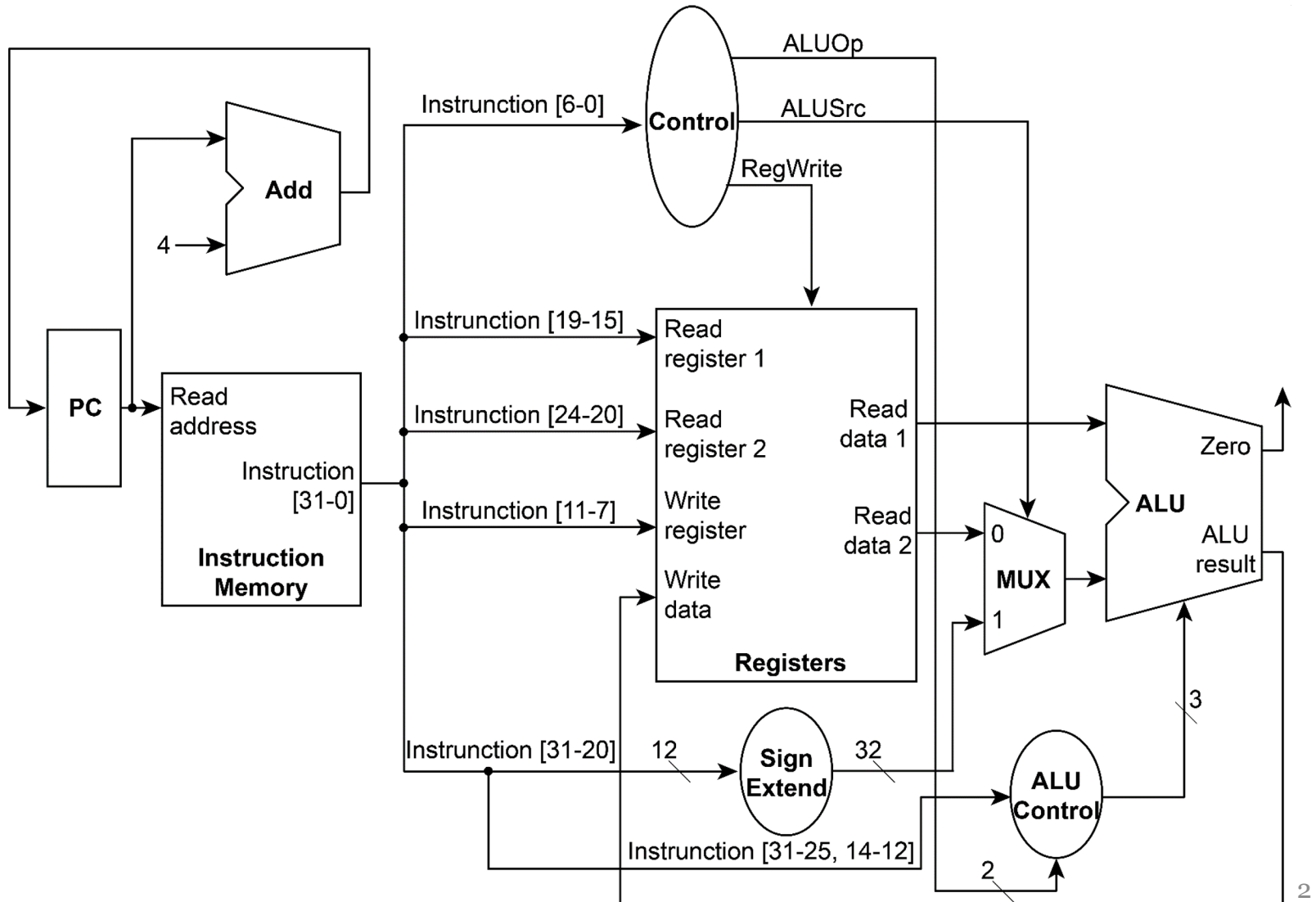


Lab 1

A Single Cycle CPU by Verilog

Data Path



Hardware Specification

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

funct7	rs2	rs1	funct3	rd	opcode	R-type
7 bits [31:25]	5 bits [24:20]	5 bits [19:15]	3 bits [14:12]	5 bits [11:7]	7 bits [6:0]	
immediate		rs1	funct3	rd	opcode	I-type
12 bits [31:20]		5 bits [19:15]	3 bits [14:12]	5 bits [11:7]	7 bits [6:0]	

