Project 3: Non-Pipelined Control Unit

Anthony Gamerman & Zuguang Liu

EECE3026 |Professor Philip Wilsey

3/13/2021

Table of Contents

1 Assignment Summary

This section summarizes the assignment as a checklist for the final delivery of the project.

A single-bus processing unit shall be designed capable of executing a specified instruction set. The design should go to the gate level.

1.1 Basic Characteristics

Basic characteristics of the machine is listed below.

- Word size of 16 bits
- Memory address and data bus size of 16 bits
- Byte addressable memory
- 64K byte main memory
- A 16-bit program status word (PSW) register illustrated in Figure 1-1, where `Z` is set when operation result is `0`, `N` is set when operation result is `0xFF`, `P` is set when machine is in privileged mode, and P is clear when machine is in user mode. Only the first 14 instructions can be executed in user mode while all 16 can be executed in privileged mode.

Figure 1-1 Program status word register spec.

- 8x16-bit general purpose register (GPR), where a constant `0` is stored at address 0, program counter (PC) is stored at address 7.
- 16-bit count-down timer
- 2's complement number representation

1.2 Instruction Specifications

There are 3 types of instruction formats (Figure 1-2) and 16 instructions (Figure 1-3) that follow them in the instruction set. `Z` and `N` bit in PSW are conditioned according to the operation result when `S` bit in instruction is set.

Figure 1-2 Instruction format.

Name	Opcode	Description
ADD	0	GPR[Rd] = GPR[Rs1] + left_shifted(GPR[Rs2],IR.Shift)
SUB	1	GPR[Rd] = GPR[Rs1] - left_shifted(GPR[Rs2],IR.Shift)
AND	2	GPR[Rd] = GPR[Rs1] **and** left_shifted(GPR[Rs2],IR.Shift)
SHL	3	GPR[Rd] = shift_left(GPR[Rs1]) by left_shifted(GPR[Rs2],IR.Shift)$_{3-0}$
SHRA	4	GPR[Rd] = shift_right(GPR[Rs1]) by left_shifted(GPR[Rs2],IR.Shift)$_{3-0}$
OR	5	GPR[Rd] = GPR[Rs1] **or** left_shifted(GPR[Rs2],IR.Shift)
NOT	6	GPR[Rd] = **not** MM[PC + Short_Offset]
LD	7	GPR[Rd] = MM[PC + Short_Offset]
ST	8	MM[PC + Short_Offset] = GPR[Rd]
BRN	9	if CC.N then PC = PC + Long_Offset
BRZ	10	if CC.Z then PC = PC + Long_Offset
BR	11	PC = PC + Long_Offset
JSR	12	GPR[Rd] = PC; PC = PC + Short_Offset
RTS	13	PC = GPR[Rd] + Short_Offset
CLK	14	Set timer to MM[PC + Long_Offset]
LPSW	15	PSW = MM[PC + Long_Offset]

Figure 1-3 Instruction set.

1.3 Exception Specifications

There are two types of exceptions in this machine.

Program check violation exception is triggered when `CLK` or `LPSW` instruction is used in user mode. The behavior of the exception is shown in Figure 1-4 (A).

Timeout exception is triggered at instruction boundary when countdown timer register is 0. Its behavior is shown in Figure 1-4 (B).

(A)

1. MM[0] = PSW
2. MM[2] = PC
3. PSW = MM[4]
4. PC = MM[6]

(B)

1. MM[8] = PSW
2. MM[10] = PC
3. PSW = MM[12]
4. PC = MM[14]

Figure 1-4 Exception behavior: program check (A) and timeout (B).

1.4 Restrictions

Some restrictions of the assignment are listed below.

- Design should be restricted to single-bus paradigm. No point-to-point connections
- Read-only memory (ROM) may be used to contain up to 8 constants. Its design can be specified descriptively.
- Main memory (MM) design can be specified descriptively and performs asynchronously
- Gate level design must only use MUX, DEMUX, encoder, decoder, flip-flop and gates.
- Detailed count-down timer design can be specified descriptively.
- Optimization is encouraged.

2 Design Overview

A top-level architecture of the solution is explained in this section. The design uses modular presentation. Each module is used extensively by the CPU to meet the design requirements and achieve optimization. The detail of each module is discussed in later sections.

An abstract overview of the CPU is shown in Figure 2-1. Data used in the instructions is pushed to a 16-bit bus as the only conveyance before being delivered to the intended locations. Apart from the components stated in the specification (MM, GPR, PSW, Timer and IR), other components include:

- Control Unit that supervises operating sequence (states) and datapath
- MDR (memory data register) and MAR (memory address register) as the interface to MM (Main Memory)
- ALU (arithmetic logic unit) to implement mathematical operations
- Y and Z register to store an operand and the result of the ALU, respectively, and
- ROM that assists trap executions

A few shortcut interfaces between these modules are designed for optimization. For example, `Rs2` as part of the IR can be connected to ALU for a reserved left-shift operation. Other examples can be found in detailed module sections. It is understood that this does not violate the single-bus paradigm since the instructions still rely on the main bus for data transfer and these connections are for control purposes and not data.

The solution was implemented to gate level then tested by simulation in Logisim-Evolution [1] for circuit generation, design, and simulation. All other diagrams such as Figure 2-1 were made in Draw.io[2]

A copy of the original project assignment can be found on Professor Wilsey's website[3]

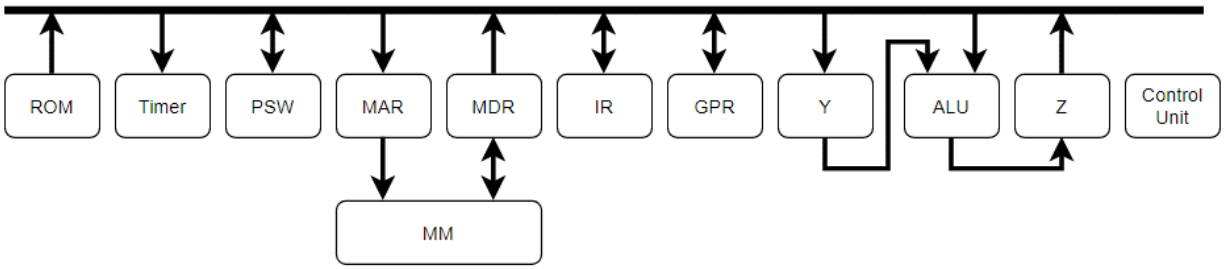

Figure 2-1 Abstract View of Control Unit Single Bus Data Path

---

[1] R. institute, *reds-heig/logisim-evolution*. 2021. Available at https://github.com/reds-heig/logisim-evolution.
[2] Available at https://app.diagrams.net/.
[3] P. Wilsey, *Project 3*. 2021. Available at https://eecs.ceas.uc.edu/~wilseypa/classes/eecs3026/project/project3.pdf

3 Control Unit Design

The solution uses a top-down approach where the control signal for operations were designed first as per the defined abstract datapath shown in Figure 2-1. This enables the machine to be freely optimized without much restriction to the hardware design. Afterwards, the components to support the design were implemented.

3.1 Control Signal

The normal operating sequence of the machine follows the fetch/decode/execute cycle unless the two trap exceptions occur. The instructions are decoded and executed in their separate branches after the same fetch sequence for 3 clock cycles, illustrated in Table 3-1 (A), where the first optimization is implemented. The CPU continues to increment the PC by one byte while waiting for the reading of the MM to finish, and additionally, keep the updated PC value in the Y register to assist later math operations done on the PC such as OP6.

During the design, it was found that the first six instructions (OP0-5) share great similarity. Not only do they share the same instruction format (line 1 in Figure 1-2), also commonly, a left-shifted `Rs2` value is used as an operand to assist different math functions done on `Rs1`, including addition, subtraction, logical-AND, left-shifting, right-shifting, and logical-OR, corresponding to the six instructions. This is the motivation of the next two optimizations. First, a separate left-shifting function for `Rs2` was reserved in the ALU, to construct a 3-bit shortcut for it to appear as the right operand, and a `ALU_Leftshift_IR.shift` control signal to enable this method was used. This removes the need to push `Rs2` onto the bus and thus reduces one step in the execution. Secondly, instead of sending six different control signals to dictate ALU functions, similarly a 4-bit connection for `Opcode` to command ALU directly, enabled by `ALU_Opcode_Task` control, such that six branched-out control sequences can combine into one. The resulting decode/execute sequence is described in Table 3-1 (B).

The control sequence of other instructions can be found in Table 3-1 (C) to (J).

The program-check exception is triggered when OP14 and OP15 are used under user mode, so it should occur after the instruction is fetched. It is found that the relevant memory address in this control sequence follows an increment-by-two pattern from 0 to 6 in each step, for which an increment-by-two function in ALU was implemented to increment the even-number-generation with `GPR[0]` since it is always 0.

Different from program-check, a timeout exception is triggered before the fetch sequence when the timer hits 0 and executes similarly except the address starts at 8. Thus, a ROM of one 16 bit register which keeps a constant number 8 was implemented, this number was used to initialize the even-number-generation instead. This merges the two exceptions into one sequence except for first step, shown in Table 3-1 (K) and is the last optimization.

(A) Common fetch for all OP codes
1) GPR[7]_Out, MAR_Bus_In, ALU_Inc_2, Z_In, READ_MM 2) Z_Out, GPR[7]_In, Y_In 3) IR_In, MDR_Bus_Out
(B) Decode/execute for OP0 to OP5
1) GPR[Rs2]_Out, ALU_In_Right, ALU_Leftshift_IR.Shift, Z_In 2) Z_Out, Y_In 3) GPR[Rs1]_Out, ALU_In_Right, ALU_In_Left, ALU_OpCode_Task, Z_In 4) Z_Out, GPR[Rd]_In
(C) Decode/execute for OP6
1) IR.Short_Off_Out, ALU_In_Right, ALU_In_Left, ALU_ADD, Z_In 2) READ_MM, Z_Out, MAR_Bus_In 3) MDR_Bus_Out, ALU_in_Right, ALU_NOT, Z_In 4) Z_Out, GPR[Rd]_In
(D) Decode/execute for OP7
1) IR.Short_Off_Out, ALU_In_Right, ALU_In_Left, ALU_ADD, Z_In 2) READ_MM, Z_Out, MAR_Bus_In 3) MDR_Bus_Out, GPR[Rd]_In
(E) Decode/execute for OP8
1) IR.Short_Off_Out, ALU_In_Right, ALU_In_Left, ALU_ADD, Z_In 2) GPR[Rd]_Out, MDR_Bus_In 3) WRITE_MM, Z_Out, MAR_Bus_In
(F) Decode/execute for OP9, OP10, and OP11
1) IR.Long_Off_Out, ALU_In_Right, ALU_In_Left, ALU_ADD, Z_In 2) Z_Out, GPR[7]_In
(G) Decode/execute for OP12
1) IR.Long_Off_Out, ALU_In_Right, ALU_In_Left, ALU_ADD, Z_In 2) Z_Out, GPR[7]_In
(H) Decode/execute for OP13
1) GPR[Rd]_Out, Y_In 2) IR.Short_Off_Out, ALU_In_Right, ALU_In_Left, ALU_ADD, Z_In
(I) Decode/execute for OP14
1) IR.Long_Off_Out, ALU_In_Right, ALU_In_Left, ALU_ADD, Z_In 2) READ_MM, Z_Out, MAR_Bus_In 3) MDR_Bus_Out, Timer_In
(J) Decode/execute for OP15
1) IR.Long_Off_Out, ALU_In_Right, ALU_In_Left, ALU_ADD, Z_In 2) READ_MM, Z_Out, MAR_Bus_In 3) MDR_Bus_Out, PSW_In
(K) Exceptions
1) GPR[0]_Out, MAR_Bus_In, Y_In //Program check violation ROM[8]_Out, MAR_Bus_In, Y_In //Timeout 2) PSW_Out, MDR_Bus_In, WRITE_MM, ALU_Inc_2, Z_In 3) Z_Out, MAR_Bus_In, Y_In 4) GPR[7]_Out, MDR_Bus_In, WRITE_MM, ALU_Inc_2, Z_In 5) Z_Out, MAR_Bus_In, Y_In, READ_MM, 6) MDR_Bus_Out, PSW_In, ALU_Inc_2, Z_In, 7) Z_Out, MAR_Bus_In, READ_MM 8) MDR_Bus_Out, PC_In

Table 3-1 Control Signal Sequence

3.2 States

With the control sequence determined, states are defined for each step (Table 3-2) and linked into a state diagram that summarizes the state transitions, shown in Figure 3-1. In addition, to assist the circuit implementation of the control unit, Boolean expressions for the control unit and state transitions are listed in Table 3-3 and Table 3-4, respectively.

State	Index	Bit Encoding	Control Signals
RST	S0	00000	Reset
F1	S1	00001	PC_Out, MAR_Bus_In, ALU_Inc_2, Z_In, READ_MM
F2	S2	00010	Z_Out, PC_In, Y_In
F3	S3	00011	IR_In, MDR_Out
FMT1A	S4	00100	GPR[Rs2]_Out, ALU_Inc_2, ALU_Leftshift_IR.shift, Z_In
FMT1B	S5	00101	Z_Out, Y_In
FMT1C	S6	00110	GPR[Rs1]_Out, ALU_In_Right, ALU_In_Left, ALU_OpCode_Task, Z_In
FMT1D	S7	00111	Z_Out, GPR[Rd]_In
FMT2A	S8	01000	IR.Short_Off_Out, ALU_In_Right, ALU_In_Left, ALU_ADD, Z_In
FMT2B	S9	01001	READ_MM, Z_Out, MAR_Bus_In
FMT3A	S10	01010	IR.Long_Off_Out, ALU_In_Right, ALU_In_Left, ALU_ADD, Z_In
FMT3B	S11	01011	Z_Out, PC_In
OP6A	S12	01100	MDR_Bus_Out, ALU_In_Right, ALU_NOT, Z_In
OP7A	S13	01101	MDR_Bus_Out, GPR[Rd]_In
OP8A	S14	01110	GPR[Rd]_Out, MDR_Bus_In
OP8B	S15	01111	WRITE_MM, Z_Out, MAR_Bus_In
OP12A	S16	10000	GPR[Rd]_In, PC_Out
OP13A	S17	10001	GPR[Rd]_Out, Y_In
OP14A	S18	10010	MDR_Bus_Out, Timer_In
OP15A	S19	10011	MDR_Out, PSW_In
TRP1A	S20	10100	GPR[0]_Out, MAR_In, Y_In
TRP1B	S21	10101	ROM[8]_Out, MAR_In, Y_In
TRP2	S22	10110	PSW_Out, MDR_Bus_In, WRITE_MM, ALU_Inc_2, Z_In
TRP3	S23	10111	Z_Out, MAR_Bus_In, Y_In
TRP4	S24	11000	PC_Out, MDR_In, WRITE_MM, ALU_Inc_2, Z_In
TRP5	S25	11001	Z_Out, MAR_Bus_In, Y_In, READ_MM,
TRP6	S26	11010	MDR_Bus_Out, PSW_In, ALU_Inc_2, Z_In,
TRP7	S27	11011	Z_Out, MAR_Bus_In, READ_MM
TRP8	S28	11100	MDR_Bus_Out, PC_In

Table 3-2 State Definition.

