# CSED311 Lab3: Multi-Cycle CPU

**Seonggon Kim**

sungonuni@postech.ac.kr

Contact the TAs at cs311-2025ta@postech.ac.kr

## Contents

## Objectives

- Understand why a multi-cycle CPU is better than single-cycle implementation
- Design and implement a multi-cycle CPU, which has its own datapath and control unit

# Why Multi-Cycle CPU?

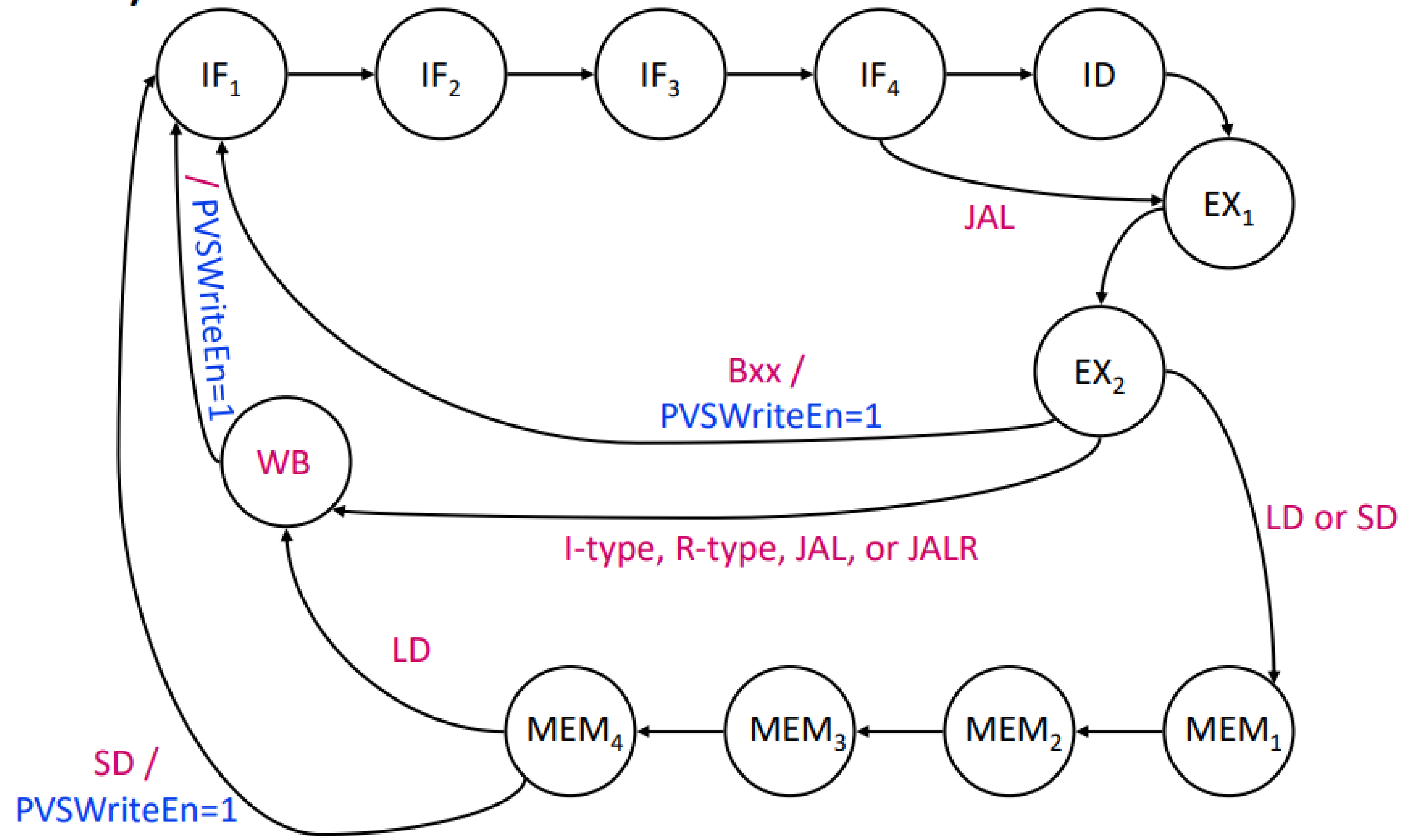
- Problem on single-cycle CPU: underutilization of resources (ALU, memory, register file, etc.)
- **Solution**: use higher clock frequency and allocate a different number of cycles for each instruction type

| | |
|---|---|
| Memory units (read or write): | 200 ps |
| ALU (add op) : | 100 ps |
| Register file (read or write): | 50 ps |
| Other combinational logic: | 0 ps |

