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

1. (8 pts.) Fill in the empty slots in the table by converting numbers into different representations. All numbers are 8 bits in length.

Binary	Hexadecimal	Unsigned Decimal	Signed Decimal
0010 0001	21	33	33
0100 1111	6F	111	111
1010 0100	A6	166	-90
1110 0111	E7	231	-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	000000 10001 10010 01000 00000 100000	# $t0 = $s1 + $s2
sll $t2, $s7, 4	000000 00000 10111 01010 00010 000000	# $t2 = $s7 << 4
sw $t4, 0 ($t2)	101011 01010 01100 0000000000000000	# store the value in $t4 to memory address ($t2+0)
j 0x092A	000010 0000000000 0000 1001 0010 1010	# PC = {(PC+4)[31:28], 16'h092A, 2'b00}

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?

Solution: Compare the runtime of the task:

CISC: P * 8T = 8PT

RISC: 2P * 2T = 4PT

Therefore in this case, RISC 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?

Solution:

(1) One example of MIPS code:

```
        clear $t0;
        addi $s0, $zero, 100
loop:   lw $t1, 0($a1)          # $t1 = b[i]
        add $t1, $t1, $s0       # $t1 = a[i]
        sw $t1, 0($a0)          # store $t1 to address of a[i]
        addi $a0, $a0, 4        # $a0 = address of a[i+1]
        addi $a1, $a1, 4        # $a0 = address of a[i+1]
        addi $t0, $t0, 1        # $t0 = $t0 + 1
        beq $t0, $s0, finish        # if ($t0 = 100) finish
        j loop
finish:
```

(Different program with same behavior can get full grade, too.)

(2) 2 instructions before loop are executed 1 time; 7 instructions between loop and beq are executed 101 times; instruction “j loop” executed 100 times. Therefore:

Total instructions executed (in this case): 2*1 + 7*101 + 1*100 = 809.

(3) Memory data reference: 101 * 2 = 202.

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 == imm) go to L
bge $t5, $t3, L	If ($t5 >= $t3) go to L

Solution: (examples, not the only solution)

move $t1, $t2:
    add $t1, $t2, $zero
clear $t0:
    and $t0, $zero, $zero
beq $t1, imm, L:
    lui $at, imm [31:16]
    addi $at, $at, imm[15:0] / ori $at, $at, imm[15:0]
    beq $t1, $at, L
bge $t5, $t3, L:
    sub $at, $t5, $t3
    and $at, $at, 0x8000 (or -32768 in decimal)
    beq $at, $zero, L

(Or students can use slt to achieve same functionality. Slightly longer sequence of instructions is fine, too)
(The students cannot use other temporary registers except $at)

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?

Solution:
(1) For both P1 and P2, the maximum IPC are 1 and 0.5, respectively.
IPS = IPC * # of cycles in 1s
Therefore IPS for P1 = 1 * 4G = 4e9;
IPS for P2 = 0.5 * 6G = 3e9;

(2) Average CPI = (2A+B+C+D+E) / (2+1+1+1+1). Here A, B, C, D, E refers to the CPI of each instruction on one of the implementation.
On P1, CPI = (2+2+3+4+3)/6 = 14/6;
Avg. time to execute 1 instruction: (14/6)/4GHz
On P2, CPI = (4+2+2+4+4)/6 = 16/6;
Avg. time to execute 1 instruction: (16/6)/6GHz

(Speed of P2) / (Speed of P1)
= (Avg. time per instruction on P1) / (Avg. time per instruction on P2)
= 21/16 or 1.31
Therefore P2 is 1.31 times faster 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 (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.

Solution:

(1) Overall CPI = 200*(20%*1 + 20%*2 + 20%*4 + 25%*1 + 7%*1 + 8%*10) / 200
= 2.52

(2) Old # of multiplies: 8% * 200 = 16

Now, 65% of 16 multiplies are replaced by 3.5 ALU instructions each.

Therefore, the new # of multiplies is 16 * 35% = 5.6 (on average)

The additional ALU instructions # is: 16 * 65% * 3.5 = 36.4

Now the total # of instructions is: 200-16+5.6+36.4= 226

New CPI = [200 * (20%*1 + 20%*2 + 20%*4 + 25%*1 + 7%*1) + (5.6 * 10 + 36.4 *1)] / 226

= 1.93

Speedup = Old CPU Time / New CPU Time

= 504 / 436.4

= 1.15

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.

Solution:

(1) The times used for multiply and memory access are 20s and 50s, respectively.

Accelerate only multiply: (100-20)+20/4 = 85 sec;

Accelerate only memory access: (100-50)+50/2 = 75 sec;

Accelerate multiply and memory access: (100-20-50) +20/4+50/2 = 50 sec.

(2) Suppose the percentage of time used by multiply is a%, the percentage of memory access is b%. To make a tie between two individual improvements, we will have

$(100-a) + a/4 = (100-b) + b/2$

$\Rightarrow \quad b = 1.5a \quad (0<a, b<100)$

For example, if multiply consumes 20% of time, then only when memory references instruction consumes 30% of time will the improvements have equal effect.

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         # $a2 and $ a3 are shift left by 4 so the -
        sll     $a3, $a3, 2         # - memory references will be aligned by word
        add     $v0, $zero, $zero   # clear $v0
        add     $t0, $zero, $zero   # clear $t0. $t0 is the word # of array 1
outer:  add     $t4, $a0, $t0       # compute the address of current word
        lw      $t4, 0 ($t4)        # load value of the word in array 1
        add     $t1, $zero, $zero   # clear $t1. $t1 is the word # of array 2
inner:  add     $t3, $a1, $t1       # compute the address of current word
        lw      $t3, 0($t3)         # load value of the word in array 2
        bne     $t3, $t4, skip      # if the two words are equal, -
        addi    $v0, $v0, 1         # - increment $v0; otherwise skip.
skip:   addi    $t1, $t1, 4         # move to next word in array 2
        bne     $t1, $a3, inner     # if the boundary of array 2 is not reached, -
                                    # continue the inner loop
        addi    $t0, $t0, 4         # move to next word in array 1
        bne     $t0, $a2, outer     # if the boundary of array 2 is not reached, -
                                    # continue the outer loop
```

Solution:

Comments are added on the right of the program. This code iteratively compares the two arrays and returns the number of same words which appear in both arrays. This value is stored in $v0.