แผนการสอนประจำสัปดาห์ที่ 6

หัวข้อเรื่อง คำสั่งโอนย้ายข้อมูล แฟล็กและคำสั่งคณิตศาสตร์
(Command Transfer, Flags and Mathematics Instruction)

รายละเอียด

ศึกษาเกี่ยวกับคำสั่งภาษาแอสแซมบลีเบื้องต้น โดยจะเป็นคำสั่งเกี่ยวกับการโอนย้ายข้อมูล ทั้งจากรีจิสเตอร์และหน่วยความจำ ในตอนท้ายของบทนี้จะเป็นการแนะนำโปรแกรม DEBUG ซึ่งจะเป็นเครื่องมือที่เราจะใช้ทดลองการโปรแกรมภาษาแอสแซมบลีเบื้องต้น การใช้คำสั่งคำนวณทางคณิตศาสตร์และการแสดงสถานะของผลลัพธ์ของการคำนวณนั้นในแฟล็ก สถานะที่เก็บในแฟล็กจะใช้สำหรับการจัดการกับผลการคำนวณนั้น ๆ รวมถึงใช้ในคำสั่งเกี่ยวกับการกระโดดแบบมีเงื่อนไขด้วย

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 6 การจัดการเรียนการสอน จะเกี่ยวข้องกับคำสั่งในการโยนย้ายข้อมูล การใช้คำสั่ง MOV เครื่องมือในการทดลองการโปรแกรมภาษาแอสแซมบลี คำสั่งในการโยนย้ายข้อมูล และคำสั่งทั่วไป การทำงานของแฟล็ก (Flags) คำสั่งทางคณิตศาสตร์ กลุ่มคำสั่งบวกและลบ กลุ่มคำสั่งคูณและหาร กลุ่มคำสั่งแปลงขนาดตัวเลข ซึ่งคำสั่งภาษาแอสแซมบลีเบื้องต้น จะเป็นคำสั่งเกี่ยวกับการโอนย้ายข้อมูล ทั้งจากรีจิสเตอร์และหน่วยความจำ ซึ่งในเนื้อหาจะมีการแนะนำการใช้โปรแกรม DEBUG ซึ่งจะเป็นเครื่องมือที่ใช้ทดลองในการเขียนโปรแกรมภาษาแอสแซมบลีเบื้องต้น

6.1 คำสั่งในการโอนย้ายข้อมูล

คำสั่ง MOV

คำสั่ง MOV เป็นคำสั่งที่ใช้สำหรับการคัดลอกข้อมูลจากแหล่งข้อมูลต้นทางไปยังแหล่งข้อมูลปลายทาง โดยมีรูปแบบดังนี้

```
MOV     reg, reg        reg : register
MOV     reg, mem        mem : memory
```

MOV mem, reg imm : immediate (ค่าคงที่)

MOV reg, imm

MOV mem, imm

ในการคัดลอกข้อมูลจะกระทำจากตัวถูกดำเนินการ (operand) ตัวหลังไปยังตัวถูกดำเนินการ (operand) ตัวหน้า ซึ่งจะเห็นว่าไม่สามารถคัดลอกข้อมูลจากหน่วยความจำไปยังหน่วยความจำได้ (ศัพท์บัญญัติ ราชบัณฑิตยสถาน, 2544)

ข้อจำกัดของคำสั่ง MOV

* ตัวถูกดำเนินการ (Operand) ทั้งสองตัวต้องมีขนาดเท่ากัน
* ไม่สามารถคัดลอกค่าคงที่ (immediate) ไปยังเซกเมนต์รีจิสเตอร์ได้โดยตรง ต้องกระทำผ่านทางรีจิสเตอร์อื่น ๆ เช่น AX เป็นต้น
* การคัดลอกค่าคงที่ไปยังหน่วยความจำจะต้องระบุขนาดของหน่วยความจำด้วย

ตัวอย่าง

MOV	AX,100h	กำหนดค่ารีจิสเตอร์ AX ให้เป็น 100h จากนั้นคัดลอกค่าจาก AX ไปยัง BX และคัดลอกจาก BX ไปยัง DX ตามลำดับ
MOV	BX,AX	
MOV	DX,BX	
MOV	AX,1234h	กำหนดค่า 1234h ให้กับ AX และ 5678h ให้กับ DX จากนั้น คัดลอกค่าในรีจิสเตอร์ DL (มีค่าเท่ากับ 78h เพราะอะไร?) ไปให้กับรีจิสเตอร์ AL และคัดลอกค่าจาก DH (มีค่าเท่ากับ 56h เพราะอะไร?) ไปยังรีจิสเตอร์ DH
MOV	DX,5678h	
MOV	AL,DL	
MOV	BH,DH	
MOV	AX,1000h	กำหนดค่า 1000h ให้กับ AX จากนั้นคัดลอกข้อมูลจาก AX ไปยังหน่วยความจำตำแหน่ง offset ที่ 100h และคัดลอกข้อมูลจากหน่วยความจำตำแหน่งนั้นมายัง BX
MOV	[100h],AX	
MOV	BX,[100h]	
MOV	BYTE PTR [200h],10h	กำหนดค่า 10h ไปยังตำแหน่งในหน่วยความจำที่ offset 200h โดยโอนย้ายข้อมูลแบบขนาด 1 Byte และกำหนดค่า 10h ที่มีขนาด 1 word (0010h) ไปยังตำแหน่งในหน่วยความจำที่ offset 300h
MOV	WORD PTR [300h],10h	
MOV	AX,2300h	กำหนดค่า 2300h ให้กับรีจิสเตอร์ DS โดยต้องกำหนดผ่านรีจิสเตอร์ขนาด 16 บิตตัวอื่น (ในที่นี้คือ AX)
MOV	DS,AX	

การโอนย้ายข้อมูลระหว่างรีจิสเตอร์กับรีจิสเตอร์ การโอนย้ายข้อมูลระหว่างรีจิสเตอร์สามารถทำได้ถ้าขนาดของรีจิสเตอร์ทั้งคู่เท่ากัน แต่ยังมีเซกเมนต์รีจิสเตอร์บางตัวซึ่งไม่ควรเข้าไปกำหนดค่าโดยตรง เช่น CS หรือ SS

คู่รีจิสเตอร์ 16 บิต กับ 8 บิต ในการใช้งานรีจิสเตอร์ทั่วไปเราจะต้องคำนึงถึงเรื่องของคู่รีจิสเตอร์ 16 บิต กับ 8 บิตด้วย จากที่เราได้ทราบมาแล้วว่ารีจิสเตอร์ 8 บิตที่เราใช้ได้นั้น เป็นส่วนหนึ่งของรีจิสเตอร์ทั่วไปขนาด 16 บิต เช่น รีจิสเตอร์ AH เป็นไบต์สูงรีจิสเตอร์ AX (บิตที่ 8–15) เป็นต้น ดังนั้นถ้าเรากำหนดค่าให้กับรีจิสเตอร์ 8 บิตก็จะมีการเปลี่ยนแปลงกระทบถึงรีจิสเตอร์ 16 บิตที่รีจิสเตอร์นั้นประกอบอยู่ด้วย และในทางกลับกันการเปลี่ยนแปลงค่าในรีจิสเตอร์ 16 บิตก็มีผลกระทบมาถึงรีจิสเตอร์ 8 บิตด้วย

ตัวอย่าง

คำสั่ง		**ค่าในริจิสเตอร์หลังการทำงานของคำสั่ง**		
MOV	AX,1000h	AX = 1000h	AH = 10h	AX = 00h
MOV	AL,3Ah	AX = 103Ah	AH = 10h	AX = 3Ah
MOV	AH,AL	AX = 3A3Ah	AH = 3Ah	AX = 3Ah
MOV	AX,234h	AX = 234h	AH = 02h	AX = 34h

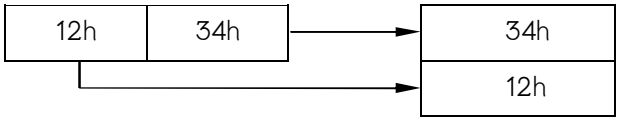
การโอนย้ายข้อมูลกับหน่วยความจำ โดยปกติการโอนย้ายข้อมูลกับหน่วยความจำนั้น โดยบุเฉพาะออฟเซ็ตของหน่วยความจำที่เราต้องการจะโอนย้ายข้อมูลด้วยค่าของออฟเซ็ตนั้นจะถูกนำมาประกอบกับเซกเมนต์รีจิสเตอร์ DS เพื่อใช้เป็นตำแหน่งในหน่วยความจำที่แท้จริงที่จะอ่านเขียนข้อมูลด้วย

ตัวอย่างโปรแกรม		**ตำแหน่ง**	**ค่าในหน่วยความจำหลังการทำงานของคำสั่งที่**						
MOV	AX,1234h	DS:	1,2	3	4	5			
MOV	BX,6789h						6	7	8,9
MOV	[100h],AH	100h	?	12h	12h	12h	12h	12h	12h
MOV	[101h],BL	101h	?	?	89h	89h	89h	89h	89h
MOV	[102h],BX	102h	?	?	?	89h	89h	89h	89h
MOV	[104h],AX	103h	?	?	?	67h	67h	67h	67h
MOV	[105h],BH	104h	?	?	?	?	34h	34h	34h
MOV	AX,[104h]	105h	?	?	?	?	12h	67h	67h
MOV	BL,[103h]	106h	?	?	?	?	?	?	?