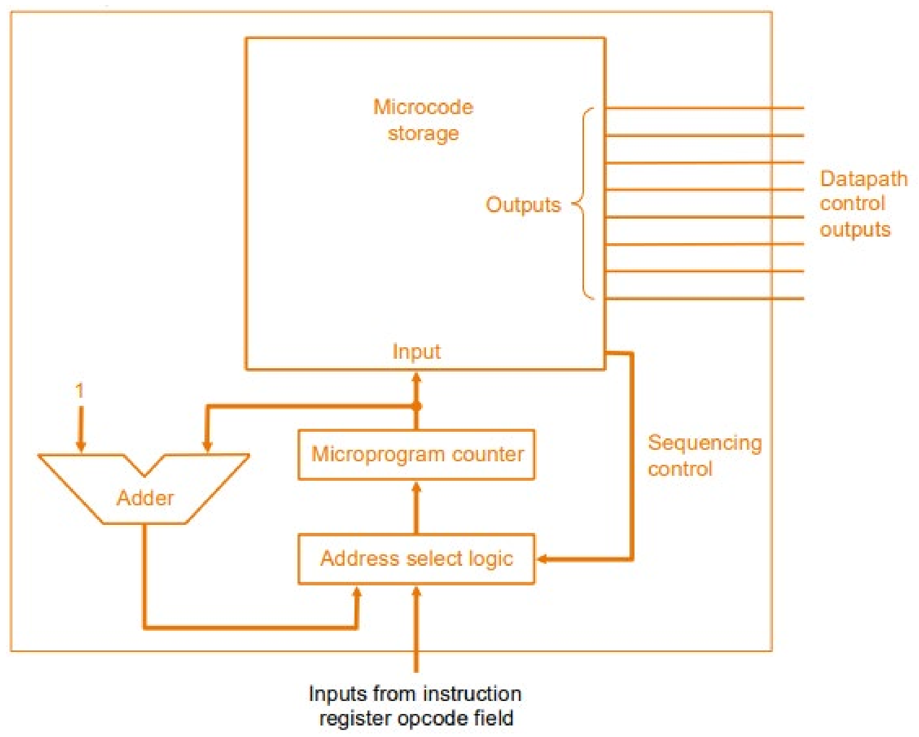
| Steps | IF | ID | EX | MEM | WB | Delay |
|---|---|---|---|---|---|---|
| Resources | mem | RF | ALU | mem | RF | |
| R-type | 200 | 50 | 100 | | 50 | 400 |
| I-type | 200 | 50 | 100 | | 50 | 400 |
| LD | 200 | 50 | 100 | 200 | 50 | 600 |
| SD | 200 | 50 | 100 | 200 | | 550 |
| Bxx | 200 | 50 | 100 | | | 350 |
| JAL | 200 | | 100 | | 50 | 350 |
| JALR | 200 | 50 | 100 | | 50 | 400 |

