

# **Introduction to Computer Architecture**

## **Project 2**

### **Single-cycle MIPS CPU Simulator**

**Hyungmin Cho**  
Department of Software  
Sungkyunkwan University

# Project 2 Overview

---

- In Project 2, you'll implement an ***instruction simulator*** that supports a subset of MIPS instructions
  - ❖ What is an instruction simulator? Similar to the MARS simulator, your program reads and mimic its behavior.
  - ❖ We only consider the register values and data memory contents.
  - ❖ That is, at the end of the execution, your program prints out the current value of the registers and data memory, and that should match with the expected output
- The basic rules (submission rule, etc...) are the same as Project 1, but please ask TAs if anything is unclear.

# Subset of MIPS Instructions to Support in Proj2

---

- Arithmetic/logical: **add**, **sub**, **and**, **or**, **slt**,
- Arithmetic/logical with immediate: **addi**, **andi**, **ori**, **slti**, **lui**
- Memory access: **lw**, **sw**
- Control transfer: **beq**, **bne**, **j**
- No-operation: **nop**
  - ❖ Actually, **nop** is a special case of **sll**. However, since we do not implement shift instructions, **nop** (instruction[31:0] = 0x00000000) should be treated separately.

# Signed / Unsigned?

- No need to explicitly distinguish signed / unsigned values for “add”, “sub”, and “addi” instructions. The 2’s complement number system will take care of additions and subtractions.
- NEED to distinguish signed / unsigned values for comparison. “slt” and “slti” treat the values as signed values. On the contrary, “sltu” and “sltiu” instructions treat the values as unsigned values. In this project, we implement the signed versions (“slt” and “slti”) only.

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

slt $t2, $t0, $t1      #$t2 = ( -2 < 1 ) ? 1 : 0
sltu $t3, $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 0xFFFF → sign extended to 0xFFFFFE → $t4 = ( 3 < -2 ) ? 1 : 0
sltiu $t7, $t0, -2      #Imm 0xFFFF → sign extended to 0xFFFFFE → $t4 = ( 3 < 4294967294 ) ? 1 : 0
```

# Data Structures to Implement

---

- 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 data bytes are initialized to **0xFF** at beginning
    - You can assume the instruction memory is always accessed at 4-byte boundaries (e.g., CPU won't fetch an instruction from address 0x00000123)
  - ❖ Data memory
    - Unlike a real CPU, data memory is separated from instruction memory
    - Data memory address range: **0x10000000 – 0x10010000** (64KB)
    - All data bytes are initialized to **0xFF** at beginning
    - Since we will implement the `lw` and `sw` instructions only out of all memory instructions, the data memory is also accessed at 4-byte boundaries
  - ❖ 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.

# Simulator Program Behavior

---

1. Similar to proj1, the input file name is given as the first command-line argument.
2. Your program reads the file and load the binary instructions to address **0x00000000** of the instruction memory.
3. Your program simulates each instruction one-by-one, up to **N** instructions.
  - ❖ **N** is given as the second command-line argument.
4. If the simulator reads an unsupported instruction before **N** instructions, print “unknown instruction” and exit.
5. Before the exit, depending on the third command-line argument, print the final status of the system.

# Output Options (1)

- If the third command-line argument is “reg”, print the register values in the following format

```
# ./mips-sim data.bin 10 reg
$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
#
```

- The PC value should be pointing to the address of the “next instruction to run”
  - For example...
    - If N is 0, the printed PC value is 0x00000000
    - If N is 1, the printed PC value is 0x00000004
    - ...
  - If the simulator is stopped due to an unknown instruction, print the PC+4 of the unknown instruction
  - If the simulator just executed jump or branch, PC should be pointing to the new instruction address

# Output Options (2)

- If the third command-line argument is “mem”, the program expects the fourth and fifth argument .
  - 4<sup>th</sup> argument : Starting address to print from the data memory (in hexadecimal)
  - 5<sup>th</sup> argument : Number of 4-byte words to print (in decimal)

```
# ./mips-sim data.bin 10 mem 0x10000010 6
```

0x00000001	← Data memory value at 0x10000010
0x00000012	← Data memory value at 0x10000014
0x00000123	← Data memory value at 0x10000018
0x00001234	← Data memory value at 0x1000001C
0xFFFFFFFF	← Data memory value at 0x10000020
0xFFFFFFFF	← Data memory value at 0x10000024
#	

# Output Options (3)

- If there is no third command-line argument or an incorrect argument is given, no need to print anything.

