CS 351 Exam 1, Fall 2012

Your name: _____

Rules

- You may use one handwritten 8.5 x 11" cheat sheet (front and back). This is the only resource you may consult during this exam.
- Include explanations and comments in your answers in order to maximize your partial credit. However, you will be penalized for giving extraneous incorrect information.
- You may use the backs of these pages if you need more space, but make it clear where to find your answer to each question.
- Unless otherwise specified, you do not need to work out the arithmetic on math problems. Just do enough algebra to set up an answer of the form: Answer = [arithmetic expression] [units]

Grade (instructor use only)

	Your Score	**Max Score**
Problem 1		25
Problem 2		25
Problem 3		10
Problem 4		10
Problem 5		15
Problem 6		15
Total		100

Problem 1: 25 points.

Answer the following questions. Be sure to explain your answers if you want to receive partial credit.

---

a) How many bits are in a byte? How many bytes are in a gigabyte? For the latter question, you can just set up the arithmetic – you don't need to compute the final numeric answer.

b) Here are some examples of instructions in the x86 instruction set that have no equivalent in MIPS. Knowing what you know about how MIPS instructions are encoded, explain why you think these instructions do not exist:

Instruction 1:

```
INC $reg                # add 1 to the value in the given register
```

Instruction 2:

```
LEA $reg1, [$reg2 + 8*reg3 + 4]
# Calculate the value of $reg2 + 8*reg3 + 4 and store it in $reg1
# Note that the 8 and the 4 are immediate operands
```

c) Critique the following arguments/taunts. The more clear and specific your critique, the more points you will receive:

Argument 1: “My smartphone uses way less power than your supercomputer, so it is obviously more energy-efficient in all cases! Why do you hate the Earth?”

Argument 2: “My processor has a higher MIPS (millions of instructions per second) rating than yours for my workload, so it’s obviously much faster!”

Problem 2: 25 points.

You are running a program on your computer. Some facts:

- There are 500 billion dynamic instructions in the program's execution.
- 20% of the instructions are floating-point instructions, 50% are integer instructions, and 30% are memory instructions.
- The average CPI of a floating-point instruction is 2, the average CPI of an integer instruction is 1, and the average CPI of a memory instruction is 3.
- Your processor's frequency is 2.2 GHz.
- The workload cannot be parallelized.

STATE THE UNITS FOR ALL OF YOUR ANSWERS FOR THIS PROBLEM!!

What is the execution time of your workload? (You should set up the arithmetic, but you don't need to fully carry it out.) Don't forget to specify units.

Using a compiler optimization, you are able to remove 10% of the memory instructions. What is the speedup of your new version over the old version?

How much power does your original processor consume if it has a capacitive load of 10 nF, an operating voltage of 2 V, and a static leakage power that is 35% of the overall power consumption? Don't forget to specify units.

How much energy does your processor consume for the optimized version of this workload? (You will not be penalized for mistakes made in earlier parts of this problem, but make clear where your expression is coming from.) Don't forget to specify units.

What is your processor's energy-delay product for this workload? (You will not be penalized for mistakes made in earlier parts of this problem, but make clear where your expression is coming from.) Don't forget to specify units.

Problem 3: MIPS if/else (10 points)

Translate the following C code to MIPS. Obey all MIPS conventions about register and stack usage. Assume that the C variable *a* is already in $s0, *b* is in $s1, and *c* is in $s2.

```
if (a < 50)
     b += 3;
else if (a > 25)
     c += 3;
```

Problem 4: Short MIPS functions (10 points)

Translate the following C code to MIPS. Obey all MIPS conventions about register and stack usage.

```
int f(int x) { return x % 16; }
```

Problem 5: MIPS array code and function calls (15 points)

Translate the following C code to MIPS. Obey all MIPS conventions about register and stack usage.

Assume that the C standard library has been included, and the rand() function being called is the standard one.

You will need to allocate the array *a* on the stack yourself. Assume that we will want the pointer *a* in $t0 at the end of the program.

```
int a[3];
a[0] = rand();
a[1] = rand();
a[2] = a[0] + a[1];
```

Problem 6: Defining and calling MIPS functions (15 points)

Translate the following C code to MIPS. Obey all MIPS conventions about register and stack usage.

```
int f(char* x, char y) {
    int i=0;
    while (*x)
        if (*x == y)
            i++;
            x++;
    return i;
}
```
