RISC-V Processor

Carlos Ornelas
CECS 440 - Computer Architecture
Supervisor/Professor: Dr. Seyyedhosseini
California State University Long Beach College of Engineering

Table of Contents

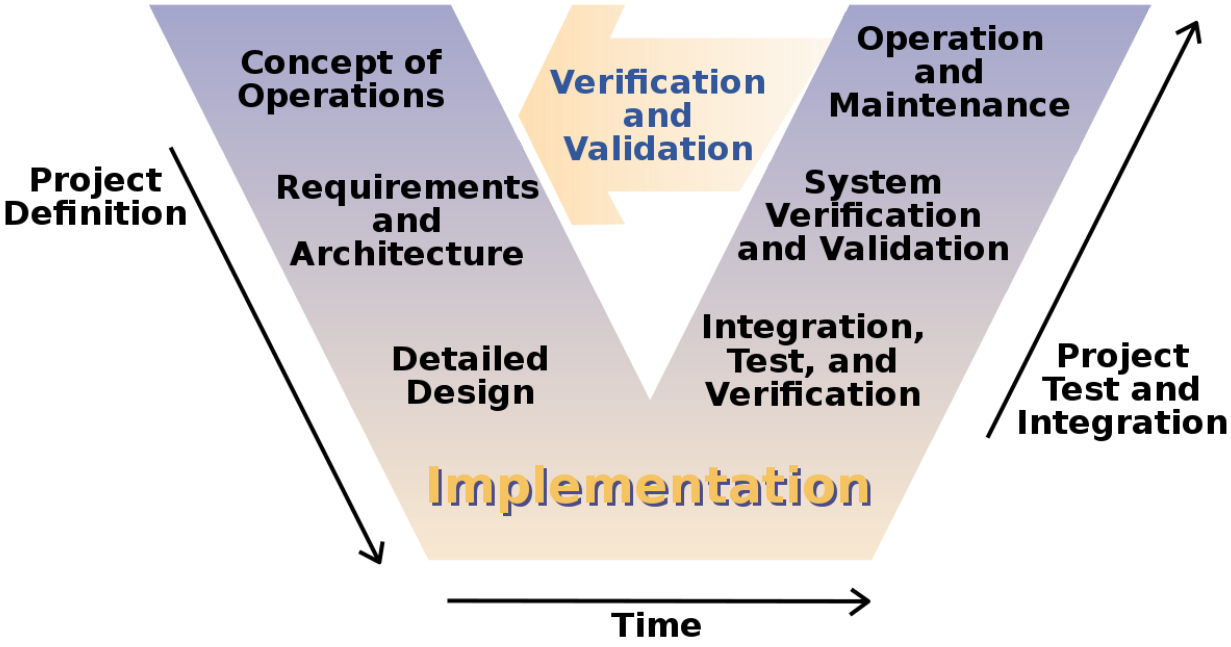
1. Overview

This report provides a technical description and guide in creating a Single Cycle 32-Bit RISC-V processor using Verilog in the Vivado Suite. The processor is made up of three modules: the datapath, the ALU controller, and the controller. The datapath is the main brain of the system and contains important building blocks like the 32 x 32 Register and the ALU. Next, is the controller which basically determines the source operand, as well as determining if data will be read/written to the memory and registers. Lastly, The ALU controller selects what operation will be performed by the ALU in the datapath.

