Computer Architecture

Project I – Milestone II Report

RISC-V FPGA Implementation and Testing

Done by:

Seif Elshabshiri 900202310

Roaa Bahaa 900203054

Abstract

This report mainly illustrates the second milestone of our project that aims at implementing a RISC-V processor and testing it on the FPGA. This milestone implements the DataPath as a single cycle data path without pipelining and supporting 40 instructions of the Risc-V instruction set. Throughout the paper, the design specifications, assumptions taken, and some test cases to demonstrate the full functionality of the program will be provided.

Table of content

1.Program Design

1.1. processor (Top Module):

This module is responsible for relating all the modules of the Risc-V data path together in a single cycle non-pipelined manner following the structure of the following figure and using the control signals illustrated in the following sub-section:

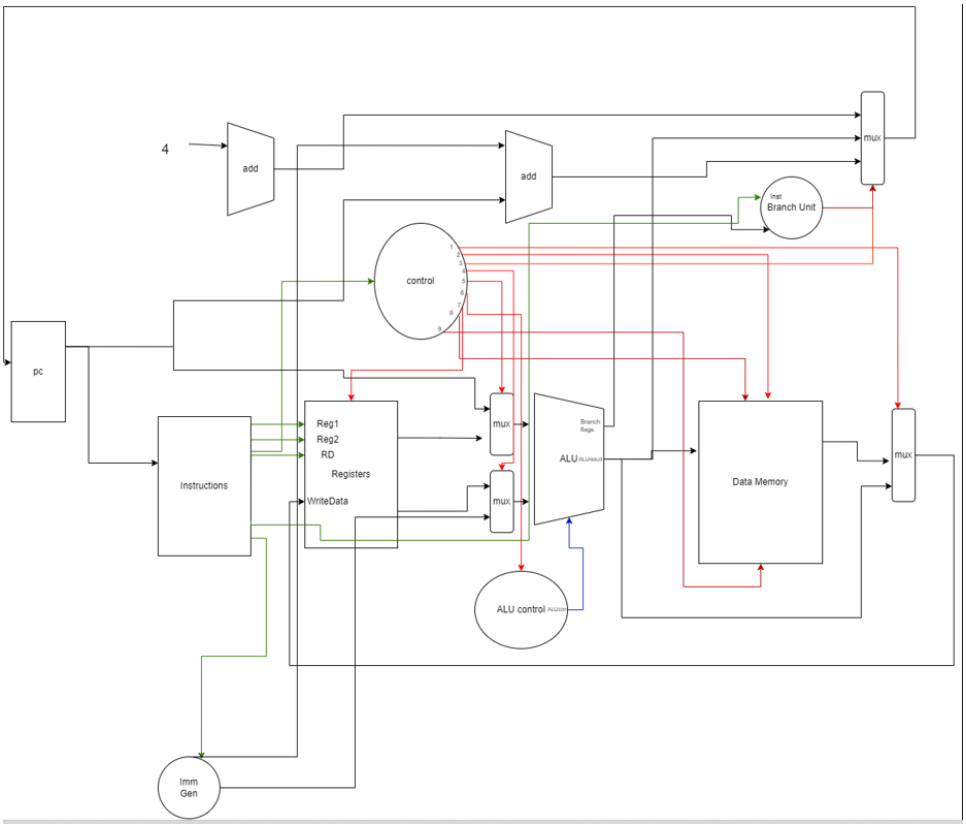

Figure 1

1.2. Control signals:

MemtoReg: a two-bit control signal that chooses between the ALU output or the data memory output or pc + 4 to write back to the register file.

MemRead: a one-bit control signal responsible for determining whether the Memory is going to be read or not.

Jalr: Unconditional jump signal that is concatenated with the branch output in otder to multiplex between pc + 4, pc + imm, or rs1 + imm

ALUsrc2: a one-bit control signal that determines what to input to the ALU, either rs1 or imm

ALUsrc1: a one-bit control signal that determines what to input to the ALU, either rs2 or pc. This multiplexer only chooses pc when the instruction is AUIPC

ALUOP: a two-bit control signal that determines the operation that is going to be executed by the ALU.

RegWrite: a one-bit control signal responsible for determining whether or not the register should be written with write back data.

MemWrite: a one-bit control signal responsible for determining whether or not the memory should be updated.

Readwrite: a four-bit control signal responsible for differentiating the load and store instructions, which areas: lw, lbu, lb, lh, lhu, sw, sb, sh.

2. Program Testing

2.1. Test case one

This test case is to test some of the instructions using the data memory, such as: lb, lw and sw.

lb x4, 5(x0)

lw x6, 0(x0)

sw x4, 8(x0)

lw x8, 8(x0)

With the memory initialized as follows:

{mem[3],mem[2],mem[1],mem[0]}=32'd17;

mem[5] = 8'd36;

And the output of the register file shown in the following figure

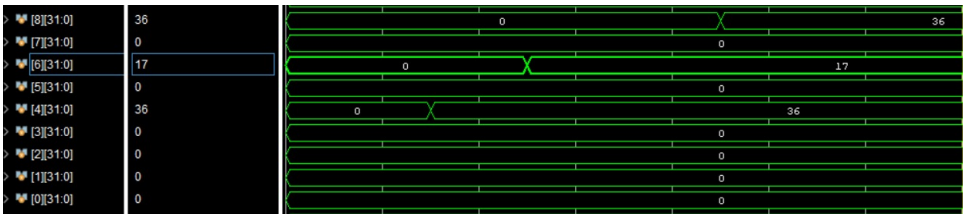

Figure 2

2.1. Test case two

This test case is to test some of the arithmetic instructions, such as: addi and add and some others of the brancing instructions, such as: blt. With the same values of memory given in the previous test case.

addi x1, x1, 1

addi x4, x4, 4

loop: lb x5, 5(x0)

add x5, x5, x1

addi x1, x1, 1

blt x1, x4, loop

add x6, x6, x4

The following figure shows the values of pc across time and how the program branched two times.

