CMPEN 331 – Computer Organization and Design
Lab 4
Due Saturday March 31, 2018 at 07:00 am (Drop box on Canvas)
10% Deduction for each day of late submission (maximum delay is 5 days)

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. 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

Instruction are stored in the instruction memory. CPU uses PC (program counter) as the address to read an instruction from the memory as shown in Figure 1. The PC will be implemented by 4 pointing to the next instruction because an instruction has 32 bits while the PC is a byte address. The multiplexer (mux) in the figure is used to select the next PC, which will be written to the PC at the rising edge of the clock.

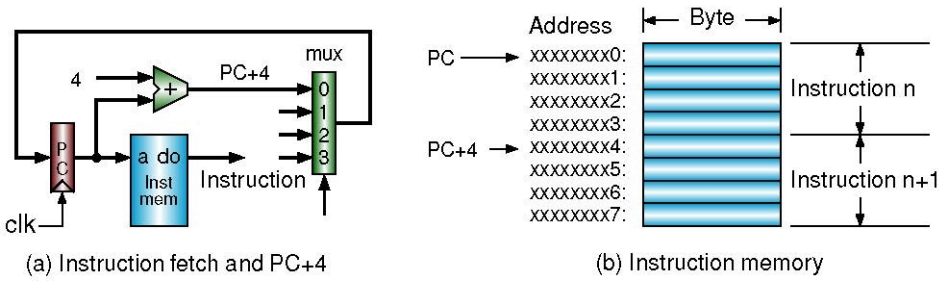


Figure 1 Block diagram of instruction fetch

2. Instruction Execution

After fetching the instruction, the CPU will execute it. The main components for executing an instruction include an arithmetic logic unit (ALU), a register file and a control unit.

3. R-Format Instructions

Figure 2 shows the design required for executing the R-type instructions (add, sub, and, or, xor). two source operands are read from qa and qb of the register file based on rs and rt, respectively. The two inputs are sent to the ALU for calculation. The difference between these five instructions is only on the ALU's operations. A control unit generates the control signals. PC source (pcsrc) is the selection signal of the multiplexer. ALU control (aluc) controls the operations of ALU. Write register (wreg) is a write enable signal if wreg is '1', the result calculated by the ALU will be written to the register rd of the register file. Figure 3 shows the design required for executing the R-type instructions (sll, srl). Different from figure 2, the input of the ALU in Figure 3 is the 5-bit shift amount (sa). The high 27 bits of ALU's input 'a' can be any value which will be ignored by the shift operations. The 'rs' field is not used. We use a multiplexer to select 'sa' or 'qa' of the register file based on the instruction as shown in Figure 4.