2. Development

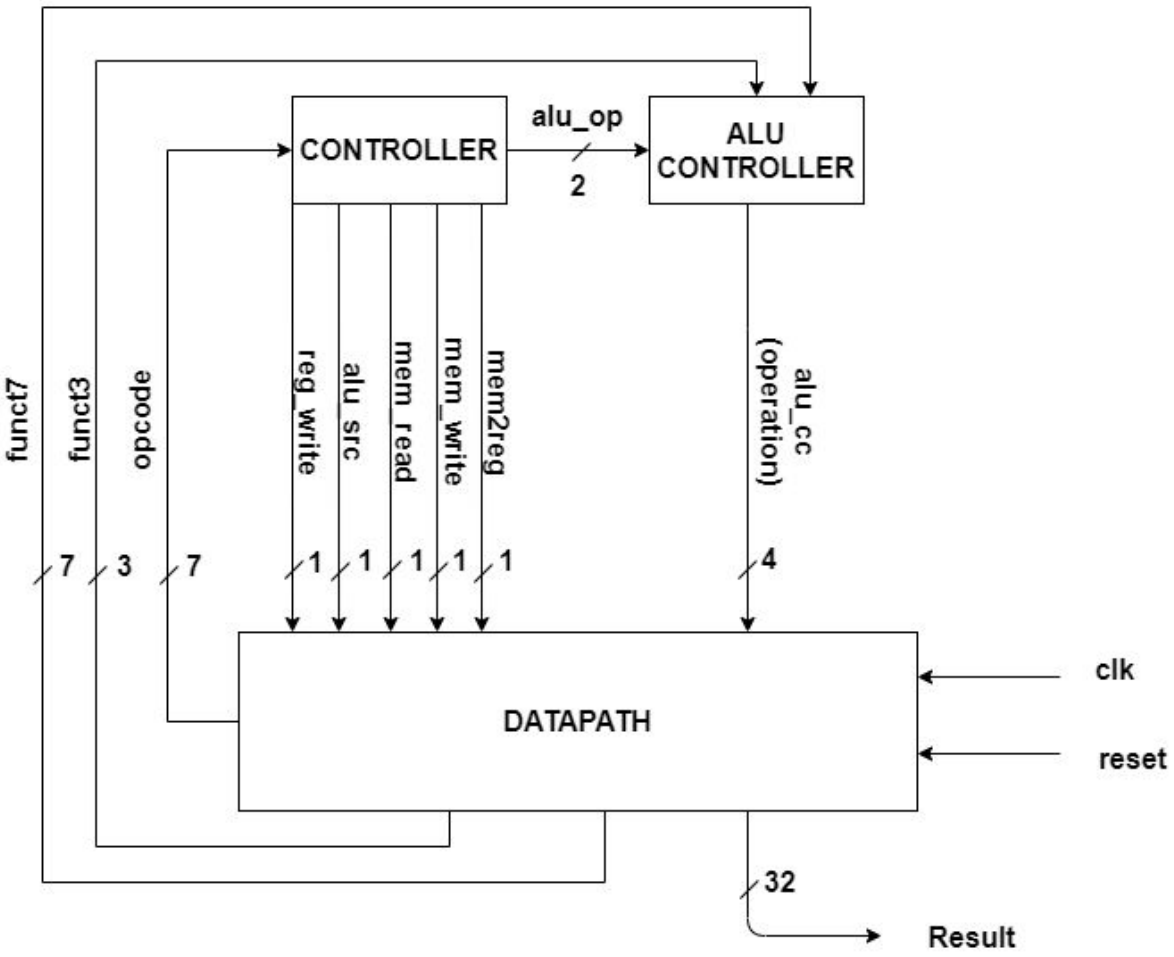
For this project I used Xilinx's Vivado suite. The Vivado suite is the successor to the Xilinx ISE design suite. This was developed alongside the class using lecture notes and some guidance from professor Dr. Seyyedhosseini. Adding onto the project I will be creating a separate github folder that will contain the function of the processor, but tested on a Nexys 4 FPGA.

The Development process loosely followed the guidelines of the commonly known V diagram. The project definition portion of the project was given to us by Dr. Seyyedhosseini, while the build and test portions of the V diagram was done independently. The V-model project flow is below.



3. Processor Design

3.1. Schematic



3.2. Detailed Description

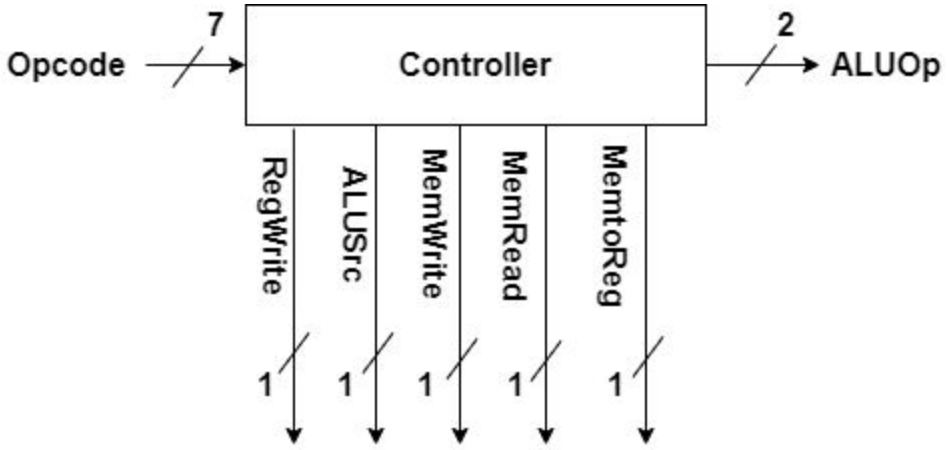
Top module for the design. The processor holds the three main modules and connects them to each other. The modules are the following: ALUController, Controller, and Datapath. As mentioned previously, this processor is based around the RISC-V architecture and is a single cycle design.

3.3. I/O Signal Chart

Signal	I/O	Size	Description
clk	**Input**	**1**	System wide clock. Synchronous.
reset	**Input**	**1**	System wide reset. Sets registers and other blocks back to “default” state.
Result	**Output**	**32**	Final result of the processor. Usually a result from the ALU, but could also be values from registers and memory.

4. Controller Design

4.1. Schematic



4.2. Detailed Description

The control unit is responsible for determining what operation that the ALU controller will further specify. Furthermore, it tells the datapath whether it will be performing a simple arithmetic operation, storing/loading data, or writing/reading data. The way the Controller determines this is by taking the opcode from the datapath and decoding it based around the truth table below. The truth table was provided by the professor and was created in conjunction with the RISC-V datasheet.

Table 1 : Control Signals.

