Introduction to Computer Architecture Project 2

Single-cycle RISC-V CPU Simulator

Hyungmin Cho

Department of Computer Science and Engineering Sungkyunkwan University

No Collaboration Rule

Programming assignments are individual projects.

- No explicit collaboration is allowed.
- If your code is found to be very similar with someone else's code, there will be a strict penalty.

Project 2 Overview

- In Project 2, you'll implement an instruction simulator that supports a subset of RISC-V instructions
 - What is an instruction simulator? Your program reads instructions from the binary file and execute the instructions one-by-one.
 - At the end of the execution, your program prints out the current value of the registers, and that should match with the expected output on a real RISC-V processor.
- The basic rules (submission rule, etc...) are the same as Project 1, but please ask TAs if anything is unclear.

RISC-V Instructions to Support

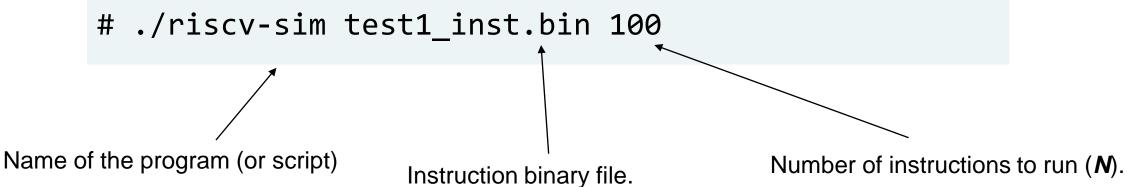
- add, sub, addi
- xor, or, and, xori, ori, andi
- slli, srli, srai, sll, srl, sra
- slti, slt
- auipc, lui
- jal, jalr
- beq, bne, blt, bge
- lw, sw

Simulator Program Behavior

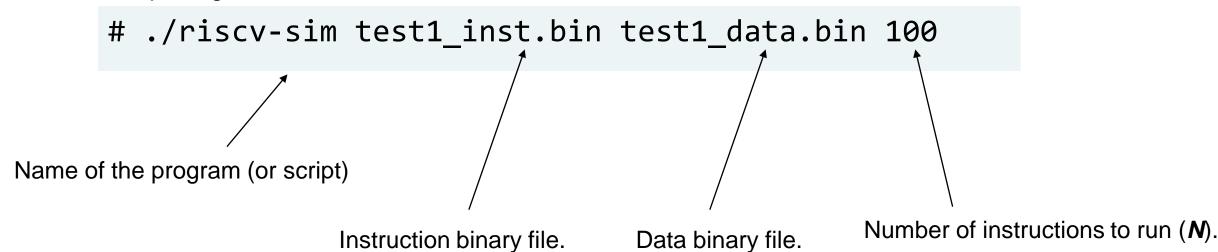
- Your program takes 2 or 3 command-line arguments.
- If the program is given with two arguments
 - 1. First argument: Input file name for binary instructions
 - This file includes binary instructions, same as proj1
 - > The contents of this file will be loaded to the instruction memory, from address 0x00000000
 - First instruction: 0x00000000
 - Second instruction: 0x00000004
 - ...
 - 2. Second argument: Number of instructions to execute (**N**)
- If the program is given with three arguments
 - 1. First argument: Input file name for binary instructions
 - 2. Second argument: Input file name for binary data
 - The contents of this file will be loaded to the data memory, from address 0x10000000
 - Third argument: Number of instructions to execute (*N*)

Simulator Program Behavior

Case1: Two input arguments



Case2: Three input arguments



Number of Instructions to Execute

- Your program simulates N instructions.
 - ❖ If there is no more instruction to execute after executing N instructions, stop simulation
 - * If there are branch or jump instructions, the number of instructions to execute can be different than the number of instructions in the program.

```
addi x2, x0, 0x123
lui x3, 0x87654
ori x3, x3, 0x321
addi x4, x0, -1
andi x5, x3, 0x7FF
addi x6, x4, -4
slli x7, x6, 16
srli x8, x7, 8
slt x9, x8, x5
slti x10, x5, 0x7FF
```

Number of instructions in the program: 10 Number of instructions to execute: 10

```
ori x8, x0, 10
ori x9, x0, 20
ori x10, x0, 0

loop:
    add x10, x10, x8
    addi x8, x8, 1
    slt x11, x8, x9
    bne x11, x0, loop
```

Number of instructions in the program: 7 Number of instructions to execute: 43

Registers

■ All registers are initialized to zero (0x00000000) at the beginning.

x0 is fixed to zero.

