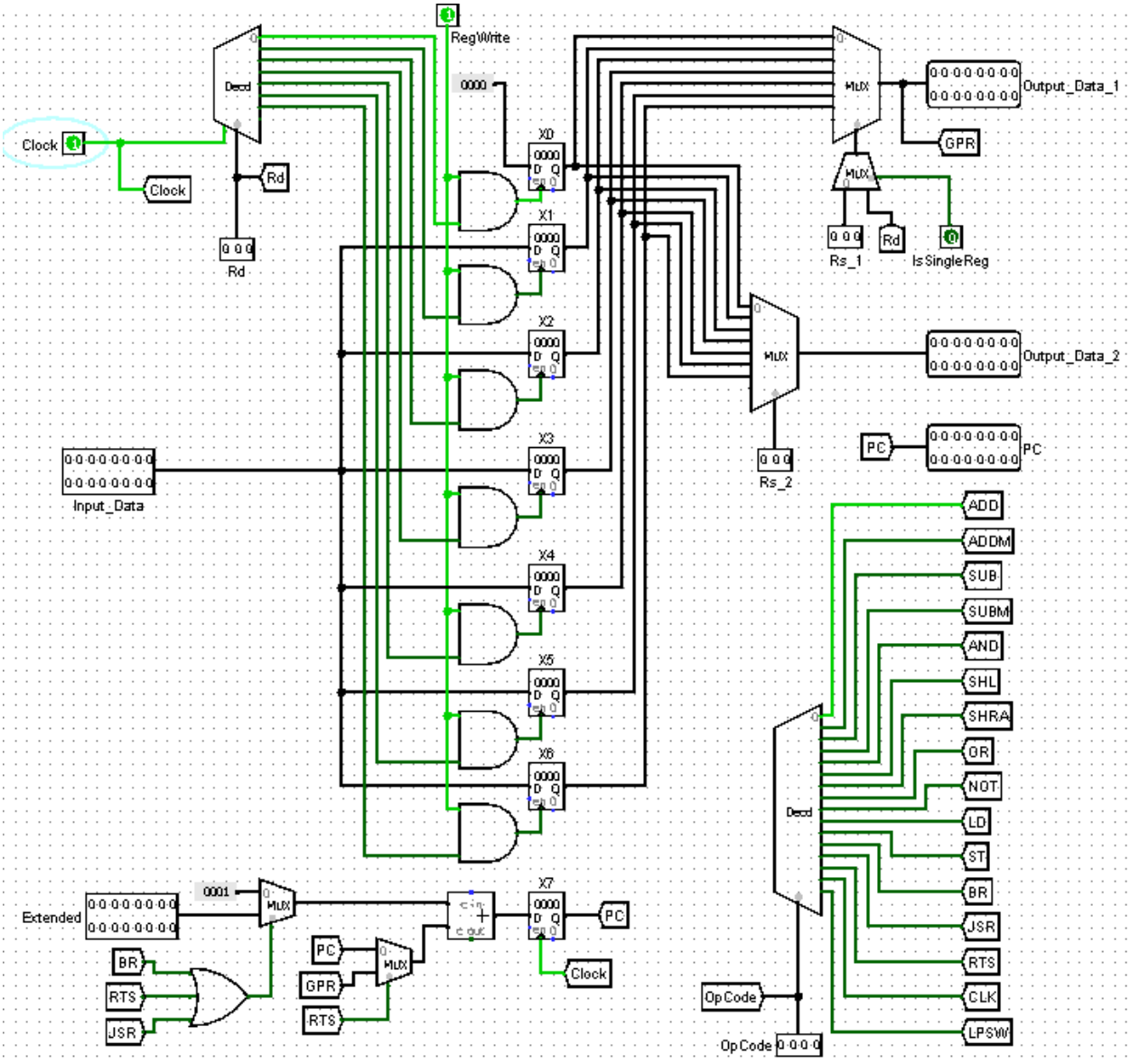
Project Title: 16-bit Processor design

Team Members: Kyle Johnson and Sandro Leon

Class: EECE 3026

Date Submitted: 7-8-16

Register File



Register File	
Input(s)	**Output(s)**
Rd	Output_Data_1
Rs1	Output_Data_2
Rs2	PC
Clock	
RegWrite	
InputData	
Extended	
IsSingleReg	
OpCode	

Description of I/O

Rd – 3-bit Input of destination register index

Rs1 – 3-bit Input of operand #1 register index

Rs2 – 3-bit Input of operand #2 register index

Clock – 1-bit Input of clock signal used to enable decoder output

RegWrite – 1-bit Input of Register Write Enable signal

InputData – 1-bit Data to be written to the selected register if RegWrite is High

Extended – 16-bit Input sign extended signal (It can be short or long offset extended based on external logic)

IsSingleReg – 1-bit Input Control signal denoting a single register instruction (Only Rd is used)

OpCode – 4-bit Input Opcode used to determine which instruction is being executed

Output_Data_1 – 16-bit Output of a single selected Register (be either Rd or Rs1 depending on logic)

Output_Data_2 – 16-bit Output of a single selected Register (Rs2)

PC – 16-bit Output of program counter (Reg[7])

Summary of Circuit Functionality

The register file brings in critical functionality to any CPU. It allows the data in the registers to be read from and written depending on a given instruction and its inherent control signal.

Writing Data to a Register

A decoder (with selection signal Rd) is used to select which register will be written to. A register can only be written to if it both selected and the RegWrite control signal is high.

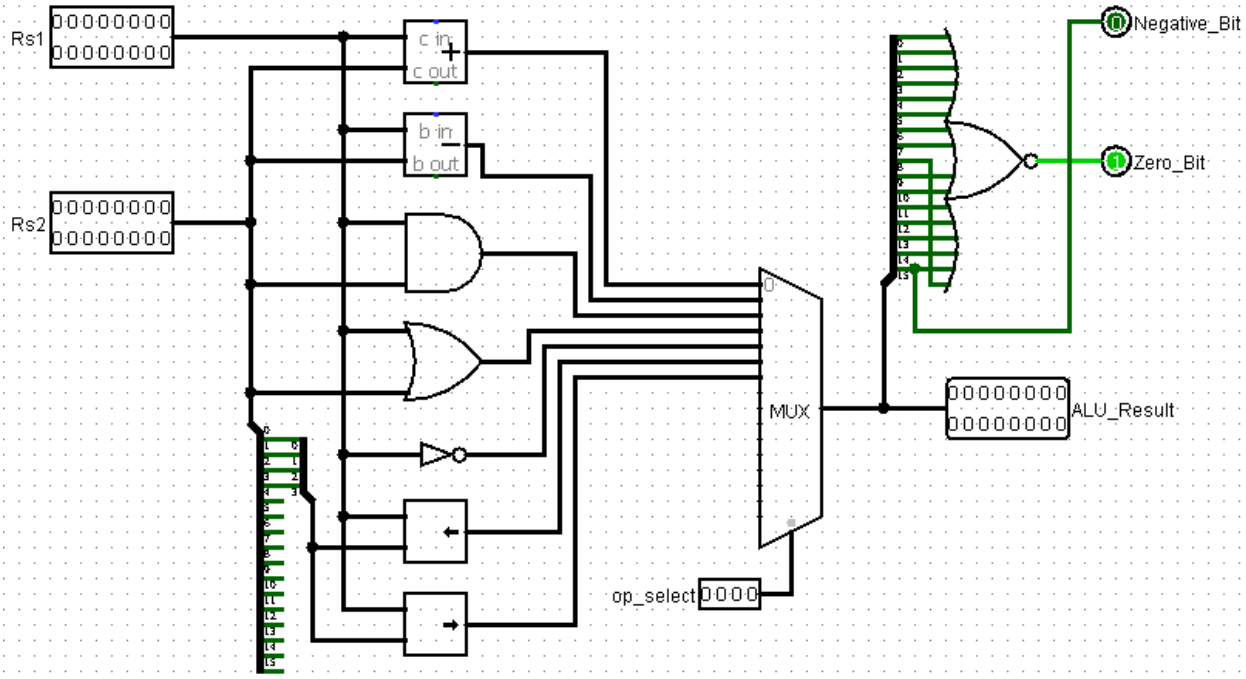
Reading Data from a Register

A multiplexer is used to select which register's data will be read from. There are 2 multiplexers found in order to accommodate the reading of 2 registers at once. Inputs Rs1 and Rs2 are used as the selector bits in order to choose which registers to read from. Special accommodations must be made for D-Type instructions where there is no Rs1 or Rs2. The control signal IsSingleReg is used to select Rd as Read Data 1 instead of Rs1 during these instructions.

