CS223 Project Assignment
Simple Processor

Submission Deadline: 06/5/2021 – 8:30 am

For the course project, you are going to implement a simple programmable processor in Systemverilog. The processor will have a simple architecture but demonstrate your knowledge of designing datapath and controller of a processor. Processors support a set of instructions according to complexity of their design. The processor here will only support four simple instructions which are Load, Store, and Add and Subtract.

Overall Architecture

Figure1 illustrates an example general purpose processor.

In the control unit, you will have a program counter (PC) register to keep track of the next instruction that is going to be executed. Instruction register (IR) will fetch that "next" instruction from instruction memory. The controller FSM will decode the instruction in the IR and send the control signals to datapath accordingly. There are 2 additional memory units to register file here, data memory and instruction memory. Data memory is to provide additional space for data since register file offer very limited space, and instruction memory is where the program(instructions) to be run is stored.

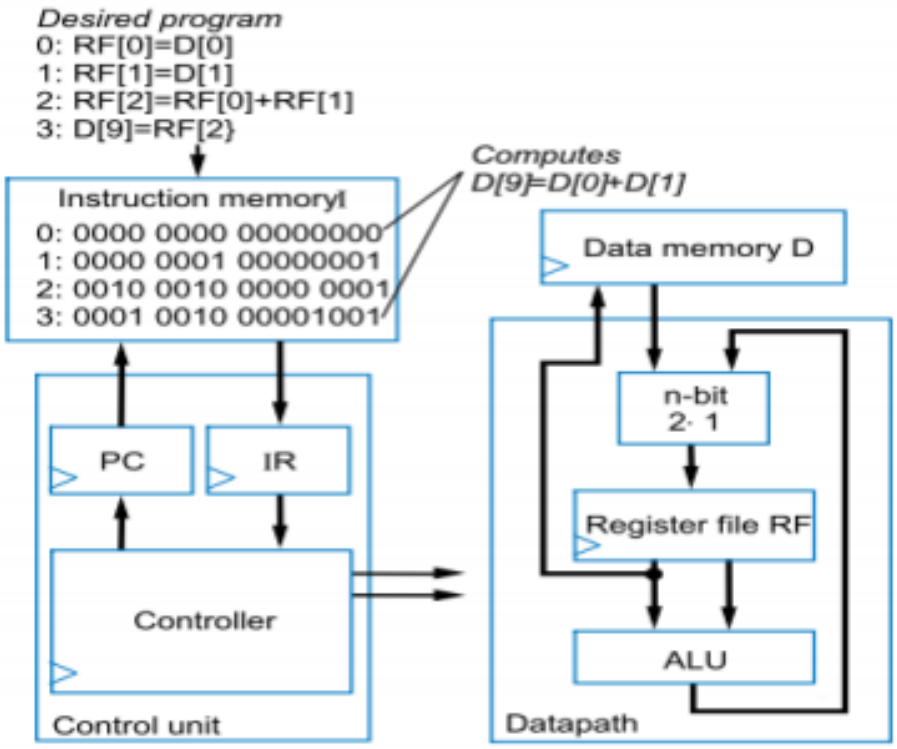


Figure 1- Example processor design

Instruction Set

In the following, we define how the instructions are represented using 12-bit and what each instruction is actually supposed to do. The processor should be able to execute 4 different types of operations: Load, Store, Add and Subtract. The most significant 3-bits of each instruction are used to identify the operation type(load,store,add or subtract) and the rest of the bits can be interpreted differently according to the instruction.

Load –000 xx r2r1r0 d3d2d1d0: This instruction specifies a move of data from the data memory location (D) whose address is specified by bits [d3d2d1d0] into the registerfile (RF) whose address location is specified by the bits [r2r1r0]. For example, the instruction "000 00 001 0010" specify a move of data memory location 0010, D[2], into register file location 001 (or RF[1]) – In other words, that instruction represents the operation RF[1] = D[2]. Notice that in load instructions, there are 2 redundant "don't care" bits, so instructions "000 00 001 0010" and "000 11 001 0010" should both do the same thing as they only differ in those bits.