# Multi-Cycle CPU (Finite State Machine)

Mealy FSM

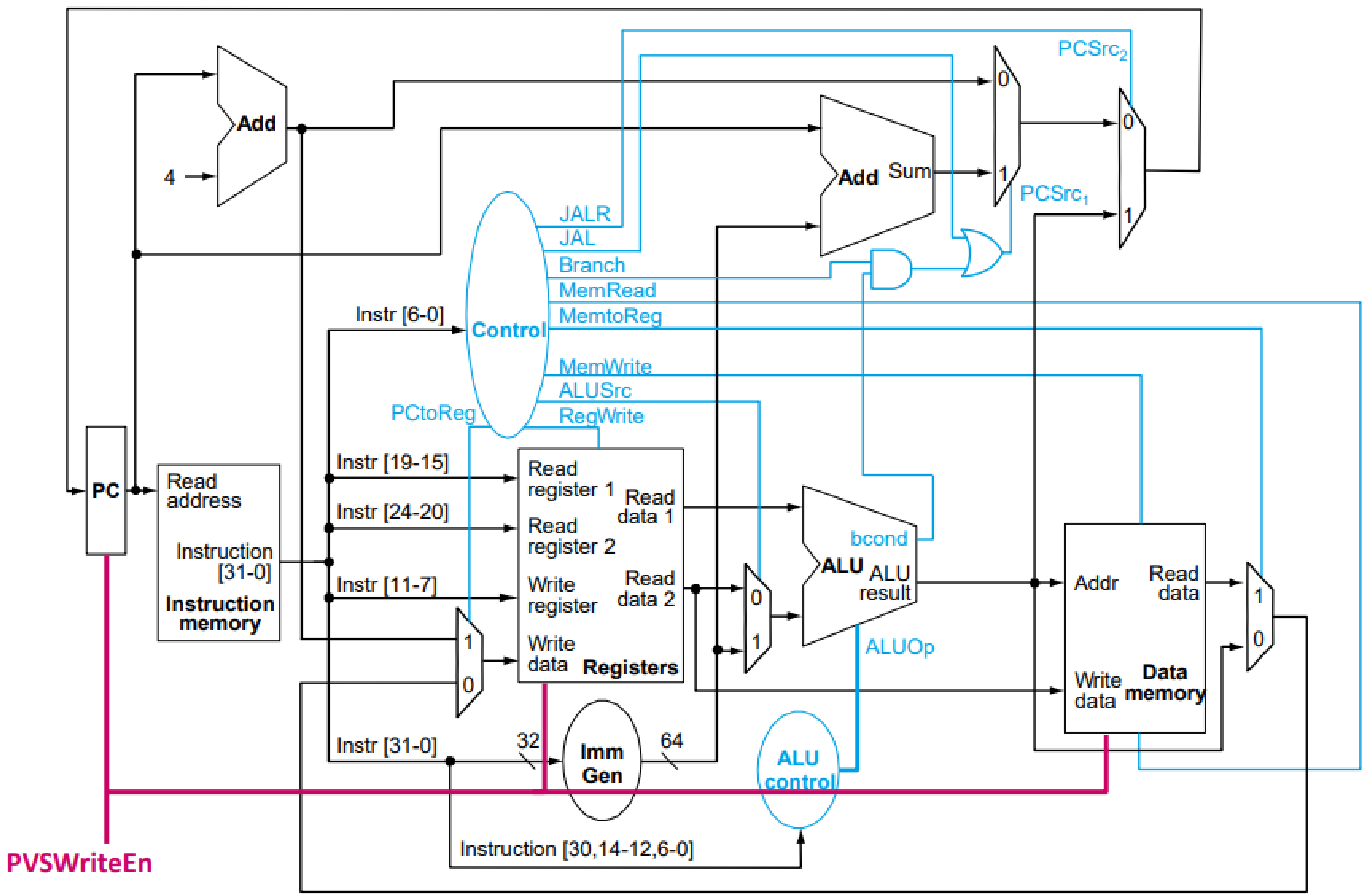


# Multi-Cycle CPU (Microcode Controller)

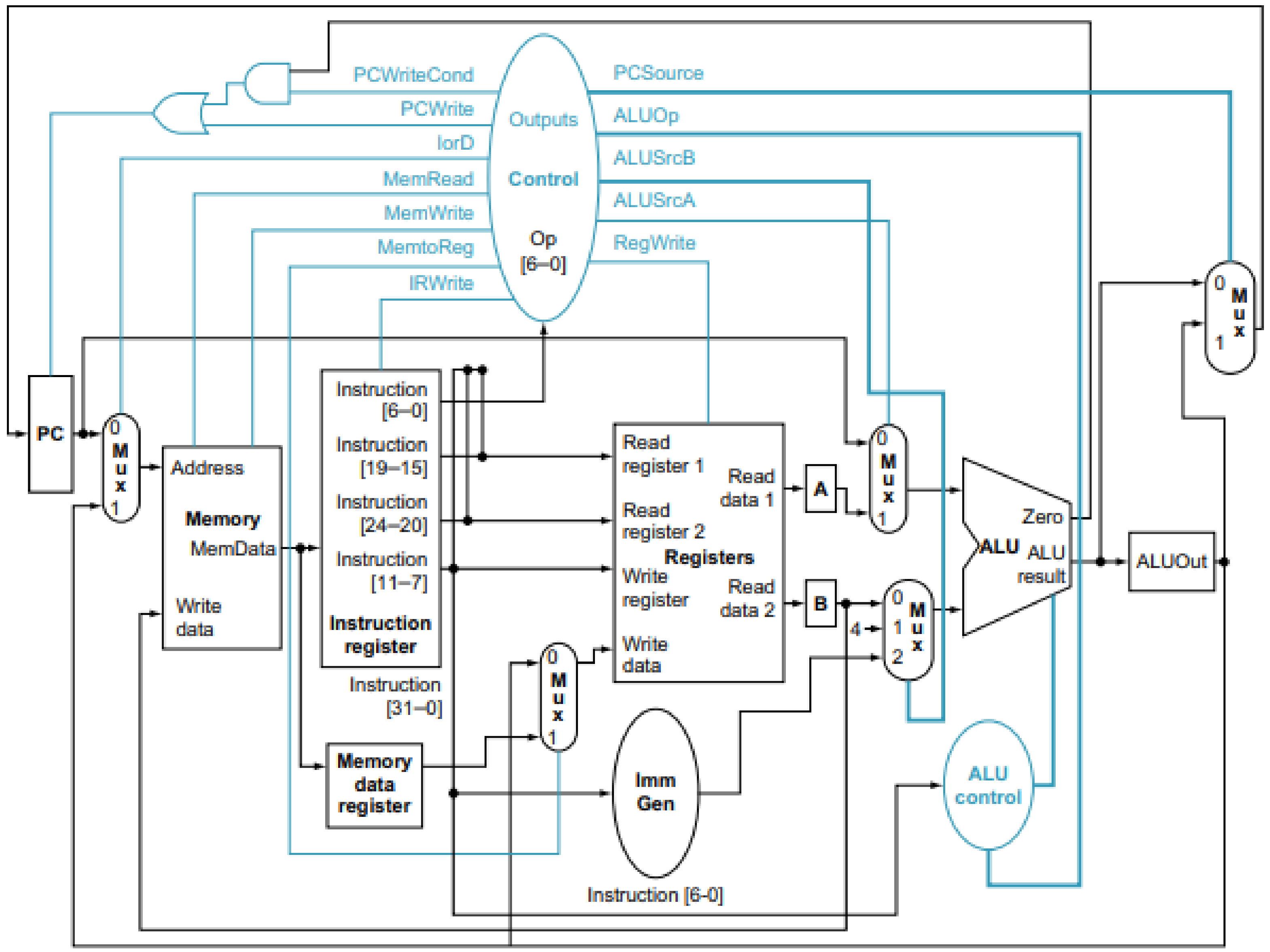


| State label | Control flow | Conditional targets | | | | | |
|---|---|---|---|---|---|---|---|
| | | R/I-type | LD | SD | Bxx | JALR | JAL |
| $IF_1$ | next | - | - | - | - | - | - |
| $IF_2$ | next | - | - | - | - | - | - |
| $IF_3$ | next | - | - | - | - | - | - |
| $IF_4$ | go to | ID | ID | ID | ID | ID | $EX_1$ |
| ID | next | - | - | - | - | - | |
| $EX_1$ | next | - | - | - | - | - | - |
| $EX_2$ | go to | WB | $MEM_1$ | $MEM_1$ | $IF_1$ | WB | WB |
| $MEM_1$ | next | | - | - | | | |
| $MEM_2$ | next | | - | - | | | |
| $MEM_3$ | next | | - | - | | | |
| $MEM_4$ | go to | | WB | $IF_1$ | | | |
| WB | go to | $IF_1$ | $IF_1$ | | | $IF_1$ | $IF_1$ |
| CPI | | 8 | 12 | 11 | 7 | 8 | 7 |

# Single-Cycle CPU (Datapath w/o Resource reuse)



## Multi-Cycle CPU (Datapath w/ Resource Reuse)



## Multi-Cycle CPU

- Details for multi-cycle CPU are given in the lecture note and textbook
  - Appendix C can also be helpful
  - Link: https://www.elsevier.com/books-and-journals/book-companion/9780128203316

## Assignment

- Use Verilator
- Implement a multi-cycle RISC-V CPU (RV32I)
  - Multi-cycle CPU
    - Datapath
      - ALU
      - Register file
    - Control unit
      - Microcode controller
      - Generate the control signals used in the datapath
  - You can use FSM with either 1 cycle or more cycles for each stage

