CMPEN 331 – Computer Organization and Design
Lab 5
Due Friday April, 13 2018 at 11:59pm (Drop box on Canvas)
Item 16 is for Honor Option students only

In this lab, the students will obtain experience with implementation and testing Instruction Fetch, Instruction Decode and Execution of the MIPS five stages using the Xilinx design package for FPGAs for the I-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.

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

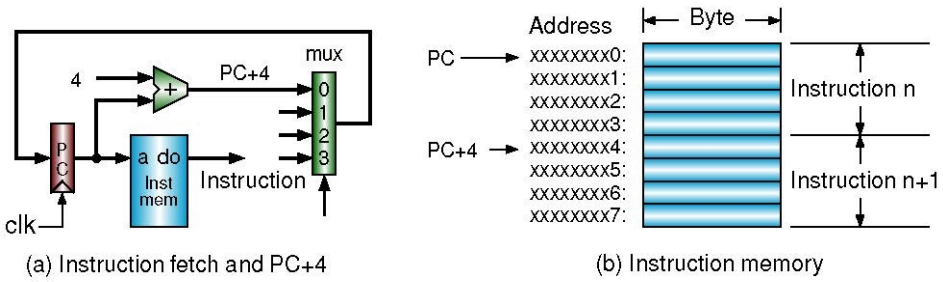

Figure 1 Block diagram of instruction fetch

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

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

4. **I-Format Instructions**

   Figure 2 shows the design required for executing the I-type instructions (addi, andi, ori, xori, lui, lw, sw). the 16-bit immediate must be extended to 32 bits. The lw and sw instructions access the data memory (Data mem component in the figure). The memory address is calculated by the ALU in the same way as the addi instruction does. The sw instruction writes the data held in the rt register to the data memory. For the lw instruction, the data word read from the data memory is written to the rt register of the register file. wmem (write memory), one of the control signals generated by the control unit, is the memory write enable signal.

5. **Conditional Branch Instructions**

   Figure 3 shows the design required for executing the (be and bne) instructions. The two operands in registers rs and rt are compared by the ALU. The result of the comparison z is sent to the control unit to determine whether to branch or not. If the branch is taken, the 2-bit pcsrc signal will be '01' to select the branch target address. An additional 32-bit adder is used to calculate the branch target address, which is the sum of the PC+4 (p4 in the figure) and the 2-bit left shifted sign extended immediate. These two instructions do not write the register file (wreg = 0).

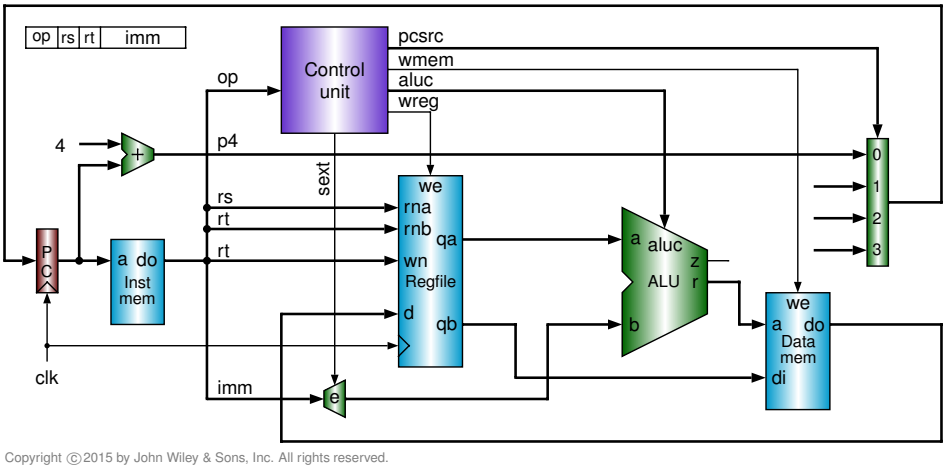

Figure 2 Block diagram for I-format arithmetic, logic, load and store instructions