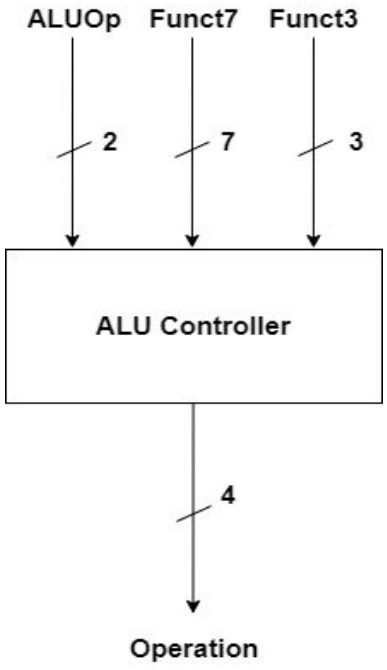
		Opcode			
		0110011	0010011	0000011	0100011
		AND, OR, ADD, SUB, SLT, NOR	ANDI, ORI, ADDI, SLTI, NORI	LW	SW
Control Signals	MemtoReg	0	0	1	0
	MemWrite	0	0	0	1
	MemRead	0	0	1	0
	ALUSrc	0	1	1	1
	Regwrite	1	1	1	0
	ALUOp	10	00	01	01

4.3. I/O Signal Chart

Signal	I/O	Size	Description
Opcode	**Input**	**7**	The opcode is the first 7 bits (LSB) of the 32 bit instruction taken from the instruction memory within datapath. The opcode location is determined by the RISC-V processor manual.
RegWrite	**Output**	**1**	This signal determines whether the register file in the datapath gets written to or not.
ALUSrc	**Output**	**1**	This signal determines whether one of the ALU's operands comes from the 32 x 32 register file or the immediate generator. Generated by decoding the opcode.
MemWrite	**Output**	**1**	Write enable signal for writing to memory in the datapath.
MemRead	**Output**	**1**	Read-enable signal for reading from memory in the datapath.
MemtoReg	**Output**	**1**	Signal that goes to a multiplexor which determines whether the result of the ALU or the output from memory goes to the register.
ALUOp	**Output**	**2**	Gets sent to the ALU and determines what operation the ALU will perform.

5. ALU Controller Design

5.1. Schematic



5.2. Detailed Description

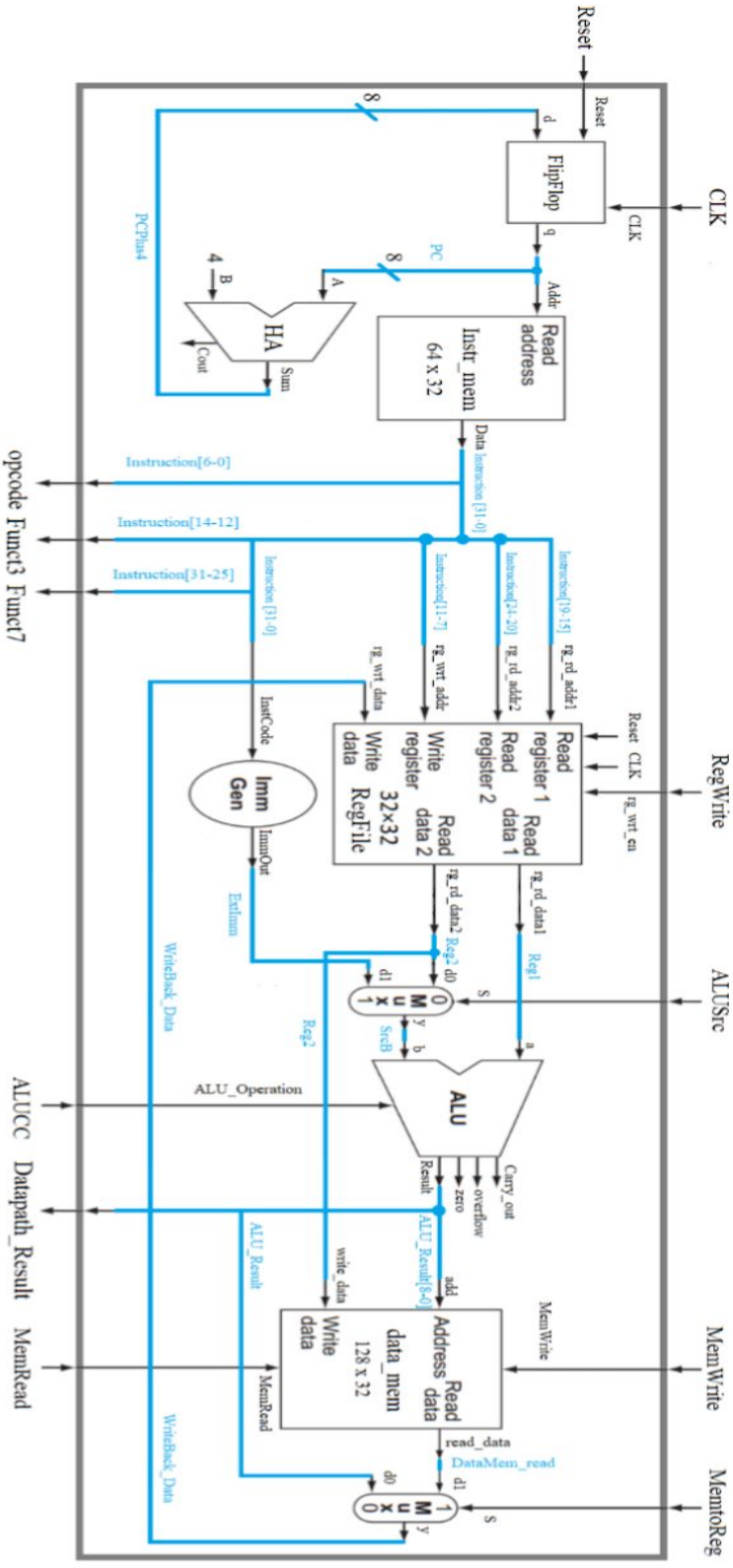
The ALU Controller is a second controller used to access more specific ALU operations if needed. Along with the ALUOp input, the Funct7 and Funct3 inputs can access different operations along the tree of the general ALUOp given. These specified operations are defined in the RISC-V datasheet.

