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? Your program reads instructions from the binary file and execute instruction one-by-one.
  - We only consider the **register values** and data **memory contents**.
  - 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 MIPS processor.
- 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, addu, sub, subu, and, or, slt, sltu`**
- Arithmetic/logical with immediate: **`addi, addiu, andi, ori, lui, slti, sltiu`**
- Control transfer: **`beq, bne, j, jal, jr`**
- Shift instructions: **`sll, srl`**
- Memory access: **`lw, sw`**
  - No need to implement **`lh, lhu, lb, sh, shu, sb`**

Signed / Unsigned?

- No need to explicitly distinguish signed / unsigned versions of "**add**", "**sub**", "**addu**", "**subu**", "**addiu**" and "**addi**" instructions. The 2's complement number system will take care of additions and subtractions.

- You 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.

```
addi $t1, $zero, 1   #$t1 = 0x1
addi $t0, $zero, -2  #$t0 = 0xFFFFFFFE (sign extension). 0xFFFFFFFE is considered as -2 if signed, 4294967294 if unsigned.

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

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

slti $t4, $t0, 2     #Imm 0x0002 → sign extended to 0x00000002 → $t4 = ( 3 < 2 ) ? 1 : 0 → $t3 is 0
sltiu $t5, $t0, 2    #Imm 0x0002 → sign extended to 0x00000002 → $t4 = ( 3 < 2 ) ? 1 : 0 → $t3 is 0

slti $t6, $t0, -2    #Imm 0xFFFE → sign extended to 0xFFFFFFFE → $t4 = ( 3 < -2 ) ? 1 : 0 → $t3 is 0
sltiu $t7, $t0, -2   #Imm 0xFFFE → sign extended to 0xFFFFFFFE → $t4 = ( 3 < 4294967294 ) ? 1 : 0  → $t3 is 1
```

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 (a word, which is 4-byte, is initialized to **0xFFFFFFFF**)
  - 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 (a word, which is 4-byte, is initialized to **0xFFFFFFFF**)
  - 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.
    - You may freely use a generic library* if you want.
    - Generic library: Libraries that provide general-purpose data structures, functions, etc. For example, you're not allowed to use a specialized library that simulates MIPS instructions automatically (if there is such a thing…)

Simulator Program Behavior

1. Your program takes 2 or 3 command-line arguments.

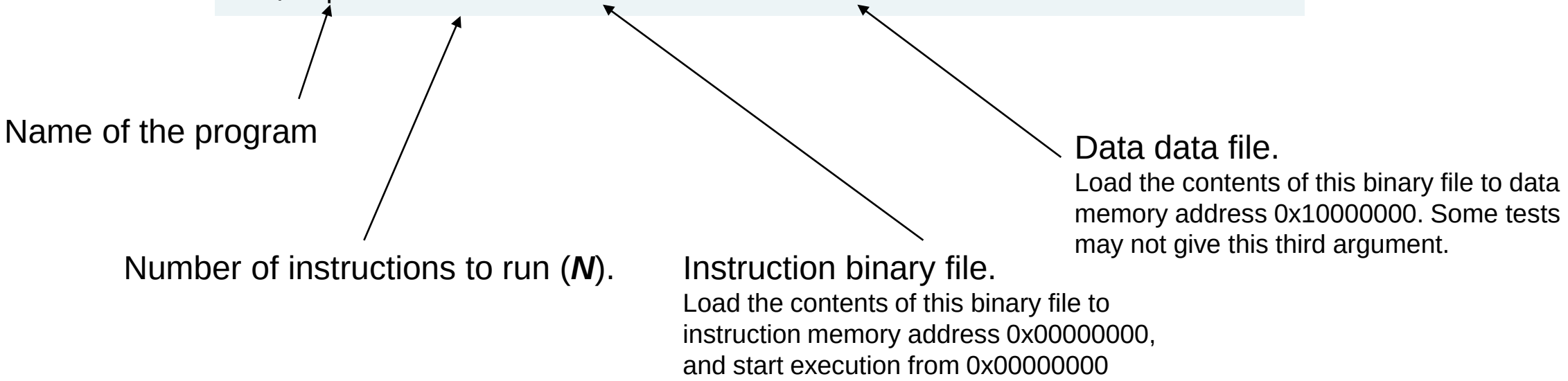
2. The first argument is the number of instructions to execute (***N***)
   - Your program simulates each instruction one-by-one, up to ***N*** instructions.
   - If the simulator reads an unsupported instruction before ***N*** instructions, print "`unknown instruction`" and exit.
   - At the end of the execution, print the final status of the registers

