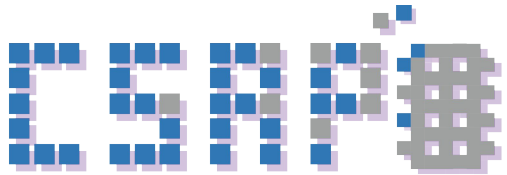
Computer Architecture Lab Session

5. Pipeline Lab

2020/05/25

comparch@csap.snu.ac.kr



Computer Systems and Platforms Laboratory
School of Computer Science and Engineering
Seoul National University

Overview

- Due: Monday, June 8, 11:00
- In this lab, you will implement a 3-stage pipelined RISC-V simulator.
- Read the README carefully. It contains the full instructions of the lab. This is an overview with some tips.
- Find the `Pipeline Lab` repository owned by `Computer Architecture TA`, fork it to your namespace and clone it to work on it locally.
- Also, you need `pyrisc` from assembly lab. If you erased it, clone it again from https://github.com/snu-csl/pyrisc
- Commit and push your work to your repository to submit. The timestamp on the last commit will count as your final submission date.

Problem specification

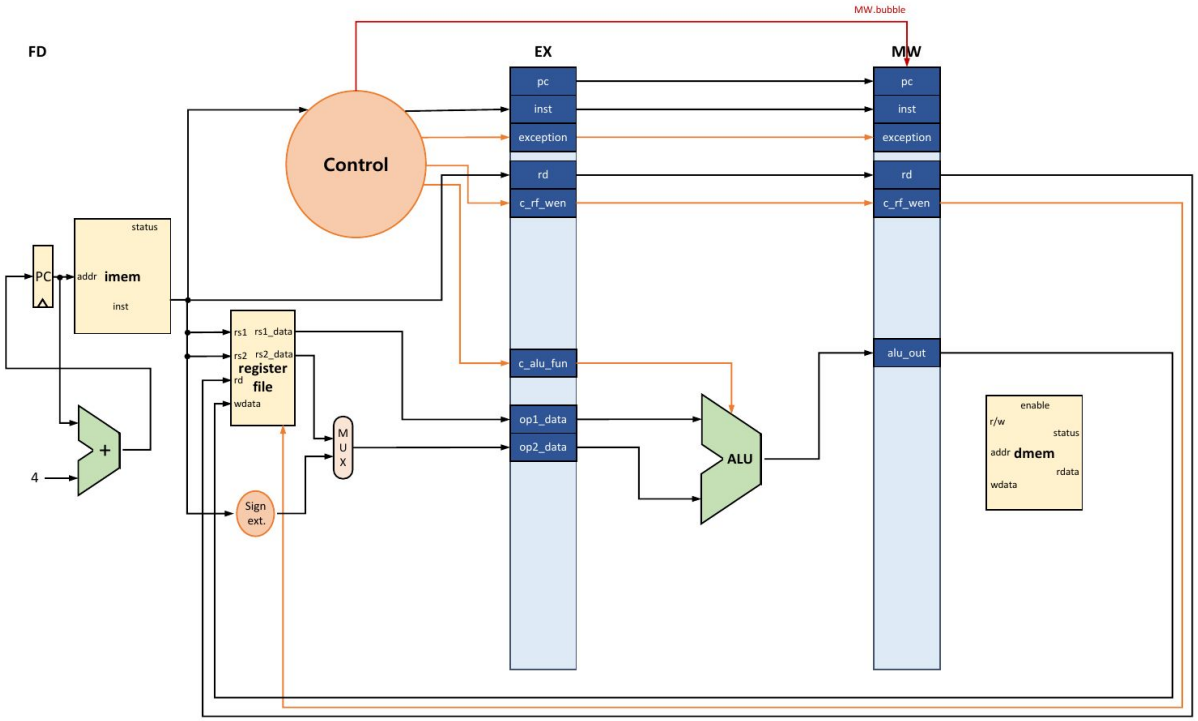
Goal

- Make 3-stage pipelined simulator
  - FD: Fetch/Decode
  - EX: Execute
  - MW: Memory/Write Back

