ECE 350 Project

Guidelines

1) There is a choice between taking the second midterm exam and doing the project.
2) You will need to make this choice within few days of announcement of the project. Unless you explicitly declare that you will be doing the project, we will assume that you will take the second midterm exam. This is necessary to have a focused a direction in the course. We need your decision by Oct 17th.
3) If we do not have enough students signing up for the project, we will have to cancel it.
4) Your final choice needs to be given to the instructor in writing.
5) Project Due Date: November 26th 2007.
6) The project is supposed to be a single student effort.

Project Overview: In this project you will implement a simpler version of the MIPS architecture that we study in class using the favorite high level language of your choice (preferably C). The computer should have the capability to support the following instructions

R Type: Add, Sub, And, Or, Xor, Nor, sll, srl, slt
I Type: Lw, Sw, Addi, Andi, Ori, Xori, Beq, Bne
J Type: Jump

The Modules that you need to simulate are
1) 32 Bit register file with 32 Registers (as discussed in class), Note that some of the registers we may not use since we don't support all the instructions
2) Program Counter that points at the current instruction that is being executed
3) Two Memory Modules, one for instructions and one for data. The memory modules are byte addressable but the word size if 4 bytes.
4) An ALU capable of performing the above instructions
5) The addressing for branch and jump is PC Relative.

INPUT for Processor Simulator: You will be given assembly code along with the corresponding addresses in the instruction memory. We will assume that all instructions sequences start with an initial PC value of 0.

Output: You will need to step through the instructions and print the status of the register file/data memory etc., along with the current instruction being executed.

After the completion of the processor simulator, we will extend the project for the development of a simple compiler as well. The details of this would be provided later.

Project Details:

You will need to implement the following instructions

R Type: Add, Sub, And, Or, Xor, Nor, sll, srl, slt
I Type: Lw, Sw, Addi, Andi, Ori, Xori, Beq, Bne
J Type: Jump

You need to Incorporate the following modules

Program Counter PC: A register that holds the address of the current instruction. Since the addresses are always integers, one could simply represent the PC as an integer variable.

Register File: There are 32 registers of 32 bits each. The addressing of the registers is exactly as MIPS although in your code you can assume that each register is essentially a variable that stores integer values.

ALU: Should be able to deal with all the instruction's computation needs.

Instruction Memory: Stores all the instructions and is byte addressable. Can store upto 100,000 bytes

Data Memory: MIPS data memory is byte addressable. Your memory should be byte addressable too. You can assume that we can store maximum of 100,000 bytes.

INPUT OUTPUT FORMATS:

We will provide a few test assembly programs. The final project will be tested on our own assembly code which you will execute on your implementation of the MIPS architecture. The assembly code will essentially form an initialization of the instruction memory. All new assembly codes will start with the instruction memory address 0. That is, instruction memory address 0 has the first instruction to be executed. We will also provide an initialization of the data memory and register file. Hence there will be three input files that we will provide. You will execute the sequence of instructions and print the status of the data memory and register file after each instruction. NOTE THAT THIS PROJECT IS NOT ABOUT BULDING AN ASSEMBLY CODE SIMULATOR. YOU ARE EXPECTED TO BUILD THE MIPS ARCHITECTURE WITH EXPLICIT FUNCTIONS/PROCEDURES FOR REGISTER FILE, ALU etc.

Instruction Memory File format:
We will provide an initialization of the instruction memory which essentially corresponds to the set of instructions that need to be executed and their corresponding addresses. The first instruction always starts at address 0. The file format is as follows

```
Instruction0
Instruction1
…
…
.end
```

It can be assumed that Instruction0 is the first instruction to be executed whose address is 0. Instruction1 is the next one with address 4 and so on. For specifying the instructions themselves, we will be following the standard MIPS conventions with a bit of care. Most notably, your program should expect commas between registers, etc. Registers will be referenced in the Instruction Memory by the Register Name rather than the Register Number. Therefore, each $_ can replaced by $zero, $v0-1, $a0-3, $t0-7, $s0-7, etc. There are 4 basic formats:

(1) R-type $_, $_, $_
where R-type is any R-type instruction

(2) I-type $_, $_, IMM
where I-type is any I-type instruction excluding lw and sw
and IMM is a signed integer

(3) lw $_, immediate ($_)
sw $_, immediate ($_)

(4) J IMM
where IMM is a signed integer

Initialization of Data Memory and the Register File:

You will need to initialize some parts of data memory and the register file before running your project. We will distribute several test programs along with some initialization values for you to test and debug your program

Data Memory input file format:

The format of this input file is shown below. Your program should parse this file and load the value on the right into the byte address of data memory specified on the left. This will only have to be done for several locations and then the file will end with a

“.end.”   Once again, “.end” functions as a sentinel value in parsing the file. The data memory initialization input file will look like this:

```
BYTE_ADDRESS_1, VALUE_1
BYTE_ADDRESS_2, VALUE_2
.
.
.
BYTE_ADDRESS_n, VALUE_n
.end
```

Obviously, 0 <= BYTE_ADDRESS < 100,000 and 0 <= VALUE < 256

Register File file input file format:
This will only have to be done for a handful of registers as well. Follows a similar format as data memory:

```
REGISTER_1, VALUE_1
REGISTER_2, VALUE_2
.
.
.
REGISTER_n, VALUE_n


.end
```

Output Format –

We will need to know whether your processor is executing the code (Instruction Memory) correctly after each instruction. Therefore, the entire register file ($0-31) and a range of the data memory should be printed to standard output or a file after each instruction. The range of memory to be printed will be requested from the user by the program before code execution. For example, a program with 2 instructions total would produced output like this:

```
What is the range of memory you would like to see? 99 105

Instruction 1 executed…
$zero = 0
$1 = 0
….
$31 = 0

M[99] = 0
M[100] = 0
…
M[105] = 0

Instruction 2 executed….
$zero = 0
$1 = 0
….
$31 = 0

M[99] = 0
M[100] = 0
…
M[105] = 0
```