CS 21 Project: MIPS Single Cycle Processor Extension

Department of Computer Science
College of Engineering
University of the Philippines - Diliman

1 Introduction

In the first arc of the laboratory exercises we saw the expressive power of assembly language by creating simple programs. A program of moderate complexity was built for the Machine Problem. During the second arc of the laboratory exercises, we incrementally built a processor design in HDL (more specifically, SystemVerilog).

As mentioned in Laboratory Exercise 12, the given MIPS single cycle processor design can only execute a *subset* of the MIPS instructions set. For your project, you would be *extending* the single cycle MIPS processor design so that it can execute more types of instructions.

The instructions that you will add to the repertoire of instructions that the processor can execute fall into three categories

1. **Normal instructions** - these are instructions that *are* in the instruction set, but just not currently implemented in the design given in Laboratory Exercise 12
2. **Pseudo-instructions** - these are instructions that are *not* in the instruction set, but are recognized by the assembler and are converted or assembled as a sequence of actual instructions; these can also be found in the MIPS Green Sheet
3. **Custom instructions** - these are instructions that are not in the MIPS Green Sheet

2 Instructions to be added

The following subsections will enumerate the instructions that would have to be added to the single cycle MIPS processor executable repertoire, as well as how many points they are worth (note that the project will be scored over 100 points).

For *each* modification or added instruction, explain in *detail*, in the documentation -

- What changed or was added in the HDL code (which modules, etc.) - *line-by-line* explanation
- Schematic diagram of what changed in the processor (this need not be of the entire processor)
- Explanation on how you tested that the instruction works, or was successfully added to the processor - describe and discuss testbench and assembly language program used, etc.

Please be very detailed regarding these explanations. We gave 0 points for some students in previous versions of this project for having insufficient or unclear explanations.

2.1 Normal instructions

1. `sll` (*15 points*) - assume unused fields (e.g., rs) are Xs (Don't cares - not necessarily zeroes)
2. `sb` (*15 points*)

2.2 Pseudo-instructions

1. `ble` - same format as beq, but opcode is now $1F_{hex}$ (*20 points*)
2. `li` (*20 points*)
    - opcode is 0x11
    - target register is in bits 20-16 (rt)
    - rs (bits 25-21) are Xs (Don't cares - not necessarily zeroes)
    - immediate field (bits 15-0) contain values to be stored to rt; when used in assembly language, assume value to be stored always no longer than 16 bits; logically- or zero-extend the 16-bit value to 32 when storing to register

2.3 Custom instruction: `Zero-from-right (zfr)`

2.3.1 Instruction details

R-type (so opcode is 00_{hex}) instruction which performs the *zero-from-right (zfr)* operation on register `rs`, based on the value of the rightmost 5 bits in `rt` and stores the result in register `rd`. (*30 points*)

Format -

- Bits 31-26 - `opcode` = 00_{hex}
- Bits 25-21 - `rs`
- Bits 20-16 - `rt`
- Bits 15-11 - `rd`
- Bits 10-6 - Xs (Don't cares - not necessarily zeroes)
- Bits 5-0 - `funct` = 33_{hex}

As for the operation -

- Take the value R[`rt`[4:0]]
- Assuming that the bits of the value in `rs` are numbered 31 to 0 (left to right) zero all bits from bit 0 to bit R[`rt`[4:0]]
- Store the result in `rd`

2.3.2 Circuit diagram

For the documentation, indicate and explain (both in detail) *clearly* how the datapath will be modified to implement `zfr`. We expect a detailed circuit diagram - a box with the label "zfr" will *not* suffice. We expect a diagram with multiplexers, gates, etc. (*that* level of detail).

2.3.3 Examples

- Example 1
  - rs: 0xFFFF FFFF
  - rt: 0x0000 0005
  - rd after execution: 0xFFFF FFC0
- Example 2
  - rs: 0xC0DE FFFF
  - rt: 0x0000 0005
  - rd after execution: 0xC0DE FFC0
- Example 3
  - rs: 0xC0DE FFFF
  - rt: 0xBABE 0005
  - rd after execution: 0xC0DE FFC0
- Example 4
  - rs: 0xFFFF FFFF
  - rt: 0x0000 0000
  - rd after execution: 0xFFFF FFFE
- Example 5
  - rs: 0xFFFF FFFF
  - rt: 0xFFFF FFFF
  - rd after execution: 0x0000 0000

3 Checking

- What will be submitted are the following
  1. Documentation in PDF, with filename `lastname_firstname_SN_section.pdf`. For example, *Watanabe_Mayu_201814321_MTHAKB.pdf*.
  2. Video documentation (in MP4) of you explaining, *per instruction* -
     - HDL edits
     - Testbench
     - Test code (in assembly and in machine code) - pay attention to instruction fields that must be Don't-cares (not necessarily zeroes)
     - Demonstration that processor can now successfully execute the instruction required

     Basically, video or mini-lecture version of the documentation.
  3. Zip file containing all HDL sources - mips module, and dependencies - no need to include imem, dmem, top, or the testbench.

     Filename is `lastname_firstname_SN_section.zip`.

     For example, *Watanabe_Mayu_201814321_MTHAKB.zip*.

- You can only modify the module `mips`, and everything below it hierarchically
- No or insufficient explanation in the documentation and video for an added instruction, *no points*
- If there is any discrepancy between what is in the docu, what is in the video, and what is in the code, you will get *no points* for that instruction
- If the added instruction does not work (as determined by our testbench), `no points` - again, pay attention to fields that must be Don't-cares
- Note that we expect you to extend the capability of the single cycle MIPS processor design given in Laboratory Exercise 12, and *not break anything* in the process. Before we check the individual added instructions, we will be running an integrity check testbench first, which would check whether the original instructions in the repertoire can still be executed correctly. If your processor design will fail this test, you *immediately get 0* for the project.
- Submitted works by students will be exhaustively compared to each other (all-against-all comparison) - students suspected of plagiarism/intellectual dishonesty will *not* be given opportunities to retract their submissions - student disciplinary cases will be filed against them (no middle ground or compromise). So work on this project independently.

4 Deadline

See UVLe.