Computer Architecture **Name:** ____________________

Midterm 1, Spring, 2013

Show Your Work! Point values are in square brackets. There are 30 points possible.

Unless a problem states otherwise, you can use any MIPS instructions, including pseudoinstructions. Be sure to document your code.

1. Consider the following data on two systems, S1 and S2, when they run a program P with input I.

System	Clock Rate	Instruction Count	CPU Time
S1	4 GHz	20×10^9	5 sec
S2	3 GHz	30×10^9	15 sec

(a) Find the CPI for S1

(b) Find the CPI for S2

(c) If we can increase the clock rate for S2, execute the same number of instructions, and maintain the same CPI, what clock rate would we need in order to have the same CPU Time as S1?

(d) If we can reduce the number of instructions for S2, and maintain the same clock rate and CPI, how many instructions should we execute in order to have the same CPU Time as S1?

[4 points]

(a) Since

$$\text{CPU Time } = \frac{\text{(Inst Count)} \times \text{CPI}}{\text{Clock Rate}},$$

we have that

$$5 = \frac{20 \times 10^9 \times \text{CPI}}{4 \times 10^9}$$

Solving for CPI gives a CPI of 1.

(b) A similar approach gives a CPI of 1.5.

(c) Now we want

$$5 = \frac{30 \times 10^9 \times 1.5}{\text{Clock Rate}}.$$

Solving for the clock rate gives 9 GHz.

(d) Now we want

$$5 = \frac{\text{(Inst Count)} \times 1.5}{3 \times 10^9}.$$

Solving for the instruction count gives 10×10^9.

2. Bob runs a program on his desktop and collects the following data.

	Compute	Load	Store	Branch
Instruction Count	9000	5000	1000	2000
CPI	1	3	2	2

Find the average CPI. (A common fraction is fine.) [2]

$$\text{Average CPI} = \frac{9000 \times 1 + 5000 \times 3 + 1000 \times 2 + 2000 \times 2}{9000 + 5000 + 1000 + 2000} = \frac{30,000}{17,000} = 30/17 \approx 1.76.$$

3. Bob has found that 90% of the run-time of program P with input I is spent in the function `Boola_boola`. So Bob has worked on `Boola_boola` and found a way to reduce its run-time by a factor of 100. What will the speedup be when Bob replaces the original function with the improved function? (A common fraction is fine.) [3]

   Suppose that T_o is the original run-time. Then $0.9T_o$ is the amount of time spent in `Boola_boola`, and $0.1T_o$ is the amount of time spent in the rest of the program. So if T_n is the run-time of the program after improvement, we have

$$\begin{aligned}\text{Speedup} &= \frac{T_o}{T_n} \\ &= \frac{T_o}{0.9T_o/100 + 0.1T_o} \\ &= \frac{1}{9/1000 + 100/1000} \\ &= \frac{1000}{109} \\ &\approx 9.17\end{aligned}$$

4. Find the decimal representation of the signed integer that has the hexadecimal two's complement representation 0xa4b. [2]

   a = 10 = 1010_2 and b = 11 = 1011_2. So

$$\text{0xa4b} = 1010\ 0100\ 1011_2.$$

   This is negative, and the corresponding positive value is

$$0101\ 1011\ 0100_2 + 1 = 0101\ 1011\ 0101_2 = 2^{10} + 2^8 + 2^7 + 2^5 + 2^4 + 2^2 + 1 = 1461.$$

   So the value of 0xa4b is -1461.

5. Find the eight-bit two's complement binary representation of the signed decimal integer -53. [2]

   We can use the remainder-division algorithm to find the binary representation of 53:

53 % 2 = 1	53/2 = 26
26 % 2 = 0	23/2 = 13
13 % 2 = 1	13/2 = 6
6 % 2 = 0	6/2 = 3
3 % 2 = 1	3/2 = 1
1 % 2 = 1	1/2 = 0

   So $53_{10} = 110101_2$, and the 8-bit two's complement representation of -53 is

$$1100\ 1010_2 + 1 = 1100\ 1011_2.$$

6. Find the hexadecimal representation of the following MIPS instruction after it's been translated into machine language.

```
sll $s0, $s0, 2
```

[3]

The instruction `sll` uses R-format. So it's opcode is 0. From the Green Sheet, we see that it's funct field is also 0. The register `$s0` has number 16, and the `rs` field isn't used. So the decimal and binary representations of the machine instruction are

	Opcode	rs	rt	rd	shamt	funct
Decimal	0	0	16	16	2	0
Binary	000000	00000	10000	10000	00010	000000

If we group the bits into sequences of four bits, we can get the hex representation of the machine instruction:

0000 0000 0001 0000 1000 0000 1000 0000 = 0x00108080.

