Project 1 - Design of an 8-bit processor (non-pipelined)

Atul Arvind Singh
50425045
atularvi@buffalo.edu

Chitravardhini Bellamkonda
50420023
chitrava@buffalo.edu

Shreyas Athreya Venkatesh
50366325
venkate7@buffalo.edu

Yash Rathi
50366308
yrathi@buffalo.edu

Contents

1 Problem Statement

In this group project, our task was to design and simulate a simple 8-bit processor without pipelining. To design the processor, the programming lnaguage Verilog HDL and the program Vivado Design Suite were used. The individual components were designed using behavioral modelling. We were only asked to implement R-type instruction (add) and I-type instruction (load and store word) only.

2 Processor Details

2.1 Instruction set

The 8-bit instruction formats for R-type and I-type instructions are as follows:

R-type instructions:

Opcode			Rt/Rd	Rs	Unused		
7	6	5	4	3	2	1	0

I – type Instructions:

Opcode			Rd	Rs	Immediate		
7	6	5	4	3	2	1	0

Figure 1: Instruction set (Reference: Project details)

The instructions to be implemented are:

Instruction	Opcode
lw	001
add	010
sw	100

Figure 2: Instruction and opcode (Reference: Project details)

2.2 Short description of every component in your design

2.2.1 Program Counter

The Program Counter (PC) is a register that holds the address of the instruction that is currently being executed. The output goes to PC Adder and to Instruction Memory. Initially when the processor is reset, the output is the 0th instruction (or the first instruction). Then, the next instruction (from PC adder) is passed as input to PC from the next cycle.

2.2.2 PC Adder

PC adder takes output from PC as the input and add one to the value. This value is then updated to the program counter value.

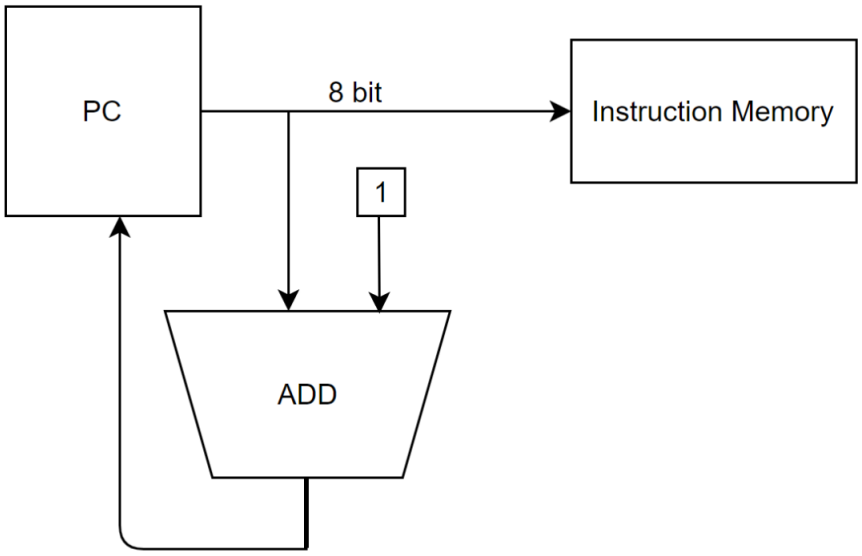

Figure 3: Program Counter (Reference: Project recitation)

2.2.3 Instruction Memory

A list of instructions is stored in instruction memory. It receives the PC output, which points to the instruction to be executed, and gives the matching instruction. In instruction memory, you get an address/index as input and look up the for the opcode of the instruction stored in that address/index of the memory/array.

