4190.308: Computer Architecture Lab. 4

Sunmin Jeong, Injae Kang

(snucsl.ta@gmail.com)

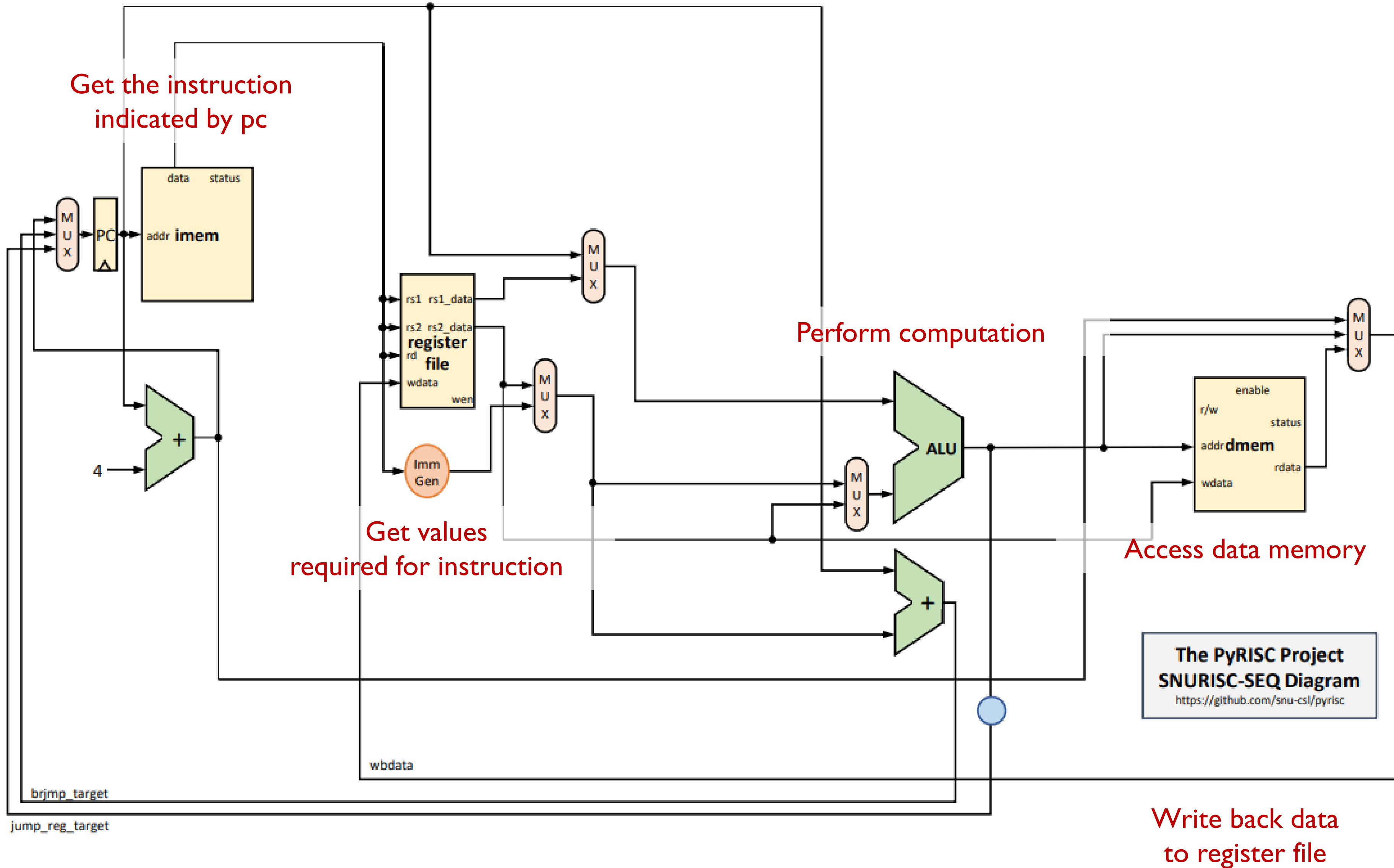
Systems Software & Architecture Lab.

