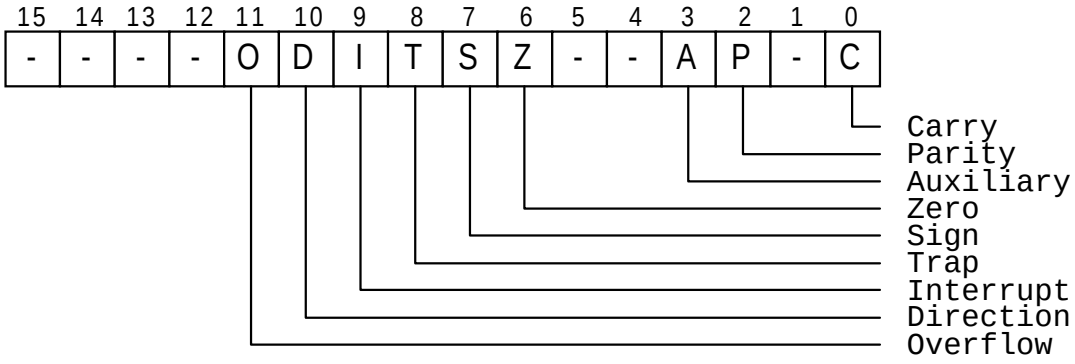
แฟล็กและคำสั่งคณิตศาสตร์

- แฟล็ก
- คำสั่งคณิตศาสตร์
  - คำสั่งเกี่ยวกับการบวกและลบ
  - คำสั่งเกี่ยวกับการคูณและหาร
- ผลของคำสั่งคณิตศาสตร์ต่อการเปลี่ยนแปลงของแฟล็ก

สัณฑิติ พัชรรุ่งเรือง
จิตร์ทัศน์ ฝักเจริญผล

แฟล็ก

- แฟล็ก คือรีจิสเตอร์ที่ใช้เก็บสถานะของระบบ



- ภายใน 8086 แฟล็กมีขนาด 16 บิต โดยในแต่ละบิตจะแทนสถานะต่าง ๆ ของระบบ
- แฟล็กทางคณิตศาสตร์ที่สำคัญมีดังต่อไปนี้

แฟล็กทด (Carry flag)	การทดและการยืม
พาริตี้แฟล็ก (Parity flag)	จำนวนบิตที่มีค่าเป็น 1
แฟล็กเสริม (Auxiliary flag)	การปรับตัวเลข
แฟล็กศูนย์ (Zero flag)	ผลลัพธ์เป็นศูนย์
แฟล็กเครื่องหมาย (Sign flag)	ผลลัพธ์เป็นลบ
โอเวอร์โฟล์แฟล็ก (Overflow flag)	มีเลขล้นหลัก

flag **โดยปกติคำสั่งทางคณิตศาสตร์เท่านั้นที่มีผลกระทบกับแฟล็กเหล่านี้**

สัณฑิติ พัชรรุ่งเรือง
จิตร์ทัศน์ ฝักเจริญผล

แฟล็กต่าง ๆ

- **แฟล็กศูนย์ (Zero-flag)**
  - จะมีค่าเป็น 1 เมื่อผลลัพธ์มีค่าเท่ากับศูนย์ นอกจากกรณีนี้จะมีค่าเป็น 0

EX

```
MOV   AL,10h      Z=?
ADD   AL,E0h      Z=1     AL=0
ADD   AL,20h      Z=0     AL=20h
SUB   AL,10h      Z=0     AL=10h
SUB   AL,10h      Z=1     AL=0
```

- **พาริตี้แฟล็ก (Parity-flag)**
  - จะมีค่าเป็น 1 เมื่อผลลัพธ์มีจำนวนบิตที่มีค่าเป็นหนึ่งเป็นเลขคู่

EX

```
MOV   AL,14h      P=?
ADD   AL,20h      P=0     AL=34h
ADD   AL,10h      P=1     AL=44h
SUB   AL,8h       P=1     AL=3Ch
SUB   AL,10h      P=0     AL=2Ch
```

แฟล็กต่าง ๆ

- **แฟล็กทด (Carry-flag)**
  - จะมีค่าเป็น 1 เมื่อมีการทดหรือการยืมในการคำนวณ โดยจะพิจารณาตัวเลขแบบ**ไม่คิดเครื่องหมาย**.

EX

```
MOV   AL,77h      C=?
ADD   AL,50h      C=0     AL=C7h
ADD   AL,50h      C=1     AL=17h
SUB   AL,A0h      C=1     AL=77h
ADD   AL,27h      C=0     AL=9Eh
```

- **โอเวอร์โฟล์แฟล็ก (Overflow-flag)**
  - จะมีค่าเป็น 1 เมื่อผลลัพธ์มีความผิดพลาดเนื่องจากเลขล้นหลัก โดยจะพิจารณาตัวเลขแบบคิดเครื่องหมาย.

