

---

---

# Computer Organization Lab 3

教授:蔡文錦

TAs:林浩君、薛乃仁、吳年茵

---

---

# Objectives

In this lab, we are going to implement a simple single cycle CPU with memory unit, which can run R-type, I-type and jump instructions.

# Demands

There are plenty of modules that you need to write in this lab:

"Simple\_Single\_CPU.v", "Adder.v", "ALU.v", "ALU\_Ctrl.v", "Decoder.v",  
"Instr\_Memory.v", "Mux2to1.v", "Mux3to1.v", "Program\_Counter.v", "Reg\_File.v",  
"Shifter.v", "Sign\_Extend.v", "Zero\_Filled.v", "Data\_Memory.v", "Testbench.v"

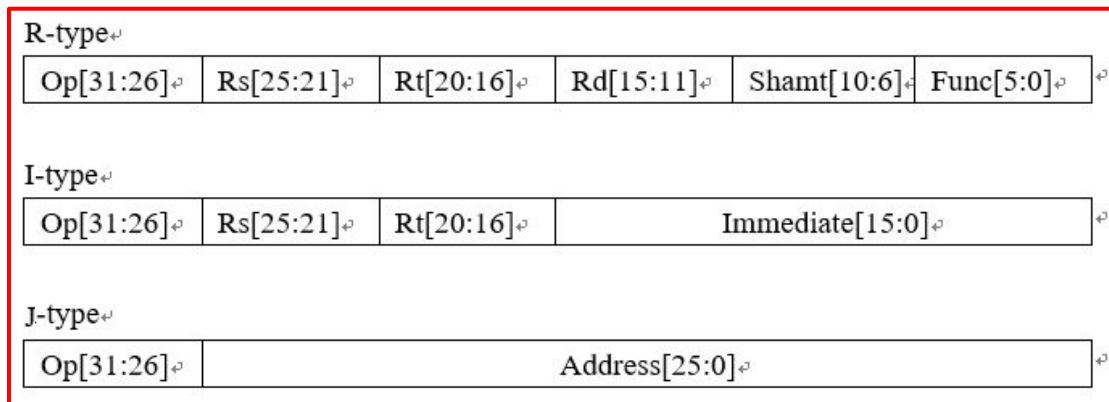
All these files are provided, please make use of them and **don't create additional .v files.**

# Task Description

We are going to implement these instructions, and we recommend you to write them in such sequence:

- R-type: add, sub, and, or, nor, slt, sll, srl, **sllv**, **srlv**, **jr**
- I-type: addi, lw, sw, beq, bne, **blt**, **bnez**, **bgez**
- J-type: jump, **jal**

Instruction Format:



# R-type Instruction set and ADDI (9% \* 5)

Instruction	Example	Meaning	Op Field[31:26]	Shamt	Function Field
add	add rd, rs, rt	$\text{Reg}[\text{rd}] = \text{Reg}[\text{rs}] + \text{Reg}[\text{rt}]$	6'b000000	x	6'b100011
sub	sub rd, rs, rt	$\text{Reg}[\text{rd}] = \text{Reg}[\text{rs}] - \text{Reg}[\text{rt}]$	6'b000000	x	6'b010011
and	and rd, rs, rt	$\text{Reg}[\text{rd}] = \text{Reg}[\text{rs}] \& \text{Reg}[\text{rt}]$	6'b000000	x	6'b011111
or	or rd, rs, rt	$\text{Reg}[\text{rd}] = \text{Reg}[\text{rs}] \mid \text{Reg}[\text{rt}]$	6'b000000	x	6'b101111
nor	nor rd, rs, rt	$\text{Reg}[\text{rd}] = \sim(\text{Reg}[\text{rs}] \mid \text{Reg}[\text{rt}])$	6'b000000	x	6'b010000
slt	slt rd, rs, rt	If $(\text{Reg}[\text{rs}] < \text{Reg}[\text{rt}])$ $\text{Reg}[\text{rd}] = 1$ else $\text{Reg}[\text{rd}] = 0$	6'b000000	x	6'b010100
sll	sll rd, rt	$\text{Reg}[\text{rd}] =$ $(\text{unsigned}) \text{Reg}[\text{rt}] \ll (\text{unsigned}) \text{amt}$	6'b000000	amt	6'b010010
srl	srl rd, rt	$\text{Reg}[\text{rd}] =$ $(\text{unsigned}) \text{Reg}[\text{rt}] \gg (\text{unsigned}) \text{amt}$	6'b000000	amt	6'b100010
addi	addi rt,rs,imm	$\text{Reg}[\text{rt}] = \text{Reg}[\text{rs}] + \text{imm}$	6'b010011	x	x