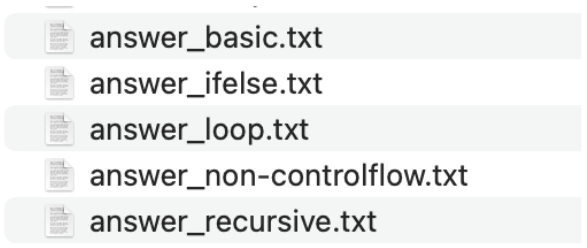
## Assignment

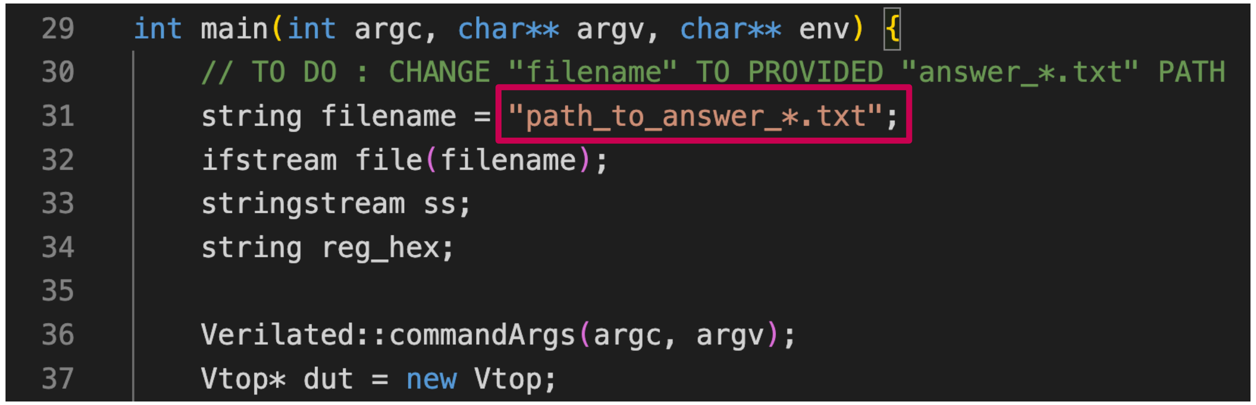
- Skeleton code updated
  - **Top.v, cpu.v, RegisterFile.v, and memory.v are updated**
  - You can take other modules (e.g., ALU) from your single-cycle CPU to implement the multicycle CPU
  - Other modules (**add more or change if you need**)
- Testbench
  - **Simulation code**
    - tb_top.cpp
  - **Instruction codes** for Verilog RTL (.txt)
    - basic_ripes.txt, non-controlflow_mem.txt, loop_mem.txt, ifelse_mem.txt, recursive_mem.txt
  - **Assembly codes** for Ripes (.asm)
    - basic_ripes.asm, non-controlflow_mem.asm, loop_mem.asm,  ifelse_mem.asm, recursive _mem.asm
- Makefile