Register Output

```
# ./riscv-sim proj1 1.bin 10
x0: 0x00000000
x1: 0x00000000
x2: 0x00000123
x3: 0x87654321
x4: 0xffffffff
x5: 0x00000321
x6: 0xffffffb
x7: 0xfffb0000
x8: 0x00fffb00
x9: 0x00000000
x10: 0x00000001
x11: 0x00000000
x12: 0x00000000
x13: 0x00000000
x14: 0x00000000
x15: 0x00000000
x16: 0x00000000
x17: 0x00000000
x18: 0x00000000
x19: 0x00000000
x20: 0x00000000
x21: 0x00000000
x22: 0x00000000
x23: 0x00000000
x24: 0x00000000
x25: 0x00000000
x26: 0x00000000
x27: 0x00000000
x28: 0x00000000
x29: 0x00000000
x30: 0x00000000
x31: 0x00000000
```

 After the execution, print the final status of the registers (x0-x31)

- Always print in 32-bit hexadecimal, with leading zeros.
 - E.g., 0x20 should be printed as 0x00000020

Data memory

- Address range: 0x10000000 0x1000FFFF (64KB)
 - Do not need to check memory address boundary. Assume all accesses are within this range.
- The instruction binary file is loaded onto this instruction memory from 0x10000000.
- The remaining bytes after loading the data binary file are initialized to 0xFF

Reference Implementation

- We provide a reference implementation (without source code) in the following location.
 - * ~swe3005/2023f/proj2/riscv-sim
- If you have difficulties in implementing your simulator, try to compare the output with the reference implementation's output.
- If you think it is difficult to match the final results of the application at once, try to match the outputs one step at a time by changing the number of instructions (N)

```
~swe3005/2023f/proj2/riscv-sim ~swe3005/2023f/proj2/proj2_1.bin 1
~swe3005/2023f/proj2/riscv-sim ~swe3005/2023f/proj2/proj2_1.bin 2
~swe3005/2023f/proj2/riscv-sim ~swe3005/2023f/proj2/proj2_1.bin 3
...
~swe3005/2023f/proj2/riscv-sim ~swe3005/2023f/proj2/proj2_1.bin 10
~swe3005/2023f/proj2/riscv-sim ~swe3005/2023f/proj2/proj2_1.bin 11
```

Test Samples

There are total 8 test cases.

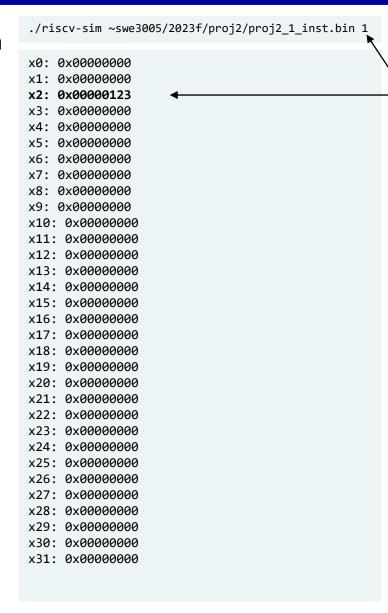
```
~swe3005/2023f/proj2/proj2_1.bin
~swe3005/2023f/proj2/proj2_2.bin
...
~swe3005/2023f/proj2/proj2_8.bin
```

 Try to match the output of your simulator program to the output of the reference program.

Test Sample (1)

- ~swe3005/2023f/proj2/proj2_1_inst.bin
- This test case corresponds to the following assembly code with 10 instructions

```
addi x2, x0, 0x123
lui x3, 0x87654
ori x3, x3, 0x321
addi x4, x0, -1
andi x5, x3, 0x7FF
addi x6, x4, -4
slli x7, x6, 16
srli x8, x7, 8
slt x9, x8, x5
slti x10, x5, 0x7FF
```



N is 1. Execute the first instruction only and print the register values.