Store –001 xx r2r1r0 d3d2d1d0: This instruction specifies a move of data in the opposite direction as the instruction load, meaning a move of data from the register file to the data memory. So, "001 00 001 0100" specify D[4] = RF[1]. Similar to load instruction, the 3^{th} and 4^{th} most significant bits are redundant.

Add – 101 wa2wa1wa0 rb2rb1rb0 ra2ra1ra0 : This instruction specifies an addition of two register-file specified by [rb2rb1rb0] and [ra2ra1ra0], with the result stored in the register file specified by [wa2wa1wa0]. For example, "010 010 000 001" specifies the instruction RF[2] = RF[0] + RF[1]. Note add is an ALU operation.

Subtract – 110 wa2wa1wa0 rb2rb1rb0 ra2ra1ra0 : This instruction specifies a subtraction operation of two register-file specified by [rb2rb1rb0] and [ra2ra1ra0], with the result stored in the register file specified by [wa2wa1wa0]. For example, "110 010 000 001" specifies the instruction RF[2] = RF[1] - RF[0]. Note that this is also again an ALU operation.

Datapath

In lab 5, you implemented the following design.

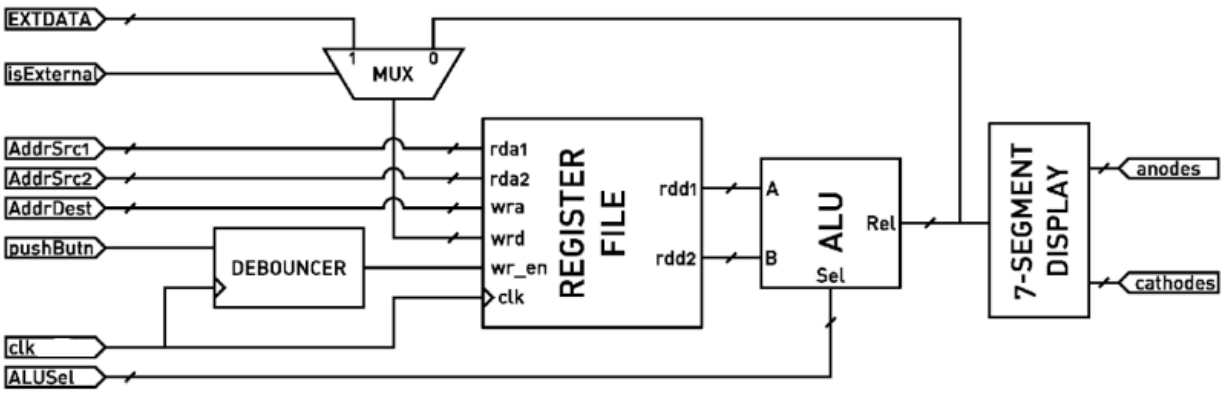


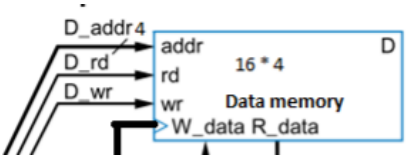
Figure 2-Processor Datapath Design(Lab 5)

You can use the same datapath you built in lab 5 with minor modifications:

1- One obvious difference is that only 2 ALU operations are needed in the new designs which are addition and subtraction. However, since the controller you build would be responsible for sending the ALU Sel bits, you can choose to use the same ALU in lab 5, provided that the controller never drives Sel bits to pick Increment and Or bitwise operations.

2- One important update is that you need to add a data memory unit. The data memory you'll add would work just like a register file but only differ in memory size and ports. It should have 16 slots, each holding 4 bit data. Unlike register file, it should only have 1 input for address selection, 4 bit (D_addr), which will be used both for reading and writing to memory(depending on the enable signals). It has two separate enable inputs for write (D_wr) and read (D_rd). The write data is 4 bit data shown as W_data. The read data is also a 4-bit output shown with R_data in the below figure. In the initial state, data memory should have values:

D[0] = 0x0, D[1] = 0x1, … D[14] = 0xE, D[15] = 0xF.



