CMPEN 331 – Computer Organization and Design
Lab 6
Due Sunday April 22, 2018 11:59 pm (Drop box on Canvas)

In this lab, the students will obtain experience with implementation and testing jump and link and jump register commands using the Xilinx design package for FPGAs for the R-type and J-type instructions. It is assumed that students are familiar with the operation of the Xilinx design package for Field Programmable Gate Arrays (FPGAs) through the Xilinix tutorial available in the class website. You can use any information available in previous labs if needed.

1. **Instruction Fetch**
   The same as in previous lab.

2. **Instruction Execution**
   The same as in previous lab.

3. **R-Format Instructions**
   The same as in previous lab.

4. **I-Format Instructions**
   The same as in previous lab.

5. **Conditional Branch Instructions**
   The same as in previous lab.

6. **Jump Instructions**
   Figure 1 shows the design required for executing the jump instructions. This instruction uses neither the ALU nor the register file. The jump target address is obtained by concatenation of the high 4 bits of PC+4 and 2-bit lift-shifted 'addr.pcsrc' will be 11 to select the jump target address. In addition to what the j instruction does, the jal instruction saves the return address to register $31 of the register file. The number 31 does not appear in the instruction; we must create it by hardware. We use PC+4 as the return address in our single- cycle CPU design. Figure 2 shows the design required for executing the jal instruction:

   jal target # $31 ← PC + 4; PC ← target (address << 2).

   Figure 3 shows the design required for executing the jr instruction:

   jr rs # PC ← rs.

   The jump target address is obtained from register rs of the register file. pcsrc will be 10 to select the jump target address.

   Selection for Next PC
   The PC will be updated on the rising edge of the clock in the single-cycle CPU. If the current instruction is neither a branch nor a jump, PC+4 will be written to the PC. A 4-to-1 multiplexer can be used to select an address for the next PC as shown in Figure 4. The input 0 of the multiplexer is PC+4; input 1 is the branch target address; input 2 is the jump target address coming from the register rs of the register file; and input 3 is the jump target address coming from addr and PC+4. The 2-bit pcsrc is the selection signal of the multiplexer.

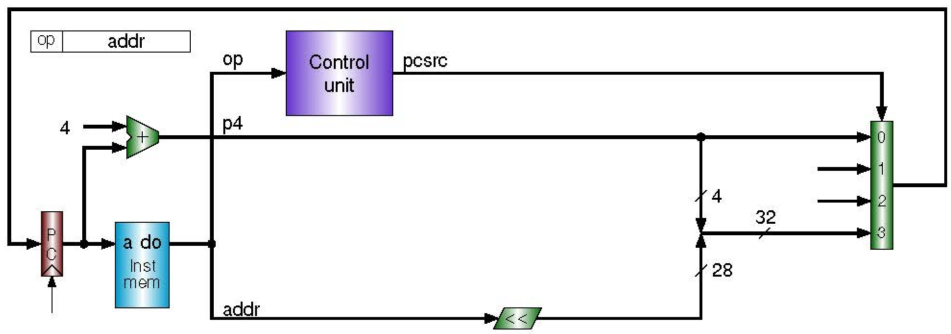

Figure 1 Block diagram for J-format using a multiplexer for ALU A-input selection