Program Counter

The logic for the program counter is implemented in the register file. The PC (Reg[7]) operates on the falling edge of the clock signal. During most instructions, the PC simply increments by one at the completion of the instruction. For certain instruction/conditions alternative values for PC must be applied. A decoder is also used (with the selector bit being the OpCode) in order to determine which instruction is active. Logic is implemented in order to handle when the PC is not just simply incremented by 1.

ALU



ALU	
Input(s)	**Output(s)**
Rs1	Negative_Bit
Rs2	Zero_Bit
op_select	ALU_Result

Description of I/O

Rs1 – 16-bit Input of operand #1

Rs2 – 16-bit Input of operand #2

op_select – 4-bit ALU operation mode select

Negative_Bit – 1-bit output denoting a negative result from ALU

Zero_Bit – 1-bit output denoting a zero result from ALU

ALU_Result – 16-bit containing the result of the ALU operation

Summary of Circuit Functionality

The ALU is used to perform a multitude of operations on the operand(s) given. The included functions of our ALU are ADD, SUB, AND, OR, NOT, Shift_Left, and Shift_Right. The ALU also has the functionality to determine whether the output is negative as well as zero.

Selection of ALU Operation

A 16x1 multiplexer is used to select which ALU operation is to be performed. The multiplexer's selection is controlled by the 4-bit op_select input.

Addition of Two Operands

The ALU accepts inputs Rs1 and Rs2. It then performs an addition operation using Logisim's built-in adder module. The result is then passed to the ouput ALU_Result.

Subtraction of Two Operands

The ALU accepts inputs Rs1 and Rs2. It then performs a subtraction operation using Logisim's built-in subtraction module. The result is then passed to the ouput ALU_Result.

AND of Two Operands

The ALU accepts inputs Rs1 and Rs2. It then performs a bit-wise AND operation using an AND gate. The result is then passed to the ouput ALU_Result.

OR of Two Operands

The ALU accepts inputs Rs1 and Rs2. It then performs a bit-wise OR operation using an OR gate. The result is then passed to the ouput ALU_Result.

NOT of an Operand

The ALU accepts an input. It then performs a bit-wise NOT operation using a NOT gate. The result is then passed to the ouput ALU_Result.

Shift Left Operation

The ALU accepts bits 3-0 of input Rs2 using a splitter. It then performs a logical shift left operation using Logisim's built-in left shift module. The result is then passed to the ouput ALU_Result.

Shift Right Operation

The ALU accepts bits 3-0 of input Rs2 using a splitter. It then performs a logical shift right operation using Logisim's built-in right shift module. The result is then passed to the ouput ALU_Result.

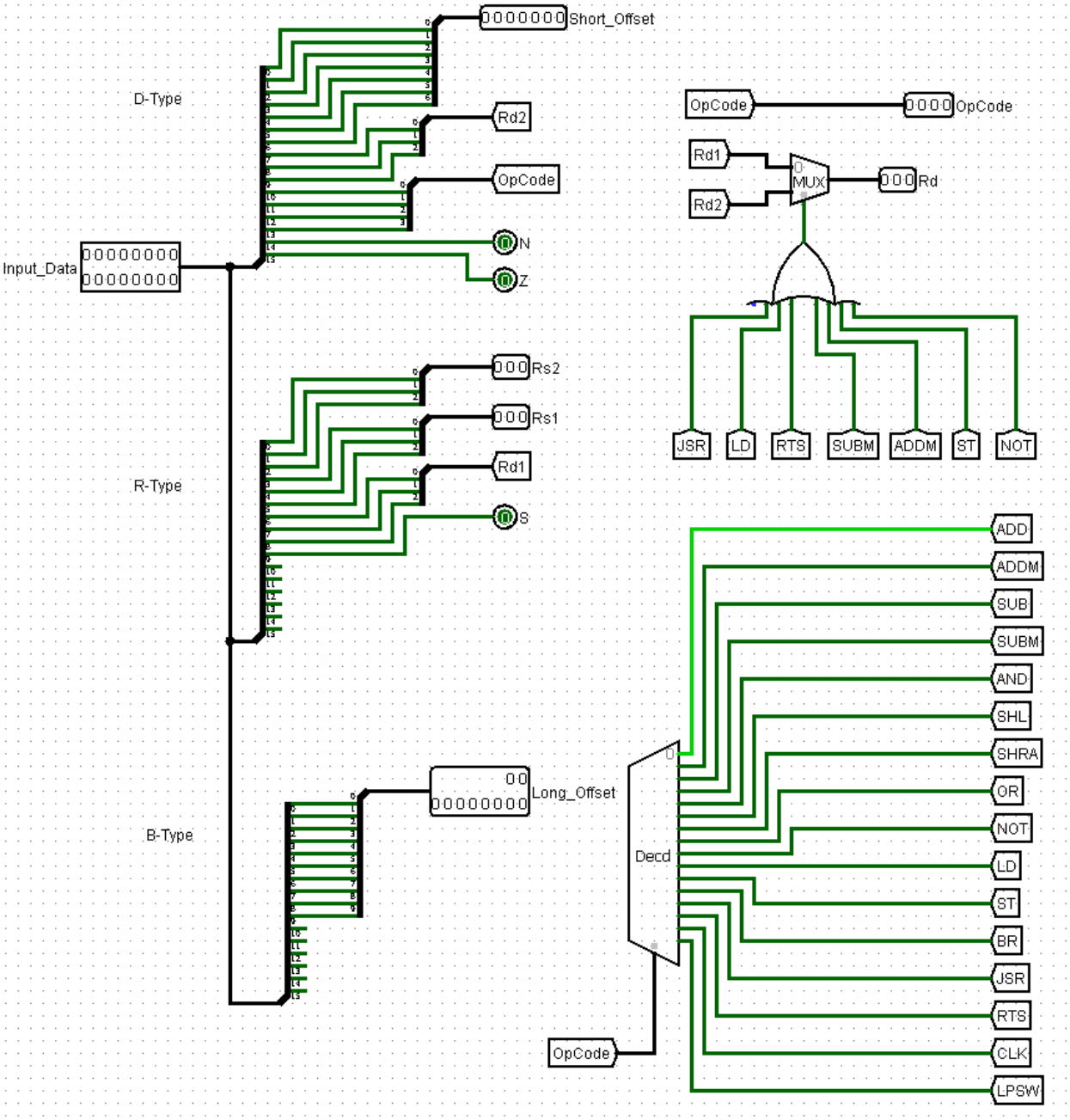
Negative Bit Detection

Using a splitter, the ALU result is parsed into 16 individual bits. The negative bit detection works by tying the most significant bit of the ALU result to the Negative_Bit output.

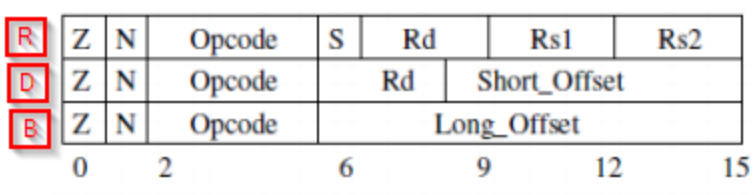
Zero Result Detection

Using a splitter, each bit of the ALU result is fed into a NOR gate. The Zero_Bit output only enables when all bits of the ALU_Result are 0.

