บทที่ 12

การคำนวณทางคณิตศาสตร์ I

ARITHMETIC I PROCESSING BINARY DATA

วัตถุประสงค์

หลังจากที่ท่านศึกษาบทนี้ท่านจะมีความเข้าใจดังต่อไปนี้

- ท่านสามารถเข้าใจเกี่ยวกับคำสั่งการคำนาณทางคณิตศาสตร์
- สามารถเข้าใจคำสั่ง ADD , ADC , MUL , IMUL , DIV , IDIV
- สามารถใช้คำสั่ง SHR , SAR , SHL , SAL

CT 215 317

บทที่ 12
การคำนวณทางคณิตศาสตร์ I
ARITHMETIC I PROCESSING BINARY DATA

บทนำ

ในชีวิตประจำวันของมนุษย์เราใช้การคำนวณทางคณิตศาสตร์ด้วยเลขฐาน 10 ส่วนรูปแบบของไมโครคอมพิวเตอร์ ใช้การคำนวณทางคณิตศาสตร์ด้วยเลขฐาน 2 ในการศึกษาบทนี้จะรวมถึงการบวก การลบ การคูณ และการหาร แบบคิดเครื่องหมาย และไม่คิดเครื่องหมาย และอธิบายการเปลี่ยนข้อมูลระหว่าง BINARY และ ASCII

การบวก และการลบ

คำสั่ง ADD และคำสั่ง SUB ใช้ประมวลผลข้อมูลเลขฐานสอง รูปแบบของการคำนวนของคอมพิวเตอร์ในการลบเลขจะใช้หลักการของ 2'คอมพลีเมนต์ โดยการเปลี่ยนค่าทุกบิทของโอเปอร์รานด์ 2 เป็นตรงข้าม แล้วบวกด้วย 1 และบวกเข้ากับผลลัพธ์ของโอเปอร์รานด์ 1 คำสั่งที่ใช้ในการบวก ลบ มีดังต่อไปนี้

```
ADD/SUB     REGISTER,REGISTER
ADD/SUB     MEMORY,REGISTER
ADD/SUB     REGISTER,MEMORY
ADD/SUB     REGISTER,IMMEDIATE
ADD/SUB     MEMORY,IMMEDIATE
```

การทำงานของการบวก การลบ จะไม่มีการกระทำระหว่างหน่วย ความจำโดยตรง จะใช้รีจิสเตอร์ช่วยในการบวก การลบ ดังตัวอย่างต่อไปนี้ ในการบวก WORDA กับ WORDB

```
WORDA     DW   123          ;DEFINE  WORDA
WORDB     DW    25          ;DEFINE  WORDB
MOV       AX,WORDA          ;MOVE WORDA TO AX
ADD       AX,WORDB          ;ADD WORDB TO AX
MOV       WORDB,AX          ;MOVE AX TO WORDB
```

ตัวอย่างรูป 12-1 เป็นตัวอย่างของการบวกและการลบ สำหรับการประมวลผลข้อมูลขนาดไบท์, เวิร์ด หรือ 2 เวิร์ด (80386/486) PROCEDURE B10ADD ใช้คำสั่ง ADD ในการประมวลผลข้อมูลขนาดไบท์ และ C10SUB ใช้คำสั่ง SUB ประมวลผลข้อมูลขนาดเวิร์ด

```
            page 60,132
            TITLE     EXADD (COM)  ADD and SUB operations
            .MODEL  SMALL
            .CODE
            ORG    100H
BEGIN:      JMP      SHORT MAIN
; ---------------------------------------------
    BYTEA     DB    64H              ;Data items
    BYTEB     DB    40H
    BYTEC     DB    16H
    WORDA     DW    4000H
    WORDB     DW    2000H
    WORDC     DW    1000H
; ---------------------------------------------
    MAIN      PROC   NEAR            ;Main procedure:
              CALL   B10ADD          ;Call ADD routine
              CALL   C10SUB          ;Call SUB routine
              MOV    AH,4CH
              INT    21H             ;Exit
    MAIN      ENDP
```

```
;Examples of ADD bytes:
;---------------------
B10ADD    PROC
          MOV     AL,BYTEA
          MOV     BL,BYTEB
          ADD     AL,BL          ;Register-to-register
          ADD     AL,BYTEC       ;Memory-to-register
          ADD     BYTEA,BL       ;Register-to-memory
          ADD     BL,10H         ;Immediate-to-register
          ADD     BYTEA,25H      ;Immediate-to-memory
          RET
B10ADD    ENDP
;Examples of SUB words:
;---------------------
C10SUB    PROC
          MOV     AX,WORDA
          MOV     BX,WORDB
          SUB     AX,BX          ;Register-from-register
          SUB     AX,WORDC       ;Memory-from-register
          SUB     WORDA,BX       ;Register-from-memory
          SUB     BX,1000H       ;Immediate-from-register
          SUB     WORDA,256H     ;Immediate-from-memory
          RET
C10SUB    ENDP

          END     BEGIN
```

รูป 12-1 ตัวอย่างการบวกการลบ

OVERFLOWS

การเกิดโอเวอร์โฟล์วในการคำนวนทางคณิตศาสตร์ อาจเกิดขึ้นได้ ข้อมูลขนาด 1 ไบท์ เราคิดเครื่องหมาย 1 บิต จะเหลือบิตข้อมูลเพียง 7 บิต การคำนวน

ทางคณิตศาสตร์ สามารถเกิดรอเวอร์รฟล่วได้ง่ายกับขนาดของรีจิสเตอร์ ขนาด 1 ไบท์ สามารถรับข้อมูลได้จาก -128 ถึง +127 ผลรวมที่เก็บใน AL จะเก็บได้สูงสุด +127 ถ้ามากกว่านี้จะมีผลต่อ AH โดยค่ารอเวอร์รฟล่วเก็บใน AH สมมติว่า ค่า AL = 60H

```
ADD  AL,20H
```

การประมวลผลจะได้ผลรวมใน AL = 80H แต่การทำงานจะไปเซ็ทค่าแฟลกรอเวอร์รฟล่ว และแฟลกเครื่องหมายเป็นค่าติดลบ เหตุผลก็เพราะว่าค่า 80H หรือค่าเลขฐานสอง 1000 0000 เป็นค่าติดลบ แทนที่จะเป็น +128 ผลรวมจะได้ -128 ปัญหาที่เกิดขึ้นเนื่องจาก AL เล็กเกินไป ผลรวมจะต้องเก็บใน AX

การขยายผลบวก (EXTENDING THE SUM)

ในตัวอย่างต่อไป คำสั่ง CBW (CONVERT BYTE TO WORD) จะเป็นตัวขยาย 60H ใน AL โดยการให้บิตเครื่องหมายของ AL = 0 และให้อยู่ที่บิตสูงสุดของ AH จะได้ 0060H ใน AX ผลจากการบวกจะได้ผลลัพธ์ใน AX = 0080H หรือ +128

```
CBW                    ;EXTEND AL SIGN INTO AH
ADD  AX,20H            ;ADD TO AX
```

