CS223 Project Assignment
Simple Processor
Project Report Submission Deadline: 29th May 2023 – 09:00 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. The processor here will only support seven instructions which are Load, Store, Add, Subtract, Multiply, Divide, and Display.

Overall Architecture

Figure 1 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. Register file, data memory will operate as a stack abstract data type and serve with respect to the last-in, first-out (LIFO) paradigm. Therefore, there will not be access to data directly by the address. Else, reading and writing data will be performed using the "**pop**" and "**push**" stack operations. Instruction memory will operate as a queue abstract data type and serve with respect to the first-in, first-out (FIFO) paradigm.

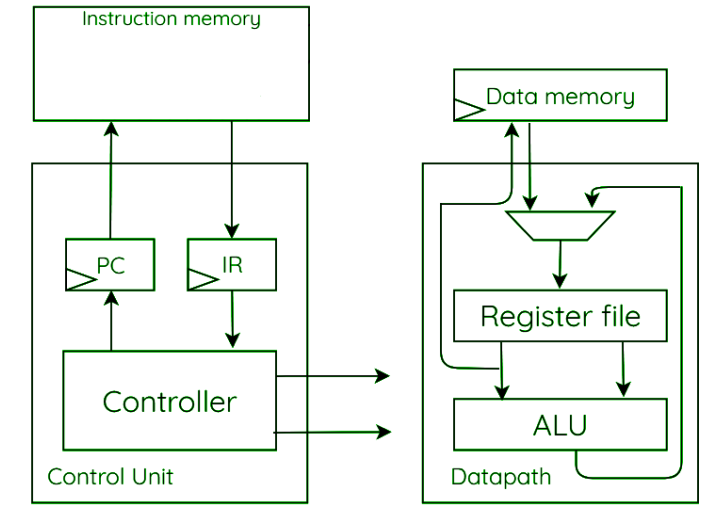

Figure1. Example processor design

Project Report Submission Deadline: 29th May 2023 – 09:00 am

Building blocks

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

Instruction Set

In the following, we define how the instructions are represented using 3-bit and what each instruction is actually supposed to do. The processor should be able to execute seven different types of instructions: **Load, Store, Sub, Add, Mul, Div and Disp**.

Load– 000: This instruction specifies a move of data from thedata memory into the register file. The most recently added data in the data memory should be popped, then it should be pushed to the register file.

Store– 001: This instruction specifies a move of data from the register file into the data memory. The most recently added data in the register file should be popped, then it should be pushed to the data memory.

Sub– 010: This instruction specifies the subtraction of two data values stored in register file. The two most recently added data in the register file should be popped and subtracted, then the resulting difference should be pushed to the register file. The first popped value should be the minuend and the second popped value should be the subtrahend.

Add– 011: This instruction specifies the addition of two data values stored in register file. The two most recently added data in the register file should be popped and summed up, then the resulting sum should be pushed to the register file.

Mul– 100: This instruction specifies the multiplication of two data values stored in register file. The two most recently added data in the register file should be popped and multiplied, then the resulting product should be pushed to the register file.

Div– 101: This instruction specifies the division of two data values stored in register file. The two most recently added data in the register file should be popped and divided, then the resulting quotient should be pushed to the register file. The first popped value should be the dividend and the second popped value should be the divisor.

Disp– 110: This instruction is responsible for popping the most recently data in the register file and display it.

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

Note: Except for the Div instruction, the instructions should operate on 2's complement signed 4-bit data values. On the other hand, you can consider the operands as unsigned values for the division operation.

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 held. Since the instruction set consists of 3-bit instructions, IR and Instruction Memory should also hold 3-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.

Register File

The register file should have 8 slots, each holding 2's complement signed 4-bit data. The address selection for reading and writing data should be done inside the stack-based memory architecture. There shouldn't be any inputs for the address selection. For writing data, there should be an enable signal, RF_we. The write data is 2's complement signed 4-bit data which can be shown as RF_wd. It can be considered as an input to the register file. The read data are 2's complement signed 4-bit data which can be shown as RF_d1 and RF_d2. They can be considered as outputs of the register file.

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 2's complement signed 4-bit data. Just like the register file, the address selection for reading and writing data should be done inside the stack-based memory architecture. There shouldn't be any inputs for the address selection. Unlike the register file, there should be two separate enable signals for writing (M_we) and reading (M_re) data. The write data is 2's complement signed 4-bit data which can be shown as M_wd. It can be considered as an input to the data memory. The read data is also 2's complement signed 4-bit output which can be shown as M_rd. It can be considered as an output of the data memory.

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_re = 1 / M_we = 0 / RF_we = 1

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. Here also, a debouncer is needed.
- Upper pushbutton will be used to push the data value that is specified by the user on top of the register-file. Here also, a debouncer is needed.
- Lower pushbutton will be used to clear everything existed in the processor and reset the controller. Here also, a debouncer is needed.
- 3 rightmost switches on Basys3 will be used to provide user-defined instruction.
- 4 leftmost switches on Basys3 will be used to provide user-defined data.
- SevenSegment Display will be used for **Sub, Add, Mul, Div, and Disp** instructions.
    - For **Sub**, **Add, Mul, and Div** instructions, 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.
    - For **Disp** instruction, the data value that is the most recently added to the register file should be popped and displayed in the rightmost digit.

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)