3. The second argument is the instruction binary file.
   - This file should be loaded to address **`0x00000000`** of the instruction memory

4. The third, optional argument is the data binary file.
   - This file should be loaded to address **`0x10000000`** of the data memory
   - Some tests does not give the third argument. In this case, no data is loaded to the data memory. All data bytes are initialized to 0xFF in this case.

Simulator Program Behavior

```
# ./mips-sim 100 test1.inst test1.data
```



Register Output

- Print the register values in the following format

```
# ./mips-sim 10 test1.inst test1.data
$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
    - i.e., consider the ‘unknown instruction’ as an executed instruction.
  - If the simulator just executed jump or branch, PC should be pointing to the new instruction address

Reference Implementation

- We provide a reference implementation (without source code) in the following location.
  - `~swe3005/2022s/proj2/mips-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*)

```
~swe3005/2022s/proj2/mips-sim 1 ~swe3005/2022s/proj2/proj2_1.bin
~swe3005/2022s/proj2/mips-sim 2 ~swe3005/2022s/proj2/proj2_1.bin
~swe3005/2022s/proj2/mips-sim 3 ~swe3005/2022s/proj2/proj2_1.bin
...
~swe3005/2022s/proj2/mips-sim 6 ~swe3005/2022s/proj2/proj2_1.bin
```

Test Sample (1)

- ~swe3005/2022s/proj2/proj2_1.inst
- “proj2_1.inst” file represents the following assembly code.

Address: 0x0
Address: 0x4

```
addi $2, $0, 0x1234
lui $3, 0x8765
ori $3, $3, 0x4321
addi $4, $0, -1
andi $5, $3, 0xFFFF
sub $6, $4, 4
sll $7, $6, 16
srl $8, $7, 8
slt $9, $8, $5
slti $10, $5, 0x7FFF
```

Address: 0x24

Loaded from proj2_1.bin

Address: 0x28
0xFFFFFFF

Address: 0x2C
0xFFFFFFF

Default memory value
→ Should be considered as an unknown instruction

- Expected results →
  - Please note that the PC register indicates the address of the 12th instruction (0x2C).
  - proj2_1.bin contains 10 instructions. Therefore, at 0x28, the 11th instruction, the instruction memory value is 0xFFFFFFFF (default value). This should be interpreted as an unknown instruction, and if the CPU executes this, the CPU stops.
  - When the CPU stops, the PC value should be PC+4 of the instruction that made the CPU to stop.

```
./mips-sim 11 ~swe3005/2022s/proj2/proj2_1.inst
```

```
unknown instruction
$0: 0x00000000
$1: 0x00000000
$2: 0x00001234
$3: 0x87654321
$4: 0xffffffff
$5: 0x00004321
$6: 0xfffffffb
$7: 0xfffb0000
$8: 0x00fffb00
$9: 0x00000000
$10: 0x00000001
$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: 0x0000002C
```

Test Sample (2)

- ~swe3005/2022s/proj2/proj2_2.inst

```
      ori $8, $0, 10
      ori $9, $0, 20
      ori $10, $0, 0

loop:
      add $10, $10, $8
      addi $8, $8, 1
      slt $11, $8, $9
      bne $11, $0, loop
```

```
./mips-sim 44 ~swe3005/2022s/proj2/proj2_2.inst
```

```
$0: 0x00000000
$1: 0x00000000
$2: 0x00000000
$3: 0x00000000
$4: 0x00000000
$5: 0x00000000
$6: 0x00000000
$7: 0x00000000
$8: 0x00000014
$9: 0x00000014
$10: 0x00000091
$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
```

Test Sample (3)

- ~swe3005/2022s/proj2/proj2_3.inst
  ~swe3005/2022s/proj2/proj2_3.data