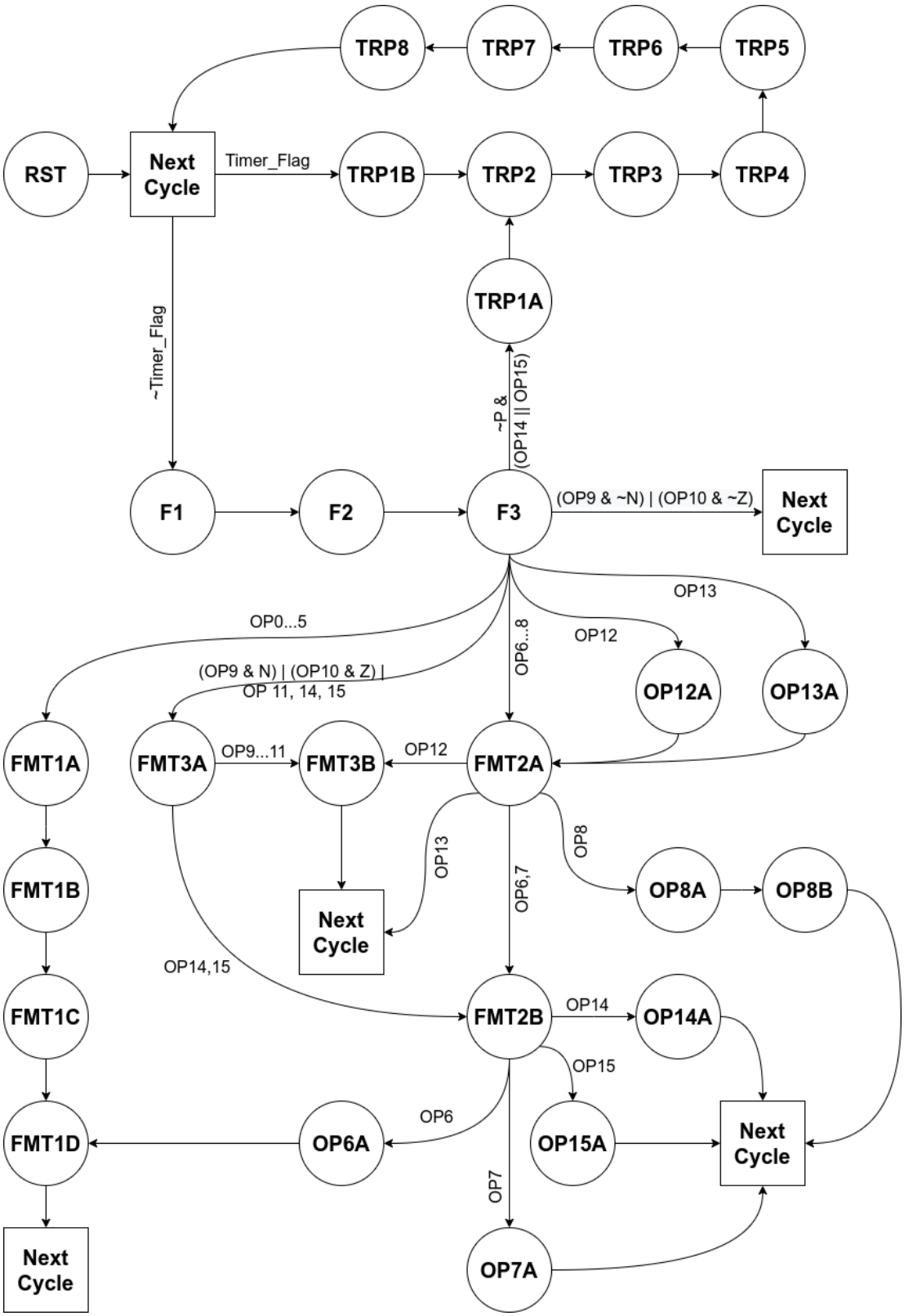

Figure 3-1 State diagram. Logic expressions loosely follow C-language style. RST is the initial state the machine starts at.
Note: Next Cycle is NOT A STATE, but a node to help the visual that signifies the next fetch/decode/execute cycle.

Contol Signal Ouput	Logical Expressions by States
IR_In	F3
IR.Short_Off_Out	FMT2A
IR.Long_Off_Out	FMT3A
PC_In	F2 | FMT3B | TRP8
PC_Out	F1 | TRP4 | OP12A
GPR[Rd]_In	FMT1D | OP7A | OP12A
GPR[Rd]_Out	OP8A | OP13A
GPR[Rs2]_Out	FMT1A
GPR[Rs1]_Out	FMT1C
GPR[0]_Out	TRP1A
MAR_Bus_In	F1 | FMT2B | OP8B | TRP1A | TRP1B | TRP3 | TRP5 | TRP7
MDR_Bus_Out	F3 | OP6A | OP7A | OP14A | OP15A | TRP6 | TRP8
MDR_Bus_In	OP8A | TRP2 | TRP4
READ_MM	F1 | FMT2B | TRP5 | TRP7
WRITE_MM	OP8B | TRP2 | TRP4
ALU_In_Right	FMT1A | FMT1C | FMT2A | FMT3A | OP6A
ALU_In_Left	FMT1C | FMT2A | FMT3A
ALU_OpCode_Task	FMT1C
ALU_Leftshift_IR.shift	FMT1A
ALU_NOT	OP6A
ALU_ADD	FMT2A | FMT3A
ALU_Inc_2	F1 | TRP2 | TRP4 | TRP6
Z_In	F1 | FMT1A | FMT1C | FMT2A | FMT3A | OP6A | TRP2 | TRP4 | TRP6
Z_Out	F2 | FMT1B | FMT1D | FMT2B | FMT3B | OP8B | TRP3 | TRP5 | TRP7
Y_In	F2 | FMT1B | OP13A | TRP1A | TRP1B | TRP3 | TRP5
ROM[8]_Out	TRP1B
Timer_In	OP14A
PSW_In	OP15A | TRP6
PSW_Out	TRP2

Table 3-3 Boolean expressions from states to control signal, in a style that loosely follows C-language

Next State	Go-to Conditions
RST	(Initial state at reset)
FCH1	Timer_Flag & Next_Cycle
FCH2	FCH1
FCH3	FCH2
FMT1A	F3 & (OP0 | OP1 | OP2 | OP3 | OP4 | OP5)
FMT1B	FMT1A
FMT1C	FMT1B
FMT1D	FMT1C | OP6A
FMT2A	F3 & (OP6 | OP7 | OP8) | OP12A | OP13A
FMT2B	FMT2A & (OP6 | OP7) | FMT3A & (OP14 | OP15)
FMT3A	FCH3 & ((OP9 & PSW.N) | (OP10 & PSW.Z) | OP11 | OP14 | OP15)
FMT3B	FMT3A & (OP9 | OP10 | OP11) | FMT2A & OP12
OP6A	FMT2B & OP6
OP7A	FMT2B & OP7
OP8A	FMT2A & OP8
OP8B	OP8A
OP12A	F3 & OP12
OP13A	F3 & OP13
OP14A	FMT2B & OP14
OP15A	FMT2B & OP15
TRP1A	F3 & ~PSW.P & (OP14 | OP15)
TRP1B	Timer_Flag & Next_Cycle
TRP2	TRP1A | TRP1B
TRP3	TRP2
TRP4	TRP3
TRP5	TRP4
TRP6	TRP5
TRP7	TRP6
TRP8	TRP7

```
where Next_Cycle = (RST | FMT1D | FMT3B | OP8B | OP14A | OP15A | TRP8 |
                    FMT2A & OP13 | F3 & OP9 & ~PSW.N | F3 & OP10 & ~PSW.Z)
```

Table 3-4 Boolean expressions for state transition, in a style that loosely follows C-language.

3.3 Control Unit Circuit Implementation

The control unit circuit is designed as a Moore machine whose input are the 16 bit `Opcode` and control flags including `PSW.P`, `PSW.N`, `PSW.P` and `Timer_Flag`, and whose output is various control signals. The sequential unit is broken down into 3 modules: `State Controller`, `State Switcher` and `State Decoder`.

The `State Decoder` module uses 5 flip-flops to implement the memory for the 29 states, then decode the current state into a one-hot form. This 29-bit-wide value goes into `State Controller` that sets up various control signals shown in Table 3-3, as well as `State Switcher` that calculates the next state in Table 3-4. The next state will go into the `State Decoder` to finish the state transitions using the flip flops. `State Switcher` also uses the Opcodes from the IR as an input to calculate the next state. The circuit on the top level is shown in Figure 3-2 and the module-level view is demonstrated in Figure 3-3 This design uses a discrete approach to reduce the complexity on the Boolean expressions. Practically, it may cause problems such as wire crosstalk or propagation delay, so gate-level optimization could be used, but in a proof of concept, such issues are not in the main scope of the project.

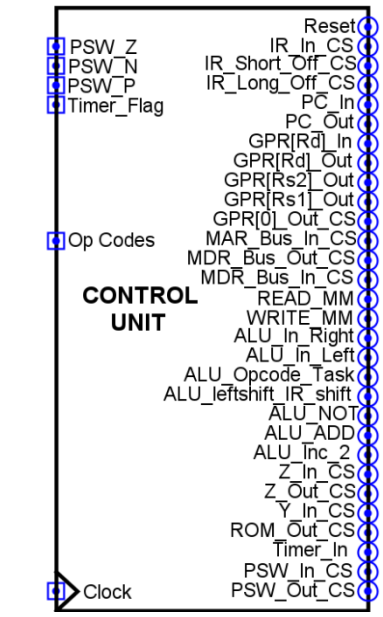

Figure 3-2 Control Unit Abstract View

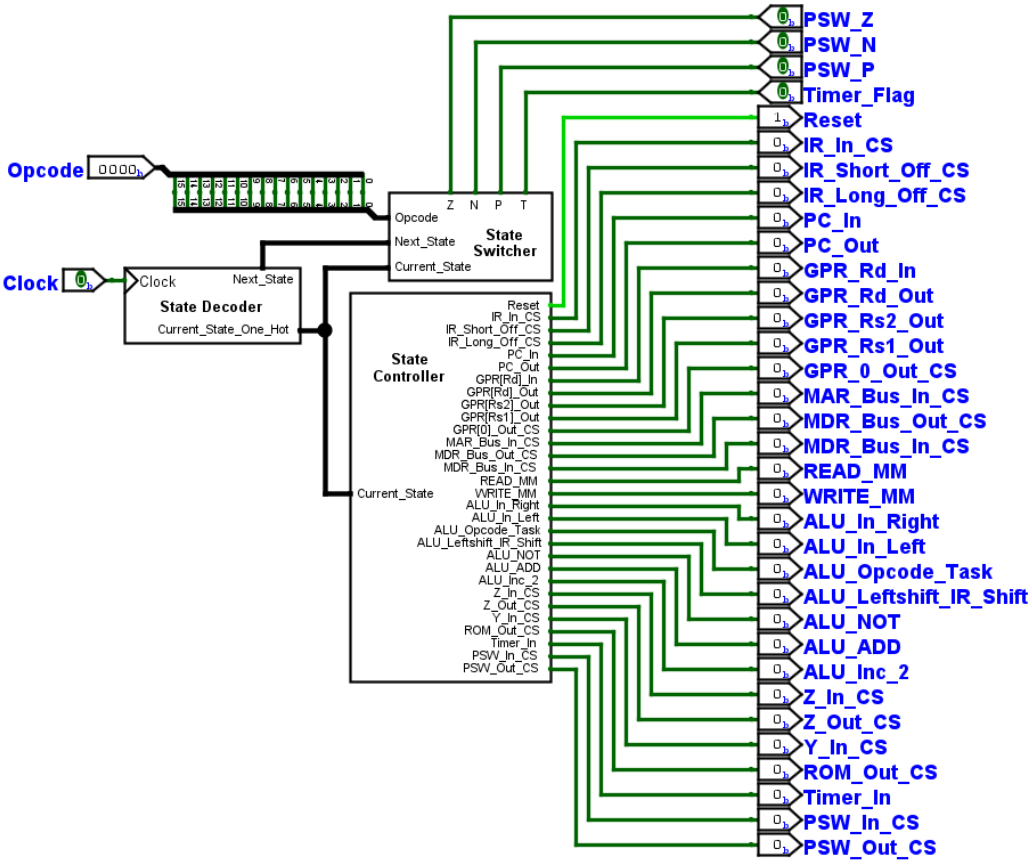

Figure 3-3 Control Unit Data Block Component View

3.3.1 State Decoder

As previously stated, the state decoder determines the following state of the machine. The abstract data block can be seen in Figure 3-3 on the previous page. Figure 3-4 shows the gate level view of the `State Decoder`. It takes in the 5 bit Next_State value from the `State Switcher` discussed in the following section and puts them into the 5 D-flip flops at the rising edge of the clock signal. With the following clock signal, it outputs the values of the flip flops into a decoder which will only turn on one state at a time which operates, hence the term one hot since only one bit of the decoder is on at a time.

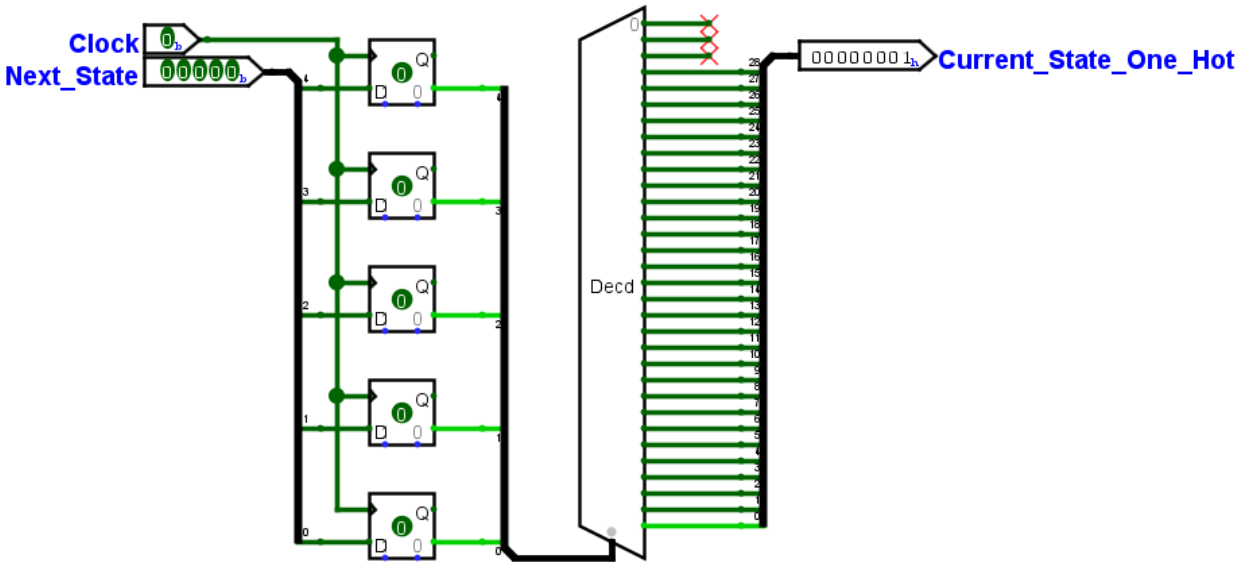

Figure 3-4 State Decoder Gate Level View

3.3.2 State Switcher

The abstract view of the State Switcher is shown on the previous page in Figure 3-3 The `State Switcher` gate level view is shown on the next page in Figure 3-5. Using the Boolean equations of the states in Table 3-4 and the component Boolean equations of the states in Table 3-2, the `State Switcher` takes the outputs of the current state, certain single bits from the rest of the circuit and combines them with the op codes from the IR into an encoder to assemble a 5 bit value of the following state. The Timer_Flag comes from the timer when it reaches 0. The PSW.P, PSW.N, and PSW.Z bits come from the PSW.