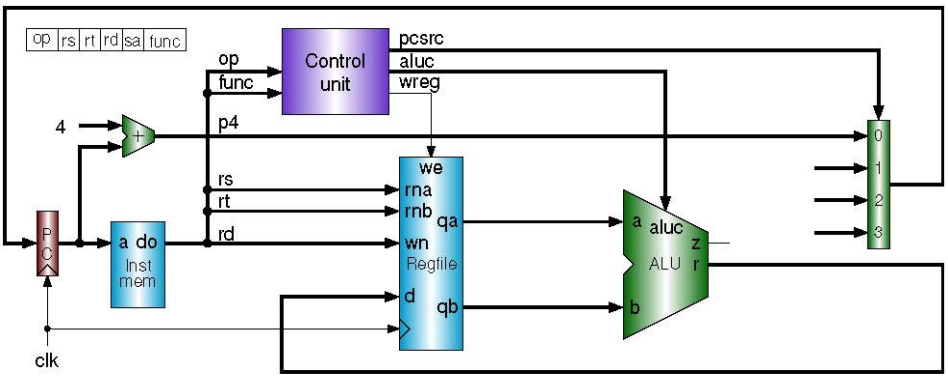


Figure 2 Block diagram for R-format arithmetic and logic instructions

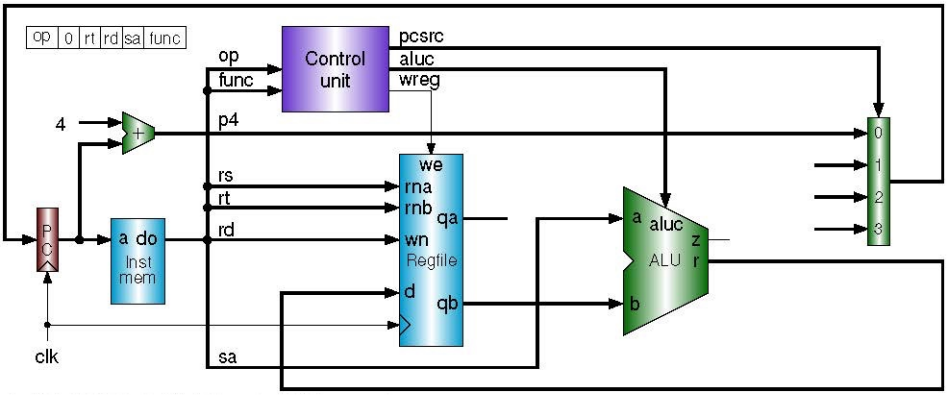


Figure 3 Block diagram for R-format shift instructions

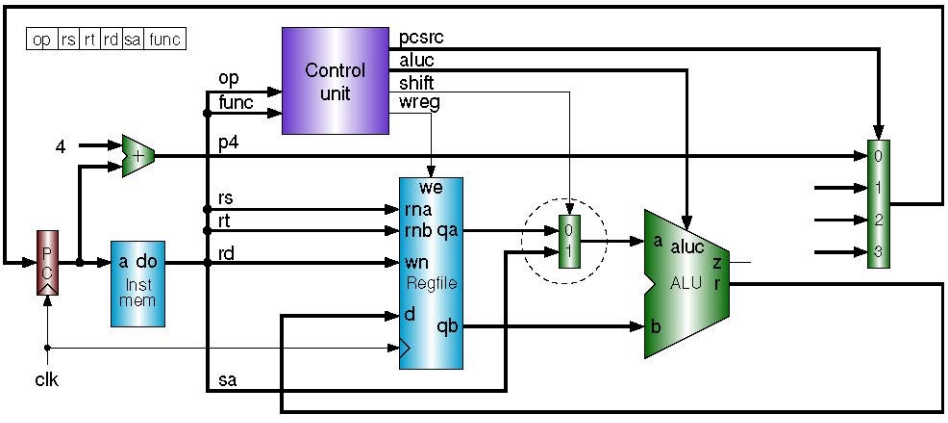


Figure 4 Block diagram for R-format using a multiplexer for ALU A-input selection

4. 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

5. Table 2 lists some MIPS instructions that will be implemented in our CPUthe

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

6. 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

7. Initialize the first 10 words of the register file with the following HEX values:

00000000
A00000AA
10000011
20000022
30000033
40000044
50000055
60000066
70000077
80000088
90000099

8. Write a Verilog code that implement the following instructions using the design shown in Figure 4. Write a Verilog test bench to verify your code:

```
32'h01244022 //sub  $t0, $t1, $a0
32'h01493025 //or   $a2, $t2, $t1
 32'h000543c0 //sll  $t0, $a1, 15
```

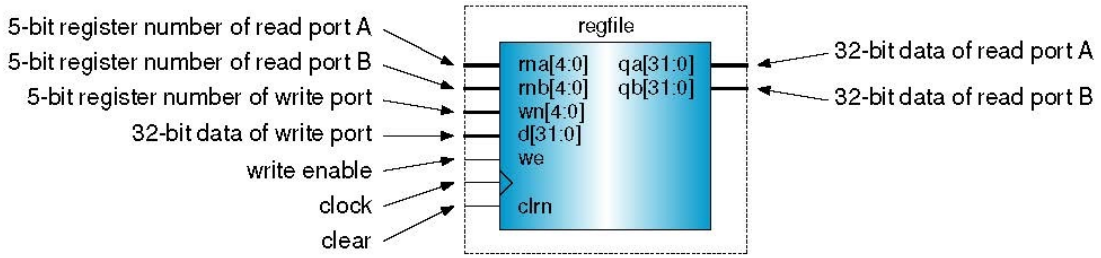
9. Write a report that contains the following:
   a. The corresponding state diagram.
   b. Your Verilog® design code. Use:
      i. Device Family: Zynq-7000
      ii. Device: XC7Z010- -1CLG400C
   c. Your Verilog® Test Bench design code. Add "`timescale 1ns/1ps" as the first line of your test bench file.
   d. The waveforms resulting from the verification of your design.
   e. 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.
   f. Snapshot of the I/O Planning and

g. Snapshot of the floor planning

10. 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

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

12. The following figure and the Verilog code should be a helpful guide to your lab:




```
module regfile (rna,rnb,d,wn,we,clk,clrn,qa,qb);      // 32x32 regfile
    input  [31:0] d;                                  // data of write port
    input   [4:0] rna;                                // reg # of read port A
    input   [4:0] rnb;                                // reg # of read port B
    input   [4:0] wn;                                 // reg # of write port
    input         we;                                 // write enable
    input         clk, clrn;                          // clock and reset
    output [31:0] qa, qb;                             // read ports A and B
    reg    [31:0] register [1:31];                    // 31 32-bit registers
    assign qa = (rna == 0)? 0 : register[rna];        // read port A
    assign qb = (rnb == 0)? 0 : register[rnb];        // read port B
    integer i;
    always @(posedge clk or negedge clrn)             // write port
        if (!clrn)
            for (i = 1; i < 32; i = i + 1)
                register[i]  <= 0;                    // reset
        else
            if ((wn != 0) && we)                      // not reg[0] & enabled
                register[wn] <= d;                    // write d to reg[wn]
endmodule
```

and its Verilog test bench

```
module regfile_tb;
    reg   [4:0] rna,rnb,wn;
    reg  [31:0] d;
    reg         we,clk,clrn;
    wire [31:0] qa,qb;
    regfile rf (rna,rnb,d,wn,we,clk,clrn,qa,qb);
    initial begin
           clk  = 0;
           clrn = 0;
```

```
            #2 clrn = 1;
               we   = 1;
               d    = 32'hffff0000;
               wn   = 0;
               rna  = 0;
               rnb  = 5'd31;
            #4 we   = 0;
            #2 we   = 1;
        end
        always #1 clk = !clk;
        always #2 d   = d + 1;
        always #2 wn  = wn + 1;
        always #2 rna = rna + 1;
        always #2 rnb = rnb + 1;
    endmodule
```