Introduction to Digital Logic Design Lab

EECS 31L

Lab 5 : Single Cycle Processor

Alex(a) Danielian

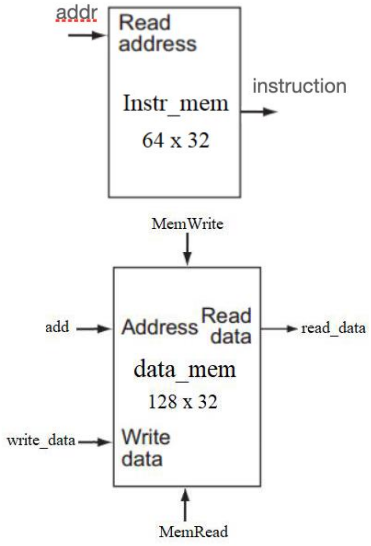
89869346

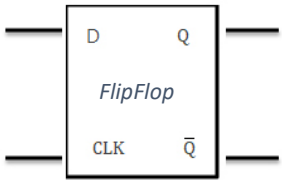
13 March 2022

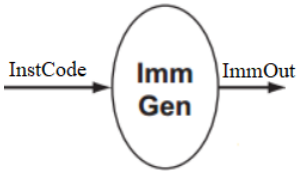
---

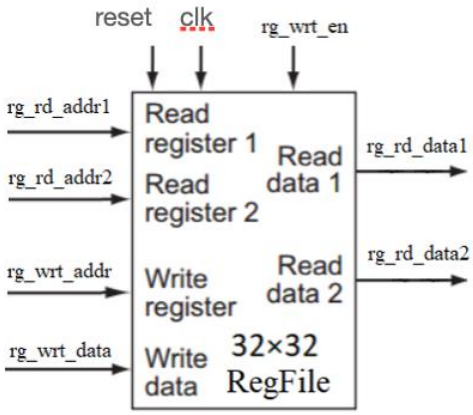
1 Objective

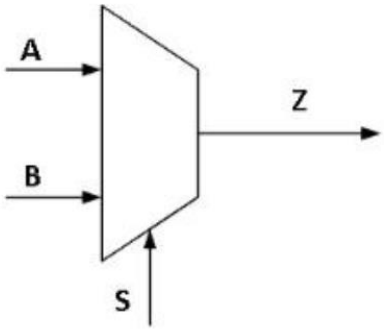
The objective of this lab is to design a RISC-V Single Cycle Processor. The RegFile, InstMem, FlipFlop, ImmGen, Mux, ALU, DataMem, Datapath, **Controller**, **ALUController**, and **processor** block diagrams are defined below:

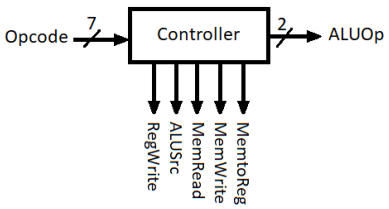


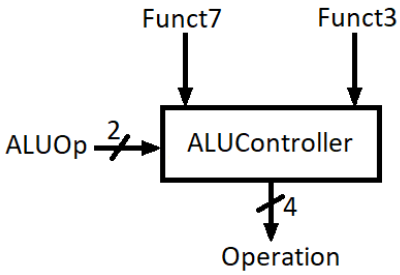


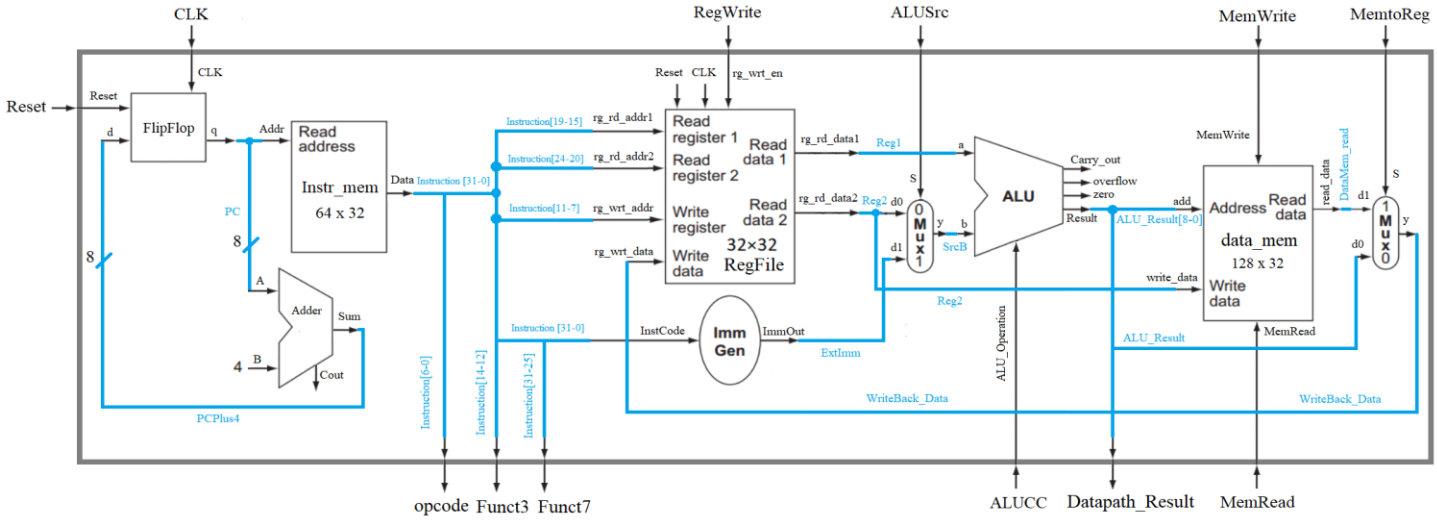


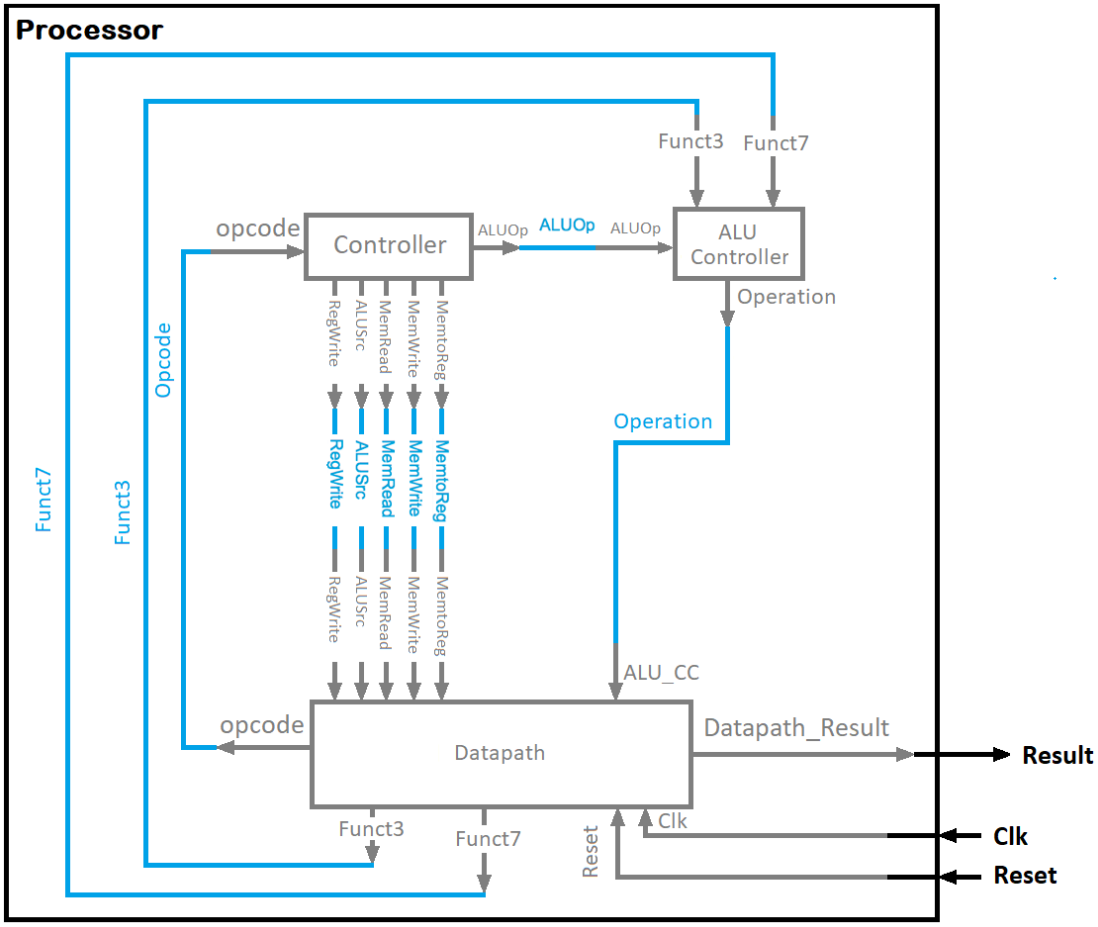









CLK
RegWrite
ALUSrc
MemWrite
MemtoReg
Reset
FlipFlop
Addr
Read address
Instr_mem
64 x 32
PC
Adder
Sum
Cout
PCPlus4
Data
Instruction [31-0]
Instruction[19-15]
Instruction[24-20]
Instruction[11-7]
rg_rd_addr1
rg_rd_addr2
rg_wrt_addr
rg_wrt_en
Read register 1
Read register 2
Write register
Write data
Read data 1
Read data 2
32×32 RegFile
rg_rd_data1
rg_rd_data2
Reg1
Reg2
SrcB
ALU
Carry_out
overflow
zero
Result
ALU_Result[8-0]
Address
data_mem
128 x 32
Read data
Write data
write_data
read_data
DataMem_read
MemRead
ALU_Result
InstCode
Imm Gen
ImmOut
ExtImm
rg_wrt_data
WriteBack_Data
ALU_Operation
Instruction[6-0]
Instruction[14-12]
Instruction[31-25]
opcode
Funct3
Funct7
ALUCC
Datapath_Result


