CS2422 Assembly Language & System Programming

November 7, 2006

Announcement

• Midterm exam next Tuesday (11/14).
• Scope: All covered sections in Chapters 1 to 6, 8, and 10.
Today’s Topics

• High-Level Language Interface
  – Section 12.1: Introduction
  – Section 12.3: Examples using Visual C++

Why Link ASM and HLL Programs?

• Use high-level language for overall project development
  – Relieves programmer from low-level details

• Use assembly language code to:
  – Speed up critical sections of code
  – Access nonstandard hardware devices
  – Write platform-specific code
  – Extend the HLL's capabilities
General Conventions

• Considerations when calling assembly language procedures from high-level languages:
  – Both must use the same naming convention (rules regarding the naming of variables and procedures)
  – Both must use the same memory model
  – Both must use the same calling convention

Calling Convention

• Identifies specific registers that must be preserved by procedures
• How arguments are passed to procedures: in registers, on the stack, in shared memory?
• The order in which arguments are passed by calling programs to procedures
• Arguments passed by value or by reference?
• Who restores the stack pointer? Calling or called procedure?
• Determines how functions return values
.MODEL Directive

- See Section 8.4.1 (pp.247-248) for details.
- AddTwo( 5, 6 ) becomes:

  Push 6
  Push 5
  Call AddTwo

  Push 6
  Push 5
  Call AddTwo
  Add ESP, 8

  STDCALL convention

  C convention

Passing Parameters by Reference

- How to pass an array?
- Can the called procedure change the parameter?
- Answer: store the address, not the value of your variables.
External Identifiers

• An external identifier is a name that has been placed in a module’s object file in such a way that the linker can make the name available to other program modules.
• The linker resolves references to external identifiers, but can only do so if the same naming convention is used in all program modules.

How to Link?

• Resolving the naming issues by EXTERN and PUBLIC
• Using stack for passing parameters.
• Follow the C naming and calling conventions
  – Add underscore (_) prefix to procedure names.
  – Parameters are pushed from right to left to the stack.
Linking Assembly Language to C++

- Basic Structure - Two Modules
  - The first module, written in assembly language, contains the external procedure
  - The second module contains the C/C++ code that starts and ends the program
- The C++ module adds the extern qualifier to the external assembly language function prototype.
- The "C" specifier must be included to prevent name decoration by the C++ compiler:

```
extern "C"
functionName( parameterList );
```

Name Decoration
(to answer why extern “C”)

Also known as name mangling. C++ compilers do this to uniquely identify overloaded functions. A function such as:

```
int ArraySum( int * p, int count )
```

would be exported as a decorated name that encodes the return type, function name, and parameter types. For example:

```
int_ArraySum_pInt_int
```

The problem with name decoration is that the C++ compiler assumes that your assembly language function's name is decorated.
FindArray Example

A C++ function that searches for the first matching integer in an array. The function returns true if the integer is found, and false if not:

```c
#include "findarr.h"

bool FindArray(long searchVal, long array[], long count )
{
    for(int i = 0; i < count; i++)
        if( searchVal == array[i] )
            return true;
    return false;
}
```

Generating ASM Code from C Programs

- See textbook 12.3.1 (page 410, Table 12-1) for the Visual C++ command-line options for ASM code generation.
- The `/FAs` option is used to generate the code in the next slide. You may need to add this to the Visual C++ Project Options string manually.
Code Produced by C++ Compiler

optimization switch turned off

```c
_searchVal$ = 8
_array$ = 12
_count$ = 16
_i$ = -4

_FindArray PROC NEAR
; 29 : {
    push ebp
    mov ebp, esp
    push ecx
    ; 30 : for(int i = 0; i < count; i++)
    mov DWORD PTR _i$[ebp], 0
    jmp SHORT $L174
; See the handout for the rest
```

Hand-Coded Assembly Language

(1 of 2)

```asm
true = 1
false = 0

; Stack parameters:
srchVal   equ [ebp+08]
arrayPtr  equ [ebp+12]
count     equ [ebp+16]
_.code
_AsmFindArray PROC near
    push ebp
    mov ebp,esp
    push edi
    mov eax, srchVal ; search value
    mov ecx, count ; number of items
    mov edi, arrayPtr ; pointer to array
```
Hand-Coded Assembly Language (2 of 2)

```
repne scasd          ; do the search
jz    returnTrue     ; ZF = 1 if found

returnFalse:
  mov   al, false
  jmp   short exit

returnTrue:
  mov   al, true

exit:
  pop   edi
  pop   ebp
  ret

_AsmFindArray ENDP
```

- If you’re using the 5th ed. of Irvine’s book and Visual Studio 2005, then the FindArray example is in: C:\Irvine\examples\ch12\VisualCPP
- If you’re using the 4th ed. or Visual C++ 6.0, then see the next two slides.
Creating the FindArray Project
(Using Microsoft Visual Studio 6.0)

• Run Visual C++ and create a project named FindArray.
• Add a CPP source file to the project named main.cpp. This file should contain the C++ main() function that calls FindArray.
• Add a new header file named FindArr.h to the project. This file contains the function prototype for FindArray.
• Create a file named Scasd.asm and place it in the project directory. This file contains the source code for the FindArray procedure.
• Use ML.EXE /c /Cx /coff to assemble the Scasd.asm file, producing Scasd.obj. Do not try to link the program.
• Insert Scasd.obj into your C++ project. (Select Add Files... from the Project menu.)
• Build and run the project.

A Visual C++ Workspace is already in: \Examples\ch12\VisualCPP\VisualCPP.dsw
Open the above in Visual C++ and try out the FindArray Project there.
   – You may need to copy the folder \Examples\ch12\VisualCPP to your hard disk.
   – Remember to use “ML /c /Cx /coff” to assemble scasd.asm into scasd.obj first.