EX

```
MOV   AL,77h      O=?
ADD   AL,50h      O=1     AL=C7h
ADD   AL,50h      O=0     AL=17h
SUB   AL,A0h      O=0     AL=77h
ADD   AL,27h      O=1     AL=9Eh
```


สัณฑิติ พัชรรุ่งเรือง
จิตร์ทัศน์ ฝักเจริญผล

แฟล็กต่าง ๆ

- **แฟล็กเครื่องหมาย (Sign-flag)**
  - จะมีค่าเป็น 1 เมื่อผลลัพธ์มีค่าลบ โดยพิจารณาข้อมูลเป็นแบบเลขคิดเครื่องหมาย.

EX

```
MOV   AL,77h      S=?
ADD   AL,50h      S=1     AL=C7h
ADD   AL,50h      S=0     AL=17h
SUB   AL,A0h      S=0     AL=77h
ADD   AL,27h      S=1     AL=9Eh
```

- **แฟล็กเสริม (Auxiliary-flag)**
  - จะมีค่าเป็น 1 เมื่อผลลัพธ์ต้องมีการปรับค่าของการเก็บตัวเลขแบบ BCD.

แฟล็กต่าง ๆ

- **แฟล็กทิศทาง (Direction-flag)**
  - ใช้สำหรับกำหนดทิศทางของการปรับค่าในคำสั่งเกี่ยวกับสายข้อมูล.
- **แทรปแฟล็ก (Trap-flag)**
  - ใช้สำหรับการตรวจสอบการทำงานของโปรแกรม โดยจะทำให้เกิดการขัดจังหวะทุกครั้งหลังการทำงานของคำสั่งต่างๆ.
- **อินเตอร์รัพท์แฟล็ก (Interrupt-flag)**
  - ใช้สำหรับกำหนดว่าหน่วยประมวลผลจะตอบสนองการขัดจังหวะจากฮาร์ดแวร์บางประเภทหรือไม่

คำสั่งสำหรับกำหนดค่าของแฟล็ก

- แฟล็กบางแฟล็กสามารถกำหนดค่าได้ เนื่องจากมีหน้าที่ในการระบุสถานะบางประการ.
- แฟล็กที่สามารถกำหนดค่าได้ และคำสั่งในการกำหนดค่า.

Flag	*Clear*	*Set*
Carry-flag	CLC	STC
Direction-flag	CLD	STD
Interrupt-flag	CLI	STI

- เราจะเรียกแฟล็กที่มีค่าเป็น 1 ว่าแฟล็กนั้น**เซ็ต** (**flag set**) และเรียกแฟล็กที่มีค่าเป็น 0 ว่าแฟล็กนั้น**เคลียร์** (**flag cleared**.)

คำสั่งคณิตศาสตร์

- คำสั่งเกี่ยวกับการบวกลบ
  - คำสั่งเพิ่มและลดค่า : INC และ DEC
  - คำสั่งบวก : ADD และ ADC
  - คำสั่งลบ : SUB และ SBB
  - คำสั่งเปรียบเทียบ : CMP
  - คำสั่งกลับเครื่องหมาย : NEG
- คำสั่งเกี่ยวกับการคูณและหาร
  - คำสั่งคูณ : IMUL และ MUL
  - คำสั่งหาร : IDIV และ DIV
  - คำสั่งปรับขนาดข้อมูล : CBW และ CWD

คำสั่งเพิ่มค่าและลดค่า

คำสั่ง INC [Increment] และ DEC [Decrement]

- รูปแบบ

```
INC   register      DEC   register
INC   memory        DEC   memory
```

- ในการเพิ่มและลดค่าในหน่วยความจำจะต้องระบุขนาดของข้อมูลด้วย

flag ทั้งสองคำสั่งนี้มีผลกระทบกับแฟล็กทางคณิตศาสตร์ทั้งหมด **ยกเว้น *แฟล็กทด***

EX

ไม่มีการเปลี่ยน C-flag

```
MOV   AL,1
DEC   AL                    AL=0h
DEC   AL                    AL=FFh
MOV   BX,200h
MOV   [BX],AL               [200h]=FFh
INC   BYTE PTR [BX]         [200h]=00h
```

คำสั่งบวก

คำสั่ง ADD [Addition] และ ADC [Add with carry]

