Introduction to Computer Architecture
Project 2

MIPS Simulator (Part 1)

Hyungmin Cho
Department of Software
Sungkyunkwan University

Project 2 Overview

- Implement the 1st part of basic MIPS instruction simulator
  - In Project 1, your program just interpreted the binary instructions into assembly format, but in Project 2, your program actually “executes” each instruction

- In Project 2, you’ll implement a subset of MIPS instructions, and in the next project, you need to finish the remaining instructions

- You need to expand Project 1’s command interface to support additional commands for this simulator.

- The basic rules (submission rule, etc…) are the same as Project 1, but please ask TAs if anything is unclear.

Instructions to Support (1)

- R-format instructions: `add, addu, and, nor, or, slt, sltu, sub, subu, xor`
- Shift instructions: `sll, sllv, sra, srav, srl, srlv,`
- I-format instructions: `addi, addiu, andi, lui, ori, slti, sltiu, xori`
- The other instructions (e.g., memory load/store, branch, jump) will be supported in the next project

Instructions to Support (2)

- No need to distinguish the “signed” / “unsigned” version of arithmetic instructions
  - e.g., “**add**” and “**addu**” behaves in the same way
  - In a real MIPS CPU, signed version of instructions (e.g., “**add**”) triggers an exception on arithmetic overflow while unsigned version of instructions don’t.
  - Since project 2 does not implement an exception mechanism, no need to distinguish “signed” and “unsigned” version of arithmetic operations
- Need to distinguish slt / sltu and slti / sltiu

Instructions to Support (2)

- No need to distinguish the “signed” / “unsigned” version of **arithmetic instructions (**`add/addu, addi/addiu, sub/subu`)
  - e.g., “**add**” and “**addu**” behaves in the same way
  - In a real MIPS CPU, signed version of instructions (e.g., “**add**”) triggers an exception on arithmetic overflow while unsigned version of instructions don’t.
  - Since project 2 does not implement an exception mechanism, no need to distinguish “signed” and “unsigned” version of arithmetic operations

Instructions to Support (3)

- NEED to distinguish the “signed” / “unsigned” version of comparison instructions (`slt/sltu, slti/sltiu`)
  - For `slti/sltiu`, perform sign extension regardless of the instruction version, then compare in signed/unsigned format
  - e.g.,

```
addi $t0, $zero, -2  #$t0 = 0xFFFFFFFE, -2 if signed, 4294967294 if unsigned
addi $t1, $zero, 1   #$t1 = 0x1

slt $t2, $t0, $t1    #$t2 = ( -2 < 1 ) ? 1 : 0
sltu $3, $t0, $t1    #$t3 = (4294967294 < 1 ) ? 1 : 0

addi $t0, $zero, 3   #$t1 = 0x3

slti $t4, $t0, 2     #Imm 0x0002 → sign extended to 0x00000002 → $t4 = ( 3 < 2 ) ? 1 : 0
sltiu $t5, $t0, 2    #Imm 0x0002 → sign extended to 0x00000002 → $t4 = ( 3 < 2 ) ? 1 : 0

slti $t6, $t0, -2    #Imm 0xFFFE → sign extended to 0xFFFFFFFE → $t4 = ( 3 < -2 ) ? 1 : 0
sltiu $t7, $t0, -2   #Imm 0xFFFE → sign extended to 0xFFFFFFFE → $t4 = ( 3 < 4294967294 ) ? 1 : 0
```

Data Structures to Implement

- To implement the MIPS instruction simulator, your program would need to model the following data structures
  - Registers
    - General-purpose registers: `$0 - $31` (`$0` should always be zero)
    - `PC register`
    - All contents are initialized to 0x00000000 at beginning
  - Instruction memory
    - Address range: 0x00000000 – 0x00010000 (64KB)
    - All contents are initialized to 0xFF at beginning
    - You can assume the instruction memory is always used at 4byte boundary (e.g., CPU won't fetch an instruction from address 0x00000123)
  - You can use any data structure (it can be arrays, dictionaries, class object, or whatever data structure you want to use) to implement these components.

Requirement #1: Program Loading

- Add the following command to your program

  `loadinst <filename>`

  - Read the binary file named `<filename>` and store the binary instructions to the simulated instruction memory

  - The beginning of the binary file is starting of address 0x00000000

Requirement #2: Instruction Execution

- Add the following command to your program

  `run <N>`

  - Start from PC address of 0x00000000, simulate up to **`N`** instructions
  - As the result of execution, registers should be updated (update PC register as well)
  - If your MIPS CPU encounters an error (e.g., unknown instruction), stop execution
    - Print "`unknown instruction`" when you encounter an instruction not supported in this version
  - When finishing the execution, either by running all `N` instructions or by an error, print the number of executed instructions
    - If the execution is stopped due to an unknown instruction, include the unknown instruction in the number of executed instructions (See test sample 2)

```
mips-sim> run 10
Executed 10 instructions
mips-sim>
```

```
mips-sim> run 100
unknown instruction
Executed 34 instructions
mips-sim>
```

