本章目標

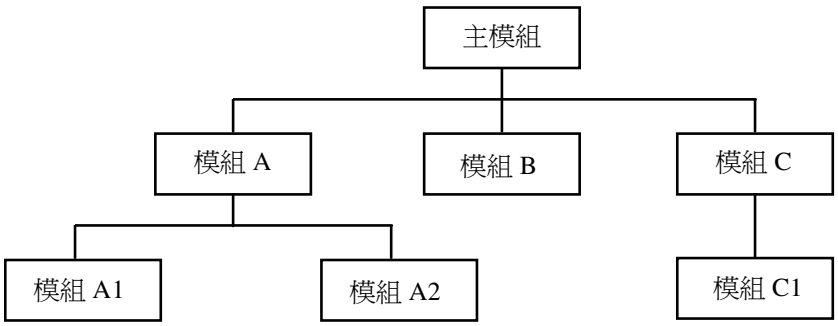
- 了解模組化與結構化程式設計技巧
- 了解80x86的程式連結與程式模組宣告方式
- 了解副程式、巢路副程式、與遞回副程式
- 了解副程式的參數傳遞方式
- 了解巨集指令的定義與使用
- 了解巨集指令相關的假指令

模組化程式設計

組合語言的模組化程式設計通常由下列幾個層次輔助完成：

1. 副程式(subroutine)
2. 組譯程式假指令
3. 巨集指令(macro)
4. 中斷結構(interrupt structure)

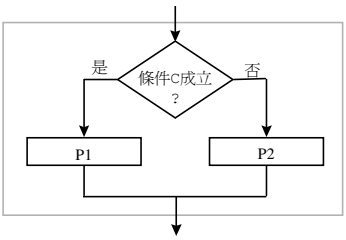
典型的模組層次圖

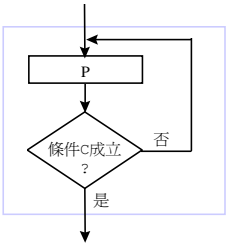


結構化程式設計

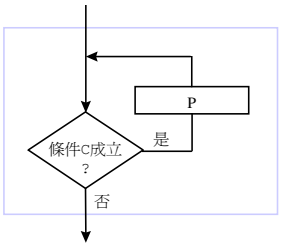
基本的模組結構只有下列三種：

1. 循序結構(sequential structure)
2. 選擇性結構(selection structure)
3. 重覆結構(iteration structure)





(a) REPEAT UNTIL 結構



(b) WHILE DO 結構

結構化程式假指令

masm6.xx組譯程式的結構化程式設計的假指令：

- .IF… .ENDIF
- .IF … .ELSE … .ENDIF
- .IF … .ELSEIF … .ELSE … .ENDIF
- .REPEAT … .UNTIL
- .WHILE … .ENDW

masm結構化程式假指令中的關係與邏輯運算子

運算子	功能	運算子	功能
==	相等	<=	小於或相等
!=	不相等	&	位元測試
>	大於	!	NOT
>=	大於或相等	&&	AND
<	小於	ll	OR

.IF... .ENDIF結構的使用例

```
                        ;ex6.1-3.asm
0000                      START    PROC  FAR
0000  B9 0010                      MOV   CX,10H  ;read 16 characters
0003  B4 08               READ_KBY: MOV   AH,08H  ;read a character
0005  CD 21                        INT   21H    ;from keyboard
                          .IF   AL >= 'a' && AL <= 'z'
0007  3C 61             *       cmp    al, 'a'
0009  72 06             *       jb     @C0001
000B  3C 7A              *       cmp    al, 'z'
000D  77 02             *       ja     @C0001
000F  2C 20                        SUB   AL,20H
                          .ENDIF
0011                    *@C0001:
0011  8A D0                UP_CASE:  MOV   DL,AL   ;display character
0013  B4 02                        MOV   AH,02H
0015  CD 21                        INT   21H
0017  E2 EA                        LOOP  READ_KBY
0019  B4 4C                        MOV   AH,4CH  ;return to MS-DOS
001B  CD 21                        INT   21H
001D                      START    ENDP
```