5.3. I/O Signal Chart

Signal	I/O	Size	Description
ALUOp	**Input**	**2**	The Alu Operation to be performed. Received from the Controller.
Funct7	**Input**	**7**	Helps define the function further if needed. Relegated to R-type instructions.
Funct3	**Input**	**3**	3 bit function selector. Helps define the operation further.
Operation	**Output**	**4**	The fully defined operation to be performed

6. Datapath Design

6.1. Schematic



6.2. Detailed Description

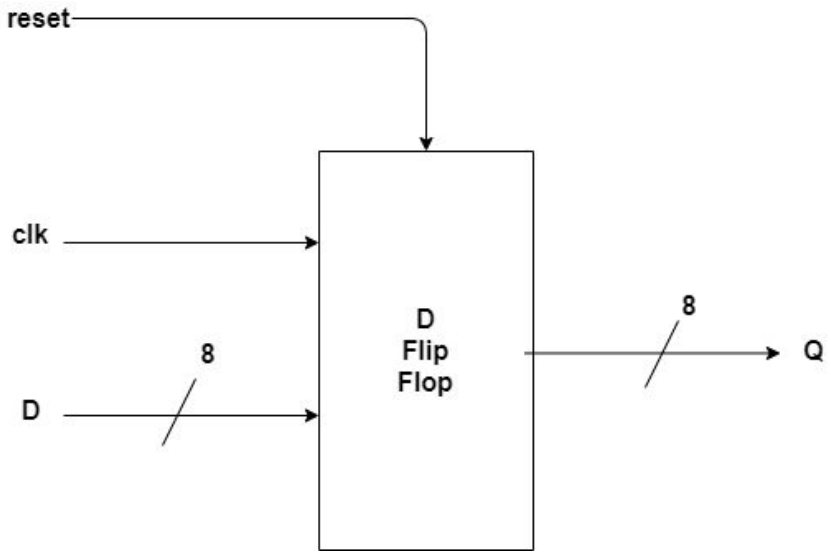
As the name suggests, the datapath is where operations are fetched and executed. The main brain of the processor, where the ALU, register file, instruction memory, and data memory lives.

6.3. I/O Signal Chart

Signal	I/O	Size	Description
clk	**Input**	**1**	System wide clock
reset	**Input**	**1**	System wide reset sets all inputs to default state
reg_write	**Input**	**1**	Input to enable the write-to-register signal in the Register file.
mem2reg	**Input**	**1**	Mux Selector for the second multiplexor
alu_src	**Input**	**1**	Mux selector for the first multiplexor
mem_write	**Input**	**1**	Enables the write to memory signal for the data memory module
mem_read	**Input**	**1**	Enables the read from memory signal for the data memory module
alu_cc	**Input**	**4**	Alu Operation selection for the ALU module
opcode	**Output**	**7**	Operation code. Is sent to the controller to be decoded and then sent to the ALU controller.
funct7	**Output**	**7**	Is sent to the ALU controller to be decoded. Specifies the ALU operation to be performed.
funct3	**Output**	**3**	Is sent to the ALU controller to be decoded. Specifies the operation further.
alu_result	**Output**	**32**	Result from the ALU.

6.3.1. Flip Flop

6.3.1.1. Schematic



6.3.1.2. Detailed Description

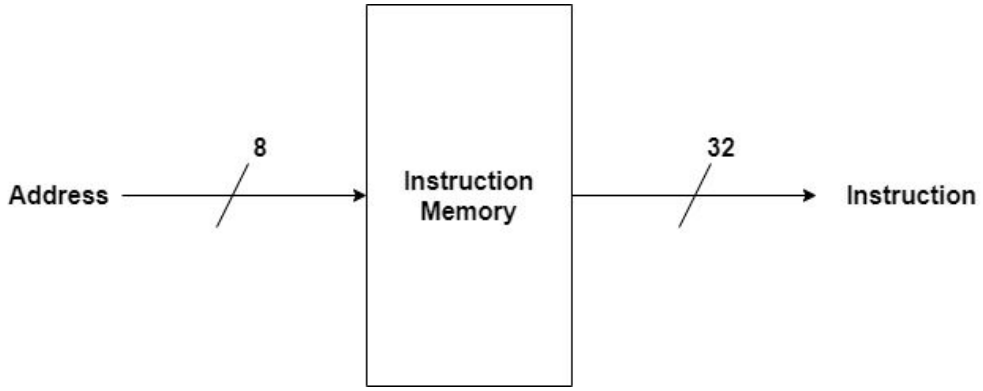
This is a Synchronous D Flip Flop. Reset will only reset values to zero upon at every positive edge clock signal. If reset is inactive, then at every positive edge of the clock q gets all 8 bits of d. The flip flop is used to hold each new memory address at each positive edge of the clock. To move to a new memory location the current memory location must be added by 4, as each memory location is 4 bytes wide.