Seoul National University

Fall 2020

SNURISC-SEQ

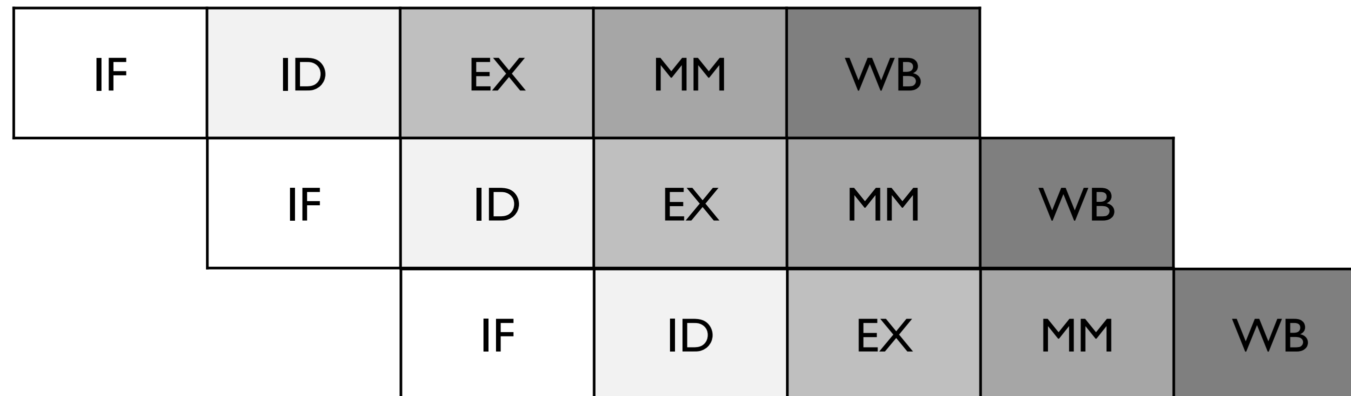
SNURISC-SEQ



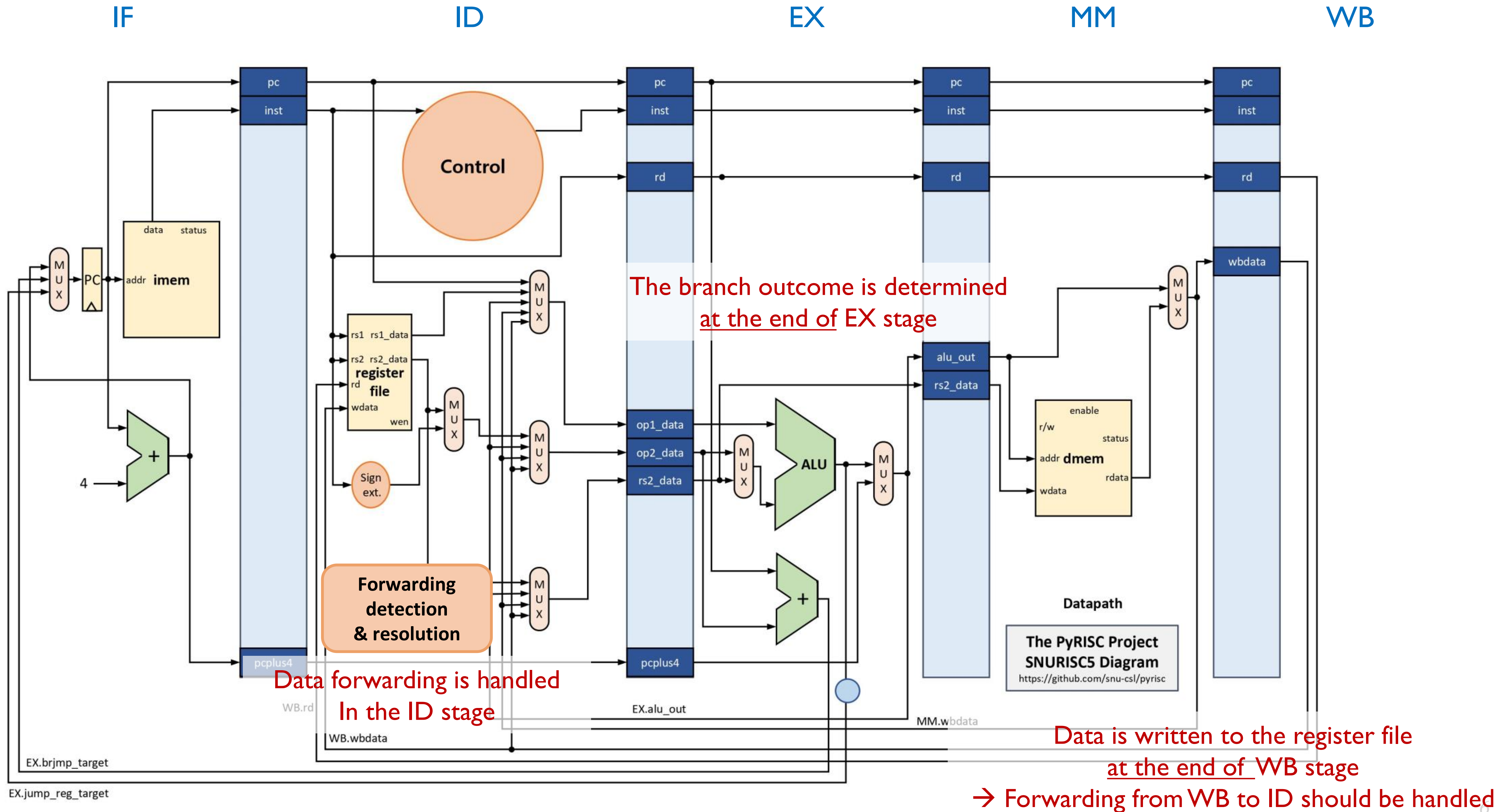
SNURISC5

SNURISC5

- A 5-stage pipelined RISC-V Simulator
- It consists of
  - IF: Instruction fetch
  - ID: Instruction decode & register read
  - EX: Execute