- คำสั่ง ADD จะบวกค่าในโอเปอร์แรนด์ตัวหลังเข้ากับตัวหน้า และนำไปเก็บในโอเปอร์แรนด์ตัวหน้า. ส่วนคำสั่ง ADC ก็จะทำเช่นเดียวกัน แต่จะมีการรวม Carry-flag เข้าไปด้วย.
- รูปแบบ

ADD	***regs,regs***	**ADC**	***regs,regs***
ADD	***regs,mem***	**ADC**	***regs,mem***
ADD	***mem,regs***	**ADC**	***mem,regs***
ADD	***regs,imm***	**ADC**	***regs,imm***
ADD	***mem,imm***	**ADC**	***mem,imm***

- เรานิยมใช้คำสั่ง ADC ในการบวกเลขที่ต้องการรวมตัวทดจากการคำนวณที่ผ่านมา ยกตัวอย่างเช่น การบวกเลขที่อยู่ในรีจิสเตอร์หลายตัวต่อเนื่องกัน.

flag คำสั่ง ADD และ ADC มีผลกระทบถึงแฟล็กทางคณิตศาสตร์ทุกตัว

คำสั่งลบ

คำสั่ง SUB [Substraction] และ SBB [Sub with borrow]

- คำสั่ง SUB จะลบค่าในโอเปอร์แรนด์ตัวหลังออกจากตัวหน้า และนำไปเก็บในโอเปอร์แรนด์ตัวหน้า. ส่วนคำสั่ง SBB ก็จะทำเช่นเดียวกัน แต่จะมีการรวมตัวยืมซึ่งเก็บอยู่ใน Carry-flag ด้วย.
- รูปแบบ

SUB	***regs,regs***	**SBB**	***regs,regs***
SUB	***regs,mem***	**SBB**	***regs,mem***
SUB	***mem,regs***	**SBB**	***mem,regs***
SUB	***regs,imm***	**SBB**	***regs,imm***
SUB	***mem,imm***	**SBB**	***mem,imm***

- เรานิยมใช้คำสั่ง SBB ในการลบเลขที่ต้องการรวมตัวยืมจากการคำนวณที่ผ่านมา โดยจะมีลักษณะการใช้งานคล้าย ๆ กับคำสั่ง ADC.

flag คำสั่ง SUB และ SBB มีผลกระทบถึงแฟล็กทางคณิตศาสตร์ทุกตัว

ตัวอย่างการใช้คำสั่งบวกและลบ

- การบวกเลข 32 บิต 34DA 1115h เข้ากับตัวเลข 32 บิตที่เก็บในคู่รีจิสเตอร์ CX,BX

EX

```
MOV   AX,1115h      นำ 34DA 115h เก็บใน DX,AX
MOV   DX,34DAh
ADD   AX,BX         บวก 16 บิตล่าง
ADC   DX,CX         บวก 16 บิตบนพร้อมตัวทด
```

- การลบเลข 16 บิต ในรีจิสเตอร์ AX ด้วยเลข 8 บิต ในรีจิสเตอร์ BL

EX

```
SUB   AL,BL         ลบ 8 บิตล่าง
SBB   AH,0          ลบ 8 บิตบนพร้อมตัวยืม
```

- การลบเลข 32 บิต ในหน่วยความจำ offset 200h ด้วย เลข 32 บิตในหน่วยความจำ offset 204h

EX

```
MOV   AX,[204h]     อ่านค่าตัวลบมาเก็บใน AX
SUB   [200h],AX     ลบ 16 บิตล่าง
MOV   AX,[206h]     อ่านค่าตัวลบมาเก็บใน AX
SBB   [202h],AX     ลบ 16 บิตบนและตัวยืม
```


สัณฑิติ พัชรรุ่งเรือง
จิตร์ทัศน์ ฝักเจริญผล

คำสั่งเปรียบเทียบ

คำสั่ง CMP [Compare]

- คำสั่ง CMP จะทำงานเหมือนคำสั่ง SUB แต่จะไม่มีการเปลี่ยนแปลงค่าในโอเปอร์แรนด์ทั้งสอง แต่จะมีการเปลี่ยนแปลงค่าใน Flag
- โดยปกติเราจะใช้คำสั่ง CMP ในการเปรียบเทียบและตัดสินใจในการกระโดด.

flag การเปลี่ยนแปลง flag ของคำสั่ง CMP จะเหมือนกับคำสั่ง SUB.

EX

```
MOV   AX,10h
CMP   AX,100h
```

Z-flag = 0, C-flag = 1,
O-flag = 0, S-flag = 1.

```
MOV   BL,10h
CMP   AL,BL
```

Z-flag = 1, C-flag = 0,
O-flag = 0, S-flag = 0.

