Chapter 1_3

Advanced C Techniques for Embedded Systems Programming

Advanced C Techniques for Embedded Systems Programming

- Linking C code and assembly code

VC++

Function Template for _asm{ }

- In VC++, assembly code must be written in a C file with the _asm{ } directive.
  - You write assembly code inside { } of _asm{ }
  - _asm{ } must appear inside a C function.
- For the function on the left, the compiler generates the assembly code on the right

```
void func2(int p1, int p2, int p3, int p4) {
   int i1, i2, i3, i4;
   _asm {    }
}
```

if i1 ~ i4 are not referred to in _asm{ }, no local variables are allocated in the stack frame and these two lines are omitted

```
_TEXT       SEGMENT
_p1$ = 8;   size = 4
_p2$ = 12; size = 4
_p3$ = 16; size = 4
_p4$ = 20; size = 4
_func2      PROC
; 5    : void func2(int p1, int p2, int p3, int p4) {
  0000055 push ebp
  000018b ec mov ebp, esp
  0000383 ec 10 sub esp, 16; 00000010H
; 6    : int i1, i2, i3, i4;
; 7    : _asm {    }
; 8    : }
  000128b e5 mov esp, ebp
  000035d pop ebp
  00004c3 ret 0
_func2ENDP
_TEXTENDS
```

Linking C code and Assembly code (VC_x86)

- In _asm{ }, you can access all global variables, parameters, and local variables just by their names.

```
int gg = 11;

int g(int g1, int g2) {
   int x, y;
   x = g1;
   y = g2;
   gg = gg + x + y;
   return f(x, y, gg);
}
```

The program on the left can be written in assembly language shown on the right

```
int g(int g1, int g2) {
   int x, y;
   _asm {
      mov  eax, g1
      mov  x, eax
      mov  eax, g2
      mov  y, eax
      mov eax, gg
      add eax, x
      add eax, y
      push gg
      push y
      push x
      call f
      add esp, 12
   }
}
```

- The compiler yields the assembly program (.cod file) on the next slide.

Linking C code and Assembly code (VC_x86) (cont)

```
_TEXT       SEGMENT
_x$ = -8;    size = 4
_y$ = -4;    size = 4
_g1$ = 8;    size = 4
_g2$ = 12; size = 4
_g               PROC
; 19   : int g(int g1, int g2) {
  0000055 push ebp
  000018b ec mov ebp, esp
  0000383 ec 08 sub esp, 8
; 20   : int x, y;
; 21   : _asm {
; 22   : mov  eax, g1
  000068b 45 08 mov eax, DWORD PTR _g1$[ebp]
; 23   : mov  x, eax
  0000989 45 f8 mov DWORD PTR _x$[ebp], eax
; 24   : mov  eax, g2
  0000c8b 45 0c mov eax, DWORD PTR _g2$[ebp]
; 25   : mov  y, eax
  0000f89 45 fc mov DWORD PTR _y$[ebp], eax
; 26   :
; 27   : mov eax, gg
  00012a1 00 00 00 00 mov eax, DWORD PTR _gg
; 28   : add eax, x
  0001703 45 f8 add eax, DWORD PTR _x$[ebp]
; 29   : add eax, y
  0001a03 45 fc add eax, DWORD PTR _y$[ebp]
; 30   : push gg
  0001dff 35 00 00 00
00 push DWORD PTR _gg
; 31   : push y
  00023ff 75 fc push DWORD PTR _y$[ebp]
; 32   : push x
  00026ff 75 f8 push DWORD PTR _x$[ebp]
; 33   : call f
  00029e8 00 00 00 00 call _f
; 34   : add esp, 12
  0002e83 c4 0c add esp, 12; 0000000cH
; 34   : }
; 35   : }
  0002e8b e5 mov esp, ebp
  000305d pop ebp
  00031c3 ret 0
_g ENDP
_TEXT ENDS
```

Linking C code and Assembly code (VC_x86) (cont)

- In the assembly program in _asm{ }, instead of writing:

```
; 22   : mov  eax, g1
```

  which is translated to:

```
000068b 45 08 mov eax, DWORD PTR _g1$[ebp]     ; where _g1$ = 8
```

  You can write "mov   eax, DWORD PTR [ebp + 8]"
  - Note: the assembler does not accept "mov   eax, DWORD PTR +8[ebp]"

- Similarly, instead of writing :

```
; 27   : mov eax, gg
```

  you can write

  "mov eax, DWORD PTR gg"

- Notes:
  - When accessing gg in data or bss, do not write _gg
  - Similarly, to call function f( ), write, in _asm { }, "call f"   Do not write "call _f"

Function Template for _asm{ } (cont)

- For the program on the left, the compiler generates the program on the right (in .cod)

```
int func1() {
   __asm {    }
}
```

```
_TEXT SEGMENT
_func1 PROC
; 1    : int func1() {
  0000055 push ebp
  000018b ec mov ebp, esp
; 2    :     __asm {    }
; 3    : }
  000035d pop ebp
  00004c3 ret 0
_func1ENDP
_TEXTENDS
```

Writing Assembly Code in __asm{ }

- Write a function that accepts three arguments and declares 3 local variables using the function template on the previous slide

```
_TEXT SEGMENT
_func1 PROC
; 1    : int func1() {
  0000055 push ebp
  000018b ec mov ebp, esp
; 2    :    __asm {    }
; 3    : }
  000035d pop ebp
  00004c3 ret 0
_func1ENDP
_TEXTENDS
```

- This function should have function prototype:
  "int func1( );"
- The caller should pass three (int) arguments:
  "x = func1(i1, i2, 34);"
- In _asm { },
  - allocate area for three local variables.
    - subtract the number of bytes for the local variables from esp
  - access parameters and local variables by
    "mov eax, DWORD PTR[ebp – 8]", or
    "mov eax, DWORD PTR[ebp + 12]", etc
  - at the end in _asm{ }, deallocate the area for the local variables
    - move the contents of ebp to esp
  - make sure that the return value is in eax

Writing Assembly Code in __asm{ } (cont)

- After you write assembly code in _asm { }, make sure to generate .cod file by

  “cl /FAcs /Od /c file.c”   /c to stop after compilation (no linking)

  and check each line of the generated assembly code
- Work on AsmTest1.c, AsmTest1_Asm1.c and AsmTest1_Asm2.c