Assignment One (30 points)

There are four parts to this assignment. Each is required. Each part also requires some preparatory work. You will be ready to do one part each of the first four weeks of class, as indicated.

Remember, only a few parts of this assignment may be graded, and those parts will determine your entire grade, so do your best on each part.

Part One (10 pts) You can do this after Week 1.

In this part you will simply do some problems from the book. You must show your work to get credit. Submit them with the rest of the assignment. From Chapter 1 of the text:

1. Problem 1.4. What is the minimum amount of RAM required for two framebuffers sufficient in size to store a 1280x1024 frame using 24-bit color (8 bits for each color with RGB color). (Two framebuffers are often used so that one buffer can be filled while the other is displaying.) Express your answers in MiB (1024x1024 bytes). Express the time to transfer a single frame over a 100Mbit/sec network in seconds. (Be careful: don't confuse bits with bytes.)

2. Problems 1.10.1-1.10.3 (using information in 1.10)

3. Problem 1.15

   It was not clear to me what the question wanted, so here is a sample calculation for 2 processors:

<table>
<thead>
<tr>
<th>p (number of cores)</th>
<th>execution time</th>
<th>speedup</th>
<th>% of ideal speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>54</td>
<td>100/54 = 1.85</td>
<td>1.85/2 = 92.5%</td>
</tr>
</tbody>
</table>

   finish filling in the table for 4, 8, 16, 32, 64 and 128 processors

4. You have a task that is 70% parallelizable. It executes in 10 sec on one CPU. How much time does it take to execute on 2, 4 and 8 CPUs?

5. For the integer representation section, do this problem: You are given an eight-bit binary (2's complement) value of 11111110 What is its value in the following representations.
   - hexadecimal
   - signed decimal
   - unsigned decimal
   - base 5 (assume the binary value is interpreted as unsigned)

   If the value is assumed to be signed and is converted to 32 bits, what is its hexadecimal value after conversion?

Part Two (5pts) You can do this after Week 2.

Before doing this part, read the handout entitled SimpleMachine.

In the SimpleMachine handout, the instructions for the Simple Machine were detailed, as well as the specification for our register transfer language (rtl). Part of the algorithm necessary for simulating (or microprogramming) the operation of our Simple Machine using rtl was also provided. In this part, you will finish this algorithm by adding the clauses necessary for each of the missing instructions. You should refer to the handout SimpleMachineDatapaths for details on which registers can be transferred to which other registers and in what directions (although you should only have to use this for checking your work). Also remember that each instruction cycle is limited to a single instruction memory cycle and a single data memory cycle.

Hand in a listing of your pseudocode using rtl, with sufficient documentation (using standard C or C++-style comments) to indicate which instructions are which. Place the code to process the instructions in the instructions' numeric order so that it is easy to follow. (i.e., the code for HALT first, then for LOAD, etc.) You can check your work by looking at the code for the real simulator. See the file

asmt01
sim/simulate.c. (You will also have to refer to the sim.h and inst.h header files in the sim directory.)

**Part Three (10 pts)** You can do this after Week 3.

Before doing this part, you should experiment with the provided test programs for the sim simulator. You should also implement the sample programs described in Exercises-SimpleMachine.

After implementing the three programs in the exercises you should see that there are some kinds of things that easy to program on the Simple Machine and some that are quite difficult. In this part you will see this more clearly by coding a simple problem using this processor.

In Exercises-SimpleMachine, you finished writing the program addloop. You saw that if you kept the loop counter in the CTR register and went through the loop backwards, the Simple Machine worked very well. One problem with this machine, however, is that the CTR register is used for all comparisons. Thus, if you have even a single if statement inside your loop, you are unable to keep the loop counter in CTR. This will happen with this homework problem.

The problem is simple: given an array of N integers, count the number of zero elements. There are two possible algorithms for this problem provided on hills in the asmt01 directory. They are in countzeroes.c and countzeroesdec.c. They differ in whether the loop is traversed normally or by decrementing the counter.

Your solution should, of course, work for any size array of (positive or negative) numbers. **This means if I modify your array to add or delete items I should not have to change your code. If I change the value of N (the number of items in the array) and the array items it should work correctly.**

You may use temporary variables to implement this program. You should begin by converting the C code to our ugly-C form, which uses gotos and labels. Do not try to code the program directly.

Transfer a copy of your program as a plain text file on hills using the procedure discussed in AssignmentGuidelines. The name of the program must be countzeroes (note: there is no extension!)

**Part Four (5 pts)** You can do this after Week 4.

In this part you will finish writing a MIPS function. Even though we have not yet written functions, your job will be to take an existing main program that calls your function, and the skeleton of the function (the entry and exit code), fill in the function while following certain restrictions, then place the value in a specific register before the [existing] function return code. The main program will verify whether your function performed correctly, and you will be able to alter main's variables to test other instructions.

The name of the function is encodertype, and its purpose will be to encode an R-type instruction. The function is called with the pieces of the instruction like this

```c
unsigned encodertype(unsigned opc, unsigned rs, unsigned rt, unsigned rd, unsigned shamt, unsigned funct);
```

Note: while the opc value of an R-type instruction is normally 0, certain other values are possible.

The actual code is quite simple if you did the Lab on Bit operations and the Exercises-Bitops. The really important part of this assignment is for you to take a good first look at what a function call looks like, and to see the first example (in main()) of the use of the stack.

The starting code is in the asmt01 directory on hills. Its name is encodertype.s. You simply set reasonable values for the instruction fields using the named labels. The values currently in the code are for the instruction

```c
srl $a0,$t9,21
```

After calling your function, the main program outputs each field of the instruction in a message so you can check it.

Transfer a copy of your program as a plain text file on hills using the procedure discussed in AssignmentGuidelines. The name of the program must be encodertype.s