# บทที่ 14

## แมคโคร และ LINKING TO SUBPROGRAMS

วัตถุประสงค์

- การสร้างแมคโคร
- การส่งผ่านพารามิเตอร์แมคโคร
- การใช้แมคโครซ้ำ
- การเชื่อมต่อภาษาแอสแซมบลีกับภาษาระดับสูง



CT 215 381

# บทที่ 14
# แมคโคร และ LINKING TO SUBPROGRAMS

## แมคโคร (Macros)

แมคโครเป็นชื่อของสัญลักษณ์ที่ใช้ในคำสั่งภาษาแอสแซมบลี 1 คำสั่งหรือมากกว่า แมคโครอาจจะนำมาใช้ได้หลายครั้งในโปรแกรม เมื่อมีการสร้างแมคโคร ท่านสามารถกำหนดการทำงานในการใช้แมคโคร โดยการเรียกใช้ชื่อแมคโคร ข้อดีของการกำหนดแมคโคร นั่นคือคำสั่งจะถอดรหัสเพียงครั้งเดียว ในการเรียกใช้แมคโครแต่ละครั้งแอสแซมเบลอร์ก็ทำงานตามคำสั่งแมคโคร สมมุติว่าเราวางแผนในการเขียนคำสั่งที่แสดงข้อมูลของ DL ในการโปรแกรมหลายๆครั้ง

```
MOV   AH,2        ;WRITE DL TO THE CONSOLE
INT   21H         ;CONSOLE DOS
```

เราสามารถเก็บคำสั่งเหล่านี้ใน Procedure และเรียกใช้ตามต้องการ โดยสร้างคำสั่งจาก CALL และ RET ซึ่งวิธีการนี้ทำให้โปรแกรมทำงานช้าลง แต่วิธีการใช้แมคโครเป็นวิธีการที่ดีกว่า เราสามารถสร้างแมคโคร เรียกว่า PUTCHAR

การกำหนด PUTCHAR เป็นแมคโครเราสามารถเรียกใช้ได้จากทุกที่ในส่วนของ Code segment การเรียกใช้แมคโครไม่เหมือนกับการเรียกใช้ Procedure หมายความว่าคำสั่งของแมคโครจะใช้แทนที่คำสั่งในโปรแกรมของการใช้คำสั่ง CALL ดังนี้

```
PUTCHAR   MACRO           ;BEGIN MACRO DEFINITION
          MOV   AH,2
          INT   21H
          ENDM            ;END MACRO
```

ในส่วนของ Code segment เราสามารถเรียก PUTCHAR ตามชื่อของแมคโครในการเรียกใช้แต่ละครั้ง MASM จะเพิ่มคำสั่งจากแมคโครเข้าไปในโปรแกรม

| SOURCE CODE | GENERATED STATEMENTS |
|---|---|
| .CODE | .CODE |
| MAIN PROC | MAIN PROC |
| - | - |
| - | - |
| MOV DL,'A' | MOV DL,'A' |
| PUTCHAR | + MOV AH,2 |
| - | + INT 21H |
| - | - |
| MOV DL,'*' | MOV DL,'*' |
| PUTCHAR | + MOV AH,2 |
| - | + INT 21H |

ตัวอย่างแมคโครในการเก็บค่าของรีจีสเตอร์และการดึงข้อมูลในรีจีสเตอร์กลับมาไว้ที่เดิม

```
PUSHA_      MACRO           ;SAVE SYSTEM'S
            PUSH AX         ;REGISTER
            PUSH BX
            PUSH CX
            PUSH DX
            PUSH BP
            PUSH SI
            PUSH DI
            ENDM

POPA_       MACRO           ;RESTORE SYSTEEM'S
            POP  DI         ;REGISTER
            POP  SI
            POP  BP
            POP  DX
            POP  CX
            POP  BX
            POP  AX
            ENDM
```

```
CLEARSCREEN   MACRO                ;CLEAR SCREEN
              MOV  AX,0600H        ;INTERRUPT PARAMETER
              MOV  CX,0            ;SET UUPER CORNER
              MOV  DX,184FH        ;SET LOWER CORNER
              MOV  BH,7            ;SET ATTRIBUTE (NORMAL)
              INT  10H             ;CALL INTERRUPT
              ENDM                 ;END MACRO
```

## การส่งผ่านพารามิเตอร์

แมคโครสามารถส่งผ่านค่าได้ เมื่อมีการเรียกใช้ PUTCHAR สำหรับตัวอย่าง ในการโหลด DL ซึ่งเก็บค่าของตัวอักขระ (Character) ที่จะแสดงผล ก่อนการเรียกแมคโคร แต่เราสามารถเพิ่มค่าพารามิเตอร์ ในการกำหนดของแมคโคร (Char) เราสามารถผ่านตัวอักขระในบรรทัดเดียวกันเมื่อมีการเรียกใช้แมคโคร

```
PUTCHAR       MACRO     CHAR       ;BEGIN MACRO
              MOV  AH,2
              MOV  DL,CHAR         ; CHAR TAKE ON NEW VALUE
              INT  21H
              ENDM

              .CODE

              PUTCHAR   'A'        ;CALL THE MACRO HERE
```

การส่งผ่านค่าตัวแปรให้กับแมคโครสามารถเปลี่ยนแปลงไปๆได้และสะดวก การใช้พารามิเตอร์ในการกำหนดของแมคโคร เราสามารถเปลี่ยนแปลงได้ง่าย ในการเขียนแมคโคร DISPLAY กับพารามิเตอร์ที่เราเรียกว่า STRING ซึ่งเป็นชื่อที่ต้องการแสดงผล

```
DISPLAY       MACRO     STRING
              PUSH AX
              PUSH DX
              MOV  AH,9
              MOV  DX,OFFSET STRING
              INT  21H
              POP  DX
              POP  AX
              ENDM
```

STRING คือการแทนค่าการเรียกแมคโครแต่ละครั้งที่มีการเรียกใช้ การแสดงผลค่าของสตริงที่แตกต่างกัน 3 ค่า เราสามารถเขียนแมคโครได้ 3 บรรทัด โดยส่งผ่านที่ ARGUMENT ที่แตกต่างกันได้ในแมคโครแต่ละครั้ง

```
        DISPLAY   MSG1
        DISPLAY   MSG2
        DISPLAY   MSG3


MSG1    DB    'THIS IS MESSAGE1.',0AH,0DH,'$'
MSG2    DB    'THIS IS MESSAGE2.',0AH,0DH,'$'
MSG3    DB    'THIS IS MESSAGE3.',0AH,0DH,'$'
```

การส่งผ่าน ARGUMENT ในแมคโคร DISPLAY ทำได้สะดวก เราสามารถหลีกเลี่ยงการเขียนโปรแกรมที่ยุ่งยาก ในการส่งค่าให้ Procedure จากตัวอย่าง แสดงการเขียนคำสั่งของ MASM โดยใช้แมคโคร

```
DISPLAY  MSG1

PUSH AX
PUSH DX
MOV  AH,9
MOV  DX,OFFSET MSG1
INT  21H
POP  DX
POP  AX
ENDM

DISPLAY  MSG2

PUSH AX
PUSH DX
MOV  AH,9
MOV  DX,OFFSET MSG2
INT  21H
POP  DX
POP  AX
```

```
DISPLAY  MSG3

PUSH AX
PUSH DX
MOV  AH,9
MOV  DX,OFFSET MSG3
INT  21H
POP  DX
POP  AX
```

## แมคโครกำหนดตำแหน่ง

การกำหนดตำแหน่งของแมคโคร เพื่อกำหนดตำแหน่งของเคอร์เซอร์ในบรรทัดและคอลัมน์ที่ต้องการบนจอภาพ

```
LOCATE      MACRO     ROW,COLUMN
            PUSH AX
            PUSH BX
            PUSH DX
            MOV  BX,0        ;CHOOSE PAGE 0
            MOV  AH,2        ;LOCATE CURSOR
            MOV  DH,ROW
            MOV  DL,COLUMN
            INT  10H         ;CALL BIOS
            POP  DX
            POP  BX
            POP  AX
            ENDM
```

LOCATE อาจจะใช้เรียกผ่านพารามิเตอร์ข้อมูล 8 บิต ,Memory operands และค่าของรีจีสเตอร์

```
LOCATE    10,20        ;Immediate value
LOCATE    ROW,COLUMN   ;Memory operands
LOCATE    CH,CL        ;Pass register
```

ข้อควรระวังในการส่งผ่านจากรีจีสเตอร์ เนื่องจากรีจีสเตอร์ที่ใช้ในแมคโคร ถ้าเราเรียกแมคโคร LOCATE ใช้ AH และ AL แมคโครจะได้ค่าใน AH และ AL ไม่ตรงตามต้องการ ฉนั้นคำสั่งในแมคโครจะต้องมีการแทนค่ารีจีสเตอร์ หลังจากพารามิเตอร์ จะต้องแทนค่าดังต่อไปนี้

```
                LOCATE    AH,AL
1:              PUSH AX
2:              PUSH BX
3:              PUSH DX
4:              MOV  BX,0       ;CHOOSE PAGE 0
5:              MOV  AH,2       ;LOCATE CURSOR
6:              MOV  DH,ROW
7:              MOV  DL,COLUMN
8:              INT  10H        ;CALL BIOS
9:              POP  DX
10:             POP  BX
11:             POP  AX
```

สมมุติค่ารีจีสเตอร์ AH ส่งผ่านข้อมูลค่า ROW และ AL ส่งผ่านข้อมูลค่า COLUMN บรรทัด 5 แทนค่าใน AH ก่อนที่เราจะเปลี่ยนการเคลื่อนย้ายค่า DH ในบรรทัดที่ 6 เพราะฉนั้นแมคโครจะกำหนดไม่ถูกต้อง เคอร์เซอร์จะเลื่อนไปบรรทัดที่ 2

## การใช้แมคโครกำหนดข้อมูล

แมคโครอาจจะเรียกใช้จาก Data segment เราอาจจะใช้กำหนดพื้นที่ของตัวแปร ในตัวอย่างต่อไป ในการสร้างจุดเข้าของตาราง ALLOC MACRO ประกอบด้วยพื้นที่ว่าง 4 ไบท์และค่า 0 จำนวน 4 ไบท์

```
ALLOC     MACRO     VERNAME    ;NUMBYTES
VERNAME   DB        NUMBYTE DUP('    ',0,0,0,0)
          ENDM

          .DATA
ALLOC     VALUE1,20            ;ALLOCATE 20 ENTRIES
ALLOC     VALUE2,50            ;ALLOCATE 50 ENTRIES
```

สามารถกำหนดคำสั่งได้ดังนี้

```
VALUE1    DB    20 DUP('     ',0,0,0,0)
VALUE2    DB    50 DUP('     ',0,0,0,0)
```

VERNAME และ NUMBYTES คือชื่อในการกำหนดค่าและผ่านไปยังแมคโคร ส่วนแรกต้องรวมชื่อของตัวแปรและส่วนที่ 2 ต้องกำหนดจำนวนไบท์ในการกำหนดค่า DUP

## การกำหนดและการเรียกแมคโคร

แมคโครสามารถกำหนดได้ทุกที่ในโปรแกรม การใช้คำสั่งเทียม MACRO และ ENDM มีกฎเกณฑ์ดังนี้

```
MACRONAME      MACRO      [PARAMETER-1] [,PARAMETER-2],
               STATEMENTS
               ENDM
```

คำสั่งที่อยู่ระหว่าง MACRO และ ENDM จะไม่ถูกแปลจนกระทั่งแมคโครถูกเรียกใช้ MACRONAME อาจจะใช้สัญลักษณ์กำหนดขึ้นส่วนพารามิเตอร์เป็นส่วนประกอบ ตัวประกอบพารามิเตอร์อาจมีมากกว่า 1 ตัว โดยใช้เครื่องหมายคอมม่า (,) คั่นระหว่างพารามิเตอร์ ที่เขียนในบรรทัดเดียวกัน พารามิเตอร์เหล่านี้เราเรียกว่า Dummy parameters แทนค่าที่กำหนดในแมคโคร เมื่อแมคโครต้องการใช้

คำสั่งทั้งหมดในแมคโครก่อนถึงคำสั่งเทียม ENDM ซึ่งเป็นส่วนประกอบของแมคโคร ถ้าไม่มีการเรียกใช้ คำสั่งเหล่านี้จะไม่ถูกแปล

## การเรียกแมคโคร

การเรียกแมคโครจะเรียกชื่อ MACRONAME และตามด้วยค่าที่จะให้กับแมคโคร มีกฎเกณฑ์ดังนี้

```
MACRONAME      [ARGUMENT-1][,ARGUMENT-2]...
```

MACRONAME จะต้องเป็นชื่อที่กำหนดโดยแมคโครในโปรแกรม ส่วน ARGUMENT คือค่าที่ส่งผ่านให้กับแมคโคร ซึ่งนำไปแทนค่าพารามิเตอร์ที่กำหนดในแมคโคร ค่า ARGUMENT จะต้องเป็นไปตามค่าของพารามิเตอร์เดิม แต่จำนวนของ ARGUMENT ไม่สามารถเข้ากับพารามิเตอร์ได้ หรือมี ARGUMENT มากเกินไป มันไม่สามารถรับได้หรือน้อยเกินไป

ตัวอย่าง DISPLAY MACRO เป็นการสร้างแมคโครในการประมวลผลชุดสตริง โดยมีการกำหนดแอดเดรสของชุดสตริงใน DX ไม่มีพารามิเตอร์กำหนดในแมคโคร

```
DISPLAY     MACRO
            PUSH AX
            MOV  AH,9
            INT  21H
            POP  AX
            ENDM
```

การเขียนแมคโครในการกับสู่ระบบ DOS

```
DOS_RTN     MACRO
            MOV  AH,4CH
            INT  21H
            ENDM
```

การเขียนแมคโครในการขึ้นบรรทัดใหม่

```
NEW_LINE    MACRO
            MOV  AH,2
            MOV  DL,0DH
            INT  21H
            MOV  DL,0AH
            INT  21H
            ENDM
```

