CS223 Project Assignment
Simple Processor
Submission Deadline: 09th May 2022 – 12:00 pm

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. The processor here will only support five instructions which are Load, Store, Add, Subtract, Multiply, Divide and Display.

Overall Architecture

Figure1 illustrates an example of a 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 the instruction memory. The controller FSM will decode the instruction in the IR and send the control signals to the datapath accordingly. There are two additional memory units to register files here, data memory and instruction memory. Data memory is to provide additional space for data since register files offer very limited space, and instruction memory is where the program (instructions) to be run is stored.

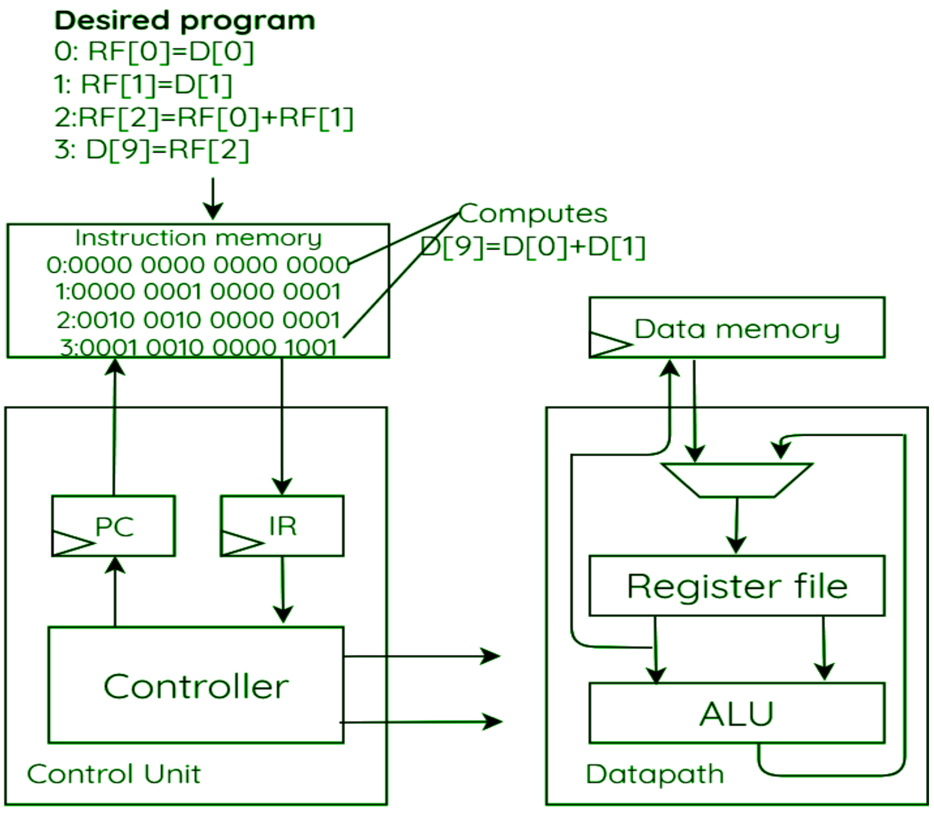

Figure1. Example processor design

Building blocks

1. **PC**: A register that holds the address of the next instruction to be executed.
2. **Instruction memory**: An array of registers that stores the instructions that will be execute.
3. **Register file**: An array of registers that keeps data.
4. **ALU**: a module that performs the arithmetic operations
5. **Data memory**: An array of registers that keeps data. (Additional space to store data)

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, Sub, Add, Mul, Div and Disp**. The most significant 3-bits of each instruction are used to identify the operation type (Load, Store, Sub, Add, Mul, Div and Disp) 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 register file (RF) whose address location is specified by the bits [r2r1r0]. For example, the instruction "000 00 001 0010" specifies 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 the load instruction, the 3th and 4th most significant bits are redundant.

Sub– 010 wa2wa1wa0 rb2rb1rb0 ra2ra1ra0: This instruction specifies a subtraction 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 sub is an ALU operation.