Test Sample (1)

./riscv-sim ~swe3005/2023f/proj2/proj2 1 inst.bin 4 x0: 0x00000000 x1: 0x00000000 x2: 0x00000123 x3: 0x87654321 **N** is 4. x4: 0xffffffff Execute 4 x5: 0x00000000 x6: 0x00000000 instructions and x7: 0x00000000 x8: 0x00000000 print the register x9: 0x00000000 0x00000001 values. x11: 0x00000000 x12: 0x00000000 0x00000000 x14: 0x00000000 x15: 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000 x21: 0x00000000 0x00000000 0x00000000 x24: 0x000000000 0x00000000 x26: 0x00000000 x27: 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000

./riscv-sim ~swe3005/2023f/proj2/proj2_1_inst.bin 10 x0: 0x00000000 0x00000000 0x00000123 0x87654321 **N** is 10. 0xffffffff Execute 10 0x00000321 x6: 0xffffffb instructions and 0xfffb0000 0x00fffb00 print the register 0x00000000 x10: 0x00000001 values. x11: 0x00000000 x12: 0x00000000 x13: 0x00000000 x14: 0x00000000 x15: 0x00000000 x16: 0x00000000 x17: 0x00000000 x18: 0x00000000 x19: 0x00000000 x20: 0x00000000 x21: 0x00000000 x22: 0x00000000 x23: 0x00000000 x24: 0x00000000 x25: 0x00000000 x26: 0x00000000 x27: 0x00000000 x28: 0x00000000 x29: 0x00000000 x30: 0x00000000 x31: 0x00000000

./riscv-sim ~swe3005/2023f/proj2/proj2_1_inst.bin 11 x0: 0x00000000 0x00000000 0x00000123 x3: 0x87654321 **N** is 11 0xffffffff x5: 0x00000321 Since there are only x6: 0xffffffb 0xfffb0000 10 instructions in the 0x00fffb00 input binary file, x9: 0x00000000 x10: 0x00000001 execute the 10 x11: 0x00000000 x12: 0x00000000 instructions and stop x13: 0x00000000 x14: 0x000000000 the execution. x15: 0x00000000 x16: 0x00000000 x17: 0x00000000 x18: 0x00000000 x19: 0x00000000 0x00000000 x21: 0x00000000 x22: 0x00000000 0x00000000 x24: 0x000000000 x25: 0x00000000 x26: 0x00000000 x27: 0x00000000 x28: 0x00000000 0x00000000 0x00000000 x31: 0x00000000

Test Sample (2)

- ~swe3005/2023f/proj2/proj2_2_inst.bin
- This test case corresponds to the following assembly code with 9 instructions

```
addi x1, x0, 10
addi x2, x0, 5
sub x3, x0, x2
sll x4, x3, x2
srl x5, x3, x2
sra x6, x3, x2
or x7, x4, x5
xor x8, x4, x5
and x9, x4, x5
```

```
x0: 0x00000000
x1: 0x0000000a
x2: 0x00000005
x3: 0xffffffb
x4: 0xffffff60
x5: 0x07ffffff
x6: 0xffffffff
x7: 0xffffffff
x8: 0xf800009f
x9: 0x07ffff60
x10: 0x00000000
x11: 0x00000000
x12: 0x00000000
x13: 0x00000000
x14: 0x00000000
x15: 0x00000000
x16: 0x00000000
x17: 0x00000000
x18: 0x00000000
x19: 0x00000000
x20: 0x00000000
x21: 0x00000000
x22: 0x00000000
x23: 0x00000000
x24: 0x00000000
x25: 0x00000000
x26: 0x00000000
x27: 0x00000000
x28: 0x00000000
x29: 0x00000000
x30: 0x00000000
x31: 0x00000000
```

./riscv-sim ~swe3005/2023f/proj2/proj2 2 inst.bin 9

Test Sample (3)

- ~swe3005/2023f/proj2/proj2_3 _inst.bin
- proj2_3_inst.bin has 100 instructions

```
slti
            a7, a4, 473
addi
           t5, s1, 1660
andi
           s5, ra, 1476
srli
           t3, gp, 28
slti
           t5, a2, -1361
andi
           zero, t0, -1797
slli
           a1, s7, 2
slli
           a5, s8, 10
slt
            zero, s1, gp
            a7, s1, s5
sub
            s3, s10, ra
xor
addi
           t6, zero, 1578
           s6, a6, t2
sub
addi
           t3, t6, -1017
```

```
./riscv-sim ~swe3005/2023f/proj2/proj2 3 inst.bin 100
x0: 0x00000000
x1: 0xffffffff
x2: 0xc9d0c000
x3: 0x00000001
x4: 0x00000000
x5: 0x00000000
x6: 0xfe4e8600
x7: 0x00000000
x8: 0x00000000
x9: 0x00000000
x10: 0x00000000
x11: 0x00000000
x12: 0x00000001
x13: 0x86541001
x14: 0x00004000
x15: 0x00000000
x16: 0xfffffe35
x17: 0xffffffff
x18: 0x00000000
x19: 0xffffff3e
x20: 0x00000000
x21: 0x00000001
x22: 0xfffffe35
x23: 0x00000000
x24: 0x00000001
x25: 0x00000000
x26: 0x00000001
x27: 0x00000000
x28: 0x00000231
x29: 0x86541000
x30: 0x86541000
x31: 0x0000062a
```

