Lab3
Single cycle CPU (Simple version)

TA-黃威淳

s094398.cs10@nycu.edu.tw

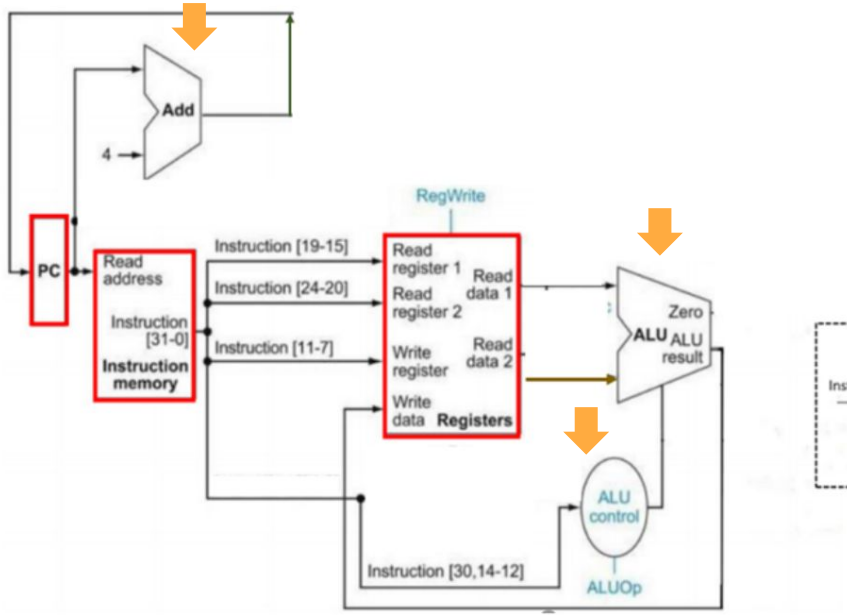
Notice

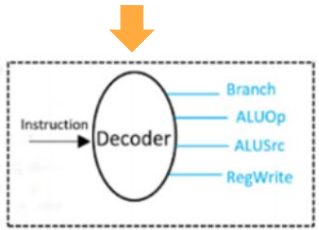
- Please fill out the grouping form (Lab 4 ~6)
  - https://forms.gle/wjPybw6HkHjQTjAd7
- Lab 3 討論區
- Lab 3 is **individual work**

Lab 4~6 grouping form

Overview

- Implement the datapath of Single Cycle CPU as below





Provided

Your work

Attached file

- **TODO**
  - **Lab3Code**
    - alu.v
    - Adder.v
    - ALU_Ctrl.v
    - Decoder.v
    - Simple_Single_CPU.v
- **Validate the correction of your implementation**
  - Please follow the **readme.txt** (Lab3 need to run in **UNIX-like** operating system)
- **Testcase**
  - Lab3Answer/testcase_for_compile.txt

Instruction

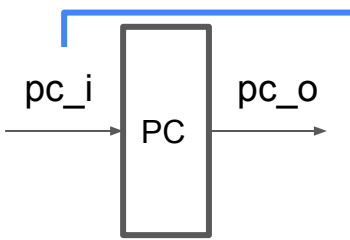
Instruction	Example	Meaning	Opcode	Funct3	Funct7
Addition	add r1, r2, r3	r1 = r2 + r3	0110011	000	0000000
Subtraction	sub r1, r2, r3	r1 = r2 - r3	0110011	000	0100000
Bitwise and	and r1, r2, r3	r1 = r2 & r3	0110011	111	0000000
Bitwise or	or r1, r2, r3	r1 = r2 | r3	0110011	110	0000000
Exclusive OR	xor r1, r2, r3	r1 = r2 ⊕ r3	0110011	100	0000000

Set on less than	slt r1, r2, r3	if(r2 < r3)
		r1 = 1
		else
		r1 = 0
			0110011	010	0000000

Shift left logical	sll r1, r2, r3	r1 = r2 << r3	0110011	001	0000000
Shift right arithmetic	sra r1, r2, r3	r1 = r2 >>> r3	0110011	101	0100000