Instruction Decode



Instruction Type Table



In order to more clearly identify the three instruction types, we have assigned types of R, D, and B for Arithmetic, Data, and Branch type instructions respectively as seen in the figure above.

Instruction Decoder	
Input(s)	**Output(s)**
Input_Data	Short_Offset
	N
	Z
	S
	Rs1
	Rs2
	Rd
	Long_Offset
	OpCode

Description of I/O

Input_Data – 16-bit input instruction read from ROM

Short_Offset - 7-bit output used in the instructions requiring the short offset

N – 1-bit output denoting whether the N bit of the instruction is HIGH

Z – 1-bit output denoting whether the Z bit of the instruction is HIGH

S – 1-bit output denoting whether the S bit of the instruction is HIGH

Rs1 – 3-bit output denoting the index of Rs1 in the register file

Rs2 – 3-bit output denoting the index of Rs2 in the register file

Rd – 3-bit output denoting the index of Rd in the register file

Long_Offset - 10-bit output used in the instructions requiring the long offset

OpCode – 4-bit output denoting which operations will be performed for the given instruction

Summary of Circuit Functionality

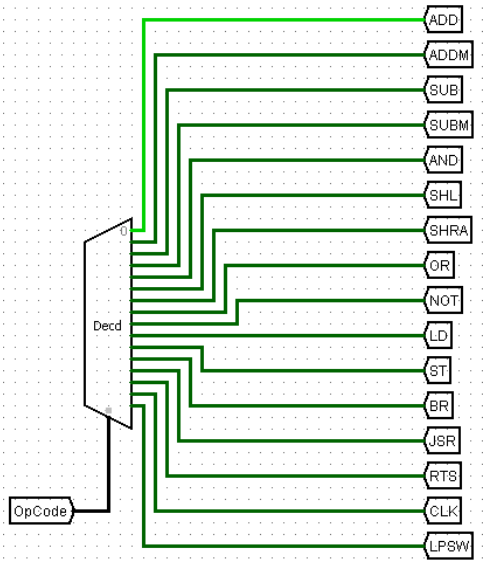
The instruction decoder (ID) is used to decode the instruction memory. The main functionality of the circuit consists of parsing the 16-bit instruction with splitters into functional indexes and values.

Parsing the 16-Bit Instruction

A 16-bit input instruction is fed into the instruction decoder. Splitters are implemented in order to extract instruction components in the following ways:

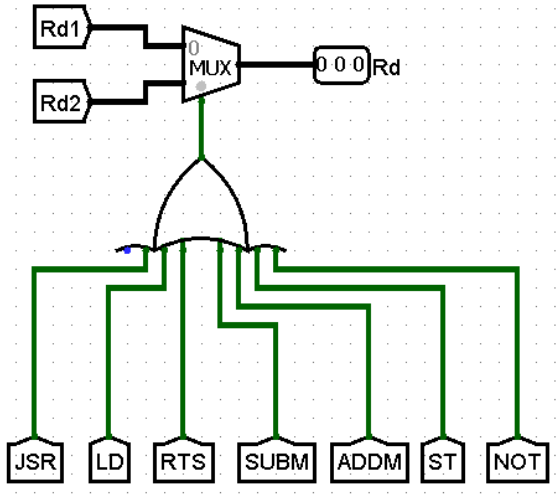
- Short_Offset
  - 7-bit output is read from the instruction bits 6-0
- N
  - 1-bit output is read from the instruction bit 15
- Z
  - 1-bit output is read from the instruction bit 14
- S
  - 1-bit output is read from the instruction bit 9
- Rs1
  - 3-bit output is read from the instruction bits 5-3
- Rs2
  - 3-bit output is read from the instruction bits 2-0
- Rd1
  - 3-bit output is read from the instruction bits 8-6
- Rd2
  - 3-bit output is read from the instruction bits 9-7
- Long_Offset
  - 10-bit output is read from the instruction bits 9-0
- OpCode
  - 4-bit output is read from the instruction bits 13-10

Determining The OpCode



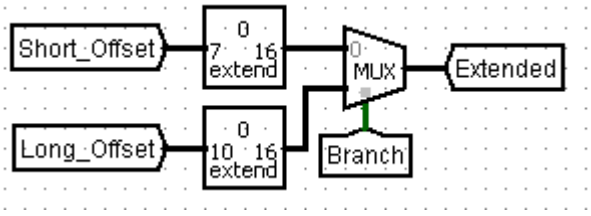
The 4-bit OpCode is used as the decoder's selector input. The output of the decoder indicates which instruction is active. The active instruction is used as a control bit in other parts of the instruction decoder.

Determining The Rd Output



A multiplexer is used to determine whether Rd1 or Rd2 should be assigned as the Rd output. The zero-state of the multiplexer assigns Rd1 to the Rd output for R type instructions. The HIGH-state of the multiplexer assigns Rd2 to the Rd output for D type instructions. Using an OR gate, the selector bit for the multiplexer is HIGH if a D type instruction is HIGH.

Sign-Extend



Sign-Extend	
Input(s)	**Output(s)**
Short_Offset	Extended
Long_Offset	
Branch	

Description of I/O

Short_Offset - 7-bit input given to Logisim's sign extension module

Long_Offset - 10-bit input given to Logisim's sign extension module

Branch – 1-bit input used to control the selection of the multiplexer

Extended – 16-bit output denoting the offset needed for the given instruction

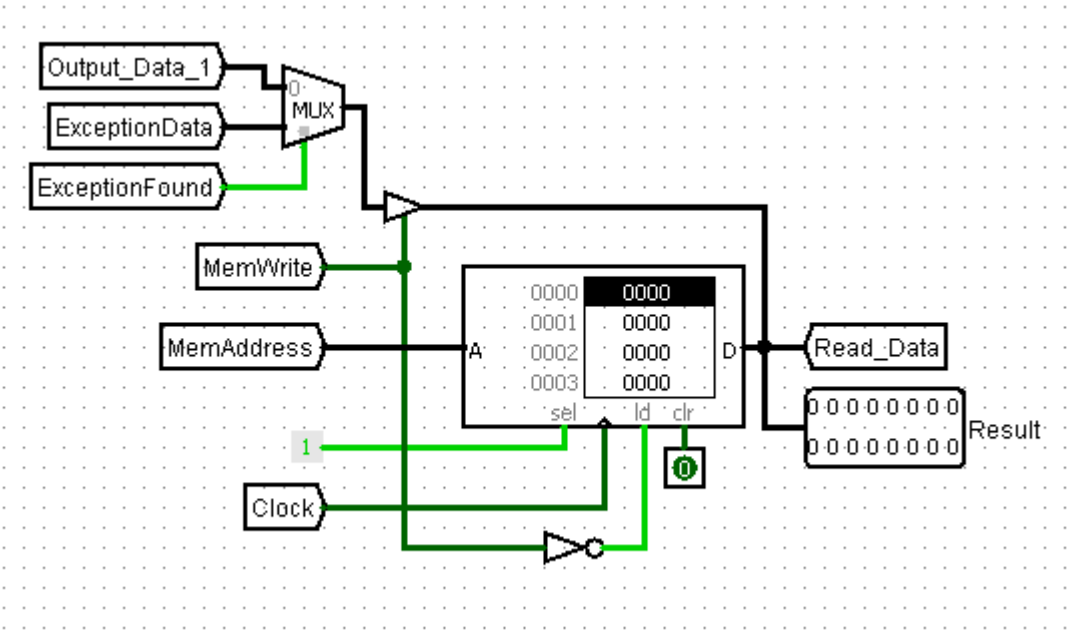
Summary of Circuit Functionality

The Sign Extend Module is able to turn either a 7-bit offset value (short), or a 10-bit offset value (long) into a 16-bit sign extended output.

Branch Multiplexer

The zero-state of the multiplexer assigns a 16-bit extended Short_Offset to the Extended output for D type instructions. The HIGH-state of the multiplexer assigns a 16-bit extended Long_ Offset to the Extended output for B type instructions. The selector bit, Branch, for the multiplexer is HIGH if a B type instruction is HIGH, and the zero-input state is used for D type instructions.