# Advanced Set: Part 1 (15%)

- slv (Shift left logical variable) and srlv (Shift right logical variable)

Instruction	Example	Meaning	Op Field[31:26]	Shamt	Function Field
slv	slv rd, rt, rs	Reg[rd]= (unsigned) Reg[rt]<<(unsigned) Reg[rs]	6'b000000	x	6'b011000
srlv	srlv rd, rt, rs	Reg[rd]= (unsigned) Reg[rt]>>(unsigned) Reg[rs]	6'b000000	x	6'b101000

# I-type Instruction and Jump (25%)

Instruction	Example	Meaning	Op Field[31:26]
lw	lw rt, imm(rs)	$\text{Reg}[\text{rt}] \leftarrow \text{Mem}[\text{Reg}[\text{rs}] + \text{imm}]$	6'b011000
sw	sw rt, imm(rs)	$\text{Mem}[\text{Reg}[\text{rs}] + \text{imm}] \leftarrow \text{Reg}[\text{rt}]$	6'b101000
beq	beq rs, rt, imm	if( $\text{Reg}[\text{rs}] == \text{Reg}[\text{rt}]$ ) then $\text{PC} = \text{PC} + 4 + (\text{imm} \ll 2)$	6'b011001
bne	bne rs, rt, imm	if( $\text{Reg}[\text{rs}] \neq \text{Reg}[\text{rt}]$ ) then $\text{PC} = \text{PC} + 4 + (\text{imm} \ll 2)$	6'b011010
jump	jump address	$\text{PC} = \{\text{PC}[31:28], \text{address}[25:0] \ll 2\}$	6'b001100

## Advanced Set: Part 2 (15%)

- jal (Jump and link)
  - In MIPS, 31th register is used to save return address for function call Reg[31] save PC+4 and perform jump
- jr (Jump to the address in the register rs)

Instruction	Example	Meaning	Op Field[31:26]	Function Field[5:0]
jal	jal address	Reg[31]=PC+4 PC={PC[31:28],address[25:0]<<2}	6'b001111	x
jr	jr rs	PC=Reg[rs]	6'b000000	6'b000001