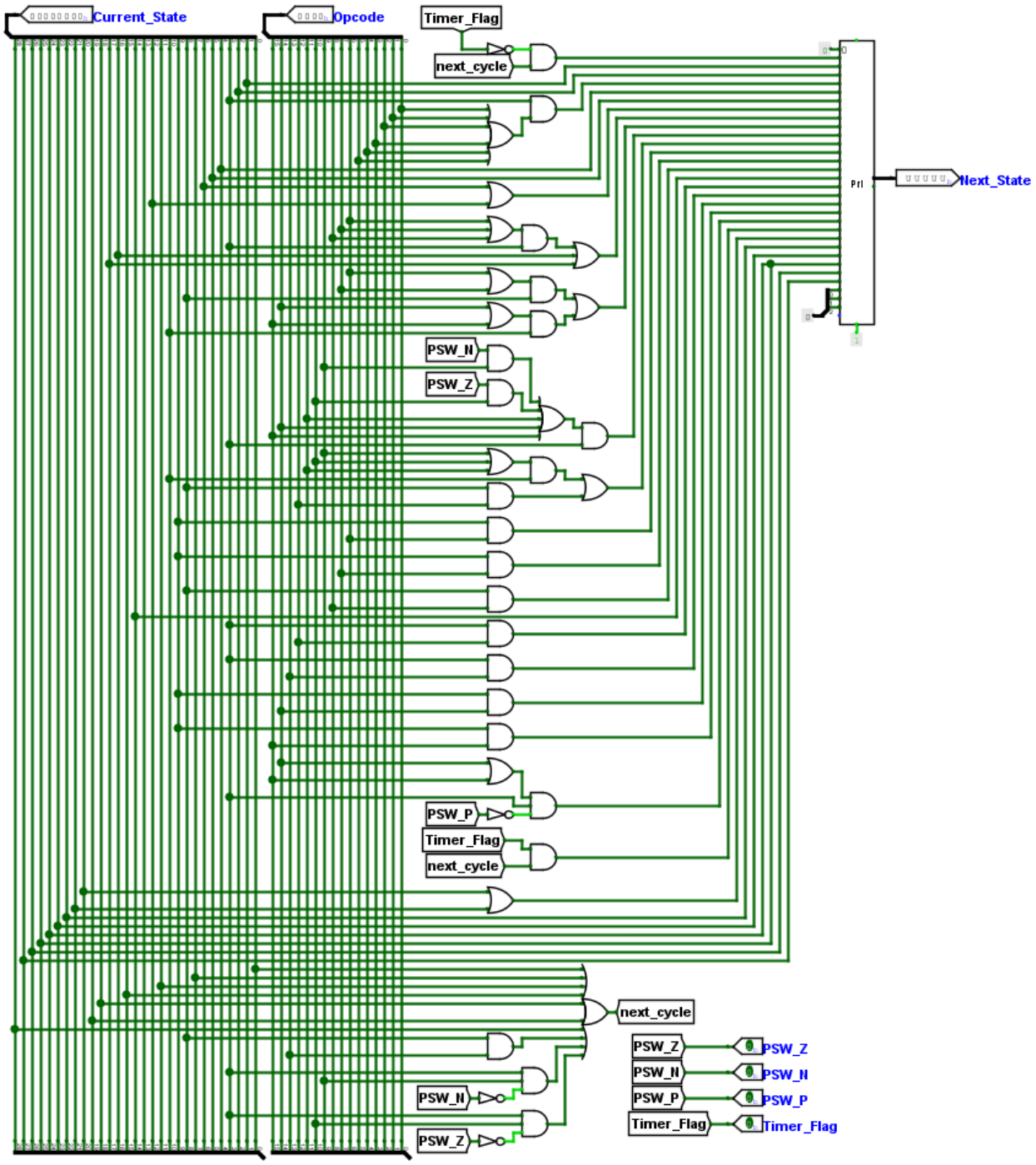

Figure 3-5 State Switcher Gate Level View

3.3.3 State Controller

The `State Controller` abstract view is shown in Figure 3-3 on page 13. The `State Controller` gate level view is shown on the next page in Figure 3-6. It takes the current state from the state decoder and breaks out the 29 states and creates the actual control signals that go to the rest of the machine based on the Boolean equations in Table 3-3 and Table 3-2 to form the sequence and combination of control signals shown in Table 3-1 that are explained in detail in Table 3-5.

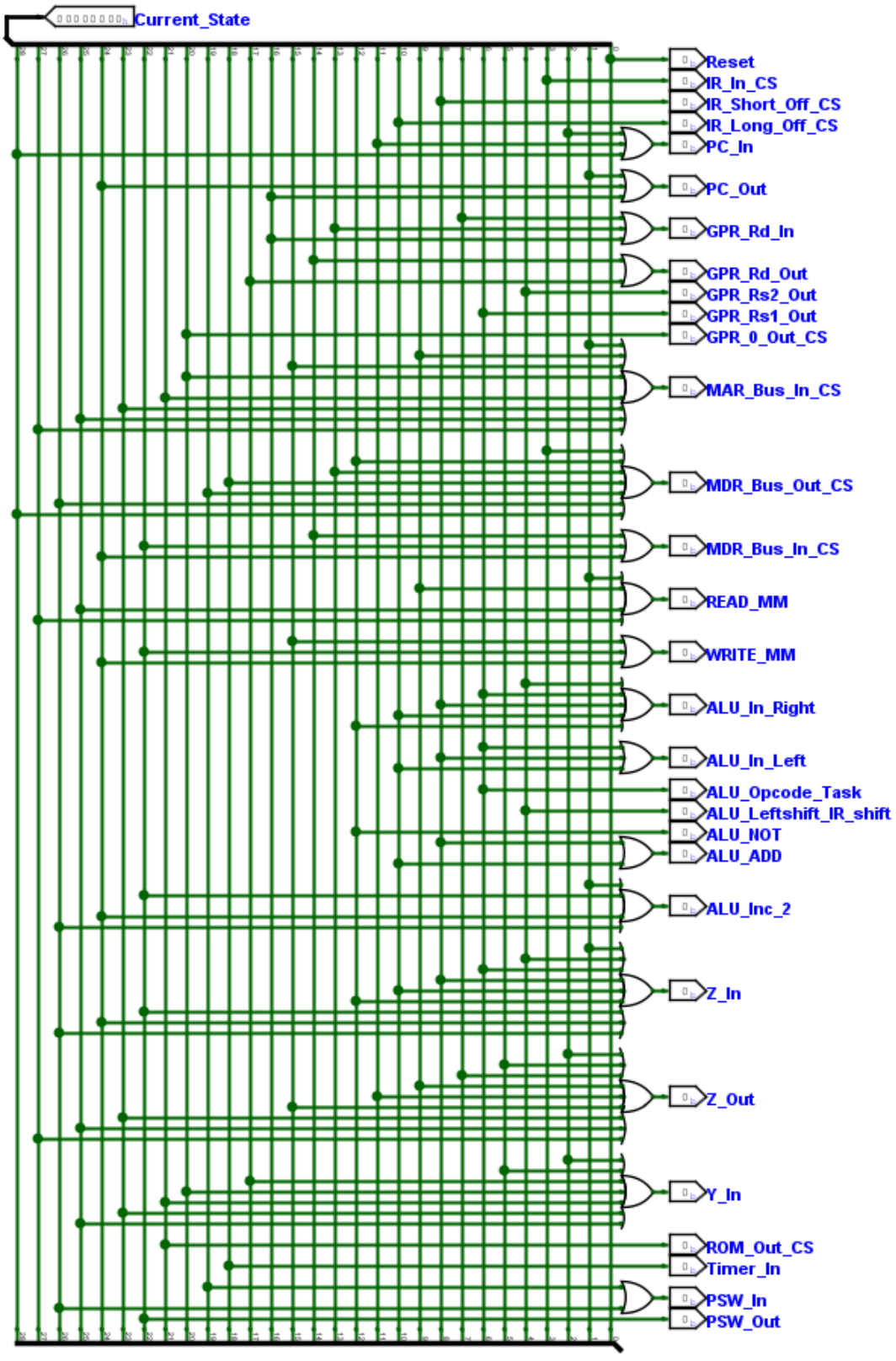

Figure 3-6 State Controller Gate Level View

Control Signal (CS)	Function	Destination
Reset	Reset	All except ALU
IR_In_CS	Load data from the bus to the IR	IR
IR_Short_Off_CS	Load the short offset from the IR to the bus	IR
IR_Long_Off_CS	Load the long offset from the IR to the bus	IR
PC_In	Load the PC into GPR[7] from the bus	GPR
PC_Out	Load the PC from GPR[7] to the bus	GPR
GPR_Rd_In	Load the GPR based on IR[Rd]	GPR
GPR_Rd_Out	Load the bus from the GPR based on IR[Rd]	GPR
GPR_Rs2_Out	Load the bus from the GPR based on IR[Rs2]	GPR
GPR_Rs1_Out	Load the bus from the GPR based on IR[Rs1]	GPR
GPR_0_Out_CS	Load the bus from the GPR with the static value “0”	GPR
MAR_Bus_In_CS	Load the MAR from the bus	MAR
MDR_Bus_Out_CS	Load the bus from the MDR	MDR
MDR_Bus_In_CS	Load the MDR from the bus	MDR
READ_MM	Load the MDR with the memory fata from the MM	MM/MDR
WRITE_MM	Load the MM with the memory data from the MDR	MAR
ALU_In_Right	Load the ALU from the bus	ALU
ALU_In_Left	Load the ALU from the Y Register	ALU
ALU_Opcode_Task	Operate the ALU based on the OP code from the IR	ALU
ALU_Leftshit_IR_Shift	Perform a 0-1 bit shift in the ALU	ALU
ALU_NOT	Perform NOT in the ALU	ALU
ALU_ADD	Perform AND in the ALU	ALU
ALU_Inc_2	Increment the value from the bus by 2	ALU
Z_In	Load the Z register from the ALU	Z Register
Z_Out	Load the bus from the Z register	Z Register
Y_In	Load the Y register from the bus	Y Register
ROM_Out_CS	Load the bus from the ROM with the static value “8”	ROM
Timer_In	Load the timer from the bus	Timer
PSW_In	Load the PSW from the bus	PSW
PSW_Out	Load the bus from the PSW	PSW

Table 3-5 Control Signal Descriptions and Destinations

NOTE: There are some minor differences between some control signals and their similarly named components in the destinations. Notably, the following:
`ALU_In_Right` is defined as `Bus_In_CS`
`ALU_In_Left` is defined as `Y_Out_CS`
`ALU_Leftshift_IR_Shift` is defined as `IR_Shift_0_CS`
`SET_CC` is controlled by `IR.S`, or `IR_S` in the report

Due to naming limitations and conflicts in Logisim Evolution, some control signals had to be written with underscores at the gate level instead of brackets while their control signals in the abstract views are labeled with brackets, however they are the same. As an example, the control signal `GPR[Rd]_In` is labeled as `GPR_Rd_In` at the gate level.

Keep this in mind through the duration of the report.

4 Component Circuit Implementation

4.1 Single Bus Data Path

Figure 4-1 shows the overall circuit data path from the single 16-bit bus. Each of the individual blocks are various components of the design and will be investigated further in this report. Each component has a connection to the orange central single data bus with the exception of the IR which additionally brings controls to the GPR, ALU and Control Unit. Most blocks also have clocks and reset signals that aren't shown for legibility of the diagram. All the control signals go to various blocks of the diagram which aren't shown in the figure to prevent clutter and are explained in Table 3-5, but all signals with the same name are connected for functionality. The exceptions to this are explained on the previous page.

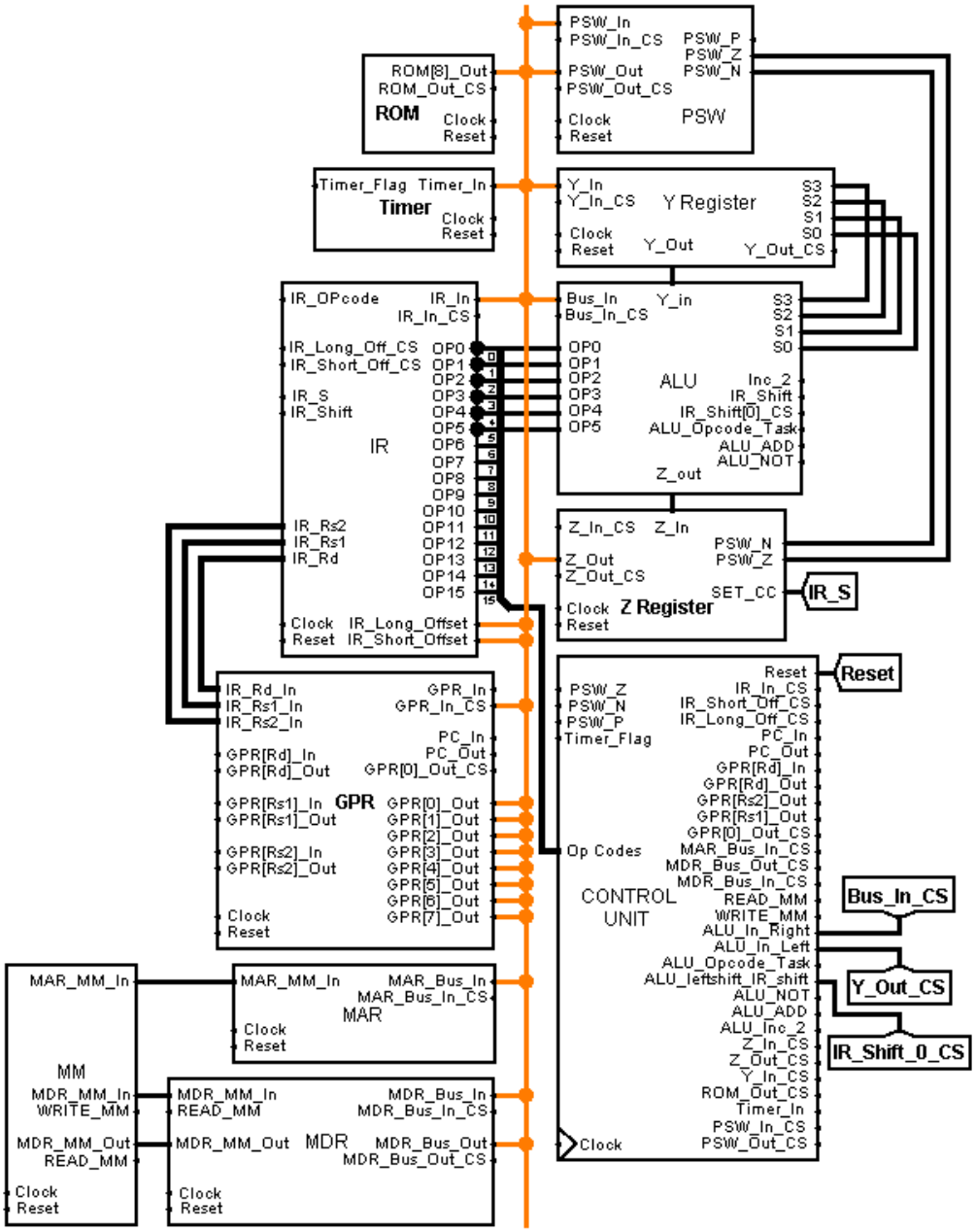

Figure 4-1 Single Bus Data Path Block Diagram

4.2 16-Bit Register

The Figure 4-2 shows an abstract view of a 16-bit register. The circles represent the outputs of the block and the squares represent the inputs. The 16-bit register is comprised of 16 D-flip flops, these are ordered in the abstract diagram from 0 going in order incrementing by 1 to 15. D-flip flops also require a reset signal to be initialized (abbreviated as `Rst`) and a clock signal to send the following value into the flip flop (abbreviated as `Clk`). This abstract view of the 16-bit register will be utilized extensively through the rest of the project and report.