RAM



RAM	
Input(s)	**Output(s)**
Output_Data_1	Read_Data
ExceptionData	
ExceptionFound	
MemWrite	
MemAddress	
Clock	

Description of I/O

Output_Data_1 – 16-bit input data to be written to the RAM

ExceptionData - 16-bit input data to be written to the RAM

ExceptionFound – 1-bit input to control data to be written to the RAM

MemWrite – 1-bit input to enable data write to the RAM

MemAddress – 16-bit input denoting which address in RAM should be accessed/ written

Clock – 1-bit input used to update RAM on rising edge of clock

Read_Data – 16-bit output denoting the data stored in the given memory address

Summary of Circuit Functionality

The RAM module acts as a volatile memory that can be written to or read from depending on the instruction.

ExceptionFound Multiplexer

The zero-state of the multiplexer assigns a 16-bit Output_Data_1 to the output for non-exception instructions. The HIGH-state of the multiplexer assigns a 16-bit ExceptionData to the output for exception instructions. The selector bit, ExceptionFound, for the multiplexer is HIGH if an exception has been detected, and the zero-input state is used if no exception has been flagged.

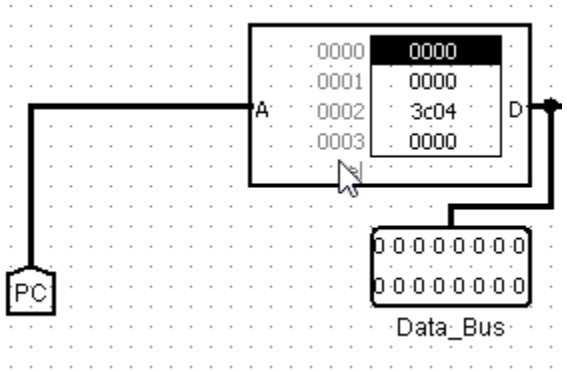
Writing To RAM

When the input MemWrite is HIGH, the RAM can be written to on the rising edge of the clock signal. A tri-state buffer conditionally allows the flow of the input data to the RAM.

Reading From RAM

When the input MemWrite is LOW, the RAM loads the contents of the current address into Read_Data. The tri-state buffer disables the flow of the input data to the RAM.

ROM



ROM	
Input(s)	**Output(s)**
PC	Data_Bus

Description of I/O

PC – 16-bit input Program Counter to indicate instruction address

Data_Bus – 16-bit output storing the instruction retrieved from ROM

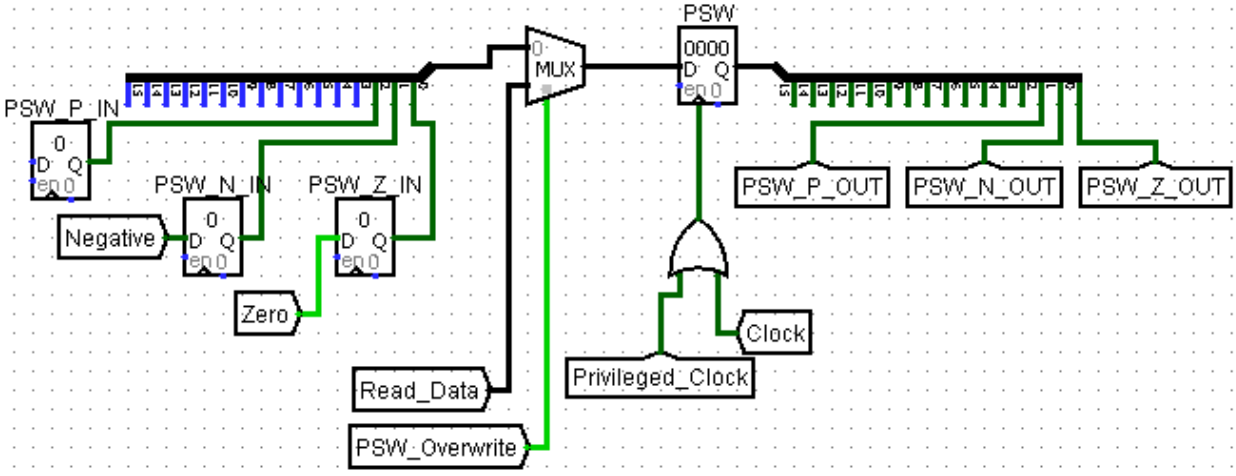
Summary of Circuit Functionality

The ROM module acts as a non-volatile memory that can be read from with each program count increment changing the instruction that is read.

Reading From ROM

The ROM uses the Program Counter input in order to select the address of the an instruction.

PSW



PSW	
Input(s)	**Output(s)**
PSW_P_IN	PSW_P_OUT
PSW_N_IN	PSW_N_OUT
PSW_Z_IN	PSW_Z_OUT
Read_Data	
PSW_Overwrite	
Privilaged_Clock	
Clock	