.IF … .ELSEIF … .ELSE … .ENDIF結構的使用例

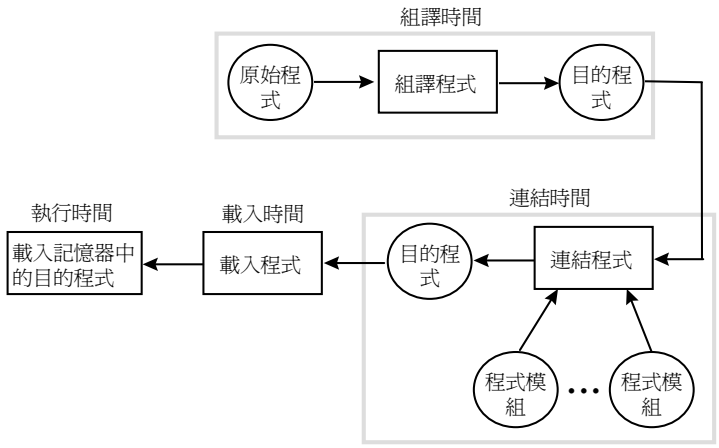
```
                        ;ex6.1-4.asm
                              ....
0005  B4 08               READ_KBY: MOV   AH,08H  ;read a character
0007  CD 21                        INT   21H    ;from keyboard
                          .IF   AL >= 'a' && AL <= 'f'
0009  3C 61             *       cmp    al, 'a'
000B  72 08             *       jb     @C0001
                              ....
0011  2C 57                        SUB   AL,57H
                          .ELSEIF   AL >= 'A' && AL <= 'F'
0013  EB 0E              *       jmp    @C0004
0015  3C 41             *@C0001: cmp    al, 'A'
                              ....
001B  77 04             *       ja     @C0005
001D  2C 37                        SUB   AL,37H
                          .ELSE
001F  EB 02              *       jmp    @C0008
0021  2C 30             *@C0005:       SUB   AL,30H
                          .ENDIF
0023                    *@C0008:
0023                    *@C0004:
0023  A2 0000 R                    MOV   TEMP,AL ;store it
```

.WHILE … .ENDW結構的使用例

```
                        ;ex6.1-5.asm

0000                    START    PROC  FAR
0000  32 C0                     XOR   AL,AL ;guarantee AL != 0DH
                              .WHILE  AL != 0DH
0002  EB 14             *       jmp    @C0001
0004  B4 08    *@C0002:  READ_KBY: MOV   AH,08H  ;read a character
0006  CD 21                     INT   21H    ;from keyboard
                              .IF   AL >= 'a' && AL <= 'z'
0008  3C 61             *       cmp    al, 'a'
                                ....
0010  2C 20                     SUB   AL,20H
                              .ENDIF
0012  0012             *@C0003:
0012  8A D0                UP_CASE:  MOV   DL,AL   ;display character
0014  B4 02                     MOV   AH,02H
0016  CD 21                     INT   21H
                              .ENDW
0018  3C 0D             *@C0001: cmp    al, 00Dh
001A  75 E8             *       jne    @C0002
001C  B4 4C                     MOV   AH,4CH  ;return to MS-DOS
001E  CD 21                     INT   21H
```


使用者程式執行的處理過程



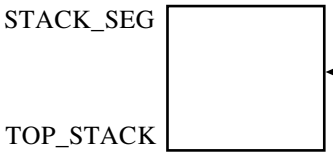
堆疊節區宣告例

模組A

```
STACK_SEG  SEGMENT  STACK
        DW    40 DUP(0)
TOP_STACK  LABEL WORD
STACK_SEG  ENDS
```

模組B

```
STACK_SEG  SEGMENT  STACK
        DW    20 DUP(0)
TOP_STACK  LABEL WORD
STACK_SEG  ENDS
```



模組 A與模組 B
合併後之STACK
節區 (60個語句)

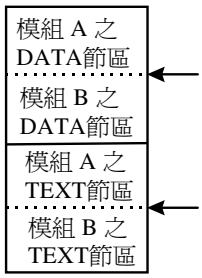
程式模組宣告方式

模組A

```
DATA  SEGMENT PUBLIC  'DATA'
DATA  ENDS
TEXT  SEGMENT PUBLIC  'CODE'
TEXT  ENDS
```

模組B

```
DATA  SEGMENT  PUBLIC  'DATA_A'
DATA  ENDS
TEXT  SEGMENT  PUBLIC  'CODE'
TEXT  ENDS
```



模組 A與模組 B合併後之DATA 節區(因兩者的class名字不相同,因而仍爲兩個節區.)

模組 A與模組 B合併後之TEXT節區(因兩者的class名字相同,因而串接成一個節區.)

典型的程式模組宣告方式

```
CODE    SEGMENT
        ASSUME  CS:CODE
MAIN:   ....                          }
        ....                          }
        CALL    NEAR PTR SUBT         } ;主程式模組
        ....      可以省略            }
SUBT    PROC    NEAR                  }
        ....                          } ;副程式模組
SUBT    ENDP
CODE    ENDS
```

(a) 宣告方式一：主程式不視爲一個副程式模組

典型的程式模組宣告方式

```
CODE    SEGMENT
        ASSUME  CS:CODE
MAIN:   PROC    FAR                   }
        ....                          }
        ....                          }
        CALL    NEAR PTR SUBT         }
        ....      可以省略            } 主程式
        ....                          }
SUBT    PROC    NEAR      }           }
        ....              } 副程式    }
        ....              }           }
SUBT    ENDP              }           }
MAIN    ENDP                          }
CODE    ENDS
```