การเขียนแมคโครแสดงตัวอักขระสตริง สตริงเป็นพารามิเตอร์ของแมคโคร

```
DISP_STR  MACRO     STRING
          LOCAL     START,MSG
          ;SAVE REGISTERS
          PUSH AX
          PUSH DX
          PUSH DS
          JMP  START
MSG       DB   STRING,'S'
```

```
START:
        MOV  AX,CS
        MOV  DS,AX      ;SET DS TO CODE SEGMENT
        MOV  AH,9
        LEA  DX,MSG
        INT  21H
        ;RESTORE REGISTER
        POP  DS
        POP  DX
        POP  AX
        ENDM
```

ตัวอย่างการใช้แมคโคร DISP_STR

```
        DISP_STR  'THIS IS STRING'
```

การใช้แมคโคร LIBRALY

```
        TITLE   EX1.ASM: MACRO DEMO
        .MODEL SMALL
        .STACK    100H
        IF1
            INCLUDE   MACROS
        ENDIF

        .CODE
MAIN    PROC
        DIS_STR   'THIS IS THE FIRST LINE'
        NEW_LINE
        DISP_STR  'AND THIS IS THE SECOND LINE'
        DOS_TRN
MAIN    ENDP
        END  MAIN
```

```
C:\> EX1
     THIS IS THE FIRST LINE
     AND THIS IS THE SECOND LINE
```

แมคโครก็จะขยายโปรแกรม EX1.asm เป็นรหัสภาษาเครื่องดังต่อไปนี้

```
                        TITLE   EX1.ASM: MACRO DEMO
                        .MODEL SMALL
                        .STACK    100H
                        .CODE
0000            MAIN    PROC
                      DIS_STR   'THIS IS THE FIRST LINE'
1                       PUSH      AX
1                       PUSH      DX
1                       PUSH      DS
1                       JMP       ??0000
1 0000 ??0001           DB        'THIS IS THE FIRST LINT','$'
1 0017 ??0000:
1                       MOV   AX,CX
1                       MOV   DS,AX      ;SET DX TO CODE SEGMENT
1                       MOV   AH,9
1                       LEA   DX,??0001
1                       INT   21H
1                       POP   DS
1                       POP   DX
1                       POP   AX
NEW_LINE
1                       MOV   AH,2
1                       MOV   DL,0DH
1                       INT   21H
1                       MOV   DL,0AH
1                       INT   21H
                      DIS_STR   'AND THIS IS THE SECOND LINE'
1                       PUSH      AX
1                       PUSH      DX
1                       PUSH      DS
1                       JMP       ??0002
1 0040 ??0003           DB        'AND THIS IS THE SECOND LINT','$'
1 0059 ??0002:
1                       MOV   AX,CX
1                       MOV   DS,AX      ;SET DX TO CODE SEGMENT
1                       MOV   AH,9
1                       LEA   DX,??0003
```

```
        1           INT   21H
        1           POP   DS
        1           POP   DX
        1           POP   AX
DOS_TRN
        1           MOV   AH,4CH
        1           INT   21H
MAIN    ENDP
            END   MAIN
```

การเขียนใน Procedure เราจะต้องมีการ PUSH และ POP ค่าของรีจีสเตอร์ภายในแมคโคร เพื่อการสำรองค่าเดิม เราสามารถเขียนการเรียกแมคโคร DISPLAY ดังนี้

```
            MOV   DX,OFFSET MESSAGE
            DISPLAY
            -

MESSAGE     DB    'THIS IS STRING.',0DH,0AH,'$'
```

**โปรแกรมการใช้ MACRO LIBRAY**

```
    ;PROGRAM TO ILLUSTRATE OF A MACRO LIBRARY
IF1                                     ;NOTICE TO LOAD A PRREVIOUSLY
    INCLUDE  B:MACLIB.MAC               ;SAVE MACRO LIBRARY FROM B:
ENDIF

STACK     SEGMENT   PARA STACK
          DB   64 DUP (0)
STACK     ENDS
MYDATA    SEGMENT   PARA 'DATA'
MESSAGE   DB   'I AM A SIMPLE COUNTING PROGRAM$'
MYDATA    ENDS
```

```
MYCODE    SEGMENT   PARA 'CODE'
MYPROC    PROC      FAR
          ASSUME    CS:MYCODE,DS:MYDATA,SS:MYSTACK
          PUSH DS
          SUB  AX,AX
          PUSH AX
          MOV  AX,MYDATA
          MOV  DS,AX
          CLEARSCREEN
          CURSOR    0019H
          DISPLAY   MESSAGE
          MOV  AH,4CH
          INT  21H
MYPROC    ENDP
MYCODE    ENDS
          END
```

## การใช้แมครโครซ้ำ

แมครโคร REPT ใช้ทำงานซ้ำของคำสั่งในกลุ่ม มีกฎเกณฑ์ดังนี้

```
REPT EXPRESSION
     STATEMENT
ENDM
```

ตัวอย่างการทำงานของแมครโครซ้ำมีดังนี้

```
A    LABEL    WORD
     REPT     5
     DW       0
     ENDM
```

คำสั่ง LABEL เป็นคำสั่งเทียมเพื่อการกำหนดข้อมูลเป็นเวิร์ด สามารถขยายได้ดังนี้

```
A    DW   0
     DW   0
     DW   0
     DW   0
     DW   0
```

การเขียนแมคโครในการเซ็ตค่าเริ่มแรกของบล๊อคในหน่วยความจำ เป็นค่าจำนวนเต็ม ค่าแรกจนถึงค่า N ถ้าโปรแกรมเซ็ตค่าอะเรย์จำนวน 100 ค่า ดังนี้

```
BLOCK    MACRO    N
         K = 1
         REPT     N
         DW       K
         K = K + 1
         ENDM
         ENDM
```

สังเกตว่าแมคโครใช้ค่าเท่ากับ (=) เป็นคำสั่งเทียมมีค่าเป็น EQU สามารถกำหนดให้ค่าคงที่กับ NAME ค่าที่อยู่ทางขวาของเครื่องหมาย = เป็นค่าตัวแปรที่เปลี่ยนได้ เช่น K = K + 1 เป็นกำหนดอะเรย์ตั้งแต่ 1 ถึง 100 ในส่วนของ DATA SEGMENT

```
A    LABEL    WORD
     BLOCK    100
```

สามารถขยายได้ดังนี้

```
A    DW    1
     DW    2
     -
     -
     DW    100
```

จงเขียนแมคโครในการเซ็ตค่าเริ่มแรก N-WORD ARRAY เป็น 1!,2!,3!,...n! แสดงดังนี้

```
FACTORIALS    MACRO    N
              M = 1
              FAC  =  1
              REPT   N
              DW     FAC
              M  =  M + 1
              FAC = M * FAC
              ENDM
              ENDM
```

การกำหนดเวิร์คอะเรย์ B ในการหาค่า FACTORIALS 8 ของ DATA SEGMENT

```
B       LABEL       WORD
        FACTORIALS      8
```

ค่าของ 8! = 40320 เป็นค่าสูงสุดสำหรับเก็บในข้อมูลขนาด 16 บิต

```
B       DW      1
        DW      2
        DW      6
        DW      24
        DW      120
        DW      720
        DW      5040
        DW      40320
```

## แมคโคร IRP

เป็นคำสั่งของแมคโครที่ให้ทำงานซ้ำๆกัน มีรูปแบบต่อไปนี้

```
IRP d,<a1,a2,a3,...an>
     STATEMENTS
ENDM
```

ตัวอย่างการเขียนแมคโครเก็บค่าของรีจีสเตอร์และโหลดค่าให้กับรีจีสเตอร์ดังนี้

```
SAVE_REGS       MACRO       REGS
                IRP   D,<REGS>
                PUSH  D
                ENDM
                ENDM

RESTORE_REGS    MACRO       REGS
                IRP   D,<REGS>
                POP   D
                ENDM
                ENDM
```

ถ้าเป็นการเก็บค่าของรีจีสเตอร์ AX,BX,CX,DX เราสามารถเขียนได้ดังนี้

```
SAVE_REGS       <AX,BX,CX,DX>
```

ทำงานได้เป็นดังนี้

```
PUSH AX
PUSH BX
PUSH CX
PUSH DX
```

หรือเป็นการโหลดข้อมูลให้กับรีจีสเตอร์ดังนี้

```
RESTORE_REGS    <DX,CX,BX,AX>
```

## แมคโครเอ้าพุต

การใช้โครงสร้างของแมคโครในการเขียนแมคโคร HEX_OUT แสดงข้อมูลเลขฐานสิบหก 4 ตัว ซึ่งมีอัลกอริทึมดังนี้

```
ALGORITHM FOR HEX OUTPUT (OF BX)

1: FOR 4 TIMES DO
2: MOVE BH,DL
3: SHIFT DL 4 TIMES TO THE RIGHT
4: IF DL < 10
5: THEN
6: CONVERT CONTENTS OF DL TO A CHARACTER IN '0'..'9'
7: ELSE
8: CONVERT CONTENTS OF DL TO A CHARACTER IN 'A'..'F'
9: END_IF
10: OUTPUT CHARACTER
11: ROTATE BX LEFT 4 TIMES
12: END_FOR
```

โปรแกรม HEX_OUTPUT MACRO DEMO

```
                TITLE  EX3.ASM HEX_OUTPUT MACRO DEMO
                .MODEL SMALL
SAVE_REGS       MACRO     REGS
                IRP  D,<REGS>
                PUSH D
                ENDM
                ENDM
RESTORE_REGS    MACRO     REGS
                IRP  D,<REGS>
                POP  D
                ENDM
                ENDM
CONVERT_TO_CHAR MACRO     BYTE
                LOCAL     ELSE_,EXIT
;Convert contents of BYT to hex digit char
;if

                CMP  BYT,9      ;contents <= 9?
                JNLE  LSE_      ;no >= Ah
;then
                OR   BYT,30     ;convert to digit char
                jmp  exit
ELSE_:
                ADD  BYT,37H    ;convert to digit char
EXIT:
                ENDM
;-----------------------------------------------
DISP_CHAR       MACRO    BYT
;display contents of BYT
                PUSH AX
                MOV  AH,2
                MOV  DL,BYT
                INT  21H
                POP  AX
                ENDM
;-----------------------------------------------
```

```
HEX_OUTPUT      MACRO      WRD
;display contents of WRD as 4 hex digits
                SAVE_REGS <BX,CX,DX)
                MOV  BX,WRD
                MOV  CL,4              ;SHIFT AND ROTATE COUNT
                REPT 4
                MOV  DL,BH
                SHR  DL,CL             ;SHIFT RIGHT 4 TIMES
                CONVERT_TO_CHAR  DL ;CONVERT DL TO DIGIT CHAR
                DISP_CHAR        DL    ;DISPLAY  D
                ROL  BX,CL             ;ROTATE LEFT 4 TIMES
                ENDM
                RESTORE_REGS    <DX,CX,BX>
                ENDM
      ;-----------------------------------------------

                .STACK
                .CODE
      ;PROGRAM TO TEST ABOVE MACROS
      MAIN      PROC
                MOV  AX,1AF4H          ;TEST DATA
                HEX_OUT    AX          ;DISPLAY IN HEX
                MOV        AH,4CH      ;DOS EXIT
                INT        21H
           MAIN ENDP
                END  MAIN
```

```
A:\> EX3
     1AF4
```

## การทำแมคโครซ้อนแมคโคร

การใช้แมคโครซ้อนแมคโครหรือเรียกว่า NESTED MACROS เพื่อเรียกแมคโครอื่นๆ การสร้างแมคโครที่เรียกว่า DISPLAY_AT เพื่อแสดงสตริงที่ตำแหน่ง ROW และ COLUMN เรียกว่า LOCATE และ DISPLAY ซึ่งแมคโครจะส่งผ่านพารามิเตอร์ผ่าน ARGUMENT ไปให้แมคโครทั้ง 2

```
DISPLAY_AT      MACRO       ROW,COLUMN,STRING
                LOCATE      ROW,COLUMN
                DISPLAY     STRING
                ENDM
```

ตัวอย่างการเรียกใช้แมคโคร

```
                DISPLAY_AT      10,15,GREEN
                    -
GREEN           DB      'HELLO$'
```

## คำสั่งเทียม LOCAL

คำสั่งเทียม LOCAL เป็นตัวบอก MASM ในการสร้างชื่อสำหรับลาเบล แต่ละครั้งที่แมคโครเลือก มีกฎเกณฑ์ดังนี้

```
                LOCAL       LABELNAME
```

แมคโครอาจจะต้องใช้ลาเบลเพื่ออ้างถึงจุดสำหรับการกระโดดและคำสั่ง LOOP เช่นการสร้างแมคโคร NAME ว่า REPEAT เพื่อแสดงผลตัวอักขระตามจำนวนครั้งที่ต้องการ คำสั่ง LOCAL บอก MASM ที่จะเปลี่ยน L1 ถึงจุดสูงสุด (UNIQUE) แต่ละครั้งที่แมคโครเลือก

```
REPEAT          MACRO       CHAR,COUNT
                LOCAL       L1          ;DECLEAR A LOCAL LABEL
                MOV         CX,COUNT
    L1:         MOV         AH,2
                MOV         DL,CHAR
                INT         21H
                LOOP        L1
                ENDM
```

การเรียกใช้ REPEAT มากกว่า 1 ครั้ง เราจะเห็นว่า MASM จะสร้างลาเบลแตกต่างกันในแต่ละเวลา จำนวนของลาเบลจาก 0000H ถึง FFFFH และนำหน้าด้วย ? (Question marks) 2 ตัว

```
                REPEAT    'A',10          ;CALL MACRO
                MOV   CX,10
??0000:         MOV   AH,2
                MOV   DL,'A'
                INT   21H
                LOOP  ??0000

                REPEAT    '*',20          ;CALL MACRO
                MOV   CX,20
??0001:         MOV   AH,2
                MOV   DL,'*'
                INT   21H
                LOOP  ??0001
```