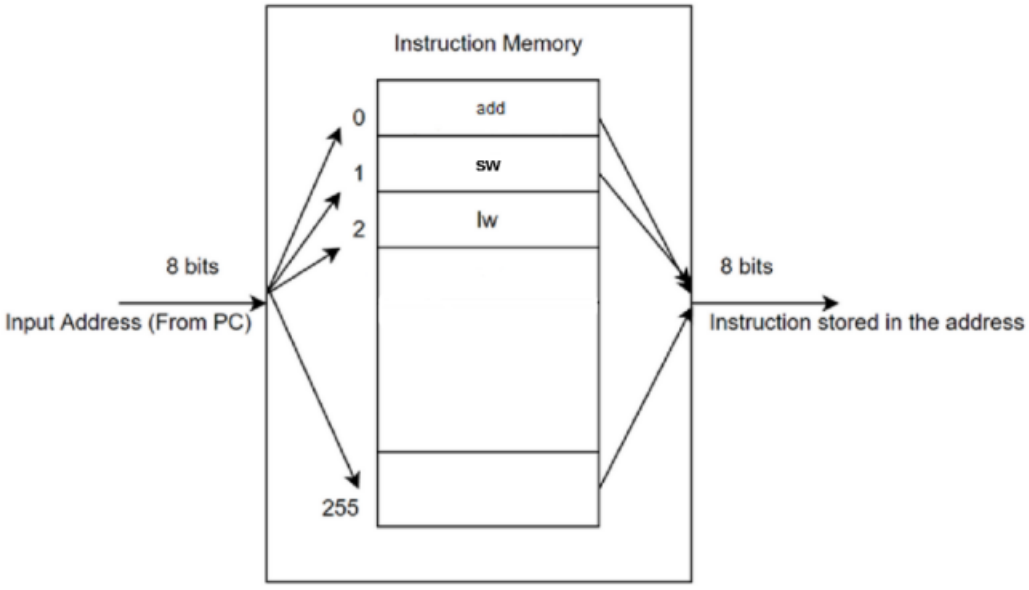

Figure 4: Instruction memory (Reference: Project details)

2.2.4 Control Unit

The control unit controls how the processor operates by translating instructions into timing and control signals. The control unit receives the opcode and distributes values to a number of further components, instructing those components on how to behave in response to that instruction.

2.2.5 Register File

The register file is an array containing all of the general-purpose registers. The register file will contain two register $s0 ans $s1 to read and write values to and from i.e. all the operations will be done on only these 2 registers.

2.2.6 Sign Extender

Sign Extender unit takes the 3 bit immediate value as the input and returns an 8 bit value b adding 0s to the MSB.

2.2.7 Arithmetic Logic Unit (ALU)

ALU carry out arithmetic and logical processes. It stands as an important building block of any processor. The operands of an ALU are the input data, and the operation to be carried out on the operands is determined by the instruction/operation code, which results in an ALU output. The ALU receives two values: one from the register and the other from the sign extender or the second register (decided by MUX). The third input from the control unit to the ALU determines how the ALU will compute the total of these values.

2.2.8 Multiplexer (MUX)

Since 2 cables cannot be joined as it is, we use multiplexer there. They work like switches that enable the processor to choose data from many sources.

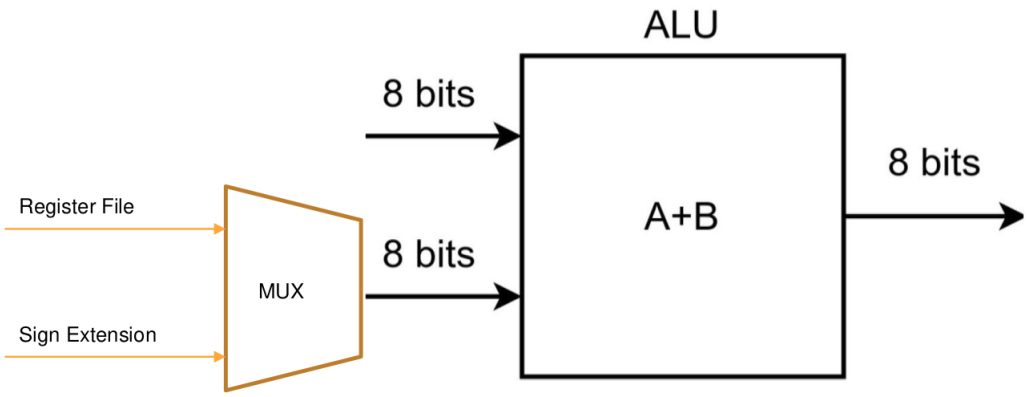

Figure 5: ALU (Reference: Project recitation)