  - MM: Memory access
  - WB: Writeback



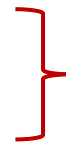
SNURISC5



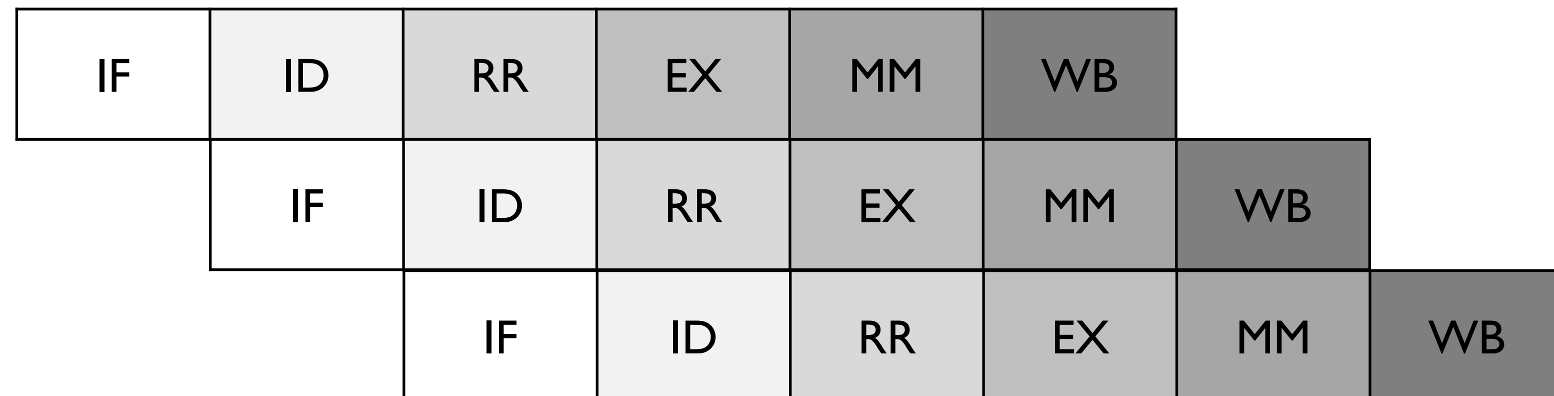
SNURISC6

SNURISC6

- A 6-stage pipelined RISC-V Simulator
- It consists of
  - IF
  - ID : Instruction decode
  - RR : Register read
  - EX
  - MM
  - WB



