CSE490 Project 1

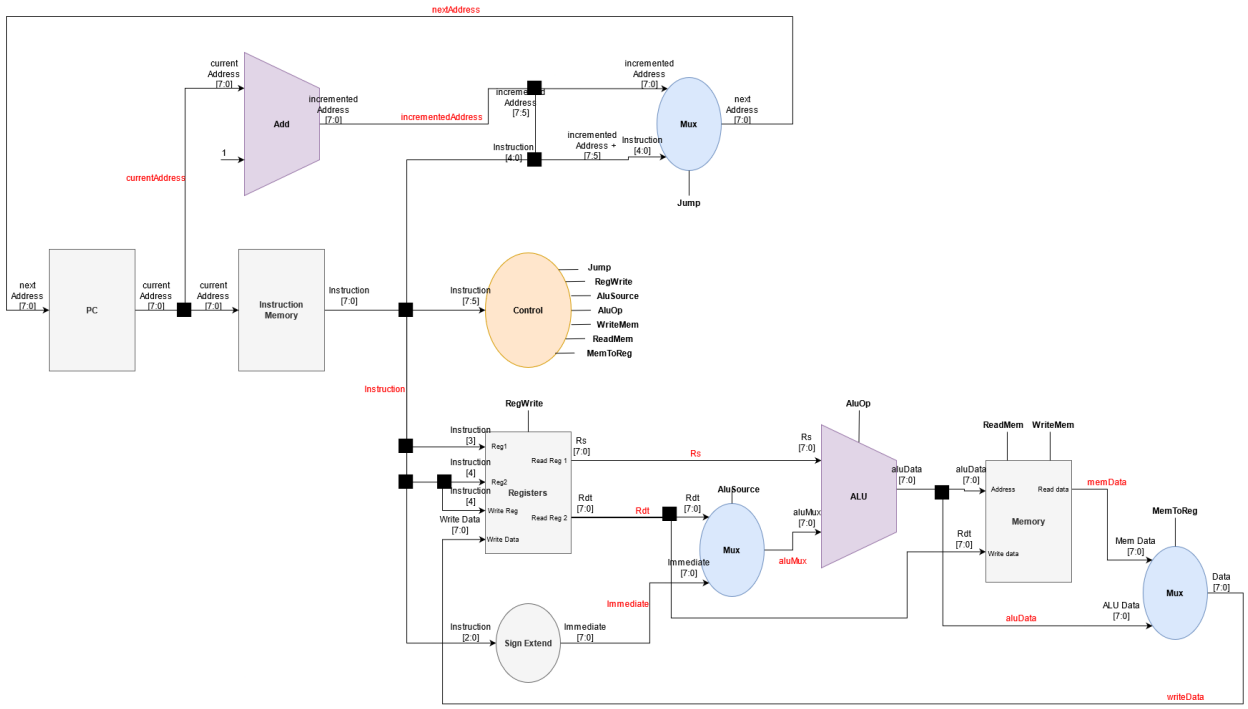
8-Bit Processor Design in Verilog

Group Members:

Tijmen Van Der Beek (50240699)

Siquan Wang (50191837)

Schematic


nextAddress
current Address [7:0]
Add
incremented Address [7:0]
incrementedAddress
incremented Address [7:5]
incremented Address [7:0]
Mux
next Address [7:0]
1
currentAddress
Instruction [4:0]
incremented Address + [7:5]
Instruction [4:0]
Jump
next Address [7:0]
PC
current Address [7:0]
current Address [7:0]
Instruction Memory
Instruction [7:0]
Instruction [7:5]
Control
Jump
RegWrite
AluSource
AluOp
WriteMem
ReadMem
MemToReg
Instruction
RegWrite
AluOp
Instruction [3]
Instruction [4]
Instruction [4]
Write Data [7:0]
Reg1
Reg2
Write Reg
Write Data
Read Reg 1
Registers
Read Reg 2
Rs [7:0]
Rs
Rs [7:0]
ALU
aluData [7:0]
ReadMem
WriteMem
aluData [7:0]
Address
Read data
Memory
Write data
memData
Rdt [7:0]
Rdt
Rdt [7:0]
AluSource
Mux
aluMux [7:0]
aluMux
Rdt [7:0]
Immediate [7:0]
Immediate
MemToReg
Mem Data [7:0]
Mux
Data [7:0]
ALU Data [7:0]
aluData
Instruction [2:0]
Sign Extend
Immediate [7:0]
writeData

Module Description

PC:

Program Counter module that initializes the instruction address and output to the next module.

PC_Adder:

Incrementing the instruction address by 1 so the program pointer goes to the next instruction after each instruction is done. If the instruction is jump, increment the number by whatever is in the last 5 bits of the J type instruction.

Mux:

The mux module outputs one of the 8bit inputs using the select bit. We are using the same mux model for all three muxes the only difference are the inputs.

Instruction_memory:

This module takes in the instruction address and access the instruction at specific address. The memory size of this module is 256 since the address is only 8 bits(2^8=256).This module outputs the instruction data.

Sign_extend:

Extended the 3-bit to 8 bits by copying the MSB in 3 bit and filling the bit 7 to 3 in 8 bit with it.

ALU:

This module performs computations according to the opcode input. This modul takes in the data(s) from register memory and opcode from instruction and computing them. The computation depends on what type of instruction it is, it can perform addition, immediate addition and subtraction.

DataMem:

This module enables an array of size 256 as memory that stores data. Depending on the instruction type, this module can either read or write data from the array.