ข้อมูลขนาด 1 เวิร์ดให้บิตสูงสุดเป็นบิตเครื่องหมาย บิตข้อมูล 15 บิต กำหนดขอบเขตข้อมูลได้ -32768 ถึง 32767

การบวกหลายเวิร์ด (MULTIWORD ADDITION)

การตรวจสอบการทำงานของคำนวณทางคณิตศาสตร์หลาย ๆ เวิร์ด ในรูป 12-2 โปรแกรมย่อย D10DWD แสดงตัวอย่างง่ายโดยการบวกทีละเวิร์ด (WORD1A และ WORD1B) และคู่ที่ 2 (WORD2A และ WORD2B) และเก็บผลรวมในคู่ที่ 3 (WORD3A และ WORD3B) การทำงานจะได้ค่าดังนี้

```
WORD1A และ WORD1B:     0123BC62
WORD2A และ WORD2B:     0012553A
WORD3A และ WORD3B:     0136119C
```

เนื่องจากการเก็บข้อมูลในหน่วยความจำจะสลับค่ากัน ค่าที่กำหนดในลักษณะสลับค่า BC62 0123 และ 553A 0012 ตัวแอสแซมเบลอจะเก็บค่าเหล่านี้ในหน่วยความจำในรูปของข้อมูลสลับกัน

WORD1A และ WORD1B: 62BC 2301

WORD2A และ WORD2B: 3A55 1200

คำสั่ง MOV และการทำงานของ ADD จะเก็บข้อมูลสลับไบท์ใน AX และบวกเวิร์ดซ้ายสุด

```
WORD1A:      BC62
WORD2A:    + 553A
TOTAL:       1119C     (9C11 เก็บใน WORD3A)
```

แต่ผลรวมของ WORD1A บวกกับ WORD2A ค่าผลบวกที่ได้จะเกินค่าของ AX แฟลกตัวทดจะเซ็ทเป็น 1 การบวกต่อไป จะต้องใช้การบวกพร้อมตัวทด ADC แทนที่ ADD ฉนั้น การบวก ADC เป็นการบวกค่า 2 ค่า พร้อมตัวทด

```
WORD1B          0123
WORD2B        + 0012
PLUS CARRY      ___1
TOTAL           0136      (3601 เก็บใน WORD3B)
```

การใช้ DEBUG ในการเอ็กซีคิวส์ทีละคำสั่งของการคำนวณทางคณิตศาสตร์ ท่านจะเห็นว่าผลรวม 0136 ใน AX และค่าที่สลับ 9C11 เก็บใน WORD3A และ 3601 ใน WORD3B

จากรูป 12-2 โปรแกรม E10DWD สามารถกำหนดค่าในการบวกได้ทุก ๆ ความยาว ถึงแม้ว่าในตัวอย่างนี้ จะบวกเพียง 2 คู่ คือ WORD1A กับ WORD1B และ WORD2A กับ WORD2B การวนรอบแต่ละครั้งจะมีการบวกทีละคู่ (เวิร์ด) ในกรณีให้ทำเพียง 2 รอบ รอบแรกจะเป็นการบวกเวิร์ดซ้ายสุด และรอบที่ 2 เป็นการบวกเวิร์ดขวาสุด แต่ในรอบที่ 2 จะทำการประมวลผลเวิร์ดขวามือ แอดเดรสที่อยู่ใน SI, DI และ BX จะเพิ่มขึ้นทีละ 2 คำสั่ง INC จะทำงานในการเพิ่มค่าของรีจิสเตอร์แต่ละตัว

สังเกตว่า คำสั่ง ADD reg,02 จะเป็นการเคลียแฟลกตัวทด และจะทำให้คำตอบผิด แต่คำสั่ง INC ไม่มีผลต่อแฟลก

เนื่องจากการวนรอบแต่ละครั้งจะมีการบวก คำสั่ง ADC ก่อนที่จะใช้คำสั่งนี้ต้องเคลียแฟลกตัวทด โดยใช้คำสั่ง CLC ก่อนการบวกทุกครั้งในเวิร์ดแรกที่มีการบวก วิธีการทำงานต้องมีวิธีดังนี้

(1) ต้องกำหนดเวิร์ดถัดไปแต่ละเวิร์ด

(2) ประมวลผลจากซ้ายไปขวา

(3) CX เก็บค่าจำนวนเวิร์ดที่จะบวก

สำหรับการทำงานของการลบหลาย ๆ เวิร์ด คำสั่งที่เทียบเท่า SBB (SUBTRACT WITH BORROW) ใช้แทนคำสั่ง ADC ด้วย SBB ในโปรแกรมย่อย E10DWD

การคำนวนทางคณิตศาสตร์ 80386/486 (ARITHMETIC)

โปรเซสเซอร์ 80386/486 มีรีจิสเตอร์ 32 บิตที่ใช้ในการคำนวณทางคณิตศาสตร์ การบวก EBX กับ EAX ดังตัวอย่างรหัสคำสั่งดังนี้

```
ADD   EAX,EBX              ;80386/486
```

เป็นการบวกครั้งละ 4 เวิร์ด ใน 80386

```
            TITLE     EXDBADD (COM)  Adding doublewords
            .MODEL    SMALL
            .CODE
            ORG       100H
BEGIN:      JMP       SHORT MAIN
; ----------------------------------------
WORD1A      DW        0BC62H                ;Data items
WORD1B      DW        0123H
WORD2A      DW        553AH
```

```
WORD2B    DW      0012H
WORD3A    DW      ?
WORD3B    DW      ?
;-----------------------------------------
MAIN      PROC    NEAR                ;Main procedure
          CALL    D10DWD              ;Call 1st ADD
          CALL    E10DWD              ;Call 2nd ADD
          MOV     AH,4CH
          INT     21H                 ;Exit
MAIN      ENDP
;   Example of ADD doublewords:
;   -------------------------
D10DWD    PROC
          MOV     AX,WORD1A           ;Add leftmost word
          ADD     AX,WORD2A
          MOV     WORD3A,AX
          MOV     AX,WORD1B           ;Add rightmost word
          ADC     AX,WORD2B           ;  with carry
          MOV     WORD3B,AX
          RET
D10DWD    ENDP
;   Generalized add operation:
;   -------------------------
E10DWD    PROC
          CLC                         ;Clear carry flag
          MOV     CX,02               ;Set loop count
          LEA     SI,WORD1A           ;Leftmost word
          LEA     DI,WORD2A           ;Leftmost word
          LEA     BX,WORD3A           ;Leftmost word of sum
E20:
          MOV     AX,[SI]             ;Move word to AX
          ADC     AX,[DI]             ;Add with carry to AX
```

```
        MOV     [BX],AX             ;Store word
        INC     SI                  ;Adjust addresses for
        INC     SI                  ;  next word to right
        INC     DI
        INC     DI
        INC     BX
        INC     BX
        LOOP    E20                 ;Repeat for next word
        RET
E10DWD  ENDP

        END     BEGIN
```

รูปที่ 12-2 การบวกหลายๆเวิร์ด

ข้อมูลแบบคิดเครื่องหมายและไม่คิดเครื่องหมาย (UNSIGNED AND SIGNED DATA)

