Lab 5

CSE141L
Announcements

• New lab due date policy
  – Assigned Friday
  – Due the next Friday or
  – Due Sunday night for 10% off.

• Applies to current Lab 4, and the Lab 5.
Clarification About Memories

- Instruction memory in fetch unit:
  - 17 bits wide
  - Accessible via special load/store instructions

- Data memory in back end:
  - 34 bits wide
  - Accessible via normal load/store instructions

• This is a Harvard Architecture
  – Easier for the lab, not how real machines work.
Lab 5

- Assembler
- Simulator
Assembler 1/2

• Manually writing machine code
  – Time consuming
  – Error prone
  – Monotonous
  – Hard to maintain
  – ....(bad bad bad bad)

Why not Automate?
Assembler 2/2

• Inputs
  – A program source written in your own assembly language
  – Instruction start address (entry point)
  – Data start address

• Outputs
  – $name_i.coe -- instructions
  – $name_d.coe – data
• Why coe?
  – Initialize vector format in Xilinx CoreGen
  – Your RAM modules will be initialized as specified in a *.coe file
The assembly process

The assembler should fill in these constants.
Testing the Assembler

• Simple test cases
  – Check output by hand.
• Lots of asserts() and error checking
• The simulator will help
Simulator

• Why?
  – verify your ISA
  – debug your applications
  – improve your ISA by spotting performance bottlenecks in benchmark programs
  – Collect data about how your processor performs -- what actually is the common case?

• Inputs
  – $name_i.coe
  – $name_d.coe

• Your simulator will have a command line interface for easy debugging
Simulator Commands

- `load $i_file.coe $d_file.coe`
  - load `*.coe` files into instruction memory and data memory

- `go $number`
  - simulate next `$number` instructions

- `dump_reg`
  - print values in all the registers

- `set_reg $reg_num $value`
  - set the register `$reg_num` with the value `$value`

- `dump_imem $addr $size`
  - print instruction memory values from `$addr` to `$addr + $size`

- `set_imem $addr $value`
  - set the value at `$addr` of the instruction memory with the value `$value`

- `dump_imem $addr $size`
  - print data memory values from `$addr` to `$addr + $size`

- `set_imem $addr $value`
  - set the value at `$addr` of the data memory with the value `$value`

- `instr_count`
  - show the number of executed instructions
Simulator Algorithm

Allocate and initialize architectural state

While (1) {
    Setup;
    Read Instruction;
    Decode Instruction;
    Execute Instruction;
    Do command line interaction;
}

Things to consider

• Both your assembler and simulator need to understand your ISA
  – Best practice: Use a *single* representation for the ISA, so the assembler and simulator cannot get out of sync.

• There are many sources of error in your system:
  – Conceptual errors in the ISA
  – Bugs in your implementation of your ISA
  – Bugs in the simulator
  – Bugs in the assembler
  – Bugs and conceptual errors in your assembly program
  – Debug carefully, and do lot of error checking!!!! Detect errors early.
  – E.g. Make sure all unused bits are zero.
  – E.g. Fill all of memory a recognizable value or non-executable instruction
Notes

• You can use any language you like (even different ones for the assembler and simulator, if you like; -- Can you still use a single representation of your ISA)?

• You can share code for bit (un)packing and 34-bit/17-bit stuff, but nothing else. You must credit your source.
  – We have provided some java code for this purpose.