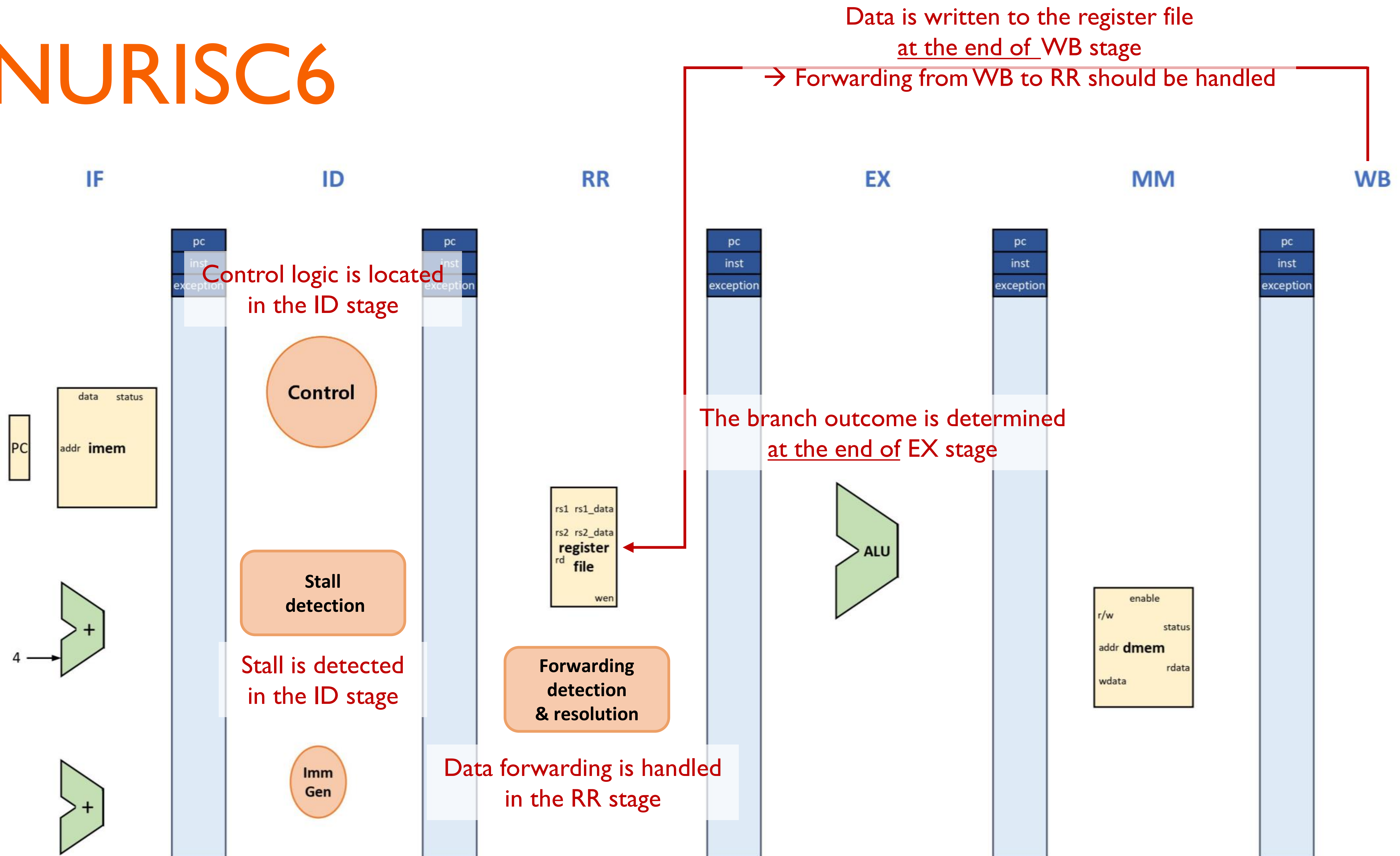
ID stage in traditional 5-stage pipeline
is divided into two stages



SNURISC6

Stage	Task
IF	Fetch an instruction from *imem* (instruction memory)
ID	Decode the instruction Prepare immediate values
RR	Read the register file
EX	Perform arithmetic/logical computation Determine the branch outcome
MM	Access *dmem* (data memory), if necessary
WB	Write back the result to the register file

SNURISC6



SNURISC6

- Overall simulator architecture
  - snurisc6.py: It parses arguments from the user and controls the overall simulation
  - program.py: It loads the contents of the input RISC-V executable file to *imem*
  - pipe.py: It controls the actual execution of the simulation
  - stage.py: It contains the datapath information for each stage and the control logic
  - components.py: It has various hardware components such as RegisterFile, Register, Memory, ALU, and Adder
  - isa.py: It has definition of each instructions and decoding logic for RISC-V instruction set
  - consts.py: It defines various constants used throughput the simulator

SNURISC6

- class Pipe (in pipe.py)

```
def set_stages(cpu, stages):
    Pipe.cpu = cpu
    Pipe.stages = stages
    Pipe.IF = stages[S_IF]
    Pipe.ID = stages[S_ID]
    Pipe.RR = stages[S_RR]
    Pipe.EX = stages[S_EX]
    Pipe.MM = stages[S_MM]
    Pipe.WB = stages[S_WB]
```