ฟิลด์ของตัวเลขที่ไม่คิดเครื่องหมาย เช่น รหัสลูกค้า และแอดเดรสหน่วยความจำ ฟิลด์ที่คิดเครื่องหมายไม่ว่าจะเป็นข้อมูลที่มีค่าบวกหรือข้อมูลที่มีค่าลบ เป็นรายการหนี้ของลูกค้า และจำนวนพีชคณิต ส่วนฟิลด์ตัวเลขที่คิดเครื่องหมาย เราสมมุติให้เป็นค่าบวก เช่น อัตราเงินที่จ่าย จำนวนวันใน 1 เดือน และค่าของ PI = 22/7

สำหรับข้อมูลไม่คิดเครื่องหมาย บิตทั้งหมดจะใช้เป็นบิตข้อมูล ค่าของข้อมูล 16 บิต แทนที่จะได้ 32676 ก็จะเป็น 65535 สำหรับข้อมูลที่คิดเครื่องหมาย บิตซ้ายสุดจะเป็นบิตเครื่องหมาย คำสั่ง ADD และ SUB จะไม่ชี้ให้เห็นความแตกต่างระหว่างไม่คิดเครื่องหมาย และคิดเครื่องหมาย ตัวอย่างการบวกเลขฐานสอง 2 จำนวน ข้อมูลบิตซ้ายสุด 1 บิต สำหรับข้อมูลไม่คิดเครื่องหมาย บิตจะมีค่า 249 แต่สำหรับข้อมูลคิดเครื่องหมาย บิตจะมีค่า -7 การบวกจะไม่เซ็ทโอเวอร์โฟล์ว หรือแฟลกตัวทด

BINARY	UNSIGNED DECIMAL	SIGNED DECIMAL	OF	CF
11111001	2 49	-7		
+ 00000010	+ 2	+ 2		
11111011	2 51	-5	0	0

ผลลัพธ์ของการบวกเลขฐานสอง คือมีลักษณะเหมือนกันทั้งไม่คิดเครื่องหมาย และคิดเครื่องหมาย อย่างไรก็ตาม บิตในฟิลด์ไม่คิดเครื่องหมาย = 251 ขณะที่บิตในฟิลด์คิดเครื่องหมาย = -5

ตัวทดทางคณิตศาสตร์ (ARITHMETIC CARRY)

แฟลกตัวทดจะถูกเซ็ทเมื่อ มีตัวทดออกจากบิตเครื่องหมาย ขณะที่มีตัวทดเกิดขึ้นในข้อมูลชนิดที่ไม่คิดเครื่องหมาย ผลลัพธ์จะไม่ถูกต้อง จากตัวอย่างต่อไปนี้เป็นการบวกที่เกิดตัวทด การทำงานแบบไม่คิดเครื่องหมายที่ผลลัพธ์ไม่ถูกต้อง ขณะที่แบบคิดเครื่องหมายถูกต้อง

BINARY	UNSIGNED DECIMAL	SIGNED DECIMAL	OF	CF
11111100	252	-4		
+ 00000101	+ 5	+ 5		
(1) 00000001	+1	1	0	1
	(INVALID)	(VALID)		

ค่าเกินขนาดทางคณิตศาสตร์ (ARITHMETIC OVERFLOW)

แฟลกโอเวอร์โฟล์วจะเซ็ทก็ต่อเมื่อ มีตัวทดไปที่บิตเครื่องหมาย และไม่มีตัวทดออก หรือมีตัวทดออก กับไม่มีตัวทดเข้า การเกิดโอเวอร์โฟล์วของข้อมูลแบบคิดเครื่องหมาย ผลลัพธ์จะไม่ถูกต้อง ดังตัวอย่าง

BINARY	UNSIGNED DECIMAL	SIGNED DECIMAL	OF	CF
01111001	121	+121		
+ 00001011	+ 11	+ 11		
10000100	132	-124	1	0
	(VALID)	(INVALID)		

การบวกอาจจะเกิดขึ้นทั้งตัวทดและรีเวอร์โฟล์ว ในตัวอย่างต่อไป ตัวทดที่เกิดขึ้น แบบไม่คิดเครื่องหมายไม่ถูกต้อง ขณะที่รีเวอร์โฟล์วแบบคิดเครื่องหมายไม่ถูกต้อง

BINARY	UNSIGNED DECIMAL	SIGNED DECIMAL	OF	CF
11110110	246	-10		
+ 10001001	+ 137	- 119	1	1
(1) 01111111	127	+127		
	(INVALID)	(INVALID)		

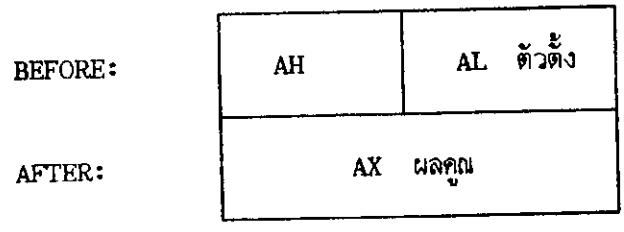
การคูณ MULTIPLICATION

สำหรับการคูณ จะใช้คำสั่ง MUL สำหรับการคูณไม่คิดเครื่องหมาย และคำสั่ง IMUL (คูณเลขจำนวนเต็ม) สำหรับการคูณแบบคิดเครื่องหมาย ซึ่งทั้งสองคำสั่งจะมีผลต่อแฟลกตัวทด และแฟลกรีเวอร์โฟล์ว ผู้เขียนโปรแกรมจะต้องควบคุมรูปแบบของข้อมูลที่ท่านประมวลผล ซึ่งมีรูปแบบคำสั่งดังนี้

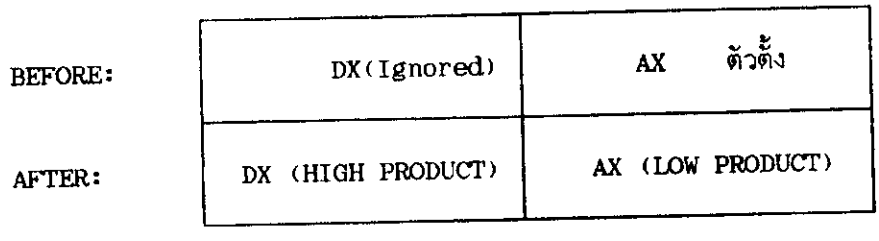
MUL/IMUL	REGISTER/MEMORY

การทำงานของการคูณจะคูณครั้งละไบท์ เวิร์ด หรือ 2 เวิร์ด ใน 1 เวลา

BYTE TIMES BYTE เป็นการคูณที่ตัวตั้งอยู่ที่ AL และตัวคูณมีขนาดไบท์อยู่ใน หน่วยความจำหรือรีจิสเตอร์ หลังจากการคูณผลคูณอยู่ใน AX



WORD TIMES WORD ตัวตั้งอยู่ใน AX และตัวคูณมีขนาดเวิร์ดอยู่ในหน่วยความจำหรือรีจิสเตอร์ หลังจากการคูณผลคูณจะมีขนาด 2 เวิร์ด ต้องใช้รีจิสเตอร์ 2 ตัวเวิร์ด สูงจะเก็บใน DX และเวิร์ดต่ำเก็บใน AX