6.3.1.3. I/O Signal Chart

Signal	I/O	Size	Description
clk	**Input**	**1**	Positive Edge Clock
reset	**Input**	**1**	Synchronous Reset
d	**Input**	**8**	Holds the new memory address from the half adder.
q	**Output**	**8**	Receives the input if reset is inactive and sends to the half adder to go to the next memory location

6.3.2. Instruction Memory

6.3.2.1. Schematic



6.3.2.2. Detailed Description

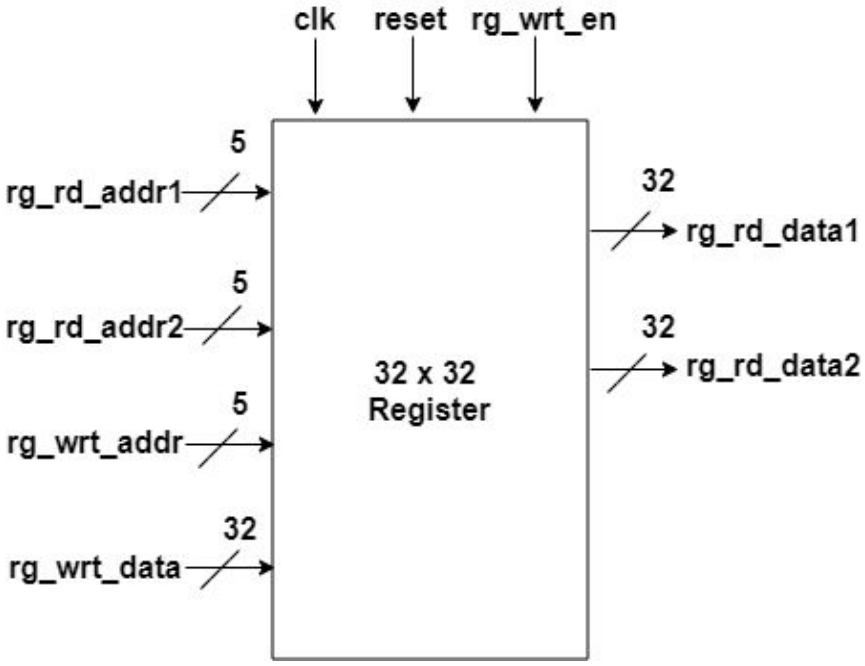
Instruction Memory works like a pointer, as the purpose for the module is to hold all the necessary instructions in memory. Within the instruction memory module, we have simulated a two-dimensional 32 x 64 memory module.

6.3.2.3. I/O Signal Chart

Signal	I/O	Size	Description
Address	**Input**	**8**	The location of the instruction in memory. We get the address from the flip flop.
Instruction	**Output**	**32**	The 32 bit instruction being pointed at.

6.3.3. 32 x 32 Register File

6.3.3.1. Schematic



6.3.3.2. Detailed Description

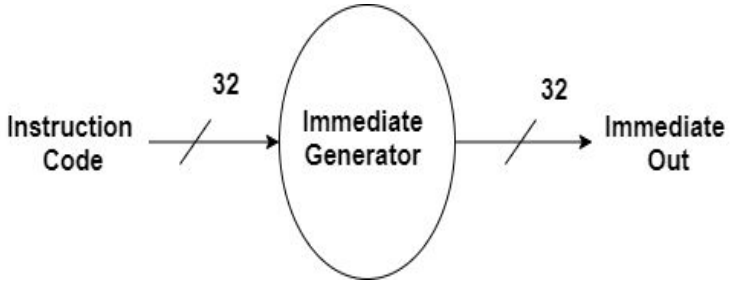
An asynchronous 32x32 register module. This register block can be written and read from. When write enable is low then the register block cannot be written to, however when enabled then the data from rg_wrt_data will be written to the location defined in rg_wrt_adddr. The two read address inputs each correspond to the rg_rd_data outputs after data had been read from the address.

6.3.3.3. I/O Signal Chart

Signal	I/O	Size	Description
clk	**Input**	**1**	Clock signal
reset	**Input**	**1**	Asynchronous reset
rg_wrt_en	**Input**	**1**	Active high enable
rg_rd_addr1	**Input**	**5**	Address to read data from to be sent to rg_rd_data1
rg_rd_addr2	**Input**	**5**	Address to read data from to be sent to rg_rd_data2
rg_wrt_addr	**Input**	**5**	Address to be written to.
rg_wrt_data	**Input**	**32**	Data to be written to specified address
rg_rd_data1	**Output**	**32**	Data read from the register. Will be sent to the “A” operand for the ALU
rg_rd_data2	**Output**	**32**	Data read from the register. Will be sent to the “B” operand for the ALU.