```
# ./mips-sim data.bin 1000  
unknown instruction  
#
```

This example is showing a case where the program encountered an unknown instruction while trying to execute 1000 instructions.

So, it printed “unknown instruction” and stopped. Since no output option is specified, no additional output is printed.

# Test Sample (1)

- ~swe3005/2020f/proj2/proj2\_1.bin
- “proj2\_1.bin” file represents the following assembly code.

```
andi $2, $0, 0x1234
lui $3, 0x8765
ori $3, $3, 0x4321
addi $4, $0, -1
andi $5, $3, 0xFFFF
sub $6, $4, 4
```

- Expected results →
  - ❖ Please note that the PC register indicates the address of the 7<sup>th</sup> instruction, which made the CPU to stop (the “unknown instruction”)
  - ❖ proj2\_1.bin only contains 6 instructions. Therefore, at 0x18, the instruction memory value is 0xFFFFFFFF (default value). This should be interpreted as an unknown instruction.
  - ❖ The printed PC value is showing PC+4 of the last instruction.

```
./mips-sim ~swe3005/2020f/proj2/proj2_1.bin 10
unknown instruction
$0: 0x00000000
$1: 0x00000000
$2: 0x00001234
$3: 0x87654321
$4: 0xffffffff
$5: 0x00004321
$6: 0xfffffff
$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: 0x0000001c
```

# Test Sample (2)

- ~swe3005/2020f/proj2/proj2\_2.bin
- “proj2\_2.bin” file represents the following assembly code.

```
lui $3, 0x1000
addi $4, 0x100
addi $5, 0x200
addi $6, 0x300
addi $7, 0x400
sw $4, 0($3)
sw $5, 4($3)
sw $6, 8($3)
sw $7, 12($3)
andi $8, $0, 0
andi $9, $0, 0
loop: lw $10, 0($3)
nop
add $9, $9, $10
addi $3, $3, 4
addi $8, $8, 1
slti $11, $8, 4
bne $11, $0, loop
nop
```

- Expected results →

```
./mips-sim ~swe3005/2020f/proj2/proj2_2.bin 40 reg
```

```
$0: 0x00000000
$1: 0x00000000
$2: 0x00000000
$3: 0x10000010
$4: 0x00000100
$5: 0x00000200
$6: 0x00000300
$7: 0x00000400
$8: 0x00000004
$9: 0x00000a00
$10: 0x00000400
$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: 0x0000004c
```

```
./mips-sim ~swe3005/2020f/proj2/proj2_2.bin 40 mem 0x10000000 5
```

```
0x00000100
0x00000200
0x00000300
0x00000400
0xffffffff
```

# Test Sample (3) - 1

- ~swe3005/2020f/proj2/proj2\_3.bin
- “proj2\_3.bin” file represents the following assembly code (A simple bubble sort).

```
lui $3, 0x1000
    addi $2, $2, 0x1
    addi $4, $0, 0x158
    addi $5, $0, 0x73
    addi $6, $0, 0x126
    addi $7, $0, 0x54
    addi $8, $0, 0x12
    addi $15, $15, 0x4
    addi $21, $0, 0x5
    addi $22, $0, 0x4
    sw $4,0($3)
    sw $5,4($3)
    sw $6,8($3)
    sw $7,12($3)
    sw $8,16($3)
loop2: add $9, $0, $3
    add $13, $15, $0
```

```
loop:   nop
        lw $10, 0($9)
        lw $11, 4($9)
        nop
        slt $12, $10, $11
        beq $12, $2, then
        nop
        sw $11, 0($9)
        sw $10, 4($9)
then:   addi $9, $9, 0x4
        sub $13, $13, $2
        slti $14, $13, 0x1
        bne $14, $0, pass
        nop
        j loop
        nop
pass:  sub $15, $15, $2
        slti $16, $15, 1
        bne $16, $2, loop2
        nop
        andi $16, $0, 0
        andi $15, $0, 0
        add $15, $15, $21
        and $17, $17, $0
        add $18, $0 ,$3
```