DOUBLEWORD TIMES DOUBLEWORD ใช้ใน 80386/486 ตัวตั้งอยู่ใน EAX ตัวคูณอยู่ในหน่วยความจำ หรือรีจิสเตอร์ขนาด 2 เวิร์ด ผลคูณจะเก็บไว้ใน EDX:EAX

ขนาดของฟิลด์ FIELD SIZES

โอเปอร์รานด์ของคำสั่ง MUL หรือ IMUL จะอ้างถึงตัวคูณ พิจารณาจากคำสั่ง

MUL MULTR:

DEFINITION OF MULTR	OPERAND ASSUMED
DB	AL TIMES BYTE
DW	AX TIMES WORD
DD	EAX TIMES DOUBLEWORD

เมื่อตัวคูณอยู่ในรีจิสเตอร์ ขนาดของรีจิสเตอร์เป็นตัวกำหนดการทำงาน

INSTRUCTION	MULTIPILER	MULTIPICAND	PRODUC
MUL CL	BYTE	AL	AX
MUL BX	WORD	AX	DX:AX
MUL EBX	DOUBLEWORD	EAX	EDX:EAX

การคูณไม่คิดเครื่องหมาย (UNSIGNED MULTIPLICATION : MUL)

คำสั่ง MUL เป็นคำสั่งการคูณแบบไม่คิดเครื่องหมาย ในรูป 12-3 โปรแกรม C10MUL เป็นตัวอย่าง 3 ตัวอย่างของการคูณ ไบท์กับไบท์ เวิร์ดกับเวิร์ด และเวิร์ดกับไบท์ ตัวอย่างคำสั่ง MUL ตัวแรกเป็นการคูณ 80H และ 40H ผลคูณเก็บใน AX คือ 2000H คำสั่ง MUL ตัวที่สองจะให้ผลลัพธ์ใน DX:AX คือ 1000 0000

คำสั่ง MUL คำสั่งที่ 3 เป็นการคูณเวิร์ดกับไบท์ โดยการขยาย BYTE1 เป็นเวิร์ด แต่ค่าที่สมมุติขึ้นไม่คิดเครื่องหมาย ตัวอย่างสมมุติ บิตซ้ายสุดใน AH จะมีค่าเป็น 0 (ปัญหานี้ใช้คำสั่ง CBW นี้จะทำให้บิตซ้ายสุดของ AL จะเป็น 0 หรือ 1) ผลคูณอยู่ใน DX:AX คือ 0040 0000H

```
TITLE     EXMULT (COM)  MUL & IMUL operations
          .MODEL  SMALL
          .CODE
          ORG          100H
BEGIN:    JMP          SHORT MAIN
; ---------------------------------------------
BYTE1     DB           80H
BYTE2     DB           40H
WORD1     DW           8000H
WORD2     DW           2000H
; ---------------------------------------------
MAIN      PROC   NEAR                    ;Main procedure
          CALL   C10MUL                  ;Call MUL  routine
          CALL   D10IMUL                 ;Call IMUL routine
          MOV          AH,4CH
          INT          21H               ;Exit
MAIN      ENDP
;                      Examples of MUL:
;                      ---------------
C10MUL    PROC
          MOV          AL,BYTE1          ;Byte x byte
          MUL          BYTE2             ;  product in AX
          MOV          AX,WORD1          ;Word x word
          MUL          WORD2             ;  product in DX:AX
          MOV          AL,BYTE1          ;Byte x word
          SUB          AH,AH             ;  extend multiplicand in AH
          MUL          WORD1             ;  product in DX:AX
          RET
C10MUL    ENDP
```


331

```
;                       Examples of IMUL:
;                       ----------------
D10IMUL     PROC
            MOV         AL,BYTE1        ;Byte x byte
            IMUL    BYTE2               ;  product in AX
            MOV         AX,WORD1        ;Word x word
            IMUL    WORD2               ;  product in DX:AX
            MOV         AL,BYTE1        ;Byte x word
            CBW                         ;  extend multiplicand in AH
            IMUL    WORD1               ;  product in DX:AX
            RET
D10IMUL     ENDP
            END         BEGIN
```

การคูณคิดเครื่องหมาย (SIGNED MULTIPLICATION : IMUL)

คำสั่ง IMUL เป็นคำสั่งคูณแบบคิดเครื่องหมาย ในรูป 12-3 โปรแกรมในส่วนของ D10IMUL ใช้เหมือนกับ 3 ตัวอย่างของ C10MUL โดยการแทนที่ MUL ด้วยคำสั่ง IMUL

คำสั่ง IMUL คำสั่งแรกเป็นการคูณ 80H (ค่าติดลบ) คูณด้วย 40H (ค่าบวก) ผลคูณเก็บใน AX คือ E000H การใช้ข้อมูลเหมือนกับคำสั่ง MUL ที่ให้ผลลัพธ์ 2000H ดังนั้นท่านสามารถเห็นความแตกต่างในการใช้คำสั่ง MUL กับ IMUL คำสั่ง MUL ค่าของ 80H หรือ +128 ขณะที่คำสั่ง IMUL ค่าของ 80H หรือ -128 ผลคูณของ -128 X (+64) เท่ากับ -8192 ซึ่งมีค่าเท่ากับ E000H (โดยการเปลี่ยนค่า E000 เป็นตรงข้าม และบวก 1 นำไปบวกกับตัวตั้ง)

คำสั่ง IMUL จากตัวอย่างการคูณ 8000H (ค่าลบ) คูณด้วย 2000H (ค่าบวก) ผลคูณเก็บไว้ใน DX:AX คือ F000 0000 และผลคูณที่เป็นค่าติดลบ

คำสั่ง IMUL คำสั่งที่ 3 ในตัวอย่าง BYTE1 ได้ขยายเป็นเวิร์ดใน AX แต่ค่า

ที่สมมุติต้องคิดเครื่องหมาย ตัวอย่างใช้คำสั่ง CBW ในการขยายบิตเครื่องหมายซ้ายสุดใน AH ค่าของ 80H ใน AL ก็จะกลายเป็น FF80H ใน AX แต่ตัวคูณ WORD1 ก็เป็นค่าติดลบ ผลคูณก็จะเป็นค่าบวก ผลคูณจะได้ 0040 0000H ใน DX:AX ผลลัพธ์เหมือนกับคำสั่ง MUL ซึ่งสมมุติตัวตั้งและตัวคูณเป็นค่าบวก

จากตัวอย่างที่ได้ ถ้าตัวตั้งและตัวคูณมีบิตเครื่องหมายเหมือนกัน คำสั่ง MUL และคำสั่ง IMUL จะให้ผลลัพธ์เหมือนกัน แต่ถ้าตัวตั้งและตัวคูณมีบิตเครื่องหมายต่างกัน คำสั่ง MUL ให้ค่าบวก และคำสั่ง IMUL จะให้ค่าลบ

