# Homework Assignment 1
**CIS501 Fall 2005**

**Due:** Tuesday, September 27$^{th}$ at noon.

**Instructions:** Your submission for this assignment must be type-written, except for your answers to Question 4, parts (a) and (b), which must be completed on the enclosed worksheet. To submit the assignment, you will hand in a print out of your answers, along with the worksheet at the start of class on the 27$^{th}$.

**Question 1:** Processor X has a clock speed of 1 GHz, and takes 1 cycle for integer operations, 2 cycles for memory operations, and 4 cycles for floating point operations. Empirical data shows that programs run on Processor X typically are composed of 35% floating point operations, 30% memory operations, and 35% integer operations. You are designing Processor Y, an improvement on Processor X which will run the same programs and you have 2 options to improve the performance:

1. Increase the clock speed to 1.2 GHz, but memory operations take 3 cycles
2. Decrease the clock speed to 900 MHz, but floating point operations only take 3 cycles

Compute the speedup for both options and decide the option Processor Y should take.

**Question 2:** You have learned about the difference between RISC and CISC ISAs, as well as many of the advantages of a RISC ISA for actual hardware. Consider for a moment, a virtual machine– a piece of software that decodes and executes instructions (for example, a Java virtual machine executes Java instructions in Java bytecode). Suppose that you were asked to design a new ISA which will only be implemented in a vitual machine, never in hardware. Analyze the benefits and disadvantages of RISC vs CISC features for this new ISA. Some of the features you should include in your analysis are:

- Load/store vs other operand models

- Simple instructions "small" instructions vs instructions which perform complex tasks
- Fixed Length instructions vs Variable length instructions

**Question 3:** Consider the expressions:

```
D =  A * B - C;
E =  (D + B) * A * B;
```

Where variables A, B, and C are initially in memory and the destinations of D and E are also in memory.

1. Write down pseudo-assembly code for this sequence for machines using for both "load-store" and "memory-only" operand models. Try to optimize the code.
2. Two evaluation criteria for code are:

   (i). static code size, and

   (ii). data memory traffic (i.e., traffic to and from memory).

   Assume that all data values are 4 bytes, opcodes are 1 byte, addresses are 2 bytes, and register specifiers are 1 nibble (half a byte). Also assume that instructions must be integral numbers of bytes. What is the static code size and data memory traffic associated for both implementations?

**Question 4:** For this question, you will be using the SimpleScalar simulator (see below) on two benchmarks- twolf (from SpecInt) and art (from SpecFP).

1. Run the functional simulator on each of the programs. Fill in the table on the worksheet with the count and percentage of instructions that are loads, stores, control instructions, system calls, integer operations, and floating point operations.
2. In the Alpha ISA, load and store instructions may contain a 16 bit offset field, which is added to the register operand to compute the address. Modify the simulator code so that whenever such an instruction executes, it computes how many bits of the offset field were actually used, and maintains a count of the number

of instructions utilizing each bit-length, then outputs the counts with the simulation statistics. Fill in the table on the worksheet with the percentage of loads and stores which have each length offset. Then compute the cumulative percentage of all loads and stores which use an offset of bit length less than or equal to a given length, and fill in the appropriate entries. Some entries are filled in to help you check your work. **N**OTE: you should modify the simulator to gather an array of statistics, and run it once.

3. Suppose that the immediate field was reduced from 16 bits to 8 bits. Assume that

   - Loads and stores take 2 cycles to execute and all other instructions take 1 cycle.
   - An instruction which has too long of an offset must be fixed by inserting one add instruction before it.

   How does this change affect performace (i.e. what is the speedup or slowdown) for twolf and art?

4. Suppose instead that the immediate field is removed completely, however, loads and store can complete in one cycle (and an additional add instruction must be inserted where needed). What speedup does this give for twolf and art?

5. Is part the scenario of part (d) always a good idea? If not, explain at least one effect that would occur in a real processor which it overlooks.

## Additional information for Question 4:

In order to obtain the source code for SimpleScalar for this homework, copy the contents of `~cis501/SimpleScalar/hwk1/` to your home directory, your computer, or wherever you plan to edit the code. You should be able to build the simulator simply by running "make". The SimpleScalar distribution for this homework is stripped down to contain only the functional simulator `sim-func`. A functional simulation means that only the effects of the instructions are simulated, not timing or power information.

SimpleScalar does not run the operating system, instead it reads a file which contains the program, as well as the results of system calls (i.e. the inputs of the program). The two programs you will be experimenting with in this assignment are found in the "programs" directory. In order to run the functional simulator on twolf, for example, you would do:

```
./sim-func programs/twolf.eio
```

Note that on eniac-l, twolf takes about 45 seconds, and art takes about 4 minutes to run.

For part (b) of Question 4, you are asked to modify the code for the simulator. In order to simplify this task, we have indicated where you will need to modify the simulator. If you look in sim-func.c, you will find 3 comments stating `/* YOUR CODE GOES HERE */`. One of these is found in the function `sim_aux_stats`, which prints the simulation statistics. This is where you should insert your code to print out the counts you gather. To accomplish this task, you should use the `print_counter` function which takes 4 arguments- the stream to print to, the name of the statistic being printed, the value, and a description of the statistic.

The second `/* YOUR CODE GOES HERE */` comment is found inside the `sim_sample_on` function- the main loop of the simulator which executes the instructions. We have provided code which tests if the current instruction is a load or store, and if it has an immediate offset. Additionally, we have provided code to extract that immediate offset into the variable "immediate" as a signed integer.

The final `/* YOUR CODE GOES HERE */` comment is found in `sim_start`, which is called once before the simulation starts. If your solution needs any one time initalization code, it should be placed here. These comments are only hints, not requirements. If you want to write a solution which ignores these hints, you are free to do so.

Two final warnings:

1. When counting things, it is advisable to use the datatype `counter_t` instead of `int`, as programs may execute more than 2 billion instructions.

2. Remember that for part (b), you are looking at a signed offset. Be careful about this when determining how many bits are needed to represent a number. For example, the number 1 requires two bits to represent as a signed number (i.e. 01) while -1 can be represented by 1 bit (with 1 bit, you can represent -1 and 0).

# Question 4 (a) and (b) Worksheet

**Homework Assignment 1**
**CIS501 Fall 2005**

**Name:**

**Part (a):**

| **Insn type** | **Count in twolf** | **% in twolf** | **Count in art** | **% in art** |
|---|---|---|---|---|
| Loads | | | | |
| Stores | | | | |
| Control Ops | | | | |
| Sys calls | | | | |
| Int Ops | | | | |
| Fp Ops | | | | |

**Part (b):**

| **Bit length** | **% in twolf** | **Cmltv % in twolf** | **% in art** | **Cmltv % in art** |
|---|---|---|---|---|
| 0 | 38.81 | 38.81 | | |
| 1 | 1.74 | 40.56 | | |
| 2 | 1.04 | 41.60 | | |
| 3 | 0.20 | 41.80 | | |
| 4 | | | | |
| 5 | | | | |
| 6 | | | | |
| 7 | | | | |
| 8 | | | | |
| 9 | | | | |
| 10 | | | | |
| 11 | | | | |
| 12 | | | | |
| 13 | | | | |
| 14 | | | | |
| 15 | | | | |
| 16 | 16.9 | 100 | | 100 |