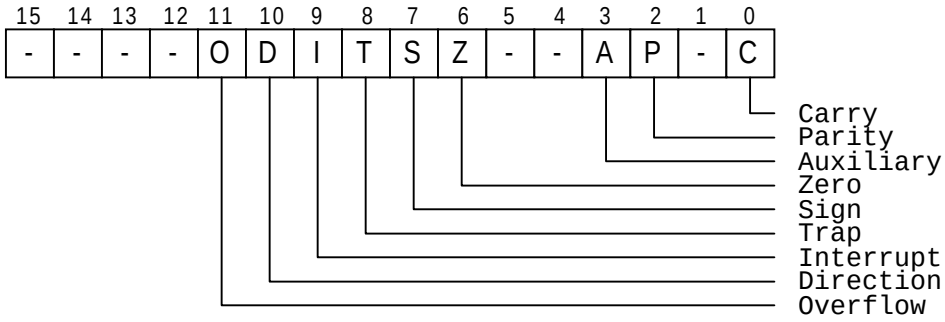
บทที่ 6 : แฟล็กและคำสั่งคณิตศาสตร์

ในบทนี้เราจะศึกษาเกี่ยวกับการใช้คำสั่งคำนวณทางคณิตศาสตร์และการแสดงสถานะของผลลัพธ์ของการคำนวณนั้นในแฟล็ก. สถานะที่เก็บในแฟล็กจะใช้สำหรับการจัดการกับผลการคำนวณนั้น ๆ รวมถึงใช้ในคำสั่งเกี่ยวกับการกระโดดแบบมีเงื่อนไขด้วย.

แฟล็ก (Flags)

แฟล็กเปรียบเสมือนรีจิสเตอร์ตัวหนึ่ง แต่แทนที่จะใช้เก็บค่าต่าง ๆ แฟล็กจะเก็บสถานะของการคำนวณทางคณิตศาสตร์ที่ผ่านมา. ตัวอย่างของสถานะของการคำนวณ เช่น การมีบิตทด มีการเก็บค่าล้นหลัก หรือผลลัพธ์มีค่าเป็นศูนย์เป็นต้น.

ใน 8086 แฟล็กจะมีขนาด 16 บิต โดยในแต่ละบิตจะเก็บค่าของสถานะการคำนวณแบบหนึ่ง ๆ ดังรูปที่ 6.1.



รูปที่ 6.1 แสดงแฟล็กต่าง ๆ

ค่าในบิตของแฟล็กนั้น ๆ จะมีค่าเป็น 1 เมื่อสถานะนั้นเป็นจริง. เราจะเรียกสถานะที่แฟล็กเป็น 1 ว่า แฟล็กเซ็ต (flag set) และถ้าแฟล็กมีค่าเป็นศูนย์เราจะเรียกว่าแฟล็กเคลียร์ (flag cleared.) โดยทั่วไปแล้วคำสั่งที่จะมีผลกับแฟล็กจะเป็นคำสั่งเกี่ยวกับการคำนวณทางคณิตศาสตร์. ส่วนคำสั่งในกลุ่มของการโอนย้ายข้อมูล เช่น คำสั่ง MOV จะไม่เปลี่ยนแปลงค่าในแฟล็ก.

ความหมายของแฟล็กแต่ละบิตเป็นดังต่อไปนี้.

1. แฟล็กศูนย์ (Zero flag)

แฟล็กศูนย์จะมีค่าเป็นหนึ่ง (flag set) เมื่อผลการคำนวณมีค่าเท่ากับศูนย์.

<u>ตัวอย่างคำสั่ง</u>

```
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.

<u>ตัวอย่างคำสั่ง</u>

```
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 (เมื่อคิดว่าผลลัพธ์นั้นเก็บตัวเลขแบบ*คิดเครื่องหมาย* 2'Complement.)

