CSE341 Computer Organization Lab 3 Spring 2012

1. Purpose: Design and test a model of the single cycle datapath discussed in Chapter 4 of your text book.

2. Pre-requisites: Familiarize yourself with RTL (Register transfer language) and behavioral definition of circuits by using Verilog. See the Verilog tutorial in the Appendix of your textbook. Also for this exercise you may find several modules of code from the tutorial at this location.

3. Problem statement:

You are required to design a 32-bit datapath to realize a simple instruction set architecture consisting of three types of instruction: (1) R-type, (2) I-type: LW and SW (load and store), and (3) BEQ (Branch equal) only.

4. Problem description:

Study the instruction, code for the instructions and the bit pattern for each type of instructions as described in Chapter 4 of your text. Understand the various modules, instruction memory, PC register and the module that updates the PC, Register file, ALU, Data Memory and finally main control unit and ALU control unit. Use the schematics and tables in the Chapter 4 and in Appendix C.

- Assume a dual port memory that will allow read of instruction and load/store memory to happen in the same instruction cycle.
- You will also have to consider the timing and edge triggering of the transitions of states within an instruction cycle.
- Design the Verilog model for each of the modules and inner modules before assembling the modules. Make sure your test the inner modules before assembling an outer module. Test the outer module before putting them altogether.
- The final module will have to test for the instructions in the instruction set and output appropriate results in the tester.

4.1 Clock Module:

Design a clock module that generates a square wave as shown in figure 1. It should have just one output port which can be connected to the clock inputs of all the other modules that we are going to design.

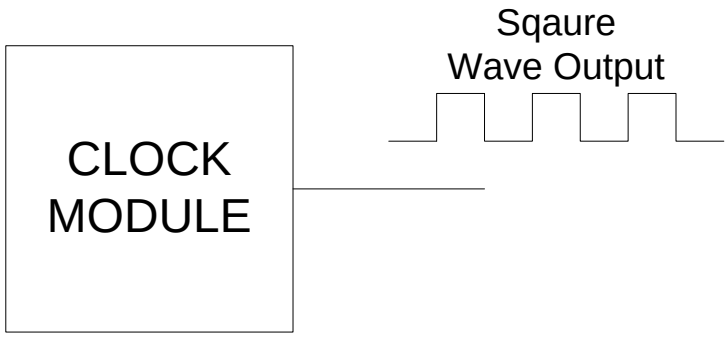

Figure 1: Clock module

Design a test bench for this module to check that the output is a 50% duty cycle square wave. Meaning, the duration for which the waveform is at logic 1 is equal to the duration for which the waveform is at logic 0. Suggestion: You can use: logic 1 duration = logic 0 duration = 5 time units

4.2 Memory Module:

Design a dual port memory module that is word addressable (i.e.) each memory location is 32-bits wide. The address lines are also 32-bits wide, hence there are 2^32 memory location (each is 32-bits wide) as shown in figure 2.