2.2.9 Data Memory

We store our variables, such as data values or addresses, in data memory. During load and store operations, we can read and write values, accordingly. We have an 8-bit memory with 256 possible storage values. However, for simulation reasons, we only utilized 100 memory values because 256 was making Vivado run more slowly. ALU and register values are read into the data memory. Additionally, it accepts two values from the control system. Depending on the values from the control unit, we will either read data from the specified location and output it, or we will write the data from the register into data memory at that address (the value from ALU).

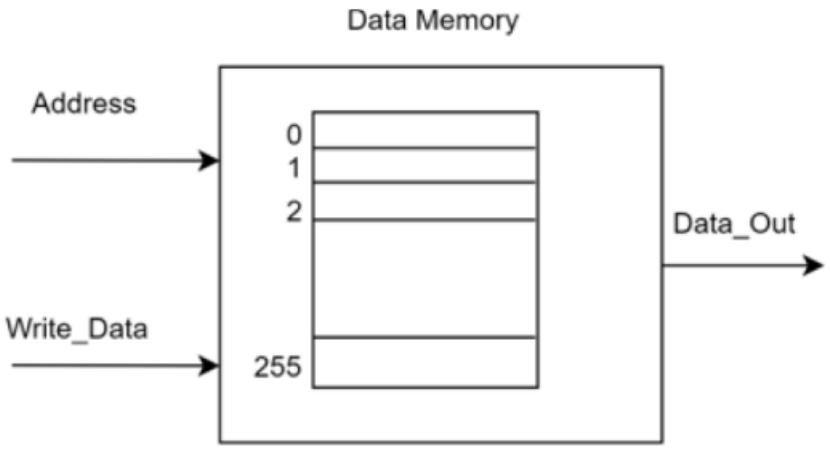

Figure 6: Data memory (Reference: Project details)