POSTECH

# Assignment (cont'd)

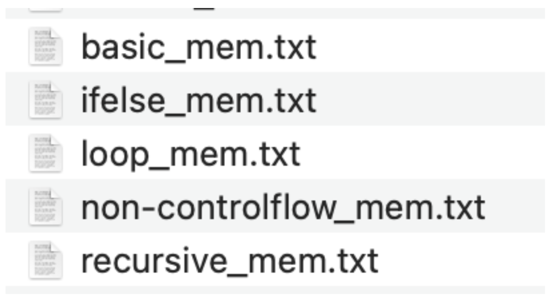
- tb_top.cpp

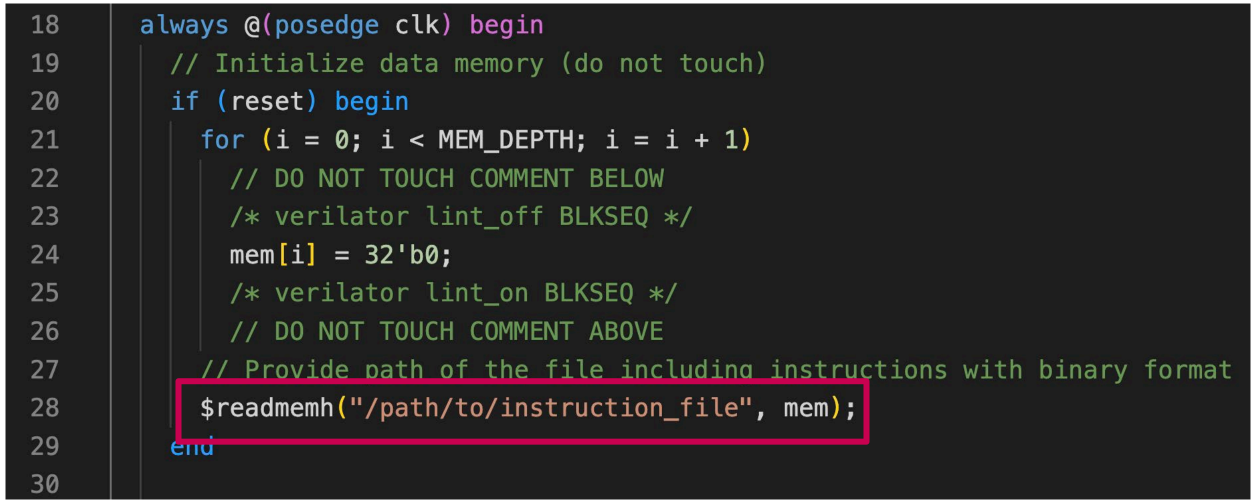

answer_basic.txt
answer_ifelse.txt
answer_loop.txt
answer_non-controlflow.txt
answer_recursive.txt


```
int main(int argc, char** argv, char** env) {
    // TO DO : CHANGE "filename" TO PROVIDED "answer_*.txt" PATH
    string filename = "path_to_answer_*.txt";
    ifstream file(filename);
    stringstream ss;
    string reg_hex;

    Verilated::commandArgs(argc, argv);
    Vtop* dut = new Vtop;
```

- memory.v


basic_mem.txt
ifelse_mem.txt
loop_mem.txt
non-controlflow_mem.txt
recursive_mem.txt

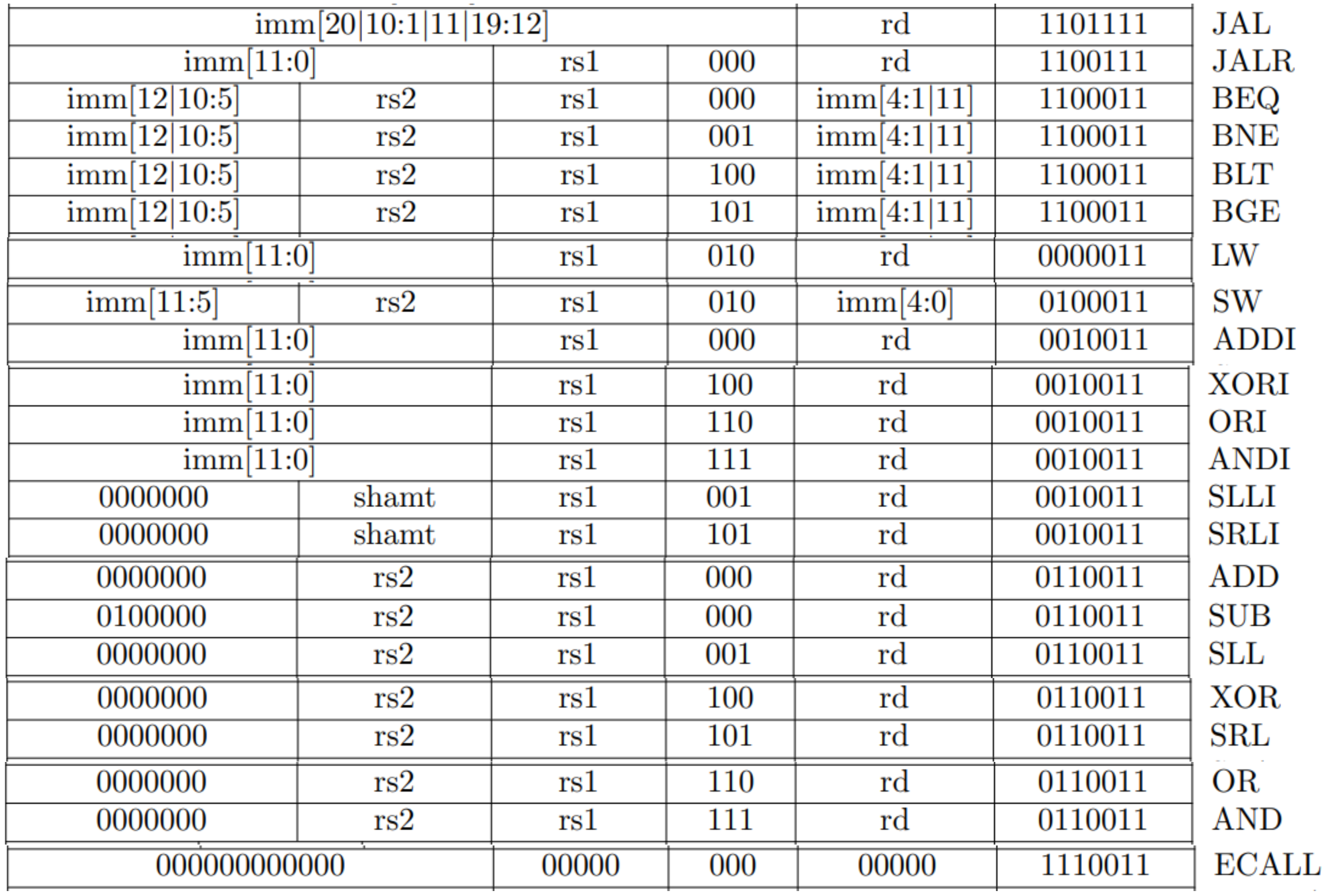

```
always @(posedge clk) begin
  // Initialize data memory (do not touch)
  if (reset) begin
    for (i = 0; i < MEM_DEPTH; i = i + 1)
      // DO NOT TOUCH COMMENT BELOW
      /* verilator lint_off BLKSEQ */
      mem[i] = 32'b0;
      /* verilator lint_on BLKSEQ */
      // DO NOT TOUCH COMMENT ABOVE
    // Provide path of the file including instructions with binary format
    $readmemh("/path/to/instruction_file", mem);
  end

