CS220: Introduction to Computer Organization

Lab #3

A Very Simple CPU

20 Sept 2011
BSV code due on: 29 Oct 2011, 11:59PM

1 Introduction

In this lab, you will design a simple processor. In particular, you will implement an ALU, a Register File, and the datapath for R-type instructions.

2 ALU

The ALU that you design will be an 4-bit ALU. It will take two 4-bit inputs for the operands of the computations. It will take 3-bit opcode to select which computation is to be performed. The output will be 4-bit result as primary output and 4 1-bit flags as described below.

- **Inputs:** A (4-bit), B (4-bit), opcode (3-bits)
- **Outputs:** Y (4-bits) , Flags: CF, ZF, SF, PF

The details of the flags are as follows.

Flag	**Description**
CF	**Carry Flag**. Set to 1 if there is carry out from addition/subtraction OR if the outgoing bit is 1 in case of left shift. In all other cases, CF is set to 0.
PF	**Parity Flag**. Set to 1 in case Y has even number of bits set to 1. It is 0 otherwise.
ZF	**Zero Flag**. Set to 1 in case Y is 8'b0000. It is set to 0 otherwise.
SF	**Sign Flag**. If Y treated a ssigned number is negative (which will be the case if most significant bit of Y is 1), SF is set to 1. It is set to 0 otherwise.

Here are the operations supported by the ALU:

Opcode	**Operation**	**Description**
3'b000	$Y = A + B$	Add
3'b001	$Y = A - B$	Subtract
3'b010	$Y = A + 1$	Increment
3'b011	$Y = A << B$	Left shift
3'b100	$Y = A >>_U B$	Right Shift Logical (Unsigned)
3'b101	$Y = A >>_S B$	Right Shift Arithmetic (Signed)
3'b110	$Y = A \ \& \ B$	Bitwise And
3'b111	NOP (Y = Don't Care)	RESERVED

2.1 Input specification for FPGA

The inputs A, B and opcode are to be provided using the slide switches. For this you will need to implement three registers which will provide the inputs of the ALU. Develop a module that will take the values of the switches and store them in the appropriate registers. Within this module instantiate the ALU. The slide switches will be set to a value and then a push button will be pressed to store it in the corresponding register.

- Push button BTN0 to strobe slide switches SW2:SW0 into register to contain opcode.
- Push button BTN1 to strobe slide switches SW3:SW0 into register A.
- Push button BTN2 to strobe slide switches SW3:SW0 into register B.

Since the ALU is a combinatorial block, whenever one of the inputs are changes, outputs may change.

Use SW7 for RST_N.

2.2 Output specification for FPGA

Output Y of the ALU is to be provided on the 7-segment LEDs. Flags will be put on four LEDs as :

CF:LD0	PF:LD1	ZF:LD2	SF:LD3

3 Register-File

Now you have to design a Register-File. This together with ALU will be used for designing a simple processor.

A register file consists of thirty-two 4-bit registers that can be read and written by supplying the necessary register number(s) to be accessed. It has two read ports and one write port, of which one, two or all three can be accessed simultaneously.

You have to create a separate module for the register-file. Write your BSV code for the register file using parameter keyword for defining static module parameters. Define two parameters called R_WIDTH as width of the registers in the register file and R_DEPTH as the number of bits needed to address a register (i.e. the number of registers = $2^{\text{R\_DEPTH}}$). For simulation, the code should work for any value of R_WIDTH and R_DEPTH. However, for integration with

ALU, and for FPGA, you will synthesize your circuit with R_WIDTH = 4 and R_DEPTH=4 (16 register each of size 4 bits).

The device has five inputs:

1. Address of read register no-1 (RS1)
2. Address of read register no-2 (RS2)
3. Address of Write register (RD)
4. Write data, if writes (DIN)
5. Write enable, whether to write (WE)

Each of the first three inputs specifies one of the 16 registers for either reading or writing. It has two output ports:

1. Read data 1, corresponding to RS1 (Dout1)
2. Read data 2, corresponding to RS2 (Dout2)

They contain the content of the registers read.