7. Write MIPS assembly code that stores the hex value 0x8903a4ab in the register `$t0`. (Your answer should work in both Mars and Spim.) [2]

```
lui     $t0, 0x8903
ori     $t0, $t0, 0xa4ab
```

8. Consider the following code from a MIPS program.

```
      addi  $t1, $0, $0     # i = 0
      addi  $s2, $0, $0     # result = 0
loop: lw    $s1, 0($s0)     # $s1 = array[i]
      add   $s2, $s2, $s1   # result += array[i]
      addi  $s0, $s0, 4     # $s0 += 4:  reference next int in array
      addi  $t1, $t1, 1     # i++
      slt   $t2, $t1, $t3   # if (i < n) $t2 = 1
                            #    else $t2 = 0
      bne   $t2, $0, loop   # if ($t2 != 0) go to loop
                            #    else go to next instruction
```

This is a translation of code from a C program in which C variables correspond to the following MIPS registers

C variable	`i`	`result`	`n`	`array`
MIPS register	`$t1`	`$s2`	`$t3`	`$s0`

At the beginning of the code segment `$s0` stores the *address* of `array[0]`. You should assume that $n > 0$.

Write the C-code from which this was translated. [4]

This is just adding the elements in `array`:

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

However, the C for-loop checks whether $i < n$ *before* each iteration, and the assembler checks it *after* each iteration. As long as n is positive, this won't make any difference.

In case n might be zero, you would probably need to implement the C code with a do-while loop or one iteration and then a while loop. Something like this would do

```
i = 0;
result = 0;
do {
   result += array[i];
   i++;
} while (i < n);
```

Of course, if n can be zero, both the assembly and this C code may have a bug: they access array[0].

Finally, note that the first two `addi's` should either be changed to `add's` or the second `$0` in each instruction should be just `0`.

9. In the following C function the array b stores n ints, and every element of b is a valid subscript of some element of a. Implement this function as a MIPS assembly language function.

```
int Indirect_sum(int a[], int b[], int n) {
   int i, sum = 0;

   for (i = 0; i < n; i++)
      sum += a[b[i]];
   return sum;
}  /* Indirect_sum */
```

[4]

```
indsum: li   $t1, 0           # $t1 = sum = 0
        li   $t0, 0           # $t0 = i = 0

lptst:  bge  $t0, $a2, done   # if (i > n) go to done
        sll  $t2, $t0, 2      # $t2 is giving the byte offset into b
        add  $t2, $t2, $a1    # $t2 has the absolute address of b[i]
        lw   $t3, 0($t2)      # $t3 = b[i]
        sll  $t3, $t3, 2      # $t3 is giving the byte offset into a
        add  $t3, $t3, $a0    # $t3 has the absolute address of a[b[i]]
        lw   $t3, 0($t3)      # $t3 = a[b[i]]
        add  $t1, $t1, $t3    # sum += a[b[i]]
        addi $t0, $t0, 1      # i++
        j    lptst

done:   move $v0, $t1
        jr   $ra
```

10. Translate the following C function into MIPS assembly language.

```
int Do_stuff(int n) {
   int p,q;

   if (n <= 0)
      return 3;
   else if (n <= 2)
      return 5;
   else {
      p = Do_stuff(n - 2);
      q = Do_stuff(n - 4);
      return p+q;
   }
}  /* Do_stuff */
```

[4]

```
dostff: addi $sp, $sp, -12
        sw   $ra, 8($sp)
        sw   $s0, 4($sp)
        sw   $s1, 0($sp)

        bgt  $a0, $zero, gt0    # if (n = $a0 > 0) go to gt0
        li   $v0, 3
        j    done

gt0:    li   $t0, 2
        bgt  $a0, $t0, gt2      # if (n = $a0 > 2) go to gt2
        li   $v0, 5
        j    done

gt2:    move $s0, $a0           # save n = $a0 = $s0
        addi $a0, $a0, -2       # n = n - 2
        jal  dostff             # Do_stuff(n-2)
        move $s1, $v0           # save $s1 = Do_stuff(n-2)
        addi $a0, $s0, -4       # n = n - 4
        jal  dostff             # Do_stuff(n-4)
        add  $v0, $v0, $s1      # Return value = Do_stuff(n-4) + Do_stuff(n-2)

done:   lw   $ra, 8($sp)
        lw   $s0, 4($sp)
        lw   $s1, 0($sp)
        addi $sp, $sp, 12
        jr   $ra
```