Computer Organization, Fall 2013

Lab 3: Single Cycle CPU Ⅱ

Due: 2013/11/14 23:59:59

1. Purpose

Based on lab2，adding a memory unit(named the module DM in Simple_Single_CPU). Implement a complete single cycle CPU that can run R-type, I-type, branch, and jump instructions.

2. Requirement (90%)

A. Please use Modelsim as simulation platform.

B. Two persons form a group. Just hand in one assignment for one group.

Please attach your names and student IDs as comments in the top of each file. The assignment you upload on e3 must have the form of "student ID1_student ID2.rar ".

C. Top module and IO ports naming rules are as following:

Top module: Simple_Single_CPU.v

```
Module Simple_Single_CPU(
        clk_i,    //system clock (input)
        rst_i     //reset signal (input)
);
```

To keep good coding style, using the following rules:

I. Only one module in one .v file.

II. module name should be the same as the file name.

D. Reg_File[29] represents stack point. Please give an initial value to Reg_File[29] as 128, others 0.

Decoder may add the following control signal:

- BranchType_o
- Jump_o
- MemRead_o
- MemWrite_o
- MemtoReg_o

E. Please name the Data_Memory module DM in Simple_Single_CPU.

F. **Basic instruction (60%)**

Lab2 instruction + mul、lw、sw、jump

Instruction	Op[31:26]	rs	rt	rd	shamt	func
mul	6’b000000	rs[25:21]	rt[20:16]	rd[15:11]	5’b00000	6’b011000
lw	6’b100011	rs[25:21]	rt[20:16]	imm[15:0]		
sw	6’b101011	rs[25:21]	rt[20:16]	imm[15:0]		
jump	6’b000010	Address[25:0]				

I. Mul

Reg[rd] ← Reg[rs]*Reg[rt]

II. lw

MemWrite to be 0, MemRead to be 1，RegWrite to be 1

Reg[rt] ← Mem[rs+imm]

III. sw

MemWrite to be 1, MemRead to be 0

Mem[rs+imm] ← Reg[rt]

IV. jump

Jump to be 1

PC={PC[31:28]，address<<2}

G. **Advanced instruction (30%)**

Instruction	Op[31:26]	rs	rt	imm
bgt	6’b000111	rs[25:21]	rt[20:16]	imm[15:0]
bnez	6’b000101	rs[25:21]	5’b00000	imm[15:0]
bgez	6’b000001	rs[25:21]	5’b00001	imm[15:0]
lui	6’b001111	5’b00000	rt[20:16]	imm[15:0]

I. bgt (branch greater than)

Branch to be 1

if (rs > rt) then PC = PC + 4 + (sign_imm<<2)

II. bnez (branch non equal zero)

Branch to be 1

if (rs != 0) then PC = PC + 4 + (sign_imm<<2)

III. bgez (branch greater equal zero)

Branch to be 1

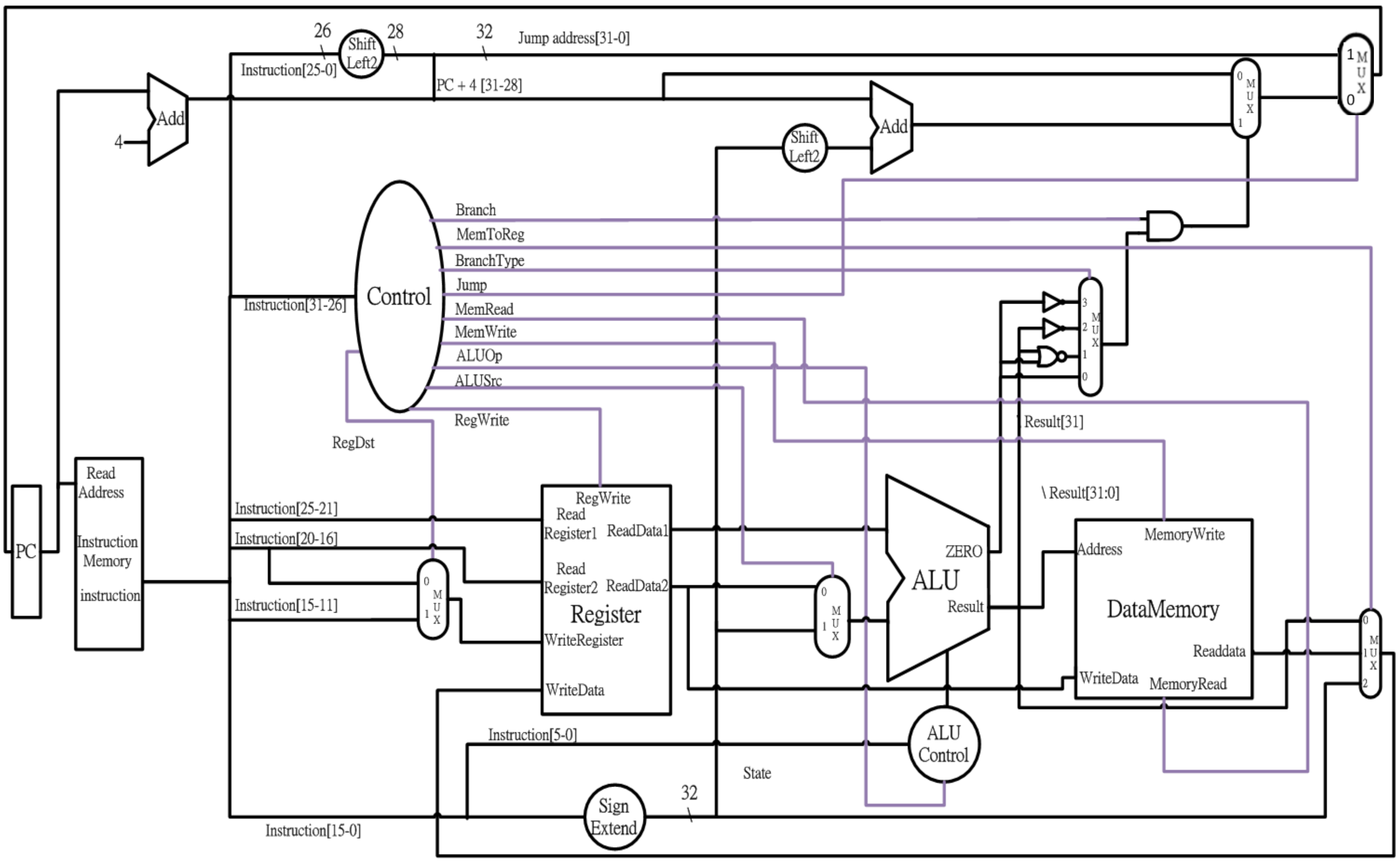
if (rs >= 0) then PC = PC + 4 + (sign_imm<<2)