Description of I/O

PSW_P_IN – 1-bit input of the PSW register's privileged bit

PSW_N_IN – 1-bit input of the PSW register's negative bit

PSW_Z_IN – 1-bit input of the PSW register's zero bit

Read_Data – 16-bit input to the multiplexer

PSW_Overwrite – 1-bit input used to control the multiplexer

Privilaged_Clock – 1-bit input of the privileged clock signal

Clock – 1-bit input of the clock signal

PSW_P_OUT – 1-bit output of the PSW register's privileged bit

PSW_N_OUT – 1-bit output of the PSW register's negative bit

PSW_Z_OUT – 1-bit output of the PSW register's zero bit

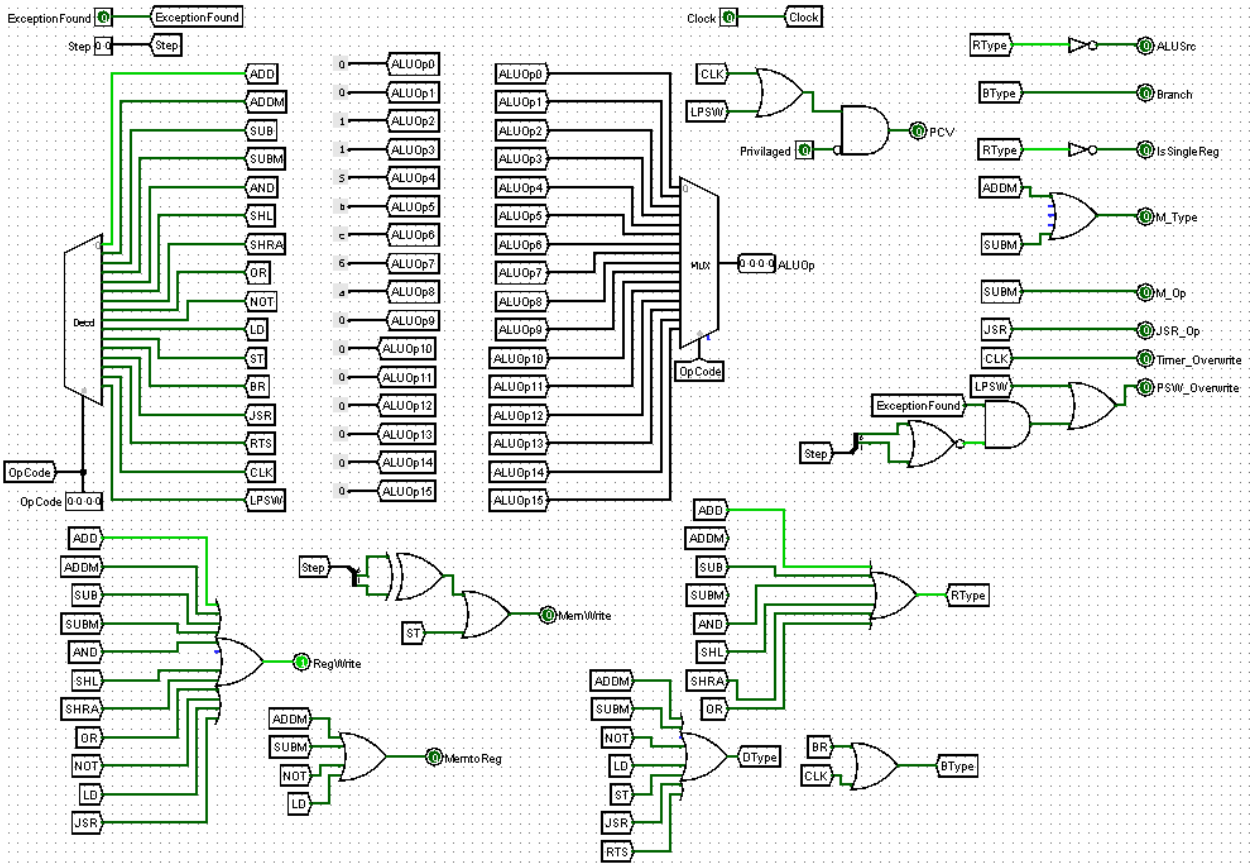
Summary of Circuit Functionality

Acts as a program status indicator with 3-bits denoting negative, zero, and privileged.

Choosing PSW Register's Input Data

The zero-state of the multiplexer assigns a 16-bit output containing the P, Z, and N values to be written to the PSW register. This multiplexer output is only used when a single bit of the PSW is to be written to. The HIGH-state of the multiplexer assigns a 16-bit output to the PSW register. This value is only selected when P, Z, and N must be written to at once. The selector bit, PSW_Overwrite, for the multiplexer is HIGH if the PSW's current P, Z, and N values must be overwritten at once, and the zero-input state is used if the PSW's current P, Z, and N bits are to be used.

Control



*Although the fine details of the picture may seem hard to read, the picture is high resolution and can be seen much more clearly if zoomed into.

Control	
Input(s)	**Output(s)**
ExceptionFound	PCV
Clock	ALUSrc
Step	Branch
PSW_P_OUT	IsSingleReg
OpCode	M_Type
	M_Op
	JSR_Op
	Timer_Overwrite
	PSW_Overwrite
	MemWrite
	RegWrite
	MemtoReg
	ALUOp

Description of I/O

ExceptionFound – 1-bit input indicating whether an exception has been flagged in the circuit

Clock – 1-bit input clock signal

Step – 2-bit input indicating which step of the exception handling process the circuit is on

PSW_P_OUT – 1-bit input indicating the status of the PSW's privileged bit

OpCode – 4-bit input indicating which instruction the circuit is handling

PCV – 1-bit output indicating the status of whether or not a program check violation has been found

ALUSrc – 4-bit output indicating where the ALU should be selecting the operands from

Branch – 1-bit output indicating if a B type instruction is being used

IsSingleReg – 1-bit output indicating if a single register instruction is executing

M_Type – 1-bit output indicating if an ADDM or SUBM instruction is being executed

M_Op – 1-bit output indicating if a SUBM instruction is being executed

JSR_Op – 1-bit output indicating if a JSR instruction is being executed

Timer_Overwrite – 1-bit output indicating whether or not the current timer value should be enabled to be overwritten

PSW_Overwrite – 1-bit output indicating whether or not the current PSW value should be enabled to be overwritten

MemWrite - 1-bit output indicating whether or not the current RAM value should be enabled to be overwritten

RegWrite - 1-bit output indicating whether or not the current selected register value should be enabled to be overwritten

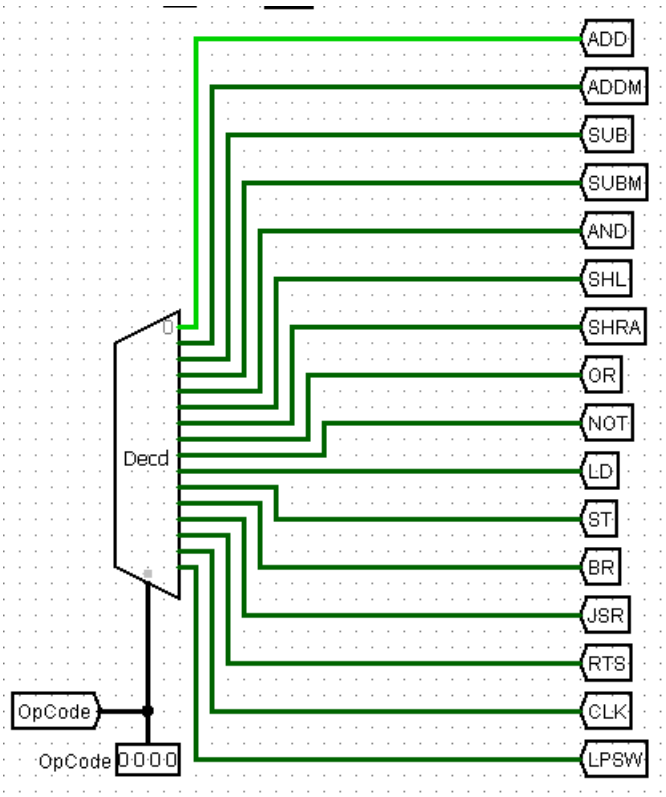
MemtoReg - 1-bit output indicating whether or not the current memory value should be written back to the selected register

ALUOp – 4-bit output indicating which ALU operation will be required for the current instruction

Summary of Circuit Functionality

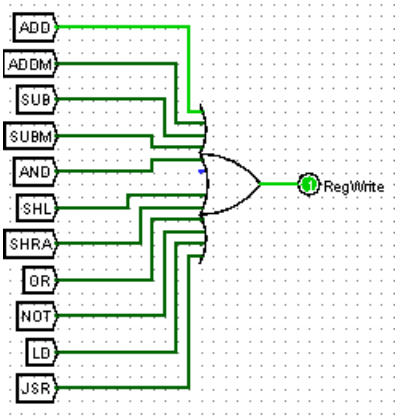
The control unit decodes the control signals needed to implement the active instruction.

Determining The OpCode



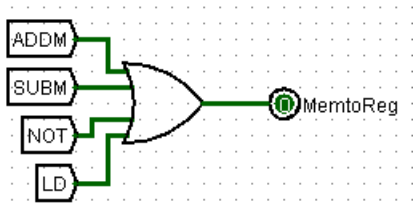
The 4-bit input OpCode is used as the decoder's selector input. The output of the decoder indicates which instruction is active. The active instruction is used as a control bit in other parts of the control unit.

Determining RegWrite



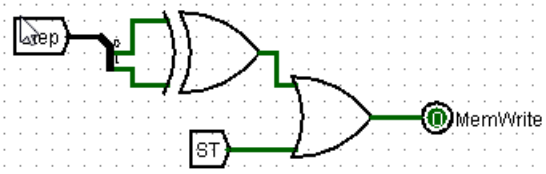
After the OpCode is decoded, the instructions requiring the RegWrite control signal to be HIGH are passed in as inputs to an OR gate. If any of the instructions (pictured above) have been selected via the OpCode, the output bit RegWrite will be toggled HIGH, otherwise it will remain in the LOW state.

Determining MemtoReg



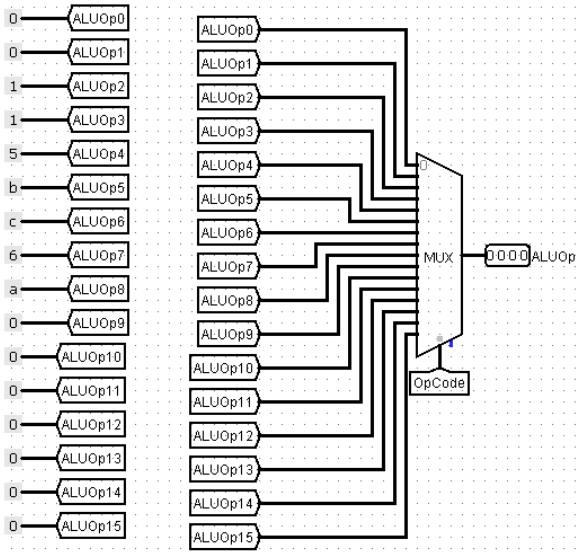
After the OpCode is decoded, the instructions requiring the MemtoReg control signal to be HIGH are passed in as inputs to an OR gate. If any of the instructions (pictured above) have been selected via the OpCode, the output bit MemtoReg will be toggled HIGH, otherwise it will remain in the LOW state.