หลังการทำงาน รีจิสเตอร์ AX มีค่าเท่ากับ 6734h และ BX มีค่าเท่ากับ 3467h

ข้อสังเกตในการจัดเรียงไบต์เมื่อเก็บข้อมูลในหน่วยความจำ

สังเกตว่าในการเก็บค่าในหน่วยความจำเมื่อเราเก็บค่าเป็น 16 บิต การเรียงไบต์ในหน่วยความจำจะเก็บค่าในไบต์ที่มีนัยสำคัญสูงไว้ในไบต์ที่มีแอดเดรสสูงกว่าและไบต์ที่มีนัยสำคัญต่ำไว้ในแอดเดรสที่มีแอดเดรสต่ำกว่า แสดงได้ดังภาพที่ 6.1



ภาพที่ 6.1 แสดงการเรียงไบต์ข้อมูลในหน่วยความจำ

ลักษณะของการเก็บข้อมูลโดยเรียงไบต์ที่มีนัยสำคัญต่ำไว้ที่แอดเดรสต่ำและไบต์ที่มีนัยสำคัญสูงไว้ที่แอดเดรสสูงเรียกว่าเป็นการเรียงแบบ **C**

ในการกำหนดออฟเซ็ตของหน่วยความจำที่จะอ่านและเขียนข้อมูลนั้น จากตัวอย่างข้างต้นเป็นการระบุโดยใส่หมายเลขออฟเซ็ตโดยตรง และสามารถที่จะระบุออฟเซ็ตโดยผ่านทางค่าในรีจิสเตอร์ BX (base register) ได้อีกทางหนึ่ง

ตัวอย่าง

```
MOV   BX,100h
MOV   AX,[BX]
MOV   BX,102h
MOV   [BX],DX
```

การกำหนดค่าคงที่ให้กับหน่วยความจำ การกำหนดค่าคงที่ให้กับหน่วยความจำสามารถกระทำได้ แต่จะต้องมีการระบุขนาดของข้อมูลที่จะคัดลอกสู่หน่วยความจำ เพื่อที่จะป้องกันการสับสนดังตัวอย่างเช่น คำสั่ง MOV [100h],10h จากคำสั่งดังกล่าว แอสเซมเบลอร์จะไม่สามารถทราบได้เลยว่าการคัดลอกข้อมูลจะเป็นในลักษณะของ 8 บิต หรือ 16 บิต

ในการระบุขนาดของการคัดลอกข้อมูลเราจะใช้ keyword ว่า BYTE PTR (Byte Pointer) หรือ WORD PTR (Word Pointer) สำหรับระบุว่าการอ้างถึงหน่วยความจำในตำแหน่งนั้นเป็นแบบไบต์ หรือ เวิร์ด

ตัวอย่าง

```
MOV   BYTE PTR [100h],10h
MOV   WORD PTR [102h],10h
```

```
MOV    BX,104h
MOV    WORD PTR [BX],2345h
```

6.2 เครื่องมือในการทดลองการโปรแกรมภาษาแอสแซมบลี : โปรแกรม DEBUG

โปรแกรม DEBUG เป็นโปรแกรมสารพัดประโยชน์สำหรับการทดลองเกี่ยวกับการเขียนโปรแกรมภาษาแอสแซมบลี โปรแกรมนี้มีอยู่ทั้งในระบบปฏิบัติการ DOS และ Windows

เมื่อเรียกโปรแกรม DEBUG โดยพิมพ์ DEBUG ที่ DOS prompt จะเห็นเครื่องหมายเตรียมพร้อมของโปรแกรม DEBUG เป็นเครื่องหมายขีด

```
-
```

ผู้เขียนโปรแกรมสามารถจะพิมพ์คำสั่งต่าง ๆ ลงไปได้ที่ prompt นี้

6.3 คำสั่งในการโอนย้ายข้อมูล

คำสั่งทั่วไป

คำสั่งแสดงความช่วยเหลือ : ?

ผู้เขียนโปรแกรมสามารถสั่งให้โปรแกรม DEBUG แสดงรายการคำสั่งต่าง ๆ ได้โดยใช้คำสั่ง ?

คำสั่งจัดการกับรีจิสเตอร์ : R (register)

คำสั่ง R คือคำสั่งให้โปรแกรม DEBUG แสดงค่าในรีจิสเตอร์รวมถึงกำหนดค่าให้กับรีจิสเตอร์ต่าง ๆ ด้วย ถ้าใช้คำสั่ง R โดยไม่ระบุชื่อรีจิสเตอร์ โปรแกรม DEBUG จะแสดงค่าในรีจิสเตอร์ออกมาให้เห็น

```
-R
AX=0000   BX=0000   CX=0000   DX=0000   SP=FFEE   BP=0000
SI=0000  DI=0000
DS=12AF  ES=12AF  SS=12AF  CS=12AF  IP=0100   NV UP EI PL NZ NA
PO NC
12AF:0100 5F            POP     DI
-
```

โปรแกรม DEBUG จะแสดงทั้งค่าในรีจิสเตอร์ คำสั่งถัดไปที่จะทำงาน รวมถึงแฟล็กต่าง ๆ ด้วย สามารถระบุชื่อรีจิสเตอร์ต่อท้ายคำสั่ง R เพื่อกำหนดค่าให้กับรีจิสเตอร์นั้น ดังตัวอย่าง

```
-RCX
CX 0000
:100
-R
AX=0000  BX=0000  CX=0100  DX=0000  SP=FFEE  BP=0000
SI=0000  DI=0000
DS=12AF  ES=12AF  SS=12AF  CS=12AF  IP=0100  NV UP EI PL NZ NA
PO NC
12AF:0100 5F            POP     DI
-
```

คำสั่งแสดงค่าในหน่วยความจำ : D (dump)

ผู้เขียนโปรแกรมสามารถสั่งให้โปรแกรม DEBUG แสดงค่าข้อมูลในหน่วยความจำได้ โดยใช้คำสั่ง D (dump) สามารถระบุตำแหน่งของหน่วยความจำที่เริ่มต้นแสดงข้อมูลได้ ถ้าไม่กำหนดโปรแกรม DEBUG จะแสดงค่าในหน่วยความจำถัดจากตำแหน่งเก่าที่แสดงไว้

```
-D100
12AF:0100    C7 06 16 00 6E 30 26 C7-06 18 00 C7 40 26 8B 06
....n0&.....@&..
12AF:0110    08 00 26 03 06 0C 00 26-3B 06 06 00 34 00 9E 12
..&....&;...4...
                    . . .
12AF:0170    C2 02 00 90 C8 04 00 00-57 56 8B 7E 0A 57 E8 57
........WV.~.W.W
-
```

การทดลองโปรแกรมภาษาแอสแซมบลีใน DEBUG

ผู้เขียนโปรแกรมสามารถป้อนโปรแกรมภาษาแอสแซมบลีเบื้องต้นได้ในโปรแกรม DEBUG และสามารถสั่งให้โปรแกรมนั้นทำงานเพื่อสังเกตผลการทำงานได้ อีกทั้งยังสามารถสั่งให้โปรแกรมทำงานทีละบรรทัดได้ด้วย

คำสั่งสำหรับแปลโปรแกรมภาษาแอสแซมบลี : A (assemble)

ผู้เขียนโปรแกรมสามารถใช้คำสั่ง A (assemble) เพื่อสั่งให้โปรแกรม DEBUG รับคำสั่งภาษาแอสแซมบลีแล้วแปลคำสั่งนั้นเป็นภาษาเครื่องเก็บไว้ในหน่วยความจำได้ ในการสั่งคำสั่ง assemble จะระบุตำแหน่งที่โปรแกรมภาษาเครื่องได้แปลแล้วจะถูกเขียนลงไปทันที แต่ถ้าไม่ระบุตำแหน่ง โปรแกรมที่เขียนจะถูกเก็บไว้ที่ตำแหน่งถัดจากการแปลครั้งสุดท้าย ในการป้อนโปรแกรมโดยป้อนโปรแกรมต่อเนื่องไปได้เรื่อย ๆ เมื่อป้อนเสร็จแล้วให้กดปุ่ม Enter ว่างไปหนึ่งบรรทัดโปรแกรม DEBUG จะแสดง prompt เพื่อรับคำสั่งใหม่ต่อไป

```
A100
12AF:0100 mov ax,10
12AF:0103 mov bx,20
12AF:0106 mov [200],ax
12AF:0109 mov [202],bx
12AF:010D mov bx,204
12AF:0110 mov cx,1234
12AF:0113 mov [bx],cx
12AF:0115 int 20
12AF:0117
-
```

