่ง CMPS มีรูปแบบดังนี้

```
CMPSB
CMPSW
CMPSD
```

คำสั่ง CMPS มักจะใช้ในการเปรียบเทียบ 2 สตริงว่าเหมือนกันหรือไม่ในการจะค้นหาว่าข้อมูลของสตริง 2 ตัวว่ามีค่าเท่ากันหรือไม่นั้น เราจะต้องทำการเปรียบเทียบข้อมูลไปทีละตัวจนกว่าจะหมดข้อมูลหรือพบข้อมูลที่ไม่เท่ากันเป็นตัวแรก จากที่ผ่านมาเราใช้คำสั่ง REP ร่วมกับคำสั่งอื่นเพื่อให้คำสั่งนั้นซ้ำเท่าจำนวนที่เรากำหนด(ในรีจิสเตอร์ CX) แต่ถ้าเราต้องการให้หยุดกระทำคำสั่งเมื่อพบจุดที่แตกต่าง นั่นก็คือเราจะใช้คำสั่ง REPZ แทนคำสั่ง REP เพราะคำสั่ง REPZ จะหยุดกระทำเมื่อ Z–flag เป็น 0 นั่นคือเมื่อคำสั่ง CMPS ทำการเปรียบเทียบพบข้อมูลที่ต่างกันนั่นเอง

ตัวอย่าง โปรแกรมเปรียบเทียบข้อมูลในหน่วยความจำที่ตำแหน่งเริ่มต้นที่ d1 และในหน่วยความจำตำแหน่งเริ่มต้นที่ d2

```
.data
d1      db      100 dup (?)
d2      db      100 dup (?)
.code
        mov     ax,@data
```

```
        mov     ds,ax
  ...
  mov     bx,ds
  mov     es,bx
  mov     si, offset d1           ;start address of first string
  mov     di, offset d2           ;start address of second string
  cld                     ;forward direction
  mov     cx,100                  ;the number of words to be compare
repz      cmpsw
  jnz     differ
same:     ...
            ...
            jmp         done
differ:   ...
            ...
done:     ...
            ...
```

13.7 คำสั่ง REPNZ

คำสั่ง REPNZ มีการทำงานคล้ายกับคำสั่ง REP แต่คำสั่ง REPZ จะกระทำซ้ำเมื่อ zero flag มีค่าเป็น 0 และ CX มีค่าไม่เท่ากับ 0

13.8 คำสั่ง SCAS

คำสั่ง SCAS จะค้นหาข้อมูลสำหรับค่าขนาดไบต์ที่กำหนดใน AL หรือขนาดเวิร์ดใน AX หรือขนาดดับเบิลเวิร์ดใน EAX โดยเริ่มต้นที่หน่วยความจำตำแหน่ง ES:DI จากนั้นจะมีการปรับค่าของรีจิสเตอร์ DI โดยอัตโนมัติ โดยมีรูปแบบดังนี้

```
SCASB
SCASW
SCASD
```

คำสั่ง SCAS มักจะใช้ในการค้นหาข้อมูลภายในสตริง ในการจะค้นหานั้น เราจะต้องทำการเปรียบเทียบข้อมูลไปทีละตัวจนกว่าจะหมดข้อมูลหรือพบข้อมูลที่ต้องการ ในทำนองเดียวกับคำสั่ง CMPS เราไม่สามารถใช้คำสั่ง REP ได้เพราะเราต้องการให้หยุดกระทำคำสั่งเมื่อพบข้อมูลที่ต้องการ เราจึงใช้คำสั่ง REPNZ แทนคำสั่ง REP เพราะคำสั่ง REPNZ จะหยุดกระทำเมื่อ Z–flag เป็น 1 นั่นคือเมื่อคำสั่ง SCAS ทำการเปรียบเทียบพบข้อมูลที่เหมือนกันนั่นเอง