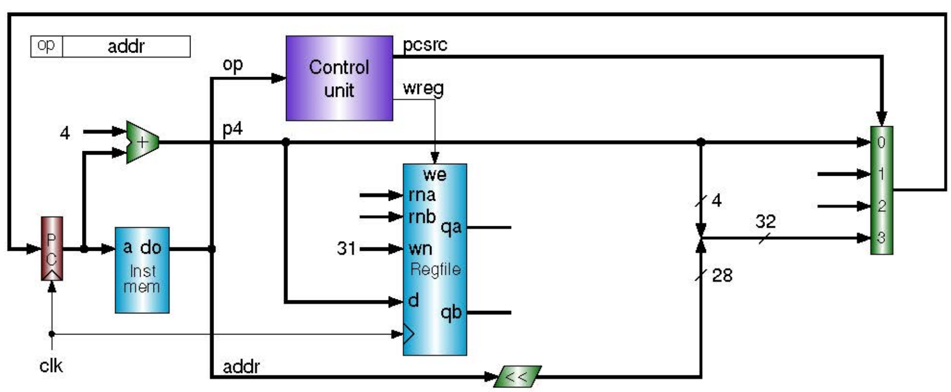

Figure 2 Block diagram for J-format jump and link instruction

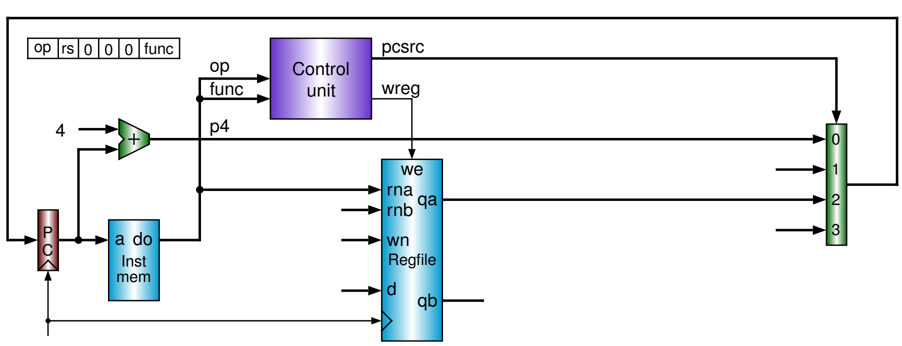

Figure 3 Block diagram for R-format jump register instruction

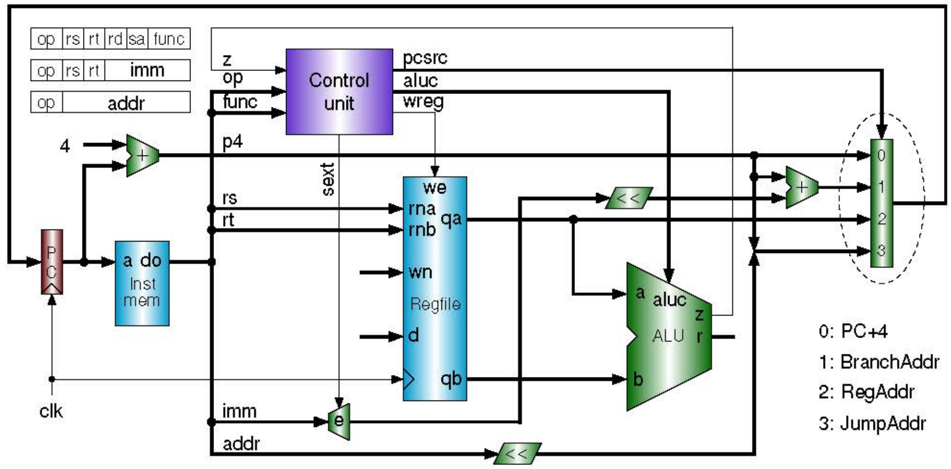

Figure 4 Using multiplexer for the next PC selection

7. **Table 1** lists the names and usages of the 32 registers in the register file.

Table 1 MIPS general purpose register

Register Name	Register Number	Usage
$zero	0	Constant 0
$at	1	Reserved for assembler
$v0, $v1	2, 3	Function return values
$a0 - $a3	4 – 7	Function argument values
$t0 - $t7	8 – 15	Temporary (caller saved)
$s0 - $s7	16 – 23	Temporary (callee saved)
$t8, $t9	24, 25	Temporary (caller saved)
$k0, $k1	26, 27	Reserved for OS Kernel
$gp	28	Pointer to Global Area
$sp	29	Stack Pointer
$fp	30	Frame Pointer
$ra	31	Return Address

8. **Table 2** lists some MIPS instructions that will be implemented in our CPU

Table 2 MIPS integration instruction

Inst.	[31:26]	[25:21]	[20:16]	[15:11]	[10:6]	[5:0]	Meaning
add	000000	rs	rt	rd	00000	100000	Register add
sub	000000	rs	rt	rd	00000	100010	Register subtract
and	000000	rs	rt	rd	00000	100100	Register AND
or	000000	rs	rt	rd	00000	100101	Register OR
xor	000000	rs	rt	rd	00000	100110	Register XOR
sll	000000	00000	rt	rd	sa	000000	Shift left
srl	000000	00000	rt	rd	sa	000010	Logical shift right
sra	000000	00000	rt	rd	sa	000011	Arithmetic shift right
jr	000000	rs	00000	00000	00000	001000	Register jump
addi	001000	rs	rt		Immediate		Immediate add
andi	001100	rs	rt		Immediate		Immediate AND
ori	001101	rs	rt		Immediate		Immediate OR
xori	001110	rs	rt		Immediate		Immediate XOR
lw	100011	rs	rt		offset		Load memory word
sw	101011	rs	rt		offset		Store memory word
beq	000100	rs	rt		offset		Branch on equal
bne	000101	rs	rt		offset		Branch on not equal
lui	001111	00000	rt		immediate		Load upper immediate
j	000010			address			Jump
jal	000011			address			Call

