CS 351 Exam 1, Fall 2011

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		30
Problem 2		20
Problem 3		10
Problem 4		10
Problem 5		15
Problem 6		15
Total		100

Problem 1: 30 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 BITS are in a kilobyte? For the latter question, you can just set up the arithmetic – you don't need to compute the final numeric answer.

b) Your friends are writing a MIPS compiler, and they are annoyed about having to save registers like $ra on the stack before calling a function. Explain what is wrong with each of their alternative proposals:

"Let's just call the function and leave $ra alone. So the function overwrites it. What's the big deal?"

"OK, let's copy $ra into $s0 before calling the function. That should be safe."

"All right, smarty-pants, but what about copying $ra into $t0? That's cool, right?"

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

Argument 1: “My computer uses less power than yours! Therefore, it’s more energy-efficient to operate it. Don’t you wish you were as environmentally responsible as I am?”

Argument 2: “My MIPS processor has a faster clock speed than your x86 processor! It will run my programs so much faster, leaving me with extra time to hang around and taunt you!”

d) Because of your awesome performance in CS 351, you land a highly lucrative job designing the next MIPS ISA. For this ISA version, they decide to use variable-width instructions instead of fixed-width 32-bit instructions. You can now make any instruction use more or fewer bits of machine code. You will still have only 32 registers to work with.

We will assume (unrealistically) that your instruction width does not have to be a multiple of 8 bits.

List at least one instruction that you would redesign to use FEWER than 32 bits, and explain:

List at least one instruction that you would redesign to use MORE than 32 bits, and explain:

Problem 2: 20 points.

You are running one of the SPEC CPU programs on your computer. Some facts:

- There are 1 trillion dynamic instructions in the program's execution. The average CPI of these instructions is 2.0.
- Your processor has 2 cores. Each of your cores has a frequency of 2.5 GHz.
- The workload is not parallelized at all.

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.

If you invested in a new processor with cores that run at 3.0 GHz, what would be the speedup of your new processor over your old one for this workload? See the above note about arithmetic. Don't forget to specify units.

How much power does your original processor consume if it has a capacitive load *C*, an operating voltage of 1.5 V, and a static leakage power that is 1/3 of the overall power consumption? Don't forget to specify units.

How much energy does your old processor consume for this workload? 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 += c;
else
     b -= c;
```

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 64 * x; }
```

Problem 5: MIPS array code (15 points)

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

You will need to allocate the array *a* yourself. Assume that it is OK to put the loop counter in $s0.

```
char a[25];
for (int i=23; i >= 0; i--) {
     a[i] = 'a' + i // fun with ASCII math
                    // note: ASCII 'a' = 97 in decimal
}
a[24] = 0;
```

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.

Assume that the function *g* obeys all MIPS conventions and is defined to take a single integer as its input parameter and return an integer.

```
int f(int* x) {
     return g(*x) + *x;
}
```