## การใช้แมคโครเรียก Procedure

การใช้งานร่วมกันระหว่างแมคโครกับ Procedure ดังนี้

```
CALL_WRITEINT   MACRO   VALUE,RADIX
                PUSH    AX          ;SAVE AX,BX
                PUSH    BX
                MOV     AX,VALUE    ;VALUE TO BE DISPLAY
                MOV     BX,RADIX    ;RADIX TO BE USED
                CALL    WRITEINT    ;DISPLAY AX ON COMSOLE
                POP     BX
                POP     AX
                ENDM
```

คำสั่ง CALL ของแมคโครสามารถใช้รีจีสเตอร์ ตัวแปร ค่าคงที่ ดังนี้

```
CALL_WRITEINT   MACRO   2000H,10        ;immediate value
CALL_WRITEINT   MACRO   DX,16           ;register value
CALL_WRITEINT   MACRO   WORDVALUE2,2    ;memory value
```

**คำสั่งเทียมที่กำหนดเงื่อนไขภาษาแอสแซมบลี**

คำสั่งเทียมที่กำหนดเงื่อนไขในภาษาแอสแซมบลี มีการทำงานที่แตกต่างกัน 12 ชนิด อาจจะนำมาใช้เชื่อมกับแมคโคร เพื่อให้การทำงานสะดวกขึ้น กฎเกณฑ์โดยทั่วๆไปสำหรับคำสั่งเทียมกำหนดเงื่อนไขภาษาแอสแซมบลีคือ

```
IF    CONDITION
      STATEMENTS
      [ELSE
            STATEMENTS]
ENDIF
```

เรามีกฎเกณฑ์ดังต่อไปนี้ คำสั่งจะอนุญาตใช้แอสแซมบลี เราหมายถึงคำสั่งที่ตามหลังจนกระทั่งคำสั่ง ENDIF

**คำสั่ง IF มีกฎเกณฑ์คือ**

```
IF   expression
```

คำสั่งนี้ถ้าค่าของนิพจน์ (expression) มีค่าจริง (Nonzero) จากตัวอย่างต่อไปนี้

```
      IF   COUNT < 20:
```

เป็น

```
      IF   COUNT LT 20
```

ตัวทำงานของความสัมพันธ์ดังนี้ LT , GT , EQ , NE , LE และ GE

**คำสั่ง IFE มีกฎเกณฑ์ดังนี้**

```
IFE  expression
```

คำสั่งนี้ถ้าค่าของนิพจน์ (expression) มีค่าเป็น 0 (zero) หรือไม่จริง จากตัวอย่างต่อไปนี้

```
      IF   COUNT = 10:
```

เป็น

```
      IFE  (COUNT - 10)
```

**คำสั่ง IF1 มีกฎเกณฑ์ดังนี้**

```
IF1
```

คำสั่งนี้ถ้ามีค่าตรงกับเงื่อนไข แอสแซมเบอล์จะส่งไป PASS แรกจนถึง SOURCE PROGRAM

คำสั่ง IF2 มีกฎเกณฑ์ดังนี้

IF2

คำสั่งนี้ถ้ามีค่าตรงกับเงื่อนไข แอสแซมเบอล์จะส่งไป PASS ที่สองจนถึง SOURCE PROGRAM

คำสั่ง IFB มีกฎเกณฑ์ดังนี้

IFB <ARGUMENT>

คำสั่งนี้ถ้าค่าจริงของ ARGUMENT เป็น BLANK เพราะไม่มีค่าส่งให้แมคโคร ชื่อของ ARGUMENT จะต้องปิดด้วยวงเล็บ (< >)

คำสั่ง IFNB มีกฎเกณฑ์ดังนี้

IFNB <ARGUMENT>

คำสั่งนี้ถ้าค่าจริงของ ARGUMENT จะต้องมีค่า และส่งค่าต่อไปยังแมคโคร

คำสั่ง IFIDN มีกฎเกณฑ์ดังนี้

IFIDN <ARGUMENT1>,<ARGUMENT2>

คำสั่งนี้ถ้าค่าจริงของ ARGUMENT ทั้งสองจะต้องมีค่าเท่ากัน

คำสั่ง IFDIF มีกฎเกณฑ์ดังนี้

IFDIF <ARGUMENT1>,<ARGUMENT2>

คำสั่งนี้ถ้าค่าจริงของ ARGUMENT ทั้งสองจะต้องมีค่าไม่เท่ากัน

คำสั่ง IFDEF มีกฎเกณฑ์ดังนี้

IFDEF name

คำสั่งนี้ถ้าค่าเป็นจริง ค่าของ NAME จะถูกกำหนดหรือ DECLEAR

คำสั่ง IFNDEF มีกฎเกณฑ์ดังนี้

IFNDEF name

คำสั่งนี้ถ้าค่าเป็นจริง ค่าของ NAME ไม่ได้กำหนดหรือ DECLEAR

คำสั่ง ENDIF เป็นจุดสุดท้ายของบล็อคในการใช้เงื่อนไขภาษาแอสแซมบลี

คำสั่ง ELSE ถ้าเงื่อนไขกำหนดของคำสั่ง IF ไม่เป็นจริงก็ทำงานในส่วนของ ELSE

การเขียนแมคโครในการกำหนดกลุ่มของหน่วยความจำชนิดเวิร์ด กับค่าจำนวน N ประกอบด้วยค่าจำนวนเต็มตัวแรกถึงค่า K ตามด้วย N - K เป็นเวิร์ด 0 และเซ็ตค่าของอะเรย์ 10 เวิร์ดด้วยค่า 1,2,3,4,5,0,0,0,0, และ 0 ดังนี้

```
BLOCK    MACRO    N,K
         I = 1
         REPT     N
         IF   K + 1 - I
         DW   1
         I = I + 1
         ELSE
         DW   0
         ENDIF
         ENDM
         ENDM
```

สามารถเขียนคำสั่งในโปรแกรมดังนี้

```
A    LABEL    WORD
     BLOCK    10,5
```

เราสามารถขยายการทำงานจาก N ถึง 10 , K ถึง 5 , I ถึง 1 และคำสั่ง REPT ทำงาน 10 ครั้ง หลังจากกำหนดค่า DW 1 จำนวน 5 ครั้ง คือ ถึง DW 5 จะทำให้ค่า K เป็นค่า 6 จะได้ K + 1 - I = 5 + 1 - 6 = 0 คำสั่งจะทำงานที่ ELSE ทำค่า DW 0 จะได้ผลลัพธ์ดังนี้

```
A    DW   1,2,3,4,5,0,0,0,0,0
```

**การตรวจสอบ ARGUMENT ของแมคโคร**

แมคโครอาจจะมีพารามิเตอร์มากกว่าหนึ่ง เราควรตรวจสอบว่าถ้าค่าของ ARGUMENT ที่ส่งผ่าน ไม่เป็นจริง MASM จะให้ค่า BLANK สำหรับพารามิเตอร์ ซึ่งมีความหมายของคำสั่งในการแอสแซมเบลอร์ สมมุติที่เราเรียกแมคโคร CALL_WRITEINT และไม่ส่งค่า RADIX แมคโครจะจะใช้แอสแซมเบอลร์ที่ผิดในบรรทัดที่ 5

```
MACRO CALL:
                    CALL_WRITEINT        1000H
```

จะเปลี่ยนเป็นคำสั่งดังต่อไปนี้

```
1:              CALL_WRITEINT
2:              PUSH    AX          ;SAVE AX,BX
3:              PUSH    BX
4:              MOV     AX,1000H    ;VALUE TO BE DISPLAYED
5:              MOV     BX,         ;RADIX TO BE USED (?)
6:              CALL    WRITEINT    ;DISPLAY AX ON COMSOLE
7:              POP     BX
8:              POP     AX
```

คำสั่ง IFB (ถ้า BLANK) ค่าเป็นจริง ถ้า ARGUMENT ของแมคโครเป็น BLANK และคำสั่งของ IFNB (ถ้าไม่ BLANK) ค่าเป็นจริง ถ้า ARGUMENT ของแมคโครกำหนดไว้

การใช้แมคโคร MYMAC ในการตรวจสอบค่า PARM1 ที่ส่งค่าให้แมคโคร ในการเรียกใช้ MYMAC ที่ไม่มี ARGUMENT คำสั่งเทียม EXITM จะป้องกันคำสั่งทุกคำสั่ง

```
MYMAC       MACRO           PARM1
            IFB  <PARM1>        ;NO ARGUMENT SUPPLIED?
               EXITM            ;THEN EXIT THE MACRO
            ENDIF               ;AND GENERATE NO STATEMENTS
            -
            -
            ENDM

            .CODE
            -
            -
            MYMAC               ;OMIT THE ARGUMENT
            MYMAC     VAL1      ;PASS AN ARGUMENT
```

## คำสั่งเทียม EXITM

คำสั่ง EXITM จะเป็นตัวบอกแอสแซมเบลอร์ ถึงการออกจากแมคโครและหยุดทำงานของคำสั่งแมคโคร เพื่อช่วยลดความยาวของโปรแกรม โดยการแยกคำสั่งที่จำเป็นเช่น แมคโคร GOTOXY เพื่อใช้กำหนดตำแหน่งเคอร์เซอร์บนจอภาพ เราต้องการออกจากแมคโครถ้า ARGUMENT น้อยกว่า 0 แต่ถ้าเงื่อนไข IF เปรียบเทียบกับค่า XVAL เป็น 0 ใช้ LT (LESS THAN) และออกจากผลลัพธ์เป็นจริง เช่นเดียวกับค่า YVAL

```
GOTOXY    MACRO     XVAL,YVAL
     IF   XVAL      LT   0     ;XVAL < 0 ?
          EXITM                ;IF SO , EXIT
     ENDIF

     IF  YVAL  LT   0                ;YVAL  <  0?
          EXITM                      ;IF SO , EXIT
     ENDIF

     MOV  BX,0       ;CHOOSE VIDEO PAGE 0
     MOV  AH,2       ;LOCATE CURSOR
     MOV  DH,YVAL
     MOV  DL,XVAL
     INT  10H
     ENDM
```

เงื่อนไขของคำสั่งเทียม เช่น คำสั่ง IF จะต้องตามด้วยกฎเกณฑ์ (Expression) ว่าค่าที่ประเมินเป็นจริงหรือเท็จ ในการทำงานของโปรแกรม กรณีจะไม่ทำงานสำหรับค่าตัวแปรในรีจีสเตอร์หรือค่าในหน่วยความจำ เพราะค่าเหล่านี้จะต้องรู้ในเวลาทำงาน คำสั่ง ENDIF จะใช้บอกจุดสิ้นสุดของคำสั่ง IF

**การแสดงข่าวสารภาษาแอสแซมบลี**

คำสั่งเทียม %OUT เป็นคำสั่งที่ใช้แสดงสตริงระหว่างการทำงานของภาษาแอสแซมบลี วิธีนี้ใช้กับแมคโคร GOTOXY ในการแสดงข่าวสาร เมื่อค่าของ ARGUMENT ผิด เราใช้คำสั่ง IF2 ในตัวอย่างต่อไปนี้ ในการใช้ %OUT ข่าวสารในการแสดงผลระหว่างแอสแซมเบลอร์ส่วนที่ 2

```
GOTOXY    MACRO     XVAL,YVAL
     IF   XVAL LT   0     ;XVAL     <     0?
          IF2
          %OUT GOTOXY: FIRST AEGUMENT (XVAL) IS INVALID.
          %OUT (VALUE MUST BE >= 0)
     ENDIF
ENDIF
```

```
        IF   YVAL   LT  0          ;YVAL   <   0?
             IF2
             %OUT GOTOXY: SECOND AEGUMENT (YVAL) IS INVALID.
             %OUT (VALUE MUST BE >= 0)
        ENDIF
ENDIF
             MOV  BX,0        ;CHOOS VIDEO PAGE 0
             MOV  AH,2        ;LOCATE CURSOR
             MOV  DH,YVAL
             MOV  DL,XVAL
             INT  10H
             ENDM
```

การเรียกใช้แมคโคร GOTOXY กับค่า ARGUMENT2 ค่าที่ผิดคือ -1 และ -2 เพื่อแสดงข่าวสารใช่วงของภาษาแอสแซมบลี

```
GOTOXY : FIRST ARGUMENT (-1) IS INVALID.
(VALUE MUST BE >= 0)
GOTOXY: SECOND ARGUMENT (-2) IS INVALID.
(VALUE MUST BE >= 0)
```

เพื่อความเข้าใจ ช่วงของการตรวจสอบพารามิเตอร์ คือทางที่ถูกต้องเพื่อให้แมคโครแน่ใจและข่าวสารคือจะช่วยในการเรียนรู้แมคโคร

**แมคโคร WRITE**

เราสร้างแมคโครที่เรียกว่า WRITE เพื่อเป็นเอ้าพุตมาตราฐาน ค่าที่ส่งผ่านไปยัง ARGUMENT จะอยู่ในเครื่องหมาย Single Quotation ส่วนพารามิเตอร์ประกอบคือ CRETURN เพื่อกำหนดค่า CR ในการแสดงผล

```
WRITE       MACRO     TEXT,CRETURN
            LOCAL     STRING,STRLEN  ;LOCAL LABELS
            PUSH      AX
            PUSH      DX
            MOV       AH,40H     ;DISPLAY THE STRING
            MOV       BX,1       ;HANDLE = CONSOLE
            MOV       DX,OFFSET STRING
            MOV       CX,STRLEN
            INT       21H
            POP       DX
            POP       AX
```

```
        .DATA           ;DEFINE THE STRING
IFB     <CRETURN>       ;CR/LF SPECIFIED?
        STRING    DB     TEXT
ELSE
        STRING    DB    TEXT,0DH,0AH
ENDIF
        STRLEN    EQU   $ - STRLEN

        .CODE
ENDM
```

จากตัวอย่างแสดงส่วนของข้อมูล .DATA จะรวมอยู่ในแมคโครก่อนการกำหนดตัวแปร ค่าของตัวแปรสตริง แอสแซมเบลอร์จะรู้อยู่ใน Data segment คำสั่ง LOCAL แน่ใจว่า STRING และ STRLEN ไม่กำหนดในชื่ออื่นๆของโปรแกรม