ข้อสังเกต ตัวเลขต่าง ๆ ในโปรแกรม DEBUG จะถือว่าเป็นเลขฐานสิบหกทั้งหมด ถ้ามีการป้อนโปรแกรมผิดโปรแกรม DEBUG จะรายงานว่าโปรแกรมผิดให้ผู้เขียนโปรแกรมทราบและสามารถป้อนโปรแกรมเข้าไปใหม่ได้

คำสั่งสำหรับแสดงโปรแกรมภาษาแอสแซมบลีจากภาษาเครื่อง : U (unassemble)

ผู้เขียนโปรแกรมสามารถสั่งให้โปรแกรม DEBUG แสดงคำสั่งภาษาเครื่องที่อยู่ในหน่วยความจำเป็นรหัสคำสั่งภาษาแอสแซมบลีโดยใช้คำสั่ง unassemble ทำการระบุตำแหน่งที่จะเริ่มแสดงได้

```
-U100
12AF:0100 B81000      MOV     AX,0010
12AF:0103 BB2000      MOV     BX,0020
12AF:0106 A30002      MOV     [0200],AX
12AF:0109 891E0202    MOV     [0202],BX
12AF:010D BB0402      MOV     BX,0204
12AF:0110 B93412      MOV     CX,1234
12AF:0113 890F        MOV     [BX],CX
12AF:0115 CD20        INT     20
12AF:0117 26          ES:
12AF:0118 3B060600    CMP     AX,[0006]
12AF:011C 3400        XOR     AL,00
12AF:011E 9E          SAHF
12AF:011F 12FE        ADC     BH,DH
-
```

โปรแกรม DEBUG จะแสดงทั้งรหัสภาษาเครื่องและรหัสนีโมนิค (รหัสภาษา assembly) ด้วยการสั่งให้โปรแกรมทำงานผู้เขียนโปรแกรมสามารถสั่งให้โปรแกรมทำงานได้ทั้งสิ้น 3 รูปแบบโดยโปรแกรมจะเริ่มทำงานที่ตำแหน่ง CS:IP เท่านั้น

คำสั่งเริ่มทำงาน : G (go)

เมื่อมีการสั่งให้โปรแกรมทำงานโดยใช้คำสั่ง g โปรแกรมจะเริ่มทำงานทันที และโปรแกรมจะทำงานจนกระทั่งจบ

```
-G
Program terminated normally
```

ข้อสังเกต โปรแกรมตัวอย่างจะมีคำสั่ง INT 20h ระบุอยู่ตอนท้าย คำสั่งนี้เป็นคำสั่งเรียกใช้บริการของระบบเพื่อที่จะจบโปรแกรม ถ้าไม่มีคำสั่งนี้ปิดท้าย โปรแกรมจะทำงานตามคำสั่งต่อไปเรื่อย ๆ จนกว่าจะพบคำสั่งที่สั่งให้โปรแกรมหยุดการทำงาน ดังนั้นเวลาทดลองเขียนโปรแกรมในโปรแกรม DEBUG ควรจะใส่คำสั่ง INT 20h ปิดท้ายไว้ทุกครั้ง

คำสั่งตามรอยการทำงาน (คำสั่งให้ทำงานทีละบรรทัดแบบที่หนึ่ง) : T (trace)

คำสั่งนี้โปรแกรมจะทำงานไปหนึ่งคำสั่งแล้วจะกลับมาที่โปรแกรม DEBUG และแสดงค่าของรีจิสเตอร์ต่าง ๆ รวมถึงผลจากคำสั่งนั้นทันที ทำให้สามารถตรวจสอบสถานะของโปรแกรมได้ตลอดขั้นตอนการทำงาน

```
-T
AX=0010  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000
SI=0000  DI=0000
DS=12AF  ES=12AF  SS=12AF  CS=12AF  IP=0103  NV UP EI PL NZ NA
PO NC
12AF:0103 BB2000        MOV     BX,0020
-T
AX=0010  BX=0020  CX=0000  DX=0000  SP=FFEE  BP=0000
SI=0000  DI=0000
DS=12AF  ES=12AF  SS=12AF  CS=12AF  IP=0106  NV UP EI PL NZ NA
PO NC
12AF:0106  A30002                   MOV             [0200],AX
DS:0200=0010
```

ในการทำงานโดยคำสั่ง T โปรแกรมที่ทำงานจะหยุดการทำงานกลับมาที่โปรแกรม DEBUG ภายหลังการทำงานของทุก ๆ คำสั่ง รวมถึงในกรณีที่มีการเรียกใช้โปรแกรมย่อยหรือเรียกใช้บริการของระบบด้วย การให้โปรแกรมทำงานไปหยุดที่บรรทัดถัดไปเลยโดยไม่ต้องคำนึงถึงการเรียกโปรแกรมย่อยต่าง ๆ ผู้เขียนโปรแกรมสามารถใช้คำสั่ง P (proceed) แทนคำสั่ง trace ได้

คำสั่งทำงานต่อจนถึงบรรทัดถัดไป (คำสั่งให้ทำงานทีละบรรทัดแบบที่สอง) : P (proceed)

ผลโดยทั่ว ๆ ไปของคำสั่งนี้เหมือนกับการใช้คำสั่ง T แต่การใช้คำสั่งนี้จะสะดวกกว่าเมื่อโปรแกรมที่ทำงานมีการเรียกโปรแกรมย่อยหรือมีการเรียกใช้บริการจากระบบ โดยโปรแกรมจะกลับมาที่ DEBUG เมื่อทำงานถึงบรรทัดถัดไป

การตั้งค่าให้กับรีจิสเตอร์ IP

เนื่องจากโปรแกรม DEBUG จะประมวลผลคำสั่งที่ตำแหน่ง CS:IP เสมอ แต่เมื่อโปรแกรมทำงานเสร็จ รีจิสเตอร์ IP จะชี้ที่ออฟเซ็ตของคำสั่งถัดไปซึ่งเป็นคำสั่งที่ไม่ใช่ส่วนของโปรแกรมของผู้เขียน ดังนั้นภายหลังการทำงานของโปรแกรม ผู้เขียนโปรแกรมควรตรวจสอบว่ารีจิสเตอร์ IP ชี้ไปยังจุดเริ่มต้นของโปรแกรมที่เขียนไว้หรือไม่ ซึ่งโดยปกติแล้วโปรแกรมจะเริ่มต้นที่ออฟเซ็ต 100h ถ้ารีจิสเตอร์ IP ชี้ไปที่อื่นและตั้งค่าให้รีจิสเตอร์ IP ได้โดยใช้คำสั่ง R

```
-RIP
IP 0106
:100
-
```

ตัวอย่างการทดลองโปรแกรมภาษาแอสแซมบลี

ยกตัวอย่างการทดลองโปรแกรมภาษาแอสแซมบลีต่อไปนี้ด้วยโปรแกรม DEBUG

ตัวอย่างโปรแกรม

```
MOV   AX,5678h
MOV   BX,204h
MOV   CX,1234h
MOV   [200h],CX
MOV   [202h],AH
MOV   [203h],AL
MOV   [BX],AX
MOV   DX,[202h]
MOV   [204h],10h
```

ป้อนโปรแกรม

ผู้เขียนโปรแกรมสามารถที่จะป้อนโปรแกรมและสั่งให้ DEBUG แปลโปรแกรมลงในหน่วยความจำโดยใช้คำสั่ง A ไม่ระบุแอดเดรสเริ่มต้นในครั้งแรกโปรแกรมจะถูกแปลลงในตำแหน่ง CS:100h และถ้าสั่งครั้งถัดไป โปรแกรมจะแปลตามลำดับต่อเนื่องกับการสั่งครั้งก่อน

```
-A100
14FF:0100 mov ax,5678
```

```
14FF:0103 mov bx,204
14FF:0106 mov cx,1234
14FF:0109 mov [200],cx
14FF:010D mov [202],ah
14FF:0111 mov [203],al
14FF:0114 mov [bx],ax
14FF:0116 mov dx,[202]
14FF:011A mov [204],10
^ Error
```

ในระหว่างการป้อนโปรแกรมอาจมีข้อผิดพลาดขึ้นโปรแกรม DEBUG จะแสดงข้อความขึ้นดังตัวอย่าง จากโปรแกรมดังกล่าวผู้เขียนโปรแกรมจะต้องระบุขนาดของการคัดลอกข้อมูลด้วย โดยสามารถใส่คำสั่งที่แก้ไขแล้วลงไปในบรรทัดใหม่ได้ทันที

```
14FF:011A mov word ptr [204],10
14FF:0120
```