6.3.4. Immediate Value Generator

6.3.4.1. Schematic



6.3.4.2. Detailed Description

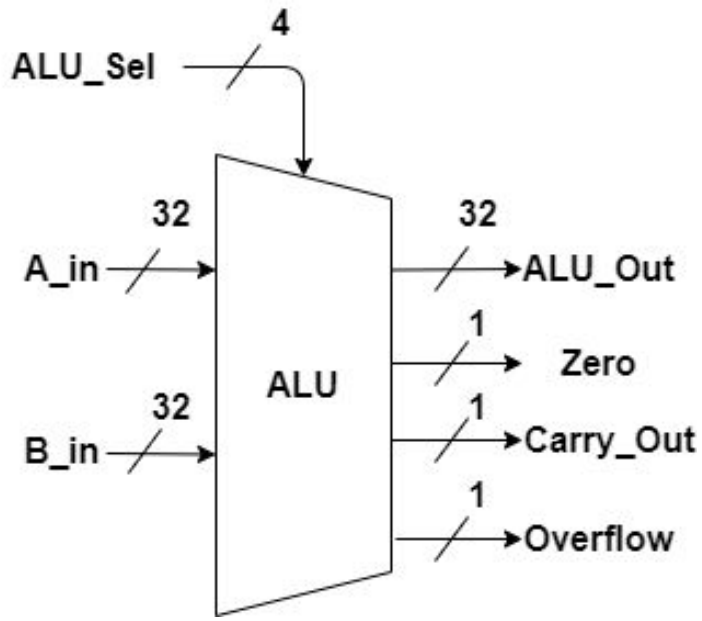
Decodes the instruction code and determines if the immediate value needs to be sign-extended to 32 bits in order to properly perform the operation without error.

6.3.4.3. I/O Signal Chart

Signal	I/O	Size	Description
Instruction Code	**Input**	**32**	The full 32-bit instruction code from instruction memory
Immediate Out	**Output**	**32**	The sign-extended value

6.3.5. ALU (Arithmetic Logic Unit)

6.3.5.1. Schematic



6.3.5.2. Detailed Description

This is a 32-Bit ALU module abstraction, not built for any particular FPGA. An ALU is an arithmetic logic unit and is often called the “brain” of the computer. An ALU performs logical and arithmetic operations depending on what the designer included, as well as what operation is being selected.

This ALU has two 32-bit inputs and a total of 7 operations **(detailed below)** using a 4-bit opcode selection. Furthermore, This ALU also includes 3 forms Status checks: Zero, Carry-Out, and Overflow. The first operand’s data, A_in, comes from the rg_rd_addr1 signal. On the other hand, the second operand, B_in, gets its data from either the rg_rd_addr2 signal or the Immediate Out signal. This is selected using a 2 to 1 Mux **(Shown below)** where the ALU_Src (select signal) comes from the Controller. Active high “1” to select the Immediate Out signal and vice versa for the rg_rd_addr2 signal.

6.3.5.3. I/O Signal Chart / ALU Operation Codes

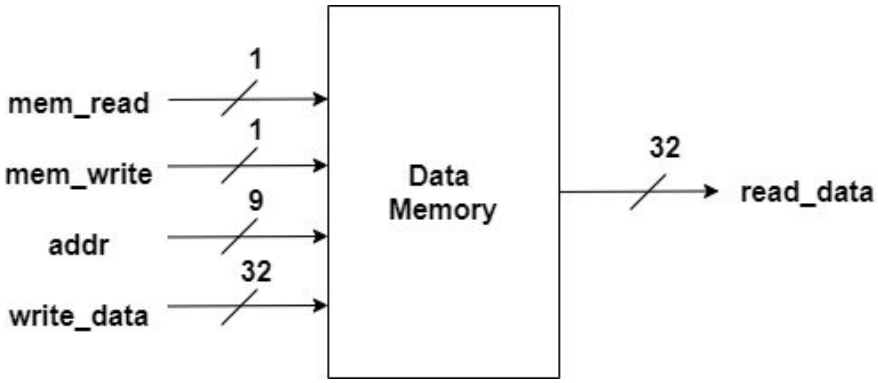
Signal	I/O	Size	Description
A_in	**Input**	**32**	1st operand. Comes from rg_rd_addr1.
B_in	**Input**	**32**	2nd operand. Comes from either rg_rd_addr2 or the Immediate generator.
ALU_Sel	**Input**	**4**	Operation Selection. 7 operations total. 4-bit length size.
ALU_Out	**Output**	**32**	32-bit result of the operation on the two operands
Carry_Out	**Output**	**1**	1-bit status flag. If Carry_Out is a 1 it means that the result had a carry out on the MSB
Overflow	**Output**	**1**	1-bit status flag. If Overflow is a 1 it means that the result of the operation could not fit in the 32-bit register
Zero	**Output**	**1**	1-bit status flag. If Zero is “on” then all 32- bits of ALU_Out are zero.