```
loop3:  lw $20, 0($18)
        nop
        add $16, $16, $20
        add $18, $18, $22
        sub $15, $15, $2
        slti $19, $15, 0x1
        bne $19, $2, loop3
        nop
        sw $16, 20($3)
        ori $15, $0, 0x5
        or $18, $0, $0
loop4:  sub $16, $16, $15
        add $17, $17, $2
        slt $18, $16, $15
        bne $18, $2, loop4
        nop
        sw $17, 24($3)
```

# Test Sample (3) - 2

- Expected results →

```
./mips-sim ~swe3005/2020f/proj2/proj2_3.bin 906 reg
```

```
unknown instruction  
$0: 0x00000000  
$1: 0x00000000  
$2: 0x00000001  
$3: 0x10000000  
$4: 0x00000158  
$5: 0x00000073  
$6: 0x00000126  
$7: 0x00000054  
$8: 0x00000012  
$9: 0x10000004  
$10: 0x00000054  
$11: 0x00000012  
$12: 0x00000000  
$13: 0x00000000  
$14: 0x00000001  
$15: 0x00000005  
$16: 0x00000000  
$17: 0x000000ab  
$18: 0x00000001  
$19: 0x00000001  
$20: 0x00000158  
$21: 0x00000005  
$22: 0x00000004  
$23: 0x00000000  
$24: 0x00000000  
$25: 0x00000000  
$26: 0x00000000  
$27: 0x00000000  
$28: 0x00000000  
$29: 0x00000000  
$30: 0x00000000  
$31: 0x00000000  
PC: 0x000000f0
```

```
./mips-sim ~swe3005/2020f/proj2/proj2_2.bin 15 mem 0x10000000 10
```

```
0x00000158  
0x00000073  
0x00000126  
0x00000054  
0x00000012  
0xffffffff  
0xffffffff  
0xffffffff  
0xffffffff  
0xffffffff
```

```
./mips-sim ~swe3005/2020f/proj2/proj2_2.bin 905 mem 0x10000000 10
```

```
0x00000012  
0x00000054  
0x00000073  
0x00000126  
0x00000158  
0x00000357  
0x000000ab  
0xffffffff  
0xffffffff  
0xffffffff
```

# Reference Implementation

---

- We provide a reference implementation (without source code) in the following location.
  - ❖ ~swe3005/2020f/proj2/mips-sim
- If you have difficulties in implementing your simulator, try to compare the output with the reference implementation's output.
- It may be difficult to match the final result of the application at once. Try to match the outputs one step at a time by changing the number of instructions to run ( $N$ )

```
~swe3005/2020s/projf/mips-sim ~swe3005/2020f/proj2/proj2_1.bin 1 reg  
~swe3005/2020s/projf/mips-sim ~swe3005/2020f/proj2/proj2_1.bin 2 reg  
~swe3005/2020s/projf/mips-sim ~swe3005/2020f/proj2/proj2_1.bin 3 reg  
...  
~swe3005/2020s/projf/mips-sim ~swe3005/2020f/proj2/proj2_1.bin 6 reg
```

# Project Environment

---

- We will use the department's In-Ui-Ye-Ji cluster
  - ❖ ~~swin.skku.edu~~
  - ❖ swui.skku.edu
  - ❖ swye.skku.edu
  - ❖ swji.skku.edu
  - ❖ ssh port: 1398
- You'll need a similar Makefile as proj1
  - ❖ Same executable file name (i.e., **mips-sim**)

# Submission

---

- Clear the build directory
  - ❖ Do not leave any executable or object file in the submission
- Use submit program
  - ❖ `~swe3005/bin/submit project_id path_to_submit`
  - ❖ If you want to submit the current directory...
    - `~swe3005/bin/submit proj2 .`

Submitted Files for proj2:

File Name	File Size	Time
<hr/>		
proj2-2020123456-Sep.05.17.22.388048074	268490	Thu Sep 5 17:22:49 2020

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

# Submission

---

- Only the last submission is accepted! This means that...
  - ❖ You can submit multiple times before the deadline.
  - ❖ You should submit all your files at once!
    - You need to submit a “directory” that contains your files
    - You should not submit individual files separately.

# Project 2 Due Date

---

- 2020 Nov 13<sup>th</sup>, 23:59:59
- No late submission