<u>ตัวอย่างคำสั่ง</u>

```
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. ในการพิจารณาเราจะพิจารณาค่าของข้อมูลแบบ*ไม่คิดเครื่องหมาย*. แฟล็กทดนี้ยังใช้ในการเก็บบิตข้อมูลในคำสั่งเกี่ยวกับการเลื่อนบิตด้วย.

<u>ตัวอย่างคำสั่ง</u>

```
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)

ในการพิจารณาโอเวอร์โฟล์แฟล็กเราจะพิจารณาค่าของข้อมูลเป็นแบบคิดเครื่องหมาย โดยโอเวอร์โฟล์แฟล็กจะมีค่าเป็นหนึ่งเมื่อผลลัพธ์มีความผิดพลาด เช่น การบวกค่าที่มากกว่าขอบเขตทำให้ผลลัพธ์ที่ได้ มีเครื่องหมายที่ผิดเป็นต้น.

<u>ตัวอย่างคำสั่ง</u>

```
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

เราสามารถกำหนดค่าของแฟล็กทด แฟล็กทิศทาง และอินเตอร์รัพท์แฟล็กได้โดยใช้คำสั่งต่อไปนี้.

แฟล็ก	คำสั่งในการเซต	คำสั่งในการเคลียร์
carry flag	STC	CLC
direction flag	STD	CLD
interrupt flag	STI	CLI

ตารางที่ 6.1 คำสั่งสำหรับการกำหนดค่าแฟล็ก.

คำสั่งทางคณิตศาสตร์

คำสั่งทางคณิตศาสตร์ใน 8086 ที่เราจะศึกษาในวิชานี้แบ่งได้เป็นกลุ่มใหญ่ ๆ 3 กลุ่มดังนี้.

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
```

<u>ตัวอย่างคำสั่ง</u>

```
MOV   AL,10h
INC   AL                    รีจิสเตอร์ AL = 0Fh.
MOV   BX,200h
MOV   WORD PTR [BX],10h
DEC   WORD PTR [BX]         ข้อมูลแบบเวิร์ดที่ตำแหน่ง [DS:BX] = 9h.
```

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

คำสั่งบวกจะนำค่าของโอเปอร์แรนด์ตัวที่สองมาบวกกับค่าของโอเปอร์แรนด์ตัวที่หนึ่ง แล้วนำค่าที่ได้เก็บในโอเปอร์แรนด์ตัวที่หนึ่ง. คำสั่ง ADD และ ADC มีผลกระทบกับแฟล็กทางคณิตศาสตร์ทุกแฟล็ก. เรานิยมใช้คำสั่ง ADC ในการบวกข้อมูลที่ต้องนำบิตที่ทดจากการบวกครั้งก่อนมารวมด้วย เช่น การบวกข้อมูลที่เก็บอยู่ในหลายรีจิสเตอร์ต่อเนื่องกันเป็นต้น.

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
```

<u>ตัวอย่างคำสั่ง</u>

```
MOV   AL,10h
ADD   AL,20h                รีจิสเตอร์ AL = 30h.
MOV   BX,200h
MOV   WORD PTR [BX],10h
```

ADD	WORD PTR [BX],70h	*ข้อมูลแบบเวิร์ดที่ตำแหน่ง [DS:BX] = 80h.*
MOV	AX,5678h	*เราจะบวก 1234 5678h เข้ากับ 8765 4321h โดยผลลัพธ์ที่ได้ 16*
MOV	DX,1234h	*บิตบนจะเก็บที่รีจิสเตอร์ DX ส่วน 16 บิตล่างจะเก็บที่รีจิสเตอร์*
ADD	AX,4321h	*AX. สังเกตว่าเราจะใช้คำสั่ง ADC ในการบวกครั้งที่สองเพื่อนำ*
ADC	DX,8765h	*ตัวทดจากการบวกครั้งแรกมารวมด้วย*

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

คำสั่ง SUB และ SBB จะทำงานคล้ายกับคำสั่ง ADD และ ADC เพียงแต่เป็นการนำค่าใน โอเปอร์แรนด์ตัวที่สองไปลบออกจากโอเปอร์แรนด์ตัวที่หนึ่ง. 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
```