Add– 011 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]. The result can overflow 4 bits. In that case you can consider only the rightmost 4 bits of the result and discard the others. Note this is an ALU operation.

Mul– 100 wa2wa1wa0 rb2rb1rb0 ra2ra1ra0: This instruction specifies a multiplication 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] x RF[1]. In that case you can consider only the rightmost 4 bits of the result and discard the others. Note this is an ALU operation.

Div– 101 wa2wa1wa0 rb2rb1rb0 ra2ra1ra0: This instruction specifies a division of two register-file specified by [rb2rb1rb0] and [ra2ra1ra0], with the result stored in the register file specified by [wa2wa1wa0]. The resulting quotient should be the stored in the register file specified by [wa2wa1wa0] and the resulting remainder should be stored in the next register file. For example, “010 010 000 001” specifies the instruction RF[2] = RF[0] / RF[1]. Note this is an ALU operation.

Disp– 110 xxxxxx r2r1r0: This instruction is responsible for displaying the data value in the register-file specified by [r2r1r0].

Note: The Multiplication and Division operations must be implemented with respect to RTL design.

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 theinstruction to be executed is fetched before being decoded. Instruction Memory (IM) is the memory where the program in machine code is being held. Since the instruction set consistsof 12-bit instructions, IR and Instruction Memory should also hold 12-bit values. IM should have 8 slots, therefore PC should be 3 bits (to specify the address). The processor should also be able to take instructions from switches. Therefore we have an extra input named isexternal. If isexternal is 1, 12 of the switches should be used to define an instruction and on the clock edge, IR should fetch the instruction defined by switches instead of the pointed instruction in IM. In this case, the PC shouldn't also be incremented. That is why the write-enable bit of the PC register is isexternal invert.

Data memory

The data memory 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 the register file, it should only have 1input for address selection, 4 bit (M_add), which will be used both for reading and writing to memory (depending on the enable signals). It has two separate enable inputs for write (M_we) and read (M_re). The write data is 4-bit data shown as M_wd. The read data is alsoa 4-bit output shown with M_rd.

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 seven states Load, Store or Addition, Subtraction, Multiplication, Division, and Display. 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 for your datapath to work properly, the controller should give the right values to the datapath. For the case of Load instruction, it should set the following signal as below:

M_add =d3d2d1d0 / M_re = 1 / M_we = 0 / RF_we = 1 / RF_wd = r2r1r0

User Interface

- Left pushbutton will be used to execute the next instruction in the instruction memory. To avoid pressing multiple times 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. It is the signal isexternal. Here also, a debouncer is needed
- Middle pushbutton will be used to load the instruction that is specified by the user to the back of the queue in the instruction memory.
- Upper pushbutton will be used to load the data value that is specified by the user to the register-file address that is also specified by the user.
- Lower pushbutton will be used to clear everything existed in the processor and reset the controller.
- 12 rightmost switches on Basys3 will be used to provide user-defined instruction.
- 7 leftmost switches on Basys3 will be used to provide user-defined data value along with the register-file address where the 4 leftmost switches correspond to the data value and the remaining ones correspond to the register-file address. For instance, if the user inputs 1010110, RF[6] should get the decimal value 5.
- SevenSegment Display will be used to show ALU inputs and outputs. The inputs A, B should be displayed in the leftmost 2 digits. If the ALU operation is division, the resulting quotient and remainder should be displayed in the rightmost 2 digits. For other operations, the result should be displayed on the rightmost digit. The remaining digit should be turned off. All the displays should be in decimal.
- SevenSegment Display will also be used for the **Disp** instruction. When the instruction is called, the output should be displayed in the rightmost digit in decimal. ,

Project Report

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

a) Cover Page

b) RTL schematics for Multiplication and Division operations

c) Controller High-Level State Machine Diagram

d) Controller Block Diagram

e) Controller/Datapath Top Module Block Diagram

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