Requirement #3: Register Print

- Add the following command to your program

  `registers`

  - Print the current value of registers (`$0-$31` and `PC`)
  - Print values in hexadecimal format
    - "0x" prefix
    - All values should be printed in 8 digits
    - In other words, use "`0x%08x`" format string

```
mips-sim> registers
$0: 0x00000000
$1: 0x00000000
$2: 0x0000000a
$3: 0x00000014
$4: 0x10000004
$5: 0x1000002c
$6: 0x00000000
$7: 0x00000000
$8: 0x00000000
$9: 0x00000000
$10: 0x00000000
$11: 0x00000000
$12: 0x00000000
$13: 0x00000000
$14: 0x00000000
$15: 0x00000000
$16: 0x00000000
$17: 0x00000000
$18: 0x00000000
$19: 0x00000000
$20: 0x00000000
$21: 0x00000000
$22: 0x00000000
$23: 0x00000000
$24: 0x00000000
$25: 0x00000000
$26: 0x00000000
$27: 0x00000000
$28: 0x00000000
$29: 0x0fffffd0
$30: 0x0fffffd0
$31: 0x00000370
PC: 0x0000037c
mips-sim>
```

Test Sample (1)

- `~swe3005/2020s/proj2/proj2_1.bin`
- "proj2_1.bin" file represent the following assembly code.

```
addi $1, $0, 0x1234
```

- Therefore, its execution results should be as →
  - For your information, the red ones are indicating the changed registers.
  - In your implementation, you don't need to use colors.

```
mips-sim> loadinst proj2_1.bin
mips-sim> run 1
Executed 1 instructions
mips-sim> registers
$0: 0x00000000
$1: 0x00001234
$2: 0x00000000
$3: 0x00000000
$4: 0x00000000
$5: 0x00000000
$6: 0x00000000
$7: 0x00000000
$8: 0x00000000
$9: 0x00000000
$10: 0x00000000
$11: 0x00000000
$12: 0x00000000
$13: 0x00000000
$14: 0x00000000
$15: 0x00000000
$16: 0x00000000
$17: 0x00000000
$18: 0x00000000
$19: 0x00000000
$20: 0x00000000
$21: 0x00000000
$22: 0x00000000
$23: 0x00000000
$24: 0x00000000
$25: 0x00000000
$26: 0x00000000
$27: 0x00000000
$28: 0x00000000
$29: 0x00000000
$30: 0x00000000
$31: 0x00000000
PC: 0x00000004
mips-sim>
```

Test Sample (2)

- `~swe3005/2020s/proj2/proj2_2.bin`
- "proj2_2.bin" file represent the following assembly code.

```
lui $2, 0x8765
ori $2, $2, 0x4321
addi $3, $0, -1
srl $4, $3, 8
sll $5, $3, 8
xor $6, $4, $5
and $7, $2, $6
```

- Therefore, its execution results should be as →
  - After executing the last "and" instruction, the CPU moves to the next instruction
  - However, because we haven't loaded any instruction to that location, the instruction memory just returns the default value (i.e., 0xFFFFFFFF), which is an unknown instruction
  - Therefore, the execution should stop at the 8th instruction.

```
mips-sim> loadinst proj2_2.bin
mips-sim> run 10
unknown instruction
Executed 8 instructions
mips-sim> registers
$0: 0x00000000
$1: 0x00000000
$2: 0x87654321
$3: 0xffffffff
$4: 0x00ffffff
$5: 0xffffff00
$6: 0xff0000ff
$7: 0x87000021
$8: 0x00000000
$9: 0x00000000
$10: 0x00000000
$11: 0x00000000
$12: 0x00000000
$13: 0x00000000
$14: 0x00000000
$15: 0x00000000
$16: 0x00000000
$17: 0x00000000
$18: 0x00000000
$19: 0x00000000
$20: 0x00000000
$21: 0x00000000
$22: 0x00000000
$23: 0x00000000
$24: 0x00000000
$25: 0x00000000
$26: 0x00000000
$27: 0x00000000
$28: 0x00000000
$29: 0x00000000
$30: 0x00000000
$31: 0x00000000
PC: 0x00000020
mips-sim>
```

Submission

- Clear the build directory
  - Do not leave any executable or object file in the submission

- Use `submit` program
  - If you want to submit "`src`" directory…
    - `~swe3005/bin/submit` **`proj2`** `src`

```
Submitted Files for proj1:
File Name                                 File Size       Time
--------------------------------------------------------------------------
proj2-2019123456-Sep.05.17.22.388048074        268490                  Thu Sep  5 17:22:49 2019
```

- Verify the submission
  - `~swe3005/bin/check-submission` **`proj2`**

Project 2 Due Date

- 2020 Apr 29th, 23:59:59

- Late penalty (max 3 days): 20% per day