เมื่อใส่โปรแกรมเสร็จจะต้องกดปุ่ม Enter ว่าง ๆ ไปเพื่อบอกโปรแกรม DEBUG ว่าโปรแกรมสิ้นสุดแล้วและสามารถเรียกดูโปรแกรมที่ใส่ลงไปได้ด้วยคำสั่ง U

```
-U
14FF:0100 B87856          MOV     AX,5678
14FF:0103 BB0402          MOV     BX,0204
14FF:0106 B93412          MOV     CX,1234
14FF:0109 890E0002        MOV     [0200],CX
14FF:010D 88260202        MOV     [0202],AH
14FF:0111 A20302          MOV     [0203],AL
14FF:0114 8907            MOV     [BX],AX
14FF:0116 8B160202        MOV     DX,[0202]
14FF:011A C70604021000    MOV     WORD PTR [0204],0010
```

ผู้เขียนโปรแกรมจะเห็นทั้งโปรแกรมภาษาแอสแซมบลีและโปรแกรมภาษาเครื่องที่แปลแล้ว

ติดตามการทำงานของโปรแกรม

ผู้เขียนโปรแกรมสามารถสั่งให้โปรแกรมทำงานโดยใช้คำสั่ง G (go) และตรวจสอบค่าในหน่วยความจำได้ แต่โปรแกรมที่จะทดลองนั้นต้องมีการสั่งให้โปรแกรมจบการทำงาน มิฉะนั้นโปรแกรมจะทำงานเลยไปถึงโปรแกรมอื่น ๆ ที่อยู่ในหน่วยความจำ ดังนั้นอาจจะใส่คำสั่ง INT 20h ปิดท้ายโปรแกรมไว้เพื่อสั่งให้โปรแกรมจบการทำงาน จากตัวอย่าง การแสดงแอดเดรสถัดไปของคำสั่งคือออฟเซ็ตที่ 0120h (สามารถหาได้โดยใช้คำสั่ง U แล้วสังเกตค่าออฟเซ็ตของคำสั่งอื่น ๆ ถัดจากโปรแกรมที่ป้อน) ดังนั้นผู้เขียนโปรแกรมจะต้องป้อนคำสั่ง INT 20h ลงไปที่แอดเดรสนี้

```
-A120
14FF:0120 int 20
14FF:0122
-U100
14FF:0100 B87856            MOV     AX,5678
14FF:0103 BB0402            MOV     BX,0204
14FF:011A C70604021000      MOV     WORD PTR [0204],0010
-U11A
14FF:011A C70604021000      MOV     WORD PTR [0204],0010
14FF:0120 CD20              INT      20
                ...
```

เมื่อใส่คำสั่งให้โปรแกรมจบการทำงานแล้ว ผู้เขียนโปรแกรมสามารถใช้คำสั่ง G (go) เพื่อสั่งให้โปรแกรมทำงานและใช้คำสั่ง D (dump) เพื่อสังเกตค่าในหน่วยความจำหรือ คำสั่ง R (register) เพื่อดูค่าในรีจิสเตอร์อีกทีได้

```
-G
Program terminated normally
-D200
14FF:0200  34 12 56 78 10 00 DA EB-04 9D F8 EB 02 9D F9 89   4.Vx............
14FF:0210   36 8A DB 5E 5F 5A 59 C3-53 51 52 57 56 9C E8 6E  6..^_ZY.SQRWV..n
14FF:0220  00 83 3E 7A DB 40 7D 57-8A F7 8B 1E 7A DB FF 06   ..>z.@}W....z...
```

```
14FF:0230  7A DB B8 BA D8 E8 95 00-C7 47 07 00 00 F6 C6 01   z........G......
14FF:0240  74 03 89 6F 07 89 4F 05-88 77 02 8B 36 84 DB 89   t..o..O..w..6...
14FF:0250  37 03 36 5C D8 2B F7 89-77 03 8B 36 8A DB 89 77   7.6\.+..w..6...w
14FF:0260  09 8B F7 8B 3E 84 DB 03-F9 3B 3E 80 DB 7D 15 2B   ....>....;>..}.+
14FF:0270  F9 FC F3 A4 B0 00 AA 89-3E 84 DB 9D F8 EB 0A B8   ........>.......
-R
AX=0000  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=14FF  ES=14FF  SS=14FF  CS=14FF  IP=0100   NV UP EI PL NZ NA PO NC
14FF:0100 B87856        MOV     AX,5678
```

จะสังเกตเห็นว่าเมื่อมีการลับมายังโปรแกรม DEBUG หลังจากโปรแกรมที่ผู้เขียนป้อนเข้าไปทำงานเรียบร้อยแล้ว รีจิสเตอร์ต่าง ๆ จะถูกกำหนดค่าเริ่มต้องใหม่รวมทั้งรีจิสเตอร์ IP ด้วย ดังนั้นถ้าสังเกตผลของการทำงานจะได้จากค่าจากหน่วยความจำเท่านั้น อีกทั้งยังสามารถสั่งให้โปรแกรมทำงานทีละบรรทัดโดยใช้คำสั่ง T หรือ P แต่จะต้องระวังในเรื่องของการตั้งค่าเริ่มต้นของรีจิสเตอร์ IP ให้มาชี้หรือแสดงที่ตำแหน่งเริ่มต้นของโปรแกรมด้วย

```
-T
AX=5678  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000
SI=0000  DI=0000
DS=14FF  ES=14FF  SS=14FF  CS=14FF  IP=0103   NV UP EI PL NZ NA
PO NC
14FF:0103 BB0402        MOV     BX,0204
-T
AX=5678  BX=0204  CX=0000  DX=0000  SP=FFEE  BP=0000
SI=0000  DI=0000
DS=14FF  ES=14FF  SS=14FF  CS=14FF  IP=0106   NV UP EI PL NZ NA
PO NC
14FF:0106 B193412       MOV  CX,1234
-T
AX=5678  BX=0204  CX=1234  DX=0000  SP=FFEE  BP=0000
SI=0000  DI=0000
```

```
DS=14FF  ES=14FF  SS=14FF  CS=14FF  IP=0109   NV UP EI PL NZ NA
PO NC
14FF:0109 890E0002      MOV     [0200],CX
DS:0200=1234
-T
AX=5678  BX=0204  CX=1234  DX=0000  SP=FFEE  BP=0000
SI=0000  DI=0000
DS=14FF  ES=14FF  SS=14FF  CS=14FF  IP=010D   NV UP EI PL NZ NA
PO NC
14FF:010D 88260202      MOV     [0202],AH
DS:0202=56
-T
AX=5678  BX=0204  CX=1234  DX=0000  SP=FFEE  BP=0000
SI=0000  DI=0000
DS=14FF  ES=14FF  SS=14FF  CS=14FF  IP=0111   NV UP EI PL NZ NA
PO NC
14FF:0111 A20302        MOV     [0203],AL
DS:0203=78
-T
AX=5678  BX=0204  CX=1234  DX=7856  SP=FFEE  BP=0000
SI=0000  DI=0000
DS=14FF  ES=14FF  SS=14FF  CS=14FF  IP=0120   NV UP EI PL NZ NA
PO NC
14FF:0120 CD20          INT     20
-P
Program terminated normally
-
```

สังเกตว่าในคำสั่งสุดท้ายผู้เขียนโปรแกรมจะใช้คำสั่ง P เพราะคำสั่ง INT 20h เป็นการเรียกใช้บริการของระบบ และโปรแกรมจะกระโดดไปทำงานที่โปรแกรมย่อยของระบบซึ่ง

อาจจะยาวมาก ถ้าใช้คำสั่ง T จะได้เห็นการกระโดดไปทำงาน ณ ตำแหน่งที่ต้องการและใช้คำสั่ง G เพื่อให้โปรแกรมทำงานจนจบต่อจากการทำงานนั้นก็ได้