Processor
Funct3
Funct7
opcode
Controller
ALUOp
ALUOp
ALUOp
ALU Controller
Operation
Opcode
RegWrite
ALUSrc
MemRead
MemWrite
MemtoReg
Operation
ALU_CC
opcode
Datapath
Datapath_Result
Result
Funct3
Funct7
Reset
Clk
Clk
Reset

2 Procedure

The procedure of creating these elements is explained as follows:

DATAPATH:

Assign both output wires to their respective data from the register index

When there is a reset signal detected, we clear all registers.
On rising clock signal,
    if reset is not detected and write_en is on
        write the data into the register index

The ImmGen module is straightforward. Upon receiving an instruction code, we return the Imm portion of the code depending on the sequence of the last seven bits.

0000011: if MSB = 1: return {20{1'b1}} else, return 20'b0, InstCode[31:20]
0010011: if MSB = 1: return {20{1'b1}} else, return 20'b0,InstCode[31:20]
0100011: if MSB = 1: return {20{1'b1}} else, return 20'b0,InstCode[31:25+11:7]
0010111: return InstCode[31:12], 12'b0
Default: return 0

The Mux will be converted to handle 32 bit data, given D1, D0, and a select bit, S:

If S = 1, return D1
Else return D0

ALU selections were programmed using the following logic:

```
Overflow = 0; Carry_Out = 0;

AND:  ALU_Result = A_in & B_in;

OR:        ALU_Result = A_in | B_in;

NOR:  ALU_Result = ~(A_in|B_in);

ADD:  ALU_Result = $signed ( A_in ) + $signed ( B_in );
      temp = {1'b0 , A_in } + {1'b0 , B_in };
      Carry_Out = temp [32];
      Overflow = ( A_in [31] & B_in [31] & ~ ALU_Out [31]) |
      33 (~ A_in [31] & ~ B_in [31] & ALU_Out [31]) ? 1’b1 : 1’b0;

SUB:  ALU_Result = $signed ( A_in ) - $signed ( B_in ) ;
      twos_com = ~( B_in ) + 1'b1;
      Overflow = ( A_in [31] & twos_com [31] & ~ ALU_Out [31]) |
                 (~ A_in [31] & ~ twos_com [31] & ALU_Out [31])
                 ? 1’b1 : 1’b0;

EQ:   ALU_Result = ( A_in == B_in ) ? 32 ' d1: 32' d0;

LT:   ALU_( $signed ( A_in ) < $signed ( B_in ))?32 ' d1:32' d0;
After the case, zero flag is calculated:
assign Zero = ALU_Result == 0;
```

DataMem is our internal memory. Like InstMem, it holds data and can write data within itself given the right instructions.

If MemRead is ON

Output the read data

If MemWrite is ON

Write the data from the bus into the address

Finally, the Datapath.

The Datapath links all our modules together with wires/buses.

The FlipFlop helps us increment and reset the pc counter.

The pc counter output from the FF is inputted int the InstMem read addr port

The output instruction from the counter address is passed into

[6:0] => opcode

[14:12] => funct3

[31:25] => funct7

[31:0] => ImmGen Input (InstrCode)

[19:15] => RegFile Read Address 1

[24:20] => RegFile Read Address 2

[11:7] => RegFile Write Address

The RegFile Takes the above inputs, as well as:

WriteBack_Data wire from the MemtoReg output

Reset, Clock, RegFile Write Signal

And given the write/read code, it will output:

Reg Read Data 1 -> Reg1 wire

Reg Read Data 2 -> Reg2 wire

The RegFile Mux then takes inputs from the ImmGen and RegFile Data 2 bus:

Inputs: Reg2 wire

ExtImm (ImmOut)

Alu_src (selector)

To output the designated signal as SrcB

The ALU then plays the part of taking in inputs of:

Inputs: Reg1

SrcB

Alu control code

To form the output Result into data bus ALU_Result[8:0]

The second to last module in the sequence is the

DataMem module. This module takes in inputs of:

Inputs: Address from ALU_Result[8:0]

WriteData from the Reg2 bus

MemWrite, MemRead signals

And if told to write

will write Reg2 into Memory[ALU_Result[8:0]]

if told to read

will output Memory[ALU_Result[8:0]]

into the DataMem_read bus

At Last, our DMem Mux, uses the:

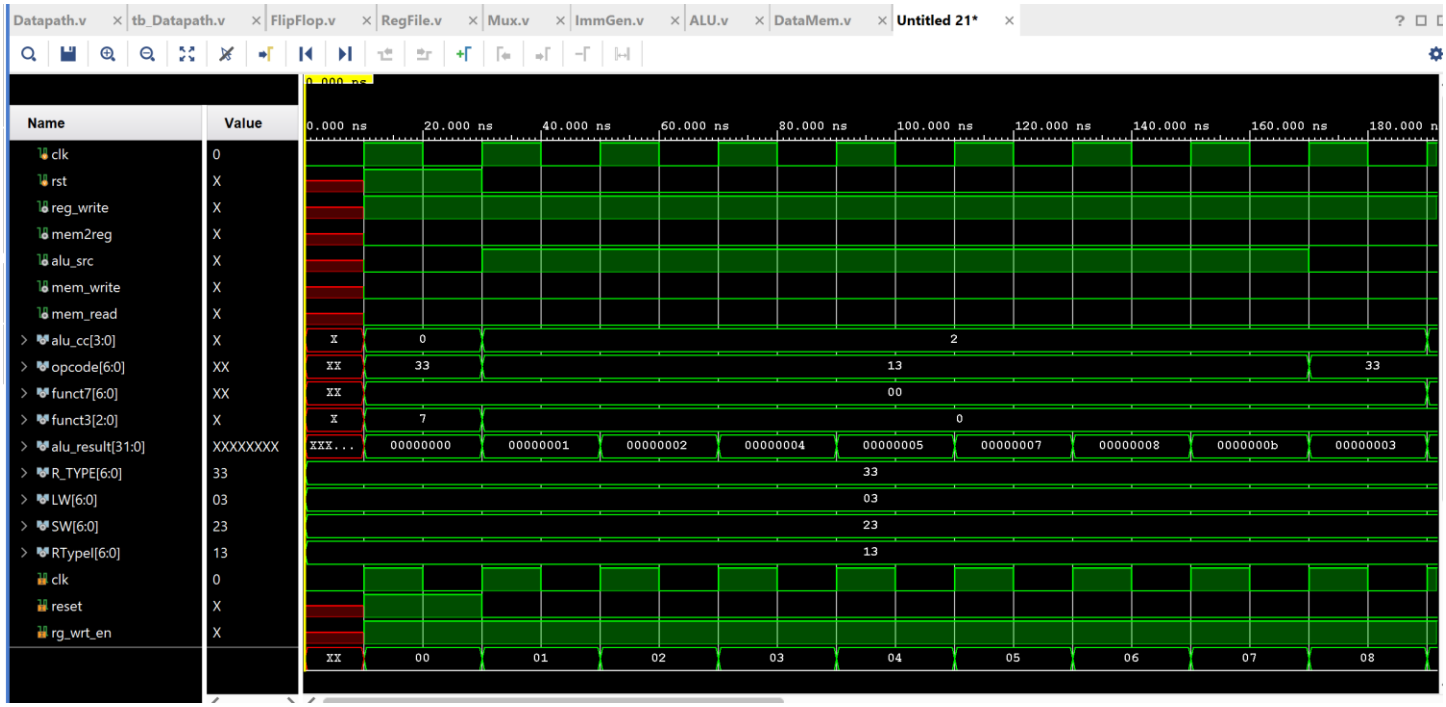
MemToReg signal as a selector

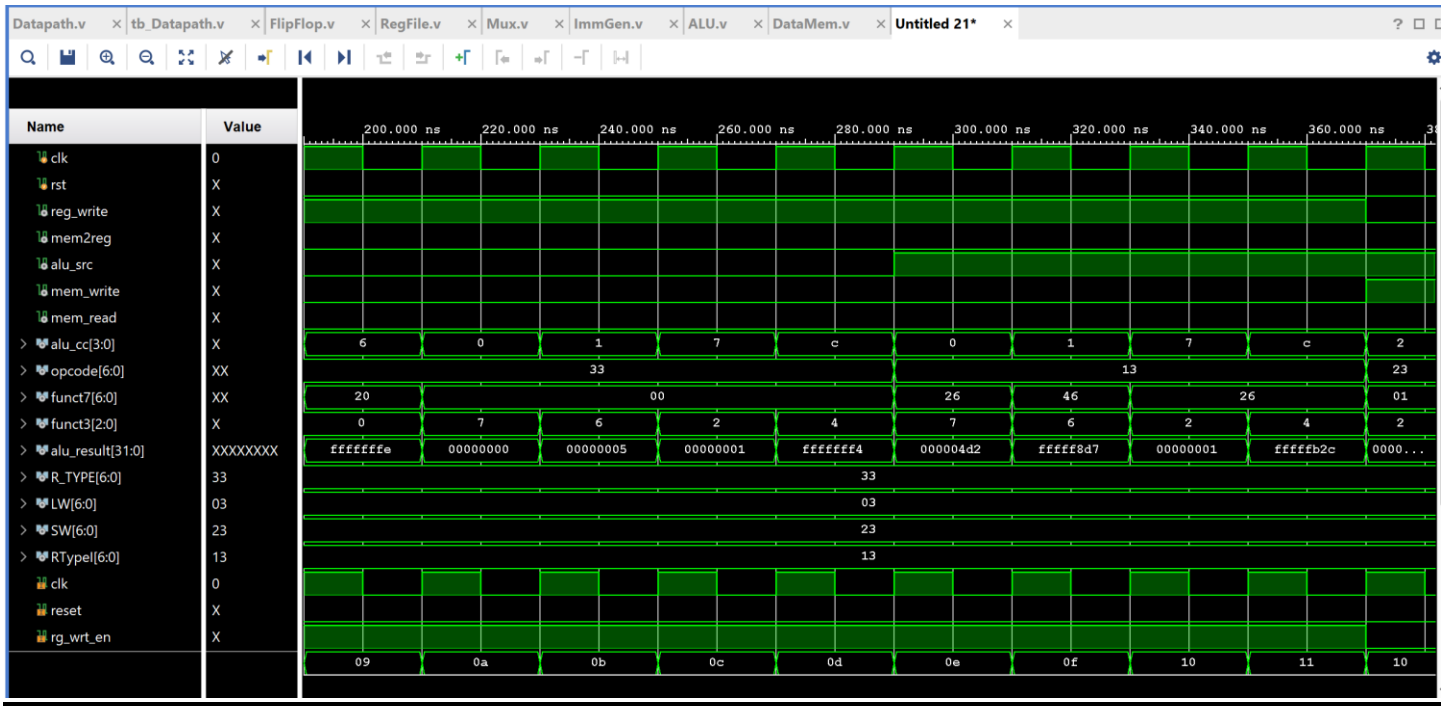