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

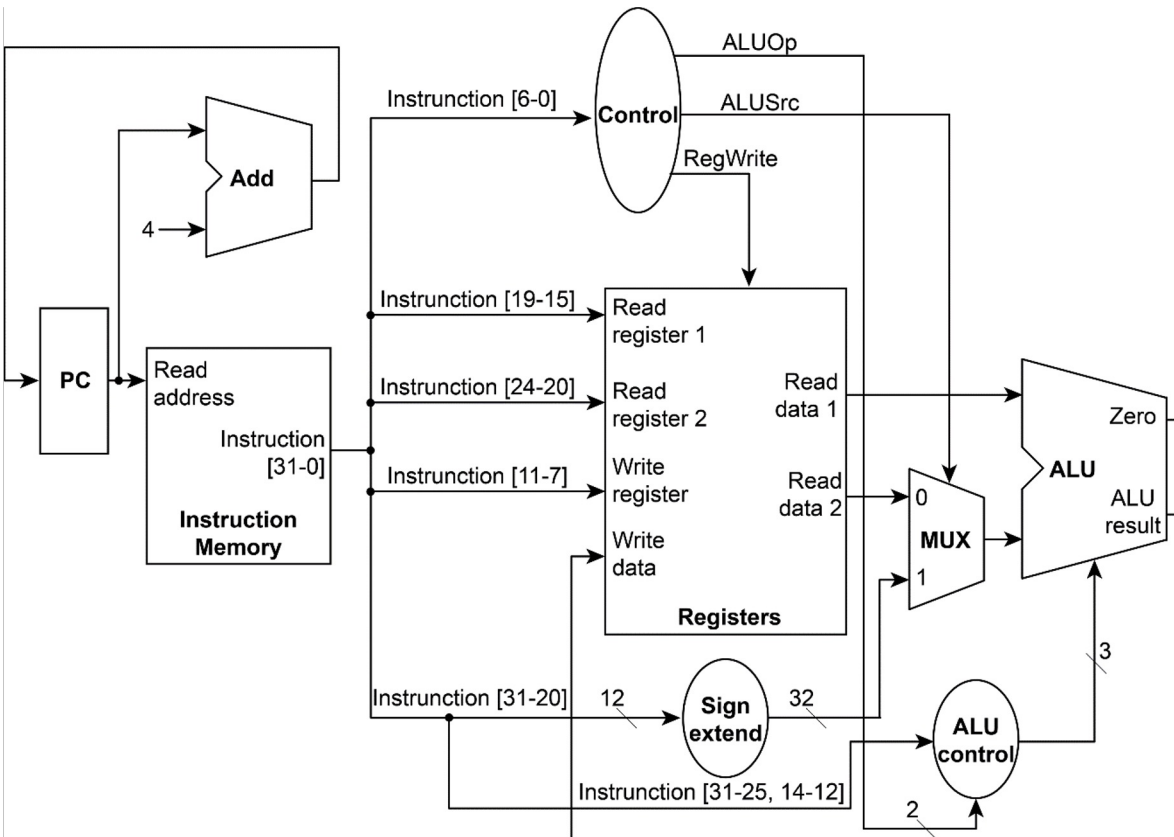
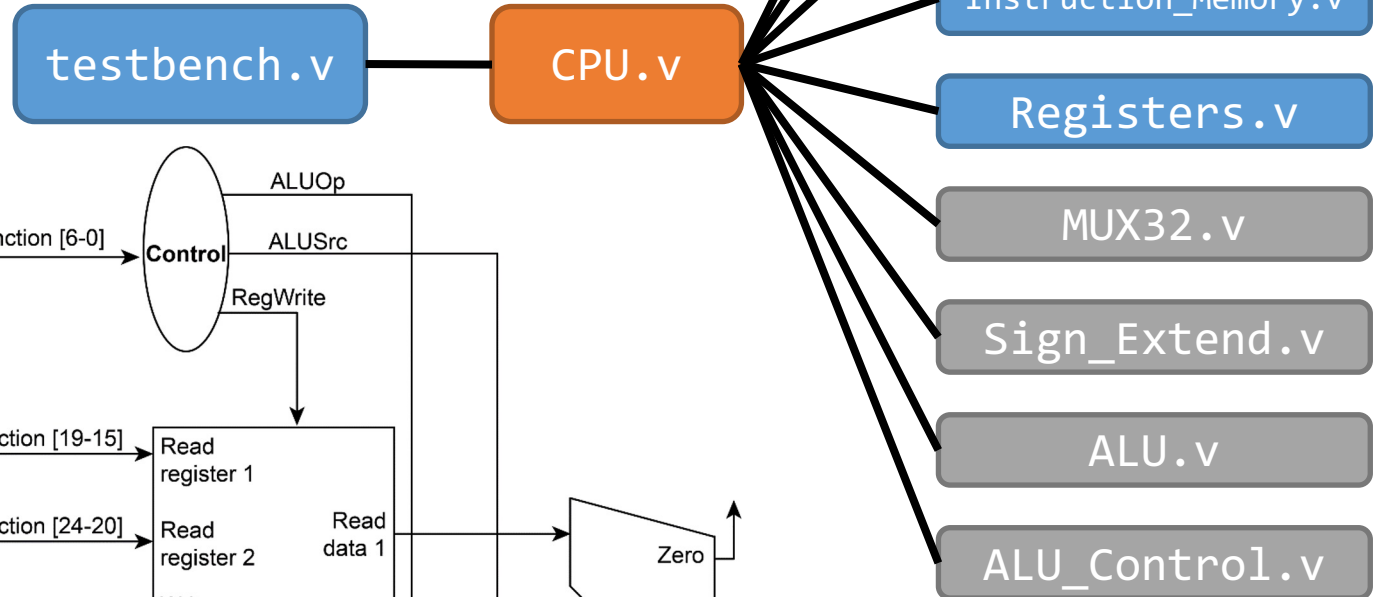
```
00000000_000000_000000_000_01000_0110011 //add $t0,$0,$0
0000000001010_00000_000_01001_0010011 //addi $t1,$0,10
0000000001101_00000_000_01010_0010011 //addi $t2,$0,13
00000001_01001_01001_000_01011_0110011 //mul $t3,$t1,$t1
0000000000001_01001_000_01001_0010011 //addi $t1,$t1,1
01000000_01001_01010_000_01010_0110011 //sub $t2,$t2,$t1
00000000_01010_01001_111_01011_0110011 //and $t3,$t1,$t2
```

