

Computer Organization, Spring 2018

Lab 2: Single-Cycle CPU

Due:2018/8/2

1. Goal

Based on Lab 1 (simple single-cycle CPU), add a memory unit to implement a complete single-cycle CPU which can run R-type, I-type and jump instructions.

2. Requirement

- (1) Please use **Icarus Verilog** as your HDL simulator.
- (2) Please attach **your names** and **student IDs** as comment at the top of each file.
- (3) Refer to Lab 1 for the top module's name and I/O ports.

Reg_File[29] represents stack point. Initialize Reg_file[29] to 128 while others to 0.

You may add control signals to Decoder, e.g.

- Branch_o
- Jump_o
- MemRead_o
- MemWrite_o
- MemtoReg_o

3. Requirement description

Lw instruction

memwrite is 0, memread is 1, regwrite is 1

$\text{Reg}[\text{rt}] \leftarrow \text{Mem}[\text{rs}+\text{imm}]$

Sw instruction

memwrite is 1, memread is 0

$\text{Mem}[\text{rs}+\text{imm}] \leftarrow \text{Reg}[\text{rt}]$

Branch instruction

branch is 1, ALU's ZERO signal is 1

$\text{PC} = \text{PC} + 4 + (\text{sign_Imm} \ll 2)$

Jump instruction

jump is 1

PC = {PC[31:28], address<<2}

4. Code (80 pts.)

(1) Basic instructions: (60 pts.)

Instructions in Lab 2 + mul, lw, sw, j

R-type

Op[31:26]	Rs[25:21]	Rt[20:16]	Rd[15:11]	Shamt[10:6]	Func[5:0]
-----------	-----------	-----------	-----------	-------------	-----------

I-type

Op[31:26]	Rs[25:21]	Rt[20:16]	Immediate[15:0]
-----------	-----------	-----------	-----------------

Jump

Op[31:26]	Address[25:0]
-----------	---------------

instruction	op[31:26]			
lw	6'b100011	Rs[25:21]	Rt[20:16]	Immediate[15:0]
sw	6'b101011	Rs[25:21]	Rt[20:16]	Immediate[15:0]
j	6'b000010	Address[25:0]		

Mul is R-type instruction

0	Rs[25:21]-	Rt[20:16]	Rd[15:11]	0	6'b011000
---	------------	-----------	-----------	---	-----------

(2) Advanced set 1: (10 pts.)

instruction	op	rs	rt	rd	shamt	func
jal	6'b000011	Address[25:0]				
jr	6'b000000	rs	0	0	0	6'b001000

Jal: jump and link

In MIPS, the 31st register is used to save return address for function call.

Reg[31] saves PC+4 and address for jump

Reg[31] = PC + 4

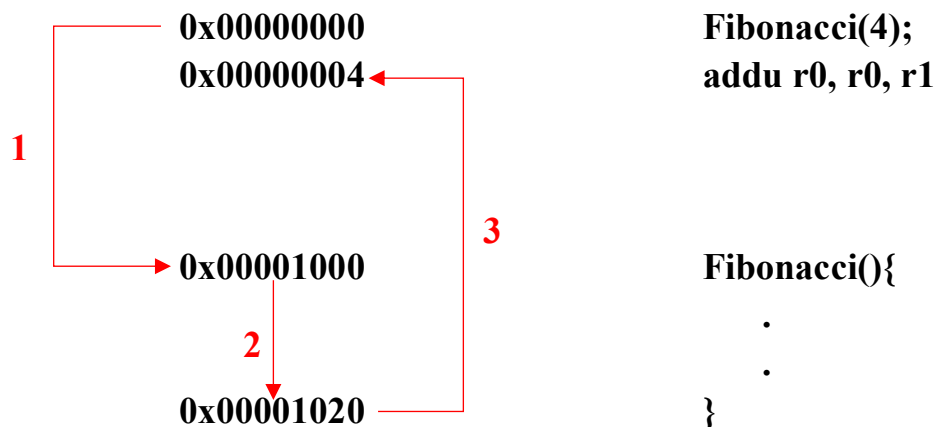
PC = {PC[31:28], address[25:0]<<2}

Jr: jump to the address in the register rs

PC = Reg[rs];

e.g. : In MIPS, return could be used by **jr r31** to jump to return address from JAL

Example: when CPU executes function call,



if you want to execute recursive function, you must use the stack point (**Reg_File[29]**).

First, store the register to memory and load back after function call has been finished. The second testbench CO_P2_test_data2.txt is the Fibonacci function. After it is done, r2 stores the final answer. Please refer to test2.txt.

(3) Advanced set 2: (20 pts.)

ble (branch less equal than): if(rs <= rt) then branch

6'b000110	rs	rt	offset
-----------	----	----	--------

bnez (branch non equal zero): if(rs != 0) then branch (same as bne)

6'b000101	rs	0	offset
-----------	----	---	--------

bltz (branch less than zero): if(rs < 0) then branch

6'b000001	rs	0	offset
-----------	----	---	--------

li (load immediate)

the instructions implement is easy, because it is similar to (and thus can be replaced by) addi.

6'b001111	0	rt	immediate
-----------	---	----	-----------

5. Testbench

CO_P2_test_data1.txt tests the **basic instructions** (50 pts.)