ตัวอย่าง โปรแกรมค้นหาอักขระ ‘X’ จากข้อมูล 100 ตัว

```
.data
d1        db      100 dup (?)
.code
  mov     ax,@data
  mov     ds,ax
  ...
  mov     bx,ds
  mov     es,bx
  mov     di,offset d1          ;first location to be searched
  cld                           ;forward direction
  mov     cx,100                ;the number of locations to be searched
            mov       al, ‘X’
repnz     scasb                 ;do the search
  jz      found
notfnd:   ...
            jmp       done
found:    ...
            ...
done:     ...
```

ตัวอย่าง โปรแกรมสำหรับนำข้อมูลที่เริ่มต้นที่ d2 จำนวน 100 ตัวไปต่อท้ายข้อมูลที่เริ่มต้นที่ d1 โดยถือว่าข้อมูลที่เริ่มต้นที่ d1 ถูกปิดท้ายด้วย null string

```
    ...
    mov     ax,@data
    mov     ds,ax
    mov     bx,ds
    mov     es,bx
    mov     al, 0
    mov     di, offset d1          ;first location to be searched
    cld                             ;forward direction
repnz       scasb                   ;do the search
    mov     si, offset d2           ;start address of second string
    mov     cx,100                  ;number of bytes to be moved
rep         movsb                   ;do the move
    ...
```

สรุป

คำสั่งจัดการกับสายข้อมูล เป็นคำสั่งที่ใช้จัดการกับข้อมูลเป็นที่เป็นชุดเรียงต่อกัน คำสั่งในกลุ่มนี้จะระบุตำแหน่งของข้อมูลด้วย Index register คือ SI (Source Index) และ DI (Destination Index) ประกอบกับ DS และ EX โดย ข้อมูลต้นทางจะอยู่ที่ตำแหน่ง DS:SI และข้อมูลปลายทางจะอยู่ที่ ES:DI การทำงานคำสั่งในกลุ่มนี้จะมีการปรับค่าของรีจิสเตอร์ที่เก็บตำแหน่งของข้อมูลให้โดยอัตโนมัติ เพื่อว่าเราจะสามารถใช้คำสั่งนั้นกระทำกับข้อมูลที่ตำแหน่งติดกันถัดไปได้โดยไม่ต้องปรับค่าของรีจิสเตอร์เอง ทิศทางในการปรับจะขึ้นกับค่าที่กำหนดในแฟล็กทิศทาง (DF : Direction flag)

คำถามทบทวน

1. จงอธิบายการทำงานของรีจิสเตอร์ต่อไปนี้ SI DI DS ES และ DF
2. จงแสดงและจงอธิบายการทำงานของคำสั่งต่อไปนี้
   - 2.1 คำสั่ง MOVS
   - 2.2 คำสั่ง REP
   - 2.3 คำสั่ง STOS
   - 2.4 คำสั่ง LODS
   - 2.5 คำสั่ง REPZ
   - 2.6 คำสั่ง CMPS
   - 2.7 คำสั่ง REPNZ
   - 2.8 คำสั่ง SCAS
3. จงเขียนโปรแกรมสำหรับนำข้อมูลที่เริ่มต้นที่ data2 จำนวน 25 ตัวไปต่อท้ายข้อมูลที่เริ่มต้นที่ data1 โดยถือว่าข้อมูลที่เริ่มต้นที่ data1 ถูกปิดท้ายด้วย null string

เอกสารอ้างอิง

ราชบัณฑิตยสถาน. (2544). *ศัพท์บัญญัติ ราชบัณฑิตยสถาน.* ค้นเมื่อ 8 กรกฎาคม 2557,จาก :http://rirs3.royin.go.th/coinages/

คำสั่งจัดการกับสายข้อมูล (2557). *วิกิพีเดีย สารานุกรมเสรี.* ค้นเมื่อ 8 กรกฎาคม 2557,จาก :http://th.wikipedia.org/wiki/

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

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