บทที่ 7

โปรแกรมลอจิกและการควบคุม

PROGRAM LOGIC AND CONTROL

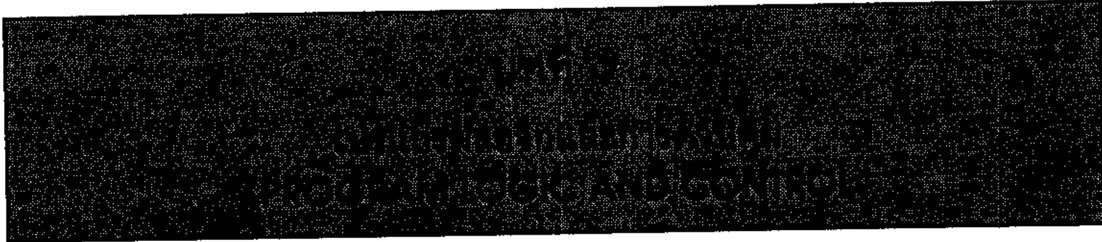
วัตถุประสงค์

หลังจากที่ท่านศึกษาบทนี้แล้วท่านจะมีความเข้าใจดังต่อไปนี้

- การกำหนดแอดเดรส SHOORT , NEAR , FAR
- การใช้คำสั่ง JMP , LOOP
- การใช้คำสั่ง CONDITION JUMP
- การเรียกโพรซีเยอร์
- การทำงานของคำสั่งบูลีน

CT 215

117



คำสั่งในการเคลื่อนย้ายการควบคุม จากการทำงานของโปรแกรมเรียงตามลำดับ โดยใช้ค่า offset บวกกับค่าของ IP แบ่งการทำงานออกเป็น 4 ชนิด

- UNCONDITIONAL JUMP : JMP
- LOOPING : LOOP
- CONDITIONAL JUMP : Jnnn (เช่น JE,JNH,JL)
- CALL PROCEDURE : CALL

การกำหนดแอดเดรส SHORT, NEAR และ FAR

คำสั่งในการอ้างที่เกี่ยวกับแอดเดรสมี 3 ชนิด คือ SHORT,NEAR,FAR

SHORT ADDRESS คือ การกำหนดดค่าแอดเดรสในช่วง -128 to 127 ไบท์

NEAR ADDRESS คือ การกำหนดค่าแอดเดรสในช่วง -32768 to 32767 ไบท์ ซึ่งอยู่ภายในเซกเมนต์เดียวกัน

FAR ADDRESS คือ ใช้ในการอ้างอิงต่างเซกเมนต์ โดยใช้เซ็ตเป็นแอดเดรสกับค่า offset

ระยะทาง	SHORT	NEAR	FAR
คำสั่ง	-128 to 127 same segment	-32768 to 32767 same segment	ANOTHER SEGMENT
JMP	YES	YES	YES
Jnnn	YES	80386/486	NO
LOOP	YES	NO	NO
CALL		YES	YES

คำสั่ง LABEL

คำสั่ง JMP.Jnnn และ LOOP จะต้องมีโอเปอรานด์ในการอ้างถึง LABEL ของคำสั่ง ดังตัวอย่างคำสั่ง jump ไปที่ A90 ซึ่งเป็น label ของคำสั่ง MOV

```
          JMP      A90
          ---
A90 :     MOV      AH,00
```

ลาเบลของคำสั่ง เช่น A90 จะต้องตามด้วย colon(:) ที่กำหนดโดย attribute NEAR นั่นคือลาเบลที่อยู่ภายใน procedure ใน code segment เดียวกัน หรือท่านสามารถเขียนรูปแบบของคำสั่งที่แตกต่างกัน ดังนี้

```
A90 :
          MOV      AH,00
```

ทั้ง 2 กรณี แอดเดรส A90 จะอ้างถึงไบท์แรกของคำสั่ง MOV

คำสั่งกระโดด JMP INSTRUCTION

ปกติการใช้คำสั่งในการควบคุมการเคลื่อนย้าย คือ คำสั่ง JMP คำสั่งนี้คือการกระโดดแบบไม่มีเงื่อนไข ในการทำงานการเคลื่อนย้าย คำสั่ง JMP จะทำให้ตัวโปรเซสเซอร์เตรียมการทำงานของ queue มีรูปแบบดังนี้

[LABEL]	JMP	SHORT, NEAR, or FAR ADDRESS

การทำงานของคำสั่ง JMP ภายในเซ็คเมนต์เดียวกัน อาจเป็น SHORT หรือ NEAR ขั้นแรกในการเขียน source program แล้วส่งให้ assembler คอมไพล์หาความยาวของแต่ละคำสั่ง อย่างไรก็ตามคำสั่ง JMP อาจมีความยาว 2 หรือ 3 ไบท์ การทำงานของคำสั่ง JMP นี้ ลาเบลของคำสั่งจะอยู่ภายใน -128 to +127 ไบท์ คือการ SHORT JUMP ตัวแอสเซมเบลอร์จะกำหนด 1 ไบท์ สำหรับการทำงาน (EB) ที่มีโอเปอรานด์มีขนาด 1 ไบท์ โอเปอรานด์นี้เป็นค่า offset ของคอมพิวเตอร์บวกกับค่า IP ในการ executeโปรแกรม ค่าของ offset จะอยู่ในช่วง 00H ถึง FFH หรือ -128 to +127 แอสเซมเบลอร์สามารถให้กระโดดกลับหลังได้

```
A50 :
        ---
        JMP     A50
```

ในกรณีที่แอสเซมเบลอร์จ่ายรหัสคำสั่งชนิด 2 ไบท์ คำสั่ง JMP จะกระโดดได้เกิน -128 to + 127 ไบท์ จะเป็นชนิด NEAR JUMP ซึ่งใช้รหัสภาษาเครื่องเป็น (E9) และซึ่งเป็นโอเปอรานด์ 2 ไบท์ (8086/80286) หรือขนาด 4 ไบท์(386/486) ในการกระโดดไปข้างหน้า

```
        JMP     A90
        ---
A90 :
```

การกระโดดนี้แอสเซมเบลอร์จะไม่ทราบว่าเป็น SHORT หรือ NEAR มันจะจ่ายหรือการทำงานโดยอัตโนมัติของคำสั่ง 3 ไบท์ อย่างไรก็ตามเราก็สามารถกำหนดได้โดยใช้

SHORT OPERATOR

```
          JMP     SHORT A90
          ---
     A90 :
```

ตัวอย่าง 7.1

```
JMP    L1              ;NEAR DEST IN CURRENT ADDRESS
JMP    NEAR PTR L1     ;NEAR DEST IN CURRENT ADDRESS
JMP    SHORT NEXTVAL;SHORT DEST WITH IN -128 TO +127 BYTE
JMP    FAR PTR ERROR_PTR ;FAR DEST IN ANOTHER SEGMENT
```

ตัวอย่างโปรแกรมใช้คำสั่ง JMP

โปรแกรมที่มีนามสกุล COM ในตัวอย่าง 7.1 แสดงการใช้คำสั่ง JMP โดยการกำหนดสภาวะเริ่มแรกของ AX,BX,CX ให้มีค่าเป็น 1 และใช้รูปแบบการทำงานของ LOOP ดังต่อไปนี้

ADD 1 TO AX

ADD AX TO BX

DOUBLE THE VALUE IN CX

การทำงานของการสิ้นสุดแต่ละ LOOP คำสั่ง JMP A20 จะเคลื่อนย้ายการควบคุมไปยังคำสั่งที่มีลาเบล A20 ผลของการทำงานซ้ำๆจะทำให้ค่าของ AX เพิ่มขึ้นทีละ 1 เป็น 1,2,3,4.. ค่าของ BX จะเพิ่มขึ้นตามผลบวกของตัวเลข 1,3,6,10,.. และค่า CXจะเป็นค่า DOUBLE คือ 1,2,4,8,.. การทำงานของ LOOP นี้ไม่มีสิ้นสุด <u>ไม่ใช่</u>ความคิดที่ดี

ในกรณีของโปรแกรมนี้ ลาเบล A20 จะกระโดดเพียง 9 ไบท์ ท่านสามารถตรวจสอบระยะการกระโดดจากรหัสภาษาเครื่องสำหรับ JMP:EBF7 ค่าของ EB คือรหัสภาษาเครื่องสำหรับ NEAR JMP และ F7 เป็นค่า -9 คือใช้หลักการ 2's COMPLEMENT การทำงานของคำสั่ง JMP เป็นการบวกค่า F7 กับค่าของ IP ซึ่งมีข้อมูล 0112 ดังนี้

	DEC	HEX	
IP	274	0112	
JMP OPERAND	-9	FFF7	2's
TRANSFER ADDRESS	265	(1) 0109	

การคำนวณแอดเดรสของคำสั่ง JMP 0109H ผลการคำนวณจะมีตัวทดออกเป็น 1 ให้ตัดทิ้งไป โดยการตรวจสอบโปรแกรมสำหรับตัว offset address ของ A20 ในทางตรงข้าม โอเปอรานด์ของการกระโดดไปด้านหน้า สำหรับคำสั่ง JMP เป็นค่าบวก

ตัวอย่าง 7:2

```
START:   MOV  AH,2
         MOV  DL,'A'
         INT  21H
         JMP  START
```

โปรแกรมจะทำงานอย่างต่อเนื่องถ้าต้องการให้หยุดให้กด CTRL-BREAK

```
                         PAGE 60,132
                  TITLE  EXJUMP (COM) ILLUSTRATE JMP FOR LOOPING
                         .MODEL SMALL
                         .CODE
0000            MAIN     PROC        NEAR
0000 B8 0001             MOV         AX,01   ;INITIALIZE AL
0003 BB 0001             MOV         BX,01   ; BX,AND
0006 B9 0001             MOV         CX,01   ; CX TO 01
0009            A20 :
0009 05 0001             ADD         AX,01   ; ADD 01 TO AX
000C 03 D8               ADD         BX,AX   ; ADD AX TO BX
000E D1 E1               SHL         CX,1    ; DOUBLE CX
0010 EB F7               JMP         A20     ; JUMP TO A20
0012            MAIN     ENDP
                         END         MAIN
```

รูป 7-1 USE OF THE JMP INSTRUCTION

ตัวอย่าง 7.3 การใช้คำสั่ง JMP ในการเขียนโปรแกรมแสดงชุดรหัส ASCII

```
            TITLE  IBM CHARACTER DISPLY
            .MODEL SMALL
            .STACK 100H
            .CODE
        MAIN PROC
            MOV  AH,2          ;DISPLY CHARACTER FUNCTION
            MOV  CX,256        ;No. OF CHAR TO DISPLAY
            MOV  DL,2          ;DL HAS ASCII CODE
PRINT_LOOP:
            INT  21H           ;DISPLAY
            INC  DL            ;INCREMENT ASCII
            DEC  CX            ;DECREMENT COUNT
            JNZ  PRINT_LOOP
;DOS EXIT
            MOV  AH,4CH
            INT  21H
MAIN        ENDP
            END  MAIN
```

คำสั่ง LOOP

คำสั่ง JMP ที่ใช้ในตัวอย่าง 7-1 จะต้องมีการกำหนด LOOP ว่าจะต้องมีการวนรอบกี่ครั้งหรือวนรอบครบตามเงื่อนไข คำสั่ง LOOP ซึ่งใช้ตามวัตถุประสงค์นี้ ต้องเซ็ตค่าของ CX (count)ในการทำงานแต่ละครั้ง คำสั่ง LOOP จะลดค่า CX โดยอัตโนมัติทีละ 1 ถ้า CX เป็น 0 ก็เป็นสิ้นสุดการทำงานของ LOOP ถ้ายังไม่เท่ากับ 0 จะกระโดดไปที่แอดเดรสของโอเปอรานด์ในช่วงของ SHORT JUMP ที่อยู่ภายใน -128 to +127 สำหรับการทำงานที่เกินค่าที่กำหนด แอสเซมเบลอร์จะส่งข่าวบอกว่า "RELATIVE JUMP OUT OF RANGE" โดยใช้รูปแบบดังต่อไปนี้

[LABEL]	LOOP	SHORT-ADDRESS

โปรแกรมในรูป 7-2 แสดงการใช้คำสั่ง LOOP ซึ่งเป็นรูปแบบเหมือนกับตัวอย่าง 7-1 ยกเว้นการสิ้นสุดของ LOOP หลังจากการทำงาน 10 รอบ คำสั่ง MOV จะเซ็ตค่า CX = 10 การวนรอบจะใช้ค่าข้อมูลใน CX เป็นเคาเตอร์ ในส่วนของโปรแกรมใช้ค่า DX เป็นค่า double ของค่า 1 คำสั่ง LOOP จะกระโดดไปทำซ้ำที่ JMP A20 และเพิ่มค่า INC AX โดยใช้แทนค่า ADD AX,01

คำสั่ง JMP โอเปอรานด์รหัสภาษาเครื่อง จะมีแอดเดรสข้อมูลของปลายทาง คำสั่ง LOOP คือ A20 ซึ่งเป็นการบวกค่าให้กับ IP

จากการเปลี่ยนแปลง ตัวอย่าง 7-1 จะได้ดังต่อไปนี้ ตามตัวอย่าง 7-2 คำสั่ง LOOP จะมีตัวแปรอีก 2 ตัว คือ LOOPE/LOOPZ (LOOP WHILE EQUAL/ZERO) และ LOOPNE/LOOPNZ (LOOP WHILE NOT EQUAL/ZERO) ซึ่งการทำงานทั้ง 2 จะเป็นการลดค่า CX ทีละ 1 คำสั่ง LOOPE จะทำงานอย่างต่อเนื่องถ้า CX <> 0 หรือ 0 ตามเงื่อนไขที่กำหนด LOOPNE จะทำงานต่อเนื่องถ้า CX <> 0 หรือ 0 ตามเงื่อนไข

```
        PAGE    60,132
TITLE   EXLOOP
        .MODEL
        .CODE
        ORG     100H
BEGIN   PROC    NEAR
        MOV     AX,01   ;  INITIALIZE AX
        MOV     BX,01   ;  ,BX
        MOV     DX,01   ;  ,DX TO 01
        MOV     CX,01   ;  NO OF LOOP
A20 :   INC     AX      ;  ADD 01 TO AX
        ADD     BX,AX   ;  ADD AX TO BX
        SHL     DX,1    ;  DOUBLE DX
```

```
        LOOP    A20       ; DECR CX,LOOP IF
                          ; NONZERO
        MOV     AH,4CH    ; TERMINATE
        INT     21H
BEGIN   ENDP
        END     BEGIN
```

รูป 7-2 การใช้คำสั่ง LOOP

ตัวอย่าง7.4 การใช้คำสั่ง LOOP ในการแสดงตัวอักษร A ดังนี้

```
        MOV  CX,12*80    ;SET CONTROL 960
NEXT:
        MOV  AH,2        ;FUNCTION DISPLAY CHARACTERS
        MOV  DL,'A'      ;DISPLAY THE LETTER A
        INT  21H         ;CALL DOS
        LOOP NEXT        ;DECREMENT CX AND REPEAT
```