Each points to the corresponding objects of IF, ID, RR, EX, MM and WB classes

```
def run(entry_point):
    IF.reg_pc = entry_point
    while True:
        Pipe.WB.compute()
        Pipe.MM.compute()
        Pipe.EX.compute()
        Pipe.RR.compute()
        Pipe.ID.compute()
        Pipe.IF.compute()
        # Update states
        Pipe.IF.update()
        Pipe.ID.update()
        Pipe.RR.update()
        Pipe.EX.update()
        Pipe.MM.update()
        ok = Pipe.WB.update()

        if not ok:
            break
```

Reverse order due to dependence of hazard/forwarding detection

Manipulation of signals using some combinational logic performed inside of the stage

Contents of the pipeline registers are updated

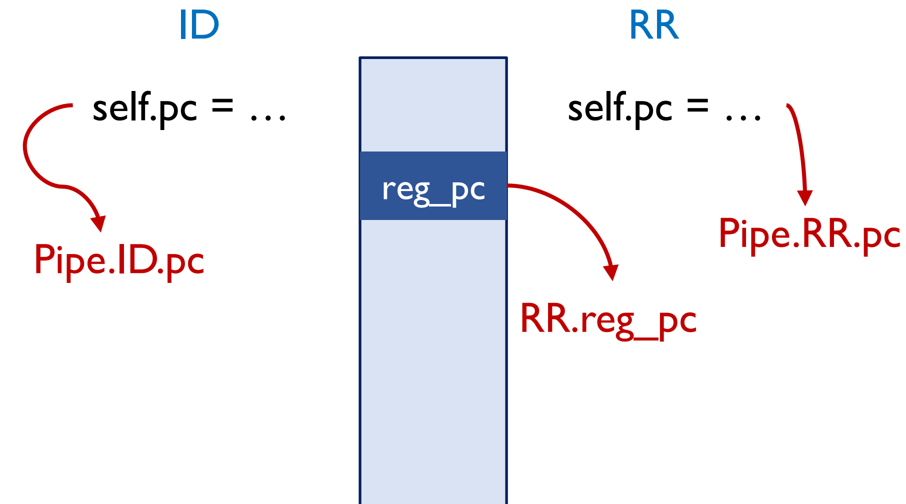
SNURISC6

- Naming convention
  - Pipeline registers
    - Implemented as class variables
      → referenced as [class name].[variable name]
    - Prefix 'reg_' is added

    e.g., RR.*reg_pc*: pipeline register '*reg_pc*' between ID and RR stage

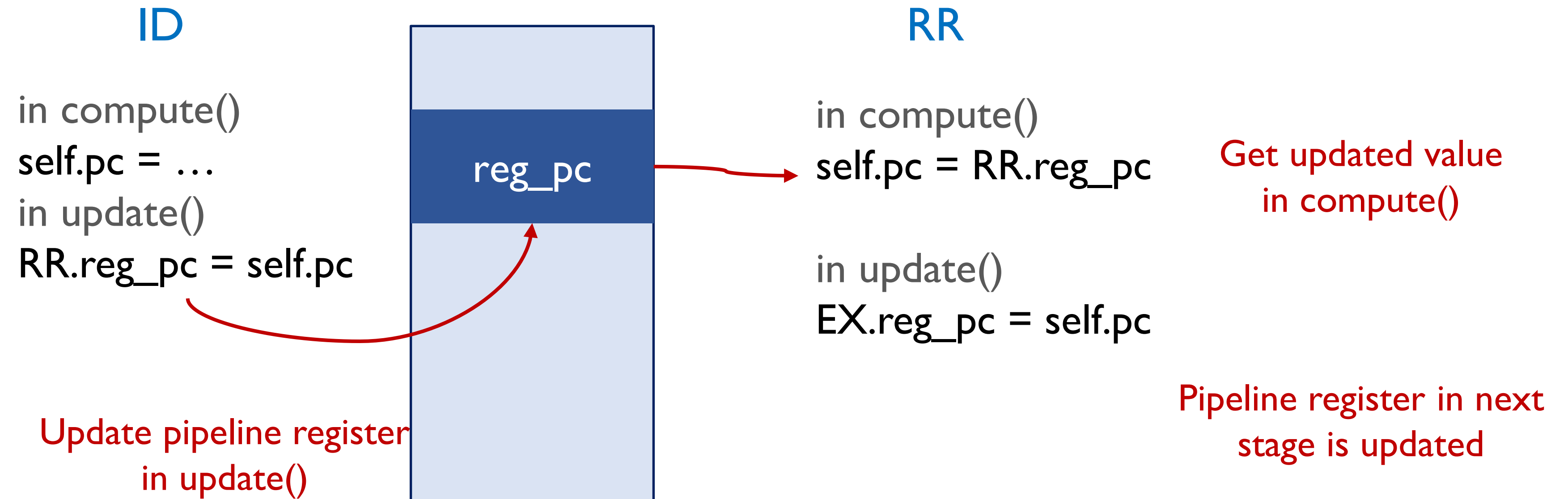
  - Internal signals within a stage
    - Implemented as instance variables
      → referenced as self.[variable name] or Pipe.[class name].[variable name]