แมคโคร WRITE จะแสดงผลที่ Console เพื่อหลีกเลี่ยงการกำหนดรหัสข้อมูลโดย DB สำหรับค่าสตริงในตัวอย่างแรก แสดงการใช้ ARGUMENT N ของทุกๆตัวอักษรตามความยาวที่ไม่มี BLANK

```
WRITE 'HELLO THERE',N             ;INCLUDE CR/LF
WRITE  'NO RETURN ON THIS LINE'  ;NO CR/LF
```

**การใช้ INCLUDES จาก MACRO LIBRARY**

การเขียนแมคโครเช่น INITZ เราสามารถกำหนด MACRO LIBRARY ที่มีชื่อว่า MACRO.LIB

```
INITZ     MACRO
          MOV   AX,@DATA
          MOV   DS,AX
          MOV   ES,AX
          ENDM

PROMPT    MACRO MESSAGE
;THIS MACRO DISPLAYS ANY MESSAGE
          MOV   AH,9
          LEA   DX,MESSAGE
          INT   21H
          ENDM
```

สมมุติว่าเราเก็บค่าของแมคโครเหล่านี้ไว้ใน MACRO ที่จุดเริ่มต้นของโปรแกรม เราสามารถใช้คำสั่งเทียม INCLUDE ดังนี้

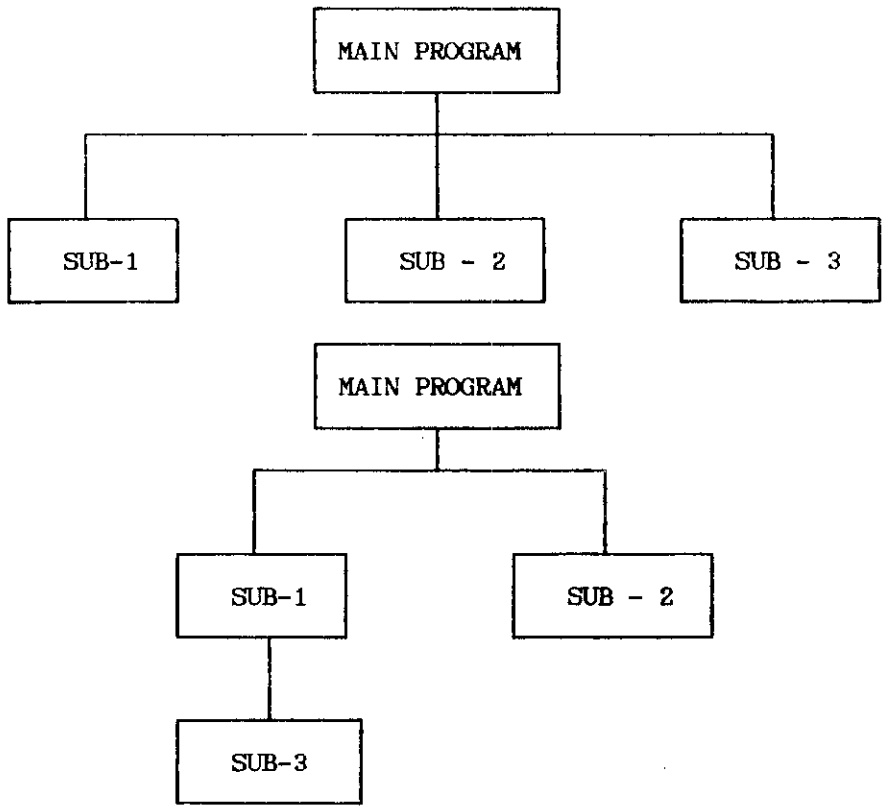
```
INCLUDE C:MACRO.LIB
....
```

แอสแซมเบลอจะเข้าถึงแฟ้มข้อมูลที่ชื่อ MACRO.LIB ในไดร์ฟ C: และจะรวมชื่อแมคโครคือ INITZ และ PROMPT รวมกับโปรแกรม ตัว MASM จะแบ่งออกเป็น 2 ส่วน ท่านสามารถเห็นการทำงานของคำสั่งเทียม INCLUDE เกิดขึ้นในส่วนแรก (Pass 1) โดยใช้คำสั่ง IF1 และ ENDIF คือเงื่อนไขของคำสั่งเทียม คำสั่ง IF1 จะบอกแอสแซมเบลอในการเข้าถึงชื่อใน LIBRARY ในส่วนแรกของแอสแซมเบลอ คำสั่ง ENDIF เป็นการสิ้นสุดของคำสั่ง IF หลังจากนั้นก็จะทำสำเนาคำจำกัดความของแมคโครลงในพื้นที่ว่าง

```
IF1
     INCLUDE  C:\MACRO.LIB
ENDIF
```

## LINKING TO SUBPROGRAMS

ตัวอย่างของโปรแกรมจะแบ่งออกเป็นโมดุล อย่างไรก็ตามในการเอ็กซีคิวส์โปรแกรมประกอบไปด้วยโปรแกรมหลัก และโปรแกรมย่อยอื่นๆนำมารวมกัน ด้วยเหตุผลของการจัดระบบงานของโปรแกรม เป็นโปรแกรมย่อย






CT 215 409

จากภาพที่แสดงลำดับขั้นของโปรแกรมโมดุล 2 แบบ ที่มีโปรแกรมย่อย 3 โปรแกรม ในรูปแรก และรูปที่ 2 ในรูปแรก โปรแกรมหลักจะเรียกโปรแกรม 1,2,3 ส่วนในรูปที่ 2 โปรแกรมหลักจะเรียกโปรแกรมที่ 1,2 และโปรแกรมย่อย 1 เรียกโปรแกรมย่อย 3 ลำดับของโปรแกรมย่อยที่แตกต่างกันในระบบงาน แต่ระบบงานจะต้องให้แอสแซมเบลอทำงานร่วมกัน การ LINKER และ EXECUTION การทำงานของโปรแกรมจะทำงานแบบการเวียนเกิด ( Recursion)

## SEGMENTS

คำสั่งเทียม SEGMENT เป็นการใช้ SEGMENTS มีรูปแบบของคำสั่งดังต่อไปนี้

| SEG - NAME | SEGMENT | [ALIGN] [COMBINE] [CLASS'] |
|---|---|---|

ALIGN TYPE

ตัวโอเปอร์เรเตอร์ของกลุ่มนี้ จะบอกแอสแซมเบลอถึงการกำหนดชื่อเซคเม้นต์ที่จุดเริ่มต้นของหน่วยความจำ

BYTE ชนิดไบท์ สำหรับเซคเม้นต์ของโปรแกรมย่อยว่า จะรวมกับโปรแกรมอื่นๆ
WORD ชนิดเวิร์ด สำหรับเซคเม้นต์ของโปรแกรมย่อยว่า จะรวมกับโปรแกรมอื่นๆ
DWORD ชนิดสองเวิร์ด ปกติใช้กับ 80386/80486
PARA ชนิดองพารากราฟ เป็นการบอกใช้โปรแกรมหลักและโปรแกรมย่อย
PAGE ชนิดเพจ

COMBINE TYPE

ตัวโอเปอร์เรเตอร์ชนิดนี้จะบอกแอสแซมเบลอ และ LINKER ของการรวมเซคเม้นต์ จะต้องใช้คำสั่ง NONE , PUBLIC และคำสั่ง COMMON

NONE ถ้ากำหนดแบบ NONE คือเซคเม้นต์จะแยกจากเซคเม้นต์อื่นๆ จะเป็นการเซ็ตค่าของเซคเม้นต์ทั้งหมด
PUBLIC ตัว LINKER จะรวมกับเซ็คเม้นต์อื่นที่กำหนดโดย PUBLIC จะเป็นเซคเม้นต์เดียวกัน หรือ CLASS เดียวกัน วิธีการนี้จะเซ็ตค่าเซคเม้นต์ที่กำหนดตามคำสั่งเทียม
COMMON ถ้าเป็น COMMON เซคเม้นต์มีชื่อเดียวกันและ CLASS เดียวกัน ตัว LINKER จะให้ BASE ADDRESS เดียวกัน สำหรับการเอ็กซีคิวส์ เซคเม้นต์ที่ 2 จะดาบเกี่ยวกับเซคเม้นต์แรก เซคเม้นต์ที่ใหญ่ที่สุดต้องการความยาวของพื้นที่ COMMON

CLASS OPTION

ท่านสามารถกำหนดชื่อ CLASS ชนิดเดียวกัน ที่มีความสัมพันธ์ในเซคเม้นต์ ดังนั้นแอสแซมเบลอ และ LINKER จะนำเข้ามารวมกัน นั่นคือ Class ต่างๆจะเป็นเซคเม้นต์หนึ่ง แต่มันจะไม่รวมกันเป็นเซคเม้นต์ จะต้องมีคำสั่ง PUBLIC เป็นตัวกำหนด จุดเข้าของ Class จะต้องมีชื่อภายใต้การกำกับของ Single quotes ถึงแม้ว่าชื่อ CODE จะเป็นตัวกำหนด Code segment

คำสั่ง 2 ของ 2 คำสั่งต่อไปนี้ จะเป็นตัวจ่ายผลลัพธ์ ชื่อของ Code segment อิสระในขอบเขตของพารากราฟ

```
CODESEG    SEGMENT    PARA NONE 'CODE'
CODESEG    SEGMENT    'CODE'
```

## INTRASEGMENT CALLS

คำสั่ง CALL ใช้สำหรับการเรียกแบบ INTRASEGMENT CALLS นั่นคือ ภายในชื่อ CODE SEGMENT เดียวกัน การทำงานแบบ INTRASEGMENT CALL เป็นชนิด NEAR ถ้าเป็นการเรียก Procedure จะต้องกำหนดค่า NEAR การทำงานของ CALL จะทำการ PUSH ค่า IP ลงในสแตคและแทนค่า IP ด้วยค่าออฟเซ็ตของแอดเดรสตัวรับ

เราจะพิจารณา INTRASEGMENT CALL จะประกอบด้วย Object code E8 2000 ขณะที่ E8 เป็นรหัสการทำงานและ 2000 เป็นค่าออฟเซ็ตของการเรียก Procedure การทำงานจะ PUSH ค่าของ IP ลงในสแตค และเก็บค่า 1000 คือ ออฟเซ็ต 0020 ใน IP สำหรับคำสั่งต่อไปที่จะนำมาเอ็กซีคิวส์ การออกจาก Procedure จะใช้คำสั่ง RET เพื่อจะทำการ POP ค่าของ IP ที่เก็บค่าในสแตค และย้อนกลับมานำคำสั่งออกไปเอ็กซีคิวส์ต่อจากคำสั่ง CALL

```
            CALL      nearproc        ;Near call: push IP,
            ....
nearproc    PROC      NEAR
            ....
            RET                       ;Near return : pop IP return
nearproc    rndp
```

## INTERSEGMENT CALLS

การ CALL ชนิด FAR ถ้ามีการเรียก Procedure ที่กำหนดเป็นชนิด FAR หรือ EXTRN การทำงานของการ CALL เริ่มแรกจะเป็นการ PUSH ข้อมูลของรีจีสเตอร์ CS ลงในสแตค และเพิ่มค่าของ INTERSEGMENT ADDRESS ลงใน CS แล้วก็ PUSH ค่า IP ลงในสแตค และเพิ่มค่าออฟเซ็ตแอดเดรดลงใน IP ในกรณีนี้ แอดเดรสทั้งสองใน Code segment และค่าออฟเซ็ต จะถูกเก็บไว้สำหรับการย้อนกลับจากการเรียกของการเรียกโปรแกรมย่อย การเรียกเซคเม้นต์อื่นๆ ก็ใช้วิธีการของ INTRASEGMENT FAR CALL

```
                CALL      farproc       ;Far call:push CS,
                ....                    ;IP,link to farproc
farproc         PROC FAR
                ......
                RET                   ;Far return :pop IP,CS
farproc         EDNP                  ;return
```

การพิจารณาการเรียก INTERSEGMENT CALL จะประกอบด้วย Object code คือ 9A 0002 AF04. เลขฐานสิบหก 9A คือรหัสการทำงานสำหรับการเรียกแบบ INTERSEGMENT CALL การทำงานจะเก็บค่า 0002 ที่ 0200 ใน IP และเก็บค่า AF04 ที่ 04AF ใน CS ซึ่งเป็นการรวมแอดเดรสของคำสั่งในการเรียกโปรแกรมย่อยที่จะมาเอ็กซีคิวส์

```
Code segment          04AF0
offset in IP      +   0200
Effective address     04CF0
```

การออกจากการเรียก Procedure ของการทำงาน INTERSEGMENT FAR ใช้คำสั่ง RET โดยการ POP ค่าเดิมของ IP และ CS เพื่อนำแอดเดรสเก็บมาไว้ที่เดิม เพื่อจะชี้แอดเดรสหลังคำสั่ง CALL เพื่อนำมาเอ็กซีคิวส์

### EXTRN AND PUBLIC ATTRIBUTES

การทำงานของโปรแกรมหลัก (MAINPROG) ในการเรียกโปรแกรมย่อย (SUBPROG) ของการเรียกแบบ INTERSEGMENT แสดงในการเขียนต่อไปนี้

```
                    EXTRN       SUBPROG:FAR
MAINPROG:           ......
                    CALL    SUBPROG

                    PUBLIC      SUBPROG
SUBPROG:            ......
                    ......
                    RET
```

การ CALL จากโปรแกรมหลัก (MAINPROG) จะต้องรู้ว่า SUBPROG อยู่นอกเซคเม้นต์ หรือแอสแซมเบลอจะบอกข้อผิดพลาดว่า SUBPROG ไม่ได้กำหนดสัญลักษณ์ คำสั่งเทียม EXTRN SUBPROG: FAR จะเป็นรูปแบบของการกำหนด SUBPROG เพื่อเป็นตัวบอกแอสแซมเบลอร์ถึงโปรแกรม SUBPROG อยู่ต่างเซคเม้นต์กัน แอสแซมเบลอก็จะรู้ว่ามี SUBPROG และก็จะสร้าง Object code operand ของคำสั่ง CALL กับตัว LINKER

```
9A 0000 -------- E   CALL SUBPROGRAM
```