ตัวอย่าง 7.5 การใช้คำสั่ง LOOP ในการทำสำเนาข้อมูลขนาด 6 ไบท์จาก ARRAY1 ไปยัง ARRAY2

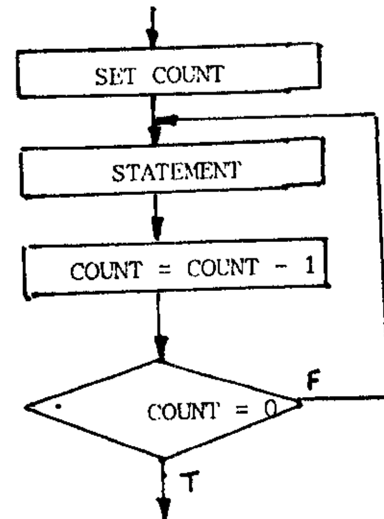
```
            MOV  SI,OFFSET ARRAY1
            MOV  DI,OFFSET ARRAY2
            MOV  CX,6
MOVE_BYTE:
            MOV  AL,[SI]
            MOV  [DI],AL
            INC  SI
            INC  DI
            LOOP MOVE_BYTE

ARRAY1   DB    10H,20H,30H,40H,50H,60H
ARRAY2   DB    6 DUP(?)
```

โครงสร้างการทำงาน LOOP

FOR LOOP

โครงสร้างของ LOOP ชนิดนี้จะเป็นการทำงานซ้ำกันขึ้นอยู่กับจำนวนครั้งของ LOOP

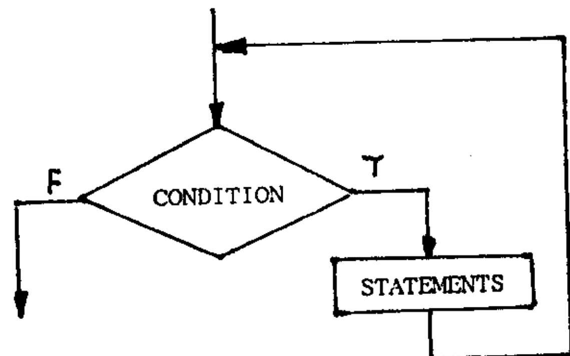
```
FOR    LOOP_COUNT TIMES   DO
       STATEMENTS
END_FOR
```



ตัวอย่าง 7.6 ใช้คำสั่ง LOOP ในการแสดงผล STAR จำนวน 80 ครั้ง

```
        MOV   CX,80
        MOV   AH,2
        MOV   DL,'*'
TOP:
        INT   21H
        LOOP    TOP
```

WHILE LOOP



โครงสร้างของ LOOP ชนิดนี้ขึ้นอยู่กับเงื่อนไขดังนี้

```
FOR  CONDITION  DO
     STATEMENTS
END_WHILE
```

ตัวอย่าง 7.7 จงเขียนโปรแกรมในการนับจำนวนตัวอักขระที่รับเข้ามา มีอัลกอริทึมดังต่อไปนี้

```
INITIALIZE COUNT TO 0
     READ A CHARACTER
          WHILE CHARACTER <> CR DO
          COUNT = COUNT + 1
     READ A CHARACTER
END_WHILE
```

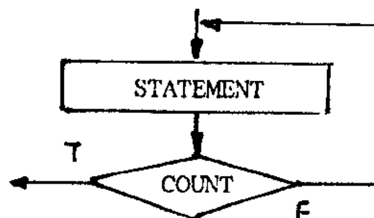
ภาษาแอสเซมบลี

```
                MOV  DX,0        ;DX = COUNTS CHAR
                MOV  AH,1        ;PREPARE TO READ
                INT  21H
     WHILE_:
                CMP  AL,0DH      ;CR
                JE   END_WHILE ; YES EXIT
                INC  DX          ;NOT CR INCREMENT COUNT
                INT  21H
                JMP  WHILE_
     END_WHILE:
```

REPEAT LOOP

โครงสร้างชนิดนี้เป็นการทำงานซ้ำๆ จนกระทั่งพบเงื่อนไข

```
REPEAT
     STATEMENTS
UNTIL CONDITION
```



ตัวอย่าง 7.8 การอ่านตัวอักขระจนกระทั่งพบ BLANK แล้วหยุดทำงาน

	ภาษาแอสแซมบลี
REPEAT	
READ A CHARACTER	
UNTIL CHARACTER IS A BLANK	

```
          MOV  AH,1
REPEAT:   INT  21H
          CMP  AL' '
          JNE  REPEAT
```

แฟลก รีจีสเตอร์ FLAG REGISTER

ในส่วนนี้จะอธิบายรายละเอียดของแฟลก ซึ่งแฟลกรีจิสเตอร์ซึ่งมีขนาด 16 บิต ซึ่งแฟลกจะเป็นตัวเซ็ตสภาวะการทำงานของคำสั่งต่างๆ แฟลกจะจำสภาวะการทำงานเดิมจนกว่าจะมีการเปลี่ยนแปลงของคำสั่งใหม่ แฟลกรีจิสเตอร์มีข้อมูลดังต่อไปนี้

บิต	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
แฟลก					O	D	I	T	S	Z		A		P		C

CF (carry flag) ข้อมูลของ CF (0 หรือ 1) จากบิตที่มีค่าสูงสุดของการคำนวณทางคณิตศาสตร์ และคำสั่ง SHIFT และ ROTATE

PF (parity flag) ข้อมูลที่ได้จากการตรวจสอบ 8 บิตต่ำของข้อมูลว่าข้อมูลแต่ละบิตที่เซ็ตเป็น 1 ถ้าเป็นจำนวนคี่ตัว แฟลกจะเป็น 0, ถ้าจำนวนคู่ตัว แฟลกจะเป็น 1

AF (auxilary carry flag) แฟลกนี้จะเซ็ตเป็น 1 ถ้ามีตัวทดออกจากบิตที่ 3 ไปบิตที่ 4 ของการทำงานในรีจิสเตอร์ขนาด 1 ไบท์ แฟลกนี้จะทำงานในการคำนวณของ ASCII และ BCD PACKED FIELDS

ZF (zero flag) แฟลกนี้จะถูกเซ็ตเป็น 1 ถ้าการคำนวณทางคณิตศาสตร์หรือลอจิกมีค่าเป็น 0, ถ้าไม่เท่ากับ 0 ZF = 1

SF (sign flag) แฟลกนี้จะถูกเซ็ตตามเครื่องหมาย(บิตซ้ายสุด) หลังจากการคำนวณทางคณิตศาสตร์ SF = 0 เป็นบวก, SF = 1 เป็นลบ

TF (trap flag) แฟลกนี้ถูกเซ็ต TF = 1 โปรเซสเซอร์จะexecuteทีละขั้นตอน นั่นคือ 1 คำสั่งใน 1 เวลาตามผู้ใช้ต้องการ ท่านเซ็ตค่าแฟลกเรียบร้อยแล้วใช้คำสั่ง T ใน debug execute

IF (interrupt flag) แฟลกนี้จะ disable interrupts เมื่อ IF=0 และ enable เมื่อ IF=1

DF (direction flag) ใช้กับการทำงานของคำสั่งสตริงที่กำหนดทิศทางการเคลื่อนย้ายข้อมูล DF = 0 จะเพิ่มค่า SI และ DI เคลื่อนข้อมูลจากซ้ายไปขวา DF = 1 จะลดค่า SI และ DI เคลื่อนข้อมูลจากขวาไปซ้าย

OF (overflow flag) เป็นตัวชี้ข้อมูลของตัวทดออกจาก บิตที่มีค่าสูงสุด(LSB) ว่าข้อมูลเกินขนาด OF = 1 ในทางตรงข้าม OF = 0

CONDITIONAL JUMPS

NAME	JUMP IF	FLAG TEST
TESTING FOR ZERO		
JZ	ZERO	ZF = 1
JNZ	NOT ZERO	ZF =0
COMPARING UNSIGNED NUMBER		
JA	ABOVE	CF AND ZF =0
JB	BELOW	CF = 1
JAE	ABOVE OR EQUAL	CF = 0
JBE	BELOW OR EQUAL	CF OR ZF = 1
JNC	NOT CARRY	CF = 0

NAME	JUMP IF	FLAG TEST
COMPARING SIGNED NUMBER		
JG	GREATER	ZF = 0 AND SF =OF
JL	LESS	SF <> OF
JGL	GREATER OR EQUAL	SF = OF
JLE	LESS OR EQUAL	ZF = 1 OR SF <> OF
TESTING FOR OVERFLOW		
JO	OVERFLOW	OF = 1
JNO	NOT OVERFLOW	OF = 0
TESTING SIGNS		
JS	SIGN	SF = 1
JNS	NO SIGNS	SF = 0
TESTING PARITY		
JPO	ODD PARITY	PF = 0
JPE	EVEN PARITY	PF = 1
CHECKING CX FOR ZERO WITHOUT IN SPECING THE FLAGS		
JCXZ	CX = 0	NONE

ตัวอย่าง 7.9 คำสั่ง CMP เป็นการเปรียบเทียบโอเปอรานด์ 2 ตัว และผลจะเกิดขึ้นที่แฟลก AF.CF,OF,PF,SF และ ZF แต่ท่านไม่จำเป็นต้องตรวจสอบแฟลกเหล่านั้นทั้งหมด ดังการตรวจสอบ BX

```
        CMP   BX,00   ; COMPARE BX กับ 0
        JZ    B50
        ---
        ---
B50 :   ---           ; JUMP POINT IF BX=0
```

ถ้าข้อมูล BX = 0 คำสั่ง CMP จะเซ็ตค่า ZF = 1 อาจจะหรือไม่มีการเปลี่ยนแปลงแฟลก ZF คำสั่ง JZ เป็นการตรวจสอบแฟลก ZF

คำสั่งกระโดดข้ามแบบมีเงื่อนไข CONDITIONAL JUMP INSTRUCTION

แอสเซมเบลอร์จะสนับสนุนคำสั่งข้ามแบบมีเงื่อนไขต่างๆ ในการควบคุมการเคลื่อนย้ายตามผลลัพธ์ของแฟลกรีจิสเตอร์ ตัวอย่าง ท่านสามารถเปรียบเทียบฟิลด์ 2 ฟิลด์ แล้วกระโดดตามค่าของแฟลกที่ได้จากการเปรียบเทียบ มีรูปแบบดังนี้

[LABEL]	Jnnn	SHORT-ADDRESS

ตัวอย่างคำสั่ง LOOP จะทำหน้าที่ลดค่าและตรวจสอบ CX ถ้า CX <> 0 คำสั่งจะเคลื่อนย้ายข้อมูลการควบคุมไปยัง แอดเดรสรีอเบอรานด์ ท่านสามารถเขียนด้วย LOOP A20 ตามรูป 7-2

```
DEC     CX     ; equivalent to loop
JNZ     A20
```

คำสั่ง DEC และ JNZ ทำงานเทียบคำสั่ง LOOP เป็นการลดค่า CX ลง 1 และกระโดดไปที่ A20 ถ้า CX ไม่เท่ากับ 0 คำสั่ง DEC เป็นการเซ็ตค่า ZF = 1 คำสั่ง JNZ เป็นการตรวจสอบแฟลก ZF

ข้อมูลแบบคิดเครื่องหมายและไม่คิดเครื่องหมาย SIGNED AND UNSIGNED DATA

จุดประสงค์ในการกระโดดข้ามแบบมีเงื่อนไขจะต้องชัดเจน ชนิดของข้อมูล (signed or unsigned) ซึ่งท่านจะต้องใช้เปรียบเทียบหรือคำนวณทางคณิตศาสตร์ ให้ตามข้อมูลตามคำสั่งที่ใช้ ข้อมูลที่ไม่คิดเครื่องหมายจะใช้จำนวนบิตทั้งหมด ได้แก่ข้อมูลเป็นชุดสตริง เช่น ชื่อ(name) และแอดเดรส ส่วนค่าที่เป็นตัวเลข เช่น หมายเลขลูกค้า ข้อมูลที่คิดเครื่องหมาย บิตซ้ายสุดเป็นบิตเครื่องหมาย ถ้าเป็น 0 คือค่าบวก ถ้าเป็น 1 คือค่าลบ ค่าตัวเลขเป็นค่าบวกหรือค่าลบ

ในตัวอย่างต่อไป AX มีข้อมูล 1100 0110 และ BX มีข้อมูล 0001 0110 ถ้าใช้คำสั่ง CMP ในการเปรียบเทียบข้อมูล

```
CMP     AX,BX
```

รีจิสเตอร์ AX กับรีจิสเตอร์ BX ถ้าไม่คิดเครื่องหมาย ค่าของ AX จะมากกว่า BX ถ้าคิดเครื่องหมาย ค่าของ AX น้อยกว่า

JUMPS BASED ON UNSIGNED DATA

คำสั่งกระโดดข้ามแบบมีเงื่อนไขประยุกต์ใช้งานของข้อมูล ไม่คิดเครื่องหมาย ดังนี้

Symbol	Description	Flags Tested
JE/JZ	Jump equal or jump zero	ZF
JNE//JNZ	Jump not equal or jump not zero	ZF
JA/JNBE	Jump above or jump not below/equal	CF,ZF
JAE/JNB	Jump above/equal or jump not below	CF
JB/JNAE	Jump below or jump not above/below	CF
JBE/JNA	Jump below/equal or jump not above	CF,AF

ท่านสามารถใช้กฎในการตรวจสอบ 1 ใน 2 รหัสคำสั่ง ตัวอย่างคำสั่ง JB และ JNAE ซึ่งให้รหัสเครื่องเหมือนกัน แต่คำสั่งที่ใช้ตรวจสอบค่าบวก คำสั่ง JB ใช้ง่ายกว่าคำสั่งตรวจสอบค่าลบ JNAE

JUMPS BASED ON SIGNED DATA

คำสั่งกระโดดข้ามแบบมีเงื่อนไขประยุกต์ใช้งานของข้อมูลที่คิดเครื่องหมาย ดังต่อไปนี้

Symbol	Description	Flags Tested
JE/JZ	Jump equal or jump zero	ZF
JNE/JNZ	Jump not equal or jump not zero	ZF
JG/JNLE	Jump greater or jump not less/equal	ZF,SF,OF
JGE/JNL	Jump greater/equal or jump not less	SF,OF
JL/JNGE	Jump less or jump not greater/equal	SF,OF
JLE/JNG	Jump less/equal or jump not greater	ZF,SF,OF

การกระโดดข้ามสำหรับการตรวจสอบ equal/zero(JE/JZ) และสำหรับ not equal/zero(JNE/JNZ) จะรวมถึงคิดเครื่องหมายและไม่คิดเครื่องหมาย equal/zero เป็นเงื่อนไขที่เกิดขึ้นจากเครื่องหมาย

