Link to Subprograms

Objectives

- Modular programming style
- Sharing data among subprograms
- Link between programs with different languages

Modular Program

- Program modules are written and stored in separate files to facilitate the development of large projects in a modular fashion
- Each program is assembled separately (generating an .obj file for each program)
- Linker links the .obj files together and generates a .exe file

Modular Program

- One subprogram (usually the main) calls another subprogram (or many other subprograms)
- A subprogram may call itself (Recursion)

SEGMENT Directive

[seg-name]	SEGMENT	[align][combine]['class']

Options for SEGMENT directive

- Align type -- align the named segment beginning on storage boundary
  - BYTE, WORD, DWORD, PARA(default), PAGE
- Combine type -- combine or keep segment separate
  - NONE(default), PUBLIC, COMMON
- Class type
  - to group segments -- 'code', 'stack', 'data'

Intra-Segment Call

- procedures defined as NEAR
- Can be called only from within the same code segment
- Save IP on the stack and load IP reg with offset address of called procedure

```
                CALL Nearproc
                ...
    Nearproc    PROC NEAR
                ...
                RETN
    Nearproc    ENDP
```

Inter-Segment Call

- **procedure defined as FAR**
- **The CALL instruction pushes the current IP and the CS onto the stack. Then IP Register is loaded with offset of called procedure and the CS is loaded with its segment address**

Example

```
CALL    Farproc
        ...
Farproc PROC FAR
        ...
        RETF
Farproc ENDP
```

RETF pops CS then IP from the stack

EXTERN and PUBLIC Directives

```
EXTERN    name:type [,…]
```

Used to tell the assembler that the following symbol is defined elsewhere (in another module)

type:

- ABS -- for constant values
- BYTE, WORD, DWORD -- for data items
- NEAR, FAR -- for procedures

EXTERN and PUBLIC Directives

```
PUBLIC    symbol [,…]
```

Used to tell the assembler that this symbol can be used by other modules

Symbol:

- label -- can be used to declare procedure name as public
- variable -- used to declare public data items

EXTERN and PUBLIC Directives

```
EXTRN           SUBPROG: FAR
MAINPROG        PROC        FAR
                ...
                CALL        SUBPROG
                ...
MAINPROG        ENDP
--------------------------------------
                PUBLIC      SUBPROG
SUBPROG         PROG        FAR
                ...
                RETF
SUBPROG         ENDF
```

Data Sharing

There are 3 ways to share data

- Using the general purpose registers
- Using global variables
- Passing parameters through the stack

General Purpose Registers

- **Example -- program compute total cost based on quantity and price per unit**
- **See page 425 of text book**

Main procedure defines input data, QTY and PRICE, then calls procedure SUB1 which computes and returns the total cost

```
TITLE          Main1 (EXE)
               .MODEL MEDIUM
               .STACK 64
               EXTRN  SUB1: FAR
;-------------------------------------
               .DATA
QTY            DW      0140h
PRICE          DW      2500h
;-------------------------------------
               .CODE
BEGIN          PROC    FAR
               MOV     AX, @data
               MOV     DS, AX
               MOV     AX, PRICE
               MOV     BX, QTY
               CALL    SUB1
               MOV     AX, 4C00h
               INT     21H
BEGIN          ENDP
               END     BEGIN
```

```
TITLE    SUB1     CALLED SUBPROGRAM
          .MODEL   MEDIUM
          .CODE
 SUB1     PROC     FAR
          PUBLIC   SUB1
          MUL      BX
          RETF
 SUB1     ENDP
          END      SUB1
```

- Data is passed from main to procedure SUB through AX and BX
- Result is passed from SUB to Main through DX:AX

```
TITLE  Main1 (EXE)
               EXTRN  SUB1: FAR
STACKSG        SEGMENT PARA STACK 'stack'
               DW      64 DUP(?)
STACKSG        ENDS
DATASG         SEGMENT PARA 'Data'
QTY            DW      0140h
PRICE          DW      2500h
DATASG         ENDS
CODESG         SEGMENT PARA PUBLIC 'Code'
BEGIN          PROC    FAR
               ASSUME  CS:CODESG, DS:DATASG, SS:STACKSG
               MOV     AX, DATASG
               MOV     DS, AX
               MOV     AX, PRICE
               MOV     BX, QTY
               CALL    SUB1
               MOV     AX, 4C00h
               INT     21H
BEGIN          ENDP
CODESG         ENDS
               END     BEGIN
```

```
TITLE    SUB1     CALLED SUBPROGRAM
CODESG   SEGMENT  PARA PUBLIC  'Code'
SUB1     PROC     FAR
         ASSUME   CS:CODESG
         PUBLIC   SUB1
         MUL      BX
         RETF
SUB1     ENDP
CODESG   ENDS
         END      SUB1
```

- The PUBLIC code segment: code segments will be combined by the linker into one code segment
- Data is passed from main to procedure SUB through AX and BX
- Result is passed from SUB to Main through DX:AX

Sharing Data Through Global Variables

```
TITLE          Main1 (EXE)
               .MODEL MEDIUM
               .STACK 64
               EXTRN  SUB1: FAR
               PUBLIC QTY, PRICE
;------------------------------
               .DATA
QTY            DW       0140h
PRICE          DW       2500h
;------------------------------
               .CODE
BEGIN          PROC     FAR
               MOV      AX, @data
               MOV      DS, AX
               CALL     SUB1
               MOV      AX, 4C00h
               INT      21H
BEGIN          ENDP
               END      BEGIN
```

```
TITLE   SUB1    CALLED SUBPROGRAM
        .MODEL  MEDIUM
        EXTRN QTY:WORD, PRICE:WORD
        .CODE
SUB1    PROC    FAR
        PUBLIC  SUB1
        MOV     AX, PRICE
        MOV     BX, QTY
        MUL     BX
        RETF
SUB1    ENDP
        END     SUB1
```