```
lui $3, 0x1000
lw $4, 0($3)
lw $5, 4($3)
addi $6, $4, 1234
sw $6, 0($3)
srl $7, $5, 12
sw $7, 8($3)
addi $8, $3, 8
lw $9, 0($8)
lw $10, -4($8)
nop


.data
val1: .word 0x456789ab
val2: .word 0xdeadbeef
```

```
./mips-sim 12 ~swe3005/2022s/proj2/proj2_3.inst ~swe3005/2022s/proj2/proj2_3.data
```

```
unknown instruction
$0: 0x00000000
$1: 0x00000000
$2: 0x00000000
$3: 0x10000000
$4: 0x456789ab
$5: 0xdeadbeef
$6: 0x45678e7d
$7: 0x000deadb
$8: 0x10000008
$9: 0x000deadb
$10: 0xdeadbeef
$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: 0x00000030
```

Test Sample (3) - Data memory status

- Initial

Address	Value (Hex)
0x10000000	0x45
0x10000001	0x67
0x10000002	0x89
0x10000003	0xAB
0x10000004	0xDE
0x10000005	0xAD
0x10000006	0xBE
0x10000007	0xEF
0x10000008	0xFF
0x10000009	0xFF
0x1000000A	0xFF
0x1000000B	0xFF

- After execution

Address	Value (Hex)
0x10000000	0x45
0x10000001	0x67
0x10000002	0x8E
0x10000003	0x7D
0x10000004	0xDE
0x10000005	0xAD
0x10000006	0xBE
0x10000007	0xEF
0x10000008	0x00
0x10000009	0x0D
0x1000000A	0xEA
0x1000000B	0xDB

Test Sample (4)

- ~swe3005/2022s/proj2/proj2_4.inst

```
    addi $1, $0, 0x3124
    or $2, $1, 0x8000
    sll $3, $2, 1
    lui $4, 3
    srl $5, $4, 1
    addi $5, $5, -7608

L1: jal L_jal
    nop

L_jr:
    beq $3, $5,  L_end
    nop

L_jal:
    j $31
    nop

L_end:
    nop
    addi $10, $0, 1
    nop
```

```
./mips-sim 14 ~swe3005/2022s/proj2/proj2_4.inst
```

```
unknown instruction
$0: 0x00000000
$1: 0x00003124
$2: 0x0000b124
$3: 0x00016248
$4: 0x00030000
$5: 0x00016248
$6: 0x00000000
$7: 0x00000000
$8: 0x00000000
$9: 0x00000000
$10: 0x00000001
$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: 0x0000001c
PC: 0x00000040
```

Test Sample (5)

- ~swe3005/2022s/proj2/proj2_5.inst ~swe3005/2022s/proj2/proj2_5.data
- proj2_5 is compiled from the following C source.

```
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;
}
```

```
./mips-sim 80 ~swe3005/2022s/proj2/proj2_5.inst ~swe3005/2022s/proj2/proj2_5.data
```

```
unknown instruction
$0: 0x00000000
$1: 0x00000000
$2: 0x0000021b
$3: 0x000000ec
$4: 0x00000003
$5: 0x000000c8
$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: 0x10010000
$30: 0x00000000
$31: 0x0000000c
PC: 0x00000154
```

Test Sample (6)

- ~swe3005/2022s/proj2/proj2_6.inst ~swe3005/2022s/proj2/proj2_6.data
- proj2_6 is compiled from the following C source.

```
#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]);

  return 0;
}
```

```
./mips-sim 2637 ~swe3005/2022s/proj2/proj2_6.inst ~swe3005/2022s/proj2/proj2_6.data
```

```
unknown instruction
$0: 0x00000000
$1: 0x00000000
$2: 0x00000000
$3: 0x00000001
$4: 0x00000004
$5: 0x10000008
$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: 0x10010000
$30: 0x00000000
$31: 0x0000000c
PC: 0x000001b0
```

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

- You'll need a similar Makefile as proj1
  - Same executable file name (i.e., **`mips-sim`**)

- 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.

Submission

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

- Use the `submit` program
  - `~swe3005/bin/submit project_id path_to_submit`
  - If you want to submit the ‘project_2’ directory…
    - **`~swe3005/bin/submit proj2 project_2`**

```
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

- 2022 May 6th, 23:59:59

- No late submission