Skeleton Code

- stages.py *you only need to modify this file!
  - Only supports ALU operations
  - No hazard detection, control

snurisc3-skel.pdf



Your Job

- Build missing parts in architecture
  - Branching
  - Memory
  - Data Forwarding

```
def compute(self):
    # DO NOT TOUCH ------------------------------------------------
    # Read out pipeline register values
    self.pc      = FD.reg_pc

    # Fetch an instruction from instruction memory (imem)
    self.inst, status = Pipe.cpu.imem.access(Pipe.CTL.imem_en, self.pc, 0, Pipe.CTL.imem_rw)

    # Handle exception during imem access
    if not status:
        self.exception = EXC_IMEM_ERROR
        self.inst = BUBBLE
    else:
        self.exception = EXC_NONE
    #-------------------------------------------------------------


    # Compute PC + 4 using an adder
    self.pcplus4     = Pipe.cpu.adder_pcplus4.op(self.pc, 4)

    self.rs1         = RISCV.rs1(self.inst)
    self.rs2         = RISCV.rs2(self.inst)
    self.rd          = RISCV.rd(self.inst)

    imm_i            = RISCV.imm_i(self.inst)
    imm_s            = RISCV.imm_s(self.inst)
    imm_b            = RISCV.imm_b(self.inst)
    imm_u            = RISCV.imm_u(self.inst)
    imm_j            = RISCV.imm_j(self.inst)

    self.op1_data    = Pipe.cpu.rf.read(self.rs1)
    rf_rs2_data      = Pipe.cpu.rf.read(self.rs2)
```

do not touch

can modify

Testing

- Run with followng command
  - `python3 snurisc3.py -l [log level] [pyrisc dir]/asm/[testfile]`
  - default log level : 4 (you can check logging options in `snurisc3.py`)

```
(venv) jiyeon@gentoo ~/Comparch/pipeline-lab $ python3 snurisc3.py ../pyrisc/asm/sum100
Loading file ../pyrisc/asm/sum100
------------------------------------------------
0 [FD] 0x80000000: addi   t0, zero, 1
0 [EX] 0x00000000: BUBBLE
0 [MW] 0x00000000: BUBBLE
------------------------------------------------
1 [FD] 0x80000004: addi   t1, zero, 100
1 [EX] 0x80000000: addi   t0, zero, 1
1 [MW] 0x00000000: BUBBLE
------------------------------------------------
2 [FD] 0x80000008: addi   t6, zero, 0
2 [EX] 0x80000004: addi   t1, zero, 100
2 [MW] 0x80000000: addi   t0, zero, 1
```

Testing

- `pyrisc/asm` contains several test cases
  - fib : fibonacci
  - sum100 : sum of integers from 1 to 100
  - forward : data forwarding
  - branch : mispredicted branch instruction
  - loaduse : load-use hazard

Testing

- You can compare your results with `snurisc` or `pyrisc5` from `pyrisc`

```
(venv) jiyeon@gentoo ~/Comparch/pyrisc $ python3 sim/snurisc.py asm/fib
Loading file asm/fib
Execution completed
Registers
=========
zero ($0): 0x00000000    ra ($1):   0x8000000c    sp ($2):   0x80020000    gp ($3):   0x00000000
tp ($4):   0x00000000    t0 ($5):   0x00000000    t1 ($6):   0x00000000    t2 ($7):   0x00000000
s0 ($8):   0x00000000    s1 ($9):   0x00000000    a0 ($10):  0x00000008    a1 ($11):  0x00000000
a2 ($12):  0x00000000    a3 ($13):  0x00000000    a4 ($14):  0x00000000    a5 ($15):  0x00000001
a6 ($16):  0x00000000    a7 ($17):  0x00000000    s2 ($18):  0x00000000    s3 ($19):  0x00000000
s4 ($20):  0x00000000    s5 ($21):  0x00000000    s6 ($22):  0x00000000    s7 ($23):  0x00000000
s8 ($24):  0x00000000    s9 ($25):  0x00000000    s10 ($26): 0x00000000    s11 ($27): 0x00000000
t3 ($28):  0x00000000    t4 ($29):  0x00000000    t5 ($30):  0x00000000    t6 ($31):  0x00000000
162 instructions executed in 162 cycles. CPI = 1.000
Data transfer:    42 instructions (25.93%)
ALU operation:    74 instructions (45.68%)
Control transfer: 46 instructions (28.40%)
```