Sharing Data By Passing Parameters Through the Stack

Passing Parameters to the Stack

- Parameters are pushed on the stack before the call
- The call instruction pushes IP then CS on the stack
- The called procedure, pushes the contents of the BP register on top of the stack
- SP is copied to BP -- BP points to the top of the stack
- When the called procedure starts execution, shared data is accessed using the BP register and indexing into the stack.
- At the end of procedure, pop BP to restore

```
TITLE          Main1 (EXE)
               .MODEL MEDIUM
               .STACK 64
                EXTRN  SUB1: FAR
;--------------------------------
               .DATA
QTY            DW       0140h
PRICE          DW       2500h
;--------------------------------
               .CODE
BEGIN          PROC     FAR
               MOV      AX, @data
               MOV      DS, AX
               PUSH     PRICE
               PUSH     QTY
               CALL     SUB1
               MOV      AX, 4C00h
               INT      21H
BEGIN          ENDP
               END      BEGIN
```

```
TITLE   SUB1    CALLED SUBPROGRAM
        .MODEL  MEDIUM
        .CODE
SUB1    PROC    FAR
        PUBLIC  SUB1
        PUSH    BP          ;SAVE ORIGINAL BP
        MOV     BP,SP       ;POINT BP TO TOP OF STACK
        MOV     AX,[BP+8]   ;GET PRICE
        MOB     BX,[BP+6]   ;GET QTY
        MUL     BX
        POP     BP          ;RESTORE ORIGINAL BP
        RETF    4           ;POP IP,CS,4 ADDITIONAL BYTES
SUB1    ENDP
        END     SUB1
```

Linking Turbo Assembly with Borland C++

Language Interface

Two methods --

- **Inline Assembly Code**
  - Insert assembly statements, preceded by the keyword `asm`

```
asm     INC  WORD PTR     count
```

- **Separate Modules**

Separate Modules

- **CASE sensitivity**
  - C++ is case sensitive

  => Assembly module should use the same case (I.e. upper and lower) for any variable or procedure names that are in common
- **Naming Convention**
  - All assembler references to functions and variables in the C module must begin with an underscore

Example

- **Count the number of all odd integer values between InitialValue and FinalValue.**

Example

```
#include <iostream.h>
extern “C” int DoTotal(void);externally defined procedure
extern int InitialValue     ;externally defined variable

int FinalValue;
int main(void)
{
        FinalValue = 10;
        InitialValue = 2;
        cout << DoTotal();
        return 0;
}
```

```
        .MODEL  SMALL
        EXTRN   _FinalValue             ; externally defined
        PUBLIC  _InitialValue, _DoTotal ; available to other
        .DATA
_InitialValue  DW      0
        .CODE
_DoTotal       PROC
        MOV     CX, _FinalValue
        MOV     DX, _InitialValue
        MOV     AX, 0                   ; Initialize Counter
LP:     TEST     DX, 0001H              ; is DX odd ?
        JZ      EVEN
        INC     AX
EVEN:   INC     DX
        CMP     DX, CX
        JLE     LP
        RET                     ;return  final total in AX
_DoTotal       ENDP
               END
```

Data Type Compatibility

C++ Data Type	Assembler Data Type
char	**byte**
short	**word**
int	**word**
long	**double word**
float	**double word**
double	**quad word**
long double	**ten bytes**

C++ Function Returned Value

Returned Value Type	Return Location
char	AX
short	AX
int	AX
long	DX:AX

Link Assembly and C/C++ Modules

- **Edit your assembly module using any text editor**
- **Edit your C/C++ module using any text editor**
- **from DOS, type the following command**

  bcc - Option file1.cpp file2.asm

  Option : ms for small model

  mm for medium

  ml for large

- **bcc command will compile C/C++ file(s), assemble tasm file(s), and link them into a single executable file called file2.exe**

Parameters Passing

- **C++ passes parameters to functions on the stack. Before calling a function, C++ first pushes the parameters to that functions onto the stack, starting with the right-most parameter and ending with the left-most.**
- **Example: the function call Test(Var1,Var2,Var3);**

compiles to:

```
PUSH  _Var3
PUSH  _Var2
PUSH  _Var1
CALL _Test
ADD   SP,6
```

```
;Assume function Test returns (Var1+Var2)-Var3
        .MODEL    SMALL
        .CODE
        PUBLIC    _Test
_Test   PROC
        PUSH      BP
        MOV       BP,SP
        MOV       AX,[BP+4]; get  Var1
        ADD       AX,[BP+6]; Add  Var2 to Var1
        SUB       AX,[BP+8]; sub  Var3
        POP       BP
        RET
_Test   ENDP
        END
```

Linking with C

- **Write a program that takes as input a list of integer numbers and computes their sum.**
- **Let's write the program in two separate modules, one in C and one in assembly.**
- **The main module (in C) declares the data items and reads the input from the keyboard**
- **The Assembly module computes the sum.**

```
#include <iostream.h>
// externally defined procedure
extern “C” int SumAll(int [ ], int);

void ReadData(int [] , int);
int main(void)
{
    int  array[10];
    int size, sum;

    cout << " Enter number of items:" << endl;
    cin >> size;
    cout << "Please enter a list of "
         <<size<< "integers: ";
    ReadData(array, size);
    sum = SumAll(array, size) ;
    cout << sum;
    return 0;
}
```

```
void ReadData(int X[ ], int N)
{
     for (int i=0; i < N; ++i)
          cin >> X[i];
}
```