(b) 宣告方式二：主程式視爲一個副程式模組並採用巢路宣告方式

典型的程式模組宣告方式

```
CODE    SEGMENT
        ASSUME  CS:CODE
MAIN:   PROC  FAR                    ┐
        ....                         │
        ....                         │
        CALL  NEAR PTR SUBT          ├ 主程式
        ....   可以省略              │
        ....                         │
MAIN    ENDP                         ┘
SUBT    PROC  NEAR                   ┐
        ....                         ├ 副程式
        ....                         ┘
SUBT    ENDP
CODE    ENDS
```

(c) 宣告方式二：主程式視爲一個副程式模組但不採用巢路宣告方式

.model模組宣告方式

.model 記憶器模式 [,模式選項]

最常用的三種模式爲：tiny、small、與flat。

1. tiny模式
2. small模式
3. medium模式
4. compact模式
5. large模式
6. huge模式
7. flat模式

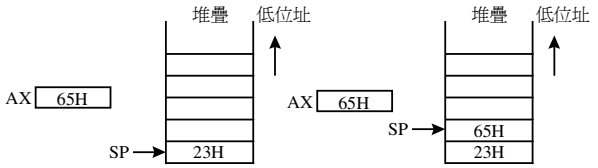
.model tiny模組宣告方式的使用例

```
                        ;ex6.2-5.asm
                        .model tiny
0000                      .code   ;code segment
                        .startup
0100                *@Startup:
0100                   START    PROC  FAR
0100  B9 0010                    MOV   CX,10H  ;read 16 characters
0103  B4 08             READ_KBY: MOV   AH,08H  ;read a character
0105  CD 21                      INT   21H     ;from keyboard
                        .IF   AL >= 'a' && AL <= 'z'
010F  2C 20                       SUB   AL,20H
                        .ENDIF
0111  8A D0              UP_CASE:  MOV   DL,AL   ;display character
0113  B4 02                       MOV   AH,02H
0115  CD 21                       INT   21H
0117  E2 EA                       LOOP  READ_KBY
                        .EXIT ;return to MS-DOS
011D                   START    ENDP
                          END
```

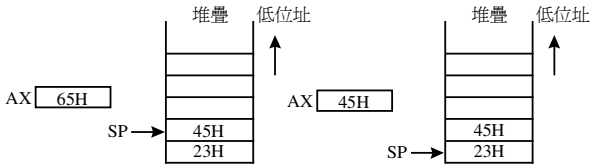

外部變數

- EXTRN <name>[,<name>....] ：宣告<name>是在其它模組中定義
- PUBLIC <name>[,<name>....]：宣告<name>是在這模組中定義的，但是允許其它模組使用。

PUSH與POP動作



(a) PUSH AX



(b) POP AX

80x86堆疊運算指令

指令	動作	OF	SF	ZF	AF	PF	CF
PUSH imm8/16	stkptr ← stkptr - 2/4	-	-	-	-	-	–
PUSH imm32 (80386↑)	(stkptr) ← imm8/16/32						
PUSH reg16	stkptr ← stkptr - 2/4	-	-	-	-	-	–
PUSH reg32 (80386↑)	(stkptr) ← reg16/reg32						
PUSH sreg	stkptr ← stkptr - 2	-	-	-	-	-	–
	(stkptr) ← sreg						
PUSH m16	stkptr ← stkptr - 2/4	-	-	-	-	-	–
PUSH m32 (80386↑)	(stkptr)←(mem16)/(mem32)						
PUSHF	stkptr ← stkptr - 2	-	-	-	-	-	–
	(stkptr) ← FLAGS						
PUSHFD (80386↑)	stkptr ← stkptr - 4	-	-	-	-	-	–
	(stkptr) ← EFLAGS						
PUSHA (80286↑)	將 AX,CX,DX,BX,及原先的 SP,BP, SI,DI 存入堆疊。	-	-	-	-	-	–
PUSHAD (80386↑)	將 EAX,ECX,EDX,EBX,及原先的 ESP,EBP,ESI,EDI 存入堆疊。	-	-	-	-	-	–

80x86堆疊運算指令

POP reg16 POP reg32 POP sreg	(80386↑)	reg16/reg32 ← (stkptr) stkptr ← stkptr + 2/4 sreg ← (stkptr) stkptr ← stkptr + 2	- - - - - -	
POP m16 POP m32	(80386↑)	(m16)/(m32) ← (stkptr) stkptr ← stkptr + 2/4	- - - - - -	
POPF		FLAGS ← (stkptr) stkptr ← stkptr + 2	除了 VM 與 RF 兩個旗號位元之外，其他旗號位元均會受影響。	
POPFD	(80386↑)	EFLAGS ← (stkptr) stkptr ← stkptr + 4	除了 VM 與 RF 兩個旗號位元之外，其他旗號位元均會受影響。	
POPA	(80286↑)	自堆疊依序取出 DI,SI,BP,SP,及 BX, DX,CX,AX 。	- - - - - -	
POPAD	(80386↑)	自堆疊依序取出 EDI,ESI,EBP, ESP, 及 EBX, EDX,ECX, EAX 。	- - - - - -	