IV. li (load unsigned immediate)

Reg[rt] ← unsigned_imm

3. Basic Architecture

If you need to use extra control signal, please draw your design to the architecture and descript the implement flow on the report.



4. Bonus (20%)

A. Bonus1 (10%)

Instruction	op[31:26]	rs	rt	rd	shamt	func
jal	6’b000011	Address[25:0]				

jr	6’b000000	rs	0	0	0	6’b001000

I. jal (jump and link)

reg[31]=PC+4

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

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

Reg[31] update to (PC+4) and do jump.

II. jr (jump to the address in the register rs)

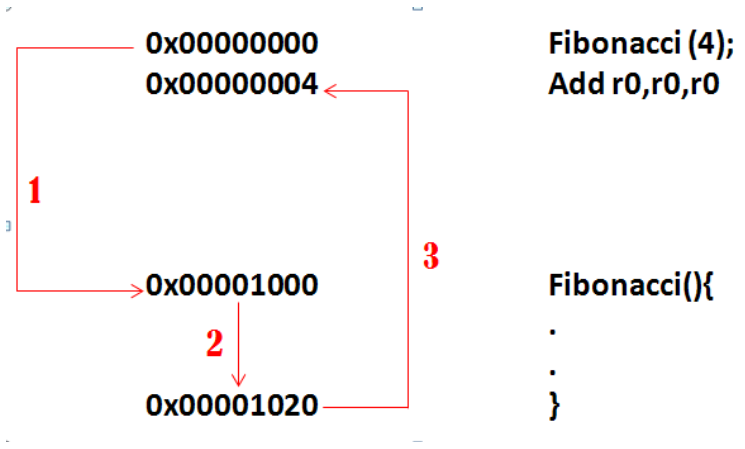
PC=reg[rs];

In MIPS, return could be used by

jr r31

to jump to return address from jal.

III. Example : function call in the program



If it execute recursive function, must collocate with stack point (Reg_File[29]). First, store the register to memory and load back after function call has been finished.

B. Bonus2 (10%)

Please use the MIPS instruction set in your single cycle CPU up to now to design a simple sorting program (no restriction to sorting algorithm). At the beginning of the program you should use lui, sw instructions to initial memory[0] ~ memory[4] to 4, 5, 3, 2, 1 respectively. And we expect the value of memory[0] ~ memory[4] should sort in ascendant order at the end of the program.

You can write the code by yourself or use some C code to MIPS translation tool to help you. You can translate your code to Binary or Hexadecimal. If you translate to Hexadecimal, you needs modify instruction memory from $readmemb to $readmemh. If your code exceeds 50 instructions, you must remember to modify

END_COUNT in TestBench.v(default:50).

5. Test

A. Basic instruction (60%)

TA offers CO_LAB3_test_data1.txt to let you test your design. Refer to test1.txt to check your result.

B. Advanced instruction (30%)

CO_LAB3_test_data2.txt extend the previous part and add more instruction to let your test your design. Refer to test2.txt to check your result.

C. Bonus 1(10%)

CO_LAB3_test_data3.txt,it is a Fibonacci function. When it done r2 is the answer we want. Refer to test3.txt to check your result.

D. Bonus 2(10%)

Refer to the test data offered in LAB3, hand in two files including CO_LAB3_bonus.txt and bonus.txt. CO_LAB3_bonus.txt is contains machine code will be executed. bonus.txt is contained the MIPS assembly code and some annotation to let your code more readable.

E. Please use Modelsim as the simulation platform. You should make sure your design can output correct answer in the Modelsim environment.

F. You can modify the instruction file be read in “Test_Bench.v”(default: CO_LAB3_test_data1.txt). And TA will use more patterns to test your design.

6. Grade

A. **Don’t copy or your will get 0 point.**

B. Basic: 60 points, advanced: 30 points, bonus: 20 points

C. Report: 10 points – refer to the form in attach file, and must descript the job distribution.

D. Delay : 10% off/day

E. This Lab needs demo, demo time : 11/19 7:30 ~9:30 p.m. 、11/20 7:00~9:00 p.m.

(please fill the demo time on the door of EC447 at 11/4~6)

7. Hand in assignment

Please submit your file to e3.

Please compress your source file(not the whole Modelsim project), bonus file(.txt),and report into one single file.

The file should be named : student ID1_student ID2.rar **(Format must be correct or you will get some penalty)**

8. Q&A

If you have any queston, please go to course forum: https://sites.google.com/site/nctuco/q-a-forum