Figure 4-2 16-Bit Register Abstract Block View

Figure 4-3 below shows the more in-depth view of the 16-bit register. Each of the individual bits are stored using a D-flip flop. The data comes into the `D[0-15]` bits and exits through `Q[0-15]`. Each flip flop will store the next value on the rising edge of the clock signal. Each flip flop is also connected to a Reset signal which will reset all the currently stored values in the flip flops to 0, this signal is also required to be set high to initialize all the flip flops.

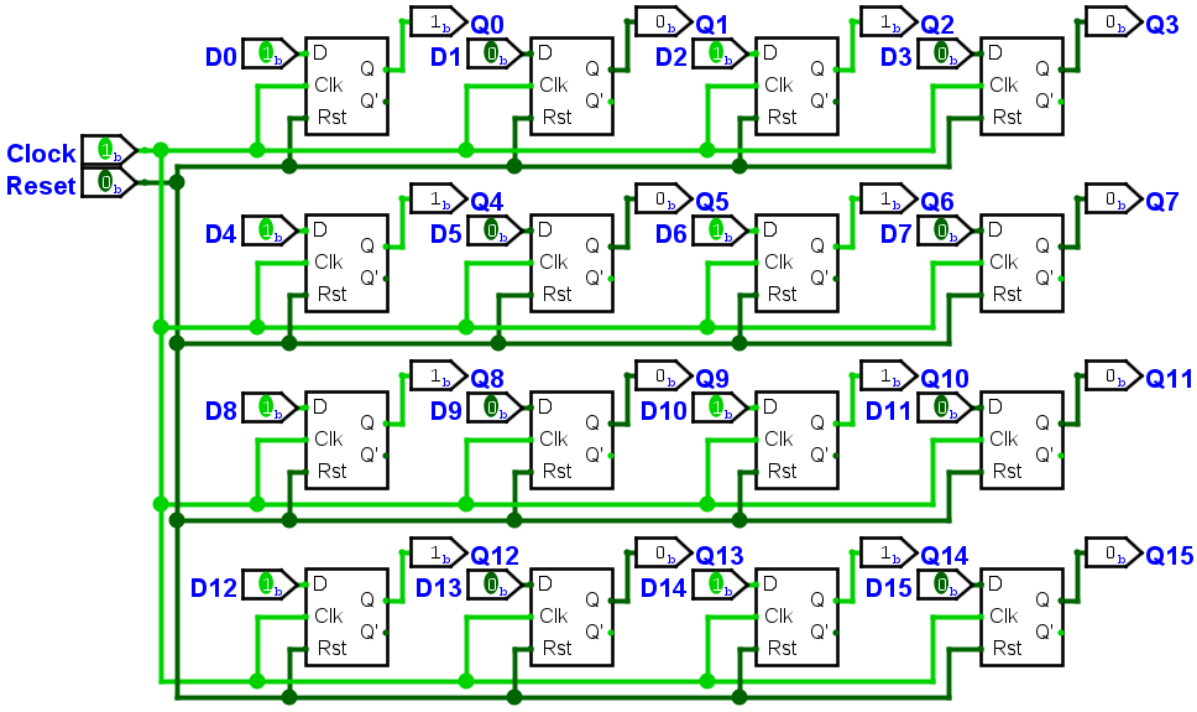

Figure 4-3 Flip-Flop Implementation of the 16-Bit Register

4.3 Instruction Register (IR)

The Figure 4-4 shows the abstract view of the Instruction Register (IR) inputs and outputs that is shown in Figure 4-1. The circles represent the outputs of the block and the squares represent the inputs.

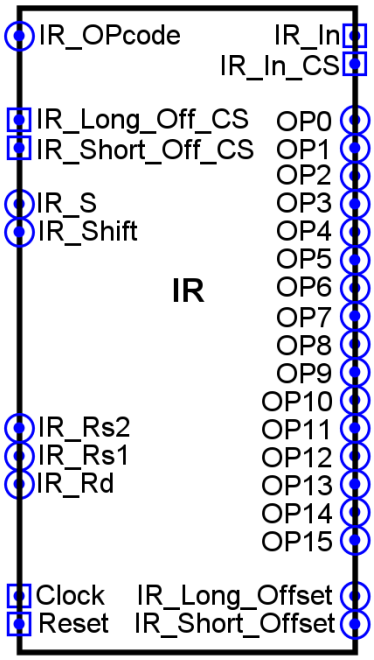

Figure 4-4 Instruction Register (IR) Abstract Data Block

The IR is comprised of a 16-bit register that will store the actual instruction operand. From the register, the 16-bit operand will determine the instruction executed by the machine. The problem states 3 different 16-bit instruction formats that result in a variety of different outputs from the IR. The outputs and inputs of the block are described below in Table 4-1. The gate level view of the IR can be seen on the next page in Figure 4-5.

IR_In	Data in from the bus to the IR
IR_In_CS	The control signal to turn on a tri-state buffer and allows data from the bus to the IR
IR_OPcode	The first four-bit Opcode in all 3 instruction formats
OP0 to Op15	The decoded OP codes from the 4 bit IR_OPcode
IR_S	1 bit signal to set the condition code in the PSW
IR_Shift	2 bit signal to determine shifting amount (0 to 3) in the ALU
IR_Rd	3 bit signal to determine GPR function
IR_Rs1	3 bit signal to determine GPR function
IR_Rs2	3 bit signal to determine GPR function
IR_Long_Offset	12 bit data in the IR, sign extended
IR_Long_Off_CS	The control signal to turn on a tri-state buffer and allows data from the long offset to the bus
IR_Short_Offset	9 bit data in the IR, sign extended
IR_Short_Off_CS	The control signal to turn on a tri-state buffer and allows data from the short offset to the bus

Table 4-1 Instruction Register Input/Output Descriptions

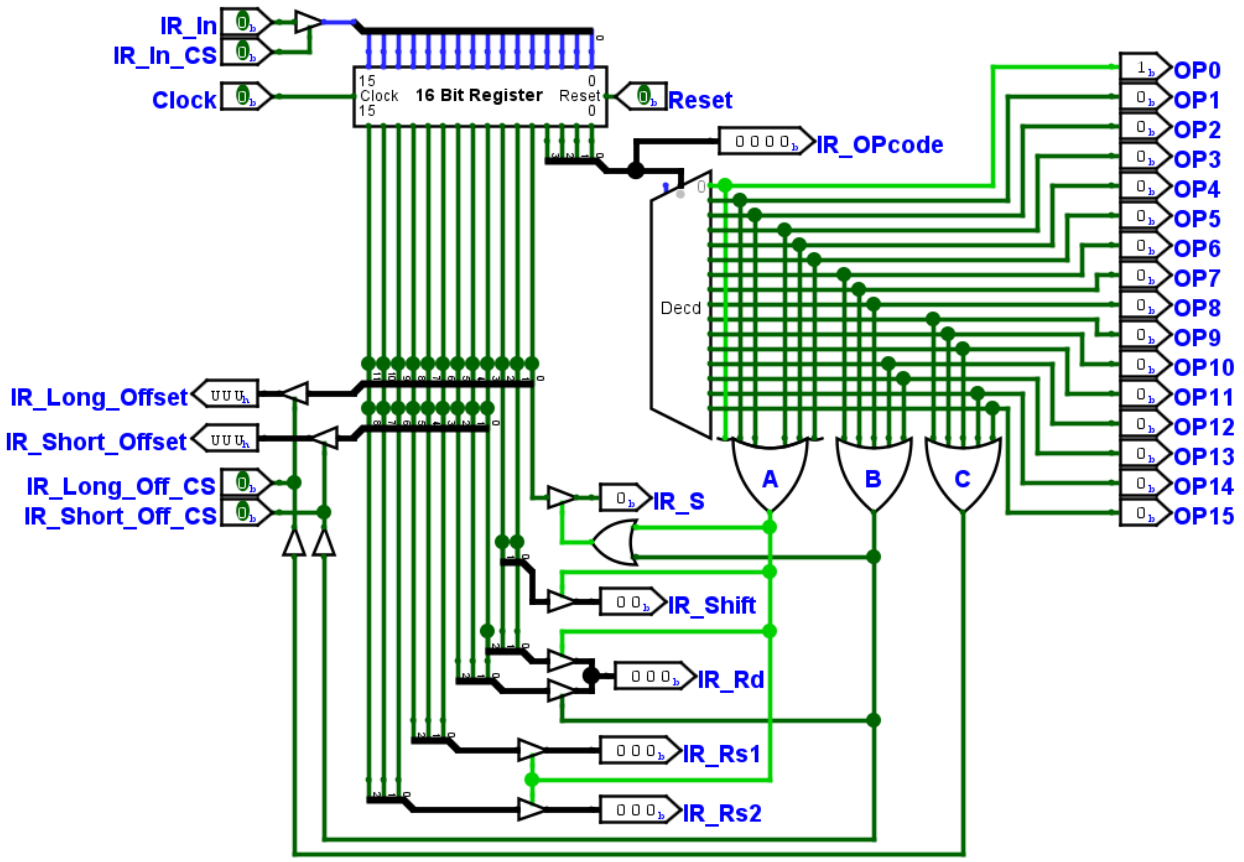

Figure 4-5 Gate Level View of the Instruction Register

In the IR, the 16-bit data comes into the register from `IR_In` when `IR_In_CS` is HIGH and opens the tri-state buffer. The first four bits go into the select bits of the decoder to decode the bits from their binary values into their decimal OP code equivalents from OP0 (`0b0000`) to OP15 (`0b1111`). From these decoded signals, three OR gates take the signals to determine the control of the output of the IR from the 3 instruction formats shown in Figure 1-2. From the control signals, it was seen that:

The 6 OP codes (0,1,2,3,4,5) can be used with OR gate A to open the tri-state buffers and output the first instruction format on the bus and to the various other blocks of the machine.

The 5 OP codes (6,7,8,12,13) can be used with OR gate B to open the tri-state buffers and output the second instruction format on the bus and to the various other blocks of the machine.

The 5 OP codes (9,10,11,14,15) can be used with OR gate C to open the tri-state buffers and output the first instruction format on the bus and to the various other blocks of the machine.

Since `IR_S` is used by instruction formats A and B an additional OR gate was used to control its tri-state buffer. Additionally, the IR long and short offset can be output on the bus independent of the ABC OR gate controls with their control signals. Buffers act as diodes underneath the control signals to ensure the proper flow of data. Finally, since the `IR_Long_Offset`/`IR_Short_Offset` have different word lengths from the 16 bit bus, they are sign extended to preserve their sign and value on the bus.

4.4 Program Status Word (PSW)

Figure 4-6 shows the abstract view of the Program Status Word (PSW) inputs and outputs that is shown in Figure 4-1. The circles represent the outputs of the block and the squares represent the inputs. The outputs and inputs of the block are described below in Table 4-2. The gate level view of the PSW can be seen on the below in Figure 4-7.

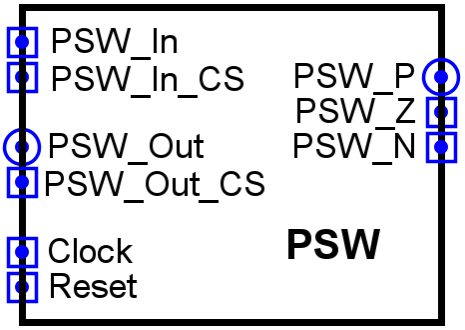

Figure 4-6 Program Status Word (PSW) Abstract Data Block

PSW_In	Data in from the bus to the PSW
PSW_In _CS	The control signal to turn on a tri-state buffer and allows data from the bus to the PSW
PSW_Out	Data out to the bus
PSW_Out_CS	The control signal to turn on a tri-state buffer and allows data from the PSW to the bus
PSW_P	Privileged mode bit, denotes execution of the machine in privileged (1) or user mode (0)
PSW_Z	Condition code bit Z
PSW_N	Condition code bit N
Clock	16-bit register clock signal
Reset	16-bit register reset signal

Table 4-2 Program Status Word Input/Output Descriptions

The PSW is a 16 bit register where the first three low order bits are the condition codes N, Z, and P from bit 0 to bit 2, respectively. The remaining bits of the PSW are not addressed in this project. The Z and N bits come from the Z register and are only set when `Set_CC` in the Z register is active. Privileged mode from `PSW_P` was further discussed earlier in section 1.1

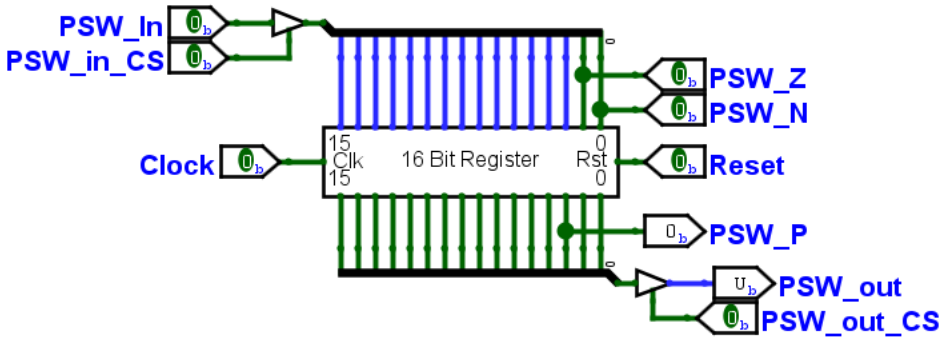

Figure 4-7 Gate Level View of the Program Status Word

4.5 General Purpose Register (GPR)

The Figure 4-8 shows the abstract view of the General Purpose Register (GPR) inputs and outputs that is shown in Figure 4-1. The circles represent the outputs of the block and the squares represent the inputs. The outputs and inputs of the block are described below in Table 4-3. The gate level view of the GPR can be seen on the following page in Figure 4-9.

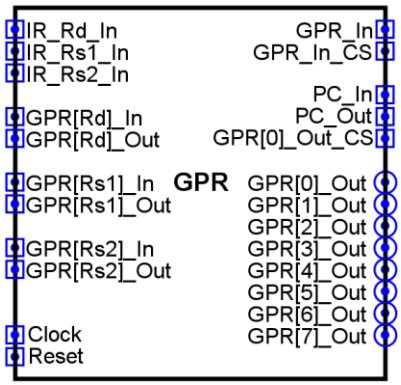

Figure 4-8 General Purpose Register (GPR) Abstract Data Block

GPR_In	Data in from the bus to the GPR
GPR_In_CS	The control signal to turn on a tri-state buffer and allows data from the bus to the GPR
GPR[0]_Out -> GPR[7]_Out	The data outputs to the bus of the 8 16-bit registers in the GPR
IR_Rd_In	3 bit signal from the IR to determine GPR function
IR_Rs1_In	3 bit signal from the IR to determine GPR function
IR_Rs2_In	3 bit signal from the IR to determine GPR function
GPR[Rd]_In	The control signal to use the Rd operand to input data into the GPR from the bus
GPR[Rd]_Out	The control signal to use the Rd operand to output data from the GPR to the bus
GPR[Rs1]_In	The control signal to use the Rs1 operand to input data into the GPR from the bus
GPR[Rs1]_Out	The control signal to use the Rs1 operand to output data from the GPR to the bus
GPR[Rs2]_In	The control signal to use the Rs2 operand to input data into the GPR from the bus
GPR[Rs2]_Out	The control signal to use the Rs2 operand to output data from the GPR to the bus
PC_In	The control signal to enable GPR[7] to load the PC into it from the bus
PC_Out	The control signal to enable GPR[7] to output the PC onto the bus
GPR[0]_Out_CS	The control signal to enable GPR[0] to output a value of 0 onto the bus
Clock	16-bit register clock signal
Reset	16-bit register reset signal

Table 4-3 General Purpose Register (GPR) Input/Output Descriptions

The GPR consists of 8 16-bit registers, `GPR[0]` to `GPR[7]`. These registers hold various words for the instruction set and get addressed based on the IR data. However, there are 2 notable registers in the GPR. `GPR[0]` is not connected to the bus and the register is grounded so it always holds the value of 0 for arithmetic operations for the traps. In addition, `GPR[7]` is the location of the program counter (PC) and gets used and then updated in each fetch cycle of every instruction.