SUBPROG เป็นข้อมูลของคำสั่งเทียม PUBLIC ที่จะบอกแอสแซมเบลอและ LINKER ว่าโมดุลอื่นๆให้รู้จักแอดเดรสของ SUBPROG เราสามารถเขียนการทำงานของ MAINPROG กับ SUBPROG โดยการแอสแซมเบลอให้อยู่ในรูป Object code และนำมา LINK ดังต่อไปนี้

| LINK PROMPT | REPLY |
|---|---|
| Object Modules[.obj]: | b:MAINPROG + b:SUBPROG |
| Run File[filespec.EXE]: | b:COMBPROG (or any legal name) |
| List File[NUL.MAP]: | CON |
| Libraries[.LIB]: | [Enter] |

ตัว LINKER จะทำการแมตส์คำสั่ง EXTRN ในรูปของ Object module กับคำสั่งเทียม PUBLIC ในโมดุลอื่นๆ และเพิ่มออฟเซ็ตแอดเดรสตามต้องการ การรวมโมดุล 2 โมดุลให้เป็นเอ็กซีคิวส์เพียงโมดุลเดียว ถ้าการเชื่อมของ 2 โมดุลไม่แมตส์กัน ตัว LINKER จะบอกข้อผิดพลาดให้แก้ไข

## คำสั่งเทียม EXTRN

คำสั่งเทียม EXTRN จะเป็นตัวบอกแอสแซมเบลอว่าชื่อในรายการของคำสั่ง ข้อมูลในรายการ โพรซีเยอร์ หรือ ลาเบล ที่กำหนดในภาษาแอสแซมบลี มีรูปแบบดังต่อไปนี้

```
EXTRN   name:type[,.....]
```

ท่านสามารถกำหนดได้มากกว่า 1 ชื่อขึ้นไปและสิ้นสุดของบรรทัดหรือรหัสของคำสั่งเทียม EXTRN โมดุลของแอสแซมบลีอื่นๆจะต้องกำหนดชื่อ และกำหนดใน PUBLIC ชนิดของจุดเข้าอาจจะเป็นค่าคงที่ เช่น BYTE , DWORD , FAR , NEAR , WORD หรือ ค่าที่กำหนดในคำสั่ง EQU จะต้องอยู่ในเทอมของชื่อที่กำหนด

.BYTE , WORD และ DWORD กำหนดรายการข้อมูลของโมดุลที่อ้างถึงหรือโมดุลที่กำหนด
.NEAR และ FAR กำหนดโพรซีเยอร์และลาเบลของคำสั่งในโมดุลที่อ้างถึง

## THE PUBLIC DIRECTIVE

คำสั่งเทียม PUBLIC จะบอกแอสแซมเบลอและ LINKER ว่าแอดเดรสที่กำหนดในรูปสัญลักษณ์ในภาษาแอสแซมบลี คือค่าของดมดุลอื่นๆที่ต้องการ มีรูปแบบดังนี้

```
PUBLIC  symbol [,.....]
```

ท่านสามารถกำหนดมากกว่า 1 สัญลักษณ์ขึ้นไปจนสิ้นสุดบรรทัด หรือการเพิ่มรหัส PUBLIC สัญลักษณ์กำหนดเป็นลาเบล (รวมทั้งลาเบล PROC) ค่าตัวแปร หรือค่าจำนวน ไม่สามารถกำหนดในรูปรีจีสเตอร์ ค่าที่กำหนดใน EQU ที่กำหนดมากกว่า 2 ไบท์ไม่ได้

## การใช้คำสั่ง EXTRN และ PUBLIC เป็นลาเบล

จากตัวอย่างในโปรแกรมประกอบด้วยโปรแกรมหลัก CALLMUL1 และโปรแกรมย่อย SUBMUL1 ซึ่งโปรแกรมทั้งสองใช้คำสั่งเทียม SEGMENT โปรแกรมหลักจะกำหนดเซคเม้นต์ของ Stack , Data และ Code ในส่วนของ Data segment จะกำหนด QTY และ PRICE ส่วนใน Code segment จะทำการโหลดค่า AX ด้วย PRICE และโหลดค่า BX ด้วย QTY หลังจากนั้นก็เรียกโปรแกรมย่อย คำสั่ง EXTRN ในโปรแกรมหลักจะกำหนดจุดเข้าในการเรียกโปรแกรมย่อย SUBMUL1.

ในโปรแกรมย่อยจะมีคำสั่ง PUBLIC (หลังจากคำสั่ง ASSUME) กำหนดค่าลาเบล SUBMUL1 เพื่อให้ตัว LINKER รู้กับตำแหน่งจุดเข้าเพื่อการเอ็กซีคิวส์ โปรแกรมย่อยจะเป็นการคูณข้อมูลของ AX (PRICE) ด้วย BX (QUANTITY) และผลคูณอยู่ที่ DX:AX อยู่ในรูปเลขฐานสิบหกคือ 002E 4000

ส่วนของโปรแกรมย่อยไม่มีการกำหนดข้อมูล ไม่ต้องการข้อมูล Data segment แต่จะทำหน้าที่จำข้อมูล และไม่มีการกำหนด Stack segment เพราะว่าใช้สแตคร่วมกับโปรแกรมหลัก

การตรวจสอบสัญลักษณ์ในตารางตามภาษาแอสแซมบลีแต่ละคำสั่ง สังเกตว่าสัญลักษณ์ในตารางสำหรับโปรแกรมหลัก แสดงใน SUBMUL1 ที่ FAR และ External สัญลักษณ์ในตารางของโปรแกรมย่อย แสดงใน SUBMUL1 ที่ F และ GLOBAL เทอมของ GLOBAL จะเป็นชื่อที่รู้จักคือ GLOBALLY ที่อยู่นอกของโปรแกรมย่อย

การ LINK MAP ที่จุดสุดท้ายของ Listing แสดงโครงสร้างของโปรแกรมในหน่วยความจำ สังเกตว่ามรส่วนของ Code segment อยู่ 2 ส่วน แต่ในภาษาแอสแซมบลีจะแตกต่างที่แอดเดรสเริ่มแรก แต่นำมารวมกันตามแบบของ NONE จะทำให้มีการเรียงลำดับในการ LINK กับโปรแกรมหลัก ในกรณีแอดเดรสเริ่มแรกของ Code segment ในโปรแกรมหลักมีค่าออฟเซ็ตคือ 00090 และแอดเดรสของ Code segment ในส่วนของโปรแกรมย่อยคือ 000B0

Trace ของโปรแกรมเอ็กซีคิวส์ในส่วนของรีจีสเตอร์ CS จะมีข้อมูล 141E[0] และ CALL SUBMUL1

9A 000 2014 (YOUR SEGMENT VALUE MAY DIFFER)

รหัสภาษาเครื่องสำหรับ INTERSEGMENT CALL คือ 9A การทำงานจะ PUSH ค่ารีจีสเตอร์ IP ลงในสแตคและโหลดค่า 0000 ลงใน IP มันจะ PUSH ค่าข้อมูลของ CS 141E[0] ลงในสแตคและโหลดค่า 1420[0] ในรีจีสเตอร์ CS (ในตัวอย่างแสดงข้อมูลของรีจีสเตอร์แสดงปกติจะไม่กลับข้อมูลไบท์ต่ำกับไบท์สูง) คำสั่งที่เอ็กซีคิวส์ต่อไปคือ CS:IP หรือ 1420[0] บวก 0020 ค่าอะไรที่อยู่ในตำแหน่ง 14200 เป็นจุดเริ่มต้นของโปรแกรมหลักกับค่ารีจีสเตอร์ CS มีข้อมูล 141E[0] เกิดขึ้นตามการ MAP ค่าออฟเซ็ตของ Code segment หลักคือ 00090 และค่าออฟเซ็ตของโปรแกรมย่อยคือ 000B0 , 20H BYTES เพิ่มจากโปรแกรมหลัก โดยค่า CS + 20H เพื่อเป็นค่า Effective address ของโปรแกรมย่อยในส่วนของ Code segment

| | | |
|---|---|---|
| CS address | | 141E0 |
| IP offset | + | 00020 |
| Effective address | | 14200 |

ตัว LINKER ต้องการค่าแอดเดรสนี้นำมาแทนโอเปอร์รานด์การเรียก (CALL) โปรแกรม SUBMUL1 จะคูณด้วย 2 ค่า กับผลคูณที่ได้อยู่ใน DX:AX และ ย้อนกลับมาที่โปรแกรมหลักด้วยคำสั่ง RET มาที่ CALLMUL1



CT 215 415

```
                        TITLE   CALLMUL1 (EXE)  Call subprogram
                                EXTRN  SUBMUL1:FAR
                        ; ---------------------------------------------
0000                        STACKSG SEGMENT PARA STACK 'Stack'
0000 0040[????]                 DW      64 DUP(?)
0080                        STACKSG ENDS
                        ; ---------------------------------------------
0000                        DATASG  SEGMENT PARA 'Data'
0000 0140                   QTY     DW      0140H
0002 2500                   PRICE   DW      2500H
0004                        DATASG  ENDS
                        ; ---------------------------------------------
0000                        CODESG  SEGMENT PARA 'Code'
0000                        BEGIN   PROC    FAR
                                    ASSUME CS:CODESG,DS:DATASG,SS:STACKSG
0000 B8 ---- R                      MOV     AX,DATASG
0003 8E D8                          MOV     DS,AX
0005 A1 0002 R                      MOV     AX,PRICE        ;Set up price
0008 8B 1E 0000 R                   MOV     BX,QTY          ;  and quantity
000C 9A 0000 ---- E                 CALL    SUBMUL1         ;Call subprogram
0011 B4 4C                          MOV     AH,4CH          ;Return
0013 CD 21                          INT     21H
0015                        BEGIN   ENDP
0015                        CODESG  ENDS
                                    END     BEGIN

Segments and Groups:
                N a m e         Length  Align   Combine Class
CODESG . . . . . . . . . . . . . 0015   PARA    NONE    'CODE'
DATASG . . . . . . . . . . . . . 0004   PARA    NONE    'DATA'
STACKSG  . . . . . . . . . . . . 0080   PARA    STACK   'STACK'
Symbols:
                N a m e         Type    Value   Attr
BEGIN  . . . . . . . . . . . . . F PROC  0000    CODESG  Length = 0015
PRICE  . . . . . . . . . . . . . L WORD  0002    DATASG
QTY  . . . . . . . . . . . . . . L WORD  0000    DATASG
SUBMUL1  . . . . . . . . . . . . L FAR   0000            External
```

```
                        TITLE   SUBMUL1 Called subprogram
                        ; -------------------------------
0000                    CODESG  SEGMENT PARA 'Code'
0000                    SUBMUL1 PROC    FAR
                                ASSUME CS:CODESG
                                PUBLIC SUBMUL1
0000 F7 E3                      MUL     BX              ;AX = price, BX = qty
0002 CB                         RET                     ;DX:AX = product
0003                    SUBMUL1 ENDP
0003                    CODESG  ENDS
                                END     SUBMUL1

Segments and Groups:
                N a m e         Length  Align   Combine Class
CODESG . . . . . . . . . . . . . 0003   PARA    NONE    'CODE'
Symbols:
                N a m e         Type    Value   Attr
SUBMUL1  . . . . . . . . . . . . F PROC  0000    CODESG  Global  Length=0003
```

```
Link Map
Object Modules: CALLMUL1+SUBMUL1

Start  Stop   Length Name              Class
00000H 0007FH 00080H STACKSG          STACK
00080H 00083H 00004H DATASG           DATA
00090H 000A4H 00015H CODESG           CODE    <-- Note: 2 code
000B0H 000B2H 00003H CODESG           CODE    <--   segments

Program entry point at 0009:0000
```

รูปที่ 14-1 ตัวอย่างโปรแกรมการใช้คำสั่ง EXTRN และ PUBLIC

## การใช้ PUBLIC ใน CODE SEGMENT

ตัวอย่างโปรแกรมต่อไปนี้ มีการเปลี่ยนแปลงโปรแกรมหลักในการเรียก CALLMUL2 และมีโปรแกรมย่อย SUBPROG2 ที่มีคำสั่ง PUBLIC อยู่ในคำสั่ง SEGMENT สำหรับ CODE คือ

```
CODESG    SEGMENT   PARA PUBLIC 'CODE'
```

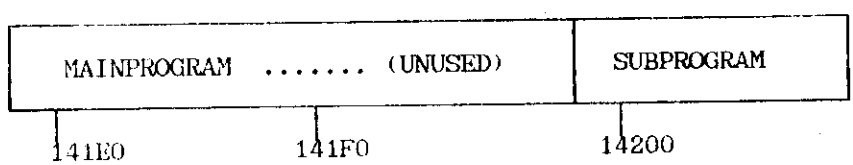
มีผลลัพธ์ที่น่าสนใจในการ LINK MAP และการ CALL Object code สัญลักษณ์ในตารางของภาษาแอสแซมบลีแต่ละคำสั่ง เราใช้ COMBINE TYPE สำหรับ CODESEG ในส่วนของ PUBLIC แต่ในตัวอย่างที่แล้ว ใช้แบบ NONE แต่สิ่งที่น่าสนใจกว่าของการ LINK MAP ที่ส่วนสุดท้ายของ CODE SEGMENT อันที่จริงเซคเม้นต์ทั้งสองมีชื่อเหมือนกัน (DATASG) ,Class ('CODE') และกำหนด PUBLIC attribute เพื่อให้ตัว LINKER รวม CODE SEGMENT ทั้งสอง ให้เป็น CODE SEGMENT เดียวกันทางกายภาพ การเอ็กซีคิวส์คำสั่งโดยการ Trace แสดงการใช้คำสั่ง CALL ในโปรแกรมแบบ FAR CALL ผ่านการเรียกในเซคเม้นต์เดียวกัน

```
9A  2000  1E14 (YOUR SEGMENT VALUE MAY DIFFER)
```

การเก็บคำสั่ง 2000H ในรีจีสเตอร์ IP ที่ 0020H และ 1E14H ในรีจีสเตอร์ CS ที่ 141E[0] เพราะโปรแกรมย่อยจะใช้ CODE SEGMENT ร่วมกัน กับโปรแกรมหลัก รีจีสเตอร์ CS จะเซ็ตที่แอดเดรสเหมือนกัน 141E

```
CS address              141E0
IP offset          +     0020
Effective address       14200
```

