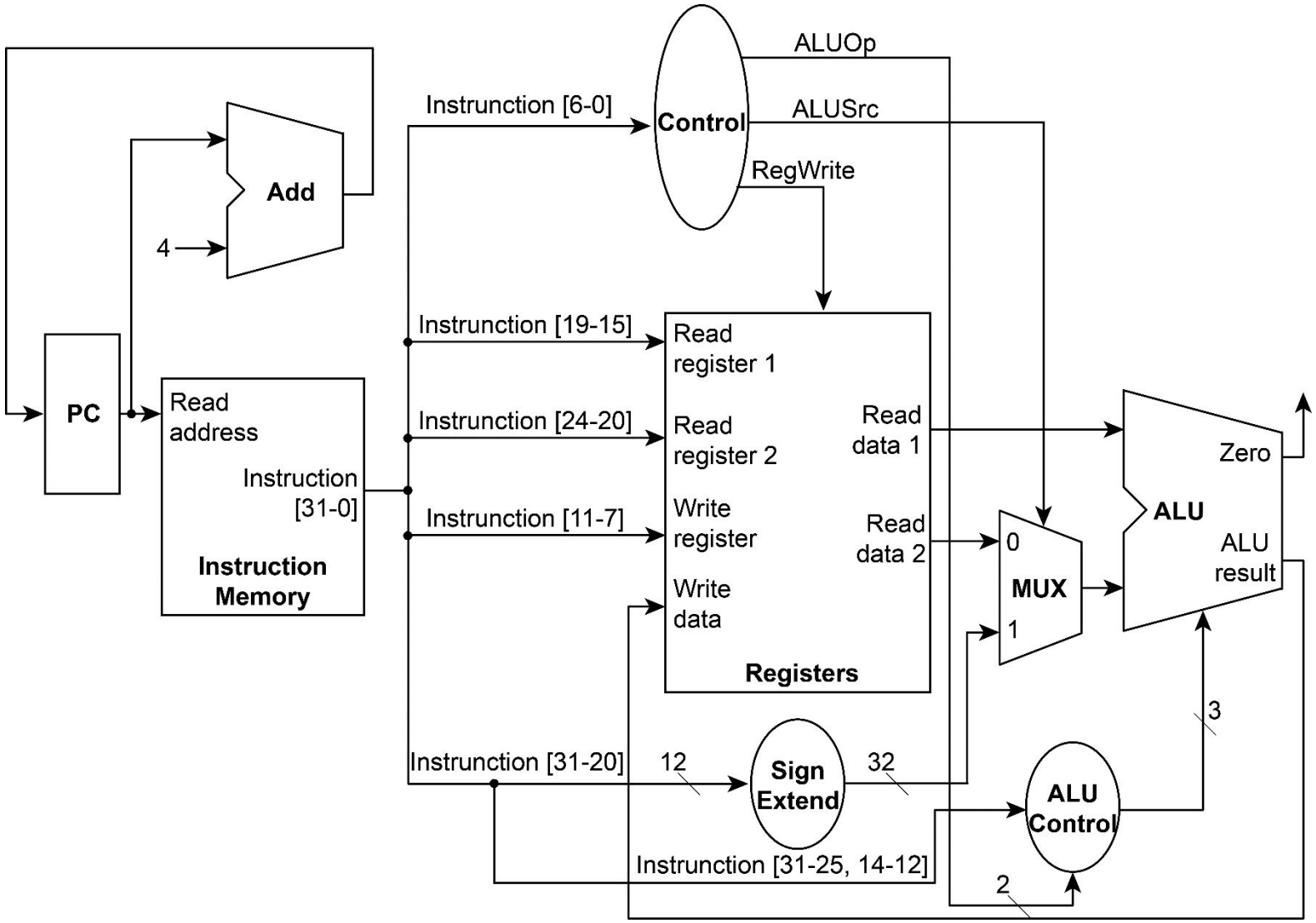
Lab 1

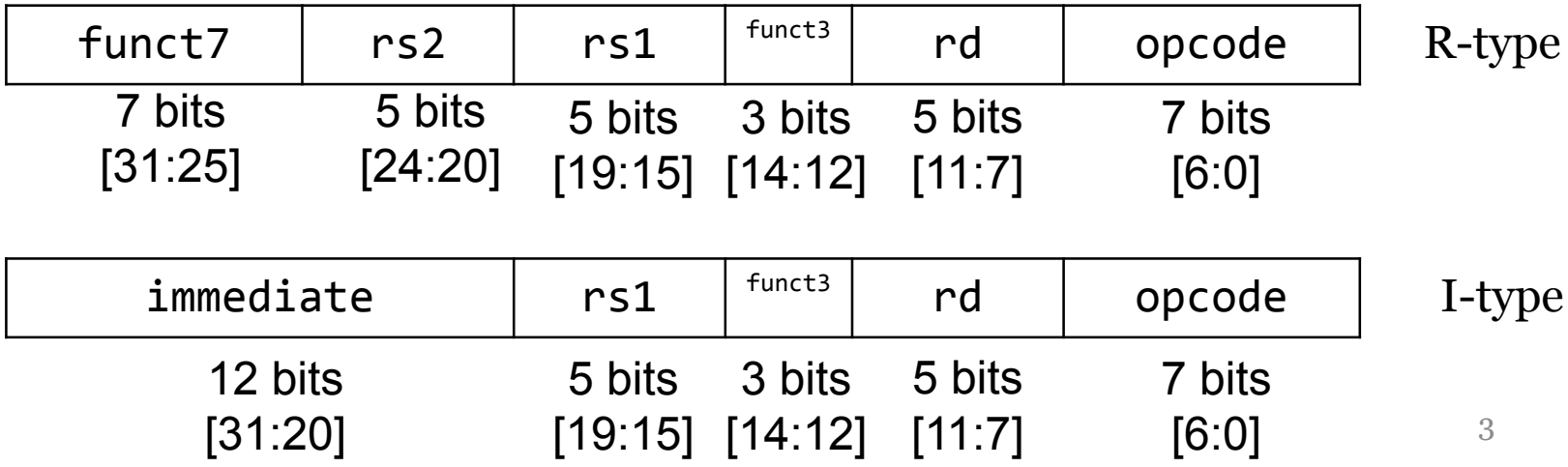
A Single Cycle CPU by Verilog

Data Path


ALUOp
Instrunction [6-0]
Control
ALUSrc
RegWrite
Add
4
Instrunction [19-15]
Read register 1
PC
Read address
Instrunction [24-20]
Read register 2
Read data 1
Zero
ALU
Instruction [31-0]
Instrunction [11-7]
Write register
Read data 2
0
MUX
1
ALU result
Instruction Memory
Write data
Registers
3
Instrunction [31-20]
12
Sign Extend
32
ALU Control
Instrunction [31-25, 14-12]
2

Hardware Specification

- Register file: 32 registers
- Instruction Memory: 1KB
- Your program should read “machine code” rather than “assembly code”
- Machine code:



Instructions

- Required Instruction Set
  - `and rd, rs1, rs2` (bitwise and)
  - `xor rd, rs1, rs2` (bitwise exclusive or)
  - `sll rd, rs1, rs2` (shift left logically)
  - `add rd, rs1, rs2` (addition)
  - `sub rd, rs1, rs2` (subtraction)
  - `mul rd, rs1, rs2` (multiplication)
  - `addi rd, rs1, imm` (addition)
  - `srai rd, rs1, imm` (shift right arithmetically)

Input Format

```
0000000_00000_00000_000_01000_0110011 //add  $t0,$0,$0
000000001010_00000_000_01001_0010011  //addi $t1,$0,10
000000001101_00000_000_01010_0010011  //addi $t2,$0,13
0000001_01001_01001_000_01011_0110011 //mul  $t3,$t1,$t1
000000000001_01001_000_01001_0010011  //addi $t1,$t1,1
0100000_01001_01010_000_01010_0110011 //sub  $t2,$t2,$t1
0000000_01010_01001_111_01011_0110011 //and  $t3,$t1,$t2
```

Input file

```
00000000000000000000010000110011
00000000101000000000010010010011
00000000110100000000010100010011
00000010100101001000010110110011
00000000000010100100001010010011
01000000100101010000010100110011
00000000101001001111010110110011
```