註:PUSH CS 爲成立的指令，但是 POP CS 則爲不成立的指令； stkptr 爲 SP 或 ESP ，由位址長度爲 16 位元或 32 位元決定； reg32/m32 只能在 80386↑中使用。

堆疊區宣告

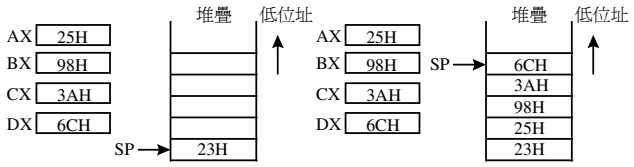
```
                    ;ex6.3-2.asm
0000                  STACK     SEGMENT  STACK 'STACK'
0000  0014 [ 0000 ]             DW    20 DUP(0)
0028                  STKTOP    LABEL WORD
0028                  STACK     ENDS
                    ;
0000                  CODE      SEGMENT PUBLIC 'CODE'
                    ASSUME    CS:CODE,SS:STACK
0000  B8 ---- R           START:    MOV   AX,STACK   ;initialize SS
0003  8E D0                       MOV   SS,AX      ;and SP
0005  BC 0028 R                    MOV   SP,OFFSET STKTOP

0008                  CODE      ENDS
                          END
```

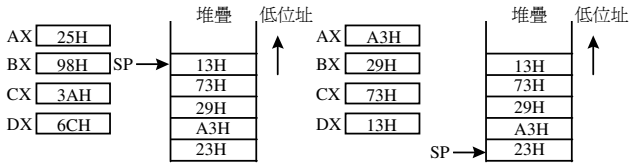
堆疊運算例

```
                    ;ex6.3-3.asm
0000                  CODE     SEGMENT  'CODE'
0000                  SRREGS   PROC  NEAR
0000  50                      PUSH  AX    ;push ax
0001  53                      PUSH  BX    ;push bx
0002  51                      PUSH  CX    ;push cx
0003  52                      PUSH  DX    ;push dx
                    ;         .
                    ;         .
                    ;         .
0004  5A                      POP   DX    ;restore dx
0005  59                      POP   CX    ;restore cx
0006  5B                      POP   BX    ;restore bx
0007  58                      POP   AX    ;restore ax
0008  C3                      RET
0009                  SRREGS   ENDP
0009                  CODE     ENDS
                        END
```


堆疊運算例



(a) 四個 PUSH 指令執行前　　(b) 四個 PUSH 指令執行後



(c) 四個 POP 指令執行前　　(d) 四個 POP 指令執行後

80x86節區內副程式呼叫與歸回指令

指令	動作	OF	SF	ZF	AF	PF	CF
CALL rel16 CALL rel32(80386↑)	若運算元長度爲 16 位元則 儲存 IP 於堆疊; (80x86) IP ← IP + rel16 EIP←(EIP + rel16) ∧ 0000FFFFH (80386↑) 否則(運算元長度爲 32 位元) (80386↑) 儲存 EIP 於堆疊; EIP ← EIP + rel32	-	-	-	-	-	-
CALL reg16 CALL m16 CALL reg32 (80386↑) CALL m32 (80386↑)	若運算元長度爲 16 位元則 儲存 IP 於堆疊; (80x86) IP ← reg16/(m16) EIP←EIP ∧ 0000FFFFH (80386↑) 否則(運算元長度爲 32 位元) (80386↑) 儲存 EIP 於堆疊; EIP ← reg32/(m32)	-	-	-	-	-	-

80x86節區內副程式呼叫與歸回指令

RET	若運算元長度爲 16 位元則 (80x86) 自堆疊取回 IP; EIP← EIP ∧ 0000FFFFH (80386↑) 否則(運算元長度爲 32 位元) 自堆疊取回 EIP;	-	-	-	-	-	-
RET imm16	若運算元長度爲 16 位元則 (80x86) 自堆疊取回 IP; SP ← SP + imm16 EIP ← EIP ∧ 0000FFFFH (80386↑) 否則(運算元長度爲 32 位元) 自堆疊取回 EIP; ESP ← ESP + imm16	-	-	-	-	-	-

80x86節區間副程式呼叫與歸回指令

指令	動作	OF	SF	ZF	AF	PF	CF
(直接遠程呼叫) CALL ptr16:16 CALL ptr16:32 (80386↑)	若運算元長度爲 16 位元則 儲存 CS 與 IP 於堆疊; (80x86) CS:IP ← ptr16:16; EIP ← EIP ∧ 0000FFFFH; (80386↑) 否則(運算元長度爲 32 位元) (80386↑) 儲存 CS 與 EIP 於堆疊; CS:EIP ← ptr16:32;	-	-	-	-	-	-
(間接遠程呼叫) CALL m16:16 CALL m16:32 (80386↑)	若運算元長度爲 16 位元則 儲存 CS 與 IP 於堆疊; (80x86) CS:IP ← (m16:16); EIP ← EIP ∧ 0000FFFFH; (80386↑) 否則(運算元長度爲 32 位元) (80386↑) 儲存 CS 與 EIP 於堆疊; CS:EIP ← (m16:32);	-	-	-	-	-	-