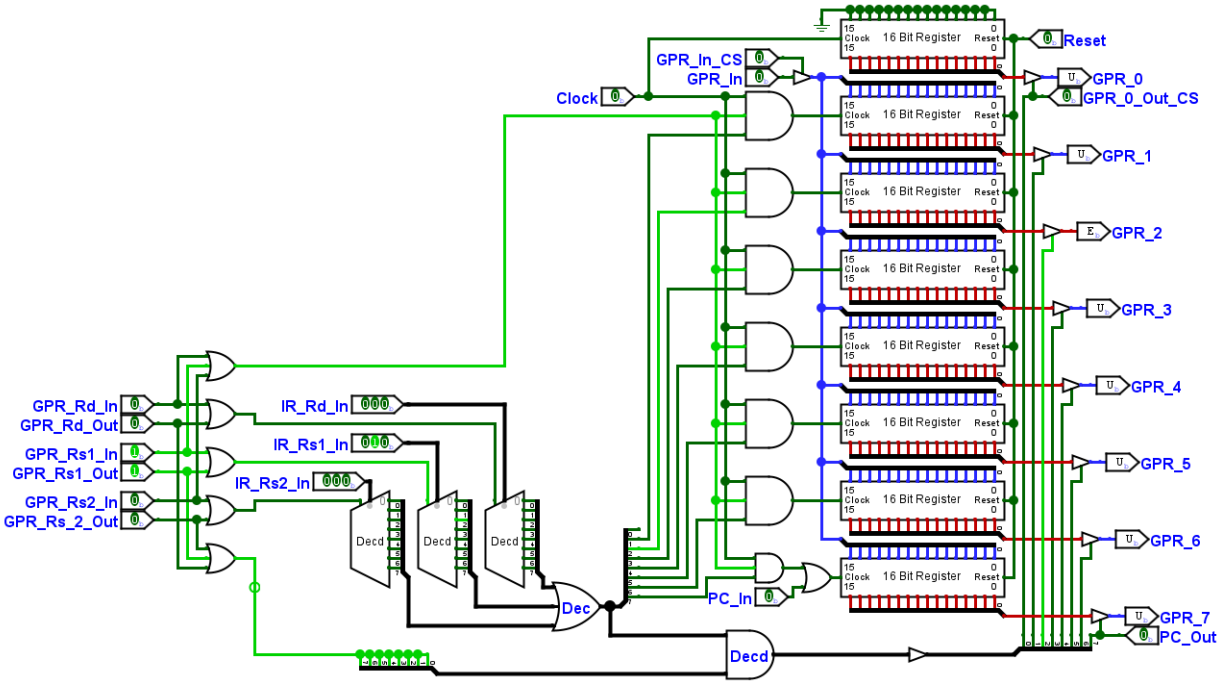

Figure 4-9 Gate Level View of the General Purpose Register

In addition to the 8 registers in the GPR, there are also control signal inputs (`GPR[Rd/Rs1/Rs2]_In`) from the control unit that when set high will turn on the appropriate decoder for the incoming 3 bit IR Rd/Rs1/Rs2 which controls a decoder to send a high signal to the specified AND gates of the appropriate registers. The decoded AND signal, along with the input control signal will turn on the clock of the appropriate registers when the clock pulses and load the new data from the bus into the registers. When the control signal inputs (`GPR[Rd/Rs1/Rs2]_Out`) are HIGH, it will also turn on the IR decoders signal and AND them with the output to open a tri state buffer in order to output the data of the specified GPR register on to the bus.

As an example, the Figure 4-9 has `GPR[Rs1]_In` set HIGH and `IR_Rs1_In` is 010. Since the other control signal inputs are low, the `IR_Rd` and `IR_Rs` decoders are off. But the 010 turns on the 2nd signal of the decoder OR gate *(Dec)*. The `GPR[Rs1]_In` signal is high so now 2 of 3 AND gate inputs are on, and when the clock pulses it will set the clock signal in `GPR[2]` high so any data coming in from the bus on `GPR_In` will get loaded into the `GPR[2]` register. Further in this example, since `GPR[Rs1]_Out` is set high, it will turn on the lower input to the AND gate *(Decd)* and the OR gate *(Dec)* will output the appropriate bus value which is still 2 in this example. This will turn on the appropriate AND signal to open the tri-state buffer that lets the `GPR[2]` register on to the bus. Additionally, since the inputs to the OR and AND gates, Dec and Decd, are buses, it can be thought of as if there are 8 separate OR/AND gates for every input and output signal from the buses. There is also no additional AND gate to control the clock signal input for `GPR[0]` since the value is always 0 in that register and does not need to be fed new values from the bus. Since `GPR[7]` is the PC, it needs an additional control signal to separately enable it to load data from and unload data to the bus. `PC_Out` enables the buffer to output to the bus and `PC_In` can separately pulse the clock to load data from the bus into `GPR[7]`. A buffer is also placed on the buffer control lines to act as a diode for data flow.

4.6 Main Memory/Memory Address Register/ Memory Data Register (MM/MAR/MDR)

Figure 4-10 shows the abstract view of the Main Memory (MM), Memory Address Register (MAR) and Memory Data Register (MDR) inputs and outputs that is shown in Figure 4-1. The outputs and inputs of the block are described below in Table 4-4. The gate level view of the MAR and MDR can be seen on the following page in Figure 4-11 and Figure 4-12, respectively.

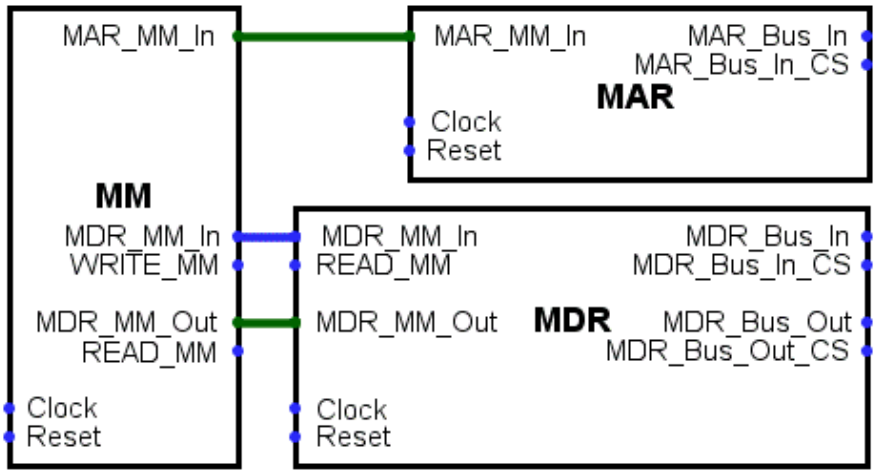

Figure 4-10 MM/MAR/MDR Abstract Data Block

MAR_Bus_In	Data in from the bus to the MAR
MAR_Bus_In_CS	The control signal to turn on a tri-state buffer and allows data from the bus to the MAR
MDR_Bus_In	Data in from the bus to the MDR
MDR_Bus_in_CS	The control signal to turn on a tri-state buffer and allows data from the bus to the MDR
MDR_Bus_Out	Data out from the MDR to the bus
MDR_Bus_Out_CS	The control signal to turn on a tri-state buffer and allows data from the MAR to the bus
MAR_MM_In	Data out from the MAR to the MM
MDR_MM_In	Data out from the MM to the MDR
MDR_MM_Out	Data in from the MDR to the MM
WRITE_MM	The control signal to turn on a tri-state buffer and allows data to be written from the MAR to the MM
READ_MM	The control signal to turn on a tri-state buffer and allows data to be read from the MM to the MDR
Clock	16-bit register clock signal
Reset	16-bit register reset signal

Table 4-4 MM/MAR/MDR Input/Output Descriptions

The Main Memory (MM) is not part of the scope of this project and therefore was implemented as a "black box" where it is assumed it operates on a basis of accepting a memory address from the MAR and fetches the correct data for the MDR and writes the data into the MDR when `READ_MM` is high. The MM will also read data from the MDR and write it into the appropriate address when `WRITE_MM` is high.

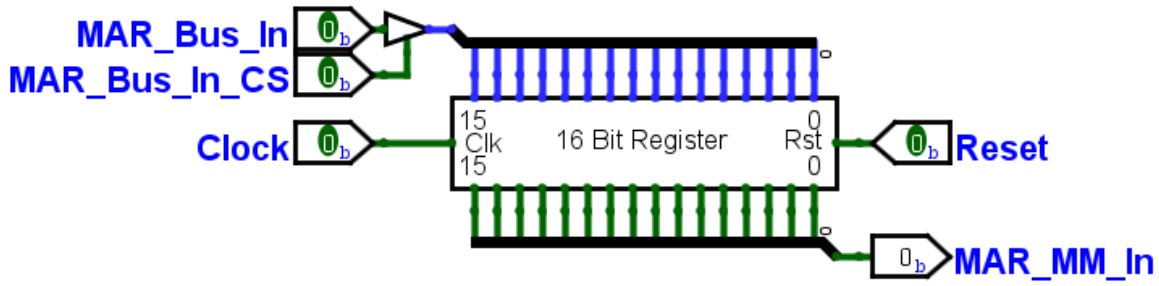

Figure 4-11 Gate Level View of the MAR

The MAR is not particularly interesting, it is comprised of a single 16 bit register to store the address of the memory to be fetched from the main memory. A tri-state buffer controls the input of the register from the bus similar to all past circuits. Since the main memory is not in the scope of this project, the memory address from the MAR will go to the MM where it is assumed the MM will fetch the correct data for the MDR from the address. The MAR cannot output anything to the bus, only receive data from the bus.

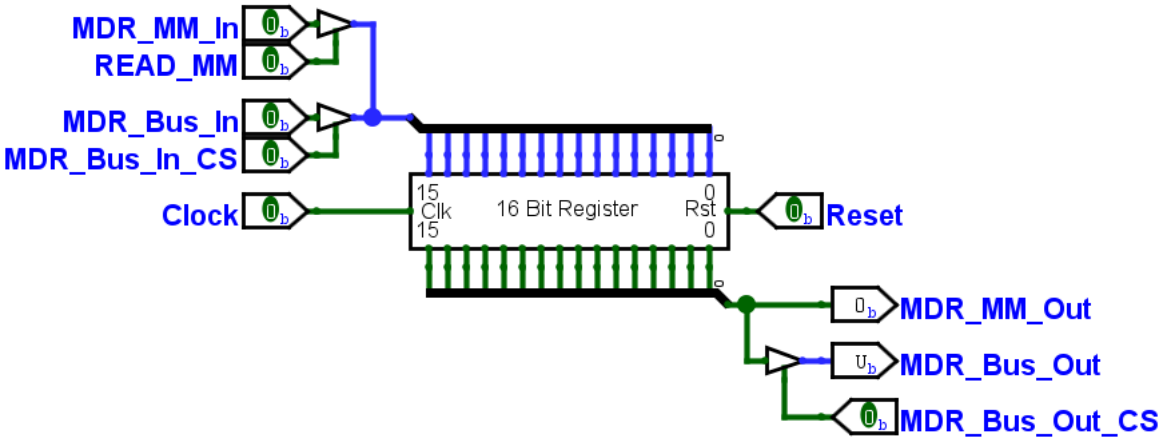

Figure 4-12 Gate Level View of the MDR

The MDR is slightly more complex than the MAR, but not by much. The main difference is an additional input and output. Unlike the MAR, the MDR can both receive and transmit data to the bus. Using READ_MM the MDR receives data from the MM.

4.7 Read Only Memory (ROM)

The Figure 4-13 shows the abstract view of the Read Only Memory (ROM) inputs and outputs that is shown in Figure 4-1. The outputs and inputs of the block are described below in Table 4-5. The gate level view of the ROM can be seen below in Figure 4-14.

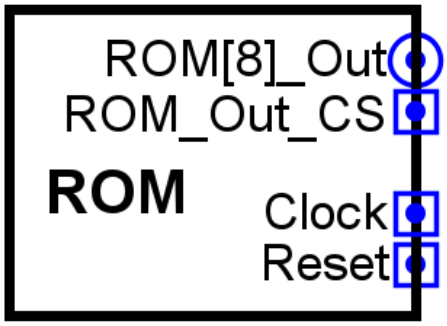

Figure 4-13 Read Only Memory (ROM) Abstract Data Block

ROM_Out	Data in from the bus to the ROM
ROM_Out_CS	The control signal to turn on a tri-state buffer and allows data from the ROM to the bus
Clock	16-bit register clock signal
Reset	16-bit register reset signal

Table 4-5 Read Only Memory (ROM) Input/Output Descriptions

The Read Only Memory serves as the name implies, a memory that can only be read from. The ROM was implemented as an optimization for the trap states. The ROM is a single 16 bit register that holds decimal 8 (`0b0000000000001000`) where the 4th bit is held high while all other bits are grounded and set to low. It requires no inputs since it is a static register but can output the value of 8 on to the bus for computation and arithmetic purposes in the trap states.

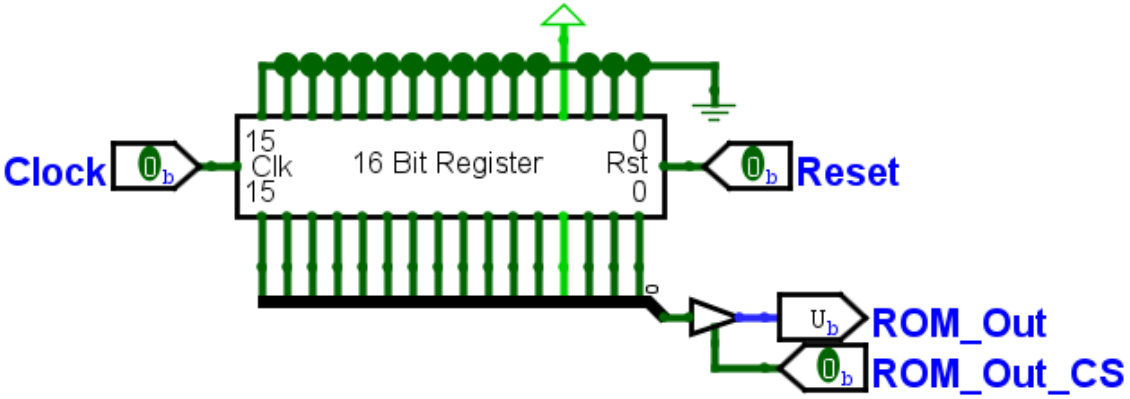

Figure 4-14 Gate Level View of the ROM

4.8 Countdown Timer

The Figure 4-15 shows the abstract view of the Countdown Timer inputs and outputs that is shown in Figure 4-1. The outputs and inputs of the block are described below in Table 4-6.