Program Counter

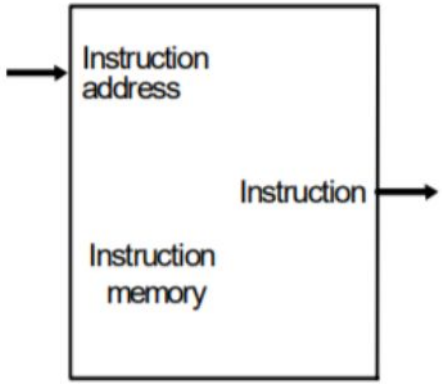
- **You don't need to implement this part**
- Register use to store program counter value




```
module ProgramCounter(
    input                clk_i,
    input                rst_i,
    input      [32-1:0] pc_i,
    output reg [32-1:0] pc_o
    );

always @(posedge clk_i) begin
    if(~rst_i)
        pc_o <= 0;
    else
        pc_o <= pc_i;
end

endmodule
```

Instruction memory

- **You don't need to implement this part**
- Fetch instruction by address
- The script we provide will automatically replace the testcase

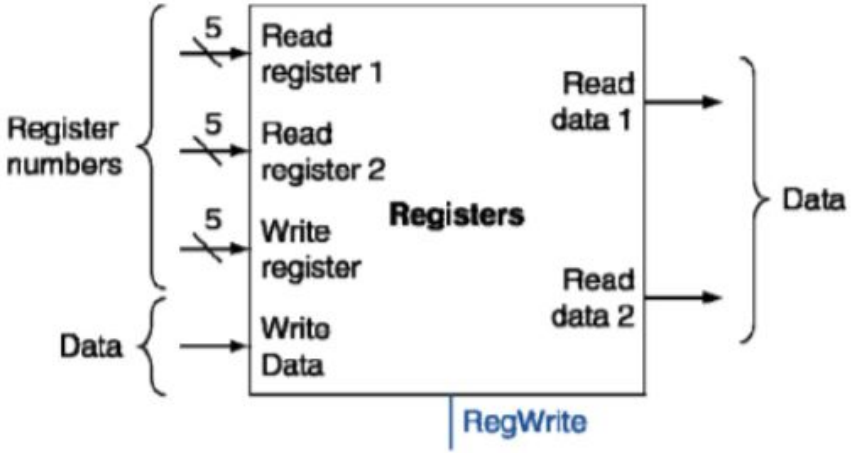



```
`timescale 1ns/1ps

module Instr_Memory(
    input       [32-1:0]     addr_i,
    output wire [32-1:0]     instr_o
);

integer                 i;
reg     [32-1:0] instruction_file   [0:31];

initial begin
    for ( i=0; i<32; i=i+1 )
        instruction_file[i] = 32'b0;
    $readmemb("test_data/c0_test_data0.txt", instruction_file);
end

assign  instr_o = instruction_file[addr_i/4];

endmodule
```