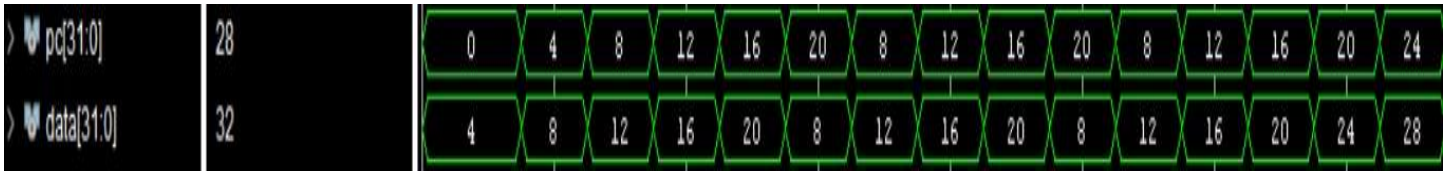

Figure 3

The following figure shows the final results of the register file after executing the program.

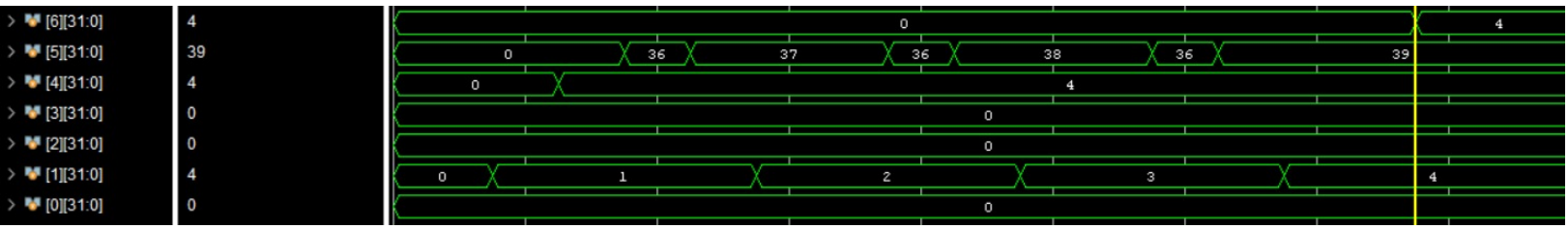

Figure 4

2.1. Test case three

This test case is to test some other instructions, such as: lui, auipc ,jal and beq with the same data memory initialization as the previous one.

lui x4, 586

auipc x5, 423

addi x6, x6, 7

jal x2, function

addi x6, x6, 1

beq x0, x0, exit

function: addi x7, x6, 20

jalr x0, 0(x2)

exit: add x0, x0, x0

The following figure shows the values of pc and how it changed over time with the jalr , jal and beq instructions.

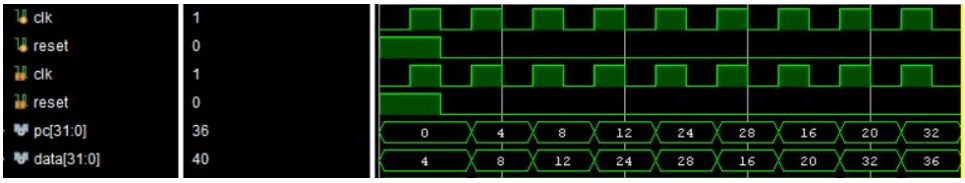

Figure 5

The following figure shows the final results of the register file after executing the program.

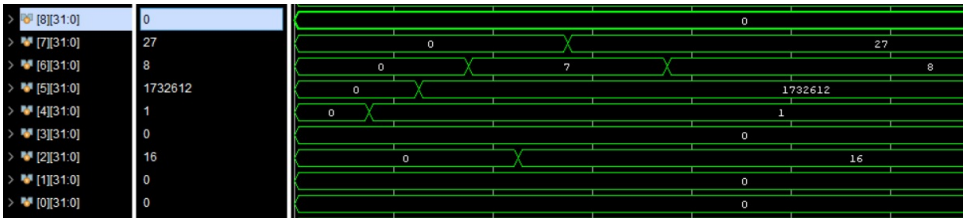

Figure 6

3. Assumptions

The following assumptions were made while designing the data path:

- There are two separate memories; one for storing instructions and another for storing the data.
- The data memory is byte addressable.

4. how to use the program

The user should assign the different values of the data memory in the datamem module and insert the instructions opcode in the instMem module as shown in the following figures:

```
{mem[3],mem[2],mem[1],mem[0]}=8'd17;
mem[4]= 2'd55;
```

Figure 7

```
mem[0]=32'h00308193; //l1: addi x3, x1, 3
mem[1]=32'h00320293; //addi x5, x4, -3
mem[2]=32'hfe519ce3; //bne x5, x3, l1
mem[3]=32'h00020497; //aupc x9,32
mem[4]=32'h008000ef; //jal x1, exit
mem[5]=32'h005180b3; //add x1, x3, x5
mem[6]=32'h00130313; //addi x6, x6,1
mem[7]=32'h00008067; //jalr x0, 0(x1)
```

Figure 8