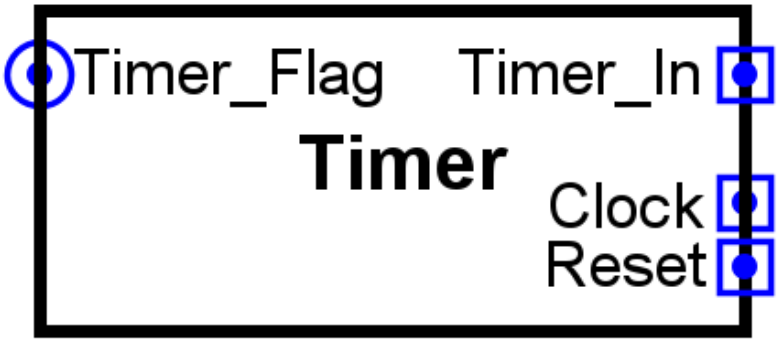

Figure 4-15 Countdown Timer Abstract Data Block

Timer_In	Data in from the bus to the Timer
Timer_Flag	The control signal to signify to the control unit that the timer has reached 0
Clock	16-bit register clock signal
Reset	16-bit register reset signal

Table 4-6 Read Only Memory (ROM) Input/Output Descriptions

The actual implementation and operation of the timer is outside the scope of this project so there is no gate level view of the timer. The timer counts down during the operation of the machine and when the timer hits 0 it turns on the `Timer_Flag` signal which causes the machine to enter into a trap state and the machine executes a series of states and controls accordingly. `Timer_In` will load the timer from the bus when the control signal is activated.

4.9 Arithmetic Logic Unit (ALU)

The Figure 4-16 shows the abstract view of the Arithmetic Logic Unit (ALU) and the associated Y Register and Z Register for intermediate data storage before and after ALU operations. Due to the comprehensive and expansive nature of the ALU and its associated parts, the following 2 pages will cover the Y and Z register and then the individual subsections of the ALU.

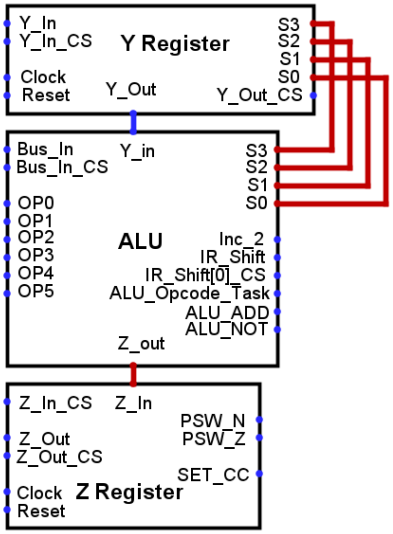

Figure 4-16 Arithmetic Logic Unit, Y Register, and Z Register Abstract Data Block

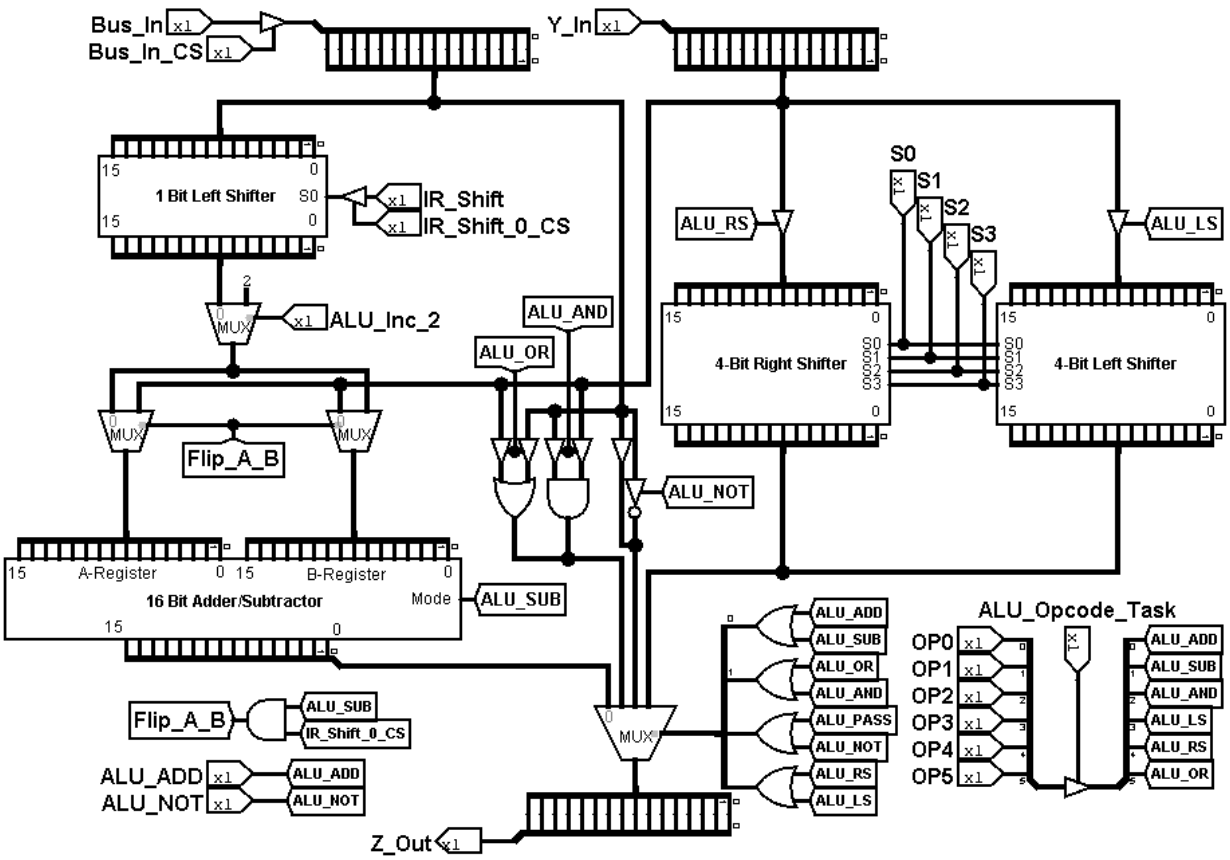

Figure 4-17 Gate Level View of the ALU

4.9.1 Y Register

The Figure 4-17 on the previous page shows the abstract view of the Y register. The outputs and inputs of the block are described below in Table 4-7. In addition, Figure 4-18 shows the gate level view of the Y register.

Y_In	Data in from the bus to the Y register
Y_In_CS	The control signal to turn on a tri-state buffer and allows data from the bus to the Y register
Y_Out	Data out from the Y Register to the ALU
Y_Out_CS	The control signal to turn on a tri-state buffer and allows data from the Y Register to the ALU
S0 -> S3	Shift amount bits, 0 – shift by one bit, 1 – shift by two bits, 2 - shift by 3 bits, 3 – shift by 4 bits
Clock	16-bit register clock signal
Reset	16-bit register reset signal

Table 4-7 Y Register Input/Output Descriptions

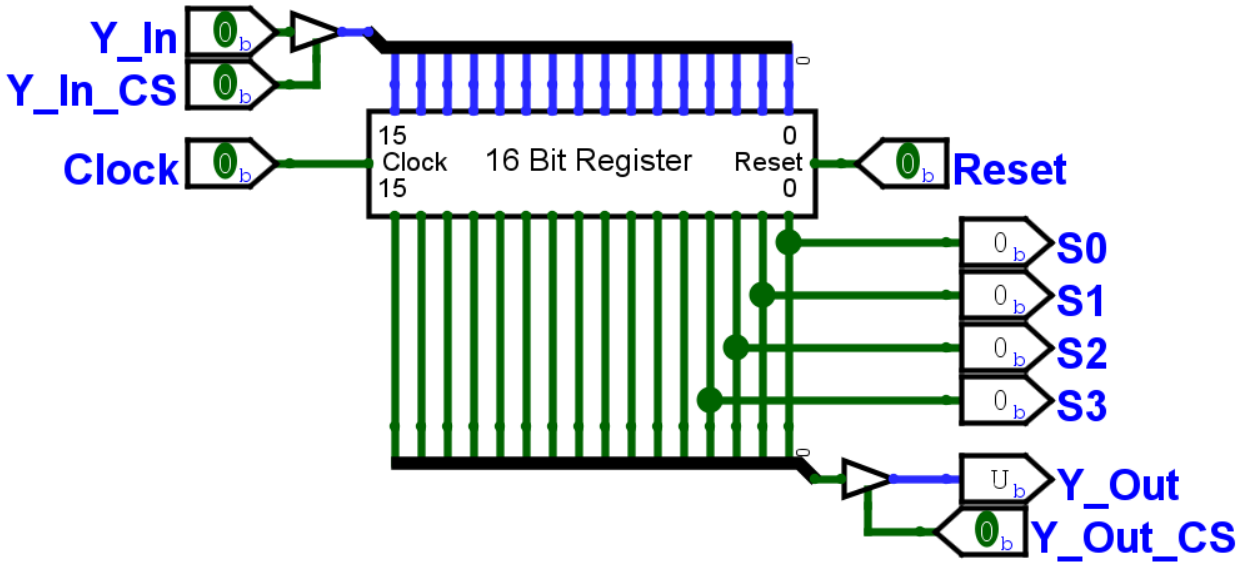

Figure 4-18 Gate Level View of the Y Register

The input and output of the Y register is similar to most other blocks in the design. The unique part of the Y register however is the select bits, S0 to S3. The first four bits of the Y register determine the shifting bit quantity of the left shifter and right shifter, utilized by OP 3 (SHL) and 4 (SHRA) in the ALU when they are needed. The shifting registers are discussed further in depth later in the report.

4.9.2 Z Register

The Figure 4-17 on Page 25 shows the abstract view of the Z register. The outputs and inputs of the block are described below in Table 4-8. In addition, Figure 4-19 shows the gate level view of the Z register.

Z_In	Data in from the bus to the Y register
Z_In_CS	The control signal to turn on a tri-state buffer and allows data from the bus to the Y register
Z_Out	Data out from the Y Register to the ALU
Z_Out_CS	The control signal to turn on a tri-state buffer and allows data from the Y Register to the ALU
PSW_N	Condition code bit N, goes to PSW
PSW_Z	Condition code bit Z, goes to PSW
SET_CC	The control signal to turn on a tri-state buffer and allow the Z register to set PSW.Z and PSW.N
Clock	16-bit register clock signal
Reset	16-bit register reset signal

Table 4-8 Z Register Input/Output Descriptions

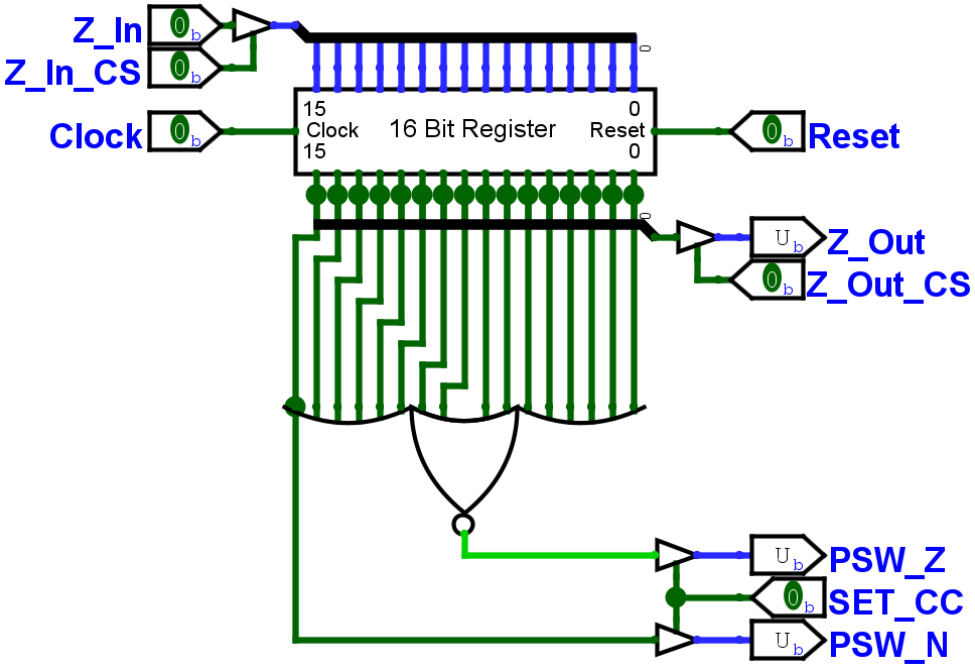

Figure 4-19 Gate Level View of the Z Register

The input and output of the Z register is similar to most other blocks in the design. The unique part of the Z register however is the OR gate attached to the output of the register. Based on the project requirements, when the `SET_CC` command is activated, the 16 input NOR gate is used to check if the value in the Z register is 0, and the most significant bit of the register is used to determine if the value of the Z register is negative and then both values are pushed to PSW.Z and PSW.N in the PSW.

4.9.3 16 Bit Carry Lookahead Adder-Subtractor

The ALU utilizes a carry lookahead adder-subtractor circuit in order to add or subtract two 16-bit values and output a single 16-bit value. The carry lookahead generator, while it utilizes more gates than the ripple-carry adder design, results in a significant decrease in propagation delay and performance time through the operations.

The base part of the carry lookahead is shown below in Figure 4-20, is created using a 4 bit carry lookahead generator adder with two inputs for every output. The G input is known as the carry generate bit and the P input is known as the carry propagate bit. Each of these bits also relate to the A and B register operands that are discussed further later in this section.

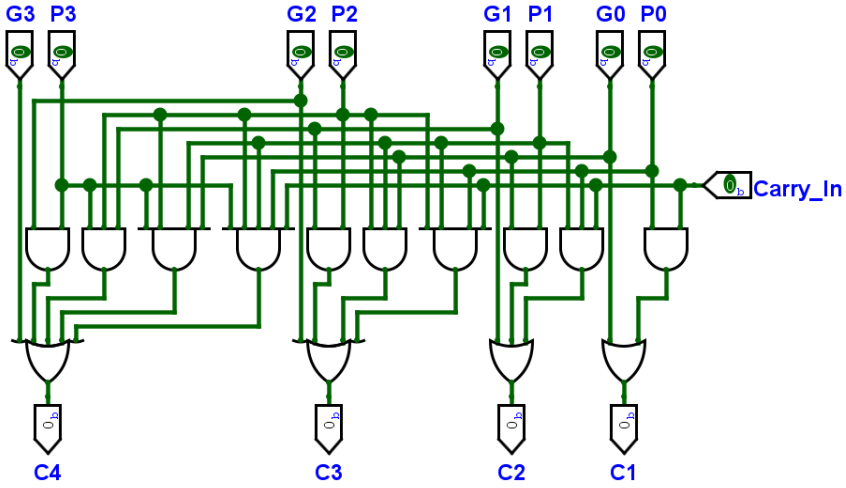

Figure 4-20 4-Bit Carry Lookahead

The 4-bit carry lookahead adds together the G and P inputs, using an optional carry in bit for larger operations. The outputs of each bit operation are shown in C. With each operation, the previous G and P bits are factored into the following bits at the same, reducing in propagation delay. Combining four of these together results in Figure 4-21 shown below.