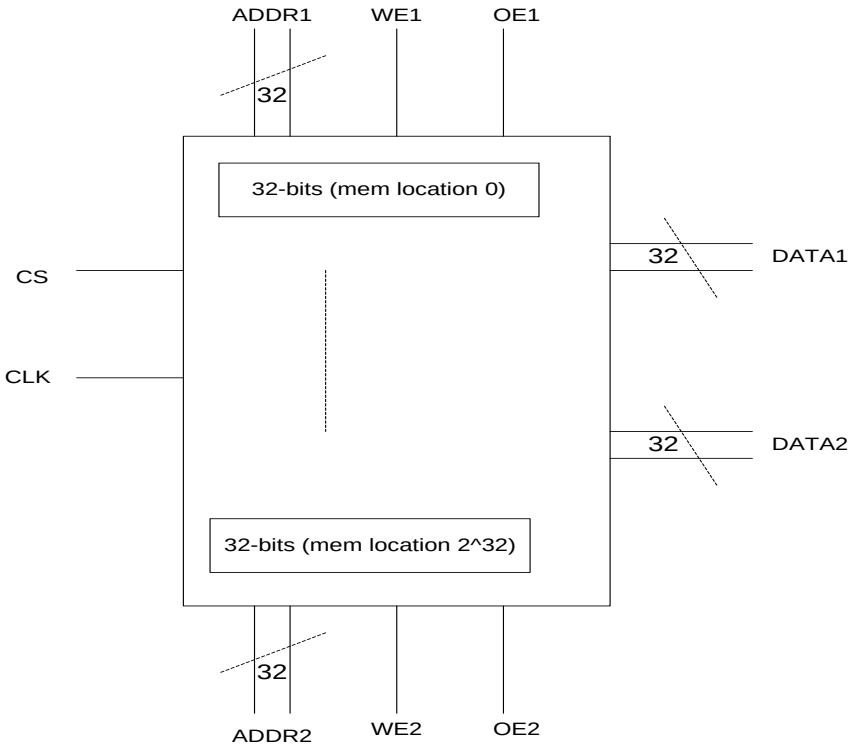

Figure 2: Dual Port Memory Module

The chip select (CS) line is the global ON/OFF control for the chip. You should be able to read at DATA1 by providing the memory address at ADDR1 and driving OE1 to logic 1 (WR1 should be driven to logic 0). You should be able to write by providing the memory address at ADDR1, driving WR1 to logic 1 and providing the data at DATA1. Port 2 will also operate similarly. NOTE: All reads and write should take place at the positive edge of the clock. Use the example for the RAM in the link provided above to design this module.

Design a test bench for the memory module as follows –

- You should write the bit pattern 10101010101010101010101010101010 to a memory location using DATA 1, ADDR1, WE1
- Wait for 20 units of time
- Read the same memory location and ensure that the output at DATA 1 is the above bit pattern.
- Repeat this for each memory location
- Repeat this for port 2

4.3 Processor:

The block diagram of the processor is below –

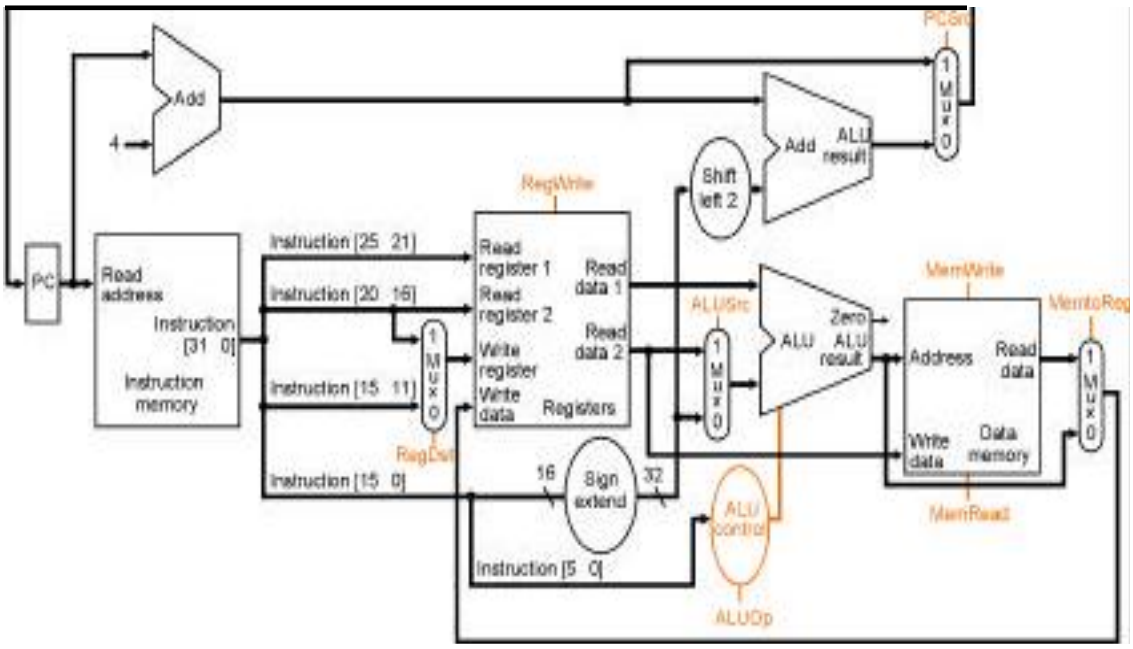

Figure 3: Block Diagram of the Datapath

The sub-modules of the processor as discussed during lecture and as shown in the figure 3 which is a copy of the on in the text.

- Instruction Fetch
- Instruction Decode module (Control Unit)
- Register File
- ALU
- Read and Write Memory (the actual module is outside the datapath).