SPECIAL ARITHMETIC TESTS

คำสั่งกระโดดข้ามแบบมีเงื่อนไขใช้กรณีพิเศษ ดังต่อไปนี้

Symbol	Description	Flags Tested
JS	Jump sign (negative)	SF
JNS	Jump no sign (positive)	SF
JC	Jump carry (same as JB)	CF
JNC	Jump no carry	CF
JO	Jump overflow	OF
JNO	Jump no overflow	OF
JP/JPE	Jump parity even	PF
JNP/JPO	Jump parity odd	PF

คำสั่ง JC และ JNC ที่ใช้ในการตรวจสอบการทำงานบ่อยๆของดิสค์ ส่วนการกระโดดเงื่อนไขอื่นๆ คำสั่ง JCXZ ใช้ตรวจสอบข้อมูลของ CX ว่าเป็น 0 หรือไม่ คำสั่ง JCXZ ไม่ต้องใช้การคำนวณทางคณิตศาสตร์หรือการเปรียบเทียบ ในการเริ่มต้นของ LOOP คำสั่งนี้จะทำงานถ้าข้อมูลใน CX<>0

สำหรับคำสั่งที่ไม่คิดเครื่องหมายของข้อมูล เช่น JE,JA,JB สำหรับการคิดเครื่องหมาย คำสั่งกระโดดข้ามคือ JE,JG,JL การกระโดดที่สำหรับตรวจสอบ carry, overflow,parity ตามวัตถุประสงค์ แอสเซมเบลอร์จะทำการแปลสัญลักษณ์ไปเป็นรหัสเครื่องตามคำสั่งของท่านจะใช้ เช่น JAE และ JGE

คำสั่ง JCXZ

คำสั่ง JCXZ เป็นการหน่วงเวลาโดยไม่มีผลต่อแฟลก การทำงานใช้คำสั่งเดียวกันกับคำสั่ง LOOP ถ้า CX = 0 จะหยุดทำงาน การทำงานที่แตกต่างกันก็คือคำสั่ง LOOP ทำงาน 1 รอบ จะลดค่า CX ทีละ 1 โดยอัตโนมัติ ส่วนคำสั่ง JCXZ ไม่มีการลดค่า CX

ตัวอย่าง ถ้าเราต้องการทำงานวนรอบ ในส่วนของ FAR LOOP โดยการกำหนด CX ในส่วนแรกของโปรแกรม และทำงานในการวนรอบซ้ำๆ จะหยุดเมื่อ CX = 0 , CX จะต้องถูกตรวจสอบก่อน การลดค่า CX ถ้าเป็น 0 มันจะลดค่าลงหนึ่ง ในการทำงานซ้ำครั้งแรก และทำงาน 65535 ครั้ง จะลดค่า Cx = 0

```
            -           ;INSTRUCTION WHICH SET CX TO THE NUMBER
            -           ;OF REPETTITIONS TO BE CARRIED OUT
        JCXZ    DONE    ;CHECK THAT CX IS NOT ZERO
NEXT:       -
            -
        LOOP    NEXT
DONE:
```

IMPLEMENTING LOOPS

การยกตัวอย่างต่อไปนี้เป็นการเปรียบเทียบภาษาปาสคาลกับภาษาแอสแซมบลี

1. REPEAT ....UNTIL

คำสั่ง

```
REPEAT   ACTION 1
         ACTION 2
         ACTION 3
            -
         ACTION N
  UNTIL  < CONDITION >
```

```
REPEAT
READ A CHARACTER
UNTIL  CHARACTER IS A BLANK
```

สามารถเขียนคำสั่งได้ดังนี้

```
        MOV  AH,1
REPEAT:
        INT  21H
        CMP  AL,' '
        JNE  REPEAT
```

ภาษาปาสคาล

```
REPEAT  AX:=AX-1;
        BX:=BX+1;
        CX:= AX-BX
UNTIL   AX=0
```

ภาษาแอสแซมบลี

```
ST_REPEAT:  DEC  AX
            INC  BX
            MOV  CX,0
            ADD  CX,AX
            SUB  CX,BX
            CMP  AX,0
            JNZ  ST_REPEAT
```

2. WHILE ....DO

ภาษาปาสคาล ภาษาแอสแซมบลี

```
WHILE < CONDITION > DO BEGIN
                       ACTION 1
                       ------
                       ACTION N
                       END
```

****

```
WHILE AX < 100D DO BEGIN        START:  CMP   AX,100D
      AX:=AX+BX;                        JNB   STOP
      BX:=BX+1                          ADD   AX,BX
      END                               INC   BX
                                        JMP   START
                                STOP:
```

```
SET COUNT TO 0
     READ A CHARACTER
          WHILE CHAR <> CR  DO
     COUNT = COUNT + 1
          READ A CHARACTER
     END_WHILE
```

****

```
                MOV   DX,0
                MOV   AH,1
                INT   21H
       WHILE_:
                CMP   AL,0DH
                JE    END_WHILE
                INC   DX
                INT   21H
                JMP   WHILE_
```

3. FOR ...... DO

ภาษาปาสคาล

```
FOR I := INITIAL_VALUE TO FINAL_VALUE DO BEGIN
                        ACTION 1
                        ------
                        ACTION N
                        END
```

```
FOR I:= 3 TO 25D DO BEGIN
        AX:=AX+BX;
        BX:=BX-1
        END
```

โครงสร้างภาษาแอสแซมบลี

```
         MOV   LOOP_COUNTER,INITIAL_VALUE
START:   CMP   LOOP_COUNTER,FINAL_VALUE
         JA    STOP
         ACTION 1
         ------
         ACTION N
         INC   LOOP_COUNTER
         JMP   START
STOP:
```

การเขียนด้วยภาษาแอสแซมบลี

```
         MOV   LOOP_COUNTER,3
START:   CMP   LOOP_COUNTER,25D
         JA    STOP
         ADD   AX,BX
         DEC   BX
         INC   LOOP_COUNTER
         JMP   START
STOP:
```

รหัสเทียม

```
FOR  80 TIMES DO
     DISPLAY  '*'
END_FOR
```

ภาษาแอสแซมบลี

```
        MOV  CX,80
        MOV  AH,2
        MOV  DL,'*'
TOP:
        INT  21H
        LOOP  TOP
```

4.DECISIONS

IF THEN ELSE

IF <CONDITION> THEN ACTION 1 ELSE ACTION 2

รหัสเทียม ของโครงสร้าง IF - THEN

```
IF CONDITION IS TURE
    THEN
        EXECUTE  TURE -BRANCH STATEMENT
END_IF
```

หรือ อัลกอริทึม

```
IF   AX < 0
    THEN
        REPLACE  AX BY -A
END_IF
```

ภาษาแอสแซมบลี

```
          CMP  AX,0        ;IF AX < 0
          JNL  END_IF      ;NO EXIT
          NEG  AX          ;YES CHANG SIGN
END_IF:
```

ภาษาปาสคาล

```
IF AX = 0 THEN CX:=CX-AX;INC AX;DX:=DX+AX
          ELSE CX:=CX-23D
```

โครงสร้างภาษาแอสแซมบลี

```
    TEST <CON> AND IF FALSE JUMP TO ACTION 2
          ----
          JMP  DONE
ACTION_2  ---
          ---
DONE:
```

ภาษาแอสแซมบลี

```
           CMP  AX,0
           JNZ  ACTION_2
           SUB  CX,AX
           INC  AX
           ADD  DX,AX
           JMP  DONE
ACTION_2:  SUB  CX,23D
           ---
DONE:
```

รหัสเทียม

```
IF AL <= BL
    THEN  DISPLAY CHAR IN AL
    ELSE  DISPLAY CHAR IN BL
END_IF
```

```
            MOV     AH,2      ;PREPARE TO DISPLAY
            CMP     AL,BL     ; AL <= BL ?
            JNBE    ELSE_     ;NO DISPLAY CHAR IN BL
            MOV     DL,AL     ;AL <= BL
            JMP     DISPLAY   ;MOVE CHAR TO DIAPLAYED
ELSE:       MOV     DL,BL     ; GOTO DISPLAY
DISPLAY:    INT     21H       ;DISPLAY IT
END_IF:
```

ตัวอย่าง 7.10 โปรแกรมการเปรียบเทียบเลข 2 จำนวนไม่คิดเครื่องหมาย ของค่าที่อยู่ใน AX กับ BX ให้ค่ามากเก็บไว้ใน DX

```
        MOV   DX,AX      ;ASSUME AX IS LARGEST
        CMP   AX,BX      ;IF AX >= BX THEN
        JAE   QUIT       ;JUMP TO QUIT
        MOV   DX,BX      ;ELSE MOV BX TO DX
QUIT:
```

ตัวอย่าง 7.11 การหาค่าน้อยที่สุดของเลข 3 จำนวน โดยไม่คิดเครื่องหมายใน AL , BL , CL และนำค่าน้อยที่สุดไว้ในตัวแปร SMALL

```
     MOV   SMALL,AL   ;ASSUME AL IS THE SMALLEST
     CMP   SMALL,BL   ;IF SMALL <= BL THEN
     JBE   L1         ;JUMP TO L1
     MOV   SMALL,BL   ;ELSE MOVE BL TO SMALL
L1:  CMP   SMALL,CL   ;IF SMALL <= CL THEN
     JBE   L2         ;JUMP TO L2
     MOV   SMALL,CL   ;ELSE MOVW CL TO SMALL
L2:
```

CASE กรณีที่มีทางเลือกหลายๆทาง ในการตรวจสอบ รีจิสเตอร์ ตัวแปร มีรูปแบบดังต่อไปนี้

```
CASE      EXPRESSION
          VALUE_1 :  STATEMENT_1
          VALUE_2 :  STATEMENT_2
          ------
          VALUE_N :  STATEMENT_N
END_CASE
```

รหัสเทียม

```
CASE      AX
          < 0 : PUT -1 IN BX
          = 0 : PUT  0 IN BX
          > 0 : PUT  1 IN BX
END_CASE
```

ภาษาแอสแซมบลี

```
            CMP   AX,0          ;TEST AX
            JL    NEGATIVE      ;AX < 0
            JE    ZERO          ;AX = 0
            JG    POSITIVE      ;AX > 0
NEGATIVE:
            MOV   BX,-1         ;PUT -1 IN AX
            JMP   END_CASE      ;AND EXIT
ZERO:       MOV   BX,0          ;PUT 0 IN AX
            JMP   END_CASE      ; AND EXIT
POSITIVE:   MOV   BX,1          ; PUT 1 IN BX
END_CASE:
```

ตัวอย่าง 7.12 จงเขียนคำสั่งในการเปรียบเทียบเลขคู่ให้พิมพ์ตัว E ถ้าเป็นเลขคี่ให้พิมพ์ตัว O ดังตัวอย่างรหัสเทียมต่อไปนี้

```
CASE    AL
1,3 : DISPLAY 'O'
2,4 : DISPLAY 'E'
END_CASE
```

การแทนด้วยภาษาแอสแซมบลี

```
        CMP     AL,1        ;AL = '1' ?
        JE      ODD         ;YES DISPLAY 'O'
        CMP     AL,3        ;AL = '3'
        JE      ODD         ;YES DISPLAY 'O'
        CMP     AL,2        ;AL = '2'
        JE      EVEN        ;YES DISPLAY 'E'
        CMP     AL,4        ;AL = '4'
        JE      EVEN        ;YES DISPLAY 'E'
        JMP     END_CASE    ;NOT 1.....4
ODD:
        MOV     DL,'O'
        JMP     DISPLAY     ;DISPLAY 'O'
EVEN:
        MOV     DL,'E'      ;DISPLAY 'E'
DISPLAY:
        MOV     AH,2
        INT     21H
END_CASE:
```

การกระโดดข้ามด้วยเงื่อนไขผสม

รูปแบบของคำสั่งมีดังต่อไปนี้

```
CONDITION_1  AND  CONDITION_2
```

หรือ

```
CONDITION_1  OR  CONDITION_2
```

การทำงานของคำสั่ง AND

การใช้เงื่อนไขผสมโดยใช้เงื่อนไขทั้ง 2 มา AND กัน เพื่อให้ผลตามเงื่อนไขทั้งสองจึงจะทำงานตามที่กำหนด เช่นถ้ามีการรับข้อมูลจากคีย์บอร์ดเข้ามาเป็นตัวอักษรตัวใหญ่แล้วให้แสดงผล ถ้าเป็นตัวอื่นๆ ไม่ต้องแสดงผล ดังรหัสเทียมที่กำหนดขึ้นมาดังนี้

```
READ  A CHARACTER  (INTO AL)
   IF  ( 'A' <= CHAR) AND ( CHAR <= 'Z')
        THEN
        DISPLAY CHAR
   END_IF
```

เราสามารถแทนด้วยภาษาแอสแซมบลีดังต่อไปนี้

```
;READ  A CHARACTER  (INTO AL)
        MOV  AH,1           ; PREPARE TO READ
        INT  21H            ;CHARACTER IN AL
; IF  ( 'A' <= CHAR) AND ( CHAR <= 'Z')
        CMP  AL,'A'         ; CHAR >= 'A'?
        JNGE END_IF         ;NO_EXIT
        CMP  AL,'Z'         ;CHAR <= 'Z'?
        JNLE END_IF         ;NO_EXIT
        MOV  DL,AL          ;GET CHAR
        MOV  AH,2           ;PREPARE TO DISPLAY
        INT  21H            ;DISPLAY CHARACTER
   END_IF:
```

การทำงานของคำสั่ง OR

การใช้เงื่อนไขผสมโดยใช้เงื่อนไขทั้ง 2 มา OR กัน เพื่อให้ผลตามเงื่อนไขอย่างใดอย่างหนึ่ง จึงจะทำงานตามที่กำหนด เช่นการตรวจสอบ Y และ y ที่ใช้แสดงการสิ้นสุดของโปรแกรม

```
READ  A CHARACTER  (INTO AL)
    IF  ( CHAR = 'Y') OR ( CHAR  = 'y')
         THEN
         DISPLAY IT
         ELSE
         TERMINATE PROGRAM
    END_IF
```

เราสามารถแทนด้วยภาษาแอสแซมบลีดังต่อไปนี้

```
;READ A CHARACTER
         MOV  AH,1       ;PREPARE TO READ
         INT  21H        ; CHARACTER IN AL
;IF (CHAR ='Y') OR ( CHAR = 'y')
         CMP  AL,'Y'     ;CHAR = 'Y'?
         JE   THEN       ;YES GOTO DISPLAY IT
         CMP  AL,'y'     ;CHAR 'y'?
         JE   THEN       ;YES GOTO DISPLAY IT
         JMP  ELSE_      ;NO TERMINATE
    THEN:
         MOV  AH,2       ;PREPARE TO DISPLAY
         MOV  DL,AL     ;GET CHAR
         INT  21H       ;DISPLAY IT
         JMP  END_IF    ;EXIT
ELSE_:
         MOV  AH,4CH
         INT  21H        ;DOS EXIT
END_IF:
```