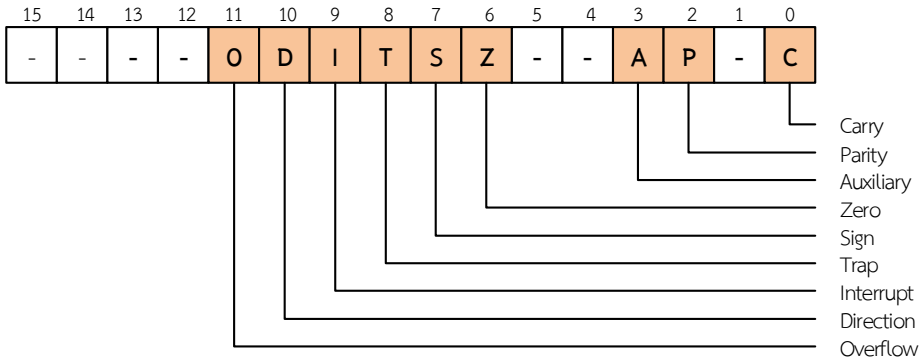
```
-T
AX=5678  BX=0204  CX=1234  DX=7856  SP=FFE8  BP=0000
SI=0000  DI=0000
DS=14FF  ES=14FF  SS=14FF  CS=14FF  IP=0120  NV UP DI PL NZ NA
PO NC
14FF:0120 CD20        INT     20
-T
AX=5678  BX=0204  CX=1234  DX=7856  SP=FFE2  BP=0000
SI=0000  DI=0000
DS=14FF  ES=14FF  SS=14FF  CS=00C9  IP=0FA8  NV UP DI PL NZ NA
PO NC
00C9:0FA8 90          NOP
-T
AX=5678  BX=0204  CX=1234  DX=7856  SP=FFE2  BP=0000
SI=0000  DI=0000
DS=14FF  ES=14FF  SS=14FF  CS=00C9  IP=0FA9  NV UP DI PL NZ NA
PO NC
00C9:0FA9 90          NOP
-T
AX=5678  BX=0204  CX=1234  DX=7856  SP=FFE2  BP=0000
SI=0000  DI=0000
DS=14FF  ES=14FF  SS=14FF  CS=00C9  IP=0FAA  NV UP DI PL NZ NA
PO NC
00C9:0FAA E8DB00      CALL    1088
-G
Program terminated normally
```

ข้อสังเกต ผู้เขียนโปรแกรมควรสังเกตค่าของรีจิสเตอร์ CS และ IP ก่อนที่จะเริ่มสั่งให้โปรแกรมทำงานใหม่เสมอ โดยปกติในโปรแกรม DEBUG ค่าของรีจิสเตอร์ CS จะมีค่าเท่ากันกับเซกเมนต์รีจิสเตอร์ตัวอื่น ๆ (DS ES และ SS) และมักนิยมป้อนหรือเขียนโปรแกรมที่จะทดสอบลงใน ณ ตำแหน่ง CS:100h เป็นตำแหน่งเริ่มต้นในการเขียนโปรแกรม

การใช้คำสั่งคำนวณทางคณิตศาสตร์และการแสดงสถานะของผลลัพธ์ของการคำนวณนั้นในแฟล็ก สถานะที่เก็บในแฟล็กจะใช้สำหรับการจัดการกับผลการคำนวณนั้น ๆ รวมถึงใช้ในคำสั่งเกี่ยวกับการกระโดดแบบมีเงื่อนไขด้วย

6.4 แฟล็ก (Flags)

แฟล็กเปรียบเสมือนรีจิสเตอร์ตัวหนึ่ง แต่แทนที่จะใช้เก็บค่าต่าง ๆ แฟล็กจะเก็บสถานะของการคำนวณทางคณิตศาสตร์ที่ผ่านมา ตัวอย่างของสถานะของการคำนวณ เช่น การมีบิตทด มีการเก็บค่าล้นหลักหรือผลลัพธ์มีค่าเป็นศูนย์เป็นต้น ใน 8086 แฟล็กจะมีขนาด 16 บิต โดยในแต่ละบิตจะเก็บค่าของสถานะการคำนวณแบบหนึ่ง ๆ แสดงได้ดังภาพที่ 6.4



ภาพที่ 6.2 แสดงแฟล็กต่าง ๆ

ค่าในบิตของแฟล็กนั้น ๆ จะมีค่าเป็น 1 เมื่อสถานะนั้นเป็นจริง จะเรียกสถานะที่แฟล็กเป็น 1 ว่า แฟล็กเซ็ต (flag set) และถ้าแฟล็กมีค่าเป็นศูนย์ (0) เรียกว่าแฟล็กเคลียร์ (flag cleared) โดยทั่วไปแล้วคำสั่งที่จะมีผลกับแฟล็กจะเป็นคำสั่งเกี่ยวกับการคำนวณทางคณิตศาสตร์ ส่วนคำสั่งในกลุ่มของการโอนย้ายข้อมูล เช่น คำสั่ง MOV จะไม่เปลี่ยนแปลงค่าในแฟล็ก

ความหมายของแฟล็ก แต่ละบิตเป็นดังต่อไปนี้

1. แฟล็กศูนย์ (Zero flag)

แฟล็กศูนย์จะมีค่าเป็นหนึ่ง (flag set) เมื่อผลการคำนวณมีค่าเท่ากับศูนย์

ตัวอย่างคำสั่ง

MOV	AX,100h	Z-flag = ?	
MOV	BX,80h	Z-flag = ?	
SUB	AX,BX	Z-flag = 0	{AX=80h}
SUB	AX,BX	Z-flag = 1	{AX=0h}
MOV	CX,80h	Z-flag = 1	
ADD	AX,CX	Z-flag = 0	{AX=80h}
SUB	AX,BX	Z-flag = 1	{AX=0h}

2. พาริตี้แฟล็ก (Parity flag)

พาริตี้แฟล็กจะมีค่าเป็นหนึ่งเมื่อในผลลัพธ์มีจำนวนบิตที่มีค่าเป็น 1 เป็นจำนวนคู่

ตัวอย่างคำสั่ง

MOV	AL,34h	P-flag = ?	
MOV	BL,11h	P-flag = ?	
ADD	AL,BL	P-flag = 0	{AL=45h[0100 0101b]}
ADD	AL,6	P-flag = 1	{AL=4Bh[0100 1011b]}
SUB	AL,BL	P-flag = 1	{AL=3Ah[0011 1010b]}
MOV	CL,10h	p-flag = 1	
ADD	AL,CL	p-flag = 0	{AL=4Ah[0100 1010b]}

3. แฟล็กเครื่องหมาย

แฟล็กเครื่องหมายจะเซ็ตเมื่อผลลัพธ์มีค่าเป็นลบ (เมื่อคิดว่าผลลัพธ์นั้นเก็บตัวเลขแบบคิดเครื่องหมายแบบ 2'Complement)

ตัวอย่างคำสั่ง

MOV	AL,50h	S-flag = ?	
ADD	AL,50h	S-flag = 1	{AL=0A0h[1010 0000b]}
ADD	AL,0A0h	S-flag = 0	{AL=40h[0100 0000b]}
SUB	AL,10h	S-flag = 0	{AL=30h[0011 0000b]}
ADD	AL,0B0h	S-flag = 1	{AL=0E0h[1110 0000b]}

4. แฟล็กทด (Carry flag)

แฟล็กทดจะเซ็ตเมื่อการคำนวณมีการทดหรือมีการยืม โดยพิจารณาค่าของข้อมูลแบบ*ไม่คิดเครื่องหมาย* แฟล็กทดนี้ยังใช้ในการเก็บบิตข้อมูลในคำสั่งเกี่ยวกับการเลื่อนบิตด้วย

ตัวอย่างคำสั่ง

MOV	AL,60h	C–flag = ?	
ADD	AL,60h	C–flag = 0	{AL=0C0h}
ADD	AL,60h	C–flag = 1	{AL=20h,0C0+60=120h}
SUB	AL,15h	C–flag = 0	{AL=0Ah}
SUB	AL,15h	C–flag = 1	{AL=F5h,0Ah–15h=0F5h}

5. โอเวอร์โฟล์แฟล็ก (Overflow flag)

ในการพิจารณาโอเวอร์โฟล์แฟล็กพิจารณาค่าของข้อมูลเป็นแบบคิดเครื่องหมาย โดยโอเวอร์โฟล์แฟล็กจะมีค่าเป็นหนึ่งเมื่อผลลัพธ์มีความผิดพลาด เช่น การบวกค่าที่มากกว่าขอบเขตทำให้ผลลัพธ์ที่ได้ มีเครื่องหมายที่ผิดเป็นต้น

ตัวอย่างคำสั่ง

MOV	AL,0E3h	O–flag = ?	
ADD	AL,79h	O–flag = 0	{AL=5Ch,–29+121 = 92[5Ch]}
ADD	AL,60h	O–flag = 1	{AL=0BCh,0BCh=–68 <> 92+96=188}
SUB	AL,20h	O–flag = 0	{AL=9Ch,9Ch=–100 = –68–32}
SUB	AL,20h	O–flag = 1	{AL=7Ch,7Ch=124 <> –100–32=–132
ADD	AL,9Ah	O–flag = 0	{AL=16h,16h=22 = 124–102

6. แฟล็กเสริม (Auxiliary Flag)

แฟล็กเสริมจะเป็นแฟล็กที่ใช้ในการปรับค่าของการคำนวณเลขแบบ BCD

7. แฟล็กทิศทาง (Direction Flag)

แฟล็กทิศทางเป็นแฟล็กที่ใช้ในการระบุทิศทางของการปรับค่ารีจิสเตอร์ดัชนีในการประมวลผลคำสั่งเกี่ยวกับสายข้อมูล

8. แทรปแฟล็ก (Trap Flag)

แทรปแฟล็กเป็นแฟล็กที่ใช้ระบุให้หน่วยประมวลผลสร้างสัญญาณขัดจังหวะเมื่อประมวลผลคำสั่งเสร็จสิ้นหนึ่งคำสั่ง โดยแฟล็กนี้จะใช้ในการตรวจสอบการทำงานของโปรแกรม