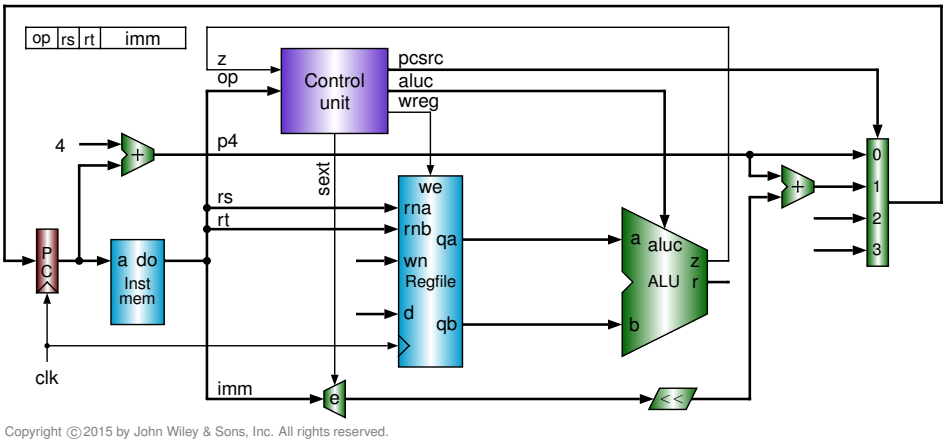

Figure 3 Block diagram for I-format branch instructions

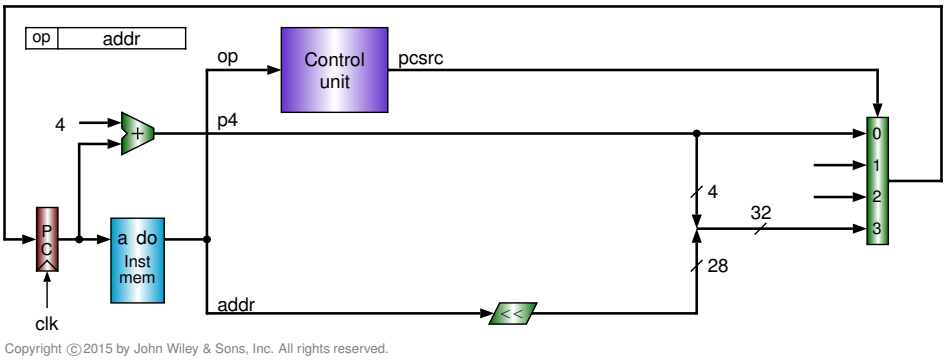

Figure 4 Block diagram for J-format using a multiplexer for ALU A-input selection

6. Jump Instructions

Figure 4 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.

Selection Signal: m2reg

The data that will be written to the register file may be the output of the ALU or the data in the data memory. The destination register number may be rd or rt. The (lw) instruction writes the data read from the memory to the register rt of the register file. Other instructions may write the output of the ALU to the register rd or rt of the register file. Two 32-bit 2-to-1 multiplexers are used for selecting the data that will be written to the register file as shown in Figure 5. If 'm2reg' is a '1', the data read from the memory is selected. Putting the memory module, Figure 6 shows the single cycle computer that consists of a single cycle CPU and two memory modules.