The register file also has clock signal input CLK and clear signal input RST_N. Note that, as you are designing register file in BSV, these signals will be generated automatically if you write `rule`-blocks properly.

3.1 Input specification for FPGA

SW3:SW0 determine the address of the registers in the register file. SW5:SW4 is used to select the source and destination register for the instruction as:

- Write-register RD, if SW5:SW4 is 11.
- Read-register RS1, if SW5:SW4 is 10.
- Read-register RS2, if SW5:SW4 is 01.
- No register is selected if SW5:SW4 is 00.

To load the value, Press BTN3. Thus for example, setting SW5:SW4 to 10, and pressing BTN3 will treat the value in slide-switches SW3:SW0 as RS1.

To feed the data input register, use BTN2. BTN3 must be low (not pressed) this time. Use SW3:SW0 to specify a value to feed. Data input register's value will be copied to Write Register (RD) when the operation is triggered.

Use BTN1 to trigger operations (read RS1, read RS2, write RD). While triggering the operation, slide-switch SW6 will indicate the write-enable signal (WE). Write to RD will be effective only if WE is enabled.

Use SW7 for RST_N.

3.2 Output specification for FPGA

The 7-segment display will show the content of the two read-registers separated by a dot "." (e.g. 8.F).

3.3 Notes

The register-file allows reading of two locations (maybe same) and writing at one location at the same time. If the write-location conflicts with either or both read-location(s), the old content should be fetched, before updating the location.

4 Processor

At this point, it will be useful to relate the register file to the processor. Assembly instructions are the instructions that can run on a processor. Consider an assembly instruction

```
add rs1, rs2, rd
```

This instruction tells the processor to add values in registers rs1, rs2 and put the result in register rd. Thus, the ALU will use rs1 and rs2 as the read registers from the register file, and give an output data dout1 + dout2 as the input (din) to the register file to store into the write register rd. The addresses rs1, rs2 and rd will all come from another entity in the processor, the instruction register (IR) (IR also stores the operation "add", and instructs the ALU to perform addition).

You will now integrate your ALU and Register-File, so that the inputs to ALU (A and B) come from register file (contents of RS1 and RS2). For filling in the register file, we will use the RESERVED opcode (3'b111) of ALU, which disconnects the output of ALU from input (Din) of Register-File, and allows us to input arbitrary value on Din.

Your final working should be as follows. First we initialize the register file to contain some valid values.

1. Set SW5:SW4 to 11 and select a Write Register (RD) using SW3:SW0. Press BTN3 to load the RD address.
2. Set SW3:SW0 to some value. Press BTN2 to load value in data input register.
3. Specify some arbitrary Read Register-1 and 2 (RS1 and RS2) addresses. Depending on your design, you may be able to skip this step.
4. Set SW2:SW0 to contain the RESERVED opcode (3'b111). Push button BTN0 to strobe SW2:SW0 into ALU as opcode. Detect this opcode as a special case where the input to write register RD will come directly from the data input register.
5. Set SW6 to 1 to enable write (WE). Press BTN1 to perform write to RD from data input register. Note that RS1 and RS2 may not be initialized yet, so the values read in this step may be invalid.
6. Repeat above steps fill several registers.

Now we perform the operations using ALU:

1. Specify RS1 and RS2 from any of the earlier registers filled. Also specify RD to store the result.
2. Set SW2:SW0 to contain a valid opcode (i.e., except 3'b111). Push button BTN0 to strobe SW2:SW0 into ALU as opcode.
3. Set SW6 to 1 to enable write (WE) if you want to update RD. Press BTN1 to perform the operation. Note that, if WE is 1, RD will be filled by the result of the operation performed by ALU.

4. Print output of RS1 (A), RS2 (B) , and the result (Y) on 7-segment LED display as (Y.A.B). The flags CF, ZF, PF and SF should be display on LEDs.

4.1 Notes

1. SW7 should still work as RST_N.
2. If any of the RS1/RS2 is same as the RD, first read old value and than update the value in RD.
3. Extra credits: If any register being read has not been written ever, show default value as "U" (Uninitialized) on 7 segment display.