สัณฑิติ พัชรรุ่งเรือง
จิตร์ทัศน์ ฝักเจริญผล

คำสั่งกลับเครื่องหมาย

คำสั่ง NEG [Negation]

- คำสั่ง NEG จะกลับค่าของโอเปอร์แรนด์ให้เป็นค่าลบของค่าเดิม. การแปลงใช้วิธีแบบ 2's complement.
- รูปแบบ

```
NEG   regs
NEG   mem
```

flag คำสั่ง NEG จะมีผลกระทบกับแฟล็กทางคณิตศาสตร์ทั้งหมด แต่สำหรับ **แฟล็กทดจะมีค่าเป็นหนึ่งเสมอหลังการทำงานของคำสั่งนี้.**

EX

```
MOV   AL,10h
NEG   AL                    AL = F0h, C-flag = 1.
MOV   BX,200h
MOV   [BX],AL
NEG   BYTE PTR [BX]         [200h] = 10h, Cf = 1.
```

ผลของการทำงานของคำสั่งต่างๆ ต่อการเปลี่ยนค่าของแฟล็ก

- ผลกระทบของคำสั่งคณิตศาสตร์

Instruction	*ZF*	*CF*	*SF*	*OF*
ADD	y	y	y	y
ADC	y	y	y	y
SUB	y	y	y	y
SBB	y	y	y	y
INC	y	**n**	y	y
DEC	y	**n**	y	y
CMP	y	y	y	y
NEG	y	y	y	y
IMUL	**n**	y	**n**	y
MUL	**n**	y	**n**	y
IDIV	**n**	**n**	**n**	**n**
DIV	**n**	**n**	**n**	**n**
CBW	**n**	**n**	**n**	**n**
CWD	**n**	**n**	**n**	**n**

EX

Instruction		*ZF*	*CF*	*SF*	*OF*	*PF*	*Remark*
MOV	**AL,70h**	**?**	**?**	**?**	**?**	**?**	
ADD	**AL,20h**	**0**	**0**	**1**	**1**	**1**	**AL=90h**
ADD	**AL,70h**	**1**	**1**	**0**	**0**	**1**	**AL=00h**
DEC	**AL**	**0**	**1***	**1**	**0**	**1**	**AL=0FFh**
ADD	**AL,5**	**0**	**1**	**0**	**0**	**0**	**AL=04h**
ADD	**AL,30h**	**0**	**0**	**0**	**0**	**0**	**AL=34h**
SUB	**AL,40h**	**0**	**1**	**1**	**0**	**0**	**AL=F4h**
SUB	**AL,7F**	**0**	**0**	**0**	**1**	**0**	**AL=75h**
NEG	**AL**	**1**	**1****	**1**	**0**	**0**	**AL=8Bh**
ADC	**AL,20h**	**1**	**0**	**0**	**0**	**0**	**AL=0ABh**

* DEC และ INC ไม่กระทบ C-flag
** NEG เปลี่ยน C-flag = 1

สัณฑิติ พัชรรุ่งเรือง
จิตร์ทัศน์ ฝักเจริญผล

คำสั่งคูณ

คำสั่ง IMUL [Integer Multiplication] และ MUL [Multiplication]

- ในการใช้คำสั่งคูณ เราจะระบุเฉพาะตัวคูณเท่านั้น สำหรับตัวตั้งและผลลัพธ์จะอยู่ในรีจิสเตอร์ที่กำหนดไว้แล้ว โดยจะขึ้นกับขนาดของการคูณ.
- รูปแบบ

IMUL *regs*	**MUL** *regs*
IMUL *mem*	**MUL** *mem*

- การคูณ 8 บิต
  - ตัวตั้ง : AL ผลลัพธ์ : AX
- การคูณ 16 บิต
  - ตัวตั้ง : AX ผลลัพธ์ : DX,AX
- คำสั่ง IMUL จะคูณแบบคิดเครื่องหมาย ส่วน MUL จะคูณแบบไม่คิดเครื่องหมาย.

flag คำสั่งทั้งสองจะมีผลกระทบกับ แฟล็กทด และ โอเวอร์โฟล์แฟล็กเท่านั้น.