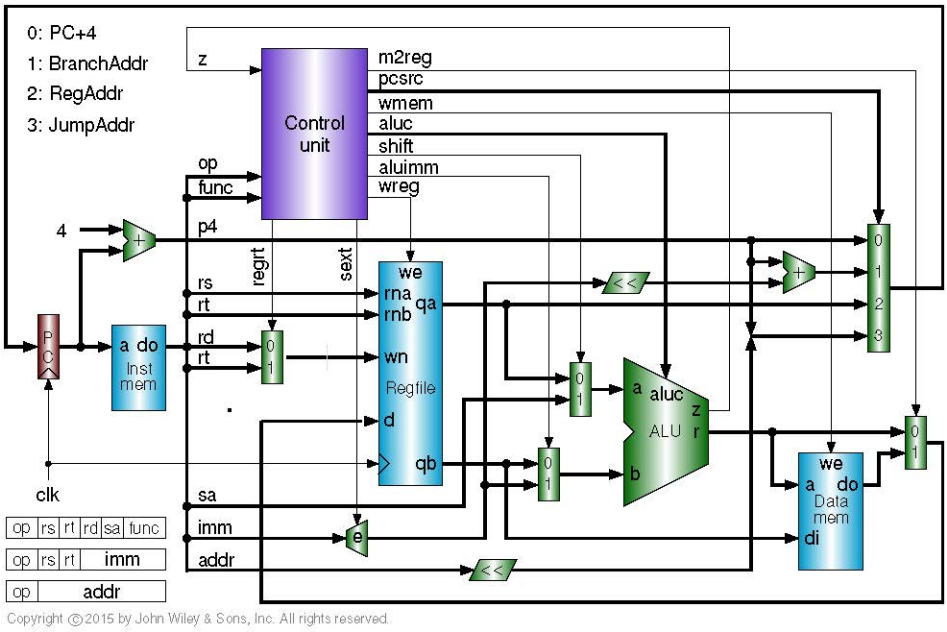

Figure 5 Block diagram for a combination of formats in this lab

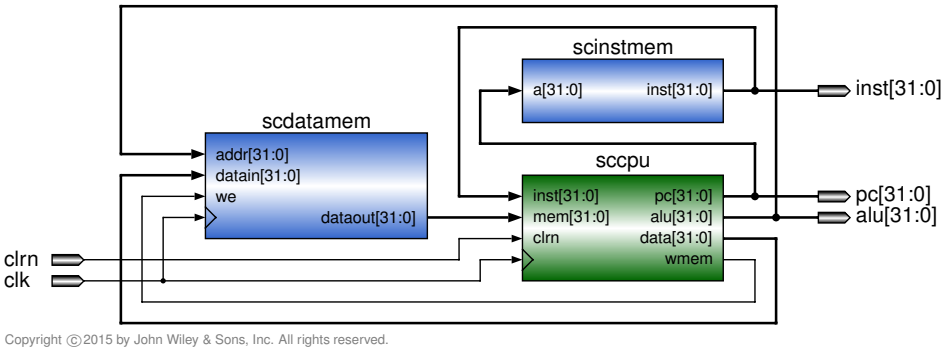

Figure 6 Block diagram of single cycle computer

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	x 0 0 0	0	0 0
i_sub	x	1	0	0	0	0	0	x	x 1 0 0	0	0 0
i_and	x	1	0	0	0	0	0	x	x 0 0 1	0	0 0
i_or	x	1	0	0	0	0	0	x	x 1 0 1	0	0 0
i_xor	x	1	0	0	0	0	0	x	x 0 1 0	0	0 0
i_sll	x	1	0	0	0	1	0	x	0 0 1 1	0	0 0
i_srl	x	1	0	0	0	1	0	x	0 1 1 1	0	0 0
i_sra	x	1	0	0	0	1	0	x	1 1 1 1	0	0 0
i_jr	x	0	x	x	x	x	x	x	x x x x	0	1 0
i_addi	x	1	1	0	0	0	1	1	x 0 0 0	0	0 0
i_andi	x	1	1	0	0	0	1	0	x 0 0 1	0	0 0
i_ori	x	1	1	0	0	0	1	0	x 1 0 1	0	0 0
i_xori	x	1	1	0	0	0	1	0	x 0 1 0	0	0 0
i_lw	x	1	1	0	1	0	1	1	x 0 0 0	0	0 0
i_sw	x	0	x	x	x	0	1	1	x 0 0 0	1	0 0
i_beq	0	0	x	x	x	0	0	1	x 0 1 0	0	0 0
i_beq	1	0	x	x	x	0	0	1	x 0 1 0	0	0 1
i_bne	0	0	x	x	x	0	0	1	x 0 1 0	0	0 1
i_bne	1	0	x	x	x	0	0	1	x 0 1 0	0	0 0
i_lui	x	1	1	0	0	x	1	x	x 1 1 0	0	0 0
i_j	x	0	x	x	x	x	x	x	x x x x	0	1 1
i_jal	x	1	x	1	x	x	x	x	x x x x	0	1 1

We take the lw instruction as an example to explain the truth table of Table 5. The ALU performs addition to calculate the memory address (aluc[3:0] = x000); one operand of the addition comes from register rs of the register file (shift = 1); the result will be written to register file (wreg = 1); it is the memory data (m2reg = 1); the destination register number is rt (regrt = 1); it does not write memory (wmem = 0); and the address of the next instruction is PC+4 (pcsrc[1:0] = 00)

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

00000000
A00000AA
10000011
20000022
30000033
40000044
50000055
60000066
70000077
80000088
90000099

12. Write a Verilog code that implement the following instructions using the design shown in Figure 5 with two outside memories as shown in Figure 6. Your top design should mimic Figure 6. The instruction memory should contain the following instructions:

```
// (pc) label   instruction
// (00) main:   lui  $1, 0
// (04)         ori  $4, $1, 80
// (08)         addi $5, $0,  4
// (0c)         add  $8, $0, $0
// (10)         sw   $2, 0($4)
// (14)         lw   $9, 0($4)
// (18)         sub  $8, $9, $4
// (1c)         addi $5, $0,  3
// (20) loop2:  addi $5, $5, -1
// (24)         ori  $8, $5, 0xffff
// (28)         xori $8, $8, 0x5555
// (2c)         addi $9, $0, -1
// (30)         andi $10,$9, 0xffff
// (34)         or   $6, $10, $9
// (38)         xor  $8, $10, $9
// (3c)         and  $7, $10, $6
// (40)         beq  $5, $0, shift
// (44)         j    loop2
// (48) shift:  addi $5, $0, -1
```

Notice that lui rt, constant

- Copies 16-bit constant to left 16 bits of rt
- Clears right 16 bits of rt to 0

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 and
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.

16. For students who took this class as an (honor option). In addition to all of the above requirements, use Xilinx's BMG (block memory generator) to design the instruction memory and data memory. And simulate your CPU designed in Verilog HDL of behavioral style.