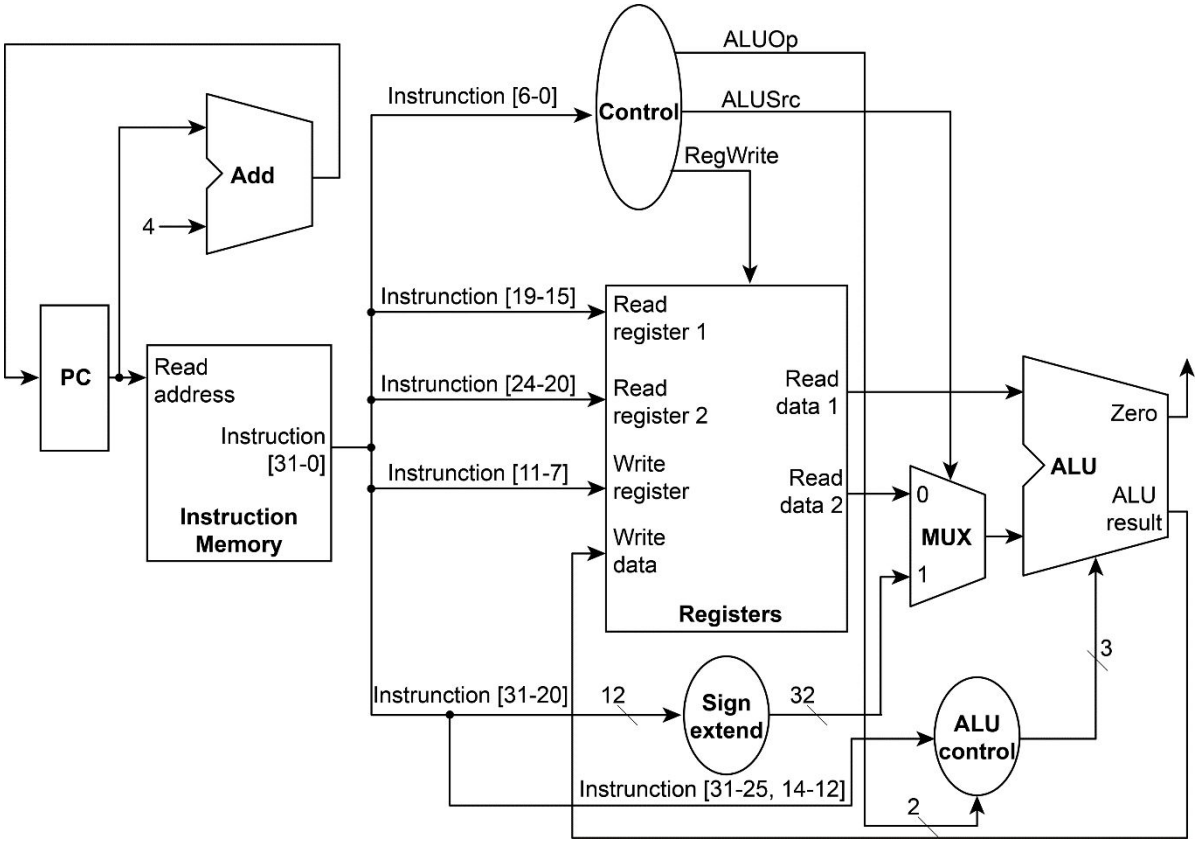
What machine actually reads

Modules

testbench.v

CPU.v

- Control.v
- Adder.v
- PC.v
- Instruction_Memory.v
- Registers.v
- MUX32.v
- Sign_Extend.v
- ALU.v
- ALU_Control.v



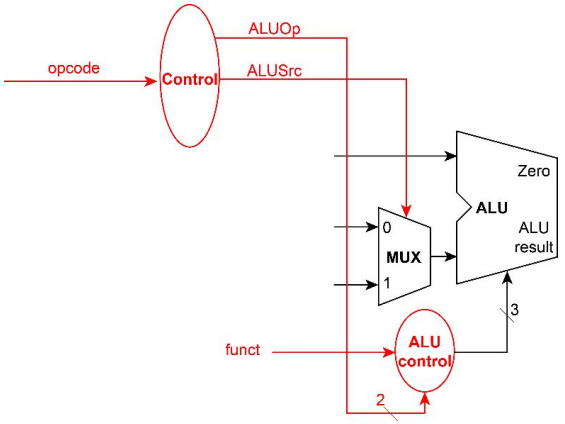
testbench.v

```v
`define CYCLE_TIME 50

module TestBench;

reg                 Clk;
reg                 Reset;
reg                 Start;
integer             i, outfile, counter;

always #(`CYCLE_TIME/2) Clk = ~Clk;

CPU CPU(
    .clk_i  (Clk),
    .rst_i  (Reset),
    .start_i(Start)
);

initial begin
    counter = 0;

    // initialize instruction memory
    for(i=0; i<256; i=i+1) begin
        CPU.Instruction_Memory.memory[i] = 32'b0;
    end


    // initialize Register File
    for(i=0; i<32; i=i+1) begin
        CPU.Registers.register[i] = 32'b0;
    end

    // Load instructions into instruction memory
    $readmemb("instruction.txt", CPU.Instruction_Memory.memory);

    // Open output file
    outfile = $fopen("output.txt") | 1;

    Clk = 0;
    Reset = 0;
    Start = 0;

    #(`CYCLE_TIME/4)
    Reset = 1;
    Start = 1;
end
```