9. **Table 3** lists all the control signals of the single cycle CPU

Table 3 Control signals of single cycle CPU

Signal	Meaning	Action
`wreg`	Write register	1: write; 0: do not write
`regrt`	Destination register is `rt`	1: select `rt`; 0: select `rd`
`jal`	Subroutine call	1: is `jal`; 0: is not `jal`
`m2reg`	Save memory data	1: select memory data; 0: select ALU result
`shift`	ALU A uses `sa`	1: select `sa`; 0: select register data
`aluimm`	ALU B uses immediate	1: select immediate; 0: select register data
`sext`	Immediate sign extend	1: sign-extend; 0: zero extend
`aluc[3:0]`	ALU operation control	x000: ADD; x100: SUB; x001: AND x101: OR; x010: XOR; x110: LUI 0011: SLL; 0111: SRL; 1111: SRA
`wmem`	Write memory	1: write memory; 0: do not write
`pcsrc[1:0]`	Next instruction address	00: select PC+4; 01: branch address 10: register data; 11: jump address

10. **Table 4** shows the truth table of the control signals

Table 4 Truth table of the control signals

Inst.	z	wreg	regrt	jal	m2reg	shift	aluimm	sext	aluc[3:0]	wmem	pcsrc[1:0]
i_add	x	1	0	0	0	0	0	x	x000	0	00
i_sub	x	1	0	0	0	0	0	x	x100	0	00
i_and	x	1	0	0	0	0	0	x	x001	0	00
i_or	x	1	0	0	0	0	0	x	x101	0	00
i_xor	x	1	0	0	0	0	0	x	x010	0	00
i_sll	x	1	0	0	0	1	0	x	0011	0	00
i_srl	x	1	0	0	0	1	0	x	0111	0	00
i_sra	x	1	0	0	0	1	0	x	1111	0	00
i_jr	x	0	x	x	x	x	x	x	xxxx	0	10
i_addi	x	1	1	0	0	0	1	1	x000	0	00
i_andi	x	1	1	0	0	0	1	0	x001	0	00
i_ori	x	1	1	0	0	0	1	0	x101	0	00
i_xori	x	1	1	0	0	0	1	0	x010	0	00
i_lw	x	1	1	0	1	0	1	1	x000	0	00
i_sw	x	0	x	x	x	0	1	1	x000	1	00
i_beq	0	0	x	x	x	0	0	1	x010	0	00
i_beq	1	0	x	x	x	0	0	1	x010	0	01
i_bne	0	0	x	x	x	0	0	1	x010	0	01
i_bne	1	0	x	x	x	0	0	1	x010	0	00
i_lui	x	1	1	0	0	x	1	x	x110	0	00
i_j	x	0	x	x	x	x	x	x	xxxx	0	11
i_jal	x	1	x	1	x	x	x	x	xxxx	0	11

11. Initialize the first 11 words of the register file with the following HEX values:

00000000
A00000AA
10000011
20000022
30000033
40000044
50000055
60000066
00000002
80000088
90000099

12. Write a Verilog code that implement the following instructions using the design shown in Figure 2, 3, 4. The instruction memory should contain the following instructions. You should show in your simulation the output of the 4-to-1 multiplexer, qa and p4 signals shown in Figure 4. You also need to show the contents of register $ra at the output.

```
// (pc) label   instruction
// (00)          jal  2
// (04)          bne  $5, $0, L1
// (08) L1:       jr   $rs  (assume rs has the address of 8 "$t0")
```

13. Write a report that contains the following:
    a. Your Verilog® design code. Use:
        i. Device Family: Zynq-7000
        ii. Device: XC7Z010- -1CLG400C
    b. Your Verilog® Test Bench design code. Add "`timescale 1ns/1ps" as the first line of your test bench file. Only one test bench to test the top design is enough.
    c. The waveforms resulting from the verification of your design.
    d. The design schematics from the Xilinx synthesis of your design. Do not use any area constraints. Use a clock period of 1 µS as the timing constraint.
    e. Snapshot of the I/O Planning
    f. Snapshot of the floor planning

14. REPORT FORMAT: Free form, but it must be:
    a. One report per student.
    b. Have a cover sheet with identification: Title, Class, Your Name, etc.
    c. Using Microsoft word and it should be uploaded in word format not PDF. If you know LaTex, you should upload the Tex file in addition to the PDF file.
    d. Double spaced

15. You have to upload the whole project design file zipped with the word file.