Test Sample (4)

- ~swe3005/2023f/proj2/proj2_4_inst.bin
- This test case corresponds to the following assembly code with 7 instructions.
- The number of instructions to execute is 43 due to the branch.

```
ori x8, x0, 10
ori x9, x0, 20
ori x10, x0, 0

loop:

add x10, x10, x8
addi x8, x8, 1
slt x11, x8, x9
bne x11, x0, loop
```

```
./riscv-sim ~swe3005/2023f/proj2/proj2 4 inst.bin 43
x0: 0x00000000
x1: 0x00000000
x2: 0x00000000
x3: 0x00000000
x4: 0x00000000
x5: 0x00000000
x6: 0x00000000
x7: 0x00000000
x8: 0x00000014
x9: 0x00000014
x10: 0x00000091
x11: 0x00000000
x12: 0x00000000
x13: 0x00000000
x14: 0x00000000
x15: 0x00000000
x16: 0x00000000
x17: 0x00000000
x18: 0x00000000
x19: 0x00000000
x20: 0x00000000
x21: 0x00000000
x22: 0x00000000
x23: 0x00000000
x24: 0x00000000
x25: 0x00000000
x26: 0x00000000
x27: 0x00000000
x28: 0x00000000
x29: 0x00000000
x30: 0x00000000
x31: 0x00000000
```

Test Sample (5)

- ~swe3005/2023f/proj2/proj2_5_inst.bin ~swe3005/2023f/proj2/proj2_5_data.bin
- This test case corresponds to the following assembly code.
- This case has an additional data file that should be loaded into the data memory

```
lui x3, 0x10000
1w x4, 0(x3)
1w x5, 4(x3)
addi x6, x4, 123
sw x6, 0(x3)
srli x7, x5, 12
sw x7, 8(x3)
addi x8, x3, 8
1w x9, 0(x8)
1w \times 10, -4(\times 8)
nop
.data
val1: .word 0x456789ab
val2: .word 0xdeadbeef
```

```
x0: 0x00000000
x1: 0x00000000
x2: 0x00000000
x3: 0x10000000
x4: 0x456789ab
x5: 0xdeadbeef
x6: 0x45678a26
x7: 0x000deadb
x8: 0x10000008
x9: 0x000deadb
x10: 0xdeadbeef
x11: 0x00000000
x12: 0x00000000
x13: 0x00000000
x14: 0x00000000
x15: 0x00000000
x16: 0x00000000
x17: 0x00000000
x18: 0x00000000
x19: 0x00000000
x20: 0x00000000
x21: 0x00000000
x22: 0x00000000
x23: 0x00000000
x24: 0x00000000
x25: 0x00000000
x26: 0x00000000
x27: 0x00000000
x28: 0x00000000
x29: 0x00000000
x30: 0x00000000
x31: 0x00000000
```

./riscv-sim ~swe3005/2023f/proj2/proj2 5 inst.bin ~swe3005/2023f/proj2/proj2 5 data.bin 11

Test Sample (6)