การเรียกโปรซีเยอร์ CALLING PROCEDURE

ในส่วนนี้ code segments จะประกอบไปด้วย 1 procedure

```
BEGIN   PROC   FAR
   .
   .
BEGIN   ENDP
```

โอเปอรานด์ FAR ในกรณีนี้ แอดเดรสของระบบจะเป็นจุดเข้าของโปรแกรม execute ขณะที่ ENDP คำสั่งเทียมที่กำหนดจุดสุดท้ายของ procedure code segment อาจมีข้อมูลทุกๆจำนวนของ procedure ภายใต้ PROC และ ENDP คำสั่ง CALL จะเป็นการเคลื่อนย้ายการควบคุมในการเรียก called procedure ไปที่จุดเดิม คำสั่ง RET เป็นคำสั่งสุดท้ายในการเรียก called procedure

การทำงานของคำสั่งCALL และ RET

คำสั่ง CALL และ RET เมื่อเป็นรหัสภาษาเครื่องขึ้นอยู่กับการทำงานใน NEAR หรือ FAR PROCEDURE

NEAR CALL AND RETURN

คำสั่ง CALL PROCEDURE ภายในเซกเมนต์เดียวกันใช้ NEAR มีรูปแบบดังต่อไปนี้

- ลดค่า SP ทีละ 2 (1 เวิร์ด)
- push ค่า IP ลงในสแตค (ซึ่งมีค่า offset ของคำสั่ง CALL)
- ใส่ค่า offset address ของคำสั่ง CALL procedure ใน IP

คำสั่ง RET ซึ่งเป็นการย้อนกลับจาก NEAR procedure มีรูปแบบดังต่อไปนี้

- pop ค่า IP เก่าจากสแตคเก็บค่าไว้ใน IP
- เพิ่มค่า SP อีก 2

FAR CALL AND RETURN

คำสั่ง CALL ชนิด FAR จะใช้เรียกลาเบล procedure ชนิด FAR ในการแยกส่วนจาก code segment คำสั่ง CALL ชนิด FAR จะทำการ push ทั้งค่า CS และIP ลงในสแตค และคำสั่ง RET จะทำการ pop ค่าออกจากสแตค

ตัวอย่าง 7.13 NEAR CALL AND RETURN

โครงสร้างชนิด NEAR CALL และ RETURN ดังในตัวอย่าง7-3 ดังต่อไปนี้

- โปรแกรมจะแบ่งใน FAR procedure โดยใช้ BEGIN และ NEAR procedure 2 ส่วนใช้ B10 และ C10 แต่ละ procedure จะมีชื่อจุดบนสุดและข้อมูลภายใน ENDP สำหรับจุดสุดท้าย
- คำสั่งเทียม PROC สำหรับ B10 และ C10 ใช้ attribute NEAR ในการกำหนด procedure เหล่านี้ภายใน code segment
- ใน procedure BEGIN คำสั่ง CALL จะเคลื่อนย้ายโปรแกรมควบคุมมาที่ procedure B10 เพื่อเริ่มต้นเอ็กซีคิวส์

```
                    TITLE  CALLPROC (EXE) Call proceddure
                           .MODEL SSMALL
                           .STACK 64
                           .CODE
0000                BEGIN  PROC    FAR
0000 E8 0007 R             CALL    B10       ; call B10
                    ;      ...
0003 B4 4C                 MOV     AH,4CH    ; terminate
0005 CD 21                 INT     21H
0007                BEGIN  ENDP
; ----------------------------------------------------------
0007                B10    PROC    NEAR
0007 E8 0008 R             CALL    C10       ; call C10
                    ;       ...
000A C3                    RET               ; return to
```

```
000B            B10   ENDP              ;  caller
; ---------------------------------------------------
000B            C10   PROC   NEAR
                ;     ...
000B C3               RET               ;  return to
000C            C10   ENDP              ;  caller
; ---------------------------------------------------
                      END    BEGIN
```

รูปที่ 7-3 Effect of execution on stack

- ใน procedure B10 คำสั่งจะควบคุมการทำงานไป procedure C10 เพื่อเริ่มต้นเอ็กซีคิวส์
- ใน procedure C10 คำสั่ง RET จะควบคุมการย้อนกลับไป ไปที่หลังคำสั่ง CALL C10
- ใน procedure B10 คำสั่ง RET จะควบคุมการย้อนกลับไป ไปที่หลังคำสั่ง CALL B10
- procedure BEGIN จะประมวลผลจากจุดนั้น
- RET จะเป็นคำสั่งที่ใช้ย้อนกลับงานประจำ ถ้า B10 ไม่สิ้นสุดด้วยคำสั่ง RET คำสั่งจะ execute ผ่าน B10 และหยุดโดยตรงใน C10 อันที่จริงถ้า C10 ไม่มีข้อมูลคำสั่ง RET โปรแกรมก็จะ execute ถึงคำสั่ง END ของ C10 และไม่สามารถให้ผลลัพธ์ได้

สแตคเซคเม้นต์ STACK SEGMENT

ในส่วนนี้ถ้าโปรแกรมของเรามีความต้องการ push ข้อมูลลงในสแตค โปรแกรมเหล่านี้ต้องการพื้นที่ของสแตคเพียงเล็กน้อย อย่างไรก็ตาม คำสั่ง CALL จะทำการ push ค่า IP ลงในสแตคโดยอัตโนมัติ ในกรณีการ call procedure แล้วจะต้องมีคำสั่ง RET ใช้สำหรับแอดเดรสย้อนกลับจากการ call procedure โดยการ pop เวิร์ดข้อมูลนั้นในสแตคไปไว้ใน IP

ตัวอย่างคำสั่งที่ PUSH ข้อมูลลงในสแตค เป็นการเคลื่อนย้ายข้อมูล 1 เวิร์ดแอดเดรสหรือค่า ลงในสแตค คำสั่ง POP จะเป็นการเข้าถึงสแตคหรือดึงข้อมูลออกมาจากสแตค ที่เก็บไว้โดยคำสั่ง PUSH การทำงานของคำสั่งทั้ง 2 ชนิด จะเปลี่ยนค่า offset address ของ SP(stack pointer) สำหรับชี้ข้อมูลเวิร์ดต่อไป เพราะ

ว่าการทำงานแบบนี้คำสั่ง RET จะต้องเก็บค่าเดิมของการ call คำสั่ง CALL สามารถเรียก procedure อื่นๆ พื้นที่ของสแตคจะต้องมีขนาดพอที่จะเก็บแอดเดรส อย่างน้อยจะต้องมี 32 เวิร์ด

คำสั่ง PUSH,PUSHF,CALL,INT และ INTO จะเป็นการลดค่าสแตคทีละ 2 และการเก็บข้อมูลของรีจิสเตอร์โดยการ push ลงในสแตค คำสั่ง POP,POPF,RET และ IRET ข้อมูลย้อนกลับที่อยู่ในสแตคมาเก็บไว้ในรีจิสเตอร์ และเพิ่มค่า SP อีก 2

การโหลดโปรแกรม EXE ในการ execute ระบบจะเซ็ตค่ารีจิสเตอร์ ต่อไปนี้

- DS และ ES แอดเดรสของ program segment prefix (PSP) 256 ไบท์ (100H) ซึ่งเป็นพื้นที่เตรียมการ execute โดยโปรแกรมในหน่วยความจำ
- CS จะเก็บแอดเดรสจุดเข้าของโปรแกรม ในการ execute คำสั่งแรก
- IP มีค่า 0 ถ้าการ execute คำสั่งแรก คือ เริ่มที่ code segment
- SS เก็บแอดเดรสของสแตคเซกเมนต์
- SP เก็บค่า offset ในยอดของสแตค ตัวอย่าง ถ้าท่านกำหนดขนาดของสแตคเป็น 32 เวิร์ด(64 ไบท์) ค่าของ SP จะมีค่า 64 หรือ 40H

ผลของโปรแกรม EXECUTE ใน STACK

ตำแหน่งของการใช้คำสั่ง PUSH,POP คือยอดของสแตค ดังตัวอย่างต่อไปนี้ ระบบจะโหลดจากการเซ็ต SP เพื่อบอกค่าขนาดของ SP เป็น 40H โปรแกรมจะมีการทำงานดังต่อไปนี้ (ตัวอย่าง 7-3)

- CALL B10 จะลดค่า SP ลง 2 ไปที่ 3EH แล้วทำการ PUSH ค่า IP (ข้อมูล 0003) ลงในสแตคที่ค่า offset 3E ซึ่งแอดเดรสนี้เป็นแอดเดรสของคำสั่งที่ตามหลังคำสั่ง CALL ตัวโปรเซสเซอร์ จะใช้แอดเดรสของ CS:IP เคลื่อนย้ายการควบคุมไป B10

- ในส่วนของ procedure B10 จะมีคำสั่ง CALL C10 จะเป็นการลดค่า SP ลง 2 ไปที่ 3CH แล้วทำการ PUSH ค่า IP (ข้อมูล 000A) ลงในสแตคที่ค่า offset 3C ตัวโปรเซสเซอร์จะใช้แอดเดรสของ CS:IP เคลื่อนย้ายการควบคุมไป C10

- การย้อนกลับจาก C10 คำสั่ง RET จะ pop ค่าแอดเดรส(000A) จากสแตคที่ตำแหน่ง 3CH ไปเก็บไว้ที่ IP และเพิ่มค่าของ SP อีก 2 ไปที่ 3EH และมันจะย้อนกลับโดยอัตโนมัติไปที่ 000A ใน procedure B10

- คำสั่ง RET ที่จุดสุดท้ายของ procedure B10 จะทำการ POP แอดเดรส(0003) จากสแตคที่ 3EH เก็บไว้ใน IP และเพิ่มค่า SP อีก 2 ที่ 40H และมันจะย้อนกลับโดยอัตโนมัติไปที่ 0003 ในส่วนของ code segment เมื่อโปรแกรมสิ้นสุดการ execute

จากภาพต่อไปนี้แสดงผลของการทำงานในสแตคในแต่ละคำสั่งที่ execute ส่วนข้อมูลเวิร์ดในหน่วยความจำจะเก็บข้อมูลสลับค่า เช่น ข้อมูล 0003 จะกลายเป็น 0300 ตัวอย่างที่แสดงค่า offset อยู่ที่ 0036 ถึง 003F เป็นข้อมูลของ SP

Operation	Stack	SP
On entry, initially:	xxxx xxxx xxxx xxxx xxxx	0040
CALL B10 (push 0003)	xxxx xxxx xxxx xxxx 0300	003E
CALL C10 (push 000A)	xxxx xxxx xxxx 0A00 0300	003C
RET (push 000A)	xxxx xxxx xxxx 0A00 0300	003E
RET (push 0003)	xxxx xxxx xxxx 0A00 0300	0040
stack offset :	0036 0038 003a 003C 003E	

คำสั่ง PUSH และ PUSHF

เป็นการบวกเวิร์ดใหม่ลงในสแตคเราใช้คำสั่ง PUSH มีกฎเกณฑ์ดังนี้

PUSH source

หรือ PUSH AX

การเอ็กซีคิวส์คำสั่ง PUSH มีดังนี้

1. SP จะลดค่าลง 2
2. ทำสำเนา Source ไปที่แอดเดรสของ SS:SP กำหนดโดยไม่เปลี่ยนแปลง

ส่วนคำสั่ง PUSHF คำสั่งนี้ไม่มีโอเปอร์รานด์ เป็นการเก็บค่าแฟลกลงในสแตค

คำสั่ง POP และ POPF

เป็นการย้ายข้อมูลจากยอดของสแตค มีกฎเกณฑ์ดังนี้

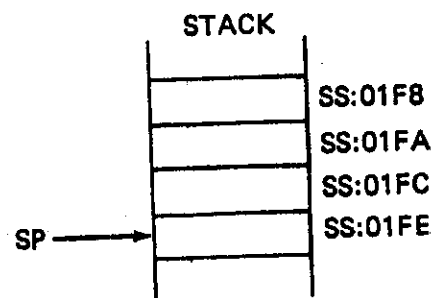
POP Destination

ตัวรับจะต้องมีขนาด 16 บิต หรือ

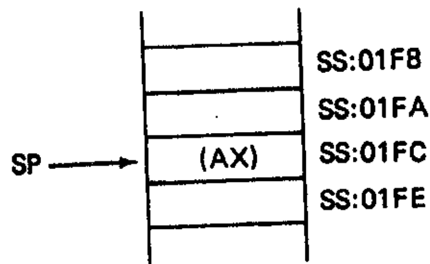
POP AX

การเอ็กซีคิวส์คำสั่ง POP มีดังนี้

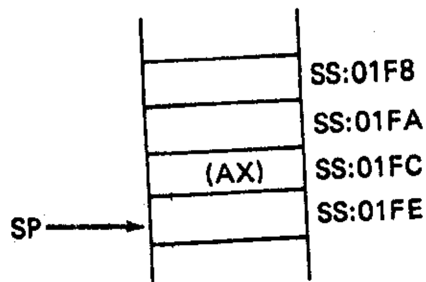
1. ข้อมูลของ SS:SP (ยอดของสแตค) เคลื่อนย้ายไปยังตัวรับ
2. SP จะเพิ่มขึ้น 2



(A) BEFORE PUSH AX



(B) AFTER PUSH AX



(C) AFTER POP AX

ตัวอย่าง 7.14 การทำงานของคำสั่งสแตคในการกลับค่าข้อมูลที่ป้อนเข้ามา

อัลกอริทึม

```
DISPLAY A '?'
    INITIALIZE COUNT TO 0
READ A CHARACTER
WHILE CHARACTER IS NOT A CARRIGE RETURN DO
    PUSH CHARACTER ONTO THE STACK
        INCREMENT COUNT
    READ A CHARACTER
END_WHILE
GO TO A NEW LINE
FOR COUNT TIMES DO
    POP A CHARACTER FROM THE STACK;
    DISPLAY IT ;
END_FOR
```

ภาษาแอสแซมบลี

```
        TITLE   REVERSE INPUT
        .MODEL SMALL
        .STACK 100H
        .CODE
MAIN    PROC
;DISPLAY USER PROMPT
        MOV  AH,2       ;PREPARE TO DISPLAY
        MOV  DL.'?'     ;CHAR TO DISPLAY
        INT  21H        ;DISPLAY ?
;INITIALIZE CHARACTER COUNT
        XOR  CX,CX      ;COUNT = 0
```

```
;READ A CHARACTER
            MOV  AH,1        ;PREPARE TO READ
            INT  21H         ;READ CHARACTER
;WHILE CHARACTER IS NOT A CARRIAGE RETURN DO
WHILE_:     CMP  AL,0DH      ;CR?
            JE   END_WHILE ;YES EXIT LOOP
;SAVE CHARACTER ON THE STACK AND INCERMENT COUNT
            PUSH AX          ;PUT IT ON STACK
            INC  CX          ;COUNT = COUNT + 1
;READ  A CHARACTER
            INT  21H
            JMP  WHILE_
END_WHILE:
  ;GO TO A NEW LINE
            MOV  AH,2        ;DISPLAY CHARACTER
            MOV  DL,0DH      ;CR
            INT  21H         ;EXECUTE
            MOV  DL,0AH      ;LF
            INT  21H         ;EXECUTE
            JCXZ EXIT        ;EXIT IF NO CHARACTER READ
  ;FOR COUNT TIMES DO
  TOP:
  ;POP A CHARACTER FROM THE STACK
            POP  DX          ;GET CHAR FROM STACK
  ;DISPLAY IT
            INT  21H         ;DISPLAY IT
            LOOP TOP
   ;END_FOR
   EXIT:
            MOV  AH,4CH
            INT  21H
     MAIN ENDP
            END  MAIN
```