CO_P2_test_data2.txt tests the **advanced set 1** (10 pts.).

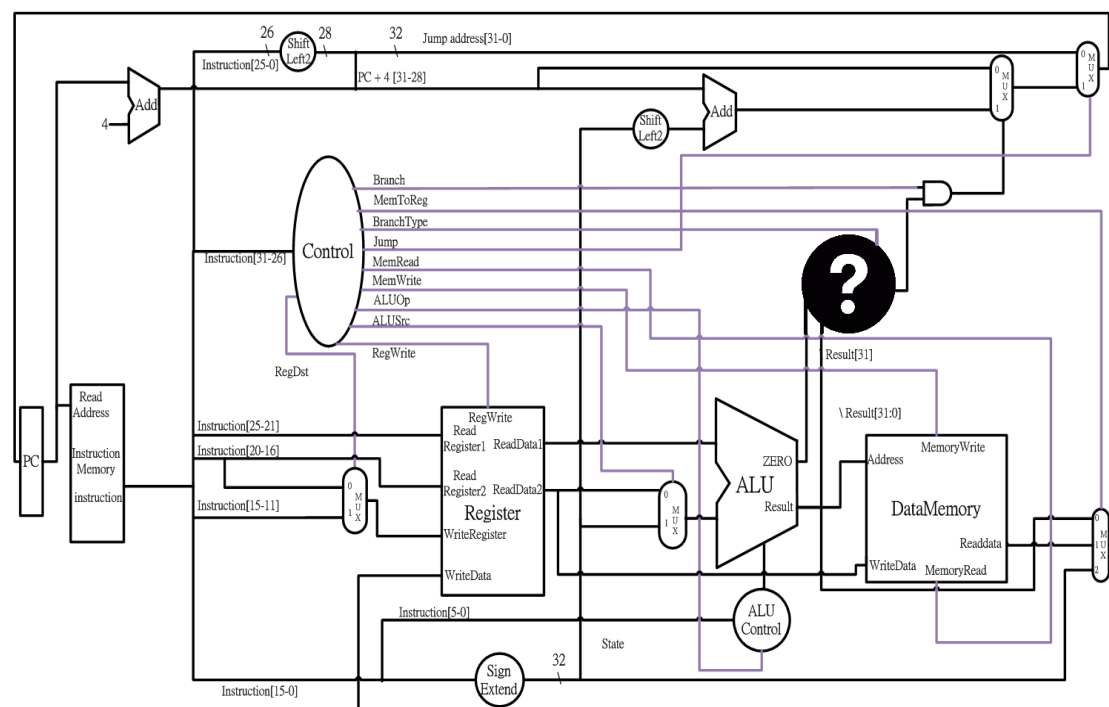
CO_P2_test_data3.txt tests the **advanced set 2** (20 pts.).

Please refer to test1.txt and test2.txt for details. The following MIPS code is bubble sort.

addu	\$t0, \$0, \$0	sw	\$t2, 0(\$t0)
addi	\$t1, \$0, 10	sw	\$t3, 4(\$t0)
addi	\$t2, \$0, 13	li	\$t1, 1
mul	\$t3, \$t1, \$t1	no_swap:	
j	Jump	addi	\$t5, \$0, 4
bubble:		subu	\$t0, \$t0, \$t5
li	\$t0, 10	bltz	\$t0, next_turn
li	\$t1, 4	j	inner
mul	\$t4, \$t0, \$t1	next_turn:	
outer:		bnez	\$t1, outer
addi	\$t6, \$0, 8	j	End
subu	\$t0, \$t4, \$t6	Jump:	
li	\$t1, 0	subu	\$t2, \$t2, \$t1
inner:		Loop:	
lw	\$t2, 4(\$t0)	addu	\$t4, \$t3, \$t2
lw	\$t3, 0(\$t0)	beq	\$t1, \$t2, Loop
ble	\$t2, \$t3, no_swap	j	bubble
		End:	

6. Reference architecture

This lab may extra signal(s) to control. Please draw the architecture you designed in your report.



7. Grade

- (1) Total score: 110 pts. (plagiarism will get 0 point)
- (2) Basic score: 60 pts. Advanced set 1: 10 pts. Advanced set 2: 20 pts.
- (3) Report: 20 pts – format is in CO_Report.docx.
- (4) Delay: 10 pts off per day

8. Hand in your assignment

- (1) Zip your folder and name it as "ID1_ID2.zip" (e.g., 0516001_0516002.zip) before uploading to e3. Other filenames and formats such as *.rar and *.7z are NOT accepted! Multiple submissions are accepted, and the version with the latest time stamp will be graded.
- (2) Please include ONLY Verilog source codes (*.v), Makefile and your report (*.pdf) in the zipped folder. There will be many files generated by the simulation tool (Xilinx) - do not include them; WE NEED ONLY VERILOG SOURCE CODES AND YOUR REPORT!
- (3) Command:


```
unzip ID1.zip or unzip ID1_ID2.zip
cd ID1 or ID1_ID2
make
//generate Single_Cycle_CPU and Single_Cycle_CPU.vcd
```

Debug (Watch Wave):
gtkwave Single_Cycle_CPU.vcd

9. Q&A

For any questions regarding Lab 2, please contact 曾威凱 (k50402k@gmail.com)
and 黃礪鞍 (vdc159753@gmail.com).