9. อินเตอร์รัพท์แฟล็ก (Interrupt Flag)

แฟล็กนี้ใช้ระบุว่าหน่วยประมวลผลจะตอบสนองการขัดจังหวะจากอุปกรณ์ฮาร์ดแวร์หรือไม่

คำสั่งเกี่ยวกับการกำหนดค่าของแฟล็ก

สามารถกำหนดค่าของแฟล็กทด แฟล็กทิศทาง และอินเตอร์รัพท์แฟล็กได้โดยใช้คำสั่งต่อไปนี้

ตารางที่ 6.1 คำสั่งสำหรับการกำหนดค่าแฟล็ก

แฟล็ก (FLAG)	คำสั่งในการเซต (SET)	คำสั่งในการเคลียร์ (CLEAR)
carry flag	STC	CLC
direction flag	STD	CLD
interrupt flag	STI	CLI

6.5 คำสั่งทางคณิตศาสตร์

คำสั่งทางคณิตศาสตร์ใน 8086 ที่เราจะศึกษาในวิชานี้แบ่งได้เป็นกลุ่มใหญ่ ๆ 3 กลุ่มดังนี้

6.5.1 กลุ่มคำสั่งบวกและลบ

คำสั่งเพิ่มและลดค่า : INC [Increment] และ DEC [Decrement]

คำสั่ง INC จะเพิ่มค่าของโอเปอร์แรนด์ขึ้นหนึ่ง ส่วนคำสั่ง DEC จะลดค่าของโอเปอร์แรนด์ลงหนึ่ง คำสั่งนี้มีผลกระทบกับแฟล็กทั้งหมด***ยกเว้นแฟล็กทด***

รูปแบบของทั่วไปของคำสั่ง INC และ DEC เป็นดังนี้

INC	*register*	DEC	*register*
INC	*memory*	DEC	*memory*

ตัวอย่างคำสั่ง

```
MOV   AL,10h
INC   AL                    รีจิสเตอร์ AL = 11h
MOV   BX,200h
MOV   WORD PTR [BX],10h
DEC   WORD PTR [BX]         ข้อมูลแบบเวิร์ด ณ ตำแหน่ง [DS:BX] = 0Fh
```

คำสั่งบวกและบวกรวมตัวทด : ADD [Addition] และ ADC [Add with carry]

คำสั่งบวกจะนำค่าของตัวถูกดำเนินการ (Operand) ตัวที่สองมาบวกกับค่าของโอเปอร์แรนด์ตัวที่หนึ่งแล้วนำค่าที่ได้เก็บในโอเปอร์แรนด์อีกตัวที่หนึ่ง โดยที่คำสั่ง ADD และ ADC มีผลกระทบกับแฟล็กทางคณิตศาสตร์ทุกแฟล็ก เรานิยมใช้คำสั่ง ADC ในการบวกข้อมูลที่ต้องนำบิตที่ทดจากการบวกครั้งก่อนมารวมด้วย เช่น การบวกข้อมูลที่เก็บอยู่ในหลายรีจิสเตอร์ที่อยู่ต่อเนื่องกันเป็นต้น (ศัพท์บัญญัติ ราชบัณฑิตยสถาน, 2544)

รูปแบบของทั่วไปของคำสั่ง ADD และ ADC เป็นดังนี้

ADD	*register,number*	ADC	*register,number*
ADD	*memory,number*	ADC	*memory,number*
ADD	*register,register*	ADC	*register,register*
ADD	*register,memory*	ADC	*register,memory*
ADD	*memory,register*	ADC	*memory,register*

ตัวอย่างคำสั่ง

MOV	AL,10h	รีจิสเตอร์ AL = 30h
ADD	AL,20h	
MOV	BX,200h	ข้อมูลแบบเวิร์ด ณ ตำแหน่ง [DS:BX] = 80h
MOV	WORD PTR [BX],10h	ทำการบวก 1234 5678h เข้ากับ 8765
ADD	WORD PTR [BX],70h	4321h โดยผลลัพธ์ที่ได้ 16 บิตบนจะเก็บที่
MOV	AX,5678h	รีจิสเตอร์ DX ส่วน 16 บิตล่างจะเก็บที่
MOV	DX,1234h	รีจิสเตอร์ AX และใช้คำสั่ง ADC ในการบวก
ADD	AX,4321h	ครั้งที่สองเพื่อนำตัวทดจากการบวกครั้งแรก
ADC	DX,8765h	มารวมด้วย

คำสั่งลบและลบรวมตัวยืม : SUB [Substraction] และ SBB [Subtract with borrow]

คำสั่ง SUB และ SBB จะทำงานคล้ายกับคำสั่ง ADD และ ADC เพียงแต่เป็นการนำค่าในโอเปอร์แรนด์ตัวที่สองไปลบออกจากโอเปอร์แรนด์ตัวที่หนึ่ง ลักษณะของการใช้งานคำสั่ง SBB จะคล้ายคลึงกับการใช้คำสั่ง ADC นั่นคือจะนิยมใช้ในกรณีที่มีการลบเลขที่เก็บอยู่ในรีจิสเตอร์หลายตัวต่อเนื่องกัน

รูปแบบของคำสั่ง SUB และ SBB จะมีลักษณะเช่นเดียวกับคำสั่ง ADD และคำสั่ง ADC โดยมีรูปแบบทั้งหมดดังนี้

SUB	*register,number*	SBB	*register,number*
SUB	*memory,number*	SBB	*memory,number*
SUB	*register,register*	SBB	*register,register*
SUB	*register,memory*	SBB	*register,memory*
SUB	*memory,register*	SBB	*memory,register*

ตัวอย่างคำสั่ง

```
MOV     CX,10h              รีจิสเตอร์ CX จะถูกลดค่าลงเท่ากับข้อมูลแบบ
SUB     CX,[200h]           เวิร์ดในแอดเดรส [DS:200h].
MOV     BX,202h
MOV     DX,345h
SUB     WORD PTR [BX],DX    ข้อมูลแบบเวิร์ด ณ ตำแหน่ง [DS:BX] มีค่าลดลง
MOV     AX,0AAFFh           เป็น 345h
MOV     DX,0BBCCh
SUB     BX,AX               ทำการลบค่าของข้อมูลขนาด 32 บิตที่เก็บที่ CX
SBB     CX,DX               ด้วยค่าขนาด 32 บิตในรีจิสเตอร์ DX และ AX
```

คำสั่งเปรียบเทียบ : CMP [Compare]

คำสั่ง CMP จะมีการทำงานเหมือนกับคำสั่ง SUB ทุกประการ แต่จะไม่มีการเปลี่ยนค่าในโอเปอร์แรนด์ใด ๆ โดยผลลัพธ์ที่แท้จริงของคำสั่งนี้คือการเปลี่ยนค่าในแฟล็กต่าง ๆ เพื่อแสดงผลลัพธ์ของการลบ ซึ่งจะใช้คำสั่งนี้ในการเปรียบเทียบค่าต่าง ๆ และนำผลที่ได้ในแฟล็กไปใช้ในการกำหนดเงื่อนไขของการกระโดด

รูปแบบของคำสั่ง CMP จะเหมือนคำสั่ง SUB โดยมีรูปแบบทั่วไปเป็นดังนี้

```
CMP     register,number
CMP     memory,number
CMP     register,register
CMP     register,memory
CMP     memory,register
```

ตัวอย่างคำสั่ง

```
MOV     CX,10h              ค่าในริจีสเตอร์ CX จะคงค่าเดิม แต่ปรับค่าแฟล็กใหม่
CMP     CX,20h              โดย S-flag=1, C-flag=1, O-flag=0
MOV     BX,40h
CMP     BX,40h              Z-flag=1
MOV     AX,30h
CMP     AX,20h              S-flag=0, C-flag=0, O-flag=0.
```

คำสั่งเปลี่ยนเครื่องหมาย : NEG [Negation]

คำสั่งเปลี่ยนเครื่องหมายจะเปลี่ยนค่าในโอเปอร์แรนด์ซึ่งจะพิจารณาเป็นตัวเลขแบบคิดเครื่องหมายเป็นค่าลบของค่านั้น โดยการเปลี่ยนค่านั้นจะเปลี่ยนแบบ 2's complement ผลจากคำสั่งนี้จะทำให้***แฟล็กทดมีค่าเป็น 1 เสมอ***

รูปแบบของคำสั่ง NEG

NEG *register*

NEG *memory*

ตัวอย่างคำสั่ง

```
MOV     CX,10h
NEG     CX              CX = 0FFF0h
MOV     AX,0FFFFh
NEG     AX              AX = 1
MOV     BX,1h
NEG     BX              BX = 0FFFFh
```

6.5.2 กลุ่มคำสั่งคูณและหาร

คำสั่งคูณแบบคิดเครื่องหมายและไม่คิดเครื่องหมาย : IMUL [Integer multiplication] และ MUL [Multiplication]