In the previous design, EXTDATA was obtained by switches. This time, R_data output of the data memory would be used as the EXTDATA to support load instruction. Similarly, R_data port should be wired to registerfiles corresponding output in order to support store instruction.

3- The inputs of the datapath previously were connected to switches in lab 5. This time, these inputs should be driven by controller outputs. Since the control signals come from the controller part, the debouncer also would not be directly connected to wr_en input of registerfile as in lab 5. Instead, wr_en signal will be obtained from the controller. Left pushbutton will be used to tell the processor to execute the next instruction, so the debouncer output will be connected to the controller.

4- 7-Segment Display was previously just showing the ALU output. It should again show the ALU output in its rightmost digit in hexadecimal base. In addition to that, leftmost 2 digits of the 7-Segment display should show the input values A and B of the ALU.

Instruction Memory

As you can see in the first figure, there are 2 important registers in the Controller module, which are PC and IR. PC is responsible for keeping the address of the next instruction that is going to be executed. When an instruction is executed, PC is incremented so that it will point to the next instruction in the program (Instruction memory). IR is the register where the instruction to be executed is fetched before being decoded. Instruction Memory(IM) is the memory where the program in machine code is being hold. Since the instruction set consists of 12-bit instructions, IR and Instruction Memory should also hold 12 bit values. IM should have 8 slots, therefor PC should be 3 bits (to specify the address). You should write some instructions into the instruction memory yourselves to test your processor. IM should be a read only memory, which means that it won't have any writeAddress/writeData ports but only readAddr and readData ports. Since it is a ROM, you need to write your instructions hardcoded on your code on Vivado. The processor should also be able to take instruction from switches. 12 of the switches should be used to define an instruction and on right button press, IR should fetch the instruction defined by switches instead of the pointed instruction in IM. In this case, PC shouldn't also be incremented.

Controller

The controller can be modeled as a state machine with states responsible for mainly fetching the instruction, decoding the instruction and executing the operation. In Fetch state, the next instruction should be written to IR register. In the next clock cycle, the instruction in IR should be decoded and next state should be determined according to the most significant 3 bits of the instruction. In other words, according to the three most significant bits of the instruction (opcode), we should move to one of the four states Load, Store or Add, Subtract. And finally, after we are done with that instruction, we go back to the Fetch state, waiting for next pushbutton press to execute the next instruction. Please note that moving from one state to another should be synchronized by the clock signal. The important thing about the controller of your processor is that, you should decide what set of control signals should be enabled or disabled in either state of the

controller. To make it clear for you, let's give an example. Assume after decoding the incoming instruction, you found out that the instruction is a Load instruction. Respectively, you will set your next state as Load state. In the Load state, in order your datapath works properly, the controller should give right values to the datapath. For the case of Load instruction, it should set the following signal as below:

D_addr =d3d2d1d0 / D_rd = 1 / D_wd = 0 / isExternal = 1 / RF_we = 1 / RF_waddr = r2r1r0

User Interface

-Left pushbutton will be used to execute next instruction in instruction memory. As in lab 5, a debouncer is needed. The processor should wait idle if no button is pressed.

-Right pushbutton will be used to execute the instruction defined by switches. As in lab 5, a debouncer is needed.

-12 rightmost switches on Basys3 will be used to provide user-defined instruction.

-SevenSegment Display will be used to show ALU inputs A,B in leftmost 2 digits and ALU result on the rightmost digit in hexadecimal. The remaining digit in between can be turned off.

-Leftmost switch on Basys3 for reset.

Project Report

In the project report, you need to submit the following:

a) Cover Page

b) Controller High-Level State Machine Diagram

c) Controller Block Diagram

d) Controller/Datapath Top Module Block Diagram

c) Testbenchs (This is optional, mainly for partial points if Basys3 doesn't work properly)