Input file

[illegible]

What machine actually reads

Modules



testbench.v

```

1 `define CYCLE_TIME 50
2
3 module TestBench;
4
5 reg          Clk;
6 reg          Reset;
7 reg          Start;
8 integer      i, outfile, counter;
9
10 always #(`CYCLE_TIME/2) Clk = ~Clk;
11
12 CPU CPU(
13     .clk_i  (Clk),
14     .rst_i  (Reset),
15     .start_i(Start)
16 );
17
18 initial begin
19     counter = 0;
20
21     // initialize instruction memory
22     for(i=0; i<256; i=i+1) begin
23         CPU.Instruction_Memory.memory[i] = 32'b0;
24     end
25
26     // initialize Register File
27     for(i=0; i<32; i=i+1) begin
28         CPU.Registers.register[i] = 32'b0;
29     end
30
31     // Load instructions into instruction memory
32     $readmemb("instruction.txt", CPU.Instruction_Memory.memory);
33
34     // Open output file
35     outfile = $fopen("output.txt") | 1;
36
37     Clk = 0;
38     Reset = 0;
39     Start = 0;
40
41     #(`CYCLE_TIME/4)
42     Reset = 1;
43     Start = 1;
44
45 end

```

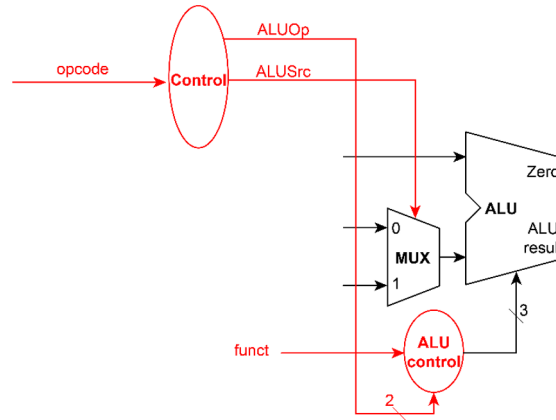
CPU.v

```

1 module CPU
2 (
3     clk_i,
4     rst_i,
5     start_i
6 );
7
8 // Ports
9 input          clk_i;
10 input          rst_i;
11 input          start_i;
12
13 /*
14 Control Control(
15     .Op_i      (),
16     .ALUOp_o   (),
17     .ALUSrc_o  (),
18     .RegWrite_o()
19 );
20 */
21
22 /*
23 Adder Add_PC(
24     .data1_in  (),
25     .data2_in  (),
26     .data_o    ()
27 );
28 */
29
30 PC PC(
31     .clk_i      (),
32     .rst_i      (),
33     .start_i    (),
34     .pc_i       (),
35     .pc_o       ()
36 );