```

Please add the path of *.txt file in ./student_tb here.

# Assignment (cont'd)

- Implement the same instructions required in the single-cycle CPU



| | | | | | | |
|---|---|---|---|---|---|---|
| imm[20\|10:1\|11\|19:12] | | | | rd | 1101111 | JAL |
| imm[11:0] | | rs1 | 000 | rd | 1100111 | JALR |
| imm[12\|10:5] | rs2 | rs1 | 000 | imm[4:1\|11] | 1100011 | BEQ |
| imm[12\|10:5] | rs2 | rs1 | 001 | imm[4:1\|11] | 1100011 | BNE |
| imm[12\|10:5] | rs2 | rs1 | 100 | imm[4:1\|11] | 1100011 | BLT |
| imm[12\|10:5] | rs2 | rs1 | 101 | imm[4:1\|11] | 1100011 | BGE |
| imm[11:0] | | rs1 | 010 | rd | 0000011 | LW |
| imm[11:5] | rs2 | rs1 | 010 | imm[4:0] | 0100011 | SW |
| imm[11:0] | | rs1 | 000 | rd | 0010011 | ADDI |
| imm[11:0] | | rs1 | 100 | rd | 0010011 | XORI |
| imm[11:0] | | rs1 | 110 | rd | 0010011 | ORI |
| imm[11:0] | | rs1 | 111 | rd | 0010011 | ANDI |
| 0000000 | shamt | rs1 | 001 | rd | 0010011 | SLLI |
| 0000000 | shamt | rs1 | 101 | rd | 0010011 | SRLI |
| 0000000 | rs2 | rs1 | 000 | rd | 0110011 | ADD |
| 0100000 | rs2 | rs1 | 000 | rd | 0110011 | SUB |
| 0000000 | rs2 | rs1 | 001 | rd | 0110011 | SLL |
| 0000000 | rs2 | rs1 | 100 | rd | 0110011 | XOR |
| 0000000 | rs2 | rs1 | 101 | rd | 0110011 | SRL |
| 0000000 | rs2 | rs1 | 110 | rd | 0110011 | OR |
| 0000000 | rs2 | rs1 | 111 | rd | 0110011 | AND |
| 000000000000 | | 00000 | 000 | 00000 | 1110011 | ECALL |

## Modularization

- Modularize the main CPU structure (strongly recommended)
  - Datapath
    - ALU
    - Register file
  - Control unit
    - Microcode controller
  - Etc.
    - MUX, …
- **You may modify the interfaces of some of the modules but keep them well modularized**
- **Keep one module in one Verilog file (otherwise, Verilator may not work well)**
- **Match file name with module name (o therwise, Verilator may not work well)**
- **You may modify the interfaces of some of the modules (except top.v, cpu.v)**