EXTENDED MULTIPLICATION OPERATIONS

โปรเซสเซอร์ 80186-80486 จะมีรูปแบบของคำสั่ง IMUL อีก 2 ชนิด สำหรับโอเปอรานด์แบบ IMMEDIATE ท่านสามารถใช้ข้อมูลแบบมีเครื่องหมายหรือไม่มีเครื่องหมายในการคูณแต่ให้ผลลัพธ์เหมือนกัน ค่าของมันจะต้องมีขนาดเหมือนกัน16 บิตหรือ 32 บิต รูปแบบของการคูณ 16 บิตมีดังนี้

IMUL	REGISTER ,IMMEDITAE

ข้อมูลของรีจีสเตอร์เป็นตัวตั้ง และค่าของ IMMEDITATE เป็นตัวคูณ ผลคูณจะเก็บไว้ในรีจีสเตอร์ ถ้าผลคูณมีค่ามากกว่ารีจีสเตอร์ แฟลกตัวทดและโอเวอร์โฟล์วแฟลกจะถูกเซท

รูปแบบการคูณ 32 บิต มีโอเปอรานด์ 3 ตัว

IMUL	REG,MEMORY,IMMEDITAE

ข้อมูลโอเปอรานด์ 2 ตัวเป็นตัวตั้ง ข้อมูลโอเปอรานด์ตัวที่ 3 เป็นตัวคูณและข้อมูลของโอเปอรานด์ 1 จะให้ผลลัพธ์

โปรเซสเซอร์ 80186-80486 ยังมีรูปแบบของคำสั่ง IMUL สำหรับ 16 บิตหรือ 32 บิต

IMUL	REG , REG, MEMORY

ข้อมูลรอเปอรานด์ 1 ตัวจะเป็นตัวตั้งและข้อมูลรอเปอรานด์ 2 ตัวเป็นตัวคูณ ผลคูณอยู่ในรอเปอรานด์ที่ 1

SHIFTING

ถ้าท่านคูณทีละ 2 เท่า (2,4,8...) การเลื่อนจะทำให้มีประสิทธิภาพมากกว่าในการเลื่อนทางซ้ายตามต้องการของบิต สำหรับ 8088/8086 การเลื่อนที่มีค่ามากกว่า 1 ข้อมูลนั้นจะต้องเก็บค่าใน CL ดังตัวอย่างต่อไปนี้

ใช้ AX เป็นตัวตั้ง การคูณ 2 (เลื่อนซ้าย 1 บิต) SHL AX,01

การคูณ 8 (เลื่อนซ้าย 3 บิต) MOV CL,03 ;88/86

SHL AX,CL

การคูณ 8 (เลื่อนซ้าย 3 บิต) SHL AX,03 ;386,486

การคูณหลายเวิร์ด (MULTIWORD MULTIPLICATION)

การคูณที่สะดวกรวมทั้งการคูณไบต์ต่อไบต์เวิร์ดต่อเวิร์ด หรือสองเวิร์ดต่อสองเวิร์ดซึ่งค่าสูงสุดในเวิร์ดคือ +32,767 การคูณด้วยค่าที่มากกว่าเวิร์ดใน 386 รวมทั้งขั้นตอนของการบวก เราจะใช้การคูณแต่ละเวิร์ดแยกออกจากกัน และบวกแต่ละผลลัพธ์เข้าด้วยกัน พิจารณาจากการคูณเลขฐานสิบ

1365

x 12

16380

ถ้าท่านคูณเลขเพียง 2 ตัวจะทำอย่างไร ถ้าท่านคูณ 13 และ 65 ด้วย 12 แยกการทำดังนี้

13	65
x 12	x 12
156	780

ต่อไปเราก็จะเอาผลคูณทั้งสองมาบวกกัน แต่อย่าลืมว่า 13 คือหลักร้อย ผลคูณที่แท้จริงคือ 15600

```
  15600
+   780
  16380
```

ภาษาแอสเซมบลี้สามารถใช้เทคนิคเดียวกัน ยกเว้นว่าข้อมูลประกอบเป็นเวิร์ด (4 ตัวเลข) ในรูปของเลขฐานสิบหก

DOUBLE WORD BY WORD

จากโปรแกรมย่อย E10XMUL รูป 12-4 เป็นการคูณ 2 เวิร์ดด้วยค่า 1 เวิร์ด ตัวตั้ง MULTICND ประกอบด้วยข้อมูล 2 เวิร์ด คือ 3206H และ 2521H เหตุที่ต้องใช้ DW 2 ตัวแทนการใช้ DD เพื่อสะดวกในการใช้คำสั่ง MOV เพราะว่าคำสั่ง MOV เคลื่อนย้ายข้อมูลได้ 16 บิต ไปเก็บที่ AX ที่ที่กำหนดเราเห็นว่ามันจะสลับตำแหน่ง และตัวแอสเซมเบลอร์จะเก็บค่าเหล่านั้นสลับกัน เช่น MULTCND ซึ่งจะมีค่า 32062521H จะเก็บไว้ดังนี้ 21250632H

ตัวคูณ MULTIPLR+2 จะมีข้อมูล 6400H ฟิลด์สำหรับเก็บผลคูณคือ PRODUCT ซึ่งจะแบ่งออกเป็น 3 เวิร์ด คำสั่ง MUL คำสั่งแรกจะคูณด้วย MULTPLR+_2 และเวิร์ดซ้ายของ MULTCND จะเก็บผลคูณ 0E80E400H เก็บใน PRODUCT+2 และ PRODUCT+4 คำสั่ง MUL ตัวที่ 2 จะคูณ MULTPLR+2 และเวิร์ดขวาของ MULTCND จะเก็บค่า 138A5800H ผลบวกของผลคูณทั้งสองมีดังนี้

```
PRODUCT 1:      0000  0E80  E400
PRODUCT 2:    + 138A  5800
TOTAL           138A  6680  E400
```

การบวกครั้งแรกใช้คำสั่ง ADD และตัวบวกครั้งที่สองใช้คำสั่ง ADC เพราะข้อมูลตัวเลขในในเก็บจะสลับไบต์กัน PRODUCT จะมีข้อมูล 00E4 8066 8A13 ก่อนการคูณ PRODUCT จะต้องเคลียเป็น 0