2.3 Schematic of your datapath and control path

2.3.1 Datapath

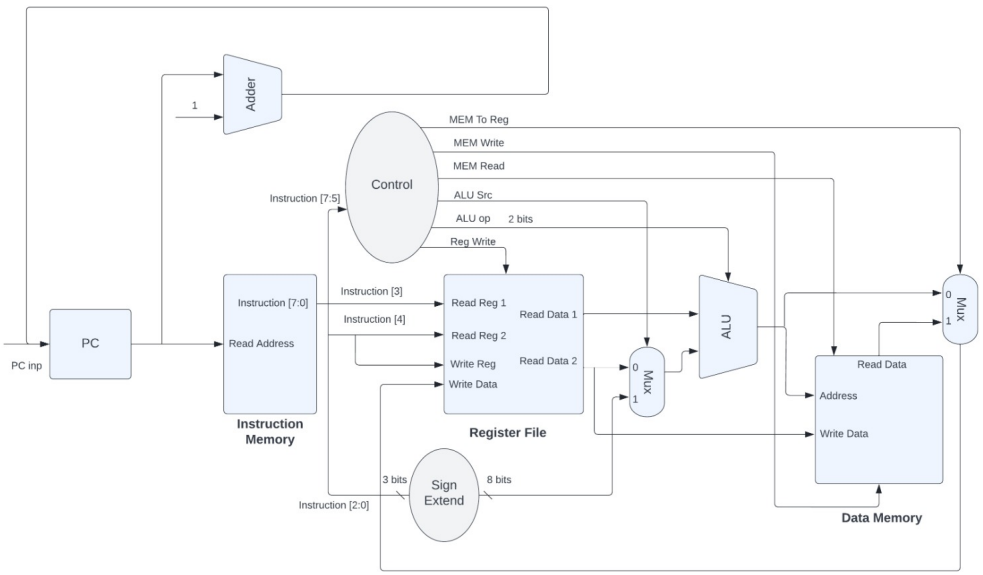

Figure 7: Datapath

2.3.2 Add instruction Datapath

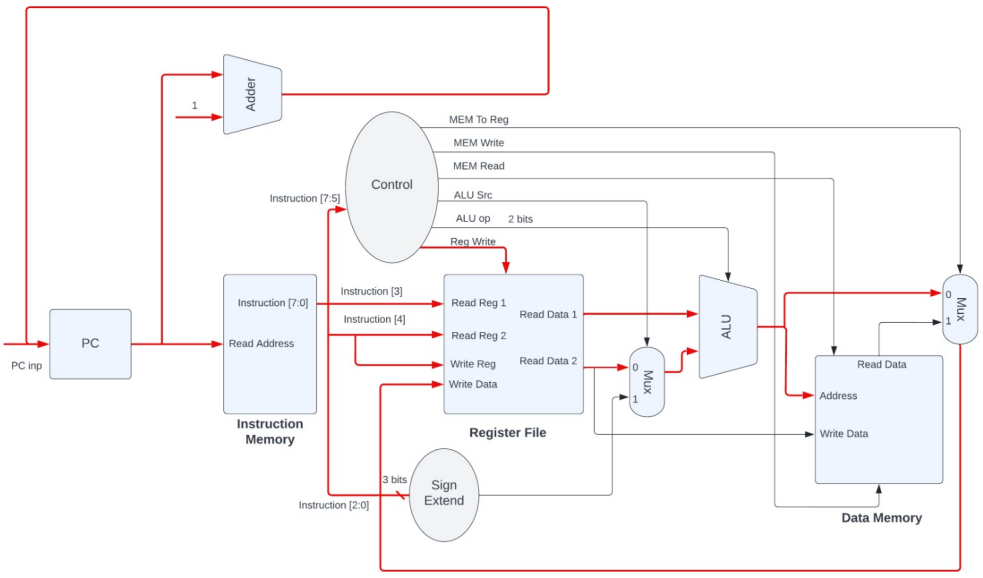

Figure 8: Add instruction Datapath

2.3.3 Load instruction Datapath

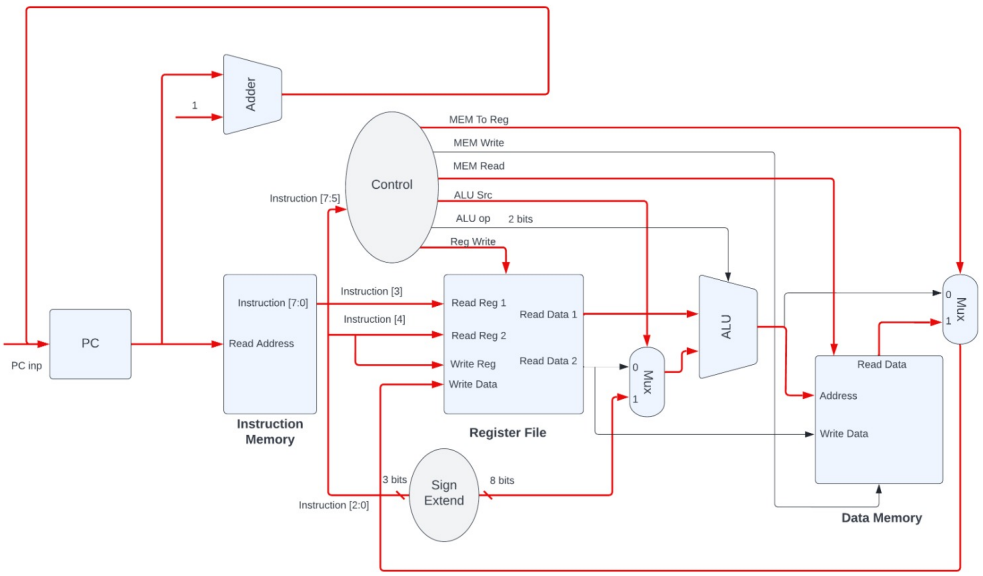

Figure 9: Load instruction Datapath

2.3.4 Store instruction Datapath

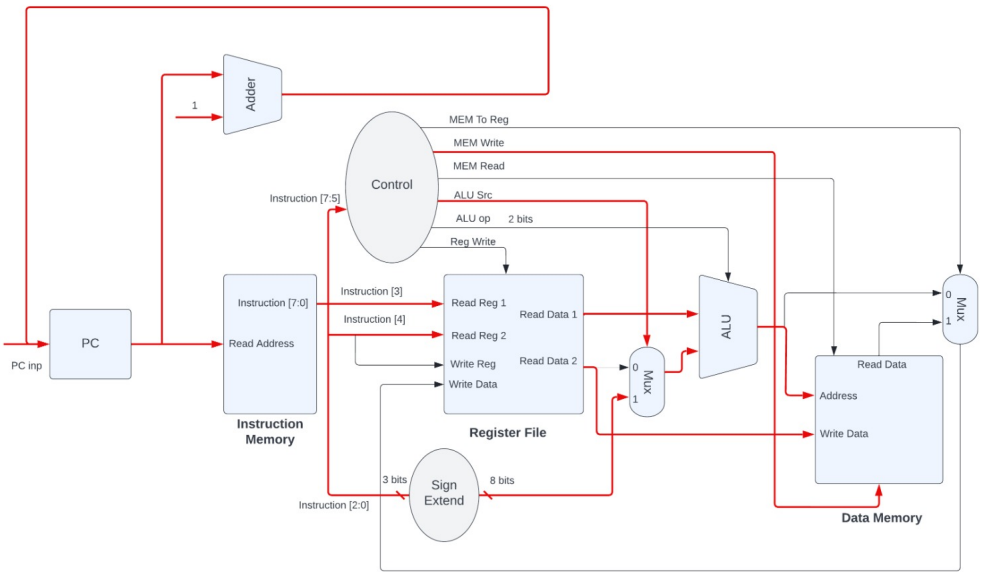

Figure 10: Store instruction Datapath

2.3.5 Schematic Diagram

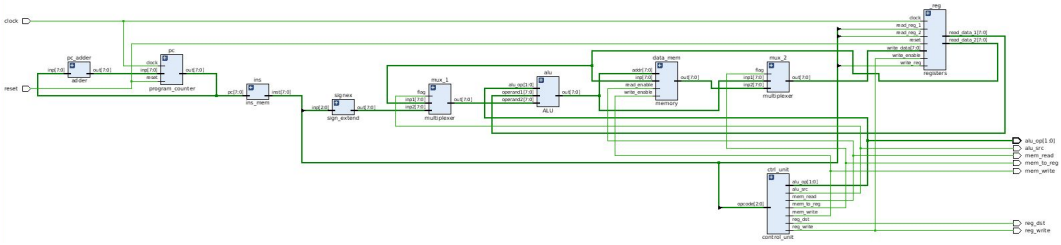

Figure 11: Schematic Diagram

2.4 Simulation results of the components

We implemented the following set of instructions:

$$add\ \$s1,\ \$s1,\ \$s1$$

$$lw\ \$s1,\ 1(\$s1)$$

$$sw\ \$s0,\ 0(\$s1)$$

The initial value of $\$s0$ is 5 and $\$s1$ is 1. The simulation output for these set of instruction is:

Figure 12: Simulation output

3 Work Distribution

3.1 Design

All four of us gave equal contributions in the designing components, integrating them using wires and multiplexers.

3.2 Implementation

All four of us made equal contributions on the implementation part on Verilog.

3.3 Report

All four of us made equal contributions on the report part following the guidelines as mentioned.

References

[1] "IEEE Standard Verilog Hardware Description Language". In: *IEEE Std 1364-2001* (2001), pp. 1–792. DOI: 10.1109/IEEESTD.2001.93352.

[2] *Hardware modeling using Verilog by Prof. Indranil Sengupta, NPTEL*. https://nptel.ac.in/courses/106105165.

[3] *Verilog Quick Reference guide*. https://cse.buffalo.edu/~rsridhar/cse490-590/lec/verilog_manual.pdf. [Course Website].