<u>ตัวอย่างคำสั่ง</u>

MOV	CX,10h	
SUB	CX,[200h]	*รีจิสเตอร์ CX จะถูกลดค่าลงเท่ากับข้อมูลแบบเวิร์ดใน แอด*
MOV	BX,202h	*เดรส [DS:200h].*
MOV	DX,345h	
SUB	WORD PTR [BX],DX	*ข้อมูลแบบเวิร์ดที่ตำแหน่ง [DS:BX] มีค่าลดลง 345h.*
MOV	AX,0AAFFh	*เราจะลบค่าของข้อมูลขนาด 32 บิตที่เก็บที่ CX BX ด้วยค่าขนาด*
MOV	DX,0BBCCh	*32 บิตในรีจิสเตอร์ DX และ AX*
SUB	BX,AX	
SBB	CX,DX	

❑ **คำสั่งเปรียบเทียบ : CMP** [Compare]

คำสั่ง CMP จะมีการทำงานเหมือนกับคำสั่ง SUB ทุกประการ แต่จะไม่มีการเปลี่ยนค่าในโอเปอร์แรนด์ใด ๆ โดยผลลัพธ์ที่แท้จริงของคำสั่งนี้คือการเปลี่ยนค่าในแฟล็กต่าง ๆ เพื่อแสดงผลลัพธ์ของการลบ. เราจะใช้คำสั่งนี้ในการเปรียบเทียบค่าต่าง ๆ และนำผลที่ได้ในแฟล็กไปใช้ในการกำหนดเงื่อนไขของการกระโดด.

รูปแบบของคำสั่ง CMP จะเหมือนคำสั่ง SUB โดยมีรูปแบบทั่วไปเป็นดังนี้.

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

<u>ตัวอย่างคำสั่ง</u>

MOV	CX,10h	*CX จะมีไม่มีการเปลี่ยนแปลง แต่จะมีการเปลี่ยนแปลงของแฟล็ก*
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
```

<u>ตัวอย่างคำสั่ง</u>

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

2. กลุ่มคำสั่งคูณและหาร

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

การคูณใน 8086 นั้นค่าของตัวตั้งของการคูณ และค่าผลลัพธ์ของการคูณนั้นจะต้องเก็บในรีจิสเตอร์ที่กำหนดไว้ โดยขึ้นกับขนาดของการคูณ. รีจิสเตอร์ที่กำหนดไว้เป็นดังนี้.

การคูณข้อมูล 8 บิต :	ตัวตั้ง AL	ผลลัพธ์ AX.
การคูณข้อมูล 16 บิต :	ตัวตั้ง AX	ผลลัพธ์ DX, AX [16 บิตบนที่ DX 16 บิตล่างที่ AX.]

เราจะระบุตัวคูณและขนาดของการคูณที่โอเปอร์แรนด์ของคำสั่ง IMUL หรือ MUL. ทั้งสองคำสั่งมีรูปแบบดังนี้.

```
IMUL  register        MUL   register
IMUL  memory          MUL   memory
```

<u>ตัวอย่างคำสั่ง</u>

MOV	AL,17h	*เราจะคูณ 17h ด้วย 13h แบบ 8 บิต*
MOV	CL,13h	
IMUL	CL	*ผลลัพธ์ที่ได้จะมีขนาด 16 บิต เก็บที่รีจิสเตอร์ AX*
MOV	BX,1234h	*เราจะนำผลลัพธ์ที่ได้จาก 17h คูณ 13h มาคูณด้วย 1234h*
IMUL	BX	*ค่าใน DX,AX มีค่าเท่ากับ (17h* x *13h)* x *1234h*

คำสั่ง MUL และ IMUL จะมีผลกระทบกับแฟล็กทดและโอเวอร์โฟล์แฟล็กเท่านั้น

❑ **คำสั่งหารแบบคิดเครื่องหมายและไม่คิดเครื่องหมาย : IDIV** [Interger division] **และ DIV** [Division]

เช่นเดียวกับคำสั่งคูณ ตัวตั้งและผลลัพธ์สำหรับการประมวลผลคำสั่งหารใน 8086 จะต้องเก็บรีจิสเตอร์ซึ่งกำหนดไว้โดยจะขึ้นกับขนาดของการหารตัวเลขเช่นเดียวกัน.

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
```

