ECE/CS 552: Introduction to Computer Architecture
ASSIGNMENT #1
Due Date: At the beginning of lecture, September 22nd, 2010

This homework is to be done individually. Total 9 Questions, 100 points

Make sure your homework contains your full name and student ID, and that all pages are stapled together.

1. (8 pts.) Fill in the empty slots in the table by converting numbers into different representations. All numbers are 8 bits in length. Use 2's compliment representation in binary and hexadecimal numbers.

Binary	Hexadecimal	Unsigned Decimal	Signed Decimal
0010 0001		33	
	6F		
1010 0100			
			-25

2. (6 pts.) Translate MIPS assembly language into machine language and specify each operation in the empty slots of "Comments"

Assembly Language	Machine Language	Comments
add $t0, $s1, $s2		# $t0 = $s1 + $s2
sll $t2, $s7, 4		#
sw $t4, 0 ($t2)		#
j 0x092A		#

3. (6 pts.) In this exercise, you will evaluate the performance difference between two CPU architectures, CSIC (complex instruction set computing) and RISC (reduced instruction set computing). Generally speaking, CISC CPUs have more complex instructions than RISC CPUs and therefore need fewer instructions to perform the same tasks. However, typically one CISC instruction, since it is more complex, takes more time to complete than a RISC instruction. Assume that a certain task needs P CISC instructions and 2P RISC instructions, and that one CISC instruction takes 8T ns to complete, and one RISC instruction takes 2T ns. Under this assumption, which one has the better performance?

4. (15 pts.) Consider the following fragment of C code:

```
for ( i=0; i<=100; i=i+1)
    a [i] = b[i] + c;
```

Assume that a and b are arrays of words and that the base address of a is in $a0 and the base address of b is in $a1. Register $t0 is associated with variable I and register $s0 with the value of c. You may also assume that any address constants you need are available to be loaded from memory.

(1) (10 pts.) Write the code for MIPS.
(2) (5 pts.) How many instructions are executed during the running of this code if there are no array out-of-bounds exceptions thrown?
(3) How many memory data references will be made during execution?

5. (16 pts.) Pseudoinstructions are not part of the MIPS instruction set but often appear in MIPS programs. For each pseudoinstruction in the following table, produce a minimal sequence of actual MIPS instructions to accomplish the same thing. You may need to use $at for some of the sequences. In the following table, "imm" refers to a specific number that requires 32 bits to represent.

Pseoduinstructions	What it accomplishes
move $t1, $t2	$t1 = $t2
clear $t0	$t0 = 0
beq $t1, imm, L	If ($t1 == big) go to L
bge $t5, $t3, L	If ($t5 >= $t3) go to L

6. (10 pts.) Consider two different implementations, P1 and P2, of the same instruction set. There are five classes of instructions (A, B, C, D, and E) in the instruction set.
P1 has a clock rate of 4 GHz. P2 has a clock rate of 6 GHz. The average number of cycles for each instruction class for P1 and P2 is as follows:

Class	CPI on P1	CPI on P2
A	1	2
B	2	2
C	3	2
D	4	4
E	3	4

(1) (5 pts.) Assume that peak performance is defined as the fastest rate that a computer can execute any instruction sequence. What are the peak performances of P1 and P2 expressed in instruction per second?
(2) (5 pts.) If the number of instructions executed in a certain program is divided equally among the classes of instructions except for class A, which occurs twice as often as each of the others, how much faster is P2 than P1?

7. (14 pts.)

(1) (6 pts.) Assume the following instruction mix for a MIPS-like RISC instruction set: 20% stores, 20% loads, 20% branches, and 25% integer arithmetic, 7% integer shift, and 8% integer multiply. Given a program with 200 instructions and that load instructions require two cycles, branches require 4 cycles, integer ALU and store

instructions require one cycle and integer multiplies require 10 cycles, compute the overall cycles-per-instruction or CPI.

(2) (8 pts.) Strength reduction is common compiler optimization that converts multiplies by a constant integer into a sequence of shift and add instructions with equivalent semantics. Given the same parameters of problem 7(1), consider such a strength-reducing optimization that converts multiplies by a compile-time constant into a sequence of shifts and adds. For this instruction mix, 65% of the multiplies can be converted into shift-add sequences with an average length of 3.5 instructions. Assuming a fixed frequency, compute the new CPI and overall program speedup.

8. (15 pts.)

(1) (10 pts.) You are going to enhance a computer, and there are two possible improvements: either make multiply instructions run four times faster than before, or make memory access instructions run two times faster than before. You repeatedly run a program that takes 100 seconds to execute. Of this time, 20% is used for multiplication, 50% for memory access instructions, and 30% for other tasks. What will the speedup be if you improve only multiplication? What will the speedup be if you improve only memory access? What will the speedup be if both improvements are made?

(2) (5 pts.) You are going to change the program described in (1) so that the percentages are not 20%, 50%, and 30% anymore. Assuming that none of the new percentages is 0, what sort of program would result in a tie (with regard to speedup) between the two individual improvements? Provide both a formula and some examples.

9. (10 pts.) The following code fragment processes two arrays and produces an important value in register $v0. Assume that each array consist of 2500 words indexed 0 through 2499, that the base addresses of the array are stored in $a0 and $a1, respectively, and their size (2500) are stored in $a2 and $a3, respectively. Add comments to the code and describe in one sentence what this code does. Specifically, what will be returned in $v0?

```
        sll     $a2, $a2, 2
        sll     $a3, $a3, 2
        add     $v0, $zero, $zero
        add     $t0, $zero, $zero
outer:  add     $t4, $a0, $t0
        lw      $t4, 0 ($t4)
        add     $t1, $zero, $zero
inner:  add     $t3, $a1, $t1
        lw      $t3, 0($t3)
        bne     $t3, $t4, skip
```

```
        addi    $v0, $v0, 1
skip:   addi    $t1, $t1, 4
        bne     $t1, $a3, inner
        addi    $t0, $t0, 4
        bne     $t0, $a2, outer
```