80x86節區間副程式呼叫與歸回指令

RET	若運算元長度爲 16 位元則 自堆疊取出 IP; (80x86) 自堆疊取出 CS; EIP← EIP ∧ 0000FFFFH; (80386↑) 否則(運算元長度爲 32 位元) (80386↑) 自堆疊取出 EIP; 自堆疊取出 CS;	-	-	-	-	-	-
RET imm16	若運算元長度爲 16 位元則 自堆疊取出 IP; (80x86) 自堆疊取出 CS; SP ← SP + imm16 EIP← EIP ∧ 0000FFFFH; (80386↑) 否則(運算元長度爲 32 位元) (80386↑) 自堆疊取出 EIP; 自堆疊取出 CS; ESP ← ESP + imm16	-	-	-	-	-	-

副程式的使用例

```
                        ;ex6.3-5.asm
0000                    DATA      SEGMENT PUBLIC 'DATA'
= 0008                  BCOUNT    EQU   08H       ;bit number
0000 47                 TDATA     DB    47H       ;test data
0001 00                 COUNT     DB    00H       ;result
0002                    DATA      ENDS
0000                    STACK     SEGMENT  STACK   ;stack segment
0000  0014 [ 0000 ]               DW    20 DUP(0)
0028                    STKTOP    LABEL WORD
0028                    STACK     ENDS
                            ....
```

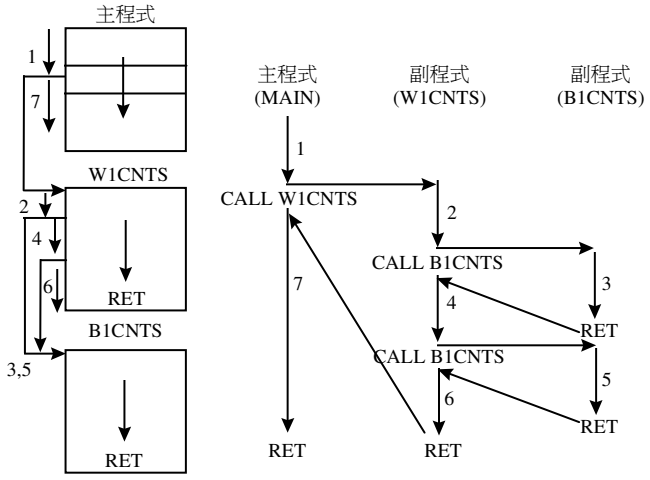

副程式的使用例

```
0000                    MAIN      PROC  NEAR
0000  B8 ---- R                   MOV   AX,STACK   ;initialize SS
0003  8E D0                       MOV   SS,AX      ;and SP
0005  BC 0028 R                     MOV   SP,OFFSET STKTOP
0008  B8 ---- R                   MOV   AX,DATA    ;initialize DS
000B  8E D8                       MOV   DS,AX
000D  A0 0000 R                     MOV   AL,TDATA ;pass parameter
0010  E8 0004                     CALL  NEAR PTR B1CNTS;to B1CNTS
0013  A2 0001 R                     MOV   COUNT,AL ;save result
0016  C3                          RET
0017                    MAIN      ENDP
                        ;subroutine starts here.
0017                    B1CNTS    PROC  NEAR
                            ....
0026  C3                          RET
0027                    B1CNTS    ENDP
```

不同節區間的副程式使用例

```
                    ;ex6.3-6.asm
                    ;main program starts here.
0000                  MAIN     PROC  NEAR
0000  B8 ---- R                MOV   AX,STACK   ;initialize SS
                               ....
0008  B8 ---- R                MOV   AX,DATA    ;initialize DS
000B  8E D8                    MOV   DS,AX
000D  A0 0000 R                  MOV   AL,TDATA ;pass parameter
0010  9A ---- 0000 R             CALL  FAR PTR B1CNTS;to B1CNTS
0015  A2 0001 R                  MOV   COUNT,AL ;save result
0018  C3                       RET
                               .....
                    ;subroutine starts here.
                    ;(another segment)
0000                  CODE_A    SEGMENT PUBLIC 'CODE_A'
                    ASSUME    CS:CODE_A,DS:DATA,SS:STACK
0000                  B1CNTS    PROC  FAR
0000  B9 0008                   MOV   CX,BCOUNT;get count
                               ....
000F  CB                       RET
0010                  B1CNTS    ENDP
```


巢路副程式



巢路副程式例

```
                        ;ex6.3-7.asm
0000                    DATA    SEGMENT PUBLIC 'DATA'
= 0008                  BCOUNT  EQU  08H      ;bit number
0000 7647               TDATA   DW   7647H    ;test data
0002 00                 COUNT   DB   00H      ;result
0003                    DATA    ENDS
                        ;
0000                    STACK   SEGMENT STACK 'STACK'
0000  0014 [ 0000 ]             DW   20 DUP(0)
0028                    STKTOP  LABEL WORD
0028                    STACK   ENDS
                        ;
0000                    CODE    SEGMENT PUBLIC 'CODE'
                        ASSUME  CS:CODE,DS:DATA,SS:STACK
                        ;
```