5. Implementation details

5.1 Memory module (MM)

- The Instruction and Data memories although shown separately in the block diagram are in fact one physical module. This is the one we designed earlier.
- The register file will have 32 registers (0 through 31) and will be addressed using 5 bits in the instruction. Each registers is 32 bits wide.
- Use Port 1 of the memory module to fetch instructions.
- User Port 2 of the memory module to read and write data.
- All READ operations are performed at the POSITIVE edge of the clock and All WRITE operations are performed at the NEGATIVE edge of the clock. (Note : This holds only for operations within the processor and not for the memory module)
- Miscellaneous stuff will be explained in the recitation.

5.2 Instruction Fetch Module (IF)

- Consists of the PC and the two adder modules at the top of the diagram
- At the POSITIVE edge of each clock cycle the contents of the place the contents of PC on ADDR1 and set WR1 to logic 1 to read the instructions.
- At the NEGATIVE edge of each clock cycle, the adder modules compute PC+4 and PC is written back with the new value.
- If the instruction is BRANCH then the second adder module in the data path computes the new address
- Note that a 2x1 MUX is implemented to decide which value (PC+4 or the output of the second adder) is written into the PC.
- All of the above components comprise the IF module.
- Note that all you need to do is instantiate 2 adder (you already have the code for this), one register (PC) and one 2x1 MUX (already done) and wire them up properly to implement this module. Details are in figure 4.