คำสั่งหาร

คำสั่ง IDIV [Integer Division] และ DIV [Division]

- เช่นเดียวกับการใช้คำสั่งคูณ เราจะระบุเฉพาะตัวหารเท่านั้น. ตัวตั้งและผลลัพธ์จะอยู่ในรีจิสเตอร์ที่กำหนดไว้แล้ว โดยจะขึ้นกับขนาดของการหาร.
- รูปแบบ

IDIV *regs*	**DIV** *regs*
IDIV *mem*	**DIV** *mem*

- การหาร 8 บิต
  - ตัวตั้ง : AX ผลลัพธ์ : AL เศษ : AH
- การหาร 16 บิต
  - ตัวตั้ง : DX, AX ผลลัพธ์ : AX เศษ : DX
- คำสั่ง IDIV จะหารแบบคิดเครื่องหมาย ส่วน DIV จะหารแบบไม่คิดเครื่องหมาย.

flag คำสั่งทั้งสองจะ**ไม่มีผลกระทบกับแฟล็ก**ทางคณิตศาสตร์.

สัณฑิติ พัชรรุ่งเรือง
จิตร์ทัศน์ ฝักเจริญผล

ตัวอย่างการใช้คำสั่งคูณและหาร

- การคูณค่า 12h ด้วย ค่าใน BL

EX

```
MOV   AL,12h      ตั้งค่าให้กับตัวตั้ง
MUL   BL          คูณด้วย BL ผลลัพธ์อยู่ใน AX.
```

- การคูณค่าในหน่วยความจำขนาด 16 บิตที่offset 200h ด้วย ค่าใน AX จากนั้นนำผลลัพธ์ที่ได้มาบวกด้วยค่าใน BX.

EX

```
MUL   WORD PTR[200h]  คูณ AX กับค่าใน
ADD   AX,BX           [200h]. จากนั้นนำ BX
ADC   DX,0            มารวมกับผลลัพธ์.
```

- การหารค่า 32 บิต 1234 5678h ด้วยค่าในหน่วยความจำแบบ 16 บิตที่ตำแหน่ง [200h].

EX

```
MOV   AX,5678h        ตั้งค่าตัวตั้ง.
MOV   DX,1234h
IDIV  WORD PTR[200h]  หารด้วย [200h].
```


สัณฑิติ พัชรรุ่งเรือง
จิตร์ทัศน์ ฝักเจริญผล

คำสั่งปรับขนาดข้อมูล

คำสั่ง CBW [Convert byte to word] และ CWD [Convert word to doubleword]

- ในการหารนั้น บางครั้งเราต้องการหารตัวเลขที่มีขนาดเท่ากัน (จำนวนบิตเท่ากัน) แต่ใน 8086 ไม่มีคำสั่งดังกล่าว.
- ถ้าข้อมูลเป็นข้อมูลแบบไม่คิดเครื่องหมาย เราสามารถที่จะกำหนดค่า 0 ให้กับข้อมูลนัยสำคัญสูงที่ขยายเพิ่มขึ้นมาได้. แต่ถ้าข้อมูลเป็นตัวเลขคิดเครื่องหมายเราจะต้องใช้การปรับค่าแบบอื่น.
- รูปแบบ

  CBW **CWD**

- คำสั่ง CBW จะขยายขนาดข้อมูลแบบคิดเครื่องหมายใน AL ให้มีขนาด 16 บิตใน AX. ส่วนคำสั่ง CWD จะขยายขนาดข้อมูลแบบคิดเครื่องหมายใน AX ให้มีขนาด 32 บิตใน DX,AX.
- การหารค่าใน AL ด้วย BL.

EX

```
CBW          ขยาย AL -> AX.
IDIV  BL     หารด้วย BL ผลลัพธ์อยู่ใน AL.
```


สัณฑิติ พัชรรุ่งเรือง
จิตร์ทัศน์ ฝักเจริญผล

ตัวอย่างการใช้คำสั่งทางคณิตศาสตร์

- คำนวณค่า $AL^2 + BL^2$. คิดตัวเลขแบบไม่คิดเครื่องหมายโดยให้ผลลัพธ์เป็นเลข 16 บิต เก็บที่รีจิสเตอร์ AX.

EX

```
MUL   AL          หาค่า AL*AL.
MOV   CX, AX      นำไปเก็บไว้ที่ CX เนื่องจาก
MOV   AL, BL      AL ต้องใช้ในการคำนวณ.
MUL   BL          ได้ผลลัพธ์แล้วนำค่า AL*AL
ADD   AX, CX      ที่เก็บใน CX มาบวก.
```

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

EX

```
MOV   AL,CL       หาค่า CL*BL.
MUL   BL
ADD   AX,DX       บวกกับ DX.
CWD               ขยายขนาดเป็น 32 บิต.
IDIV  SI          หารด้วย SI ผลอยู่ที่ AX เศษที่ DX.
```

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

สัณฑิติ พัชรรุ่งเรือง
จิตร์ทัศน์ ฝักเจริญผล