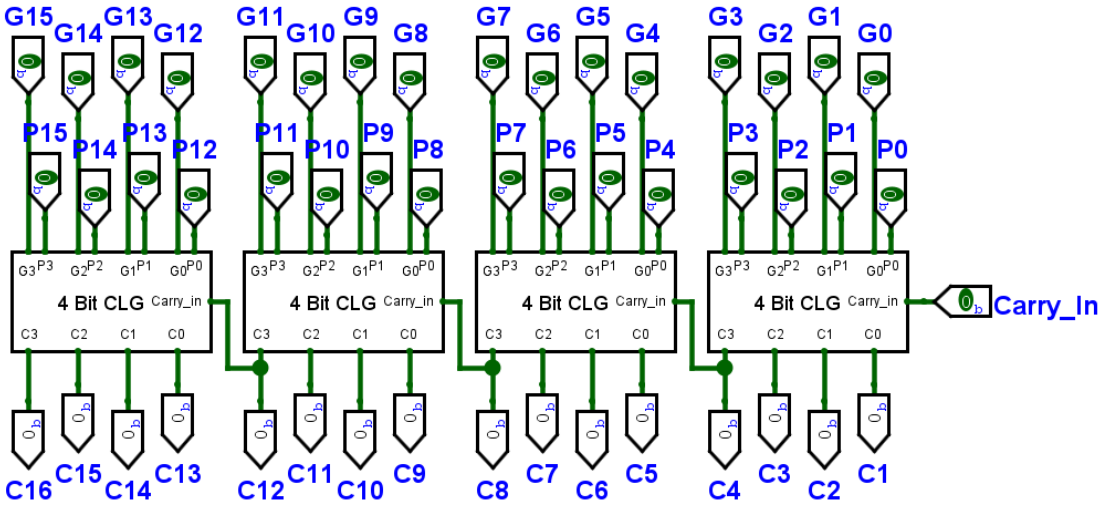

Figure 4-21 4X4 Carry Lookahead

Each of the four 4 bit CLG's input their last bit as the carry in to the subsequent 4 bit CLG. The combination of the 4X4 bit CLG creates the full carry lookahead adder/subtractor with some additional supporting circuitry shown below in Figure 4-22.

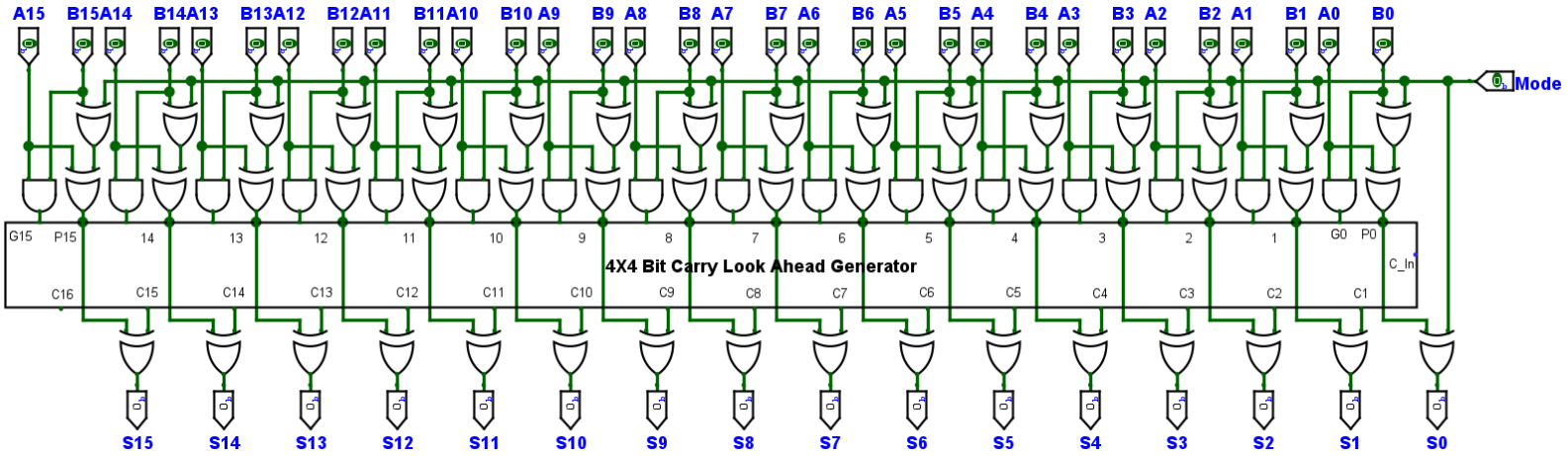

Figure 4-22 16 Bit Carry Lookahead Adder/Subtractor

Utilizing the carry generate (G), carry propagate (P) and outputs (C), each bit from the two 16-bit input registers A and B can be added/subtracted from each other according to the equations shown below to equal their output S. $\oplus$ stands for the XOR gate/operation and $*$ stands for AND gate/operation. In addition, [x] stands for the individual bits 0 to 15.

$$S[x] = P[x] \oplus C[x]$$
$$G[x] = A[x] * B[x]$$
$$P[x] = A[x] \oplus B[x]$$

The additional XOR gates and supporting circuitry above the carry lookahead generator are necessary for the full 16-bit carry look ahead generator to be able to perform a 2's complement operation on the bits as per the project requirements. The Mode input on the right-hand side works in combination with the 2's complement to change the function of the carry lookahead generator between addition and subtraction. When Mode is HIGH the CLA will subtract B from A or, in other words, S[x] = B[x] – A[x]. When Mode is LOW the CLA will add B to A, or in other words S[x] = B[x] + A[x]. Mode will stay low in all cases except when `ALU_Sub` is set high in the ALU.

Figure 4-23 on the following page shows the full carry lookahead circuit and the supporting circuitry for additional functionality inside the ALU. `ALU_Sub` is denoted as subtract in this figure and the A and B registers are broken out into 16 bits to display the bus size of each wire for clarity.

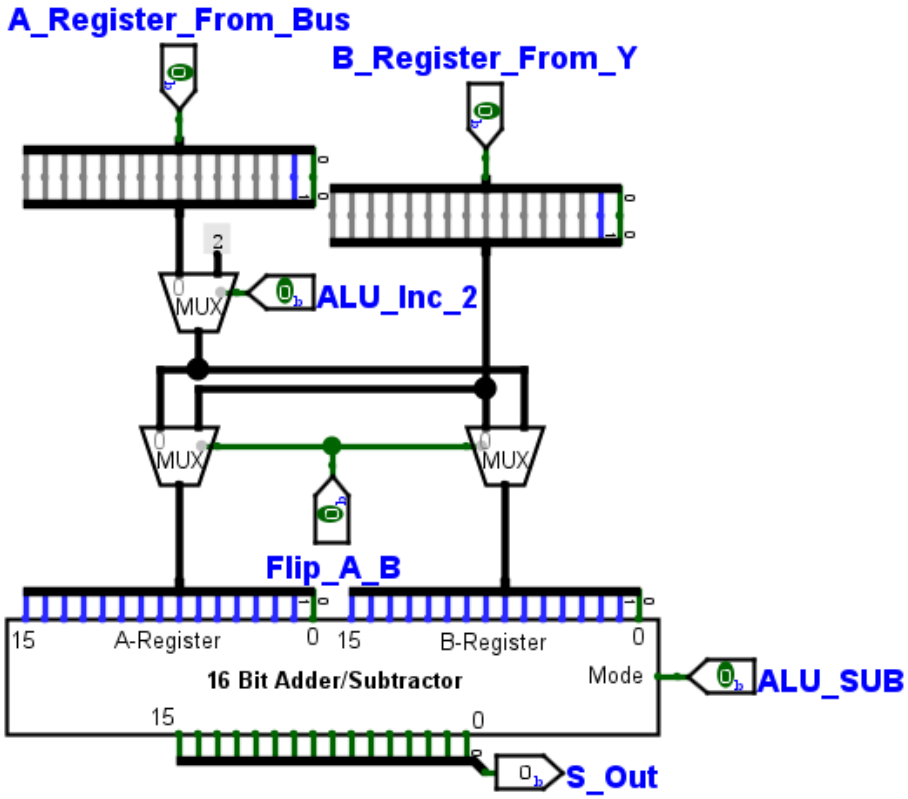

Figure 4-23 Full 16 Bit Carry Lookahead Adder/Subtractor

In some cases, such as when the binary value of A is bigger than B, the subtract operation cannot be performed with correct 2's complement results. To account for this, two multiplexers were put before adder/subtractor to correct this to flip which operand is being subtracted from the other. The signal `Flip_A_B` when LOW will place the correct A register from the bus into the A-register of the adder/subtractor circuit and vice versa with the B register from the Y register. However, when `Flip_A_B` is HIGH, it will flip the inputs, thus putting the B register from the Y register into the A-register input in the adder/subtractor and the A register from the bus into the B-Register for the proper 2's complement operation. Since `Flip_A_B` only needs to be turned on in certain scenarios, it was ascertained that this only is necessary when the value going into A was not shifted previously and the subtract operation is active, the equation below was made to determine when the multiplexers would flip the inputs.

$$Flip\_A\_B \; = \; ALU\_SUB \; * \; (IR.Shift[1] \; == \; 0)$$

The `IR.Shift[1]` is the result of possible left shifting to the 16 bit input from the bus and will be discussed in the next section of this report.

Finally, there is a third multiplexer after the A register from the bus. This multiplexer is utilized to add the constant decimal value "2" (`0b0000000000000010`) to the B register coming from the Y register. This is utilized as an optimization in the execution of the trap conditions since it was observed that the trap steps shown earlier in Figure 1-4 are always incrementing by steps by 2, when `ALU_Inc_2` is set HIGH, it will insert "2" into the A-register to be added to the value coming into the B-register.

4.9.4 Left Shift and Right Shift

The Figure 4-17 on Page 25 shows the abstract view of the 3 shifters in the ALU. Figure 4-24 shows the two 4-bit shifters in the ALU. The 1-bit left shifter will be discussed later in this section.

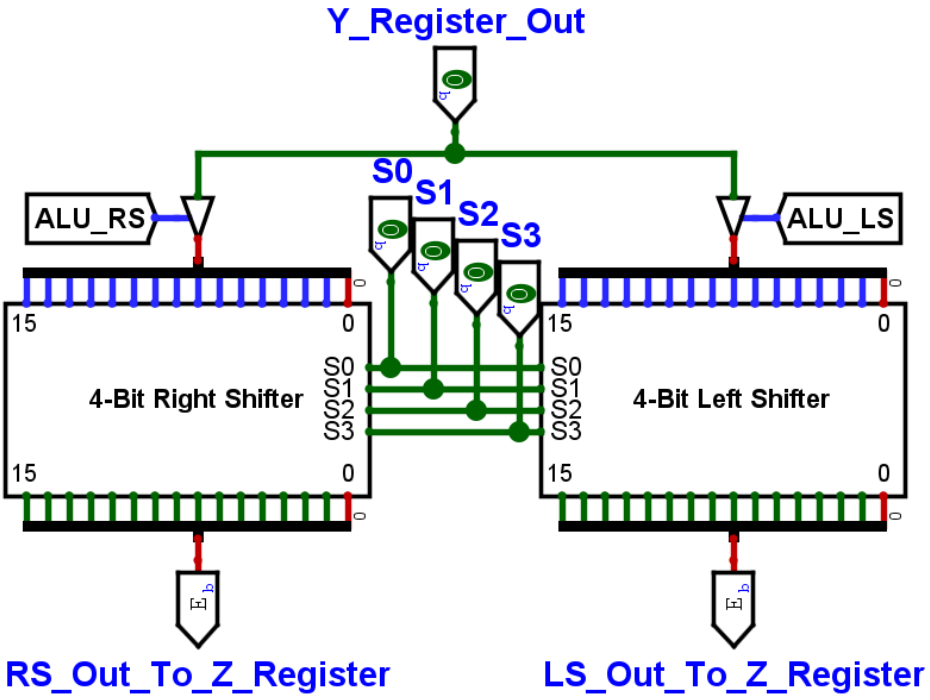

Figure 4-24 4-Bit Left Shifter and 4-Bit Right Shifter

Both shifters utilize the same input from the Y register, since there are control buffers (which will be elaborated on at the end of the ALU section) only the left or right shift will be active at once and output to the Z register at one time. The four selection bits (S0, S1, S2 and S3) come in from the Y register's low four order bits. They control the amount of bits to be shifted by in the right or left shifter by up to 15 bits. However, only one of the selection bits will ever be active at one time, therefore shifting by 1 bit (S0 is HIGH), 2 bits (S1 is HIGH), 3 bits (S2 is HIGH) and 4 bits (S3 is HIGH). The 4 bit left shifter operates

4.9.4.1 4-Bit Left Shift

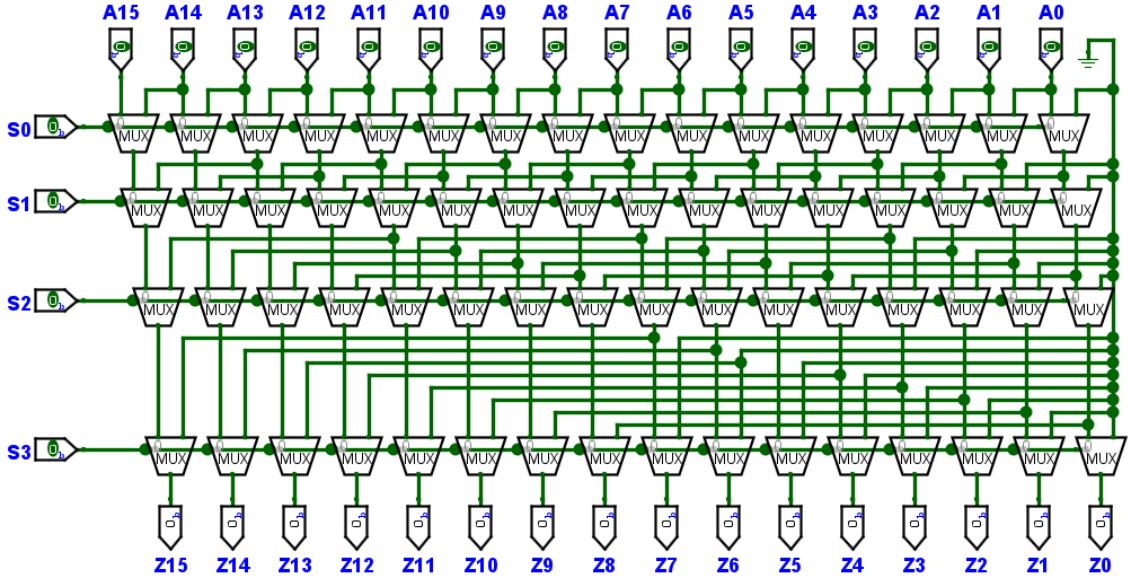

Figure 4-25 Gate Level View of the 4-bit Logical Left Shift

The 4-bit left shift is a logical shift based on the selection bits, due to this the first input of the least significant bit multiplexer is grounded to ensure this. The gate level view of the entire left shift operation can be seen above in Figure 4-25