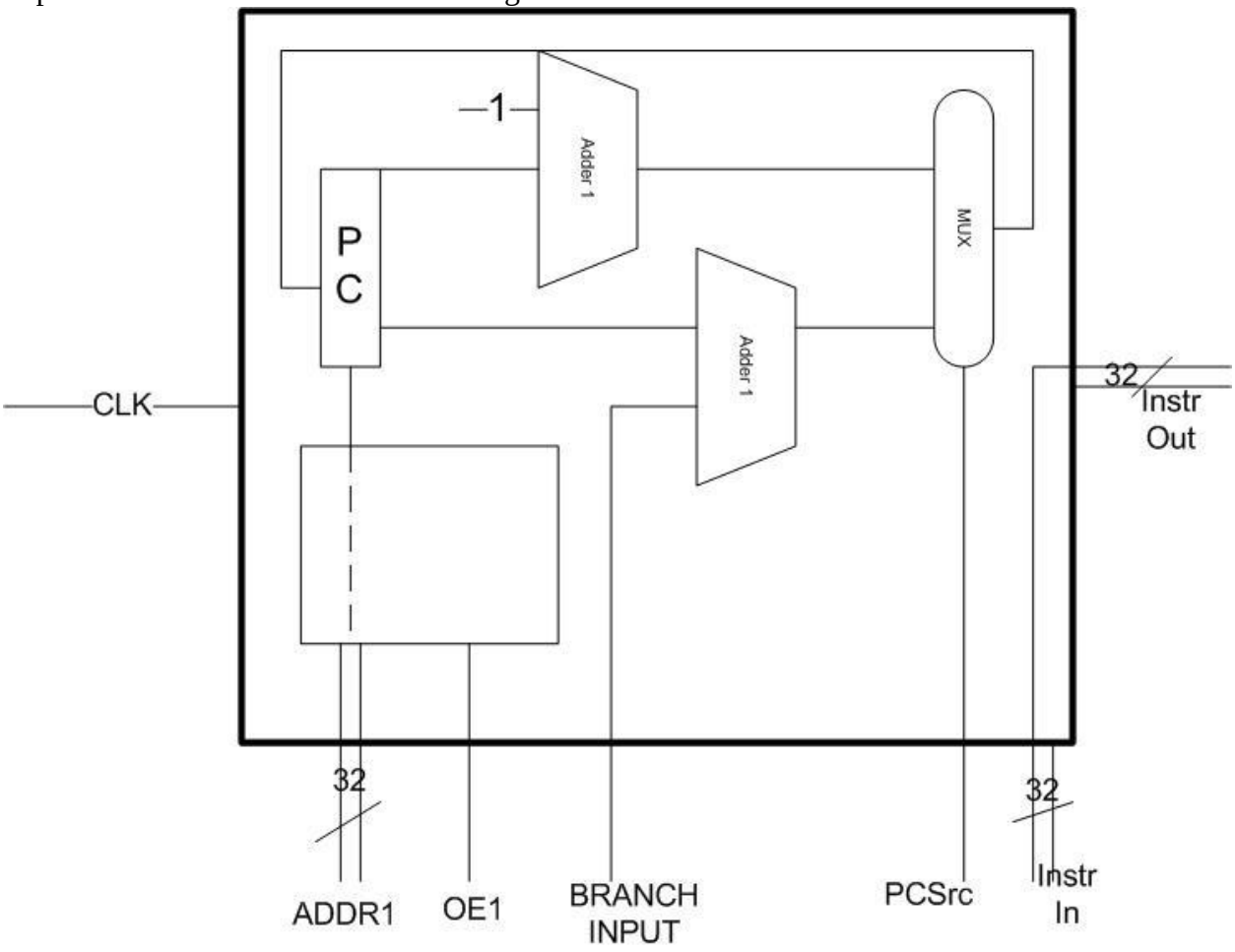

Figure 4: Block Diagram of the ALU

5.3 Instruction Decode Module (ID)

We will use the same instruction format as in MIPS and only the instructions we discussed during lecture. More specifically we will be implementing 6 instructions –

1) ADD <RS>, <RT>, <RD> - This instruction reads RT and RD, adds their contents and puts it into RS. The format of this instruction is

1 0 0 0 0 0	RS	RT	RD	0 0 0 0 0	0 0 0 0 0 0

RS, RT, RD are the register addresses from 0 through 31 (since we have 32 registers). Each field is 5 bits long.

2) SUB <RS>, <RT>, <RD> - This instruction reads RT and RD, computes RT – RD and stored it in RS. The format of this instruction is

1 0 0 0 0 1	RS	RT	RD	0 0 0 0 0	0 0 0 0 0 0

3) LW <RS>, IDX(<RT>) – This instruction reads the contents of RT, computes RT + IDX. Fetches the contents of the memory location (RT + IDX) and stores it into the register RS.

1 0 0 0 1 0	RS	RT	0 0 0 0 0	IDX	0 0 0 0 0 0

IDX is 5 bits wide

4) SW <RS>, IDX(<RT>) – This instructions computes RT + IDX and stores the contents of RS in the memory location (RT+IDX).

1 0 0 0 1 1	RS	RT	0 0 0 0 0	IDX	0 0 0 0 0 0

5) BEQ <RS>, <RT>, BRANCH – Checks if RS = RT. If true branches to PC + BRANCH and continues execution from there.

1 0 0 1 0 0	RS	RT	0 0 0 0 0	BRANCH	0 0 0 0 0 0

6) LI <RS>, IMM – Loads IMM into the register RS

1 0 0 1 0 1	RS	IMM – 21 bits

5.4 Register File (RF)

This module is implemented in a manner very similar to the memory module. We have 5 address lines and a total of 32 registers. The only difference is that the memory module can perform either a read or a write in one clock cycle. However, the register file should be able to read the contents of the register at the POSITIVE edge of the clock cycle and write into the register at the NEGATIVE edge of the clock cycle.

5.5 ALU module (EX)

- Consists of one full adder and one full subtractor modules as shown in figure 5.
- The Instruction Decode module (Control Unit) will select either the adder or subrtactor based on the instruction. Consists of one full adder and one full subtracter module and one comparator.
- Setup the decoder such that if the ALU Instr == 0 the adder is selected, 1 selects the subtracter and 2 selects the comparator
- The Z Output is 1 if both inputs are equal and 0 otherwise. This will be used for the branch instruction
- Make sure you OR the ouputs of the adder and subtracter and the output of the OR gate is connected to the ALU output.
- Design a suitable test bench that supplies two inputs and different ALU instr values(0,1,2) to select the different submodules and verify the output