Testing

- Open the assembly code in asm and you can find what to check for each case.
  - example: forward.s

```
15 # The following program has several situations that require data forwarding.
16 # After successful completion, the x31 register should have the
17 # value of 9.
18
19     .text
20     .align  2
```

```
Execution completed
Registers
=========
zero ($0):  0x00000000    ra ($1):    0x00000000    sp ($2):    0x00000000    gp ($3):    0x00000000
tp ($4):    0x00000000    t0 ($5):    0x00000001    t1 ($6):    0x00000002    t2 ($7):    0x00000003
s0 ($8):    0x00000000    s1 ($9):    0x00000000    a0 ($10):   0x00000000    a1 ($11):   0x00000000
a2 ($12):   0x00000000    a3 ($13):   0x00000000    a4 ($14):   0x00000000    a5 ($15):   0x00000000
a6 ($16):   0x00000000    a7 ($17):   0x00000000    s2 ($18):   0x00000000    s3 ($19):   0x00000000
s4 ($20):   0x00000000    s5 ($21):   0x00000000    s6 ($22):   0x00000000    s7 ($23):   0x00000000
s8 ($24):   0x00000000    s9 ($25):   0x00000000    s10 ($26):  0x00000000    s11 ($27):  0x00000000
t3 ($28):   0x00000000    t4 ($29):   0x00000000    t5 ($30):   0x00000000    t6 ($31):   0x00000001
Memory 0x80010000 - 0x8001ffff
==============================
```

current result of skeleton code.. wrong!

Testing

- Open the assembly code in asm and you can find what to check for each case.
  - example: forward.s

```
(venv) jiyeon@gentoo ~/Comparch/pipeline-lab $ python3 ../pyrisc/sim/snurisc.py ../pyrisc/asm/forward
Loading file ../pyrisc/asm/forward
Execution completed
Registers
=========
zero ($0): 0x00000000    ra ($1):   0x00000000    sp ($2):   0x00000000    gp ($3):   0x00000000
tp ($4):   0x00000000    t0 ($5):   0x00000001    t1 ($6):   0x00000002    t2 ($7):   0x00000003
s0 ($8):   0x00000000    s1 ($9):   0x00000000    a0 ($10):  0x00000000    a1 ($11):  0x00000000
a2 ($12):  0x00000000    a3 ($13):  0x00000000    a4 ($14):  0x00000000    a5 ($15):  0x00000000
a6 ($16):  0x00000000    a7 ($17):  0x00000000    s2 ($18):  0x00000000    s3 ($19):  0x00000000
s4 ($20):  0x00000000    s5 ($21):  0x00000000    s6 ($22):  0x00000000    s7 ($23):  0x00000000
s8 ($24):  0x00000000    s9 ($25):  0x00000000    s10 ($26): 0x00000000    s11 ($27): 0x00000000
t3 ($28):  0x00000000    t4 ($29):  0x00000000    t5 ($30):  0x00000000    t6 ($31):  0x00000009
```