    e.g., *self.pc* defined in the ID stage can be referenced as *Pipe.ID.pc*



SNURISC6

- Usage conventions
  - When you want to pass the pipeline register to next stage,



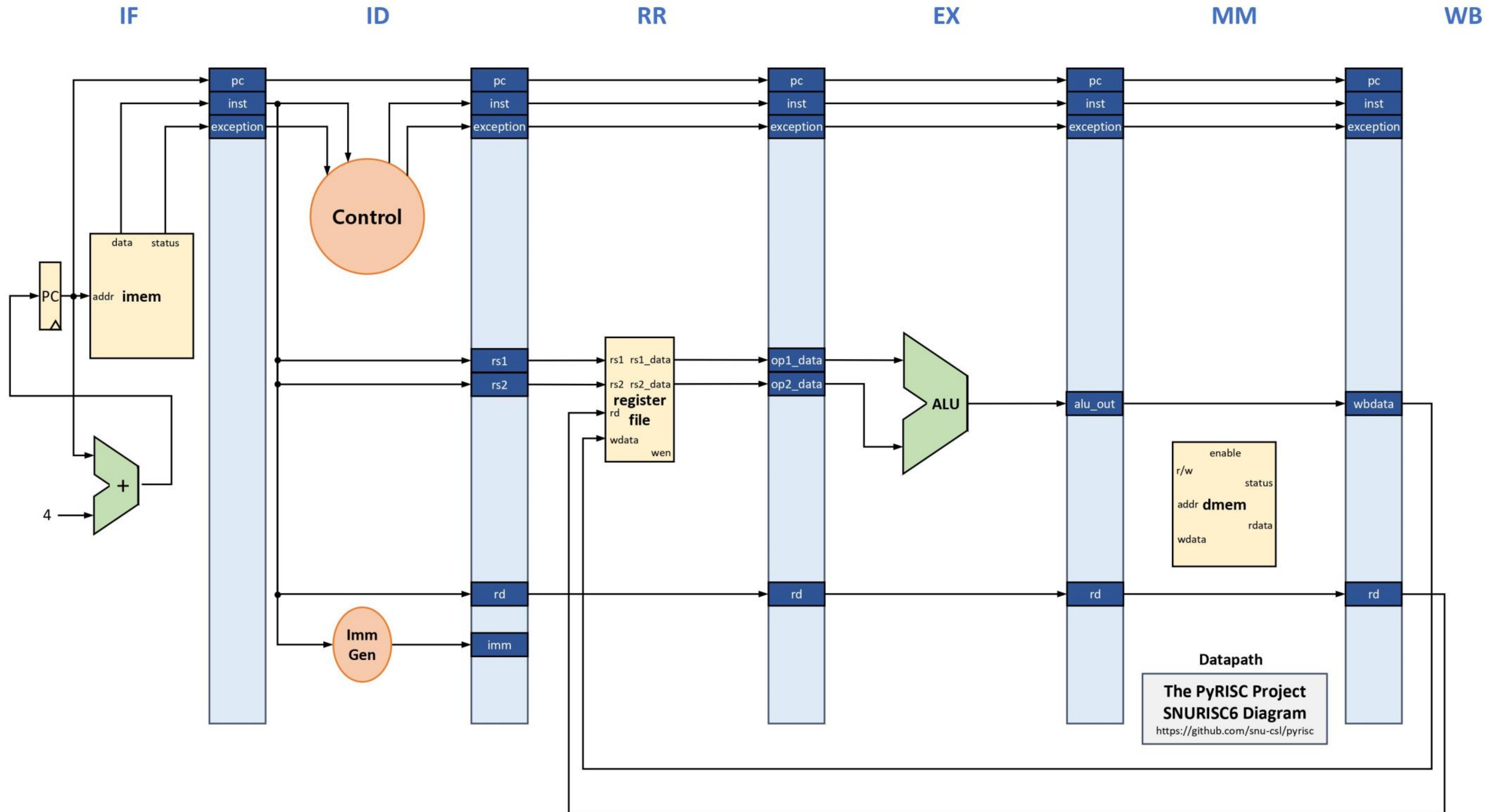
SNURISC6

- For more detailed information, refer to SNURISC5
  - pyrisc/pipe5/README.md
  - pyrisc/pipe5/GUIDE.md