CPU.v

```v
module CPU
(
    clk_i,
    rst_i,
    start_i
);

// Ports
input               clk_i;
input               rst_i;
input               start_i;

/*
Control Control(
    .Op_i       (),
    .ALUOp_o    (),
    .ALUSrc_o   (),
    .RegWrite_o ()
);
*/

/*
Adder Add_PC(
    .data1_in   (),
    .data2_in   (),
    .data_o     ()
);
*/

PC PC(
    .clk_i      (),
    .rst_i      (),
    .start_i    (),
    .pc_i       (),
    .pc_o       ()
);
```

Control.v / ALU_Control.v



funct7	rs2	rs1	funct3	rd	opcode	function
0000000	rs2	rs1	111	rd	0110011	and
0000000	rs2	rs1	100	rd	0110011	xor
0000000	rs2	rs1	001	rd	0110011	sll
0000000	rs2	rs1	000	rd	0110011	add
0100000	rs2	rs1	000	rd	0110011	sub
0000001	rs2	rs1	000	rd	0110011	mul
imm[11:0]		rs1	000	rd	0010011	addi
0100000	imm[4:0]	rs1	101	rd	0010011	srai

Reminder

- Lab 2 and 3 will be strongly related to this homework
- This homework is rather simple, it is recommended that you get familiar with waveform visualization tool (e.g. gtkwave) in this homework

Submission Rule

- Source codes (`*.v` files)
  - `CPU.v`
  - `Control.v`
  - `ALU_Control.v`
  - `Sign_Extend.v`
  - `ALU.v`
  - …
- MUST REMOVE
  - `testbench.v`, `Instruction_Memory.v`, `Registers.v`, `PC.v`
  - `instruction.txt`, `output.txt`
- Report (`<student_ID>_lab1_report.pdf`)
  - Development environment
  - Module implementation explanation
  - Either English or Chinese is fine
  - No more than 2 pages

Module Explanation Example

PC module reads clock signals, reset bit, start bit, and next cycle PC as input, and outputs the PC of current cycle. This module changes its internal register “pc_o” at positive edge of clock signal. When reset signal is set, PC is reset to 0. And PC will only be updated by next PC when start bit is on.

Module Explanation

The inputs of PC are clk_i, rst_i, start_i, pc_i, and ouput pc_o.
It works as follows:

```
always@(posedge clk_i or negedge
rst_i) begin
    if(~rst_i) begin
        pc_o <= 32'b0;
    end
    else begin
        if(start_i)
            pc_o <= pc_i;
        else
            pc_o <= pc_o;
    end
end
```

Submission Rule

- Submission format
  - `<student_ID>_lab1/`
    - `<student_ID>_lab1/<student_ID>_lab1_report.pdf`
    - `<student_ID>_lab1_/codes/*.v`
  - Pack the folder into a .zip file
    - e.g. b09902000_lab1.zip
  - Case sensitive (all alphabets being lower cases)

- Deadline: 04/25/2023 (Tue.) 23:59

- Upload to NTU COOL

Directory Structure

We should see a single directory like following structure after we type

$ unzip b09902000_lab1.zip

in Linux terminal:

b09902000_lab1/

b09902000_lab1/codes

b09902000_lab1/codes/CPU.v

b09902000_lab1/codes/ALU.v

…

b09902000_lab1/b09902000_lab1_report.pdf

Evaluation Criteria

- Report: 20%
- Programming: 80%

- Wrong format: -10 points
- Compilation error: coding 0 points
  - Please make sure your code can be compiled before submitting
- 10 points off per day for late submission
- Plagiarism: 0 points