DataMem_read as D1

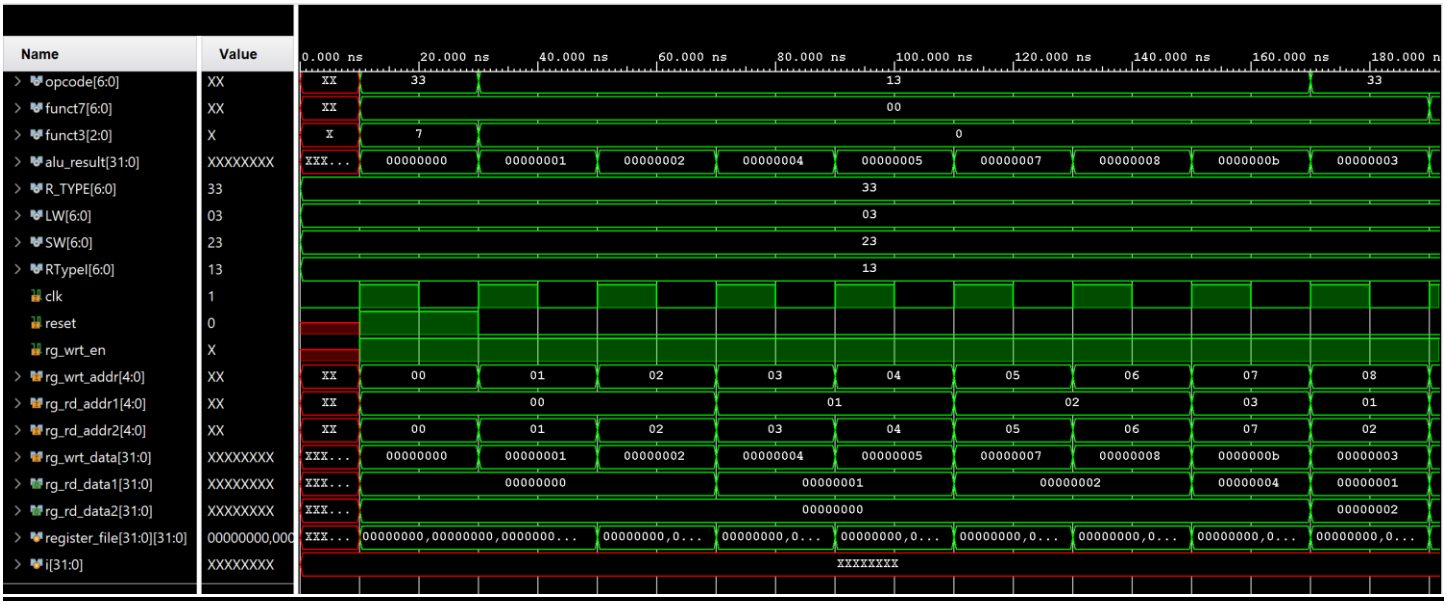
ALU_Result as D0

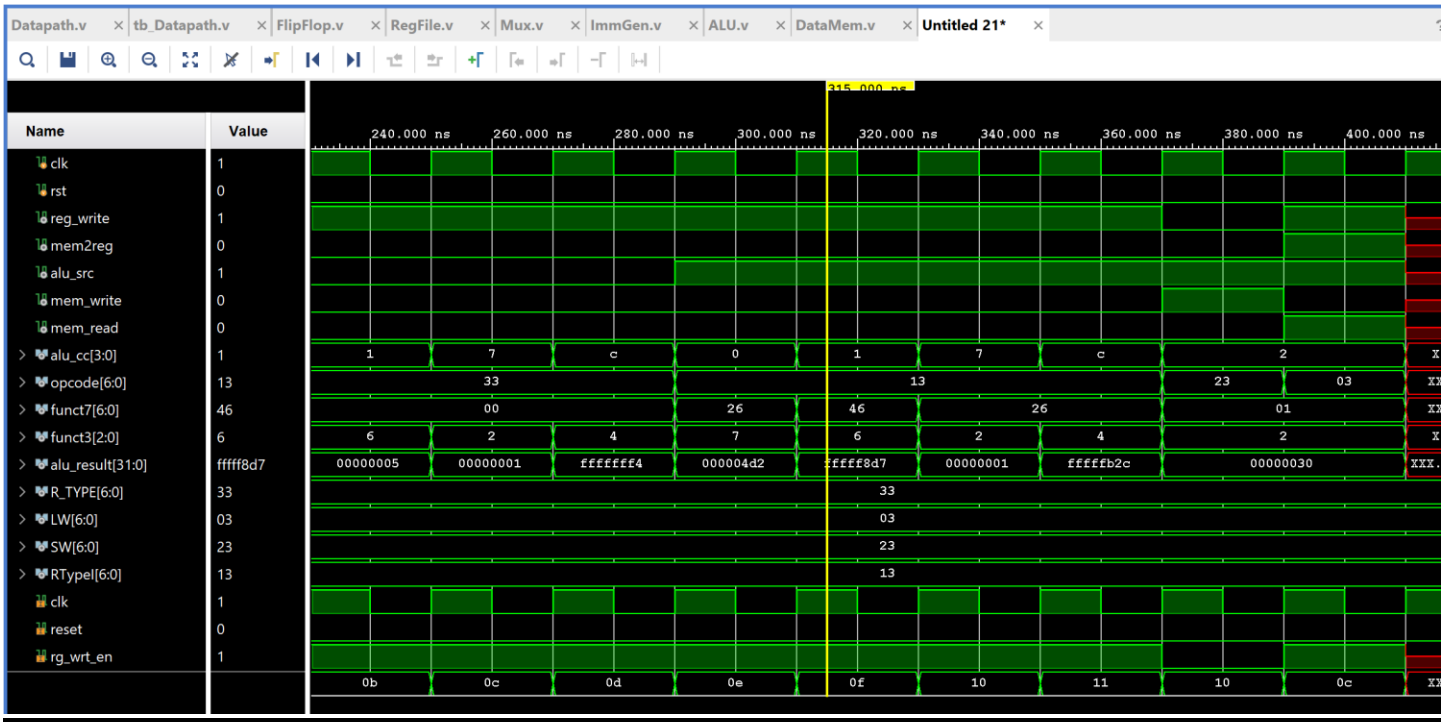
To select an output to send in the WriteBack_Data bus, which gets inputted back into the RegFile as Reg Write Data.