ก่อนที่จะมีการเขียนโปรแกรมเคลื่อนย้ายข้อมูลไปที่รีจิสเตอร์ ข้อมูลที่ถูกเคลื่อนย้ายจากหน่วยความจำไปยังรีจิสเตอร์ การเคลื่อนย้ายจากรีจิสเตอร์ไปหน่วยความจำ หรือการเคลื่อนย้ายของรีจิสเตอร์ไปส่วนอื่นๆ ทุกๆกรณีความยาวของข้อมูลจะต้องมีขีดจำกัด 1 หรือ 2 ไบท์ และไม่มีการเคลื่อนย้ายข้อมูลจากที่หนึ่งของหน่วยความจำไปยังที่อื่นๆของหน่วยความจำ ในส่วนนี้จะอธิบายการเคลื่อนย้ายข้อมูลที่มีขนาดเกิน 2 ไบท์ วิธีอื่นๆที่ใช้กับคำสั่งสตริงอยู่ใน บทที่ 11

ตัวอย่าง โปรแกรม 7-4 ข้อมูลในส่วนของ data segment มีขนาด 9 ไบท์ 3 ชุด ในชื่อของ NAME1,NAME2,NAME3 รหัสภาษาเครื่องของโปรแกรมจะเป็นการเคลื่อนย้ายข้อมูลของ NAME ไป NAME2 และข้อมูลของ NAME2 ไป NAME3 ซึ่งฟิลด์เหล่านี้แต่ละฟิลด์มี 9 ไบท์ มากกว่าคำสั่ง MOV จะทำการเคลื่อนย้าย

procedure ของ BEGIN จะเซ็ตค่ารีจิสเตอร์เซกเมนต์และทำการเรียก B10 MOVE และ C10 MOVE B10 MOVE จะเคลื่อนย้ายข้อมูลของ NAME1 ไปยัง NAME2 เป็นการเคลื่อนย้ายข้อมูล 1 ไบท์ใน 1 เวลา งานจะทำณ.จุดเริ่มต้นที่ไบท์ซ้ายสุดของ NAME1 และจะวนรอบการเคลื่อนย้ายไปไบท์ที่ 2 ,ไบท์ที่ 3 ตามลำดับ

```
NAME1 :  A   B   C   D   E   F   G   H   I
         ¦   ¦   ¦   ¦   ¦   ¦   ¦   ¦   ¦
NAME2 :  J   K   L   M   N   O   P   Q   R
```

ขั้นตอนในการทำงานของ NAME1 และ NAME2 นั้น จะต้องเซ็ตค่าเคาเตอร์ของ CX=9 และใช้ index register SI และ DI กำหนดแอดเดรสคำสั่ง LEA เป็นการโหลดค่า offset address ของ NAME1 และ NAME2 ไปเก็บไว้ที่ SI และ DI ดังนี้

```
LEA     SI,NAME1   ; load offset address
LEA     DI,NAME2   ; of NAME1 and NAME2
```

การวนรอบใช้แอดเดรสในรีจิสเตอร์ SI และ DI ในการเคลื่อนย้ายข้อมูลไบท์แรกของ NAME1 ไปยังไบท์แรกของ NAME2 วงเล็บสี่เหลี่ยมรอบ SI และ DI ในส่วนโอเปอรานด์ของคำสั่ง MOV หมายความว่า คำสั่งจะใช้แอดเดรสในรีจิสเตอร์นั้นสำหรับ

การอ่านเขียนข้อมูลในหน่วยความจำ เช่น

```
        MOV     AL,[SI]
```

คำสั่งนี้หมายความว่า ใช้แอดเดรสใน SI (NAME1+0) เป็นแอดเดรสอ้างอิงของหน่วยความจำที่จะเคลื่อนย้ายข้อมูลไปที่ AL

```
        MOV     [DI],AL
```

คำสั่งนี้หมายความว่า เป็นการเคลื่อนย้ายข้อมูลของ AL ไปยังหน่วยความจำที่แอดเดรสอ้างอิงอยู่ใน DI (NAME2+0)

```
        page    65,132
TITLE   EXMOVE (EXE) Extended move operations
; -------------------------------------------
        .MODEL SMALL
        .STACK 64
; -------------------------------------------
        .DATA
NAME1   DB      'ABCDEFGHI'
NAME2   DB      'JKLMNOPQR'
NAME3   DB      'STUVWXYZ*'
        .CODE
BEGIN   PROC    FAR ;-------------------------
        MOV     AX,@data     ;initialize segment
        MOV     DS,AX        ;  registers
        MOV     ES,AX
        CALL    B10MOVE      ;call jump routine
        CALL    C10MOVE      ;call loop routine
        MOV     AH,4CH       ;Terminate processing
        INT     21H
BEGIN   ENDP
```

```
;         EXtened move using Jump-On-Condition:
B10MOVE   PROC ;-----------------------------------
          LEA    SI,NAME1    ;initialize address of
          LEA    DI,NAME2    ;NAME1 & NAME2
          MOV    CX,09       ;initialize to move 9 chars
B20 :
          MOV    AL,[SI]     ;move from NAME1
          MOV    [DI],AL     ;move to NAME2
          INC    SI          ;increment next char in NAME1
          INC    DI          ;increment next pos'n in NAME2
          DEC    CX          ;decrement loop count
          JNZ    B20         ;count not zero? Yes,loop
          RET                ;if count=0 return to
B10MOVE   ENDP               ;caller
;         Extended move using LOOP
C10MOVE   PROC ;-----------------------------------
          LEA    SI,NAME2    ;initialize address of
          LEA    DI,NAME3    ;NAME2 & NAME3
          MOV    CX,09       ;initialize to move 9;chars
C20 :
          MOV    AL,[SI]     ;move from NAME2
          MOV    [DI],AL     ;move to NAME3
          INC    SI       ;increment next char ;in NAME2
          INC    DI       ;increment next pos'n;in NAME3
          LOOP   C20      ;decrement count,loop ;nonzero
          RET                ;if count=0 return to
C10MOVE   ENDP               ;caller
          END    BEGIN
```

รูปที่ 7-4 Extended move operation

คำสั่ง INC เป็นการเพิ่มค่าของ SI และ DI คำสั่ง DEC เป็นการลดค่าของ CX ถ้า CX<>0 การวนรอบก็จะกลับไปที่ B20 และ ค่าของ SI และ DI จะเพิ่มขึ้นทีละ 1 คำสั่ง MOV ต่อไปจะอ้างถึง NAME1+1 และ NAME2+1 การวนรอบอย่างต่อเนื่องจะทำจนกระทั่งถึง NAME1+8 และ NAME2+8

ในส่วน procedure C10MOVE ก็ทำงานคล้ายกับ B10MOVE ซึ่งแยกออกเป็น 2 ส่วน ส่วนนี้เป็นการเคลื่อนย้ายจาก NAME2 ไปยัง NAME3 และใช้คำสั่ง LOOP แทนค่า DEC/JNZ

การทำงานของบูลีน BOOLEAN OPRATIONS : AND,OR,XOR,TEST,NOT

boolean logic คือความสำคัญในการออกแบบวงจร และ โปรแกรมลอจิกแบบขนาน คำสั่งที่ใช้ในพีชคณิตตรรก คือ AND,OR,XOR และ TEST ซึ่งคำสั่งเหล่านี้สามารถใช้ CLEAR และ SET บิต และเก็บข้อมูล ASCII สำหรับการคำนวณทางคณิตศาสตร์ มีรูปแบบดังนี้

[LABEL]	OPERATION	{ REG/MEM }, {REG/MEM/IMMEDIATE }

โอเปอรานด์แรกจะอ้างถึงข้อมูลขนาด 1 ไบท์ หรือ 1 เวิร์ด ในรีจิสเตอร์หรือหน่วยความจำ และจะทำให้ค่าใน รีจิสเตอร์ หรือ หน่วยความจำ เปลี่ยนแปลงหลังจากการ execute โอเปอรานด์ที่ 2 อ้างถึงรีจิสเตอร์หรือค่าคงที่ (immediate) การทำงานโดยการตรวจสอบแต่ละบิตของโอเปอรานด์ทั้ง 2 จะมีผลต่อ CF,OF,PF,SF และ ZF

OPERATION	COMMENT
AND	ผลลัพธ์จะมีค่าเป็นหนึ่งถ้าบิตอินพุตทั้งสองเป็นหนึ่ง
OR	ผลลัพธ์จะมีค่าเป็นหนึ่งถ้าอินพุตใดอินพุตหนึ่งมีค่าเป็นหนึ่ง
XOR	ผลลัพธ์จะมีค่าเป็นหนึ่งถ้าอินพุตมีค่าต่างกัน
NOT	ผลลัพธ์ที่ได้จะตรงกันข้ามกับค่าเดิม

AND การเปรียบเทียบแต่ละบิต ถ้าบิตทั้ง 2 เป็น 1 จะเป็นการเซ็ตผลลัพธ์เป็น 1 กรณีอื่นๆเป็น 0

```
AND  DESTINATION,SOURCE
```

```
         0 0 1 1
         0 1 0 1
         -------
ผลลัพธ์    0 0 0 1
```

ตัวอย่าง 7.15

```
MOV  AL,00111011B
AND  AL,00001111B        ;AL = 00001011B
```

การใช้คำสั่ง AND

```
AND  AX,BX                ;16 BIT REGISTER
AND  BL,BYTEVAL           ;8 BIT REGISTER , MEMORY
AND  WORDVAL,CX           ;16 BIT MEMORY , REGISTER
AND  AL,30H               ;8 BIT REGISTER , IMMEDIATE
AND  VAL1,00111111B       ;8 BIT MEMORY , IMMEDIATE
AND  BYTE PTR [BX],AL ;INDIRECT OPERAND , REGISTER
```

ตัวอย่าง 7.16 เขียนคำสั่ง AND ในการเคลียบิตสูงสุดของตัวอักขระในแฟ้มข้อมูล

```
MOV  AH,6                 ;CONSOLE INPUT FUNCTION
MOV  DL,0FFH              ;CHECK FOR INPUT CHARACTER
INT  21H
AND  AL,01111111B         ;STRIP HIGH BIT
MOV  DL,AL                ;MOVE CHAR INTO AL
MOV  AH,2                 ;WRITE CHAR TO STANDARD OUTPUT
INT  21H
```

OR การเปรียบเทียบแต่ละบิต ถ้าบิตทั้ง 2 เพียงบิตใดบิตหนึ่งมีค่าเป็น 1 จะเป็นการเซ็ตผลลัพธ์เป็น 1 แต่ถ้าบิตทั้ง 2 เป็น 0 ผลลัพธ์จะเป็น 0

```
OR   DESTINATION,SOURCE

          0  0  1  1
          0  1  0  1
          ----------
ผลลัพธ์    0  1  1  1
```

ตัวอย่าง 7.17 การ OR 3BH ด้วยค่า 0FH

```
MOV   AL,00111011B     ;3BH
OR    AL,00001111B     ;AL = 3FH
```

ตัวอย่าง 7.18

```
OR    AX,BX              ;16 BIT REGISTER
OR    BL,BYTEVAL         ;8 BIT REGISTER , MEMORY
OR    WORDVAL,CX         ;16 BIT MEMORY , REGISTER
OR    AL,30H             ;8 BIT REGISTER , IMMEDIATE
OR    VAL1,00111111B     ;8 BIT MEMORY , IMMEDIATE
OR    BYTE PTR [BX],AL   ;INDIRECT OPERAND , REGISTER
```

XOR เป็นการเปรียบเทียบแต่ละบิต ถ้าบิตหนึ่งเป็น 0 และอีกบิตมีค่าเป็น 1 จะเซ็ตผลลัพธ์เป็น 1 ถ้าบิตทั้ง 2 มีค่าเหมือนกัน(ไม่ว่าจะเป็น 0 หรือ 1) ผลลัพธ์เป็น 0

```
XOR   DESTINATION,SOURCE

          0  0  1  1
          0  1  0  1
          ----------
ผลลัพธ์    0  1  1  0
```

ตัวอย่าง 7.19 การใช้คำสั่ง XOR

```
        MOV  AL,10110100B
        XOR  AL,10000110B            ;AL = 00110010B , OR 32H
```

ตัวอย่าง 7.20

```
        XOR   AX,BX                 ;16 BIT REGISTER
        XOR   BL,BYTEVAL            ;8 BIT REGISTER , MEMORY
        XOR   WORDVAL,CX            ;16 BIT MEMORY , REGISTER
        XOR   AL,30H                ;8 BIT REGISTER , IMMEDIATE
        XOR   VAL1,00111111B        ;8 BIT MEMORY , IMMEDIATE
        XOR   BYTE PTR [BX],AL      ;INDIRECT OPERAND , REGISTER
```

TEST การเซ็ตค่าแฟลกมีลักษณะเหมือนคำสั่ง AND แต่ไม่มีผลการเปลี่ยนแปลงของบิตคำสั่ง มีการทำงานดังต่อไปนี้

```
        TEST  DESTINATION,SOURCE

        TEST  AX,BX                 ;16 BIT REGISTER
        TEST  BL,BYTEVAL            ;8 BIT REGISTER , MEMORY
        TEST  WORDVAL,CX            ;16 BIT MEMORY , REGISTER
        TEST  AL,30H                ;8 BIT REGISTER , IMMEDIATE
        TEST  VAL1,00111111B        ;8 BIT MEMORY , IMMEDIATE
        TEST  BYTE PTR [BX],AL      ;INDIRECT OPERAND , REGISTER
```

ตัวอย่าง 7.21 การตรวจสอบสถานะของเครื่องพิมพ์ คำสั่ง INT 17H เป็นการตรวจสอบสถานะของเครื่องพิมพ์ ข้อมูลขนาด 1 ไบท์จะถูกนำมาไว้ที่ AL ถ้าบิตที่ 5 เป็น 1 เครื่องพิมพ์ไม่มีกระดาษ การตรวจสอบใช้คำสั่ง TEST ดังนี้

```
        MOV  AH,2             ;FUNCTION READ PRINTERSTATUS
        INT  17H
        TEST AL,00100000B     ;ZF = 0 IT OUT OF PAPER
```