巢路副程式例

```
                        ;main program starts here.
0000                    MAIN    PROC  NEAR
0000  B8 ---- R                 MOV   AX,STACK  ;initialize SS
                                ....
0008  B8 ---- R                 MOV   AX,DATA   ;initialize DS
.                               ....
000D  A1 0000 R                 MOV   AX,TDATA ;pass parameter
0010  E8 0004                   CALL  NEAR PTR W1CNTS;to W1CNTS
0013  A2 0002 R                 MOV   COUNT,AL ;save result
0016  C3                       RET
                                ....
0017                    W1CNTS  PROC  NEAR
0017  E8 0008                   CALL  NEAR PTR B1CNTS
001A  86 C4                     XCHG  AL,AH   ;exchange AL and AH
001C  E8 0003                   CALL  NEAR PTR B1CNTS
001F  02 C4                     ADD   AL,AH
0021  C3                       RET     ;return to main program
                        ;count the number of 1 bits in a byte
0022                    B1CNTS  PROC  NEAR
0022  B9 0008                   MOV   CX,BCOUNT;get count
                                .....
002F  8A C3                     MOV   AL,BL   ;return result
0031  C3                       RET
```

副程式參數傳遞方式

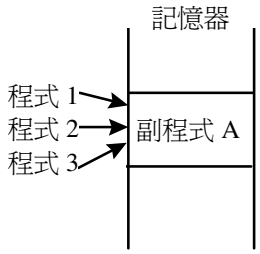
在組合語言中，最常用的參數傳遞方式爲：

1. 利用微處理器內部的暫存器；
2. 利用共同的記憶器區域；
3. 內線參數區(in-line parameter area)；
4. 利用堆疊。

四種參數傳遞方式

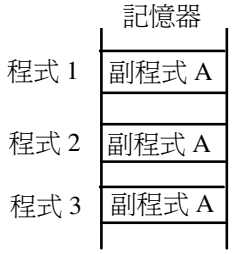
類型	傳遞方法	特性
暫存器	利用微處理器內部暫存器	1. 參數數目受到限制。 2. 動態方式。
共同記憶器區 (參數區)	利用一塊公用的記憶器區域	1. 靜態方式:若該區域在組譯時即確定時。 2. 動態方式:若該區域的基底位址是由暫存器傳遞時。
內線參數區	參數儲存於 CALL 指令後，副程式再計算出參數的位址。	1. 靜態方式。 2. 歸回位址必須處理。
堆疊	利用堆疊	動態方式。

副程式的共用



程式 1, 2, 與 3 可共用副程式 A

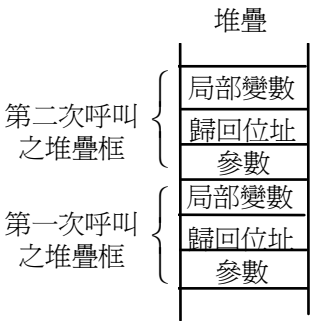
(a) 可重入副程式方法



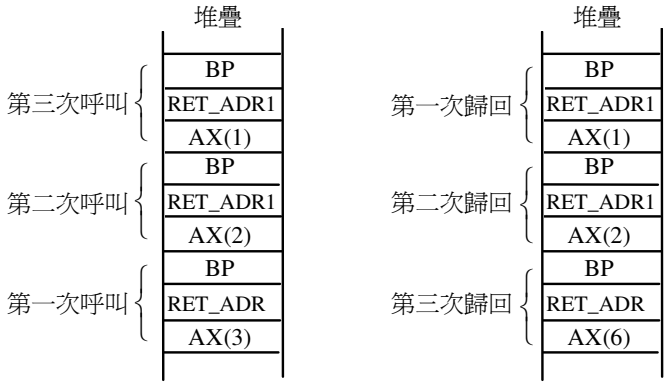
程式 1, 2, 與 3 必須各自擁有一份副程式 A之副本

(b) 普通副程式方法

遞回呼叫期間堆疊框的增長



遞迴副程式—*N*!計算程式



(a) 遞回呼叫　　(b) 反算過程

計算N! (N<=8)的程式例

```
                        ;ex6.3-11.asm
                        ;recursive and reentrant subroutine example
                        ;----calculates N! (N factorial)
                        ;assumes 0 < N <= 8 (i.e. the length of
                        ;product is less than one word).
                        ;parameters are passed on the stack
                        ;
0000                      DATA      SEGMENT PUBLIC 'DATA'
0000 0003                   NUMBER    DW    03H       ;test data
0002 0000                   RESULT    DW    00H       ;result
0004                      DATA      ENDS
                        ;
0000                      STACK     SEGMENT STACK 'STACK'
0000  0032 [ 0000 ]               DW    50 DUP(0)
0064                      STKTOP    LABEL WORD
0064                      STACK     ENDS
                        ;
0000                      CODE      SEGMENT PUBLIC 'CODE'
                        ASSUME    CS:CODE,DS:DATA,SS:STACK
```