- ~swe3005/2023f/proj2/proj2_6_inst.bin
- This test case corresponds to the following assembly code with 10 instructions.
- The number of instructions to execute is 11 due to the jump & branch.

```
addi x1, x0, 0x124
    ori x2, x1, 0x400
    slli x3, x2, 1
    lui x4, 48
    srl x5, x4, 2
    addi x5, x5, -440
L1: jal L jal
L_jr:
    beq x3, x5, L end
L jal:
    jalr x1
L end:
    addi x10, x0, 1
```

```
x0: 0x00000000
x1: 0x00000024
x2: 0x00000524
x3: 0x00000a48
x4: 0x00030000
x5: 0x0000be48
x6: 0x00000000
x7: 0x00000000
x8: 0x00000000
x9: 0x00000000
x10: 0x00000001
x11: 0x00000000
x12: 0x00000000
x13: 0x00000000
x14: 0x00000000
x15: 0x00000000
x16: 0x00000000
x17: 0x00000000
x18: 0x00000000
x19: 0x00000000
x20: 0x00000000
x21: 0x00000000
x22: 0x00000000
x23: 0x00000000
x24: 0x00000000
x25: 0x00000000
x26: 0x00000000
x27: 0x00000000
x28: 0x00000000
x29: 0x00000000
x30: 0x00000000
x31: 0x00000000
```

./riscv-sim ~swe3005/2023f/proj2/proj2 6 inst.bin 11

Test Sample (7)

- ~swe3005/2023f/proj2/proj2_7_inst.bin ~swe3005/2023f/proj2/proj2_7_data.bin
- This test case is compiled from the following c code.

```
int data[] = {3,6,9,12,15,18,21,24,27,30,33};
int funcA(int a, int b);
int funcB(int idx);
int funcC();
int main () {
  int i = 100:
 int j = 200;
  int k = 3;
 int res1 = funcA(i,j);
  int res2 = funcB(k);
  return res1+res2;
int funcA(int a, int b) {
  return a | b;
int funcB(int idx) {
 int val = data[idx] + funcC();
  return val;
int funcC() {
  return 0x123;
```

./riscv-sim ~swe3005/2023f/proj2/proj2 7 inst.bin ~swe3005/2023f/proj2/proj2 7 data.bin 73

```
x0: 0x00000000
x1: 0x0000000c
x2: 0x10010000
x3: 0x00000000
x4: 0x00000000
x5: 0x00000000
x6: 0x00000000
x7: 0x00000000
x8: 0x00000000
x9: 0x00000000
x10: 0x0000021b
x11: 0x000000c8
x12: 0x00000000
x13: 0x00000000
x14: 0x000000ec
x15: 0x0000021b
x16: 0x00000000
x17: 0x00000000
x18: 0x00000000
x19: 0x00000000
x20: 0x00000000
x21: 0x00000000
x22: 0x00000000
x23: 0x00000000
x24: 0x00000000
x25: 0x00000000
x26: 0x00000000
x27: 0x00000000
x28: 0x00000000
x29: 0x00000000
x30: 0x00000000
x31: 0x00000000
```

Test Sample (8)

- ~swe3005/2023f/proj2/proj2_8_inst.bin ~swe3005/2023f/proj2/proj2_8_data.bin
- This test case is compiled from the following c code.

```
#define N 10
int list[N] = \{300, -22, 0, 123, -512, 30, 20, 10, 40, -400\};
void swap(int *xp, int *yp)
   int temp = *xp;
    *xp = *yp;
    *yp = temp;
int main () {
 int i,j;
  for (i = 0; i < N-1; i++)
   for (j = 0; j < N-i-1; j++)
     if (list[j] > list[j+1])
         swap(&list[j], &list[j+1]);
  unsigned int addr = 0x10000000;
  asm("lw t0, 0(%0)"::"r" (addr));
  asm("lw t1, 4(%0)"::"r" (addr));
  asm("lw t2, 8(%0)"::"r" (addr));
  asm("lw t3, 12(%0)"::"r" (addr));
  asm("lw t4, 16(%0)"::"r" (addr));
  asm("lw t5, 20(%0)"::"r" (addr));
  asm("lw t6, 36(%0)"::"r" (addr));
  return 0;
```

./riscv-sim ~swe3005/2023f/proj2/proj2 8 inst.bin ~swe3005/2023f/proj2/proj2 8 data.bin 2001

```
x0: 0x00000000
x1: 0x0000000c
x2: 0x10010000
x3: 0x00000000
x4: 0x00000000
x5: 0xfffffe00
x6: 0xfffffe70
x7: 0xffffffea
x8: 0x00000000
x9: 0x00000000
x10: 0x00000000
x11: 0x10000008
x12: 0x00000000
x13: 0x10000000
x14: 0x00000009
x15: 0x00000000
x16: 0x00000000
x17: 0x00000000
x18: 0x00000000
x19: 0x00000000
x20: 0x00000000
x21: 0x00000000
x22: 0x00000000
x23: 0x00000000
x24: 0x00000000
x25: 0x00000000
x26: 0x00000000
x27: 0x00000000
x28: 0x00000000
x29: 0x0000000a
x30: 0x00000014
x31: 0x0000012c
```