ส่วน CODE SEGMENT ในโปรแกรมย่อยจุดเริ่มต้นที่ 14200 คือแอดเดรสที่ถูกต้อง การ LINK MAP จะไม่ทำให้จุดนี้เป็นศูนย์ ให้ท่านดูแอดเดรสจาก LISTING ของโปรแกรมหลัก ซึ่งค่าออฟเซ็ตจุดสิ้นสุดเป็น 0015 แต่ CODE SEGMENT สำหรับโปรแกรมย่อย คือการกำหนด PARA มันจะเริ่มต้นการกำหนดขอบเขตของพารากราฟ




คำสั่ง PUBLIC จะเป็นการรวมเซคเม้นต์เป็นเซคเม้นต์เดียว ท่านสามารถใช้โปรแกรม COM ให้ดูจากโปรแกรม LINKER ในการจัดการข้อมูลที่กำหนดในโปรแกรมหลัก และอ้างถึงโปรแกรมย่อย

```
                        TITLE   CALLMUL2 (EXE)  Call subprogram
                                EXTRN  SUBMUL2:FAR
                        ; ---------------------------------------------
0000                    STACKSG SEGMENT PARA STACK 'Stack'
0000 0040[????]                 DW      64 DUP(?)
0080                    STACKSG ENDS
                        ; ---------------------------------------------
0000                    DATASG  SEGMENT PARA 'Data'
0000 0140               QTY     DW      0140H
0002 2500               PRICE   DW      2500H
0004                    DATASG  ENDS
                        ; ---------------------------------------------
0000                    CODESG  SEGMENT PARA PUBLIC 'Code'
0000                    BEGIN   PROC    FAR
                                ASSUME CS:CODESG,DS:DATASG,SS:STACKSG
0000 B8 ---- R                  MOV     AX,DATASG
0003 8E D8                      MOV     DS,AX
0005 A1 0002 R                  MOV     AX,PRICE        ;Set up price
0008 8B 1E 0000 R               MOV     BX,QTY          ; & quantity
000C 9A 0000 ---- E             CALL    SUBMUL2         ;Call subprogram
0011 B4 4C                      MOV     AH,4CH          ;Return
0013 CD 21                      INT     21H
0015                    BEGIN   ENDP
0015                    CODESG  ENDS
                                END     BEGIN

Segments and Groups:
                N a m e         Length Align  Combine Class
CODESG . . . . . . . . . . . .  0015  PARA   PUBLIC  'CODE'
DATASG . . . . . . . . . . . .  0004  PARA   NONE    'DATA'
STACKSG  . . . . . . . . . . .  0080  PARA   STACK   'STACK'
Symbols:
                N a m e         Type   Value  Attr
BEGIN  . . . . . . . . . . . .  F PROC 0000   CODESG  Length = 0015
PRICE  . . . . . . . . . . . .  L WORD 0002   DATASG
QTY  . . . . . . . . . . . . .  L WORD 0000   DATASG
SUBMUL2  . . . . . . . . . . .  L FAR    0000         External
```

---

```
                        TITLE   SUBMUL2 Called subprogram
                        ; --------------------------------
0000                    CODESG  SEGMENT PARA PUBLIC 'Code'
0000                    SUBMUL2 PROC    FAR
                                ASSUME CS:CODESG
                                PUBLIC SUBMUL2
0000 F7 E3                      MUL     BX          ;AX = price, BX = qty
0002 CB                         RET                 ;DX:AX = product
0003                    SUBMUL2 ENDP
0003                    CODESG  ENDS
                                END     SUBMUL2

Segments and Groups:
                N a m e         Length Align  Combine Class
CODESG . . . . . . . . . . . .  0003  PARA   PUBLIC  'CODE'
Symbols:
                N a m e         Type   Value  Attr
SUBMUL2  . . . . . . . . . . .  F PROC 0000   CODESG Global Length=0003
```

---

```
Link Map
Object Modules: CALLMUL2+SUBMUL2

Start  Stop   Length Name           Class
00000H 0007FH 00080H STACKSG        STACK
00080H 00083H 00004H DATASG         DATA
00090H 000B2H 00023H CODESG         CODE  <-- 1 code segment

Program entry point at 0009:0000
```

**รูปที่ 14-2 การกำหนดคำสั่ง PUBLIC ใน CODE SEGMENT**

## SIMPLIFIED SEGMENT DIRECTIVES

จากตัวอย่างต่อไปนี้เป็นการแสดงการใช้คำสั่งเทียม SEGMENT ในตัวอย่างที่แล้ว ในส่วนของ CODE SEGMENT ใช้คำสั่งเทียม PUBLIC อย่างไรก็ตามการใช้คำสั่งเทียม SEGMENT มีความสำคัญที่แตกต่างกัน ข้อที่ 1 SEGMENT ที่ใช้ในการ LINK MAP คือจะเรียงตามลำดับของ CODE , DATA , STACK ข้อที่ 2 โปรแกรมย่อยใน CODE SEGMENT (_TEXT) จะกำหนดในรูปของเวิร์ด การ Trace ของภาษาเครื่องแสดงในรหัส Object สำหรับการเรียก (CALL)

9A 1600 1514 (your segemnt value may differ)

เพราะว่าในโปรแกรมย่อยใช้ CODE SEGMENT ร่วมกันกับโปรแกรมหลัก รีจีสเตอร์ CS จะถูกเซ็ตที่แอดเดรสเริ่มแรกเดียวกันคือ 1415

| | | |
|---|---|---|
| CS address | | 14150 |
| IP offset | + | 0016 |
| Effective address | | 14166 |

ใน CODE SEGMENT ของโปรแกรมย่อยเริ่มต้นที่ 14166H ท่านสามารถดูแอดเดรสจาก Listing ของโปรแกรมหลัก ซึ่งค่าออฟเซ็ตสุดท้ายที่ 0015 แต่การแมพแสดงใน CODE SEGMENT หลักเริ่มต้นที่ 00000H ขอบเขตของเวิร์ดต่อไปที่ 00016H CODE SEGMENT ของโปรแกรมนี้จะมีความยาว 19H ไบท์ ขณะโปรแกรมก่อนหน้านี้มีความยาว 13H (เพราะโปรแกรมแกรมย่อยกำหนดเป็น Paragraph boundary)

### COMMON DATA IN SUBPROGRAMS

ความต้องการในการประมวลผลข้อมูลของโมดุลในภาษาแอสแซมบลีหนึ่ง นั่นคือจะต้องกำหนดโมดุลแอสแซมบลีอื่นๆ การปรับปรุงดูจากตัวอย่างโปรแกรมหลักที่กำหนดค่าข้อมูล QTY และ PRICE ส่วนโปรแกรมย่อยจะทำการใส่ค่าลงใน BX และ AX ตามตัวอย่าง CALLMUL3 ซึ่งมีการกลับรหัส จะเปลี่ยนแปลงดังต่อไปนี้

- โปรแกรมหลัก CALLMUL3 กำหนดค่า QTY และ PRICE ที่คำสั่ง PUBLIC ส่วนของ DATA SEGMENT จะถูกกำหนดในคำสั่ง PUBLIC สังเกตจากสัญลักษณ์ในตาราง Global attribute สำหรับ QTY และ PRICE
- โปรแกรมย่อย SUBMUL3 จะกำหนดค่า QTY และ PRICE ที่คำสั่ง EXTRN และคำสั่งทั้งสองอยู่ในรูปเวิร์ด แอสแซมเบลอจะกำหนดความยาวเป็น 2 ฟิลด์ แอสแซมเบลอจะจ่ายรหัสการทำงานสำหรับคำสั่ง MOV แต่ตัว LINKER จะมีตัวโอเปอร์รานด์ที่สมบูรณ์ (สังเกตจากตาราง PRICE และ QTY คืออยู่ส่วนนอก

```
                        TITLE   CALLML2A (EXE)  Call subprogram
                                .MODEL  SMALL
                                .STACK  64
                                EXTRN   SUBML2A:FAR
                        ; ---------------------------------------------
                                .DATA
0000 0140               QTY     DW      0140H
0002 2500               PRICE   DW      2500H
                        ; ---------------------------------------------
                                .CODE
0000                    BEGIN   PROC    FAR
0000 B8 ---- R                  MOV     AX,@data
0003 8E D8                      MOV     DS,AX
0005 A1 0002 R                  MOV     AX,PRICE        ;Set up price
0008 8B 1E 0000 R               MOV     BX,QTY          ;  and quantity
000C 9A 0000 ---- E             CALL    SUBML2A         ;Call subprogram
0011 B4 4C                      MOV     AH,4CH          ;Return
0013 CD 21                      INT     21H
0015                    BEGIN   ENDP
                                END     BEGIN

Segments and Groups:
                N a m e         Length Align   Combine Class
DGROUP . . . . . . . . . . . .  GROUP
  _DATA  . . . . . . . . . . .  0004  WORD    PUBLIC  'DATA'
  STACK  . . . . . . . . . . .  0040  PARA    STACK   'STACK'
_TEXT  . . . . . . . . . . . .  0015  WORD    PUBLIC  'CODE'
Symbols:
                N a m e         Type   Value   Attr
BEGIN  . . . . . . . . . . . .F PROC  0000    _TEXT   Length = 0015
PRICE  . . . . . . . . . . . .L WORD  0002    _DATA
QTY  . . . . . . . . . . . . .L WORD  0000    _DATA
SUBML2A  . . . . . . . . . . .L FAR   0000            External
```

---

```
                        TITLE   SUBML2A Called subprogram
                                .MODEL SMALL
                                .CODE
0000                    SUBML2A PROC    FAR
                                PUBLIC SUBML2A
0000 F7 E3                      MUL     BX          ;AX = price, BX = qty
0002 CB                         RET                 ;DX:AX = product
0003                    SUBML2A ENDP
                                END     SUBML2A

Segments and Groups:
                N a m e         Length Align   Combine Class
DGROUP . . . . . . . . . . . .  GROUP
  _DATA  . . . . . . . . . . .  0000  WORD    PUBLIC  'DATA'
_TEXT  . . . . . . . . . . . .  0003  WORD    PUBLIC  'CODE'
Symbols:
                N a m e         Type   Value   Attr
SUBML2A  . . . . . . . . . . .F PROC  0000    _TEXT Global Length=0003

    Link Map
    Object Modules: CALLML2A+SUBML2A

    Start  Stop   Length Name          Class
    00000H 00018H 00019H _TEXT         CODE
    0001AH 0001DH 00004H _DATA         DATA
    00020H 0005FH 00040H STACK         STACK

    Program entry point at 0000:0000
```

รูปที่ 14 : การใช้คำสั่งเทียม SEGMENT

แอสแซมเบลอจะจ่ายคำสั่ง MOV ในโปรแกรมย่อยต่อไปนี้

```
A1  0000  E        MOV   AX,PRICE
8B  1E 0000  E     MOV   BX,QTY
```

รหัสคำสั่ง A1 หมายความว่าเป็นการเคลื่อนย้ายข้อมูลชนิดเวิร์ดจากหน่วยความจำไปยัง AX ขณะที่ 8B หมายความว่าเป็นการเคลื่อนย้ายข้อมูลจากหน่วยความจำไป BX ดูจากการเอ็กซีคิวส์ด้วย TRACE จะได้ Object code ดังต่อไปนี้

```
A1   0200
8b   1E   0000
```

รหัส Object จะเป็นตัวกำหนดในการจ่ายขณะที่คำสั่ง MOV คือการเรียกโปรแกรม ผลลัพธ์การทำงาน โอเปอร์รานด์ของโปรแกรมทั้ง 3 จะอ้างถึงรีจีสเตอร์ DS และค่าออฟเซ็ตเดียวกัน

โปรแกรมหลักและโปรแกรมย่อยจะกำหนดรายการข้อมูลอื่นๆทุกรายการ เพียงแต่กำหนดด้วยคำสั่ง PUBLIC และ EXTRN

**DEFINING DATA IN BOTH PROGRAMS**

จากตัวอย่าง CALLMUL3 กำหนดค่า QTY และ PRICE ขณะที่ SUBMUL3 ไม่ได้กำหนดข้อมูล เหตุผลเพราะว่า SUBMUL3 สามารถอ้างถึงข้อมูลของ CALLMUL3 เพราะไม่เปลี่ยนแปลงแอดเดรสในรีจีสเตอร์ DS ซึ่งยังคงชี้ Data segment ของ CALLMUL3

รูปแบบของการเตรียมโปรแกรม ในตัวอย่างต่อไป จะกำหนด QTY ใน CALLMUL4 แต่กำหนดค่า PRICE ใน SUBMUL4 จากการเรียกของ CALLMUL4 แสดงการกำหนด ค่าข้อมูล PRICE ก็สามารถใช้ได้ ถึงแม้ว่า SUBMUL4 รู้ตำแหน่งของรายการข้อมูลทั้งสอง โปรแกรม SUBMUL4 ที่อยู่ใน Code segment มี QTY ขณะที่รีจีสเตอร์ DS ยังมีข้อมูลแอดเดรสของ CALLMUL4 ใน Data segment ส่วน SUBMUL4 จะทำการ PUSH ค่า DS ลงในสแตค และทำการโหลดแอดเดรสของข้อมูลใน Data segment โปรแกรม SUBMUL4 ก็สามารถนำข้อมูล PRICE มาใช้ใหม่