NOT คำสั่ง NOT จะทำการเปลี่ยนค่าทุกบิตเป็นตรงกันข้ามดังนี้

```
NOT  DESTINATION
```

ตัวอย่าง 7.22 การทำ 1's คอมพลีเม้นต์

```
MOV  AL,11110000B
NOT  AL                 ;AL = 00001111B
```

ตัวอย่าง 7.23 การทำ 2's คอมพลีเม้นต์

```
MOV  AL,1               ;AL = 0000001B
NOT  AL                 ;AL = 1111110B
INC  AL                 ;AL = 1111111B
```

ตัวอย่าง 7.24 การใช้คำสั่ง NOT

```
NOT  AL                 ;8 BIT REGISTER
NOT  BX                 ;16 BIT REGISTER
NOT  BYTE PTR [SI]      ;INDIRECT OPERAND
NOT  WORD1              ;16 BIT MEMORY
NOT  BYTEVAL1           ;8 BIT MEMORY
```

NEG คำสั่ง NEG เป็นการเปลี่ยนค่าบวกให้เป็นค่าลบ หรือค่าตรงกันข้าม การเปลี่ยนโดยใช้หลักการของ 2's คอมพลีเม้นต์ ดังนี้

```
NEG  DESTINATION
```

ตัวอย่าง 7.25 การใช้คำสั่ง NEG

```
NEG  AL                 ;8 BIT REGISTER
NEG  BX                 ;16 BIT REGISTER
NEG  BYTE PTR [SI]      ;INDIRECT OPERAND
NEG  WORD1              ;16 BIT MEMORY
NEG  BYTEVAL1           ;8 BIT MEMORY
```

ตัวอย่าง 7.26

```
        MOV  AL,-128   ;AL = 10000000B
        NEG  AL        ;AL = 10000000B , OF = 1
หรือ
        MOV  AL,+127   ;AL = 01111111B
        NEG  AL        ;AL = 10000000B , OF = 0
```

การทำงานของคำสั่ง AND , OR , XOR

	AND	OR	XOR
	0101	0101	0101
	<u>0011</u>	<u>0011</u>	<u>0011</u>
ผลลัพธ์	0001	0111	0110

ตัวอย่าง 7.27 การทำงานของพีชคณิต

ตัวอย่างที่แสดงต่อไปนี้ แต่ละคำสั่งไม่มีความสัมพันธ์กัน สมมติว่า AL มีข้อมูล 1100 0101 และ BH มีข้อมูล 0101 1100

```
1. AND  AL,BH  ; SETS AL TO 0100 0100
2. OR   BH,AL  ; SETS BH TO 1101 1101
3. XOR  AL,AL  ; SETS AL TO 0000 0000
4. AND  AL,00  ; SETS AL TO 0000 0000
5. AND  AL,0FH ; SETS AL TO 0000 0101
6. OR   CL,CL  ; SETS SF AND ZF
```

ตัวอย่าง 7.28 ข้อ 3,4 จะเป็นการเคลียร์รีจิสเตอร์ให้เป็น 0 ตัวอย่าง 5 เป็นการเคลียร์ 4 บิตซ้ายมือให้เป็น0ของ AL แต่ท่านสามารถใช้คำสั่ง OR ในการเคลียร์ได้ ดังนี้

```
1. OR   CX,CX   ; TEST CX FOR ZERO
   JZ   ...     ; JUMP IF ZERO
```

```
2. OR   CX,CX    ; TEST CX FOR SIGN
   JS   ...      ; JUMP IF NEGATIVE
```

คำสั่ง TEST เหมือนกับคำสั่ง AND ทำหน้าที่เพียงเซ็ตค่าแฟลก ดังตัวอย่างต่อไปนี้

```
1. TEST  BL,11110000B  ; ANY OF LEFTMOST BITS
   JNZ   ...           ; IN BL NONZERO?
2. TEST  AL,00000000B  ; DOES THE AL CONTAIN AN
   JNZ   ...           ; ODD NUMBER?
3. TEST  DX,0FFH       ; DOES THE DX CONTAIN A
   JZ    ...           ; ZERO VALUE?
```

ส่วนคำสั่งอื่นๆ เช่น NOT ทำหน้าที่เปลี่ยนบิตทุกบิตในไบท์หรือเวิร์ด ให้มีค่าตรงข้าม ในรีจิสเตอร์หรือหน่วยความจำคือ 0 เป็น 1 และ 1 เป็น 0 ตัวอย่าง AL=1100 0101 หลังจาก NOT AL แล้วจะเป็น AL=0011 1010 ไม่มีผลต่อแฟลก คำสั่ง NOT ไม่เหมือนกับ NEG ซึ่งเปลี่ยนค่าจากค่าบวกไปเป็นค่าลบ โดยการเปลี่ยนบิตทุกบิตเป็นตรงข้ามแล้วบวก 1 เข้ากับบิต LSB

การเปลี่ยนตัวอักขระตัวเล็กเป็นตัวใหญ่ CHANGING LOWERCASE TO UPPERCASE

จะมีเหตุผลต่างๆในการที่จะเปลี่ยนตัวอักษรตัวใหญ่ และตัวอักษรตัวเล็ก ดังตัวอย่าง ท่านอาจจะได้รับข้อมูลจากระบบการประมวลผล ในรูปของตัวอักษรตัวใหญ่ หรือโปรแกรมของผู้ใช้ต้องรับคำสั่งในรูปของ ตัวใหญ่หรือตัวเล็ก(เช่น Y หรือ y) และเปลี่ยนข้อมูลนั้นตัวใหญ่ สำหรับการตรวจสอบคำสั่ง ตัวอักษรตัวใหญ่ A-Z คือ 41H ถึง 5AH ส่วนตัวอักษรตัวเล็ก a-z คือ 61H ถึง 7AH มันแตกต่างกันอยู่ที่บิตที่ 5 เป็น 0 สำหรับตัวใหญ่ และ 1 สำหรับตัวเล็กดังนี้

	Uppercase		Lowercase
Letter A:	01000001	Letter a:	01100001
Letter Z:	01011010	Letter z:	01111010
Bit:	76543210	Bit:	76543210

โปรแกรม COM ในรูป 7-5 เป็นการเปลี่ยนรายการข้อมูล TITLEX จากตัวเล็กเป็นตัวใหญ่ เริ่มต้นที่แอดเดรส TITLEX+1 โปรแกรมจะเซ็ตค่า BX เป็นแอดเดรสของ TITLEX+1ไปที่ AH ถ้าค่าข้อมูลอยู่ในช่วง 61Hและ7AH ทำการ AND ในบิต 5 ให้เป็น 0

```
AND     AH,11011111B
```

ตัวอักษรทั้งหมดที่มากกว่า Z จะไม่เปลี่ยนแปลง การทำงานมันจะเคลื่อนย้ายตัวอักษรที่เปลี่ยนแล้วไปเก็บไว้ที่ TITLEX และเพิ่มค่า BX ทีละ 1 สำหรับตัวอักษรต่อไป

การใช้วิธีนี้ รีจิสเตอร์ BX จะทำเหมือน index register สำหรับแอดเดรสหน่วยความจำ หรือท่านอาจจะใช้ SI และ DI สำหรับวัตถุประสงค์นี้ก็ได้

```
TITLE  CASE    (COM) Change lowercase to uppercase
       .MODEL  SMALL
       .CODE
       ORG     100H
BEGIN: JMP     MAIN
; ------------------------------------------------
TITLEX DB      'Change to uppercase letters'
; ------------------------------------------------
MAIN   PROC    NEAR
       LEA     BX,TITLEX+1    ;1st char to change
       MOV     CX,26          ;No. of char to change
B20:
       MOV     AH,[BX]        ;Character from TITLEX
       CMP     AH,61H         ;Is it
       JB      B30            ;    lower
       CMP     AH,7AH         ;    case
       JA      B30            ;    letter?
       AND     AH,11011111B   ;Yes-convert
       MOV     [BX],AH        ;Restore in TITLEX
B30:
```

```
        INC     BX              ;Set for next char
        LOOP    B20             ;Loop 26 times
        MOV     AH,4CH          ;Done--exit
        INT     21H
MAIN    ENDP
        END     BEGIN
```

รูปที่ 7-5 Changing lowercase to uppercase

การเลื่อน SHIFTING

คำสั่ง SHIFT ซึ่งเป็นคำสั่งหนึ่งในกลุ่มของคำสั่งลอจิก สามารถใช้ได้ดังนี้

- อ้างอิงถึงรีจิสเตอร์หรือแอดเดรสหน่วยความจำ
- เลื่อน(shift)บิตไปทางซ้ายหรือขวา
- เลื่อนได้ 8 บิตถ้าข้อมูลเป็นไบท์ เลื่อนได้ 16 บิตถ้าข้อมูลเป็นเวิร์ด และ 32 บิต (80386/80486)
- เลื่อนทางลอจิก (ไม่คิดเครื่องหมาย) หรือคำนวณทางคณิตศาสตร์(คิดเครื่องหมาย)

โอเปอรานด์ที่ 2 เป็นข้อมูลของค่าที่จะเลื่อนซึ่งเป็นค่าคงที่ หรือค่าที่อ้างอิงอยู่ใน CL สำหรับ 8088/8086 ข้อมูลที่เป็นค่าคงที่อาจจะเป็น 1 ถ้ามีการเลื่อนมากกว่า 1 จะต้องเก็บข้อมูลใน CL 80186-80486 สามารถเซ็ตค่าคงที่ได้ 1-31 ตามรูปแบบดังนี้

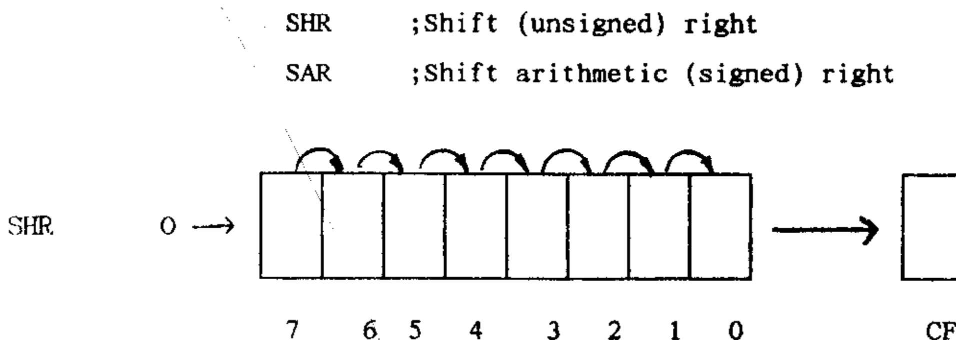
```
OP-CODE     DEST,1
OP-CODE     DEST,CL
```

[LABEL]	SHIFT	{ REG/MEM },{ CL/IMMEDIATE

การเลื่อนขวา SHIFT RIGHT

การเลื่อนทางขวาเป็นการเคลื่อนย้ายบิตในรีจิสเตอร์ไปทางขวา บิตที่ถูกเลื่อนนั้นจะเลื่อนไปที่แฟลกตัวทด คำสั่ง right shift จะแบ่งออกเป็นไม่คิดเครื่องหมายและคิดเครื่องหมาย

```
SHR    ;Shift (unsigned) right
SAR    ;Shift arithmetic (signed) right
```




```
SHR  DEST,1
SHR  DEST,CL
```

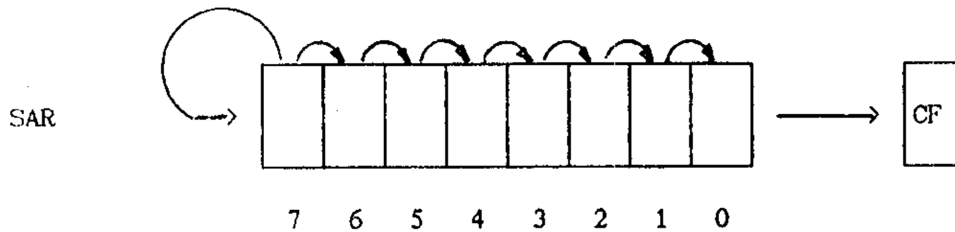
The following related instruction illustrate SHR and unsigned data :

```
MOV   CL,03         ;   AL:
MOV   AL,10110111B  ;10110111
SHR   AL,01         ;01011011 Shift right1
SHR   AL,CL         ;00001011 Shift right3
SHR   AX,12         ;80186-80486 only
```

ขั้นแรกคำสั่ง SHR เป็นการเลื่อนข้อมูลใน AL ไปทางขวา 1 บิต การเลื่อนไป 1 บิตนั้น จะส่งไปที่แฟลกตัวทดด้วย และบิต 0 จะใส่ลงบิตซ้ายสุดของ AL ส่วนขั้นที่ 2 คำสั่ง SHR จะเลื่อนใน AL มากกว่า 3 บิต แฟลกตัวทดเป็น 1,1และ0 นั้น เมื่อเลื่อนแล้วค่า 0 จะใส่ลงบน 3 บิตซ้ายสุดของ AL

```
MOV   AL,0D0H      ;AL = 0D0H
SHR   AL,1         ;AL = 68H
MOV   AX,65143
MOV   CL,2
SHR   AX,CL        ; เป็นการหาร 4
```

SAR



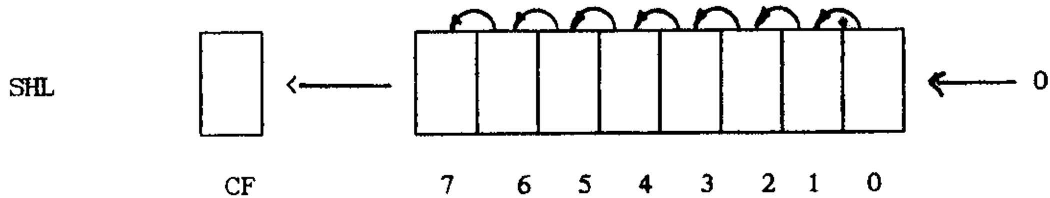
SAR จะต่างกับ SHR SAR ใช้บิตเครื่องหมายอยู่ด้านซ้ายสุด กรณีที่เป็นค่าบวกและเป็นค่าลบ จากคำสั่งต่างๆเหล่านี้ แสดงคำสั่ง SAR และข้อมูลไม่คิดเครื่องหมายซึ่งบิตเครื่องหมายเป็น 1

```
MOV     CL,03          ; AL:
MOV     AL,10110111B   ; 10110111B
SAR     AL,01          ; 11011011B Shift right1
SAR     AL,CL          ; 11111011 Shift right3
SAR     AX,12          ; 80186-80486 only

MOV     AX,8000H
MOV     CL,5
SAR     DX,CL          ;DX = FC00H
```