DOUBLEWORD BY DOUBLEWORD

การคูณแบบ 2 เวิร์ดด้วยค่า 2 เวิร์ดใช้ใน 80386 มีการคูณ 4 ชนิด

ตัวตั้ง		ตัวคูณ
WORD 2	*	WORD 2
WORD 2	*	WORD 1
WORD 1	*	WORD 2
WORD 1	*	WORD 1

ท่านสามารถบวกผลคูณแต่ละตัวใน DX และ AX เพื่อเก็บผลคูณครั้งสุดท้ายในรูป 12-4 F10XMUL จะเป็นตัวอย่างของ MULTCND ซึ่งมีข้อมูล 32062521H MULTPLR จะมีข้อมูล 64000A26H และผลคูณจะมีขนาด 4 เวิร์ด

```
TITLE     EXDWMUL (COM) Multiplication of doublewords
          .MODEL  SMALL
          .CODE
          ORG          100H
BEGIN:    JMP          SHORT MAIN
; ----------------------------------------------
MULTCND   DW           2521H                ;Data items
          DW           3206H
MULTPLR   DW           0A26H
          DW           6400H
PRODUCT   DW           0
```

```
            DW          0
            DW          0
            DW          0
; ---------------------------------------------
MAIN        PROC    NEAR                    ;Main procedure
            CALL    E10XMUL                 ;Call 1st multiply
            CALL    Z10ZERO                 ;Clear product
            CALL    F10XMUL                 ;Call 2nd multiply
            MOV         AH,4CH
            INT         21H                 ;Exit
MAIN        ENDP
;                       Doubleword x word:
;                       ------------------
E10XMUL      PROC
            MOV         AX,MULTCND          ;Multiply left word
            MUL         MULTPLR+2           ;  of multiplicand
            MOV         PRODUCT,AX          ;Store product
            MOV         PRODUCT+2,DX

            MOV         AX,MULTCND+2        ;Multiply right word
            MUL         MULTPLR+2           ;  of multiplicand
            ADD         PRODUCT+2,AX        ;Add to stored product
            ADC         PRODUCT+4,DX
            RET
E10XMUL      ENDP
;                       Doubleword x doubleword:
;                       ------------------------
F10XMUL      PROC
            MOV         AX,MULTCND              ;Multiplicand word 1
            MUL         MULTPLR                 ;  x multiplier word 1
            MOV         PRODUCT+0,AX            ;Store product
            MOV         PRODUCT+2,DX
```

```
                MOV     AX,MULTCND      ;Multiplicand word 1
                MUL     MULTPLR+2       ;  x multiplier word 2
                ADD     PRODUCT+2,AX    ;Add to stored product
                ADC     PRODUCT+4,DX
                ADC     PRODUCT+6,00    ;Add any carry
                MOV     AX,MULTCND+2    ;Multiplicand word 2
                MUL     MULTPLR         ;  x multiplier word 1
                ADD     PRODUCT+2,AX    ;Add to stored product
                ADC     PRODUCT+4,DX
                ADC     PRODUCT+6,00    ;Add any carry
                MOV     AX,MULTCND+2    ;Multiplicand word 2
                MUL     MULTPLR+2       ;  x multiplier word 2
                ADD     PRODUCT+4,AX    ;Add to product
                ADC     PRODUCT+6,DX
                RET
F10XMUL         ENDP

;                               Clear product area:
;                               -------------------
Z10ZERO         PROC
                MOV     PRODUCT,0000    ;Clear words
                MOV     PRODUCT+2,0000  ;  left to right
                MOV     PRODUCT+4,0000
                MOV     PRODUCT+6,0000
                RET
Z10ZERO         ENDP
                END     BEGIN
```

ดูจากลอจิกจะมีลักษณะเหมือนกับการคูณ2 เวิร์ดด้วยค่า 1เวิร์ดจากคำสั่ง ADD/ADC มีดังนี้คำสั่ง ADC จะบวก 0 เข้ากับ PRODUCT คำสั่ง ADC คำสั่งแรกจะบวกพร้อมตัวทดซึ่งจะมีคำสั่งในการเคลียตัวทดคำสั่ง ADC คำสั่งที่สองจะบวก ถ้ามีตัวทดและบวก 0 ไม่มีตัวทด ผลลัพธ์ของ PRODUCT ครั้งสุดท้ายจะได้ 138A 687C 8E5C CCE6H เก็บไว้ใน PRODUCT ซึ่งจะสลับไบท์

SHIFTING THE DX:AX REGISTERS

จากการทำงานของการเลื่อนผลคูณใน DX:AX เลื่อนไปทางซ้ายหรือขวาจะเป็นวิธีที่ดีกว่า แต่ตัวอย่างต่อไปนี้จะต้องใช้เลื่อนทุกๆรอบใน CX สังเกตว่าการเลื่อน 1 บิตเพื่อเซ็ทแฟลกตัวทด

```
; SHIFT LEFT 4 BITS
          MOV     CX,04           ;INITIALIZE FOUR LOOPS
C20:      SHL     DX,1            ;SHIFT DX
          SHL     AX,1            ;SHIFT AX
          ADC     DX,00           ;ADD DX CARRY, IF ANY
          LOOP    C20             ;REPEAT
;SHIFT RIGHT 4 BITS
          MOV     CX,04           ;INITIALIZE 4 LOOPS
D20:      SHR     AX,01           ;SHIFT AX
          SHR     DX,01           ;SHIFT DX
          JNC     D30             ; IF DX CARRY
          OR      AH,1000,0000B   ;INSERT 1-BIT IN AH
D30:      LOOP    D20             ;REPEAT
```

วิธีการแบบนี้มีประสิทธิภาพสูงสำหรับการเลื่อนไปทางซ้ายโดยไม่ต้องใช้LOOP เช่น เก็บค่าตัวเลื่อนไว้ใน CL กำหนดการเลื่อน 4 บิต ดังนี้

```
MOV     CL,04       ;SET SHIFT
SHL     DX,CL       ;SHIFT DX LEFT 4 BITS
MOV     BL,AH       ;STORE AH IN BL
SHL     AX,CL       ;SHIFT AX LEFT 4 BITS
SHR     BL,CL       ;SHIFT BL RIGHT 4 BITS
OR      DL,BL       ;INSERT BL 4 BITS IN DL
```

การหาร DIVISION

การหาร (DIV) ใช้คำสั่ง DIV แบบไม่คิดเครื่องหมายและใช้คำสั่ง IDIV แบบคิดเครื่องหมาย จะมีรูปแบบคำสั่งดังนี้

DIV/IDIV	REG / MEMORY

พื้นฐานของการหาร คือ ไบท์ในค่าเวิร์ด เวิร์ดในค่า 2 เวิร์ดและ 2 เวิร์ดในค่า 4 เวิร์ด (80386/80486)

การเปลี่ยนไบท์เป็นเวิร์ด (BYTE INTO WORD)

ตัวตั้งอยู่ใน AX และตัวหารมีค่าเป็นไบท์อยู่ในรีจีสเตอร์หรือหน่วยความจำ หลังจากการหารเศษอยู่ใน AH และผลหารอยู่ใน AL ผลหารขนาด 1 ไบท์เป็นค่าเล็กที่สุด ค่าสูงสุดของไบท์ คือ 225 หรือ FFH ถ้าไม่คิดเครื่องหมาย แต่ถ้าคิดเครื่องหมาย +127 หรือ 7FH มีการคำนวณดังนี้

BEFORE:

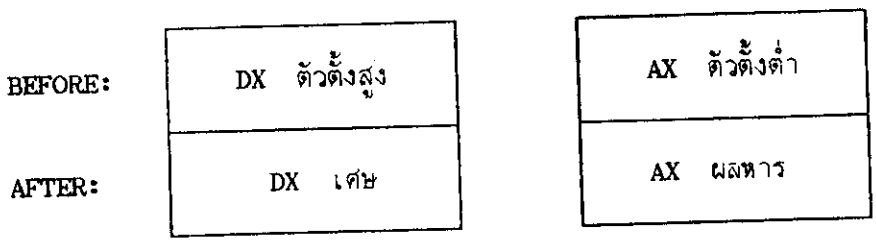
AX ตัวตั้ง	

AFTER:

AL เศษ	AL ผลหาร

การเปลี่ยนเวิร์ดเป็นสองเวิร์ด (WORD INTO DOUBLEWORD)

ตัวตั้งอยู่ใน DX:AX และตัวหารเป็นเวิร์ดอยู่ในรีจีสเตอร์หรือหน่วยความจำ หลังจากการหารเศษอยู่ใน DX และผลหารอยู่ใน AX ผลหารมีขนาด 1 เวิร์ดค่าสูงสุด +32,767 หรือ FFFFH ไม่คิดเครื่องหมายและ +16383 (7FFFH) คิดเครื่องหมาย



การเปลี่ยนสองเป็นแปดเวิร์ด (DOUBLEWORD INTO QUADWORD)

ตัวตั้งอยู่ใน EDX:EAX และตัวหารขนาด 2 เวิร์ดอยู่ในรีจีสเตอร์หรือหน่วยความจำ หลังจากการหราเศษอยู่ใน EDX ผลหารอยู่ใน EAX

ขนาดของฟิลด์ (FIELD SIZE)

ตัวรอเปอรานด์ DIV หรือ IDIV จะอ้างถึงการหาร พิจารณาจากคำสั่ง DIV

ตัวหาร (DIVISOR)

การกำหนดการหาร	สมมุติการทำงาน
DB	BYTE INTO WORD
DW	WORD INTO DOUBLEWORD
DD	DOUBLE INTO QUADWORD

เมื่อตัวหารอยู่ในรีจีสเตอร์ ขนาดของรีจีสเตอร์เป็นตัวกำหนดการทำงาน

ตัวทำงาน	การหาร	ตัวตั้ง	ผลหาร	เศษ
DIV CL	BYTE	AX	AL	AH
DIV CX	WORD	DX:AX	AX	DX
DIV EBX	DOUBLEWORD	EDX:EAX	EAX	EDX

เศษ (REMAINDER)

ถ้าหาร 13 ด้วย 3 ผลลัพธ์ 4 1/3 ผลหารคือ 4 และ เศษที่แท้จริงคือ 1 สังเกตจากการคำนวณใช้ภาษาเบสิกจะให้ผลลัพธ์ 4.333 ค่าของมันจะเป็นเลขจำนวนเต็ม 4 และเศษส่วน .333

การหารไม่คิดเครื่องหมาย (UNSIGNED DIVISION : DIV)

คำสั่ง DIV จะเป็นการหารแบบไม่คิดเครื่องหมาย ในรูป 12-5 โปรแกรม D1ODIV จะมี 4 ตัวอย่าง BYTE INTO WORD , WORD INTO DOUBLEWORD , BYTE INTO BYTE และ WORD INTO WORD คำสั่งหารคำสั่งแรกจะเป็นการหาร 2000H ด้วยค่า 80H เศษอยู่ใน AH=00H ผลหารอยู่ใน AL=40H

คำสั่ง DIV คำสั่งที่ 2 เป็นการขยาย BYTE 1 เป็น WORD เป็นค่าที่ไม่คิดเครื่องหมาย ตัวอย่าวสมมุติบิตซ้ายสุดใน AH มีค่าเป็น 0 เศษจะอยู่ใน AH คือ 12 และผลหารอยู่ใน AL=05H

คำสั่ง DIV คำสั่งที่ 3 เศษที่อยู่ใน DX=1000H และผลหารอยู่ใน AX=0080H

คำสั่ง DIV คำสั่งที่ 4 เป็นการขยาย WORD 1 เป็น DOUBLEWORD ใน DX หลังจากการหารเศษอยู่ใน DX=0000H และผลหารอยู่ใน AX=0002H

```
TITLE     EXDIV (COM)  DIV and IDIV operations
          .MODEL  SMALL
          .CODE
          ORG          100H
BEGIN:    JMP          SHORT MAIN
; ---------------------------------------------
BYTE1     DB           80H                    ;Data items
BYTE3     DB           16H
WORD1     DW           2000H
WORD2     DW           0010H
WORD3     DW           1000H
; ---------------------------------------------
MAIN      PROC   NEAR                         ;Main procedure
            CALL   D10DIV                     ;Call DIV  routine
            CALL   E10IDIV                    ;Call IDIV routine
            MOV          AH,4CH
            INT          21H                  ;Exit
MAIN      ENDP
;                        Examples of DIV:
;                        ----------------
D10DIV      PROC
            MOV          AX,WORD1             ;Word / byte
            DIV          BYTE1                ;  rmdr:quot in AH:AL
            MOV          AL,BYTE1             ;Byte / byte
            SUB          AH,AH                ;  extend dividend in AH
            DIV          BYTE3                ;  rmdr:quot in AH:AL

            MOV          DX,WORD2             ;Doubleword / word
            MOV          AX,WORD3             ;  dividend in DX:AX
            DIV          WORD1                ;  rmdr:quot in DX:AX
```

```
                MOV         AX,WORD1        ;Word / word
                SUB         DX,DX           ;  extend dividend in DX
                DIV         WORD3           ;  rmdr:quot in DX:AX
                RET
D10DIV          ENDP
;               Examples of IDIV:
;               ----------------
E10IDIV         PROC
                MOV         AX,WORD1        ;Word / byte
                IDIV    BYTE1               ;  rmdr:quot in AH:AL
                MOV         AL,BYTE1        ;Byte / byte
                CBW                         ;  extend dividend in AH
                IDIV    BYTE3               ;  rmdr:quot in AH:AL

                MOV         DX,WORD2        ;Doubleword / word
                MOV         AX,WORD3        ;  dividend in DX:AX
                IDIV    WORD1               ;  rmdr:quot in DX:AX
                MOV         AX,WORD1        ;Word / word
                CWD                         ;  extend dividend in DX
                IDIV    WORD3               ;  rmdr:quot in DX:AX
                RET
E10IDIV         ENDP

                END         BEGIN
```

SIGNED DIVISION : IDIV

คำสั่ง IDIV เป็นการหารแบบคิดเครื่องหมาย ในรูป 12-5 โปรแกรม E10IDIV ใช้ตัวอย่างที่เหมือนกัน 4 ตัวอย่าง คือ D10IDIV ซึ่งแทนตัวค่า IDIV แทนคำสั่ง DIV คำสั่ง IDIV คำสั่งแรกเป็นการหาร 2000H(ค่าบวก) ด้วยค่า 80H(ค่าลบ) เศษเก็บใน AH คือ 00H และผลหารเก็บใน AL เท่ากับ C0H (-64) (ส่วนโปรแกรมที่ใช้คำสั่ง DIV ผลหารเท่ากับ +64)