Test case

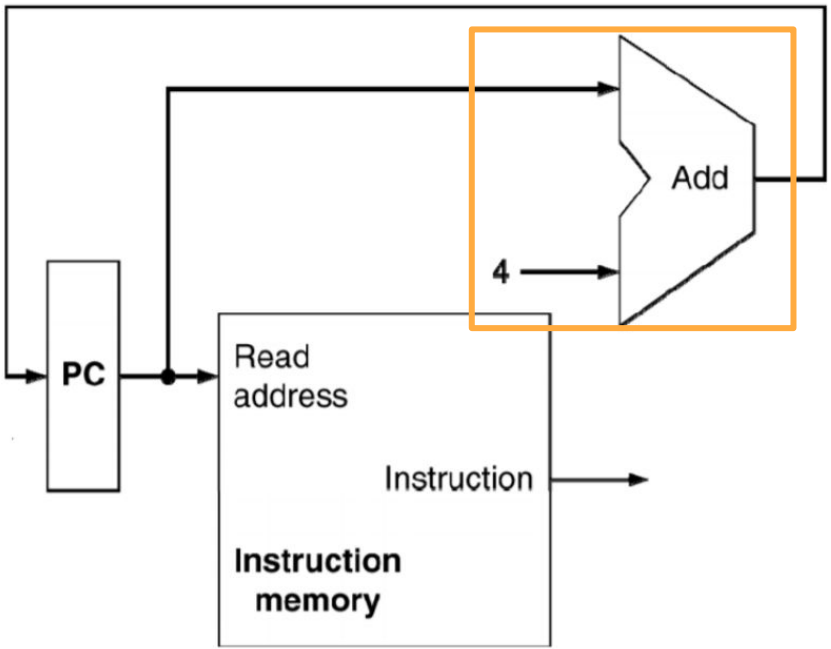
Register file

- **You don't need to implement this part**
- Consists of a set of registers that can be read and written by supplying a register number to be accessed



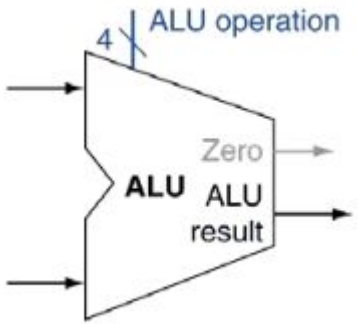
PC + 4 Adder

- Increment by 4 for next instruction



ALU

- Extend ALU operations (LAB 2) to support more instructions
  - You can use any method to extend ALU operations (e.g. sra, sll, xor)
- ALU performs the corresponding operation depend on ALU control



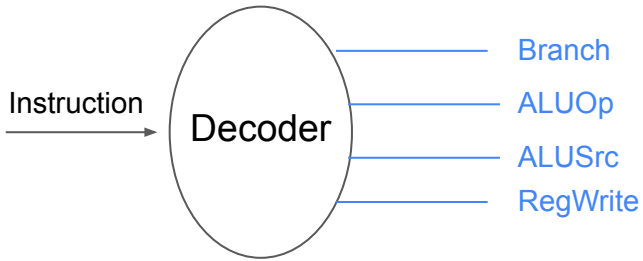
ALU Control

- You can define **sra, sll, xor** ALU Control by yourself
- Other R-type instructions ALU Control please follow the table below

opcode	ALUOp	Operation	I_{30}+fun3	ALU function	ALU control
ld	00	load register	XXXX	add	0010
sd	00	store register	XXXX	add	0010
beq	01	branch on equal	XXXX	subtract	0110
R-type	10	add	0000	add	0010
		subtract	1000	subtract	0110
		AND	0111	AND	0000
		OR	0110	OR	0001

Decoder

- Generate the corresponding control signal according to the instruction
- e.g. R-type instuctions
  - Branch = ?
  - ALUOp = ?
  - ALUSrc = ?
  - RegWrite = ?



Simple Single CPU

- Combine all the above parts to build a CPU
- You need to complete Simple_Single_CPU.v all the parts input and output

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

Instr_Memory IM(
        .addr_i(),
        .instr_o()
        );

Reg_File RF(
        .clk_i(),
        .rst_i(),
        .RSaddr_i(),
        .RTaddr_i(),
        .RDaddr_i(),
        .RDdata_i(),
        .RegWrite_i(),
        .RSdata_o(),
        .RTdata_o()
        );
```

Testcase

- Lab3Answer/testcase_for_compile.txt
- In all case, initially
  - **x1 = 1**
  - **x2 = 2**
  - **other = 0**

Case 1 :
xor x1,x1,x1
add x2,x2,x1
or x3,x2,x1
sll x4,x3,x2
sub x5,x3,x4

result:
x1 = 0 , x2 = 2,x3 = 2 ,x4 = 8 ,x5 = -6

Case 2 :

sra x6,x2,x1
sub x4,x4,x2
sub x4,x4,x2
sra x5,x4,x2

result:
x1 = 1, x2 = 2,x4 = -4 ,x5 = -1 ,x6 = 1

Testbench

- **This script cannot run in Windows**
- Put your .v file in Lab3Code
- $ chmod +x ./lab3TestScript.sh && ./lab3TestScript.sh