ALU OPERATION CODES

ALU Control Line	Function	Description
0000	**AND**	Boolean And operation
0001	**OR**	Boolean or operation
0010	**ADD**	Adds both operands
0110	**SUBTRACT**	Subtract both operands
0111	**SET LESS THAN**	If A_in is less than B_in then ALU_Out is a 1, else 0
1100	**NOR**	Boolean NOR operation
1111	**EQUAL COMPARE**	If A_in and B_in are equal, then ALU_Out is a 1, else 0

6.3.6. Data Memory

6.3.6.1. Schematic



6.3.6.2. Detailed Description

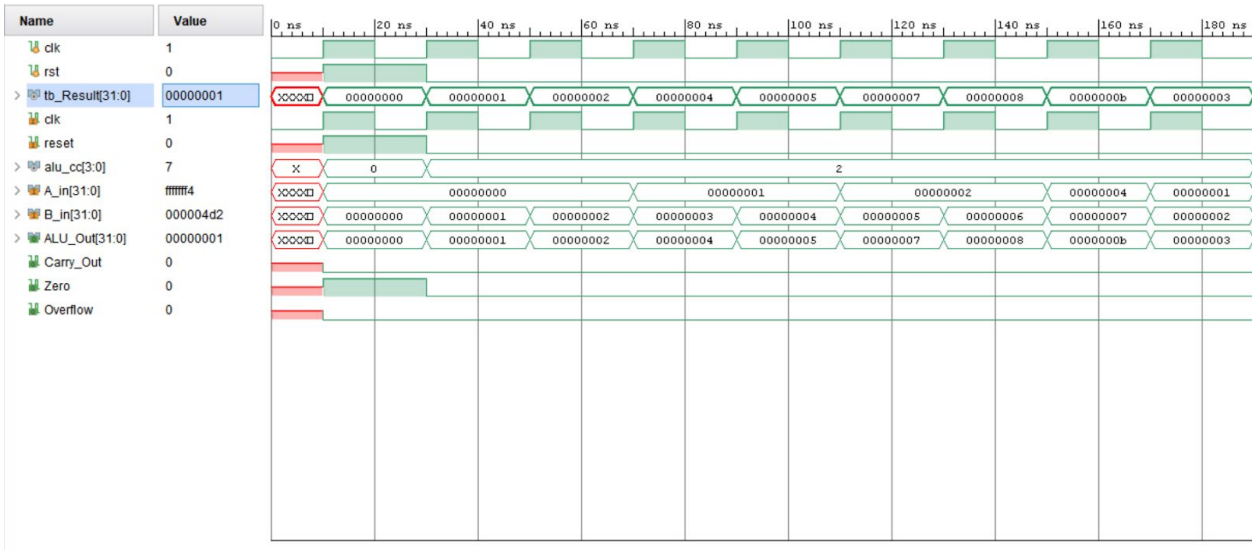
Holds the result from the ALU. The memory size is 128x32 and is byte accessible.