การคูณใน 8086 นั้นค่าของตัวตั้งของการคูณ และค่าผลลัพธ์ของการคูณนั้นจะต้องเก็บในรีจิสเตอร์ที่กำหนดไว้ โดยขึ้นกับขนาดของการคูณ รีจิสเตอร์ที่กำหนดไว้เป็นดังนี้

การคูณข้อมูล 8บิต :ตัวตั้ง ALผลลัพธ์ AX

การคูณข้อมูล 16 บิต :ตัวตั้ง AX ผลลัพธ์ DX, AX [16 บิตบนที่ DX 16 บิตล่างที่ AX] โดยจะระบุตัวคูณและขนาดของการคูณที่โอเปอร์แรนด์ของคำสั่ง IMUL หรือ MUL ทั้งสองคำสั่งมีรูปแบบดังนี้

IMUL *register* MUL *register*

IMUL *memory* MUL *memory*

ตัวอย่างคำสั่ง

```
MOV     AL,17h          ทำการคูณ 17h ด้วย 13h แบบ 8 บิต
MOV     CL,13h
IMUL    CL              ผลลัพธ์จะมีขนาด 16 บิต เก็บในรีจิสเตอร์ AX
MOV     BX,1234h        นำผลลัพธ์ของ 17h คูณกับ 13h มาคูณด้วย 1234h
IMUL    BX              ค่าใน DX,AX มีค่าเท่ากับ (17h x 13h) x 1234h
```

คำสั่ง MUL และ IMUL จะมีผลกระทบกับแฟล็กทดและโอเวอร์โฟล์แฟล็กเท่านั้น

คำสั่งหารแบบคิดเครื่องหมายและไม่คิดเครื่องหมาย : IDIV [Integer division] และ DIV [Division]

เช่นเดียวกับคำสั่งคูณ ตัวตั้งและผลลัพธ์สำหรับการประมวลผลคำสั่งหารใน 8086 จะต้องเก็บรีจิสเตอร์ซึ่งกำหนดไว้ โดยจะขึ้นกับขนาดของการหารตัวเลขเช่นเดียวกัน

การหารข้อมูล 8 บิต:ตัวตั้ง AX	ผลลัพธ์ AL	เศษ AH
การหารข้อมูล 16 บิต:ตัวตั้ง DX, AX	ผลลัพธ์ AX	เศษ DX

คำสั่ง IDIV และ DIV มีรูปแบบดังนี้

IDIV	*register*	DIV	*register*
IDIV	*memory*	DIV	*memory*

ตัวอย่างคำสั่ง

MOV	AX,4022h	หาร 4022h ด้วย 1000h โดยการหารดังกล่าวเป็นการหาร
MOV	DX,0000h	แบบ 16 บิต จึงต้องกำหนดค่าใน DX,AX
MOV	CX,1000h	
DIV	CX	ผลลัพธ์จากการหารจะมีขนาด 16 บิต เก็บในรีจิสเตอร์ AX โดยเศษจะเก็บใน DX. AX = 4 และ DX = 22h
MOV	BL,3h	นำผลลัพธ์มาหารด้วย 3h
DIV	BL	ค่าใน AL จะเก็บผลหาร มีค่าเท่ากับ 1 และ AH จะเก็บเศษ โดยมีค่าเท่ากับ 1 เช่นกัน

คำสั่ง DIV และ IDIV จะไม่มีผลกระทบกับแฟล็กใด ๆ แต่ถ้าในการหารมีการหารด้วยศูนย์(0) หรือเกิดการหารที่ไม่สามารถเก็บผลลัพธ์ลงในรีจิสเตอร์ที่ต้องการได้ เช่น การหาร 1234 5678h ด้วย 2h หน่วยประมวลผลจะสร้างสัญญาณในการขัดจังหวะขึ้นเพื่อแจ้งกับโปรแกรมที่จัดการระบบต่อไป การเกิดการขัดจังหวะในลักษณะนี้เรียกว่าเกิด **Exception**

6.6 กลุ่มคำสั่งแปลงขนาดตัวเลข

จากข้อจำกัดของการใช้คำสั่งหารที่ตัวตั้งจะต้องมีขนาดมากกว่าตัวหาร ทำให้ในบางครั้งจะต้องกำหนดการหารข้อมูลให้มีขนาดเท่ากัน โดยจะต้องแปลงตัวตั้งให้มีขนาดที่เหมาะสมเสียก่อน สำหรับการแปลงขนาดของเลขไม่คิดเครื่องหมายนั้น เราสามารถกระทำได้โดยกำหนดให้ข้อมูลนัยสำคัญสูงที่ขยายเพิ่มมานั้นมีค่าเป็นศูนย์ ตัวอย่างเช่น การขยาย AL ที่เป็นตัวเลขแบบไม่คิดเครื่องหมายให้เป็นข้อมูล 16 บิตใน AX สามารถกระทำได้โดยกำหนดค่าศูนย์ (0) ให้กับ AH แต่ในกรณีของตัวเลขแบบคิดเครื่องหมายนั้นถ้าตัวเลขมีค่าเป็นลบการกำหนดค่าศูนย์ให้กับข้อมูลนัยสำคัญสูงที่ขยายเพิ่มมานั้น จะทำให้ค่าของเลขที่ได้มีความผิดพลาดได้ ในการขยายขนาดของเลขคิดเครื่องหมายเราจึงต้องใช้คำสั่งที่เหมาะสม

คำสั่งแปลงจากไบต์เป็นเวิร์ด : CBW [Convert byte to word]

คำสั่งนี้จะแปลงเลขแบบคิดเครื่องหมายขนาด 8 บิตใน AL เป็นเลขคิดเครื่องหมายขนาด 16 บิตใน AX

รูปแบบของคำสั่ง

CBW

ตัวอย่างคำสั่ง

MOV	AL,22h	AL = 22h
CBW		หลังจากแปลงแล้วค่า AX=0022h
MOV	AL,F0h	AL = F0h = –16
CBW		หลังจากแปลงแล้วค่า AX=FFF0h = –16

คำสั่งแปลงจากเวิร์ดเป็นดับเบิลเวิร์ด : CWD [Convert word to doubleword]

คำสั่งนี้จะแปลงเลขแบบคิดเครื่องหมายขนาด 16 บิตใน AX เป็นเลขคิดเครื่องหมายขนาด 32 บิตใน DX,AX

รูปแบบของคำสั่ง

CWD

ตัวอย่างคำสั่ง

MOV	AX,3422h	AX = 3422h
CWD		หลังจากแปลงแล้วค่า DX = 0000h , AX=3422h
MOV	AX,FFF0h	AX = FFF0h = –16
CWD		หลังจากแปลงแล้วค่าา DX = FFFFh, AX=FFF0h

ตัวอย่างการใช้คำสั่งทางคณิตศาสตร์

ตัวอย่างที่ 1 คำนวณค่า $AL^2 + BL^2$ คิดตัวเลขแบบไม่คิดเครื่องหมายโดยให้ผลลัพธ์เป็นเลข 16 บิต เก็บที่รีจิสเตอร์ AX

MUL	AL	หาค่า AL*AL เสียก่อน
MOV	CX, AX	
MOV	AL, BL	นำค่าไปเก็บไว้ใน CX เพราะรีจิสเตอร์ AL ต้องใช้ในการ
MUL	BL	คำนวณ BL*BL
ADD	AX, CX	ได้ผลลัพธ์แล้วนำค่า AL*AL เก็บใน CX มาบวกกัน

ตัวอย่างที่ 2 คำนวณค่า (CL*BL+DX)/SI โดยคิดเป็นเลขคิดเครื่องหมาย

```
MOV   AL,CL     หาค่าของ CL*BL โดยผลลัพธ์เก็บในรีจีสเตอร์ AX
MUL   BL
ADD   AX,DX     นำค่าใน DX มาบวกเข้ากับค่าใน AX
CWD             ขยายข้อมูลใน AX เป็น 32 บิตใน DX,AX โดยนำ SI มาหาร
IDIV  SI        ผลลัพธ์จะเก็บไว้ใน AX เศษของการหารเก็บไว้ใน DX
```

ตัวอย่างที่ 3 คำนวณค่า (DX+AX*BX)/(DI-CX) โดยคิดเป็นเลขคิดเครื่องหมาย

```
MOV   SI,DX     นำค่าของ DX ไปเก็บที่ SI เสียก่อนเนื่องจาก DX จะถูก
MUL   BX        การคำนวณ AX*BX. เมื่อคำนวณ AX*BX แล้วนำ SI มา
ADD   AX,SI     โดยต้องนำตัวทดของการบวก SI กับ AX มาทดยัง DX
ADC   DX,0
SUB   DI,CX     ลบ DI ด้วย CX
IDIV  DI        นำไปหารด้วย DI
```

