CMPSCI 201 – Fall 2005

Midterm #2 Part #2 – November 9, 2005

Professor William T. Verts

Open Book, Open Notes!

This part of the exam is taken-home and is due in class on Monday, November 14 (which has Friday's schedule), when you will take part 1 of this exam. Over the long weekend you may discuss the questions with the other students or work alone, as you choose, but *your answers must be unique and in your own words*.

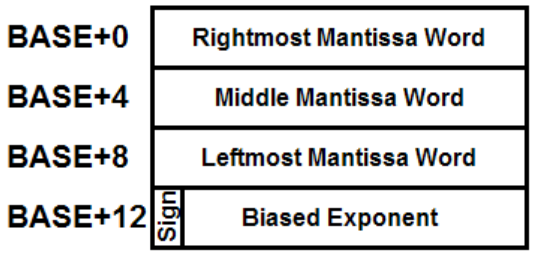
You are allowed to type in your answers into a word processor such as Microsoft Word, but if you do please follow these guidelines:

1. Put your name in the header/footer region so it appears on every page,
2. Put automatic page numbers in the header/footer as well,
3. Write your answers in the same order as the questions in this document,
4. Write all text in 12-point Times New Roman, except:
5. Use the `Courier New` typeface for any ARM code,
6. Staple your printout to this exam before you turn it in.

NAME	

PROBLEM	SCORE	POSSIBLE
9		20
10		10
TOTAL		30

<9> 20 Points – In this question we are inventing a new floating-point format that follows many of the basic ideas behind the IEEE-754 rules. In this format, numbers are 128 bits in length, and occupy four successive 32-bit integer words of storage (little endian). The most significant word contains the sign bit and 31 bits of biased exponent; the other three words contain the 96-bit mantissa. This layout is shown below:



The range of numbers is thus between $\pm 1.0 \times 2^{-e}$ and $\pm 1.FFFF\ldots FFF \times 2^{+e}$ (ignoring infinity, NaNs, and denormals of the form $\pm 0.xxxxx\ldots xxx \times 2^{-e}$), for some value of e.

A. (2 points) What is the decimal value of the ***bias***?

B. (2 points) What is the decimal range of valid ***exponent values***?

C. (2 points) Approximately how many decimal ***digits of precision*** can be represented with this floating-point format?

D. (2 points) What is the ***hexadecimal*** value of the floating-point number -12.125 in this format? Show breaks between the four words of your answer as follows:

xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx
Base+12 Base+8 Base+4 Base+0

E. (12 points) Write a chunk of ARM code using the model shown (where symbol BASE is defined as the address of the first of the four successive words of memory) to compute an estimate of the ***square root*** of the stored number. Use only integer registers `R0` through `R11` in your answer; no floating point registers. You may assume that the sign bit is 0 to avoid issues with imaginary numbers. To compute your estimate you must divide the *signed exponent* by 2 as we have done in previous examples (discarding any remainder from the division), but this time you must also divide the *mantissa* by 2 instead of simply setting it to zero. Thus, for non-zero binary floating-point values of the form $1.abcd\ldots xyz \times 2^{e}$, the approximation we want is $1.0abcd\ldots wxy \times 2^{e/2}$. (Hint: you may have to do some thinking and/or research about shifting and rotating values in ways we haven't discussed in class, as well as for the generation of some of the constants used by your routine.)

<10> 10 Points – Essay answer – Here are three ways of handling stack frames for subroutines. The three subroutines have identical purposes and internal functions, except for the *offsets* used in referencing their parameters and local variables. In all three cases there are parameters pushed onto the stack before the call, and inside each subroutine the registers `R0`, `R1`, `R2`, `R3`, `IP`, and `LR` are pushed on the stack, as well as three words of storage reserved for local variables (represented by the `SUB SP,SP,#12` instruction). Your answer to this problem doesn't require knowing exactly how many parameters were pushed onto the stack before each subroutine call, but the number is greater than zero and it is the same number of parameters in each of the three cases. Discuss the ***advantages and disadvantages*** of each of the three strategies shown below.

```
SUBROUTINE1
     STMDB SP!,{R0-R3,IP,LR}
     SUB   SP,SP,#12
     MOV   IP,SP
     …
     …     ; do useful work
     …
     ADD   SP,SP,#12
     LDMIA SP!,{R0-R3,IP,PC}


SUBROUTINE2
     STMDB SP!,{R0-R3,IP,LR}
     MOV   IP,SP
     SUB   SP,SP,#12
     …
     …     ; do useful work
     …
     ADD   SP,SP,#12
     LDMIA SP!,{R0-R3,IP,PC}


SUBROUTINE3
     STMDB SP!,{IP,LR}
     MOV   IP,SP
     SUB   SP,SP,#12
     STMDB SP!,{R0-R3}
     …
     …     ; do useful work
     …
     LDMIA SP!,{R0-R3}
     ADD   SP,SP,#12
     LDMIA SP!,{IP,PC}
```