ก่อนการย้อนกลับของการเรียก CALLMUL4 ส่วนของโปรแกรมย่อย SUBMUL4 จะทำการ POP

```
                        TITLE   CALLMUL3 (EXE)  Call subprogram
                                EXTRN   SUBMUL3:FAR
                                PUBLIC  QTY,PRICE
                        ; ----------------------------------------
0000                    STACKSG SEGMENT PARA STACK 'Stack'
0000 0040[????]                 DW      64 DUP(?)
0080                    STACKSG ENDS
                        ; ----------------------------------------
0000                    DATASG  SEGMENT PARA PUBLIC 'Data'
0000 0140               QTY     DW      0140H
0002 2500               PRICE   DW      2500H
0004                    DATASG  ENDS
                        ; ----------------------------------------
0000                    CODESG  SEGMENT PARA PUBLIC 'Code'
0000                    BEGIN   PROC    FAR
                                ASSUME CS:CODESG,DS:DATASG,SS:STACKSG
0000 B8 ---- R                  MOV     AX,DATASG
0003 8E D8                      MOV     DS,AX
0005 9A 0000 ---- E             CALL    SUBMUL3         ;Call subprogram
000A B4 4C                      MOV     AH,4CH          ;Return
000C CD 21                      INT     21H
000E                    BEGIN   ENDP
000E                    CODESG  ENDS
                                END     BEGIN

Segments and Groups:
                N a m e         Length Align   Combine Class
CODESG . . . . . . . . . . . .  000E   PARA    PUBLIC  'CODE'
DATASG . . . . . . . . . . . .  0004   PARA    PUBLIC  'DATA'
STACKSG  . . . . . . . . . . .  0080   PARA    STACK   'STACK'
Symbols:
                N a m e         Type   Value   Attr
BEGIN  . . . . . . . . . . . .  F PROC 0000    CODESG  Length = 000E
PRICE  . . . . . . . . . . . .  L WORD 0002    DATASG  Global
QTY  . . . . . . . . . . . . .  L WORD 0000    DATASG  Global
SUBMUL3  . . . . . . . . . . .  L FAR  0000            External
```

---

```
                        TITLE   SUBMUL3 Called subprogram
                                EXTRN   QTY:WORD,PRICE:WORD
                        ; ----------------------------------
0000                    CODESG  SEGMENT PARA PUBLIC 'CODE'
0000                    SUBMUL3 PROC    FAR
                                ASSUME CS:CODESG
                                PUBLIC SUBMUL3
0000 A1 0000 E                  MOV     AX,PRICE
0003 8B 1E 0000 E               MOV     BX,QTY
0007 F7 E3                      MUL     BX              ;DX:AX = product
0009 CB                         RET
000A                    SUBMUL3 ENDP
000A                    CODESG  ENDS
                                END     SUBMUL3

Segments and Groups:
                N a m e         Length Align   Combine Class
CODESG . . . . . . . . . . . .  000A   PARA    PUBLIC  'CODE'
Symbols:
                N a m e         Type   Value   Attr
PRICE  . . . . . . . . . . . .  V WORD 0000    External
QTY  . . . . . . . . . . . . .  V WORD 0000    External
SUBMUL3  . . . . . . . . . . .  F PROC 0000    CODESG Global Length=000A
```

---

```
Link Map
Object Modules: CALLMUL3+SUBMUL3

Start  Stop   Length Name           Class
00000H 0007FH 00080H STACKSG        STACK
00080H 00083H 00004H DATASG         DATA
00090H 000A9H 0001AH CODESG         CODE

Program entry point at 0009:0000
```

รูปที่ 14 4 การกำหนดข้อมูลในโปรแกรมย่อย

## PASSING PARAMETERS

วิธีการอื่นๆที่จะทำให้โปรแกรมรู้ข้อมูลของการเรียกโปรแกรมย่อย โดยการส่งผ่านพารามิเตอร์ ซึ่งโปรแกรมจะส่งผ่านข้อมูลกับสแตค ในกรณีนี้เพื่อความแน่ใจว่าคำสั่ง PUSH ที่อ้างอิงข้อมูลเวิร์ด ทั้งหน่วยความจำและรีจีสเตอร์ จากตัวอย่างในหน้า 14-42 เป็นการเรียกโปรแกรมย่อย CALLMUL5 จะทำการ PUSH ค่า PRICE และ QTY ก่อนการเรียกโปรแกรมย่อย SUBMUL5 การเซ็ตค่าเริ่มแรกของ SP จะมีขนาดของสแตค 80H ค่าแต่ละเวิร์ดจะ PUSH ค่าลงในสแตค และลดค่า SP ลง 2

หลังจากการ CALL สแตคจะมีการทำงานดังต่อไปนี้

| | | | | |
|---|---|---|---|---|
| ..... | 1200 | 1E14 | 4001 | 0025 |
| | 78 | 7A | 7C | 7E |

1. คำสั่ง PUSH จะโหลดค่า PRICE (2500) ลงในสแตคที่ตำแหน่ง 7E
2. คำสั่ง PUSH จะโหลดค่า QTY (0140) ลงในสแตคที่ตำแหน่ง 7C
3. คำสั่ง CALL จะทำการ PUSH ข้อมูลของ CS (141E สำหรับการเอ็กซีคิวส์) ลงในสแตคที่ตำแหน่ง 7A ในโปรแกรมย่อยมีคำสั่ง PUBLIC เพื่อให้ LINKER รวม Codesegment ทั้งสอง และใช้แอดเดรสของ CS ร่วมกัน
4. คำสั่ง CALL จะทำการ PUSH ข้อมูลของ IP คือ 0012 ลงในสแตคที่ตำแหน่ง 78

การเรียกโปรแกรมย่อยต้องการใช้รีจีสเตอร์ BP ในการเข้าถึงพารามิเตอร์ของสแตค การทำงานในช่วงแรกจะเก็บข้อมูลของ BP สำหรับการเรียกโปรแกรมย่อย โดยการ PUSH ค่าลงในสแตค ในกรณี นี้ BP จะมีข้อมูลเป็น 0 และ PUSH เก็บไว้ในสแตคที่ตำแหน่ง 76

| | | | | |
|---|---|---|---|---|
| 0000 | 1200 | 1E14 | 4001 | 0025 |
| 76 | 78 | 7A | 7C | 7E |

โปรแกรมจะเพิ่มข้อมูลของรีจีสเตอร์ SP (0076) ในรีจีสเตอร์ BP เพราะใช้ BP เป็นอินเดครีจีสเตอร์ ทำให้รีจีสเตอร์ BP มีข้อมูล 0076 ค่าของ PRICE ที่อยู่ในสแตคคือ BP + 8 (Offset 7C) การเคลื่อนย้ายข้อมูลของค่าเหล่านี้จากสแตคไปยัง AX และ BX ใช้รูปแบบของการคูณ

```
                        TITLE   CALLMUL4 (EXE)  Call subprogram
                                EXTRN   SUBMUL4:FAR
                                PUBLIC  QTY
                        ; ---------------------------------------
0000                    STACKSG SEGMENT PARA STACK 'Stack'
0000  0040[????]                DW      64 DUP(?)
0080                    STACKSG ENDS
                        ; ---------------------------------------
0000                    DATASG  SEGMENT PARA 'Data'
0000 0140               QTY     DW      0140H
0002                    DATASG  ENDS
                        ; ---------------------------------------
0000                    CODESG  SEGMENT PARA 'Code'
0000                    BEGIN   PROC    FAR
                                ASSUME CS:CODESG,DS:DATASG,SS:STACKSG
0000 B8 ---- R                  MOV     AX,DATASG
0003 8E D8                      MOV     DS,AX
0005 9A 0000 ---- E             CALL    SUBMUL4         ;Call subprogram
000A B4 4C                      MOV     AH,4CH          ;Exit
000C CD 21                      INT     21H
000E                    BEGIN   ENDP
000E                    CODESG  ENDS
                                END     BEGIN

Segments and Groups:
                N a m e         Length Align  Combine Class
CODESG . . . . . . . . . . . .  000E  PARA   NONE    'CODE'
DATASG . . . . . . . . . . . .  0002  PARA   NONE    'DATA'
STACKSG  . . . . . . . . . . .  0080  PARA   STACK   'STACK'
Symbols:
                N a m e         Type   Value   Attr
BEGIN  . . . . . . . . . . . .F PROC 0000   CODESG  Length = 000E
QTY  . . . . . . . . . . . . .L WORD 0000   DATASG  Global
SUBMUL4  . . . . . . . . . . .L FAR  0000           External
```

---

```
                        TITLE   SUBMUL4 Called subprogram
                                EXTRN   QTY:WORD
                        ; ---------------------------------------
0000                    DATASG  SEGMENT PARA 'Data'
0000 2500               PRICE   DW      2500H
0002                    DATASG  ENDS
                        ; ---------------------------------------
0000                    CODESG  SEGMENT PARA 'CODE'
0000                    SUBMUL4 PROC    FAR
                                ASSUME CS:CODESG
                                PUBLIC SUBMUL4
0000 8B 1E 0000 E               MOV     BX,QTY          ;Get QTY from CALLMUL
0004 1E                         PUSH    DS              ;Save CALLMUL's DS
                                ASSUME DS:DATASG
0005 B8 ---- R                  MOV     AX,DATASG       ;Set up own DS
0008 8E D8                      MOV     DS,AX
000A A1 0000 R                  MOV     AX,PRICE        ;Price from own data segment
000D F7 E3                      MUL     BX              ;DX:AX = product
000F 1F                         POP     DS              ;Restore CALLMUL's DS
0010 CB                         RET
0011                    SUBMUL4 ENDP
0011                    CODESG  ENDS
                                END     SUBMUL4

Segments and Groups:
                N a m e         Length Align  Combine Class
CODESG . . . . . . . . . . . .  0011  PARA   NONE    'CODE'
DATASG . . . . . . . . . . . .  0002  PARA   NONE    'DATA'
Symbols:
                N a m e         Type   Value   Attr
PRICE  . . . . . . . . . . . .L WORD 0000   DATASG
QTY  . . . . . . . . . . . . .V WORD 0000       External
SUBMUL4  . . . . . . . . . . .P PROC 0000   CODESG Global Length=0011
```

---

รูปที่ 14.5 การกำหนดข้อมูลโปรแกรมหลักและโปรแกรมย่อย

```
                        TITLE   CALLMUL5 (EXE)  Passing parameters
                                EXTRN   SUBMUL5:FAR
                        ; ---------------------------------------------
0000                    STACKSG SEGMENT PARA STACK 'Stack'
0000 0040[????]                 DW      64 DUP(?)
0080                    STACKSG ENDS
                        ; ---------------------------------------------
0000                    DATASG  SEGMENT PARA 'Data'
0000 0140               QTY     DW      0140H
0002 2500               PRICE   DW      2500H
0004                    DATASG  ENDS
                        ; ---------------------------------------------
0000                    CODESG  SEGMENT PARA PUBLIC 'Code'
0000                    BEGIN   PROC    FAR
                                ASSUME CS:CODESG,DS:DATASG,SS:STACKSG
0000 B8 ---- R                  MOV     AX,DATASG
0003 8E D8                      MOV     DS,AX
0005 FF 36 0002 R               PUSH    PRICE
0009 FF 36 0000 R               PUSH    QTY
000D 9A 0000 ---- E             CALL    SUBMUL5         ;Call subprogram
0012 B4 4C                      MOV     AH,4CH          ;Return
0014 CD 21                      INT     21H
0016                    BEGIN   ENDP
0016                    CODESG  ENDS
                                END     BEGIN

Segments and Groups:
                N a m e         Length Align  Combine Class
CODESG . . . . . . . . . . .    0016   PARA   PUBLIC  'CODE'
DATASG . . . . . . . . . . .    0004   PARA   NONE    'DATA'
STACKSG  . . . . . . . . . .    0080   PARA   STACK   'STACK'
Symbols:
                N a m e         Type  Value   Attr
BEGIN  . . . . . . . . . .      F PROC 0000    CODESG  Length = 0016
PRICE  . . . . . . . . . .      L WORD 0002    DATASG
QTY  . . . . . . . . . . .      L WORD 0000    DATASG
SUBMUL5  . . . . . . . . .      L FAR  0000            External
```

---

```
                        TITLE   SUBMUL5 Called subprogram
0000                    CODESG  SEGMENT PARA PUBLIC 'Code'
0000                    SUBMUL5 PROC    FAR
                                ASSUME CS:CODESG
                                PUBLIC SUBMUL5
0000 55                         PUSH    BP
0001 8B EC                      MOV     BP,SP
0003 8B 46 08                   MOV     AX,[BP+8]       ;Get price
0006 8B 5E 06                   MOV     BX,[BP+6]       ;Get quantity
0009 F7 E3                      MUL     BX              ;DX:AX = product
000B 5D                         POP     BP
000C CA 0004                    RET     4
000F                    SUBMUL5 ENDP
000F                    CODESG  ENDS
                                END

Segments and Groups:
                N a m e         Length Align  Combine Class
CODESG . . . . . . . . . . .    000F   PARA   PUBLIC  'CODE'
Symbols:
                N a m e         Type   Value   Attr
SUBMUL5  . . . . . . . . .      F PROC 0000    CODESG Global Length=000F
```

---

```
Link Map
Object Modules: CALLMUL5+SUBMUL5

Start  Stop   Length Name          Class
00000H 0007FH 00080H STACKSG      STACK
00080H 00083H 00004H DATASG       DATA
00090H 000BEH 0002FH CODESG       CODE

Program entry point at 0009:0000
```

รูปที่ 14.6 การส่งผ่านพารามิเตอร์

ก่อนการย้อนกลับจากโปรแกรมย่อย จะทำการ POP ค่าของ BP (นำค่า 0 ไปเก็บไว้ที่ BP) และค่า SP เพิ่มขึ้น 2 จาก 76 เป็น 78 คำสั่งสุดท้าน RET จะเป็นการย้อนกลับชนิด FAR จากโปรแกรมย่อย มีการทำงานดังต่อไปนี้

- ทำการ POP ค่าเวิร์ดที่ยอดของสแตค (1200) ไปยังรีจีสเตอร์ IP และเพิ่มค่า SP อีก 2 จาก 78 เป็น 7A
- ทำการ POP ค่าเวิร์ดที่ยอดของสแตค (141E) ไว้ใน CS และเพิ่มค่า SP อีก 2 จาก 7A เป็น 7C
- เนื่องจากการส่งผ่านพารามิเตอร์ทั้ง 2 ที่ตำแหน่ง 7C และ 7E เมื่อคำสั่ง RET ถูกถอดรหัส

**RET 4**

ค่า 4 หมายถึงเราทำการ POP ค่าจำนวน 4 ไบท์ ในการส่งผ่านพารามิเตอร์ จำนวน 2 เวิร์ด การทำงานของคำสั่ง RET จะทำการบวกในการ POP ค่าของ SP