Determining MemWrite



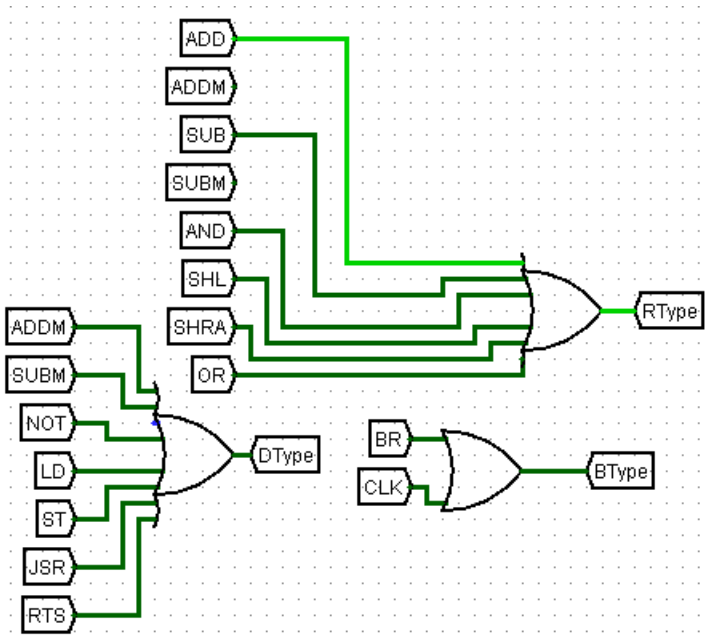
After the OpCode is decoded, the store (ST) instruction is passed as 1 input of an or gate. The other input of the OR gate has to do with exception handling. The exception handling logic will be explained to more detail in a future section. When the 2-bit step # is at either 01 OR 10 (but not both) a MemWrite will be necessary and the XOR gate's output will be HIGH. A splitter is used to extract each of the 2 bits individually.

Determining ALUOp



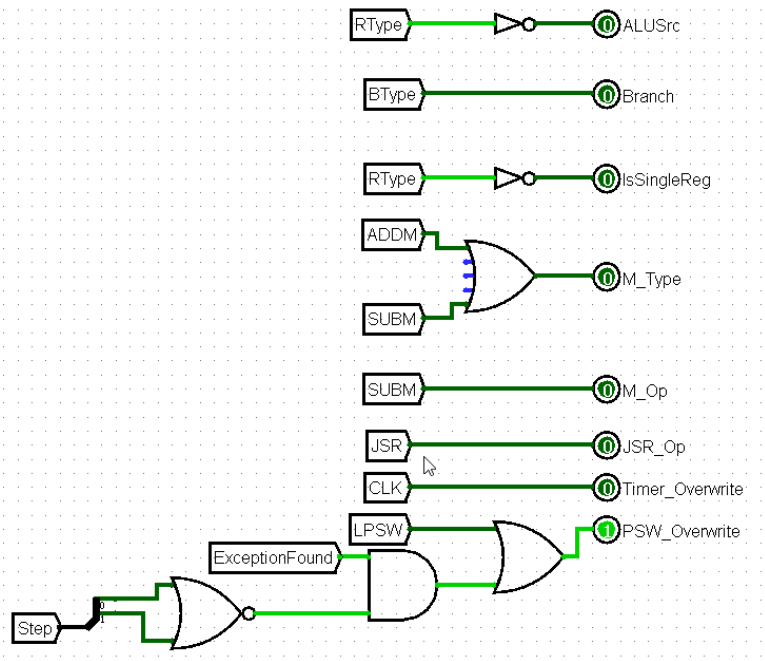
First the ALU operations required for instructions 0-15 are hardcoded with constant values. Zero corresponds to Add, one corresponds to Subtract, and so on (see ALU section for more details). The OpCode is then used as the selector for a 16x1 multiplexer. The multiplexer will then select the correct ALU operation to perform for the active instruction.

Determining Instruction Type



After the instruction has been decoded, it is passed as an input to the OR gate corresponding to its Instruction type (ADD would go to R-type, Branch would go to B-type, etc.).

Outputting Remaining Control Signals



* PSW_Overwrite output will be described in the Exception Handling section.

The 1-bit AluSrc Output determines what input to use for the ALU. When R-type is selected, Rs1 and Rs2 will be used as the ALU's inputs. If the instruction type is not R-type other inputs may be used (this is handled by a multiplexer in the outer circuit).

Outputting Remaining Control Signals (cont'd.)

The 1-bit Branch output is simply HIGH when a Branch type operation has been detected and LOW when it has not been detected.

The 1-bit IsSingleReg determines if the instruction is a 1 register operation (D-type in this processor implementation). This bit will be used to control whether or not the Register file should send Rs1 as its output or Rd for a given instruction.

The 1-bit M_Type output simply uses an OR gate to indicate if the ADDM or SUBM instruction has been detected. This control bit is used as a multiplexer control signal on the outer circuit.

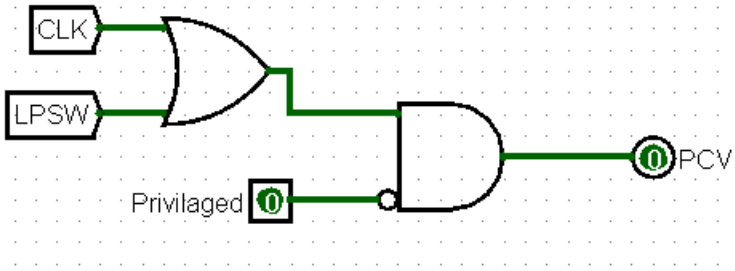
The 1-bit M_Op output determines if the SUBM instruction has been detected. This control bit is used as a multiplexer control signal on the outer circuit.

The 1-bit M_Op output determines if the JSR instruction has been detected. This control bit is used as a multiplexer control signal on the outer circuit.

The 1-bit TimerOverwrite output determines if the CLK instruction has been detected. This control bit is used as a multiplexer control signal on the outer circuit

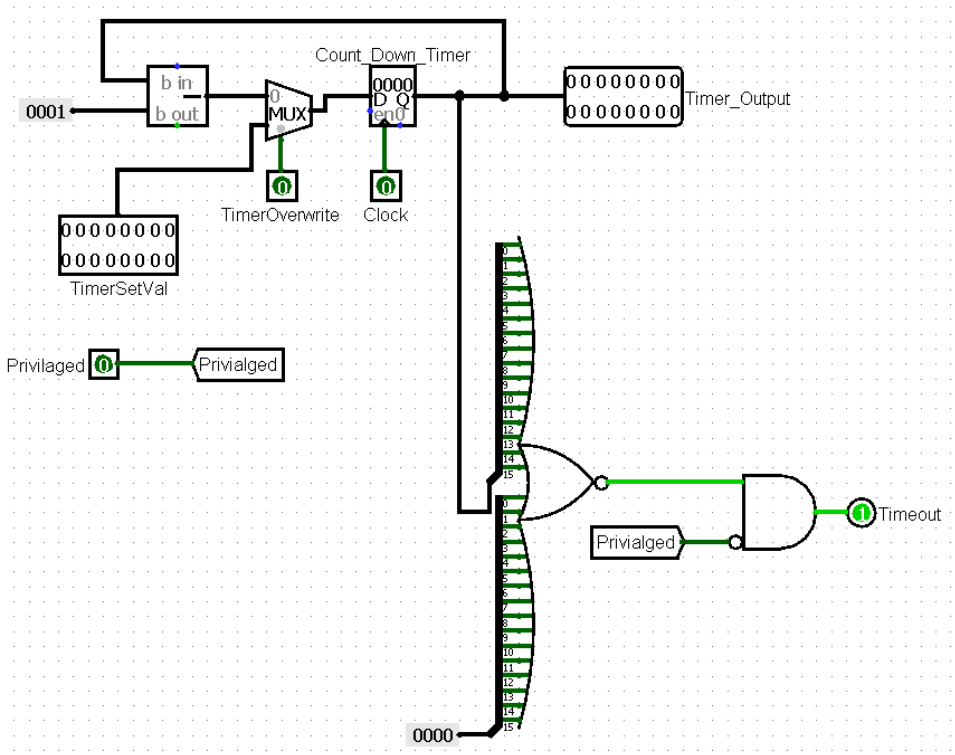
Exception Handling

Detecting a Program Check Violation (PCV)