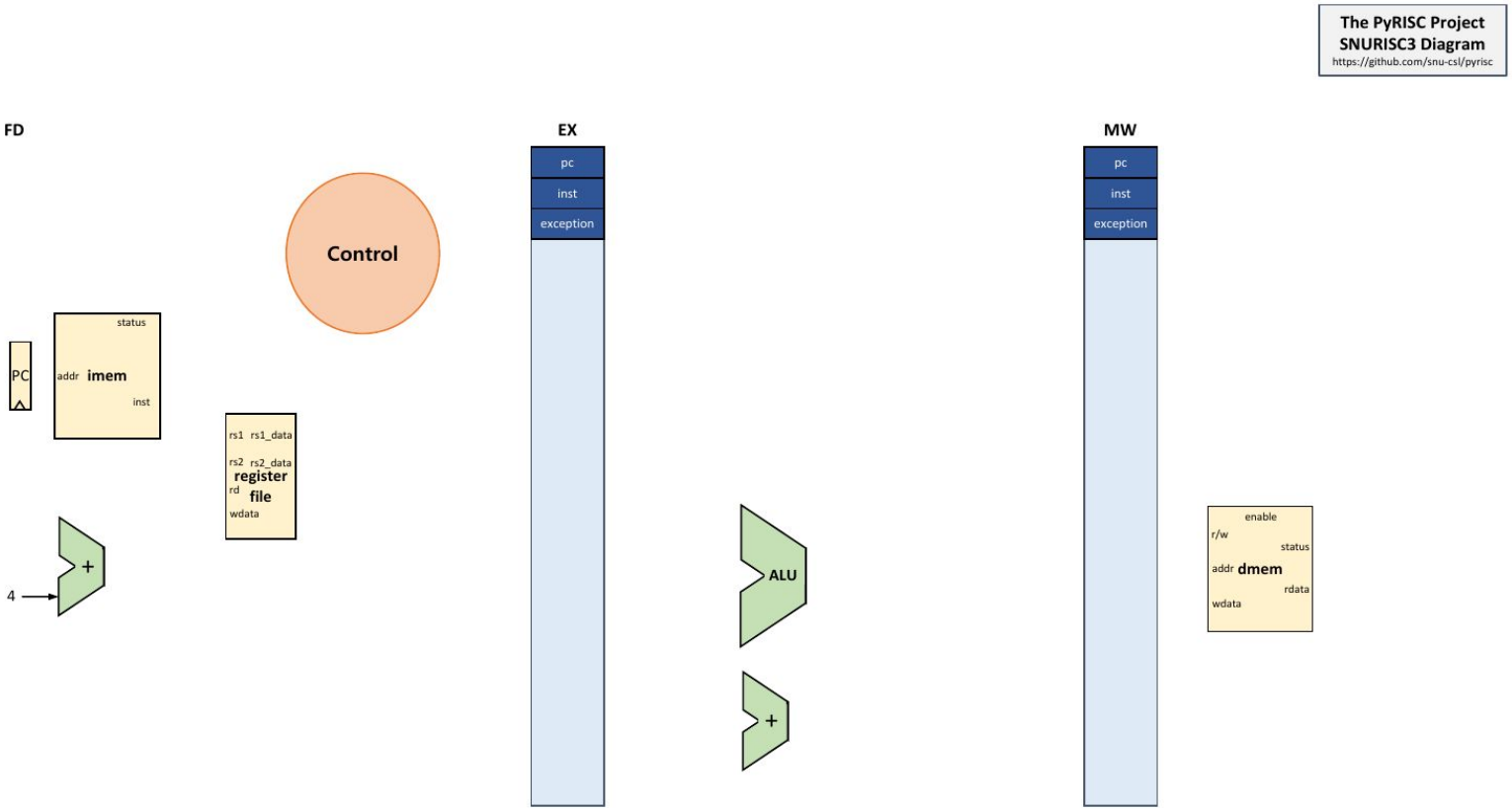
Simulation result of `snurisc.py`

```
Execution completed
Registers
=========
zero ($0): 0x00000000    ra ($1):   0x00000000    sp ($2):   0x00000000    gp ($3):   0x00000000
tp ($4):   0x00000000    t0 ($5):   0x00000001    t1 ($6):   0x00000002    t2 ($7):   0x00000003
s0 ($8):   0x00000000    s1 ($9):   0x00000000    a0 ($10):  0x00000000    a1 ($11):  0x00000000
a2 ($12):  0x00000000    a3 ($13):  0x00000000    a4 ($14):  0x00000000    a5 ($15):  0x00000000
a6 ($16):  0x00000000    a7 ($17):  0x00000000    s2 ($18):  0x00000000    s3 ($19):  0x00000000
s4 ($20):  0x00000000    s5 ($21):  0x00000000    s6 ($22):  0x00000000    s7 ($23):  0x00000000
s8 ($24):  0x00000000    s9 ($25):  0x00000000    s10 ($26): 0x00000000    s11 ($27): 0x00000000
t3 ($28):  0x00000000    t4 ($29):  0x00000000    t5 ($30):  0x00000000    t6 ($31):  0x00000001
Memory 0x80010000 - 0x8001ffff
==============================
```

current result of skeleton code.. wrong!