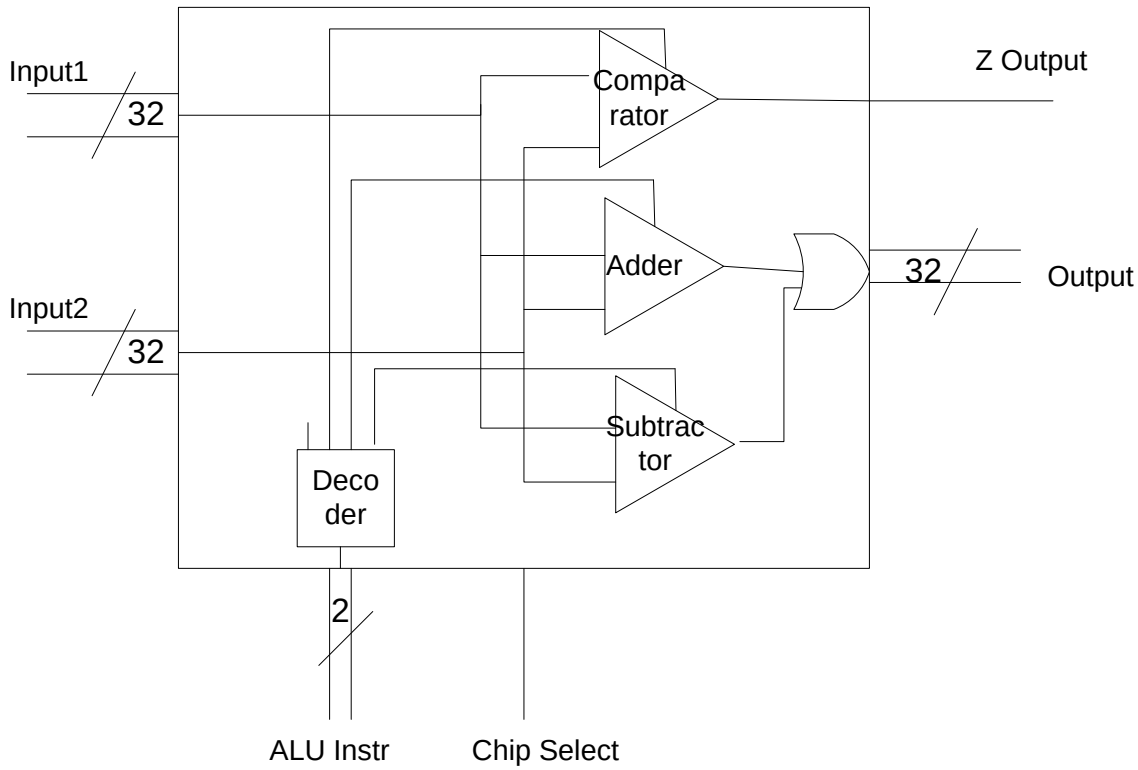

Figure 5 ALU Module

5.6 The Control Unit

The control unit will be similar to the one described in your text in the Appendix. We can reuse almost all of the design and implementation provided in the Appendix since the instructions handled by the processor described here are a subset of the instructions in the text.

6. Test bench Details

Test bench details will be given to you during recitation. Please attend recitation to get more detailed explanation of each of the module. Your TA has a complete worked out solution and he will be able to trouble shoot any problems you may during implementation.

Final Test Bench –

- Once you are done wiring up all the modules together as shown in the block diagram of the processor, you should implement a test bench that loads the following program to memory.

Mem Loc	Instr	Machine Code
0	li $t0, 50;	//10010100000000000000000000110010
1	li $t1, 10;	//10010100001000000000000000001010
2	beq $t1, $t0, jmp1;	//10010000000000000000100111000000
3	li $t2, 101;	//10010100010000000000000001100101
4	lw $t3, 0($t2);	//10001000011000100000000000000000
5	add $t4, $t1, $t0;	//10000000100000010000000000000000;
6	sub $t5, $t3, $t1;	//10000100101000110000100000000000;
7	li $t6, 102;	//10010100110000000000000001100110;
8	sw $t5, 0($t6);	//10001100101001100000000000000000;
9	jmp1:nop	//00000000000000000000000000000000

- We will assume that memory location 0-100 are for instruction and 101 onwards is for data.
- We will assumes that the address of the registers t0, t1, t2 … t6 in the register file is 0, 1, 2 …6 respectively.
- As shown above load the instructions into memory (the machine code for the instructions has been provided). This memory load is somewhat similar to what happens when BIOS runs on your laptops.
- You must also store the value ’20’ in the memory location 101, so that the lw instruction executes properly.
- Then activate the processor by setting the global chip select to 1. Now the processor is in free running mode, which means that it will execute the instructions sequentially.
- Monitor the contents of the registers t0 – t6 through the execution.
- Your final output should show the changes in the values of each of these register for each step of the execution.

7. Deliverables and Due dates:

We highly recommend an incremental development, module by module. We will develop this project in steps with submissions each week.

Module	Suggested Due date
Memory (Instruction and Data)	4/2
Register files + ALU module + PC Module	4/12
Control Unit	4/19
Test bench and complete testing	4/26

8. Your Notes