การเลื่อนซ้าย SHIFTING LEFT

การเลื่อนซ้ายเป็นการเคลื่อนย้ายข้อมูลในรีจิสเตอร์ไปทางซ้าย บิตซ้ายสุดจะเลื่อนไปที่แฟลกตัวทด คำสั่งเลื่อนซ้ายมีทั้งคิดเครื่องหมายและไม่คิดเครื่องหมาย

```
SHL     ;Shift (unsigned) left
SAL     ;Shift arithmetic (signed) left
```



The following related instruction illustrate SHL and unsigned data :

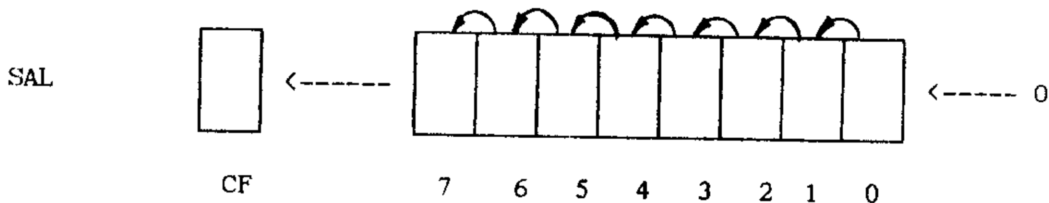
```
MOV    CL,03          ;   AL:
MOV    AL,10110111B   ; 10110111
SHL    AL,01          ; 01101110 Shift left1
SHL    AL,CL          ; 01110000 Shift left3
SHL    AX,12          ; 80186-80386 only
```

ขั้นแรกคำสั่ง SHL จะเลื่อนข้อมูลใน AL 1 บิตไปทางซ้าย และเลื่อนบิตซ้ายสุดไปแฟลกตัวทด และใส่ค่า 0 ที่บิตขวาสุดใน AL ขั้นที่ 2 คำสั่ง SHL เลื่อนข้อมูลใน AL 3 บิต แฟลกตัวทดจะมีข้อมูล 0,1และ1 ส่วน 3 บิตขวาสุดจะเป็น 0 ของ AL การเลื่อนทางซ้ายจะใส่ค่า 0 ทางขวา ผลของการทำงานคำสั่ง SHL และ SAL

การเลื่อนทางซ้ายโดยเฉพาะใช้ประโยชน์ในการทำให้ค่าเป็น 2 เท่า และเลื่อนทางขวาทำให้ลดครึ่งหนึ่ง การทำงานทั้งสองความสำคัญของมันก็คือ ทำงานได้รวดเร็วกว่าการคูณและการหาร

ท่านสามารถใช้คำสั่ง JC ตรวจสอบการเลื่อนบิตในแฟลกตัวทดที่สิ้นสุดการทำงานของการเลื่อน

```
MOV   BL,8FH    ;BL = 8FH
SHL   BL,1      ;BL = 1EH  CF = 1
```



MOV CL,3

SAL AX,CL ; AX x 8

คำสั่ง SAL ใช้ในการคูณตัวเลขที่มีเครื่องหมาย รหัสของคำสั่งนี้เหมือนกับ SHL เช่นการคูณติดลบของรีจิสเตอร์ AX = -1 (0FFFFH) ถ้าเลื่อนซ้าย 3 ครั้ง AX = -8 (0FFF8H)

การหมุน ROTATING

คำสั่ง ROTATE ซึ่งเป็นส่วนหนึ่งของการทำงานทางลอจิก ในคอมพิวเตอร์ มีรูปแบบดังต่อไปนี้

- อ้างถึงข้อมูลไบท์หรือเวิร์ด
- อ้างถึงข้อมูลในรีจิสเตอร์หรือหน่วยความจำ
- หมุนทางซ้ายหรือขวา
- หมุนขนาด 8บิต,16บิต และ32บิต
- หมุนชนิดลอจิกไม่คิดเครื่องหมาย หมุนคณิตศาสตร์คิดเครื่องหมายรอเปอรานด์ตัวที่ 2 จะเป็นค่าของการหมุนซึ่งเป็นค่าคงที่ หรือค่าที่อยู่ใน CL

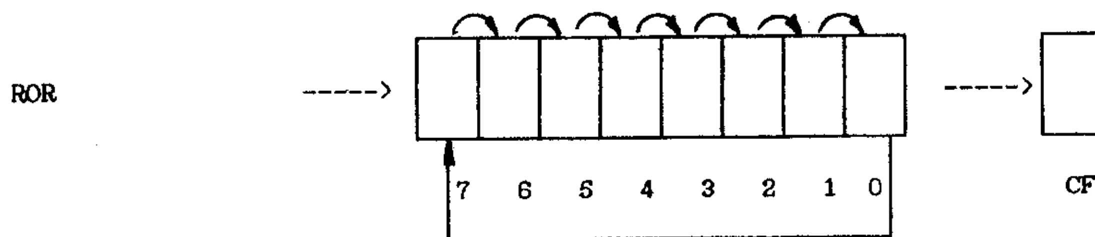
สำหรับ8088/8086 ค่าคงที่จะมีเพียง 1 การหมุนค่าที่มากกว่า 1 จะต้องเก็บข้อมูลในCL ส่วน 80186/80486 ค่าคงที่สามารถกำหนดได้ถึง 31 ดังรูปแบบต่อไปนี้

[LABEL]	ROTATE	{ REG/MEM },{ CL/IMMeDIATE }

การหมุนขวา ROTATING RIGHT

การหมุนทางขวาที่กำหนดในรีจิสเตอร์ สำหรับการคิดเครื่องหมายและไม่คิดเครื่องหมายดังต่อไปนี้

```
ROR    DEST,1
ROR    DEST,CL
ROR                 ;Rotate right
RCR                 ;Rotate with carry right
```



The following related instructions illustrate ROR :

```
MOV    CL,03          ;      BH:
MOV    BH,10110111B   ; 10110111
ROR    BH,01          ; 11011011 Rotate right1
ROR    BH,CL          ; 01111011 Rotate right3
ROR    BX,12          ; 80186-80486 only
MOV    AL,01h         ; AL = 01H
ROR    AL,1           ; AL = 80H  CF = 1
ROR    AL,1           ; AL = 40H  CF = 0
```

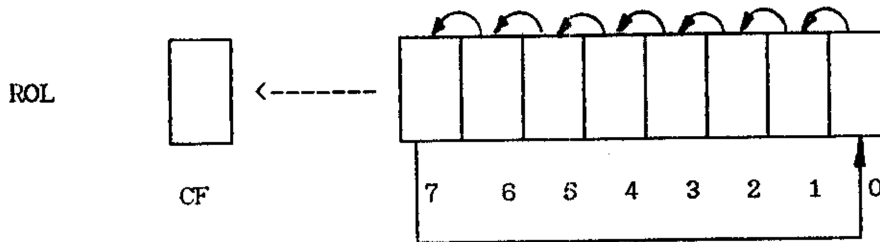
ขั้นแรกคำสั่ง ROR โดยการหมุนบิตขวามือสุด 1 บิตของBHไปยังบิตซ้านสุด ขั้นตอนที่2 คำสั่ง ROR เป็นการหมุน 3 บิต

คำสั่ง ROR จะหมุนผ่านแฟลกตัวทด บิตขวามือสุดจะเลื่อนไปที่ CF และ CFเลื่อนไปบิตซ้ายสุด ท่านสามารถใช้คำสั่ง JC ตรวจสอบบิตที่ได้จากการหมุนใน CF หลังจากสิ้นสุดการทำงาน

การหมุนซ้าย ROTATING LEFT

การหมุนทางซ้ายเป็นการหมุนบิตในรีจิสเตอร์ไปทางซ้าย คำสั่งการหมุนทางซ้ายจะคิดเครื่องหมายและไม่คิดเครื่องหมาย

```
ROL     ;Rotate left
RCL     ;Rotate with carry left
```



ตัวอย่าง 7.29 การใช้คำสั่ง ROL ในการนับจำนวนบิต 1 ของ BX โดยค่าของ BX ไม่เปลี่ยนแปลง ผลลัพธ์เก็บไว้ใน AX

```
        XOR   AX,AX        ;AX COUNTS BITS
        MOV   CX,16        ;LOOP COUNTS
TOP:    ROL   BX,1         ;CF = BIT RITATED OUT
        JNC   NEXT         ;0 BIT
        INC   AX           ;1 BIT
NEXT:   LOOP  TOP
```

The following related instructions illustrate ROL:

```
MOV    CL,03          ;      BH:
MOV    BH,10110111B   ; 10110111
ROL    BH,01          ; 01101011 Rotate left1
ROL    BH,CL          ; 01111011 Rotate left3
ROL    BX,12          ; 80186-80486 only
```


171

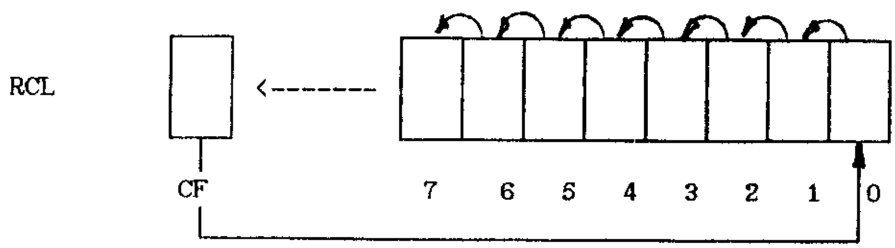
ขั้นแรกคำสั่ง ROL จะหมุนบิตไปทางซ้าย 1 บิตของ BLไปที่บิตขวาของ BL
ขั้นที่ 2 ROL จะหมุน 3 บิต
คำสั่ง RCL เป็นการหมุนผ่านตัวทด บิตซ้ายสุดเคลื่อนไปที่แฟลกตัวทด (CF)
และบิต CF ไปที่บิตขวาสุด

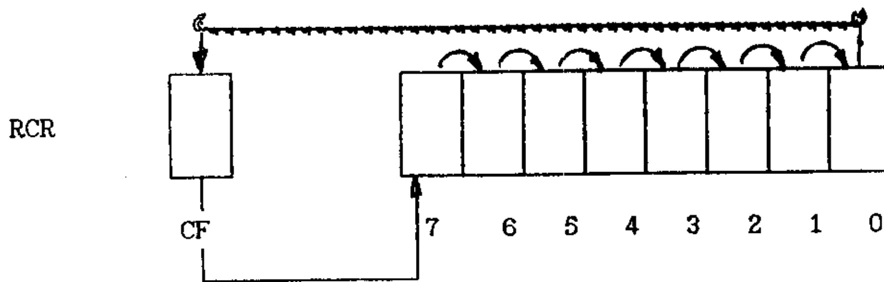
การใช้คำสั่ง ROL แลกเปลี่ยน 4 บิตต่ำกับ 4 บิตสูง

```
          MOV     CL,4
          MOV     AL,26H
          ROL     AL,CL         ; AL = 62H
          ROL     BYTEVAL,CL    ; BYTEVAL = F0H
          ...
BYTEVAL   DB      0FH
```

การใช้คำสั่ง ROL ในการนับจำนวนบิตใน BX โดยไม่เปลี่ยนค่าใน BX คำตอบเก็บใน AX

```
          XOR     AX,AX      ;AX = COUNTS BITS
          MOV     CX,16      ;LOOP
TOP:      ROL     BX,1       ;CF = BIT ROTATE OUT
          JNC     NEXT       ; 0 BIT
          INC     AX
NEXT:
          LOOP    TOP
```





ตัวอย่าง 7.30 SHIFT & ROTATE

พิจารณาจากค่า 32 บิตซึ่งค่า 16 บิตในซ้ายมือคือ DX และ 16 บิตขวามือ คือ AX

```
SHL     AX,1        :multiply DX:AX pair
RCL     DX,1        ; by 2
```

คำสั่ง SHL เลื่อนบิตทั้งหมดใน AX ไปทางซ้าย และบิตซ้ายสุดเลื่อนไปใน CF
คำสั่ง RCL เลื่อน DX ไปทางซ้าย และนำค่าบิตใน CF ไปไว้บิตขวามือสุด

ตัวอย่าง 7.31 การใช้งานของคำสั่ง ROTATE ในการเลื่อนหลายๆไบท์

	BYTE 1	BYTE 2	BYTE 3
BEFORE	00111011	01000110	11111111
AFTER	00011101	10100011	01111111

ตัวอย่าง 7.32 การเขียนโปรแกรมดังนี้

```
                MOV  CX,4         ;REPEAT THE SHIFT FOR TIMES
AGAIN:
                SHR  BYTE1,1      ;HIGHEST BYTE
                RCR  BYTE2,1      ;MIDDLE BYTE INCLUDE CARRY FLAGE
                RCR  BYTE3,1      ;LOW BYTE INCLUDE CARRY FLAGE
                LOOP AGAIN
BYTE1     DB   3BH                ;AFTER = 03H
BYTE2     DB   46H                ;AFTER = B4H
BYTE3     EB   0FFH               ;AFTER = 6FH
```

ตัวอย่าง 7.33 สมมุติค่าของ DH = 8AH , CF = 1 และ CL = 3 จงหาค่าใน DH และ CF หลังจากการเอ็กซีคิวส์คำสั่ง RCR DH,CL

	CF	DH
Initial values	1	1 0 0 0 1 0 1 0
After 1 right rotation	0	1 1 0 0 0 1 0 1
" 2 " "	1	0 1 1 0 0 0 1 0
" 3 " "	0	1 0 1 1 0 0 0 1 = B1H

ตัวอย่าง 7.34 การประยุกต์ใช้คำสั่ง SHIFT และ ROTATE ในการสลับบิตข้อมูลของรีจิสเตอร์ดังนี้

1 1 0 1 1 1 0 0 ------- 0 0 1 1 1 0 1 1

เราสามารถใช้คำสั่ง SHIFT เลื่อนบิตไปทางซ้ายของ AL ไปยัง CF และใช้คำสั่ง ROTATE (RCR) เลื่อนไปทางซ้าย เช่น BL มีข้อมูล 8 บิตและสลับบิตข้อมูลของ BL เราสามารถใช้ AL มาช่วยในการทำสำเนาข้อมูลดังนี้

```
            MOV  CX,8         ;NUMBER OF OPERATION TO DO
REVERSE:
            SHL  AL,1         ;GET A BIT INTO CF
            RCR  BL,1         ;ROTATE INTO BL
            LOOP REVERSE      ;LOOP UNTIL DONE
            MOV  AL,BL        ;AL GETS REVERSED PATTERN
```

ตัวอย่าง 7-35

การรับข้อมูลไบนารี สมมุติว่าโปรแกรมอ่านข้อมูลจากคีย์บอร์ด และทำการตรวจสอบด้วย CR ค่าที่รับเป็นสตริง 0 และ 1 ของตัวอักขระที่รับข้อมูลเข้ามา เราต้องการเปลี่ยนให้อยู่ในรูปของบิต และนำไปเก็บไว้ในรีจิสเตอร์ จากอัลกอริทึมในการอ่านค่าของไบนารีจากคีย์บอร์ดและเก็บไว้ใน BX