Specification

Skeleton

Currently, it just supports some of ALU operations without any hazard detection and control logic



Part 1 (30 points)

Implementing a 6-stage pipelined RISC-V processor simulator

- It should accept the same RISC-V executable file accepted by SNURISC5
- Its values of registers and data memory should be same with SNURISC5
- Data forwarding should be fully implemented

Part 1 (30 points)

Implementing a 6-stage pipelined RISC-V processor simulator

- When data forwarding can't solve the dependency among instructions, the pipeline should be stalled (e.g., load-use hazard)
- You should minimize the number of stalled cycles (e.g., 1 cycle stall for load-use hazard)

Part 1 (30 points) – example (1)

Data forwarding	1	2	3	4	5	6	7	8	9
`add t0, t1, t2`	IF	ID	RR	EX	MM	WB			
`add t3, t0, t3`		IF	ID	RR	EX	MM	WB		

Data forwarding is required
&
Data is determined at EX stage

Data forwarding occurs at the end of EX stage

Part 1 (30 points) – example (2)

Load-use hazard	1	2	3	4	5	6	7	8	9
lw t0, 0(sp)	IF	ID	RR	EX	MM	WB			
add t3, t0, t3		IF	ID	ID	RR	EX	MM	WB	
...			IF	IF	ID	RR	EX	MM	WB

Data forwarding occurs at the end of MM stage

Load-use hazard detected → Stall

Load-use hazard

&

Data forwarding is required

&

Data is determined at MM stage

Part 2 (40 points)

Implementing the "always-taken" branch prediction scheme

- Branch prediction should be performed in the IF stage
- Branch outcome is determined in the EX stage
- When the prediction was wrong, you need to cancel the incorrectly fetched instructions and forward the correct value for next pc (the address of the original branch instruction + 4)

Part 2 (40 points)

Implementing the "always-taken" branch prediction scheme

- You should use the same prediction scheme for the *jal* instruction

- *jalr* instruction should be handled as "always-not-taken" scheme

Part 2 (40 points) – example (1)

branch instruction → "always taken" branch prediction

Taken branch	1	2	3	4	5	6	7	8	9
`beq  t0, t0, L1`	IF	ID	RR	EX	MM	WB			
`add  t1, t2, t3`									
`addi t1, t1, -1`									
`sub  t4, t1, t2`									
…									
…									
`L1: sub  t5, t6, t7`		IF	ID	RR	EX	MM	WB		
`xori t5, t5,  1`			IF	ID	RR	EX	MM	WB	
`add  t6, t6, t5`				IF	ID	RR	EX	MM	WB
`addi t6, t6, 10`					IF	ID	RR	EX	MM

Part 2 (40 points) – example (2)

branch instruction → "always taken" branch prediction

Not-Taken branch	1	2	3	4	5	6	7	8	9
`bne t0, t0, L1`	IF	ID	RR	EX	MM	WB			
`add t1, t2, t3`					IF	ID	RR	EX	MM
`addi t1, t1, -1`						IF	ID	RR	EX
`sub t4, t1, t2`							IF	ID	RR
…									
…									
`L1: sub t5, t6, t7`		IF	ID	RR	BUBBLE	BUBBLE	BUBBLE		
`xori t5, t5, 1`			IF	ID	BUBBLE	BUBBLE	BUBBLE	BUBBLE	
`add t6, t6, t5`				IF	BUBBLE	BUBBLE	BUBBLE	BUBBLE	BUBBLE
`addi t6, t6, 10`									

MISPREDICTED

Part 2 (40 points) – example (3)

"always-taken" branch prediction & never mispredicted

jal instruction	1	2	3	4	5	6	7	8	9
`jal  ra, L1`	IF	ID	RR	EX	MM	WB			
`add  t1, t2, t3`									
`addi t1, t1, -1`									
`sub  t4, t1, t2`									
…									
…									
`L1: sub  t5, t6, t7`		IF	ID	RR	EX	MM	WB		
`xori t5, t5,  1`			IF	ID	RR	EX	MM	WB	
`add  t6, t6, t5`				IF	ID	RR	EX	MM	WB
`addi t6, t6, 10`					IF	ID	RR	EX	MM

Part 2 (40 points) – example (4)