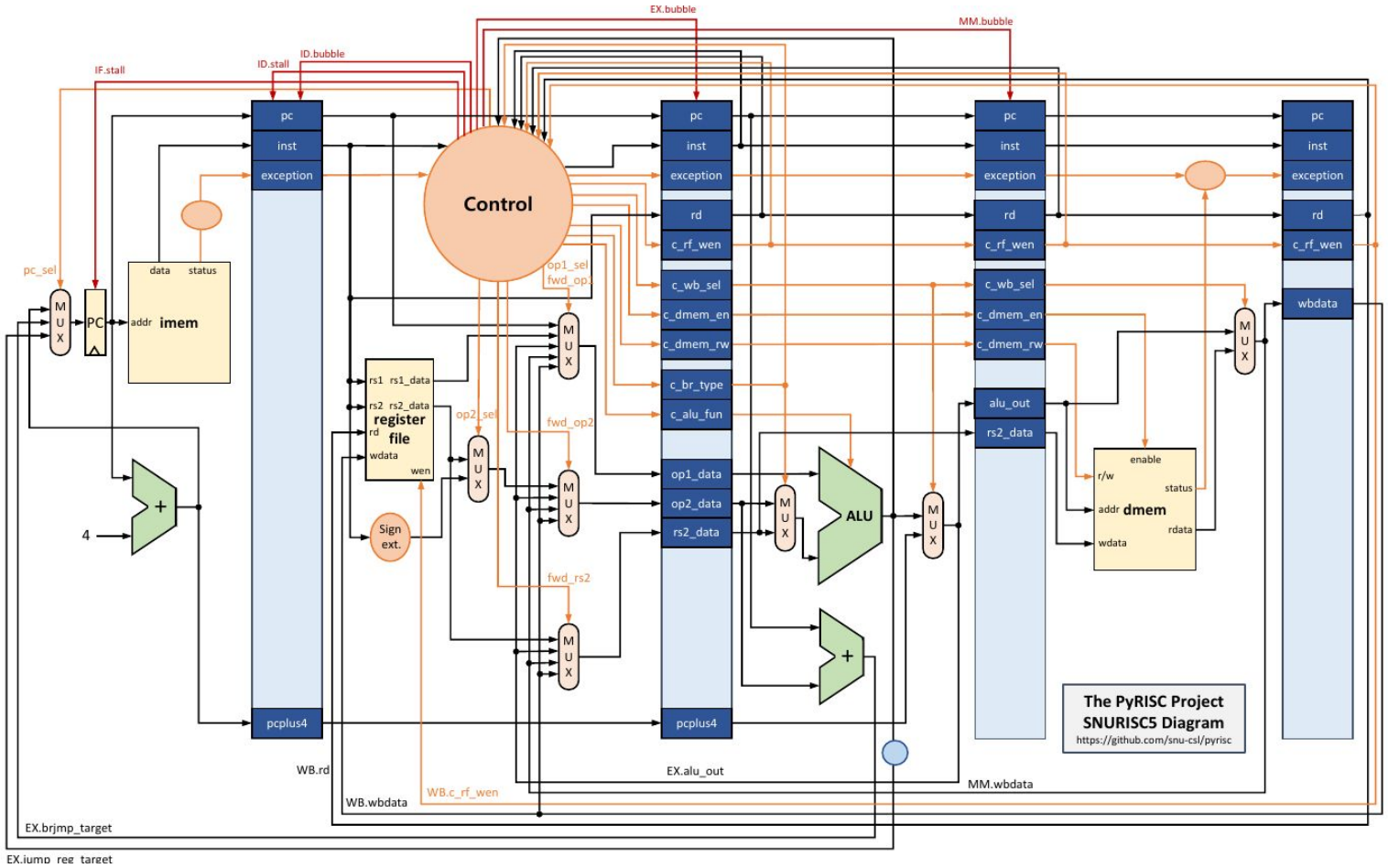
Report

- Fill the diagram and give a brief explanation of your implementation
- Write possible data/control hazards and how did you handle it.



Report

- For diagram, you can see the 5-stage example at https://github.com/snu-csl/pyrisc/blob/master/pipe5/snurisc5.pdf



Evaluation

- Design Report - 40 points

- Implementation - 60 points
  - Check correctness and CPI

    Checking CPI is to determine hazard prevention.

    There will be no extra bonus points for super fast simulator.

Good Luck!

For questions contact

comparch@csap.snu.ac.kr