<u>ตัวอย่างคำสั่ง</u>

MOV	AX,4022h	*เราจะหาร 4022h ด้วย 1000h. การหารดังกล่าวเป็นการหารแบบ*
MOV	DX,0000h	*16 บิตดังนั้นเราจึงต้องกำหนดค่าตัวตั้งใน DX,AX.*
MOV	CX,1000h	
DIV	CX	*ผลลัพธ์จากการหารจะมีขนาด 16 บิต เก็บที่รีจิสเตอร์ AX โดยเศษจะเก็บที่ DX. AX = 4 และ DX = 22h.*
MOV	BL,3h	*เราจะนำผลลัพธ์ที่ได้จากมาหารด้วย 3h.*
DIV	BL	*ค่าใน AL จะเก็บผลหารซึ่งมีค่าเท่ากับ 1 และ AH จะเก็บเศษโดยจำมีค่าเท่ากับ 1.*

คำสั่ง DIV และ IDIV จะไม่มีผลกระทบกับแฟล็กใด ๆ. แต่ถ้าในการหารมีการหารด้วยศูนย์ หรือเกิดการหารที่ไม่สามารถเก็บผลลัพธ์ลงในรีจิสเตอร์ที่ต้องการได้ เช่นการหาร 1234 5678h ด้วย 2h หน่วยประมวลผลจะสร้างสัญญาณการขัดจังหวะขึ้นเพื่อแจ้งกับโปรแกรมที่จัดการระบบต่อไป. การเกิดการขัดจังหวะในลักษณะนี้เราเรียกว่าเกิด Exception.

3. กลุ่มคำสั่งแปลงขนาดตัวเลข

จากข้อจำกัดของการใช้คำสั่งหารที่ตัวตั้งจะต้องมีขนาดมากกว่าตัวหาร ทำให้ในบางครั้งที่เราต้องการหารข้อมูลที่มีขนาดเท่ากัน เราจะต้องแปลงตัวตั้งให้มีขนาดที่เหมาะสมเสียก่อน. สำหรับการแปลงขนาดของเลขไม่คิดเครื่องหมายนั้น เราสามารถกระทำได้โดยกำหนดให้ข้อมูลนัยสำคัญสูงที่ขยายเพิ่มมานั้นมีค่าเป็นศูนย์ ตัวอย่างเช่น การขยาย AL ที่เป็นตัวเลขแบบไม่คิดเครื่องหมายให้เป็นข้อมูล 16 บิตใน AX สามารถกระทำได้โดยกำหนดค่าศูนย์ให้กับ AH. แต่ในกรณีของตัวเลขแบบคิดเครื่องหมายนั้นถ้าตัวเลขมีค่าเป็นลบการกำหนดค่าศูนย์ให้กับข้อมูลนัยสำคัญสูงที่ขยายเพิ่มมานั้น จะทำให้ค่าของเลขที่ได้มีความผิดพลาดได้. ในการขยายขนาดของเลขคิดเครื่องหมายเราจึงต้องใช้คำสั่งที่เหมาะสม.

❑ **คำสั่งแปลงจากไบต์เป็นเวิร์ด : CBW** [Convert byte to word]

คำสั่งนี้จะแปลงเลขแบบคิดเครื่องหมายขนาด 8 บิตใน AL เป็นเลขคิดเครื่องหมายขนาด 16 บิตใน AX.