"Always-not-taken" prediction

jalr instruction	1	2	3	4	5	6	7	8	9
`jalr x0, 0(ra)`	IF	ID	RR	EX	MM	WB			
`add  t1, t2, t3`		IF	ID	RR	BUBBLE	BUBBLE	BUBBLE		
`addi t1, t1, -1`			IF	ID	BUBBLE	BUBBLE	BUBBLE	BUBBLE	
`sub  t4, t1, t2`				IF	BUBBLE	BUBBLE	BUBBLE	BUBBLE	BUBBLE
…									
(ra contains L1) …									
`L1: sub  t5, t6, t7`					IF	ID	RR	EX	MM
`xori t5, t5,  1`						IF	ID	RR	EX
`add  t6, t6, t5`							IF	ID	RR
`addi t6, t6, 10`									

MISPREDICTED

Part 3 (30 points)

Design document (each 10 points)

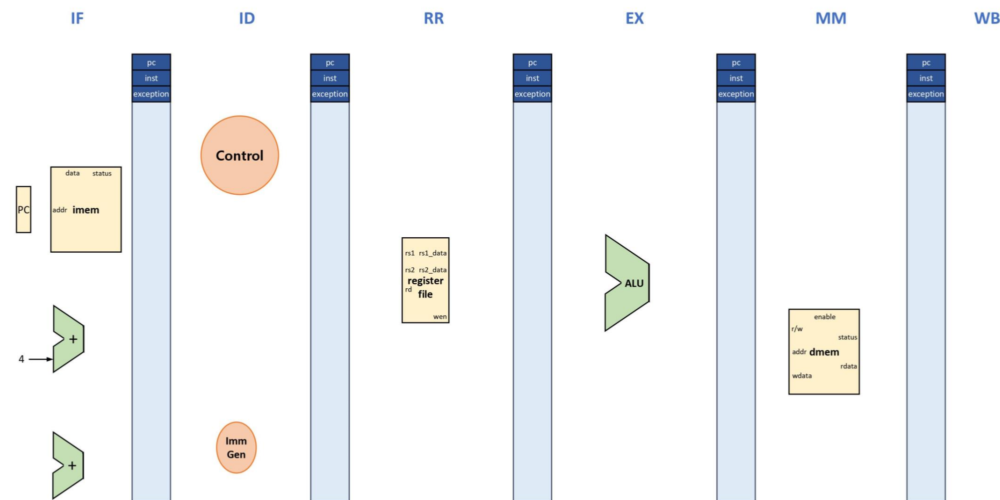
1. What does the overall pipeline architecture look like?
2. (About Part 1) When do data hazards occur and how do you deal with them?
3. (About Part 2) When do control hazards occur and how do you deal with them with the always-taken branch prediction scheme?

Part 3 (30 points)

Design document (each 10 points)

1. What does the overall pipeline architecture look like?

- Complete the diagram in snurisc6-design.pdf according to your pipeline design
- A hand-drawn diagram is OK
- Take a picture of your diagram and attach it in your design document



Part 3 (30 points)

Design document (each 10 points)

2. When do data hazards occur and how do you deal with them?
   - Show all the possible cases when data hazards can occur and your solutions to them
   - What hardware has been added to detect and resolve data hazards and how does it work?

Part 3 (30 points)

Design document (each 10 points)

3. When do control hazards occur and how do you deal with them with always-taken branch prediction scheme?
   - Show all the possible cases when control hazards can occur and your solutions to them
   - What hardware has been added to detect and resolve control hazards and how does it work?

Specification

- Your task is to modify the stages.py file and make it work correctly for any combination of instructions

- You can test your simulator with RISC-V executable files in *pyrisc/asm/*

```
$ ./snurisc6.py -l 4 [path_to_pyrisc]/pyrisc/asm/sum100
```

Specification

- You should not change any files other than stages.py
- Your stages.py file should not contain any print() function even in comment lines
- Your simulator should minimize the number of stalled cycles

Specification

- Your code should finish within a reasonable number of cycles
  - If your simulator runs beyond the predefined threshold, you will get TIMEOUT error

- The number of submissions to the server will be limited to 50 times

Submission

- Due: 11:59PM, December 16 (Wednesday)
  - 25% of the credit will be deducted for every single day delay
  - This is the final project → feel free to use your slip days ☺

- Submit only the stages.py file to the submission server
- Also, submit the design document(in PDF file only) to the submission server

Thank you!

- If you have any question about the assignment, feel free to ask us in email or KakaoTalk

- This file will be uploaded after the lab session☺