Additional Functions

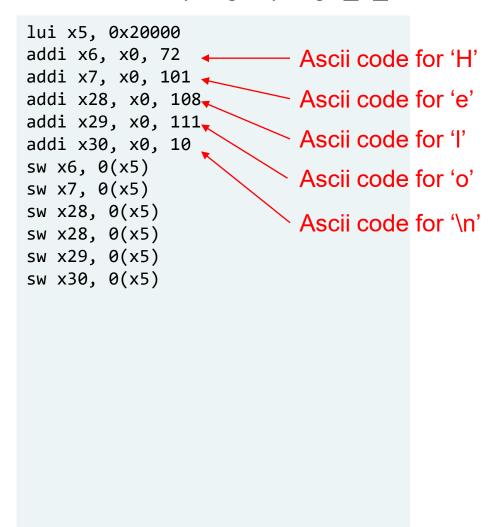
- Store a word (sw) to 0x20000000
 - Print the ascii character corresponds to the stored data the console (stdout)
 - Do not print the newline after the character.
 - > If you're using C, printf("%c", value_8bit)
 - > If you're using C++, cout << (char)value_8bit;</pre>
 - If you're using Python2, print chr(value_8bit),
 - , is added to omit the newline character after the print
 - > If you're using Python3, print(chr(value_8bit),end='')
 - end='' is added to omit the newline character after the print.

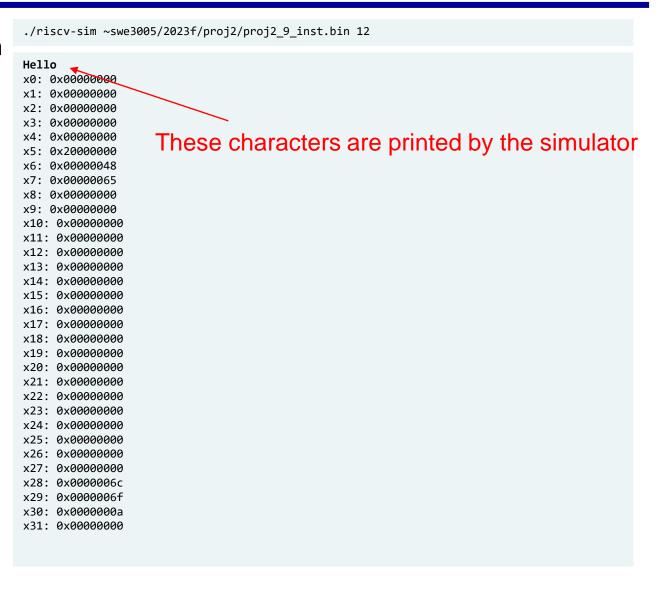
Additional Functions

- Load a word (**Iw**) from 0x20000000
 - Wait for the user to enter a number though the console (stdin)
 - > The number will be a **decimal integer**
 - We would not test error handling. This means you can safely assume that the user only enters a valid integer.
 - Once the user enters a number, the number shall be loaded to the rd register, as if the value is loaded from memory.

Test Sample (9)

~swe3005/2023f/proj2/proj2_9_inst.bin



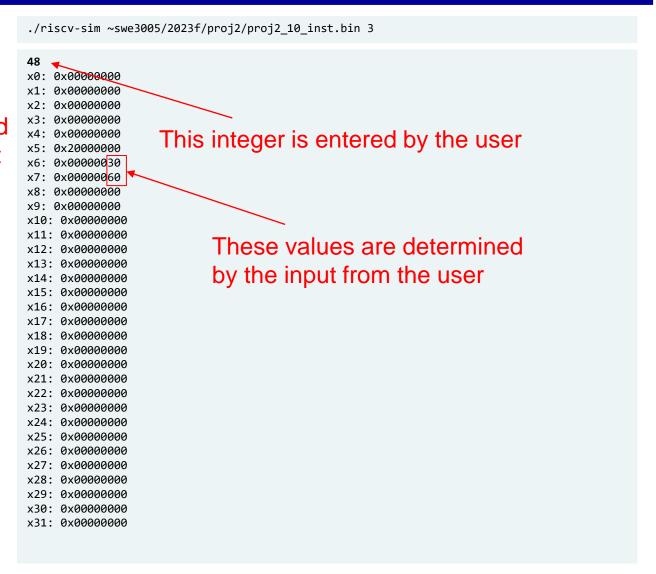


Test Sample (10)