計算N! (N<=8)的程式例

```
                        ;
                        ;main program starts here.
                        ;
0000                    MAIN    PROC  NEAR
0000  B8 ---- R                 MOV   AX,STACK   ;initialize SS
0003  8E D0                     MOV   SS,AX      ;and SP
0005  BC 0064 R                 MOV   SP,OFFSET STKTOP
0008  B8 ---- R                 MOV   AX,DATA    ;initialize DS
000B  8E D8                     MOV   DS,AX
                        ;the parameters are passed to and from
                        ;subroutine on the stack
000D  A1 0000 R                 MOV   AX,NUMBER;pass parameter
0010  50                        PUSH  AX     ;to FACTOR on the
0011  E8 0005                   CALL  NEAR PTR FACTOR;stack
0014  58                RET_ADR: POP  AX     ;get result and
0015  A3 0002 R                 MOV   RESULT,AX  ;save it
0018  C3                        RET
0019                    MAIN    ENDP
```


計算N! (N<=8)的程式例

```
                        ;
                        ;subroutine starts here.
                        ;
0019                    FACTOR  PROC  NEAR
0019  55                        PUSH  BP     ;make a stack frame
001A  8B EC                     MOV   BP,SP  ;and get parameter
001C  8B 46 04                  MOV   AX,[BP+4] ;from the stack
001F  83 E8 01                  SUB   AX,01    ;subtract 1
0022  75 02                     JNE   F_CONT   ;if zero than
0024  EB 0B                     JMP   SHORT RETURN  ;return
0026  50                F_CONT: PUSH  AX       ;else call FACTOR
0027  E8 FFEF                   CALL  NEAR PTR FACTOR
002A  58                RET_ADR1: POP  AX      ;backwards calculate
002B  F7 66 04                  MUL   WORD PTR [BP+4];N! and save
002E  89 46 04                  MOV   [BP+4],AX;result in stack
0031  5D                RETURN: POP   BP       ;pop BP and return
0032  C3                       RET
0033                    FACTOR  ENDP
0033                    CODE    ENDS
                            END   MAIN
```

80286高階語言支援指令

指令	動作	OF	SF	ZF	AF	PF	CF
ENTER imm16,0	產生副程式堆疊框(見內文說明)。	-	-	-	-	-	-
ENTER imm16,1	產生副程式堆疊框(見內文說明)。	-	-	-	-	-	-
ENTER imm16,imm8	產生副程式堆疊框(見內文說明)。	-	-	-	-	-	-
LEAVE(16 位元位址)	儲存 BP 於 SP ，然後自堆疊取回 BP 。	-	-	-	-	-	-
LEAVE(32 位元位址) (80386↑)	儲存 EBP 於 ESP ，然後自堆疊取回 EBP 。	-	-	-	-	-	-
SETcc r8/m8 (80386↑) (cc 條件與 Jcc 相同)	若 cc 條件滿足，則設定 r8/(m8) 爲 1; 否則清除 r8/(m8) 爲 0 。	-	-	-	-	-	-

巨集指令定義

巨集指令名稱 MACRO 虛擬參數

雛型碼

ENDM

巨集指令定義

例題 6.4-1　(巨集指令定義)

定義一個巨集指令執行下列表式：

X ← X+Y-3

解：完整的巨集指令定義如程式 6.4-1 所示。

程式 6.4-1　定義巨集指令： X ← X+Y-3

```
;ex6.4-1.asm
CAL_EXP   MACRO X,Y   ;X <- X+Y-3
          MOV   AX,X
          ADD   AX,Y
          SUB   AX,3
          MOV   X,AX
          ENDM
```


巨集指令使用例

```
                    ;ex6.4-2.asm
                    CAL_EXP   MACRO X,Y  ;X <- X+Y-3
                          MOV   AX,X
                          ADD   AX,Y
                          SUB   AX,3
                          MOV   X,AX
                          ENDM
                    ;
0000                  DATA      SEGMENT PUBLIC 'DATA'
0000 0012               OPR1      DW    12H
0002 0034               OPR2      DW    34H
0004 0056               OPR3      DW    56H
0006 0000               RESULT    DW    00H
0008                  DATA      ENDS
                    ;
```