ALGORITHM FOR BINARY INPUT

```
CLEAR  BX  /* BX WILL HOLD BINARY VALUE*/
     INPUT A CHARACTER  /* '0' OR '1'*/
          WHILE CHARACTER <> CR  DO
               CONVERT CHARACTER TO BINARY VALUE
          LEFT SHIFT BX
          INSERT VALUE INTO 1sb OF BX
     INPUT CHARACTER
END_WHILE
```

DEMONSTRATION (FOR INPUT 110)

```
CLEAR  BX
     BX = 0   0   0   0     /* HEX
INPUT CHAR    '1' ., CONVERT TO 1
     LEFT SHIFT  BX
     BX = 0   0   0   0    /*HEX
INPUT VALUE INTO 1sb
     BX = 0   0   0   1    /*HEX
INPUT CHAR '1' , CONVERT TO 1
     LEFT SHIFT BX
     BX = 0   0   0   2      /*HEX
INSERT VALUE INTO 1sb
     BX = 0   0   0   3     /*HEX
INPUT CHAR '0' , CONVERT TO 0
     LEFT SHIFT BX
     BX = 0   0   0   6      /*HEX
INSERT VALUE INTO 1sb
     BX = 0   0   0   6                      ***  BX = 110B
```

```
                    XOR   BX,BX         ;CLEAR BX
                    MOV   AH,1          ;INPUT CHAR FUNCTION
                    INT   21H           ;READ A CHAR
          WHILE_:
                    CMP   AL,0DH            ;CR
                    JE    END_WHILE         ;YES DONE
                    AND   AL,0FH            ;NO CONVERT TO BINARY VALUE
                    SHL   BX,1              ;MAKE ROOM FOR NEW VALUE
                    OR    AL,AL             ;PUT INTO BX
                    INT   21H               ;READ A CHAR
                    JMP   WHILE_            ;LOOP BACK
      END_WHILE:     -
```

BINARY OUTPUT แสดงข้อมูลที่เอ้าพุตในรูปของไบนารีจากข้อมูลในรีจีสเตอร์ BX โดยใช้คำสั่ง SHIFT ดูรายละเอียดจากอัลกอริทึมต่อไปนี้

```
FOR  16  TIMES  DO
     ROTATE LEFT BX /* BX HOLDS OUTPUT VALUE,
          PUT msb INTO CF */
     IF CF = 1
          THEN
     OUTPUT '1'
          ELSE
     OUTPUT '0'
     END_IF
END_FOR
```

ตัวอย่าง 7.36 การรับข้อมูลเลขฐานสิบหกที่เป็นตัวเลข 0 - 9 และตัวอักษร A - F และตามด้วยตัว CR จากตัวอย่างเราสมมุติว่ารับข้อมูลตัวอักษรตัวใหญ่เท่านั้น การป้อนข้อมูลไม่เกิน 4 ตัว และให้เปลี่ยนข้อมูลอยู่ในรูปของไบนารี สำหรับค่าของไบนารีใน BX จะต้องเลื่อน 4 ครั้ง ต่อเลขฐานสิบหก 1 ตัว ดังอัลกอริทึมของ HEX INPUT

```
CLEAR  BX  /* BX WILL HOLD INPUT VALUE */
     INPUT HEX CHARACTER
     WHILE CHARACTER <> CR DO
          CONVERT CHARACTER TO BINARY VALUE
          LEFT SHIFT BX 4 TIMES
          INSERT VALUE INTO LOWER 4 BITS OF BX
          INPUT CHARACTER
     END_WHILE
```

ภาษาแอสแซมบลี

```
                XOR  BX,BX          ;CLEAR BX
                MOV  CL,4           ;COUNTER FOR 4 SHIFTS
                MOV  AH,1           ;INPUT CHAR FUNCTION
                INT  21H
     WHILE_:
                CMP  AL,0DH         ;CR?
                JE   END_WHILE      ;YES EXIT
;Convert character to binary value
                CMP  AL,39H         ;A DIGIT?
                JG   LETTER         ;NO A LETTER
;Input a digit
                AND  AL,0FH         ;CONVERT DIGIT TO BINARY
                JMP  SHIFT          ;GO TO INSERT BX
     LETTER:
```

```
                SUB   AL,37H          ;CONVERT LETTER TO BINARY
        SHIFT:
                SHL   BX,CL           ;MAKE ROOM FOR NEW VALUE
;Insert value into BX
                OR    BL,AL           ;PUT VALUE INTO LOW 4 BITS BX
                INT   21H             ;INPUT A CHAR
                JMP   WHILE_          ;LOOP UNTIL CR
        END_WHILE:
```

HEX OUTPUT

ต่อไปนี้เป็นการแสดงผลลัพธ์ข้อมูลขนาด 16 บิตในรีจีสเตอร์ BX หรือมีเลขฐานสิบหก 4 ตัว ซึ่งข้อมูลของ BX จะเริ่มต้นจากข้อมูลที่เก็บไว้ในทางซ้ายทีละตัว การเปลี่ยนเป็นเลขฐานสิบหกเพื่อแสดงผลค่าของเลขฐานสิบหก มีอัลกอริทึมดังต่อไปนี้

```
FOR  4 TIMES DO
     MOVE BH TO DL /*BX HOLDS PUTPUT VALUE*/
     SHIFT DL 4 TIMES TO THE RIGHT
     IF DL < 10
          THEN
     CONVERT TO CHARACTER IN '0' ..'9'
          ELSE
     CONVERT TO CHARACTER IN 'A'...'F'
     END_IF
          OUTPUT CHARACTER
     ROTATE BX LEFT 4 TIMES
END_FOR
```

DEMONSTRATION (FOR INPUT 6AB)

```
CLEAR  BX
     BX = 0  0  0  0    /* HEX
INPUT CHAR   '6' , CONVERT TO 0110
     LEFT SHIFT  BX 4 TIMES
     BX = 0  0  0  0   /*HEX
INSERT VALUE INTO LOWER 4 BITS OF BX
     BX = 0  0  0  6   /*HEX
INPUT CHAR 'A' , CONVERT TO AH = 1010
     LEFT SHIFT BX 4 TIMES
     BX = 0  0  6  0     /*HEX
INSERT VALUE INTO LOWER 4 BITS OF BX
     BX = 0  0  6  A    /*HEX
INPUT CHAR 'B' , CONVERT TO 1011
     LEFT SHIFT BX  4 TIMES
     BX = 0  6  A  0     /*HEX
INSERT VALUE INTO LOWER 4 BITS OF BX
     BX = 0  6  A  B                     *** BX = 06ABH
```

ระบบงานของโปรแกรม PROGRAM ORGANIZATION

ขั้นตอนในการเขียนโปรแกรมแอสแซมบลีมีดังต่อไปนี้

1) ต้องมีแนวคิดในการแก้ปัญหาอย่างชัดเจน
2) กำหนดรูปแบบทั่วๆไปและวางแผนทางลอจิก ตัวอย่าง เช่นปัญหาการเคลื่อนย้ายข้อมูลหลายๆไบท์ ตามรูปที่ 7.4 จะต้องเริ่มต้นกำหนดไฟล์ ที่จะเคลื่อนย้ายข้อมูล และวางแผนในการใช้คำสั่ง การเซ็ทค่าเริ่มแรก การใช้คำสั่งกระโดดแบบมีเงื่อนไข และการใช้ LOOP ซึ่งแสดงการทำงานทางลอจิกหลัก การใช้คำสั่งเทียม โปรแกรมเมอร์จะต้องมีการวางแผนดังนี้

เซ็ตค่าเซคเม้นต์รีจีสเตอร์
ใช้การกระโดด (JUMP)
ใช้การกระโดด (LOOP)
RETURN TO DOS

ใช้การกระโดด (JUMP) มีดังนี้

```
        เซ็ตรีจีสเตอร์เก็บค่าตัวนับ  แอดเดรสของชื่อ
JUMP1:  เคลื่อนย้ายตัวอักขระของชื่อ
        เพิ่มแอดเดรสของชื่อกำหนดตัวอักขระต่อไป
        ลดค่าตัวนับ ถ้า <> 0 กระโดดไปที่ JUMP1
                ถ้า = 0 กลับสู่ DOS
```

การทำงานของ LOOP ก็เหมือนกับคำสั่ง JUMP

สรุป

- Short address ค่าออฟเซ็ตจะมีขีดจำกัดในช่วง -128 ถึง +127
- Near address ค่าออฟเซ็ตจะมีขีดจำกัดในช่วง -32,768 ถึง +32,767 อยู่ภายในเซคเม้นต์เดียวกัน
- Far address สามารถกระโดดข้ามเซคเม้นต์ โดยมีแอดเดรสเซคเม้นต์และออฟเซ็ตแอดเดรส
- Label เช่น B20: ที่อยู่ภายใน Procedure ตามด้วยโคลอน (:) ซึ่งเป็น Near label
- Label สำหรับการกระโดดข้ามแบบมีเงื่อนไขจะเป็น Short label
- การใช้คำสั่ง LOOP จะต้องมีการเซ็ตค่าตัวรีจีสเตอร์ CX การทำงานจะตรวจสอบค่า CX เป็น 0 หรือยัง ถ้าเป็น 0 จะหยุดทำงาน
- การทำงานของคำสั่ง CALL จะต้องคู่กับคำสั่ง RET ที่จุดสุดท้ายของ Procedure

- ให้ระวังการใช้ Index operand จากตัวอย่างคำสั่งต่อไปนี้

```
MOV  AX,SI
MOV  AX,[SI]
```

คำสั่งแรกจะเคลื่อนย้ายข้อมูลของ SI ไปที่ AX คำสั่งที่ 2 เป็นการเคลื่อนย้ายข้อมูลที่ SI เก็บค่าออฟเซ็ตแอดเดรสจากหน่วยความจำไปที่ AX

แบบฝึกหัด

7-1. จำนวนค่าสูงสุดของจำนวนไบท์ในการใช้คำสั่ง Near jump , Loop และ Conditional jump สามารถกระโดดข้ามได้กี่ไบท์

7-2. คำสั่ง JMP เริ่มต้นที่แอดเดรส 0624H ถ้าเราต้องการกระโดดไปข้างหน้าได้สูงสุดที่แอดเดรสอะไร

7-3. จงใช้คำสั่ง LOOP ในการคำนวนค่า Fibonacci series: 1,1,2,3,5,8,13,......ให้คำนวน 12 รอบ

7-4. สมมุติว่าค่า AX และ BX มีข้อมูลที่มีเครื่องหมาย และ CX และ DX มีข้อมูลไม่คิดเครื่องหมาย ใช้คำสั่ง CMP ในการกระโดดข้ามแบบมีเงื่อนไข ดังต่อไปนี้

1. ค่าของ DX มากกว่าค่าของ CX
2. ค่าของ BX มากกว่าค่าของ AX
3. ค่าของ CX เป็น 0
4. เกิดค่าเกินขนาด (Overflow?)
5. ค่าของ BX =< AX
6. ค่าของ DX =< CX

7-5. จากข้อย่อยต่อไปนี้ แฟลกอะไรจะมีผลต่อการทำงาน และในแฟลกมีค่าอะไร

1. เกิดค่าเกินขนาด
2. ผลลัพธ์เป็นค่าลบ
3. ผลลัพธ์เป็น 0
4. ประมวลผลทีละคำสั่ง
5. การเคลื่อนย้ายข้อมูลสตริงจากขวาไปซ้าย

7-6. จงอธิบายข้อแตกต่างระหว่าง PROC NEAR กับ PROC FAR

7-7. จงเขียนคำสั่งในการใช้คำสั่ง SHIFT ในการคูณ AX ด้วย 10

7-8. จงเขียนโปรแกรมในการรับข้อมูลที่เป็นตัวเลข 18 ตัวจากคีย์บอร์ด ส่วนตัวอื่นๆไม่ยอมรับ เมื่อพบคีย์ ENTER กลับสู่ดอส

7-9. จงเขียนโปรแกรมในการแสดงสตริงที่กำหนดให้และให้แสดงผลกลับค่ากันกับสตริงชุดเดิมดังตัวอย่างต่อไปนี้

THIS STRING WAS TYPED AT THE KEYBOARD.

OUTPUT.

DRAOBYEK EHT TA DEPYT SAW GNIRTS SIHT

7-10. จงเขียนโปรแกรมในการนับเวิร์ดข้อมูลว่ามีตัวอักษรกี่ตัว จากข้อมูลต่อไปนี้

COMPUTETR ORGANIZATION AND ASSEMBLY LANGUAGE.

7-11. จงโปรแกรมในการบันทึกเรคคอร์ดข้อมูลต่อไปนี้

ACCOUNT INPUT SCREEN

NAME:..................................ACC NUM:...............

PREVIOUS BALANCE:..

PAYMENTS:..

CRDITS:..

7-12. หลังจากเอ็กซีคิวส์คำสั่งต่อไปนี้ จงหาค่าที่อยู่ใน AL และ BL

```
        MOV   AL,VAL2
        MOV   BL,VAL1
        XOR   BL,0FFH
        TEST  AL,3
        JZ    LABEL1
        MOV   AL,1
        JMP   EXIT
LABEL1:
        MOV   BL,1
EXIT:
VAL1    DB    35H
VAL2    DB    3FH
```

7-13. หลังจากเอ็กซีคิวส์คำสั่งต่อไปนี้ จงหาค่าที่อยู่ใน CX และ DX และ SI

```
            MOV  SI,0
            MOV  CX,VAL1
            XOR  DX,VAL2
            AND  CX,0FFH
            NOT  DX
            XCHG DX,VAL1
AGAIN:
            INC   SI
            DEC  DX
            LOOP   AGAIN

VAL1        DW   026AH
VAL2        DB   3FD9H
```

7.14 จงเขียนโปรแกรมในการรับข้อมูลตัวอักขระ 1 ตัว และพิมพ์ผลลัพธ์เป็นรหัส ASCII ในรูปของเลขฐานสิบหก ดังการแสดงผลลัพธ์ต่อไปนี้

```
TYPE A CHARACTER :  Z
THE ASCII CODE OF Z IN HEX IS  5A
TYPE A CHARACTER:
```

7.15 จงเขียนโปรแกรมในการรับข้อมูลตัวอักขระ 1 ตัว และพิมพ์ผลลัพธ์เป็นรหัส ASCII ในรูปของเลขฐานสอง ดังการแสดงผลลัพธ์ต่อไปนี้

```
TYPE A CHARACTER :  A
THE ASCII CODE OF A IN HEX IS  01000001
TYPE A CHARACTER:
```

7.16 จงเขียนโปรแกรมในการบวกเลขฐานสิบที่รับเข้ามาจากคีย์บอร์ดและพิมพ์ผลลัพธ์ในบรรทัดต่อไปดังต่อไปนี้

```
ENTER A DECIMAL DIGIT STRING: 1299843
THE SUM OF THE DIGITS IN HEX IS: 0024
```