~swe3005/2023f/proj2/proj2_10_inst.bin

lui x5, 0x20000 lw x6, 0(x5) slli x7, x6, 1

Your simulator should wait for the user input when executing this instruction



Test Sample (11)

- ~swe3005/2023f/proj3/proj2_11_inst.bin ~swe3005/2023f/proj3/proj2_11_data.bin
- proj2_11 is a simple calculator that performs add or subtraction
- Simulate the program with a large N

```
./riscv-sim ~swe3005/2023f/proj2/proj2 11 inst.bin ~swe3005/2023f/proj2/proj2 11 data.bin 1000000000
SWE3005 Project 3
-- Calculator --
Choose your function:
1. Add
2. Subtract
3. Exit
Enter operand 1: 123
Enter operand 2: 255
Result: 378
Choose your function:
1. Add
2. Subtract
3. Exit
Enter operand 1: 200
Enter operand 2: 99
Result: 101
                                                 Red ones indicate the numbers entered by the user
Choose your function:

    Add

2. Subtract
3. Exit
Enter operand 1: 100
Enter operand 2: -45
Result: 55
Choose your function:

    Add

2. Subtract
Exit
Enter operand 1: 95
Enter operand 2: -33
Result: 128
Choose your function:

    Add

2. Subtract
Exit
x0: 0x00000000
x1: 0x00000000
x2: 0x10010000
x3: 0x00000000
x4: 0x00000000
x5: 0x00000000
x6: 0x00000000
x7: 0x00000000
x8: 0x00000000
```

Project Environment

- We will use the department's In-Ui-Ye-Ji cluster
 - * swui.skku.edu
 - * swye.skku.edu
 - * swji.skku.edu
 - * ssh port: 1398

- If you have a problem with the account, send an e-mail to the server admin
 - inuiyeji-skku@googlegroups.com
 - Do not send an email that is not related to the account itself!
 - If you're not sure, ask the TAs first.

Makefile / Script

- If you're using C/C++, you need to submit Makefile as proj1
 - The output executable file name should be the same as proj1 (i.e., riscv-sim)
 - Beware about the tabs (not spaces) in Makefile
 - You need to include Makefile in your submission
- If you're using Python, add a script file named "riscv-sim"
 - Don't forget to add executable permission (chmod +x riscv-sim)
 - * You need to include this script file in your submission

Makefile Example

C

Makefile

```
CC=gcc
CCFLAGS=
#add C source files here
SRCS=main.c
TARGET=riscv-sim
OBJS := $(patsubst %.c,%.o,$(SRCS))
all: $(TARGET)
%.o:%.c
           $(CC) $(CCFLAGS) $< -c -o $@
$(TARGET): $(OBJS)
           $(CC) $(CCFLAGS) $^ -o $@
.PHONY=clean
clean:
           rm -f $(OBJS) $(TARGET)
```

■ C++

Makefile

```
CXX=g++
CXXFLAGS=
#add C++ source files here
SRCS=main.cc
TARGET=riscv-sim
OBJS := $(patsubst %.cc, %.o, $(SRCS))
all: $(TARGET)
%.o:%.cc
           $(CXX) $(CXXFLAGS) $< -c -o $@
$(TARGET): $(OBJS)
           $(CXX) $(CXXFLAGS) $^ -o $@
.PHONY=clean
clean:
           rm -f $(OBJS) $(TARGET)
```

Script Example

Python (if your python file is riscv-sim.py)

riscv-sim — Don't forget to give the excute permission: chmod +x riscv-sim

python3 riscv-sim.py \${@}

- Also, be aware of the python version on the server
 - python: python 2.7.17
 - python3: python 3.6.9

\${@} means all arguments

Submission

- Clear the build directory
 - Do not leave any executable or object file in the submission
- Use the submit program
 - * ~swe3005/bin/submit project_id directory_to_submit

```
      Submitted Files for proj2:

      File Name
      File Size
      Time

      proj2-2020123456-Sep.05.17.22.388048074
      268490
      Thu Sep 5 17:22:49 2020
```

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

Project 2 Due Date

■ 2023, Nov. 17th, 23:59:59

No late submission