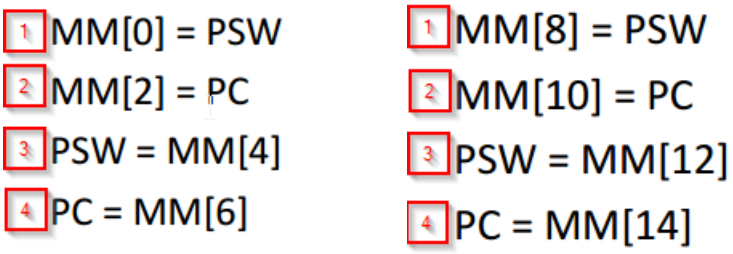
After decoding the instruction, CLK and LPSW are passed as inputs to an OR gate. The output of this OR gate is then passed as an input to an AND gate. The 1-bit Privileged input is also a negated input to the AND gated. This means that if either of the two privileged instructions (CLK and LPSW) are being executed and the PSW register's Privileged bit is LOW, flag the PCV output.

Detecting Program Timeout

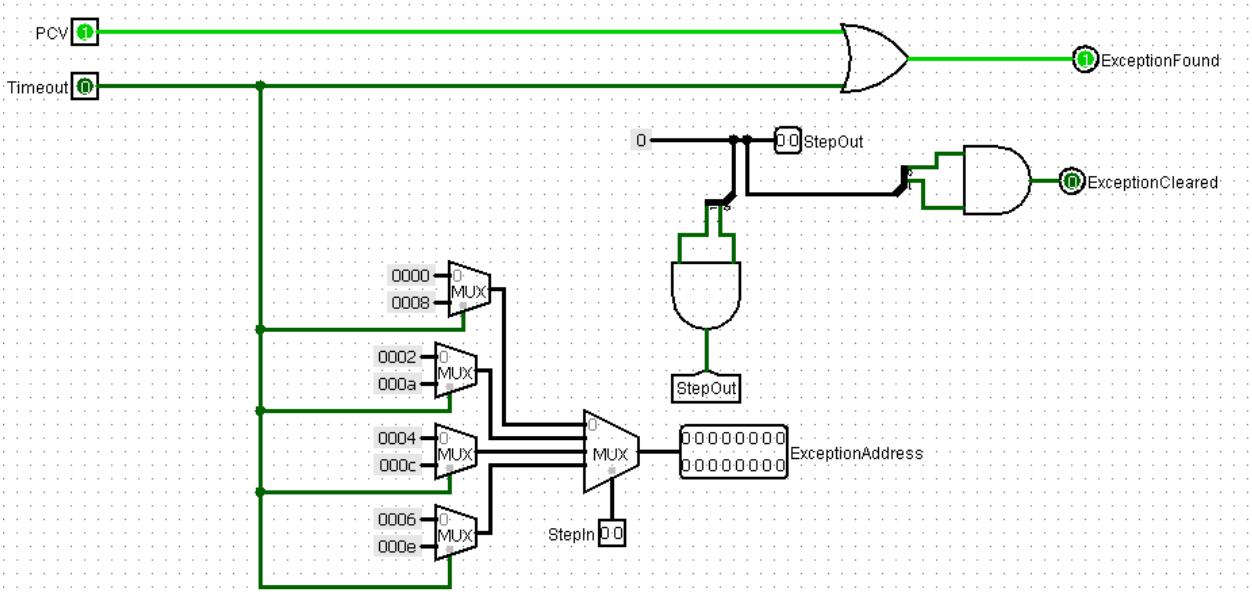


Normally, the register holding the count down timer value would be decremented by one (unless an instruction needs to overwrite it). The output of the register is fed into a NOR gate. This means that if the countdown has expired (reached zero), the NOR gate's output will be toggled HIGH. The output of the NOR gate is then used as an input to an AND gate. The PSW register's Privileged bit is sent as a negated input to the AND gate as well. This means that if the timer has reached zero and the the PSW register's Privileged bit is LOW, a 1-bit Timeout output will be toggled HIGH.

Resolving Exceptions



Both the Program Check Violation (PCV) and Timeout exceptions are handled in four steps (pictured above). The first two steps involve writing to memory. This charectoristic is taken advantage of in the control units MemWrite logic (see Determining MemWrite in the Control section). The last two steps involve reading from memory.



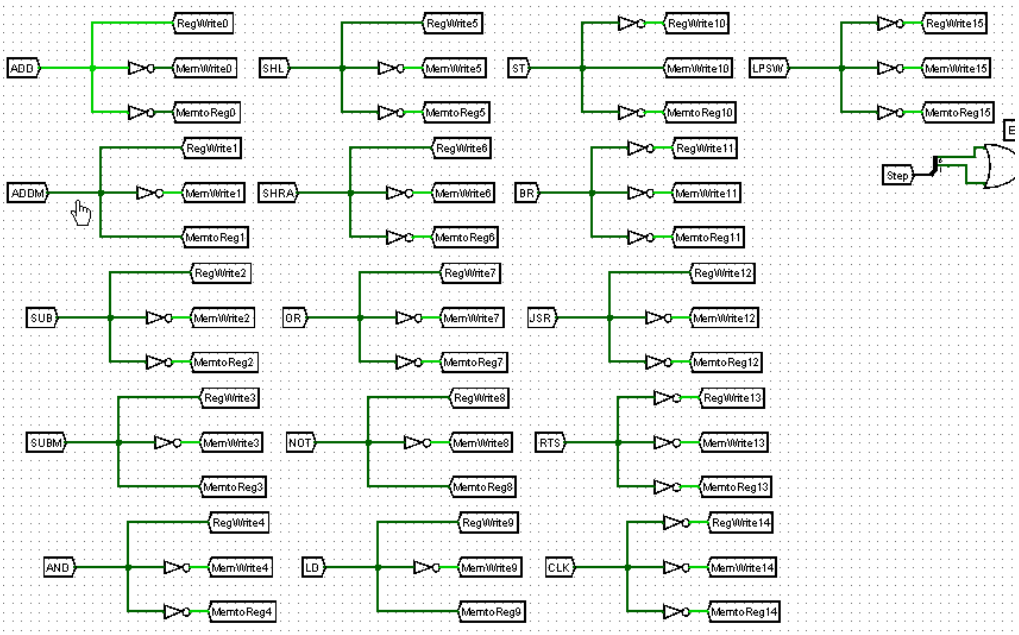
The circuit pictured above has a few key components. An OR gate is used to signal that an exception has been detected. It uses the two inputs PCV and Timeout, generated by the control unit and count down timer unit respectively, in order to toggle the 1-bit output ExceptionFound.