ผลกระทบของคำสั่งทางคณิตศาสตร์ต่อแฟล็ก

ตารางที่ 6.2 ผลกระทบของคำสั่งต่าง ๆ ต่อแฟล็ก

Instruction	Flag affected				
	Z-flag	C-flag	S-flag	O-flag	A-flag
ADD	*yes*	*yes*	*yes*	*yes*	*yes*
ADC	*yes*	*yes*	*yes*	*yes*	*yes*
SUB	*yes*	*yes*	*yes*	*yes*	*yes*
SBB	*yes*	*yes*	*yes*	*yes*	*yes*
INC	*yes*	*no*	*yes*	*yes*	*yes*
DEC	*yes*	*no*	*yes*	*yes*	*yes*
NEG	*yes*	*yes*	*yes*	*yes*	*yes*
CMP	*yes*	*yes*	*yes*	*yes*	*yes*
MUL	*no*	*yes*	*no*	*yes*	*no*
IMUL	*no*	*yes*	*no*	*yes*	*no*
DIV	*no*	*no*	*no*	*no*	*no*
IDIV	*no*	*no*	*no*	*no*	*no*
CBW	*no*	*no*	*no*	*no*	*no*
CWD	*No*	*no*	*no*	*no*	*no*

การเปลี่ยนแปลงของแฟล็กในการทำงานของคำสั่งต่าง ๆ

ตัวอย่างที่ 1

Instruction		Z-flag	C-flag	O-flag	S-flag	P-flag	หมายเหตุ
MOV	AX,7100h	?	?	?	?	?	
MOV	BX,4000h	?	?	?	?	?	
ADD	AX,BX	0	0	1	1	1	AX=0B100h
ADD	AX,7700h	0	1	0	0	1	AX=2800h
SUB	AX,2000h	0	0	0	0	0	AX=0800h
SUB	AX,1000h	0	1	0	1	0	AX=F800h
ADD	AX,0800h	1	1	0	0	1	AX=0000h

ตัวอย่างที่ 2

Instruction		Z-flag	C-flag	O-flag	S-flag	P-flag	หมายเหตุ
MOV	AL,10	?	?	?	?	?	
ADD	AL,F0h	0	0	0	1	1	AL=0FAh
ADD	AL,6	1	1	0	0	1	AL=0
SUB	AL,5	0	1	0	1	0	AL=0FBh
INC	AL	0	0	0	1	1	AL=0FCh
ADD	AL,10	0	1	0	0	1	AL=6h
ADD	AL,FBh	0	1	0	0	0	AL=1h
DEC	AL	1	0	0	0	1	AL=0h
DEC	AL	0	0	0	1	1	AL=0FFh
INC	AL	1	0	0	0	1	AL=0

หมายเหตุ คำสั่ง DEC และ INC ไม่กระทบแฟล็กทด

ตัวอย่างที่ 3

Instruction		Z-flag	C-flag	O-flag	S-flag	P-flag	หมายเหตุ
MOV	AL,120	?	?	?	?	?	
ADD	AL,15	0	0	1	1	1	AL=87h=-121
NEG	AL	0	1	0	0	0	AL=79h
SUB	AL,130	0	1	1	1	0	AL=0F7h

หมายเหตุ คำสั่ง NEG ทำให้แฟล็กทดมีค่าเป็น 1 เสมอ

สรุป

คำสั่งภาษาแอสแซมบลีเบื้องต้น มีคำสั่งเกี่ยวกับการโอนย้ายข้อมูล อยู่ด้วยกัน 2 แบบ คือ

1. การโอนย้ายข้อมูลระหว่างรีจิสเตอร์กับรีจิสเตอร์ เป็นการโอนย้ายข้อมูลระหว่างรีจิสเตอร์สามารถทำได้ถ้าขนาดของรีจิสเตอร์ของทั้งคู่เท่ากัน แต่ยังมีเซกเมนต์รีจิสเตอร์บางตัวซึ่งไม่ควรเข้าไปกำหนดค่าโดยตรง เช่น CS หรือ SS

2. การโอนย้ายข้อมูลกับหน่วยความจำ โดยปกติการโอนย้ายข้อมูลกับหน่วยความจำนั้น เราจะระบุเฉพาะออฟเซ็ตของหน่วยความจำที่เราต้องการจะโอนย้ายข้อมูลด้วย ออฟเซ็ตนั้นจะถูกนำมาประกอบกับเซกเมนต์รีจิสเตอร์ DS เพื่อเป็นตำแหน่งในหน่วยความจำที่แท้จริงที่จะอ่านเขียนข้อมูลด้วย

ดีบัก (Debug) คือ โปรแกรมที่พัฒนาเพื่อแก้ไขปัญหาพื้นฐานในระบบปฏิบัติการดอส (DOS = Disk Operation System) เป็นโปรแกรมสำหรับแก้ไขแฟ้มอย่างง่าย เป็นคำสั่งภายนอก (External Command) ซึ่งเป็นที่นิยมใช้งานในกลุ่มนักพัฒนาโปรแกรมคอมพิวเตอร์มาตั้งแต่ยุคระบบปฏิบัติการดอส (DOS) เครื่องมือในการทดลองการโปรแกรมภาษาแอสแซมบลีก็คือโปรแกรม DEBUG เป็นโปรแกรมสารพัดประโยชน์สำหรับการทดลองเกี่ยวกับการเขียนโปรแกรมภาษาแอสแซมบลี โปรแกรมนี้มีอยู่ทั้งในระบบปฏิบัติการ DOS และ Windows

แฟล็ก (Flags) เป็นรีจิสเตอร์ขนาด 16 บิตที่ใช้บ่งบอกผลลัพธ์ที่ได้จากการคำนวณ คำสั่งที่เกี่ยวข้องกับการคำนวณจะส่งผลกระทบให้แฟล็กเปลี่ยนแปลงค่าไป การคัดลอกข้อมูลจะไม่ทำให้แฟล็กเปลี่ยนแปลง ค่าแฟล็กเปรียบเสมือนรีจิสเตอร์ตัวหนึ่ง แต่แทนที่จะใช้เก็บค่าต่าง ๆ แฟล็กจะเก็บสถานะของการคำนวณทางคณิตศาสตร์ที่ผ่านมา

คำถามทบทวน

1. จงให้คำจำกัดความของคำว่าตัวถูกดำเนินการ (operand)
2. โปรแกรม DEBUG คืออะไรและมีหน้าที่อย่างไรในภาษาโปรแกรมภาษาแอสแซมบลี
3. รีจีสเตอร์ AX มีหน้าที่อย่างไรในคำสั่ง MOV
4. คำสั่งต่อไปนี้ทำงานอย่างไร
   - คำสั่ง G (go)
   - คำสั่ง D (dump)
   - คำสั่ง R (register)
5. คำสั่ง A (assemble) มีหน้าที่และทำงานอย่างไร
6. คำสั่งภาษาแอสแซมบลีที่ใช้สำหรับการโอนย้ายข้อมูลมีกี่แบบอะไรบ้าง
7. จงอธิบายความหมายและหน้าที่ของแฟล็ก (Flags) ต่าง ๆ ว่ามีหน้าที่อย่างไร
8. จงอธิบายความแตกต่างระหว่างคำสั่งลบ SUB และลบรวมตัวยืม SBB
9. จงอธิบายความแตกต่างระหว่างคำสั่งคูณแบบคิดเครื่องหมาย IMUL และไม่คิดเครื่องหมาย MUL
10. จงอธิบายความหมายของคำสั่งเปรียบเทียบ CMP และคำสั่งแปลงจากไบต์เป็นเวิร์ด CBW
11. จากตัวอย่างคำสั่งข้างล่างจงอธิบายความหมายของคำสั่งแต่ละบรรทัด

    ตัวอย่างคำสั่ง

```
MOV   AL,25h
MOV   CL,13h
IMUL  CL
MOV   BX,3456h
IMUL  BX
```

12. จากตัวอย่างคำสั่งข้างล่างจงอธิบายความหมายของคำสั่งแต่ละบรรทัด

    ตัวอย่างคำสั่ง

```
MOV   AX,2513h
MOV   DX,0000h
MOV   CX,1000H
DIV   CX
MOV   BL,3h
DIV   BL
```

เอกสารอ้างอิง

ราชบัณฑิตยสถาน. (2544). *ศัพท์บัญญัติ ราชบัณฑิตยสถาน.* ค้นเมื่อ 30 ธันวาคม 2556,จาก :http://rirs3.royin.go.th/coinages/

โปรแกรมดีบัก. (2556). *วิกิพีเดีย สารานุกรมเสรี.* ค้นเมื่อ 30 ธันวาคม 2556, จาก: http://th.wikipedia.org/wiki/

แฟล็ก. (2556). *วิกิพีเดีย สารานุกรมเสรี.* ค้นเมื่อ 30 ธันวาคม 2556, จาก: http://th.wikipe.org/wiki/

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

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