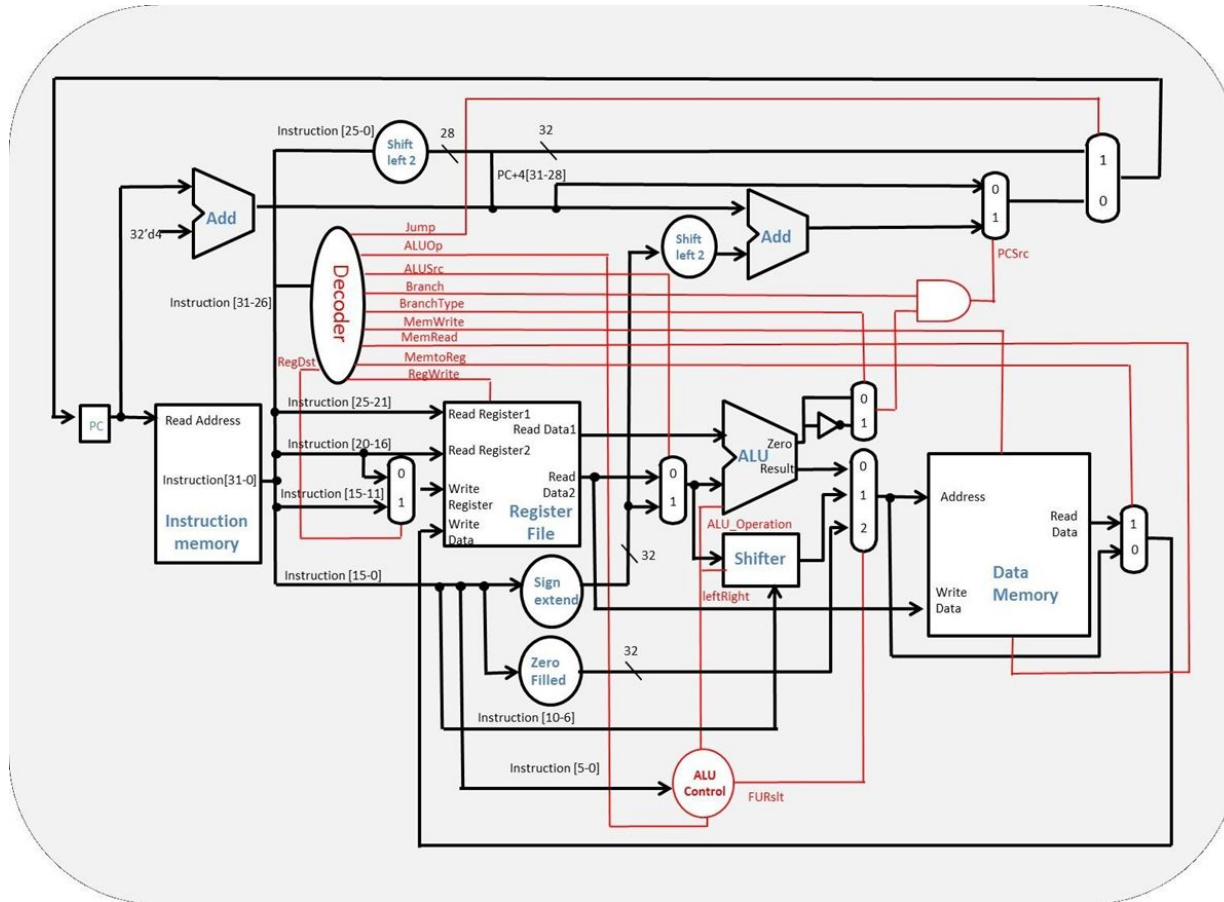


## Advanced Set: Part 3 (15%)

- blt(branch on less than): if( $\text{Reg}[\text{rs}] < \text{Reg}[\text{rt}]$ ) then branch
- bnez(branch non equal zero): if( $\text{Reg}[\text{rs}] \neq 0$ ) then branch
- bgez(branch greater equal zero): if( $\text{Reg}[\text{rs}] \geq 0$ ) then branch

Instruction	Example	Meaning	Op Field[31:26]
blt	blt rs, rt, imm	if( $\text{Reg}[\text{rs}] < \text{Reg}[\text{rt}]$ ) then $\text{PC} = \text{PC} + 4 + (\text{imm} \ll 2)$	6'b011100
bnez	bnez rs, 0, imm (set rt = 0, note: Reg[0] always equal zero)	if( $\text{Reg}[\text{rs}] \neq 0$ ) then $\text{PC} = \text{PC} + 4 + (\text{imm} \ll 2)$	6'b011101
bgez	bgez rs, 0, imm (set rt = 0, note: Reg[0] always equal zero)	if( $\text{Reg}[\text{rs}] \geq 0$ ) then $\text{PC} = \text{PC} + 4 + (\text{imm} \ll 2)$	6'b011110

# Architecture



- You should follow the architecture to implement your cpu (but with some tweak).

# Testing Data (for student)

- testcases/test\_[1-3].txt: basic instructions using R-type and addi.
- testcases/test\_4.txt: basic instructions using I-type and jump.
- **No testing for instruction in advance sets! Try to write your own testing data.**

# Testing Method

Command:

```
> iverilog Testbench.v  
> ./a.out  
+++++ R-type & addi +++++  
test 1  
test 2  
test 3  
+++++ I-type & jump +++++  
test 4  
Score: 52/52
```

- Ensure the program compiles successfully by executing the command `iverilog Testbench.v`

# Submission

- Compress all the **\*.v files** into one zip file, and name your zip file as **HW3\_{studentID}.zip** (e.g. **HW3\_0811510.zip**)
- **No additional directory!**
- **Wrong format will have 10% penalty.**
- **Any assignment work by fraud will get a zero point.**
- **No late submission (deadline: 8/9 23:55)!**

```
> zipinfo -1 HW3_0987654321.zip
Adder.v
ALU_Ctrl.v
ALU.v
Data_Memory.v
Decoder.v
Instr_Memory.v
Mux2to1.v
Mux3to1.v
Program_Counter.v
Reg_File.v
Shifter.v
Sign_Extend.v
Simple_Single_CPU.v
Testbench.v
Zero_Filled.v
```