## Evaluation Criteria

- Source code
  - The score will be **calculated based on the final register values (x1-x31)** of the Verilog RTL **after test cases for evaluation are executed (same as single-cycle CPU)**
    - **You can check the correct register values with single-cycle Ripes simulation (Ripes doesn’t support multi-cycle simulation)**
  - Implementation guidelines
    - Your control unit should be a well-implemented state machine
      - Each state should generate its control signals
    - All storage units (registers, PC, etc.) must be updated only at the clock’s positive edges
    - Your code should have resource reuse, which affects your control unit design
      - E.g.) Combining “PC + 1” logic with the ALU
    - **If you don’t follow guidelines, you will get penalty**

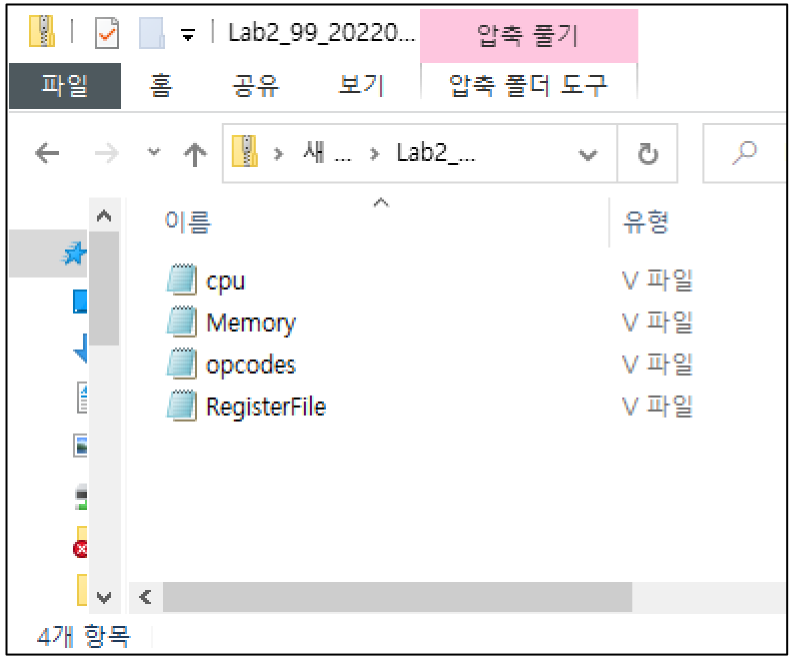
## Evaluation Criteria (cont'd)

- Report
  - The report should include **(1) introduction, (2) design, (3) implementation, (4) discussion, and (5) conclusion sections**
  - Attach **screenshots of your microcode controller, control unit code** in the report
  - **Key points:**
    - Difference between single-cycle CPU and multi-cycle CPU
    - Why multi-cycle CPU is better?
    - Multi-cycle CPU design and implementation
    - Description of whether each module (RF, memory, PC, control unit, ..) is clock synchronous or asynchronous
    - Microcode controller state design
    - Resource reuse design and implementation
    - Number of cycles took it took to run basic_ripes, and loop_ripes examples

## Submission

- Submit your report and source code on PLMS with filename **(ascending order of student ID):**
  - Lab3_{TeamID}_{StudentID1}_{StudentID2}.pdf
    - PDF file of your report
  - Lab3_{TeamID}_{StudentID1}_{StudentID2}.zip
    - Zip file of your source code (without **testbench**)
      - Only *.v files
    - Do not create a folder within the zip file
      - One directory including all codes when unzipped

Zip file contents
(note there is no folder):



| 이름 | 유형 |
|---|---|
| cpu | V 파일 |
| Memory | V 파일 |
| opcodes | V 파일 |
| RegisterFile | V 파일 |

- **There can be penalties for submissions that do not adhere to the guidelines**

## Deadline

- Submission
  - If you wish to present a demo in the Lab-3b session, please submit your code by April 1st 9 a.m.
    - (Optional submission) **Code: 2025. 4. 1 / 09:00 a.m.**
    - Code resubmitted after the demo will not be accepted.

  - **Code: 2025. 4. 15 / 09:00 a.m.**
  - **Report: 2025. 4. 15 / 18:00 p.m.**
  - Evaluation will be done with all instructions (both control-flow and non-control-flow instructions)

## Questions?