```

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

- | | |
|--|---|
| <ul style="list-style-type: none">• Source codes (*.v files)<ul style="list-style-type: none">• CPU.v• Control.v• ALU_Control.v• Sign_Extend.v• ALU.v• ...• MUST REMOVE<ul style="list-style-type: none">• testbench.v,
Instruction_Memory.v,
Registers.v, PC.v• instruction.txt,
output.txt | <ul style="list-style-type: none">• Report
(<student_ID>_lab1_report.pdf)<ul style="list-style-type: none">• Development environment• Module implementation explanation• Either English or Chinese is fine• No more than 2 pages |
|--|---|

Module Explanation Example

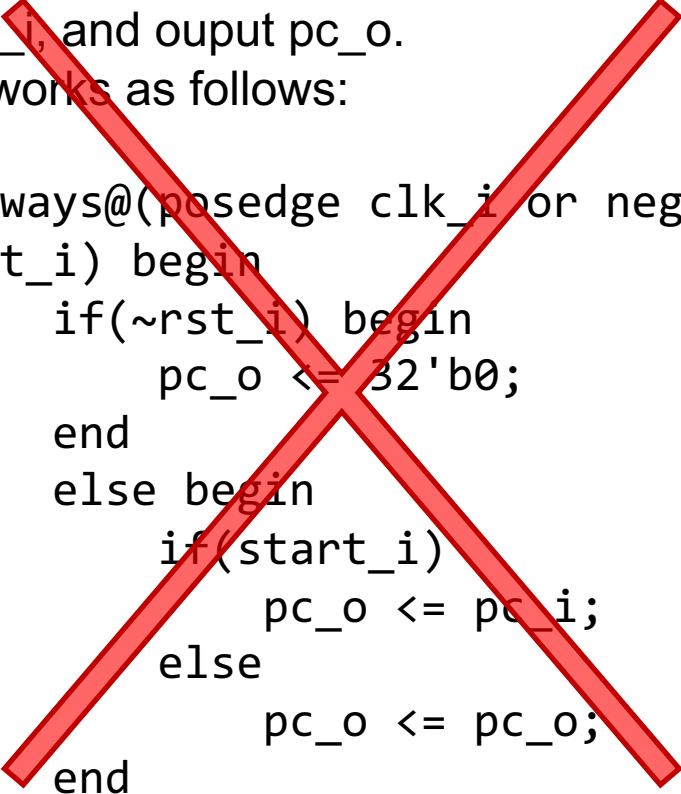
PC module reads clock signals, reset bit, start bit, and next cycle PC as input, and outputs the PC of current cycle. This module changes its internal register “pc_o” at positive edge of clock signal. When reset signal is set, PC is reset to 0. And PC will only be updated by next PC when start bit is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

Module Explanation

The inputs of PC are clk_i, rst_i, start_i, pc_i, and output 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  
end
```



Submission Rule

- Submission format
 - <student_ID>_lab1/
 - <student_ID>_lab1/<student_ID>_lab1_report.pdf
 - <student_ID>_lab1_/src/*.v
 - Pack the folder into a **.zip** file
 - e.g. b09902000_lab1.zip
 - Case sensitive (all alphabets being lower cases)
- Deadline: **11/06/2023 (Mon.) 23:59**
- Upload to **NTU COOL**

Directory Structure

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

```
$ unzip bo9902000_lab1.zip
```

in Linux terminal:

```
bo9902000_lab1/
```

```
bo9902000_lab1/src
```

```
bo9902000_lab1/src/CPU.v
```

```
bo9902000_lab1/src/ALU.v
```

```
...
```

```
bo9902000_lab1/bo9902000_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