巨集指令使用例

```
0000                    CODE    SEGMENT PUBLIC 'CODE'
                       ASSUME   CS:CODE,DS:DATA
0000                    START    PROC  NEAR
0000  B8 ---- R                 MOV   AX,DATA
0003  8E D8                     MOV   DS,AX
                      CAL_EXP OPR1,OPR2
0005  A1 0000 R          1       MOV   AX,OPR1
0008  03 06 0002 R       1       ADD   AX,OPR2
000C  83 E8 03           1       SUB   AX,3
000F  A3 0000 R          1       MOV   OPR1,AX
0012  A1 0000 R                  MOV   AX,OPR1
                      CAL_EXP OPR3,OPR2
0015  A1 0004 R          1       MOV   AX,OPR3
0018  03 06 0002 R       1       ADD   AX,OPR2
001C  83 E8 03           1       SUB   AX,3
001F  A3 0004 R          1       MOV   OPR3,AX
0022  03 06 0004 R                ADD   AX,OPR3
0026  C3                       RET
0027                    START    ENDP
0027                    CODE     ENDS
                            END  START
```


不當的巨集指令使用

```
                        ;ex6.4-3.asm
                                ....
0000  B8 ---- R                 MOV   AX,DATA
0003  8E D8                     MOV   DS,AX
                      CAL_EXP OPR1,54H
0005  A1 0000 R          1       MOV   AX,OPR1
0008  83 C0 54           1       ADD   AX,54H
000B  83 E8 03           1       SUB   AX,3
000E  A3 0000 R          1       MOV   OPR1,AX
0011  A1 0000 R                  MOV   AX,OPR1
                      CAL_EXP 23,OPR2
0014  B8 0017            1       MOV   AX,23
0017  03 06 0002 R       1       ADD   AX,OPR2
001B  83 E8 03           1       SUB   AX,3
                    1         MOV   23,AX
ex643.asm(23): error A2001: immediate operand not allowed
001E  03 06 0004 R                ADD   AX,OPR3
0022  C3                       RET
```

標記重複定義問題

```
                        ;ex6.4-5.asm
                        ABS_V     MACRO X       ;find absolute
                                  CMP   X,00H   ;value
                                  JGE   NEXT
                                  NEG   X
                        NEXT:     NOP
                                  ENDM
                                  ....
                                  ABS_V OPR1    ;macro call
0005  83 3E 0000 R 00      1          CMP   OPR1,00H   ;value
000A  7D 04                1          JGE   NEXT
000C  F7 1E 0000 R         1          NEG   OPR1
0010  90                   1  NEXT:   NOP
0011  A1 0000 R                       MOV   AX,OPR1
                                  ABS_V OPR2    ;macro call
0014  83 3E 0002 R 00      1          CMP   OPR2,00H   ;value
0019  7D F5                1          JGE   NEXT
001B  F7 1E 0002 R         1          NEG   OPR2
001F  90                   1  NEXT:   NOP
ex645.asm(22): error A2005: symbol redefinition : NEXT
 ABS_V(4): Macro Called From
```


局部標記宣告

```
                        ;ex6.4-6.asm
                        ABS_V     MACRO X       ;find absolute
                                  LOCAL NEXT
                                  CMP   X,00H   ;value
                                  JGE   NEXT
                                  NEG   X
                        NEXT:     NOP
                                  ENDM
                                  ....
                                  ABS_V OPR1    ;macro call
0005  83 3E 0000 R 00      1          CMP   OPR1,00H   ;value
000A  7D 04                1          JGE   ??0000
000C  F7 1E 0000 R         1          NEG   OPR1
0010  90                   1  ??0000: NOP
0011  A1 0000 R                       MOV   AX,OPR1
                                  ABS_V OPR2    ;macro call
0014  83 3E 0002 R 00      1          CMP   OPR2,00H   ;value
0019  7D 04                1          JGE   ??0001
001B  F7 1E 0002 R         1          NEG   OPR2
001F  90                   1  ??0001: NOP
                                  ....
```

巢路巨集指令

```
;ex6.4-7.asm
PSH_REG   MACRO          ;save registers
          PUSH  AX
          PUSH  DX
          ENDM
;
POP_REG   MACRO          ;restore registers
          POP   DX
          POP   AX
          ENDM
;
DIF_SQR   MACRO X,Y,ERR   ;find SQR(X-Y)
          PSH_REG         ;call macro
          MOV   AX,X
          SUB   AX,Y
          IMUL  AX
          MOV   ERR,AX
          POP_REG         ;call macro
          ENDM
```


巢路巨集指令

```
                    ;
0000                  DATA     SEGMENT PUBLIC 'DATA'
0000 0025               OPR1     DW    25H
0002 0047               OPR2     DW    47H
0004 0000               RESULT   DW    00H
0006                  DATA     ENDS
                          ....
0000                  START    PROC  NEAR
0000  B8 ---- R                 MOV   AX,DATA
0003  8E D8                     MOV   DS,AX
                          DIF_SQR OPR1,OPR2,RESULT
0005  50              2          PUSH  AX
0006  52              2          PUSH  DX
0007  A1 0000 R          1          MOV   AX,OPR1
000A  2B 06 0002 R       1          SUB   AX,OPR2
000E  F7 E8              1          IMUL  AX
0010  A3 0004 R          1          MOV   RESULT,AX
0013  5A              2          POP   DX
0014  58              2          POP   AX
0015  C3                        RET
                          ....
```