Floating Point and Calling Assembly from C Code

Lecturers: ARD, SWJ Lab Tutors: VDE, WUL

Floating Point

Floating-point constants are expressed in the form: sign, digits, a period, then optionally more digits, then optionally an 'E' followed by an exponent. In NASM, the period is mandatory so the assembler can distinguish a data value as floating-point and not an integer. Optional underscores are used to break up large numbers.

Example of floating-point constants:

```
-0.5                          ; BYTE decimal value
1.2                           ; WORD decimal value
1.315_426_537                 ; DWORD decimal
1.e10                         ; QWORD 10,000,000,000.0
3.141_592_653_589_793_384_62 ; 80-bit TWORD floating-point representation of pi
1.0e+3500                     ; 80-bit double-extended QWORD
```

NOTE: NASM requires a period in floating-point values so the assembler can distinguish floating-point numbers from integers.

Basic Floating-Point Arithmetic Instructions.

FCHS	Change sign
FADD	Add source to destination
FSUB	Subtract source from destination
FSUBR	Subtract destination from source
FMUL	Multiply source by destination
FDIV	Divide destination by source
FDIVR	Divide source by destination

Calling Assembly Code from C

Programmers who create device drivers and code for embedded systems must often integrate C/C++ modules with specialized code written in assembly language. Assembly language is particularly good at direct hardware access, bit mapping, and low-level access to registers and CPU status flags. It would be tedious to write an entire application in assembly language, but it can be useful to write the main application in C/C++ and use assembly language to write only the code that would otherwise be awkward to write in C. Let's discuss some of the standard requirements for calling assembly language routines from 32-bit C/C++ programs.

add.asm

```
; make the add function visible to the linker
global _add

; prototype: int __cdecl add(int a, int B)
; desc: adds two integers and returns the result
_add:
        mov eax, [esp+4]        ; get the 2nd parameter off the stack
        mov edx, [esp+8]        ; get the 1st parameter off the stack
        add eax, edx            ; add the parameters, return value in eax
        ret                     ; return from sub-routine
```

The C code will simply be a basic main function that calls our add function In a C program, use the **extern** qualifier when declaring an external assembly language procedure.

```
#include <stdio.h>
/*
 * declaring add as extern tells the compiler that the definition
 * can be found in a seperate module
 */
extern int add(int a, int B);
int main() {
        int ret = add(10, 20);
        printf("add returned %d¥n", ret);
        return 0;
}
```

TASK

1. Write a C program to calculate the area of a flat surface using floating-point numbers. Enter three sets of floating-point numbers for the width and length.
2. Write an assembly language module embedded in a C program that calculates the area of a circle from a radius that is entered from the keyboard. Enter both integer radii and noninteger radii. Display the resulting areas.
3. Write an assembly language module embedded in a C program to find the average of five floating-point numbers that are entered from the keyboard. Enter two sets of numbers. Display the sum and the average