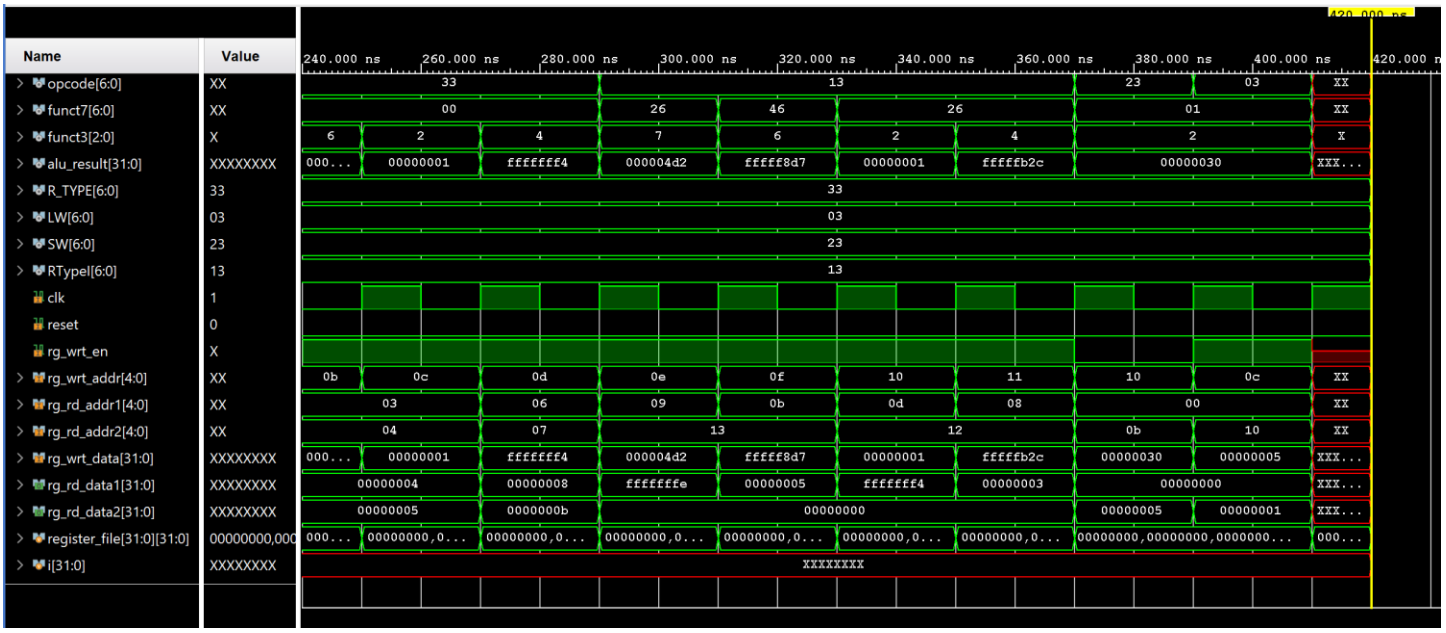
The Waveform that we observe at the

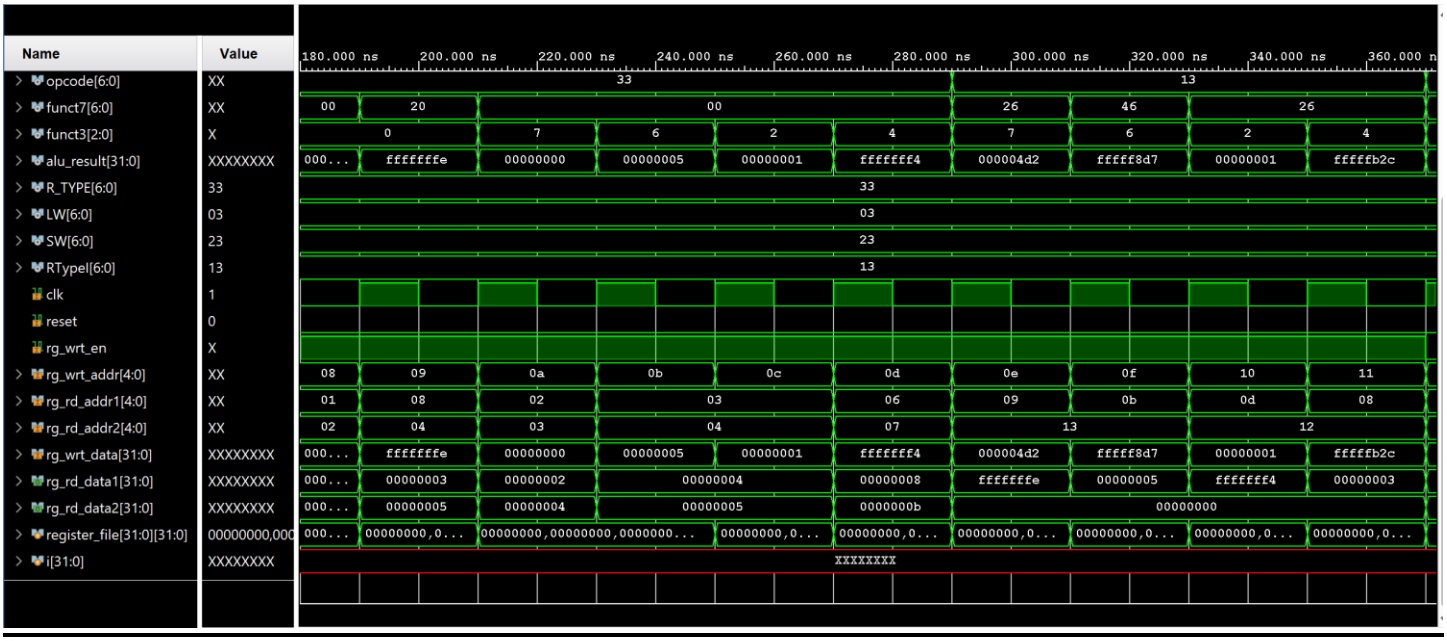



Datapath.v
tb_Datapath.v
FlipFlop.v
RegFile.v
Mux.v
ImmGen.v
ALU.v
DataMem.v
Untitled 21*
Name
Value
clk
rst
reg_write
mem2reg
alu_src
mem_write
mem_read
alu_cc[3:0]
opcode[6:0]
funct7[6:0]
funct3[2:0]
alu_result[31:0]
R_TYPE[6:0]
LW[6:0]
SW[6:0]
RTypeI[6:0]
clk
reset
rg_wrt_en
200.000 ns
220.000 ns
240.000 ns
260.000 ns
280.000 ns
300.000 ns
320.000 ns
340.000 ns
360.000 ns


Name
Value
opcode[6:0]
funct7[6:0]
funct3[2:0]
alu_result[31:0]
R_TYPE[6:0]
LW[6:0]
SW[6:0]
RTypeI[6:0]
clk
reset
rg_wrt_en
rg_wrt_addr[4:0]
rg_rd_addr1[4:0]
rg_rd_addr2[4:0]
rg_wrt_data[31:0]
rg_rd_data1[31:0]
rg_rd_data2[31:0]
register_file[31:0][31:0]
i[31:0]
0.000 ns
20.000 ns
40.000 ns
60.000 ns
80.000 ns
100.000 ns
120.000 ns
140.000 ns
160.000 ns