ผลลัพธ์ในรูปฐานสิบหกในการใช้ IDIV 3 ตัวอย่าง ดังนี้

ตัวอย่าง IDIV	เศษ	ผลหาร
2	EE = (-18)	FB(-5)
3	1000 (4096)	0080 (128)
4	0000	0002

ส่วนตัวอย่างที่ 4 เหมือนกับการใช้คำสั่ง DIV ถ้าตัวตั้งและตัวหารมีบิทเครื่องหมายเหมือนกัน คำสั่ง DIV และ IDIV จะให้ผลลัพธ์เหมือนกัน แต่ถ้าตัวตั้งและตัวหารเครื่องหมายต่างกัน คำสั่ง DIV จะให้ผลหารเป็นค่าบวกและ IDIV เป็นค่าลบ

การเลื่อน (SHIFTING)

ถ้าการหารเป็นลักษณะของ POWER OF 2 (2,4,8...) การเลื่อนจะมีประสิทธิภาพมากกว่า โดยการเลื่อนไปทางขวา สำหรับ 8088/8086 การเลื่อนมากกว่า 1 บิตจะต้องกำหนดค่าใน CL ดังตัวอย่างต่อไปนี้ สมมุติการหารใน AX

```
หาร 2 (เลื่อนขวา 1 บิต )    SHR     AX,01
หาร 8 (เลื่อนขวา 3 บิต)     MOV     CL,03       ;8088/8086
                            SHR     AX,CL
หาร 8 (เลื่อนขวา 3 บิต)     SHR     CL,03       ;80186-80486
```

โอเวอร์โฟวล์และอินเตอร์รัพท์ (OVERFLOWS AND INTERRUPTS)

การทำงานของคำสั่ง DIV และ IDIV อาจเกิดโอเวอร์โฟลว์ได้ง่าย จะทำให้การขัดจังหวะของผลลัพธ์ การทำงานผลหารจะมีความสำคัญน้อยกว่าตัวตั้งเดิม การหารด้วย 0 จะเกิดการขัดจังหวะ แต่การหารด้วย 1 จะให้ผลหารเหมือนกับตังตั้ง

การใช้ประโยชน์ของกฎนี้ ถ้าเป็นการหารขนาด 1 ไบท์ ข้อมูลอาจจะใหญ่กว่าไบท์ทางซ้ายมือ (AH) ของตัวตั้ง ถ้าตัวหารมีขนาดเป็นเวิร์ด ข้อมูลจะอาจจะใหญ่กว่าเวิร์ดทางซ้ายมือ (DX) ของตัวตั้ง แสดงการใช้ตัวหาร

Dvide Operation	Dividend	Dvisor	Quotient
Word by byte:	0123	01	(1)23
Doubleword by word	0001 4026	0001	(1)4026

ทั้งสองกรณีผลหารจะใหญ่เกินกว่าพื้นที่ที่กำหนด จากตัวอย่าง DIVBYTE คือตัวหาร 1 ไบท์ และตัวตั้งอยู่ใน AX

```
CMP   AH,DIVBYTE          ;COMPARE AH TO DIVISOR
JNB   OVERFLOW-RTNE       ;EXIT IF NOT SMALLER
DIV   DIVBYTE             ;DIVIDE WORD BY BYTE
```

ตัวอย่างที่ 2 ของ DIVWORD คือตัวหารขนาด 1 เวิร์ด และตัวตั้งอยู่ที่ DX:AX

```
CMP   DX,DIVWORD          ;COMPARE DX TO DIVISOR
JNB   OVERFLOW-RTNE       ;EXIT IF NOT SMALLER
DIV   DIVWORD             ;DIVIDE DOUBLEWORD BY BYTE
```

การหารโดยการลบ (DIVISION BY SUBTRACTION)

ถ้าผลหารใหญ่มากสำหรับตัวหาร ท่านสามารถใช้รูปแบบวิธีการลบ นั่นคือ ลบตัวหาร

จากตัวตั้ง และเพิ่มค่าของผลหารที่ละ 1 และทำการลบต่อไป จนกระทั่งตัวตั้งน้อยกว่าตัวหาร ดังตัวอย่างต่อไปนี้ ตัวตั้งอยู่ใน AX และตัวหารอยู่ใน BX ผลหารเก็บไว้ใน CX

```
        SUB   CX,CX          ;CLEAR QUOTIENT
C20:    CMP   AX,BX          ;IF DIVIDEND < DIVISOR
        JB    C30            ;EXIT
        SUB   AX,BX          ;SUBTRACT DIVISOR FROM DIVIDEND
        INC   CX             ;ADD 1 TO QUOTIENT
        JMP   C20            ;REPEAT
C30:    RET                  ;QUOTIENT IN CX , REMAINDER IN AX
```

ที่จุดสุดท้ายของการทำงาน CX จะเก็บผลหารและ AX จะเก็บค่าเศษ จากตัวอย่างถ้าตัวตั้งอยู่ใน DX:AX จะต้องรวมคำสั่ง 2 คำสั่งต่อไปนี้

1. ที่ C20 เปรียบเทียบ AX กับ BX ถ้า DX เป็น 0
2. หลังคำสั่ง SUB ให้เพิ่มคำสั่ง SBB DX,00

บทสรุป

- ข้อมูลขนาด 1 ไบท์จะมีค่าสูงสุด +127 และ -128
- คำสั่งบวกหลายๆครั้งใช้คำสั่ง ADC ก่อนการบวกใช้คำสั่ง CLC
- คำสั่งคูณคือคำสั่ง MUL , IMUL
- คำสั่งหารคือคำสั่ง DIV , IDIV
- การคูณและการหารในลักษณะ ค่ายกกำลังสอง ใช้คำสั่ง SHR ไม่คิดเครื่องหมาย
- คำสั่ง SAR คิดเครื่องหมาย

แบบฝึกหัด

12-1. ค่าสูงสุดของข้อมูลขนาด 1 ไบท์และค่าต่ำสุดมีค่าเท่าไร

12-2. ค่าสูงสุดของข้อมูลขนาด 1 เวิร์ดและค่าต่ำสุดมีค่าเท่าไร

12-3. จงเขียนโปรแกรมในการคูณข้อมูลที่มีตัวตั้งขนาด 1 ไบท์และตัวคูณขนาด 1 ไบท์ผลคูณเก็บไว้ในหน่วยความจำ

12-4. จงเขียนโปรแกรมในการคูณข้อมูลที่มีตัวตั้งขนาด 1 เวิร์ดและตัวคูณขนาด 1 ไบท์ผลคูณเก็บไว้ในหน่วยความจำ

12-5. จงเขียนโปรแกรมในการคูณข้อมูลที่มีตัวตั้งขนาด 1 เวิร์ดและตัวคูณขนาด 1 เวิร์ดผลคูณเก็บไว้ในหน่วยความจำ

12-6. จงเขียนโปรแกรมในการหารข้อมูลที่มีตัวตั้งขนาด 1เวิร์ดและตัวหารขนาด 1 ไบท์ผลหารเก็บไว้ในหน่วยความจำ

12-7. จงเขียนโปรแกรมในการหารข้อมูลที่มีตัวตั้งขนาด 1 เวิร์ดและตัวหารขนาด 1เวิร์ดผลหารเก็บไว้ในหน่วยความจำ
