CSCE 212: Exam 1 **Spring 2008**

Name (please print):_________________________________ **Total points: ___/210**

Instructions
This is a CLOSED BOOK and CLOSED NOTES exam. However, you may use calculators, scratch paper, and the green MIPS reference card from your textbook. Ask the instructor if you have any questions. Good luck!

1. (Chapter 2, 5 points)

Complete the following table by filling in the empty cells to connect MIPS instruction types with their corresponding encoding format.

Instruction type	Encoding format
Load and store instructions	
Branch instructions	
Arithmetic instructions having 2 register input operands	
Shift instructions	
	J-type

2. (Chapter 2/3, 15 points)

When executing the **`ADDI`** instruction, the MIPS architecture **sign-extends** its 16-bit immediate field before adding it to the contents of register **rs**. Assume for this question that the hardware designers saved costs by **always zero-extending** the immediate value instead. As a result, any **`ADDI`** instruction with a negative immediate value is treated as a pseudoinstruction by the assembler. When the pseudoinstruction is assembled, it is converted into a series of instructions that perform a software-based sign extension of the immediate. Assuming there have been no other architectural changes, how is the following pseudoinstruction translated:

`ADDI $s0, $s1, -12`

Hint: the first instruction in the translation moves the 16-bit immediate into temporary register $1, which (in this case) results in register **$1** having value **0000 FFF4**. Three more instructions are needed:

`1. ADDI $1, $0, -12`

`2.` ______________________

`3.` ______________________

`4.` ______________________

3. (Chapter 3, 10 points)

Write a sequence of MIPS assembly code that will detect overflow for **unsigned subtraction**. **Hint:** overflow occurs when the operation's result falls outside the valid range.

4. (Chapter 3, 10 points)

Convert 8762 from base 10 to base 9.

5. (Chapter 3, 15 points)

Convert -987.667 to IEEE 754 single-precision floating point. Represent your answer in hexadecimal.

6. (Chapter 2, 30 points)

Write a MIPS assembly-language subroutine having the following requirements. The subroutine takes two arguments, in \$a0 and \$a1, which hold the base memory addresses of two equal-sized arrays, and a third argument in \$a2 that holds the arrays' lengths. The subroutine's function is to copy the contents of the first array into the second array but in reverse order. This subroutine also must preserve the values of **all** the caller's registers.

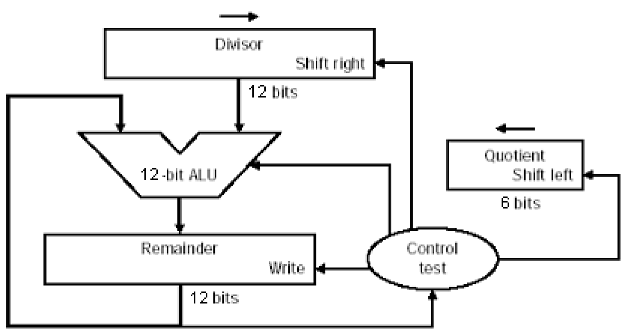
7. (Chapter 2/3, 10 points)

Describe what computation the following segment of assembly code is performing.

```
            mtc1 $0, $0
loop:       lwc1 $f1, 0($a0)
            lwc1 $f2, 0($a1)
            mul.s $f3, $f1, $f2
            add.s $f0, $f0, $f3
            addi $a0, $a0, 4
            addi $a2, $a2, -1
            bnez $a2, loop
```

8. (Chapter 3, 15 points)

Assume the following divider design:



Assume the divisor and remainder registers are 12 bits and the quotient register is 6 bits. Give the values of each register for each step of the divide operation. Use decimal or binary representation. Assume the divider is dividing **40** by **12** (all register values are unsigned).

Remainder Register	Divisor Register	Quotient Register

9. (Chapter 2, 20 points)

Translate the following high-level code to MIPS assembly:

```
for (i=0;i<n;i++) a[i]=i;
```

Before your translation begins, assume variable **n** is stored in register **$s0** and that array **a** is an array of **half-words**.

10. (Chapter 2, 15 points)

Describe three different functions of the immediate field in the MIPS instruction set.

11. (Chapter 2/3, 25 points)

Assume there is a MIPS pseudoinstruction named **`ADD64 rd, rs, rt`** that performs a 64-bit integer addition.

The instruction **`ADD64 $2, $4, $6`** combines registers $4 and $5 to form one 64-bit operand (i.e. $4 is high-order 32 bits and $5 is low-order 32 bits) and combines registers $6 and $7 as the other 64-bit operand. The result would be stored in a 64-bit register spanning register $2 and $3. Translate this pseudocode instruction.

Hint: Think of how you perform binary addition on paper.

12. (Chapter 3, 10 points)

Would the following operation cause an overflow, assuming the result is stored in a 11-bit signed integer register? Why or why not?

$$-567 + (-458)$$

13. (Chapter 1, 10 points)

Assume two processors, a CISC processor and a RISC processor, having different ISAs. In order to run a particular program, the CISC processor must execute 10 million instructions and requires an average of 12 ns per instruction. To run the same program, the RISC processor must execute 25 million instructions. How much faster must the RISC processor execute each instruction, on average, to execute the program in the same amount of time as the CISC processor?

14. (Chapter 2, 10 points)

Describe the difference between a subroutine call and a system call.

15. (Chapter 2/3, 40 points) **BONUS QUESTION**

Many CISC instruction set architectures, such as Intel IA-32 (x86) and IBM 360, include a set of instructions that do computation using a datatype called binary-coded decimal (BCD). BCD stores integer values by assigning binary-encoded decimal values (0-9) into 4-bit fields. Eight of these 4-bit fields can be *packed* into a 32-bit word, representing a value between 0 and 99,999,999. For example, the (16-bit) value 0011 1001 0001 0101 represents 3,915 (instead of 14,613 as it would as a regular base-2 integer).

Assume we have a pseudoinstruction called **`CBCD rd, rs`** that converts the binary value in register **rs** to an **8-digit** packed BCD value written to register **rd**. *Show* (for full credit) or *describe* (for partial credit) how **`CBCD $2, $3`** could be translated into MIPS instructions. **Hint:** how would you perform this base conversion yourself?

16. (Chapter 2, 20 points) **BONUS QUESTION**

Write a sequence of MIPS assembly instructions that will swap the values of registers $s0 and $s1 **without using any additional registers or any loads and stores**.

Hint: this requires that you use the XOR instruction.