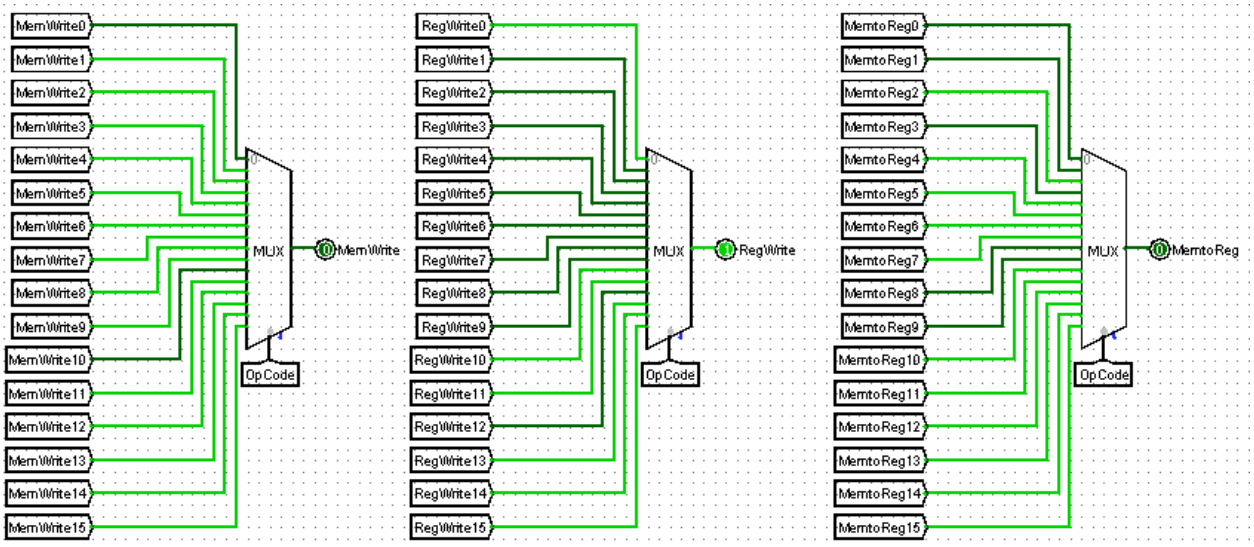
A series of multiplexers are used in order to calculate the address in main memory to be accessed (for either reading or writing depending on the step #). Addresses 0, 2, 4, and 6 are used for steps 1-4 respectively. If timeout is the type of exception detected, the multiplexers will instead choose addresses 8, A, C, and E.

The final stage is clearing the Exception. A splitter is used to extract the individual bits of the step #. Once the step # has reached step 4 (binary 11), the AND gate the splitter is attached to will be toggled HIGH indicating the exception has been cleared.

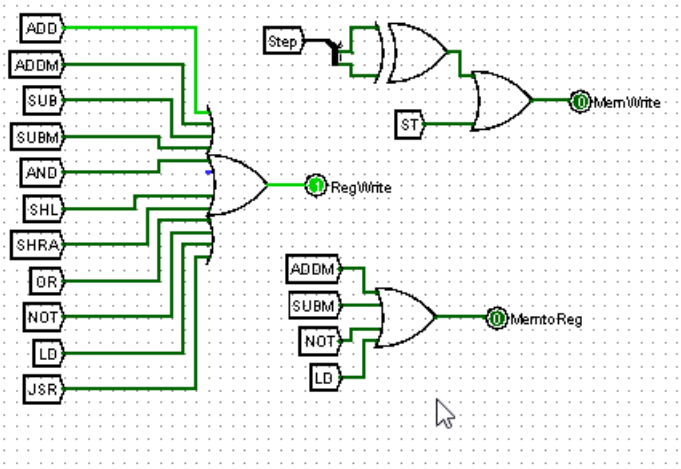
Optimization

Reworking RegWrite, MemWrite, and MemtoReg Logic

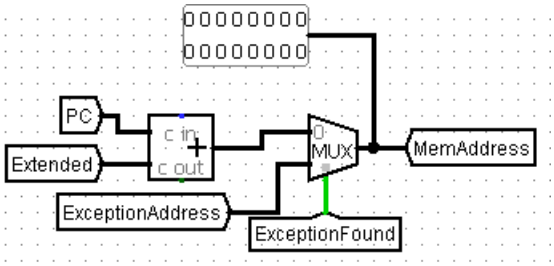




Originally MemWrite, RegWrite, and MemtoReg values were determined for each and every instruction regardless if the instruction used the control signal or not. The correct values of the control signals were then determined using multiplexers with the OpCode being their selector. This convoluted but functional method was then replaced with the much more streamlined use of OR gates pictured below.

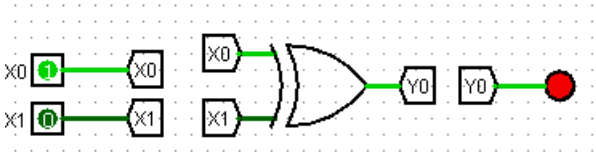


Calculating RAM Address

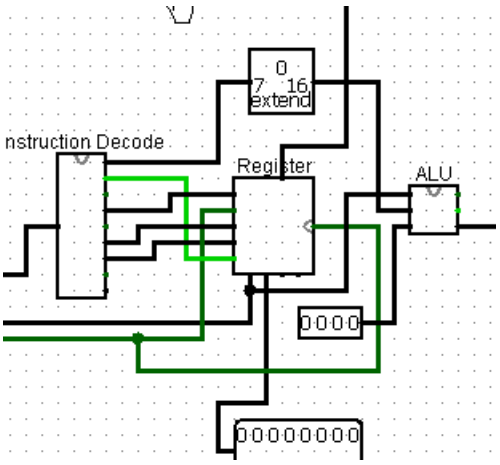


Originally, the main memory address was calculated in the ALU. As the instruction began getting more complex, it became more apparent that a better solution could be implemented. We added hardware to our circuit outside the of the ALU. This additional hardware (an adder and 2x1 multiplexer) allowed for addresses to be more easily selected for instructions with varying complexity. The address will either be the program counter + 16-bit extended offset (can be either short or long) or the address associated with the step # of the exception (see exception handling section).

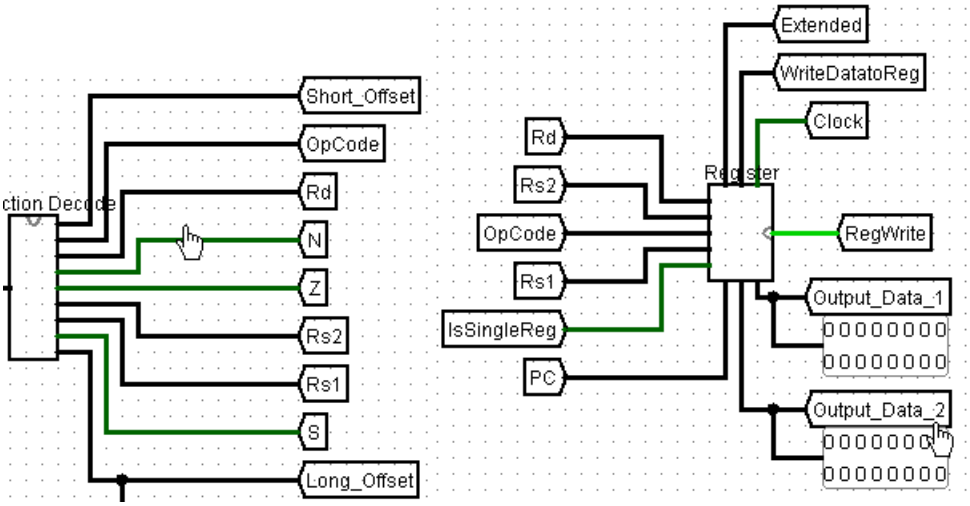
Use of The Tunnel Object in Logisim



We made the decision as a team to utilize the Tunnel Object in Logisim. The tunnels allow you to connect I/O without tracing wires from point A to point B. Although this is not necessarily an optimization that translates to the physical circuit design. The ease of understanding it brought during the design process was invaluable.



Before Use of Tunnels



After Use of Tunnels

Appendix

Final Circuit Overview