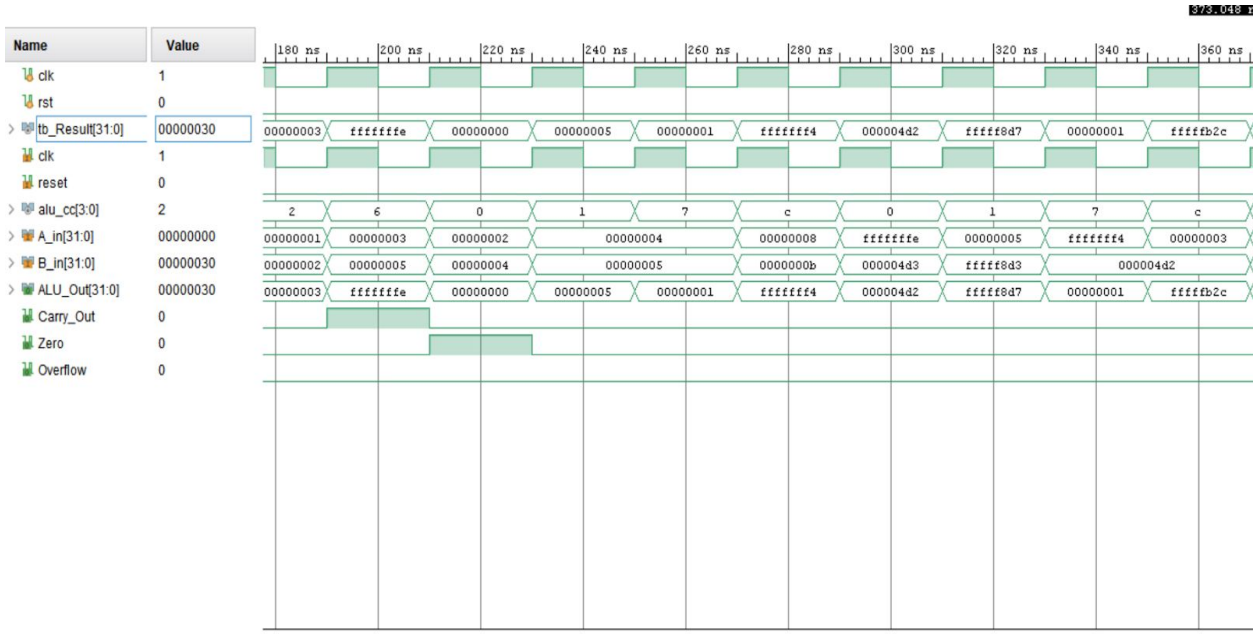
6.3.6.3. I/O Signal Chart

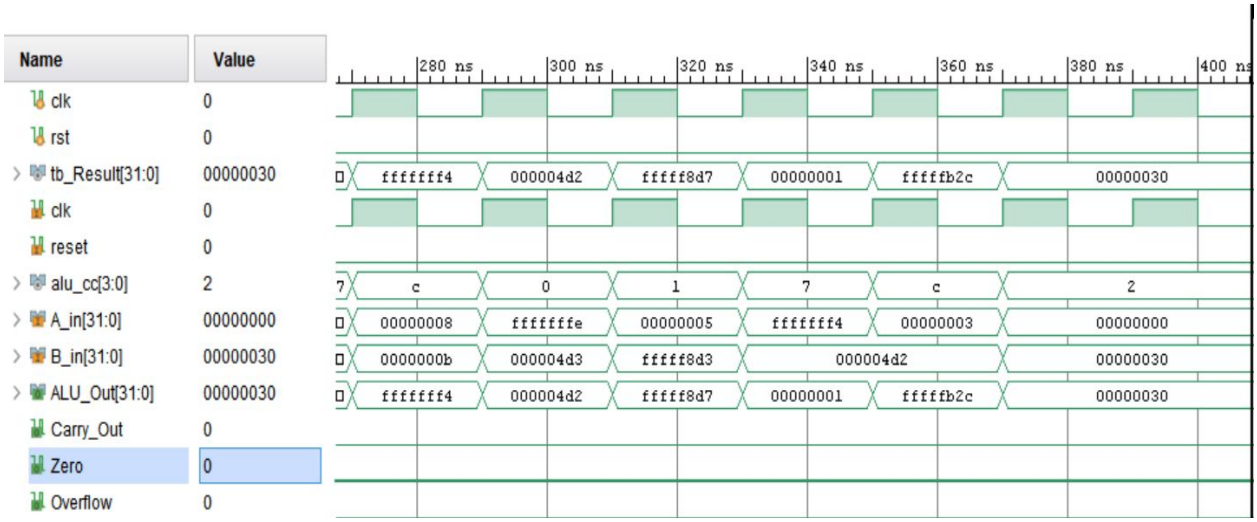
Signal	I/O	Size	Description
mem_read	**Input**	**1**	Active high signal to initiate the read process.
mem_write	**Input**	**1**	Active high signal to initiate the write process
addr	**Input**	**9**	Memory address to read or write from
write_data	**Input**	**32**	Data to be written to memory
read_data	**Output**	**32**	Data read from memory

7. Simulation/Verification

The waveforms test each instruction of the processor at each clock cycle and show the status flags working dependent on what the output is.




Name
Value
clk 1
rst 0
tb_Result[31:0] 00000030
clk 1
reset 0
alu_cc[3:0] 2
A_in[31:0] 00000000
B_in[31:0] 00000030
ALU_Out[31:0] 00000030
Carry_Out 0
Zero 0
Overflow 0
180 ns
200 ns
220 ns
240 ns
260 ns
280 ns
300 ns
320 ns
340 ns
360 ns
00000003 fffffffe 00000000 00000005 00000001 fffffff4 000004d2 fffff8d7 00000001 fffffb2c
2 6 0 1 7 c 0 1 7 c
00000001 00000003 00000002 00000004 00000008 fffffffe 00000005 fffffff4 00000003
00000002 00000005 00000004 00000005 0000000b 000004d3 fffff8d3 000004d2
00000003 fffffffe 00000000 00000005 00000001 fffffff4 000004d2 fffff8d7 00000001 fffffb2c


Name
Value
clk 0
rst 0
tb_Result[31:0] 00000030
clk 0
reset 0
alu_cc[3:0] 2
A_in[31:0] 00000000
B_in[31:0] 00000030
ALU_Out[31:0] 00000030
Carry_Out 0
Zero 0
Overflow 0
280 ns
300 ns
320 ns
340 ns
360 ns
380 ns
400 ns
fffffff4 000004d2 fffff8d7 00000001 fffffb2c 00000030
7 c 0 1 7 c 2
00000008 fffffffe 00000005 fffffff4 00000003 00000000
0000000b 000004d3 fffff8d3 000004d2 00000030
fffffff4 000004d2 fffff8d7 00000001 fffffb2c 00000030

8. Verilog

Please refer to the following Git repository for the Verilog.

Please refer to the README in the repository.

https://github.com/skecherboy/RISC-V_Processor_32Bits

9. References

https://riscv.org/technical/specifications/

Dr. Seyyedhosseini's lectures and code examples.