Datapath.v
tb_Datapath.v
FlipFlop.v
RegFile.v
Mux.v
ImmGen.v
ALU.v
DataMem.v
Untitled 21*
Name
Value
clk
rst
reg_write
mem2reg
alu_src
mem_write
mem_read
alu_cc[3:0]
opcode[6:0]
funct7[6:0]
funct3[2:0]
alu_result[31:0]
R_TYPE[6:0]
LW[6:0]
SW[6:0]
RTypeI[6:0]
clk
reset
rg_wrt_en
240.000 ns
260.000 ns
280.000 ns
300.000 ns
320.000 ns
340.000 ns
360.000 ns
380.000 ns
400.000 ns


Name
Value
opcode[6:0]
funct7[6:0]
funct3[2:0]
alu_result[31:0]
R_TYPE[6:0]
LW[6:0]
SW[6:0]
RTypeI[6:0]
clk
reset
rg_wrt_en
rg_wrt_addr[4:0]
rg_rd_addr1[4:0]
rg_rd_addr2[4:0]
rg_wrt_data[31:0]
rg_rd_data1[31:0]
rg_rd_data2[31:0]
register_file[31:0][31:0]
i[31:0]
240.000 ns
260.000 ns
280.000 ns
300.000 ns
320.000 ns
340.000 ns
360.000 ns
380.000 ns
400.000 ns
420.000 ns


Name
Value
opcode[6:0]
funct7[6:0]
funct3[2:0]
alu_result[31:0]
R_TYPE[6:0]
LW[6:0]
SW[6:0]
RTypeI[6:0]
clk
reset
rg_wrt_en
rg_wrt_addr[4:0]
rg_rd_addr1[4:0]
rg_rd_addr2[4:0]
rg_wrt_data[31:0]
rg_rd_data1[31:0]
rg_rd_data2[31:0]
register_file[31:0][31:0]
i[31:0]
180.000 ns
200.000 ns
220.000 ns
240.000 ns
260.000 ns
280.000 ns
300.000 ns
320.000 ns
340.000 ns
360.000 ns

Processor

On each rising clock signal:

The datapath will output:

OPCODE

FUNCT3

FUNCT7

Which will feed into the input busses of:

Opcode -> Controller

In which case: the controller will output

```
MemtoReg =  (Opcode == 7'b0000011) ? 1'b1 : 1'b0;
MemWrite =  (Opcode == 7'b0100011) ? 1'b1 : 1'b0;
MemRead  =  (Opcode == 7'b0000011) ? 1'b1 : 1'b0;
ALUSrc   =  (Opcode == 7'b0110011) ? 1'b0 : 1'b1;
RegWrite =  (Opcode == 7'b0100011) ? 1'b0 : 1'b1;
ALUOp[1] =  &Opcode[5:4];
ALUOp[0] =  ~Opcode[4];
```

Funct3 -> ALUController

Funct7 -> ALUController

Controller -> ALUOp -> ALUController

In which case: the ALU controller will output

```
assign Operation[0] = (Funct3 == 3'b110) ||
(Funct3 == 3'b010 && (ALUOp == 2'b00 || ALUOp == 2'b10))
    ? 1'b1 : 1'b0;
assign Operation[1] = (Funct3 == 3'b000) || (Funct3 == 3'b010)
    ? 1'b1 : 1'b0;
```

```
assign Operation[2] = ((ALUOp == 2'b10) &&
     (Funct3 == 3'b100 || Funct3 == 3'b010) &&
     (|Funct7 == 1'b0)) || ((Funct7[5] == 1'b1) &&
     (Funct3 == 3'b000) && (ALUOp == 2'b10)) ||
     ((ALUOp == 2'b00) && (Funct3 == 3'b100 || Funct3 == 3'b010))
          ? 1'b1 : 1'b0;
assign Operation[3] =   (Funct3 == 3'b100) ? 1'b1 : 1'b0;
```

This is parallel to what the table given in the manual described:

	Funct7	Funct3	ALUop	Operation			
AND	0000000	111	10	0	0	0	0
OR	0000000	110	10	0	0	0	1
NOR	0000000	100	10	1	1	0	0
SLT	0000000	010	10	0	1	1	1
ADD	0000000	000	10	0	0	1	0
SUB	0100000	000	10	0	1	1	0
ANDI	-	111	00	0	0	0	0
ORI	-	110	00	0	0	0	1
NORI	-	100	00	1	1	0	0
SLTI	-	010	00	0	1	1	1
ADDI	-	000	00	0	0	1	0
LW	-	010	01	0	0	1	0
SW	-	010	01	0	0	1	0

Afterwards, the values of MemtoReg, MemWrite, MemRead, ALUSrc, and RegWrite (from the Controller) and the 4 bit Operation code (from the ALUController) are fed back into their respective Datapath input busses in order to output the datapath Result and start the next instruction at the rising clock signal.