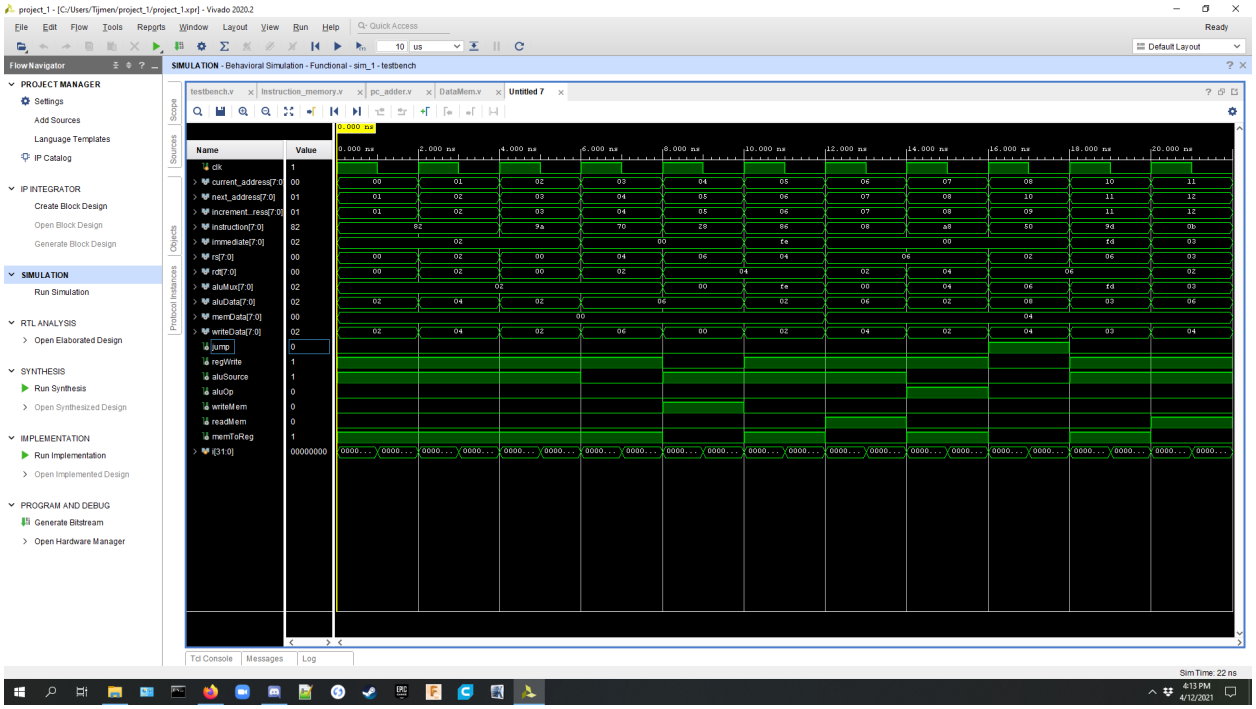
Control:

This module controls the datapath of an instruction depending on the type of instruction. It outputs 1 bit signals of jump, regWrite, ALUsource, ALUop, WriteMem, ReadMem, MemtoReg that control other modules to output the correct result of each instruction.

Reg_file:

This module accesses the data in registers in a register array size of 2 (t0 and t1). This module also outputs 2 8-bit numbers rs and rtd. If write enabled, this module stores data into the designated register according to the instruction.

Simulation Result



*The names on the left hand side correspond with the names on the schematic.

These are the instructions in memory:
memory[0] = 8'b10000010; //addi r0, r0, 2
memory[1] = 8'b10000010; //addi r0, r0, 2
memory[2] = 8'b10011010; //addi r1, r1, 2
memory[3] = 8'b01110000; //add r1, r1, r0
memory[4] = 8'b00101000; //sw r0, 0(r1)
memory[5] = 8'b10000110; //addi r0, r0, -2
memory[6] = 8'b00001000; //lw r0, 0(r1)
memory[7] = 8'b10101000; //sub r0, r0, r1
memory[8] = 8'b01010000; //jmp 0x10
memory[16] = 8'b10011101;//addi r1, r1, -3
memory[17] = 8'b00001011;//lw r0, 3(r1)

Table of values along datapath

address	instruction	rs	rdt	immediate	writeData	nextAddress
0x00	addi r0, r0, 2	0	0	2	2	0x01
0x01	addi r0, r0, 2	2	2	2	4	0x02
0x02	addi r1, r1, 2	0	0	2	2	0x03
0x03	add r1, r1, r0	4	2	n/a	6	0x04
0x04	sw r0, 0(r1)	6	4	0	n/a	0x05
0x05	addi r0, r0, -2	4	4	-2	2	0x06
0x06	lw r0, 0(r1)	6	2	0	4	0x07
0x07	sub r0, r0, r1	6	4	n/a	2	0x08
0x08	jmp 0x10	n/a	n/a	n/a	n/a	0x10
0x10	addi r1, r1, -3	6	6	-3	3	0x11
0x11	lw r0, 3(r1)	3	2	3	4	0x12

Work Distribution

PC: Tijmen
PC Adder: Tijmen
Mux: Tijmen
Instruction Memory:Tijmen
Control: Tijmen
Registers: Siquan
Sign Extend: Siquan
ALU: Siquan
Memory: Siquan

Reference

https://stackoverflow.com/questions/36388931/verilog-program-counter-with-branching
https://stackoverflow.com/questions/29628469/how-to-store-input-into-reg-from-wire-in-verilog
https://www.chipverify.com/verilog/verilog-scalar-vector
https://www.chipverify.com/verilog/verilog-assign-statement

https://www.youtube.com/watch?v=PJGvZSlsLKs