รูปแบบของคำสั่ง.

```
CBW
```

<u>ตัวอย่างคำสั่ง</u>

MOV	AL,22h	*AL = 22h*
CBW		*เมื่อแปลงแล้ว AX=0022h*
MOV	AL,F0h	*AL = F0h = -16*
CBW		*เมื่อแปลงแล้ว AX=FFF0h = -16*

❑ **คำสั่งแปลงจากเวิร์ดเป็นดับเบิลเวิร์ด : CWD** [Convert word to doubleword]

คำสั่งนี้จะแปลงเลขแบบคิดเครื่องหมายขนาด 16 บิตใน AX เป็นเลขคิดเครื่องหมายขนาด 32 บิตใน DX,AX.

รูปแบบของคำสั่ง.

```
CWD
```

<u>ตัวอย่างคำสั่ง</u>

MOV	AX,3422h	*AX = 3422h*
CWD		*เมื่อแปลงแล้ว DX = 0000h , AX=3422h.*
MOV	AX,FFF0h	*AX = FFF0h = -16*
CWD		*เมื่อแปลงแล้ว 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	*จากนั้นนำไปเก็บไว้ที่ CX เนื่องจากรีจิสเตอร์ AL ต้องใช้ในการ*
MUL	BL	*คำนวณ BL*BL.*
ADD	AX, CX	*ได้ผลลัพธ์แล้วนำค่า AL*AL ที่เก็บใน CX มาบวก.*

❑ **ตัวอย่างที่ 2**

คำนวณค่า (CL*BL+DX)/SI โดยคิดเป็นเลขคิดเครื่องหมาย.

MOV	AL,CL	*หาค่าของ CL*BL โดยผลลัพธ์ที่ได้เก็บใน AX.*
MUL	BL	
ADD	AX,DX	*จากนั้นนำ DX มาบวกเข้ากับ AX.*
CWD		*ขยายข้อมูลใน AX เป็น 32 บิตใน DX,AX เพื่อที่จะได้นำ SI มา*
IDIV	SI	*หาร. ผลลัพธ์ที่ได้อยู่ที่ AX. เศษของการหารเก็บที่ DX.*

❑ **ตัวอย่างที่ 3**

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

MOV	SI,DX	*นำค่าของ DX ไปเก็บที่ SI เสียก่อนเนื่องจาก DX จะถูกใช้ในการ*
MUL	BX	*คำนวณ AX*BX. เมื่อคำนวณ AX*BX แล้วนำ SI มาบวกโดยต้อง*
ADD	AX,SI	*นำตัวทดของการบวก SI กับ AX มาทดยัง DX ด้วย.*
ADC	DX,0	
SUB	DI,CX	*ลบ DI ด้วย CX.*
IDIV	DI	*นำไปหาร.*

ผลกระทบของคำสั่งทางคณิตศาสตร์ต่อแฟล็ก

โดยสรุปแล้วคำสั่งคณิตศาสตร์ที่มีผลกระทบกับแฟล็กต่าง ๆ แสดงดังตารางที่ 6.2.

	Flag affected				
Instruction	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*

ตารางที่ 6.2 ผลกระทบของคำสั่งต่าง ๆ ต่อแฟล็ก

ตัวอย่างการเปลี่ยนแปลงของแฟล็กในการทำงานของคำสั่งต่าง 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*

เอกสารอ้างอิง

Thorne M., *Computer Organization and Assembly Language Programming 2nd Edition*, Benjamin/Cumming Publishing Company, 1991.

Agarwal R. K., *80x86 Architecture and Programming Vol 2. Architecture Reference*, Prentice-Hall Inc, NJ, 1991.

สุรศักดิ์ สงวนพงษ์, *เอกสารประกอบการสอนวิชา 204221 องค์ประกอบคอมพิวเตอร์และภาษาแอสเซมบลี้*, 1994.