## การเชื่อมภาษาปาสคาลกับภาษาแอสแซมบลี

ส่วนนี้จะเป็นการอธิบายเกี่ยวกับการเชื่อม IBM ของ Microsoft Pascal กับ ภาษาแอสแซมบลี ตัวอย่างโปรแกรมภาษาปาสคาล ในหน้า 14-44 ที่ใช้เชื่อมต่อกับภาษาแอสแซมบลี ซึ่งภาษาแอสแซมบลีทำหน้าที่เป็นโปรแกรมย่อย โปรแกรมภาษาปาสคาลจะถูกแปลอยู่ในรูปของ Object code และภาษาแอสแซมบลีอยู่ในรูปของ Object code ตัว LINKER จะทำหน้าที่รวม Object ของภาษาทั้งสอง ให้อยู่ในรูปของโปรแกรมเอ็กซีคิวส์ (EXE)

โปรแกรมภาษาปาสคาลจะกำหนด 2 ชื่อรายการคือ Temp_row และ Temp_col เพื่อรับข้อมูลในตำแหน่ง ROW และ COLUMN จากคีย์บอร์ด เพื่อป้อนให้กับตัวแปล โปรแกรมก็จะส่งแอดเดรสของ Temp_row และ Temp_col ผ่านพารามิเตอร์ไปยังโปรแกรมย่อยของภาษาแอสแซมบลี ที่ทำหน้าที่กำหนดตำแหน่งเคอร์เซอร์ โปรแกรมภาษาปาสคาลจะกำหนดชื่อโปรแกรมย่อยของภาษาแอสแซมบลี ในโพรซีเยอร์ที่เรียกว่า Move_cursor และกำหนดพารามิเตอร์ 2 ตัวในคำสั่ง EXTRN คำสั่งในภาษาปาสคาล "CALLS" เป็นชื่อของโปรแกรมย่อยภาษาแอสแซมบลี และทำหน้าที่ส่งผ่านพารามิเตอร์คือ

```
MOVE_CURS (temp_row , temp_col);
```

ค่าที่ PUSH ลงในสแตคคือ ค่าของการเรียกโปรแกรมของสแตคพอยเตอร์ (SP) ค่าของการย้อนกลับมาเซคเม้นต์เดิม ค่าออฟเซ็ตของการย้อนกลับ และแอดเดรสของการส่งผ่านพารามิเตอร์ 2 ตัว ดังแสดงค่าออฟเซ็ตของจุดเข้าในสแตค

โปรแกรมย่อยภาษาแอสแซมบลีจะไปใช้รีจีสเตอร์ BP ท่านจะต้อง PUSH ค่าลงในสแตคและเก็บแอดเดรสสำหรับการย้อนกลับมายังภาษาปาสคาล สังเกตขั้นตอนของการเรียกโปรแกรมย่อยจะเหมือนกับตัวอย่างที่แล้วหน้า 14-42

```
program pascall ( input, output );

    procedure move_curs( const row: integer;
                         const col: integer ); extern;

    var
        temp_row:       integer;
        temp_col:       integer;


    begin
        write( 'Enter cursor row: ' );
        readln( temp_row );

        write( 'Enter cursor column: ' );
        readln( temp_col );

        move_curs( temp_row, temp_col );
        write( 'New cursor location' );
    end.
```

```
TITLE   MOVCUR  Assembler subprogram called by Pascal
        PUBLIC MOVE_CURS
;-------------------------------------------------------
;    MOVE_CURS: Set cursor on screen at passed location
;    Passed:    const row       Row and column where
;               const col       cursor is to be set
;    Returned:  nothing
;-------------------------------------------------------
CODESEG     SEGMENT PARA PUBLIC 'CODE'

MOVE_CURS   PROC FAR
            ASSUME CS:CODESEG
            PUSH   BP              ;Caller's BP register
            MOV    BP,SP           ;Point to parameters passed

            MOV    SI,[BP+8]       ;SI points to row
            MOV    DH,[SI]         ;Move row to DH

            MOV    SI,[BP+6]       ;SI points to column
            MOV    DL,[SI]         ;Move column to DL

            MOV    AH,02           ;Move cursor
            MOV    BH,0            ;Page #0
            INT    10H

            POP    BP              ;Return to caller
            RET    4
MOVE_CURS   ENDP
CODESEG ENDS
            END
```

00 Caller's stack pointer
02 Caller's return segment pointer
04 Caller's return offset
06 Address of second parameter
08 Address of first parameter

รูปที่ 14.7 การเชื่อมต่อภาษาซีกับภาษาแอสเซมบลี

รีจีสเตอร์ SP จะเป็นแอดเดรสของสแตค แต่ไม่สามารถใช้ SP ในการทำหน้าที่เป็นอินเดค รีจีสเตอร์ ขั้นตอนหลังจากการ PUSH BP จะเคลื่อนยย้ายแอดเดรสจาก SP ไปยัง BP ขั้นตอนนี้สามารถใช้รีจีสเตอร์ BP เป็นอินเดครีจีสเตอร์เพื่อเข้าถึงข้อมูลของสแตค

ขั้นตอนต่อไป การเข้าถึงแอดเดรสของพารามิเตอร์ 2 ตัวในสแตค พารามิเตอร์ตัวแรกคือ ROW มีค่าออฟเซ็ตเป็น 08 ในสแตค และสามารถเข้าถึงข้อมูลในตำแหน่ง BP + 08 พารามิเตอร์ที่ 2 คือ COLUMN มีค่าออฟเซ็ตเป็น 06 สามารถเข้าถึงข้อมูลในตำแหน่ง BP + 06

แอดเดรส 2 แอดเดรสในสแตคจะมีการเคลื่อนย้ายข้อมูลไปยังอินเดครีจีสเตอร์ BX,DI หรือ SI ดังตัวอย่าง [BP+08] จะเคลื่อนย้ายแอดเดรสของ ROW ไปยังรีจีสเตอร์ SI และใช้ [SI] ทำการเคลื่อนย้ายข้อมูลไปยังพารามิเตอร์ ของรีจีสเตอร์ DH

ค่าของ COLUMN จะเคลื่อนย้ายข้อมูลไปยังรีจีสเตอร์ DL เมื่อโปรแกรมย่อยใช้ค่า ROW และ COLUMN ที่อยู่ในรีจีสเตอร์ DX เพื่อเรียก BIOS ในการกำหนดตำแหน่งเคอร์เซอร์ การออกจากโปรแกรมย่อย โปรแกรมย่อยจะทำการ POP ค่า BP คำสั่ง RET ต้องการค่าโอเปอร์รานด์ นั่นคือจำนวนพารามิเตอร์ 2 จำนวน ในกรณีเป็น 2x2 = 4 จะทำการ POP ค่าโดยอัตโนมัติจากสแตค และย้อนกลับไปยังโปรแกรมที่เรียกมา

ถ้าท่านเปลี่ยนเซคเม้นต์รีจีสเตอร์ ต้องแน่ใจว่าการ PUSH ที่จุดเข้า หรือการ POP ที่จุดออก ท่านจะใช้สแตคส่งผ่านค่าจากโปรแกรมย่อย ไปยังโปรแกรมที่เนียกมา ถึงแม้ว่าโปรแกรมย่อยจะไม่มีค่าย้อนกลับ ภาษาปาสคาลจะรับค่าย้อนกลับถ้าเป็นเวิร์คอยู่ใน AX ถ้าเป็น 2 เวิร์คอยู่ใน DX:AX

จากตัวอย่างทั้ง 2 โปรแกรมคือ จากการแมพที่ตัว LINKER เป็นตัวจ่าย จุดเข้าแรกของภาษาปาสคาลชื่อ PASCALL จุดเข้าที่ 2 คือ สำหรับโปรแกรมภาษาแอสแซมบลีชื่อ CODESEG (ซึ่งเป็นชื่อของ CODE SEGMENT)

## การเชื่อมภาษาซีกับภาษาแอสแซมบลี

ปัญหาของการเชื่อมภาษาซีกลับภาษาแอสแซมบลี คือ Version ของภาษาซีที่แตกต่างกัน ต้องดูรายละเอียดจากคู่มือภาษาซี จุดที่น่าสนใจมีดังนี้

- Version ของภาษาซีส่วนมาก จะส่งผ่านพารามิเตอร์ลงในสแตคในลักษณะตรงกันข้ามกับภาษาอื่นๆ พิจารณาตามกฎเกณฑ์ของภาษาซีดังต่อไปนี้

```
Adds (m , n);
```

คำสั่งที่แสดงจะเป็นการ PUSH n และ m ลงในสแตค การย้อนกลับจากการเรียกโปรแกรมย่อย ในโมดุลของภาษาซี (ไม่ใช่โมดุลภาษาแอสแซมบลี) การ Adds 4 ให้กับ SP ให้กับพารามิเตอร์ชนิดของโพรซีเยอร์ในการเข้าถึงสำหรับพารามิเตอร์ทั้งสองมีดังต่อไปนี้

```
PUSH    BP
MOV     BP,SP
MOV     DH,[BP+4]
MOV     DL,[BP+6]
...
POP     BP
RET
```

- สำหรับภาษา C รุ่นที่ใช้ตัวใหญ่และตัวเล็ก เป็นชื่อของโมดุลภาษาแอสแซมบลี จะต้องเป็นตัวใหญ่หรือตัวเล็กเหมือนกัน
- สำหรับภาษา C บางรุ่นที่ต้องการให้โปรแกรมภาษาแอสแซมบลีเปลี่ยนรีจีสเตอร์ SI และ DI ทำการ PUSH ที่จุดเข้า และทำการ POP ที่จุดออก
- โปรแกรมภาษาแอสแซมบลีจะมีค่าย้อนกลับ ถ้าต้องการค่า 1 เวิร์ดใช้ AX ถ้าเป็น 2 เวิร์ดใช้ DX:AX
- ภาษา C บางรุ่น โปรแกรมภาษาแอสแซมบลีจะต้องเซ็ตค่า DF และต้องเคลียแฟลก CLD ก่อนการย้อนกลับ

## บทสรุป

- ชื่อของโปรแกรมหลักที่เรียกโปรแกรมย่อยจะต้องมีจุดเข้าที่คำสั่ง EXTRN ในโปรแกรมย่อยกำหนดจุดเข้าโดยคำสั่ง PUBLIC
- ถ้ามีรหัส Code segment 2 ชุด จะต้องมีการเชื่อมเป็นเซคเม้นต์เดียว โดยมีการกำหนดชื่อเหมือนกัน Class เดียวกัน และ Combine เป็นชนิด Public
- การเอ็กซีคิวส์โปรแกรมจะต้องเริ่มที่โปรแกรมหลัก
- แมคโคร คือ ชื่อของกลุ่ม Text จะประกอบด้วยคำสั่งเทียม หรือการอ้างอิงแมคโครอื่นๆ
- ความสำคัญในการใช้แมคโคร คือ สร้างคำสั่งใหม่
- โปรแกรมแมคโครต่างๆที่เขียนเก็บไว้ในแฟ้ม LIB สามารถนำมาใช้ได้โดยคำสั่งเทียม Include
- แมคโคร REPT ใช้ในการทำงานซ้ำของกลุ่มคำสั่ง ซึ่ง Argument เดียวทำหน้าที่กำหนดจำนวนครั้งของการทำงานซ้ำๆ
- แมคโคร IRP ใช้ในการทำงานซ้ำๆ

# แบบฝึกหัด

1. จงเขียนแมคโครต่อไปนี้ รีจีสเตอร์ทั้งหมดใช้โดยแมคโครในการเก็บข้อมูล ของรีจีสเตอร์และโหลดข้อมูลเข้ามาในรีจีสเตอร์
2. จงเขียนแมคโครในการพิมพ์ชุดสตริงโดยผ่านตัวแปร
3. จงเขียนแมคโครในการวาดรูปสี่เหลี่ยมด้านขนานพร้อมระบายสีในพื้นที่สี่เหลี่ยม
4. จงอธิบายการใช้แมคโครในตัวอย่าง

```
MAC1    MACRO    M
        IF    M-1
        MOV   AX,M
        M=M-1
        IFE    M
        MOV   BX,M
        ENDIF
        ENDIF
        ENDM
```

   a. ถ้าค่าของ M เป็นค่า 1
   b. ถ้าค่าของ M เป็นค่า 2

5. จงเขียนแมคโครในการทำงานของ Fibonacci series 1,1,2,3,5,8,13,21,34... โดยมีอัลกอริทึมต่อไปนี้

```
IF    N = 1
      THEN   FN = 1
      ELSE
        LO = 0
        HI = 1
      REPEAT   N-1 TIMES
        X = LO
        LO = HI
        HI = X + LO
      FN = HI
```

6. จากตัวอย่างของแมคโครต่อไปนี้

```
REPEAT    MACRO     CHAR,COUNT
          LOCAL     L1
          MOV   CX,COUNT
L1:       MOV   AH,2
          MOV   DL,CHAR
          INT   21H
          LOOP  L1
          ENDM
```

จงเขียนรหัสคำสั่งที่จ่ายโดย MASM เมื่อแมคโครถูกเรียกใช้ดังนี้

```
REPEAT   'X',50
```

7. จากตัวอย่างของแมคโครต่อไปนี้

```
GOTOXY    MACRO     XVAL,YVAL
     IF XVAL LT 0              ;XVAL < 0?
          EXITM                ;IF SO ,EXIT
     ENDIF
IF   YVAL  LT  0               ;YVAL < 0
     EXITM
ENDIF

          MOV   BX,0           ;CHOOSE VIDEO PAGE 0
          MOV   AH,2           ;LOCATE CURSOR
          MOV   DH,YVAL
          MOV   DL,XVAL
          INT   10H
          ENDM
```

จงเขียนรหัสคำสั่งที่จ่ายโดย MASM ถ้ามีการเรียกแมคโครดังนี้

```
GOTOXY    -2,20
```

8. จงเขียนคำสั่งที่มีการทำงานเหมือนกับแมคโคร REPEAT

```
IRP  VAL,<100,20,30>
     DB 0,0,0,VAL
```