4.9.4.2 4-Bit Right Shift

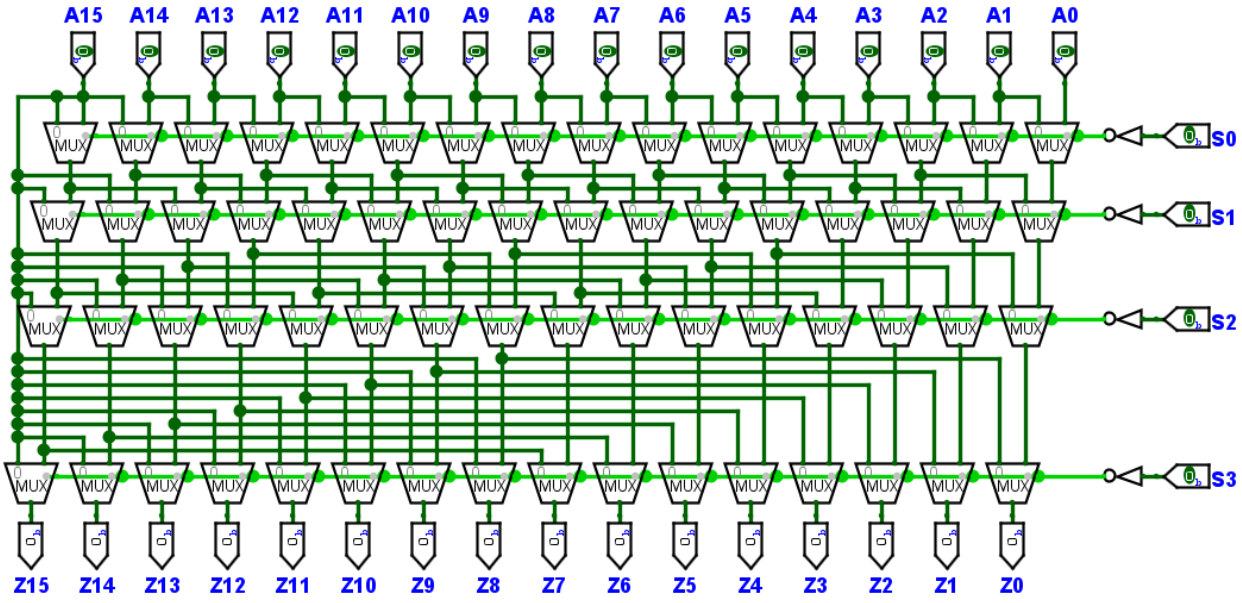

Figure 4-26 Gate Level View of the 4-bit Logical Right Shift

The 4-bit right shift is an arithmetic shift based on the selection bits. It is similar to the left shift in its operation; However, the most significant bit is set to the last input to preserve the sign bit. The gate level view of the entire right shift operation can be seen above in Figure 4-26

4.9.4.3 1 Bit Left shift

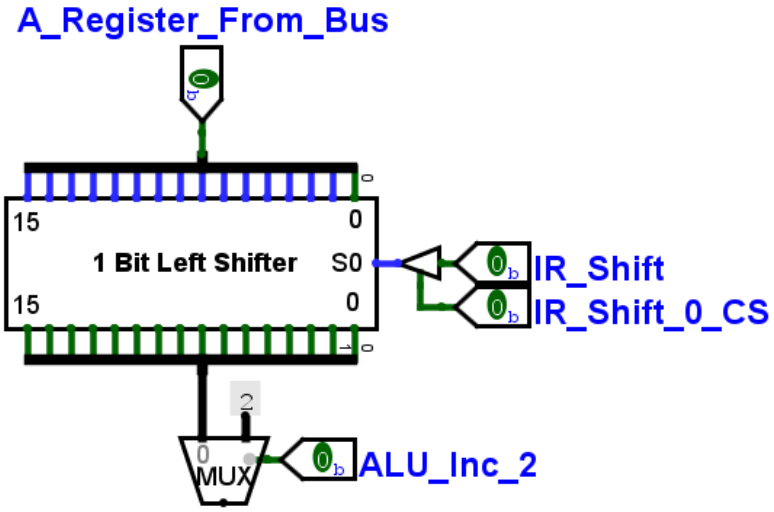

Figure 4-27 1-Bit Left Shifter

As part of an optimization, directly after the bus input, a 1-bit left shifter was added. The optimization, at the expense of slightly more hardware, is meant to speed up the execution of opcodes 0 (AND) ,1 (SUB) , 2 (AND) and 5 (OR). Since these operands are always shifted by a value of IR.Shift, we assume that the opcodes 3 (SHL) and 4 (SHRA) occur less often than the other instructions and the 1-bit shifter being in line with the adder/subtractor allows opcodes 0,1,2 and 5 to occur faster since they don't need to go through the 4-bit shifter, into the Z register and then back into the ALU. It is also known that the IR.Shift is more often a value of 0 or 1 instead of 2 or 3 which allows the 1 bit shifter to shift by 0 or 1 bit for faster operation. The IR_Shift comes from the instruction register and the `IR_Shift[0]_CS` will shift by 0 or 1 when the control signal is HIGH from the control unit for it. Figure 4-28 below shows the gate level view of this circuit; however, it operates identically to the 4-bit shifter but with only one layer.

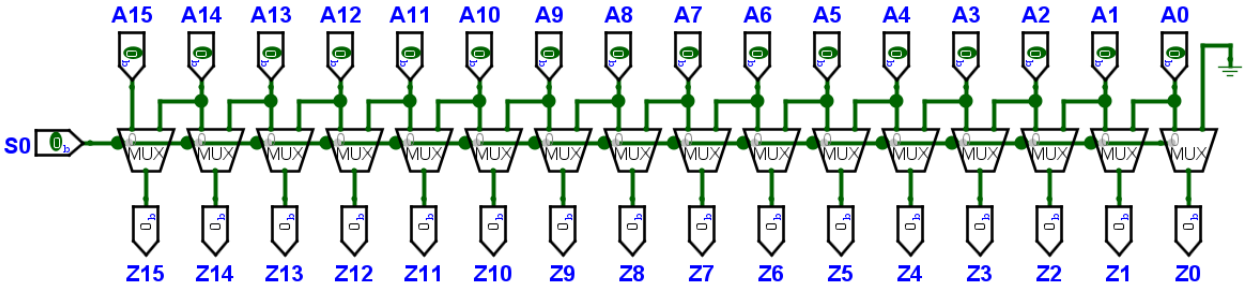

Figure 4-28 Gate Level View of the 1 Bit Shifter

4.9.5 OR/AND/NOT in the ALU

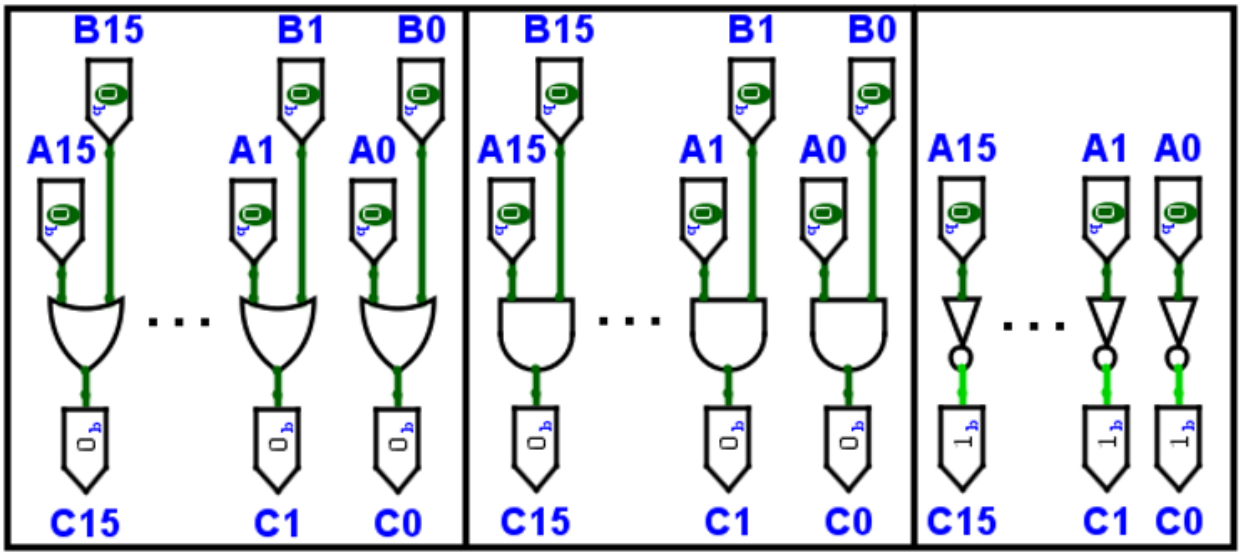

Figure 4-29 OR/AND/NOT Operations

The standard OR/AND/NOT operations are fairly simple. For the OR and AND gates, one side of the gate takes the input from the bus (A0 to A15) and then ORs/ANDs it with the input from the Y register (B0 to B15). The NOT input only NOTs the input from the bus register. This can be seen in the 3 slices shown in Figure 4-29

4.9.6 Z Register Output Multiplexer

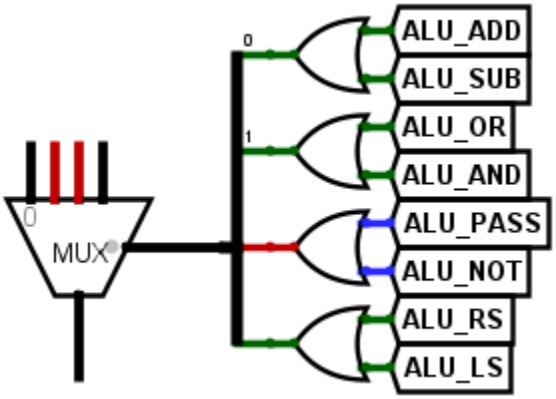

Figure 4-30 Output Multiplexer Gate Level View

After all operations are complete in the ALU, a multiplexer will select one of the four outputs from the carry lookahead adder/subtractio, the OR/AND gates, a pass through from the bus register and the NOT gate, and the left shift/right shift Figure 4-29. This is further explained in the final section of the ALU on the next page. Since there are 8 inputs that get tied together into 4 inputs, there are 8 selection bits that are appropriatetly OR'd together for 4 selection bit inputs.

4.9.7 ALU Final Additions

Figure 4-31 is identical to the full ALU view Figure 4-17 on page 28. It has been placed here for the readers convenience.

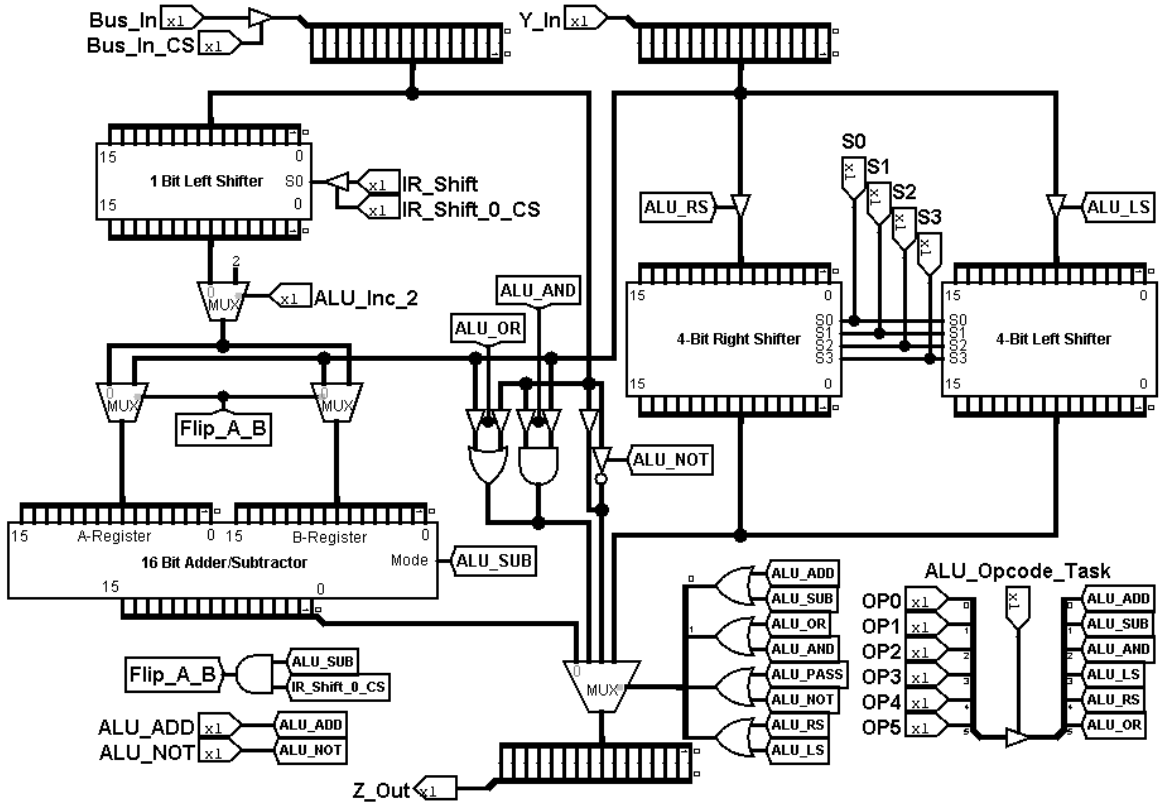

Figure 4-31 Gate Level View of the ALU

The overall ALU is shown again above. Each of the individual components are mentioned in the previous sub sections. In accordance with the optimizations mentioned in the following section, OP codes 0 to 5 are brought in from the IR to reduce on control signals and simplify the control unit. The control signal `ALU_Opcode_Task` turns on the appropriate control buffer to allow the proper ALU operation to take place based on the first 6 OP codes.

Additionally, the 16 bit buses broken out at `Bus_In`, `Y_In`, and `Z_Out` are purely for clarity purposes to show the 16 bit width of the buses going through the ALU and are NOT separate storage registers.

Since `Y_In` and `Z_Out` have their own control signals in the Y and Z registers they are not replicated here.

5 Optimization Summarization and Appendix

This section is a re-cap of the various optimizations through this project and their locations in the report as well as an appendix showing the original control signals before optimization.

Optimization 1, Control Signals:

The normal operating sequence of the machine follows the fetch/decode/execute cycle unless the two trap exceptions occur. The instructions are decoded and executed in their separate branches after the same fetch sequence for 3 clock cycles, illustrated in Table 3-1 (A), where the first optimization is implemented. The CPU continues to increment the PC by one byte while waiting for the reading of the MM to finish, and additionally, keep the updated PC value in the Y register to assist later math operations done on the PC such as OP6. (Page 6)

Note: There was extensive refinement of the control signals. This can be further seen in Figure 5-1 in the appendix below with the original control signals which resulted in 29 control signals versus the original 39, which caused a further optimization of only requiring 5 D-flip flips instead of 6 for the states transitions.

Optimization 2, Separate 1-Bit Left Shifter:

As part of an optimization, directly after the bus input, a 1-bit left shifter was added. The optimization, at the expense of slightly more hardware, is meant to speed up the execution of opcodes 0 (AND) ,1 (SUB) , 2 (AND) and 5 (OR). Since these operands are always shifted by a value of `IR.Shift`, we assume that the opcodes 3 (SHL) and 4 (SHRA) occur less often than the other instructions and the 1-bit shifter being in line with the adder/subtractor allows opcodes 0,1,2 and 5 to occur faster since they don't need to go through the 4-bit shifter, into the Z register and then back into the ALU. (Page 36)

Optimization 3, OP code control signals in the ALU:

Secondly, instead of sending six different control signals to dictate ALU functions, similarly a 4-bit connection for `Opcode` to command ALU directly, enabled by `ALU_Opcode_Task` control, such that six branched-out control sequences can combine into one. The resulting decode/execute sequence is described in Table 3-1 (B). (Page 6)

Optimization 3, Utilizing GPR[0] for the trap control signals:

The program-check exception is triggered when OP14 and OP15 are used under user mode, so it should occur after the instruction is fetched. It is found that the relevant memory address in this control sequence follows an increment-by-two pattern from 0 to 6 in each step, for which an increment-by-two function in ALU was implemented to increment the even-number-generation with `GPR[0]` since it is always 0. (Page 6 and Page 23)

Optimization 4, a ROM with a single register holding the value of 8:

Different from program-check, a timeout exception is triggered before the fetch sequence when the timer hits 0 and executes similarly except the address starts at 8. Thus, a ROM of one 16 bit register which keeps a constant number 8 was implemented, this number was used to initialize the even-number-generation instead. This merges the two exceptions into one sequence except for first step, shown in Table 3-1 (K) and is the last optimization. (Page 6 and Page 26)

Optimization 5, Carry lookahead generator over a ripple carry:

The ALU utilizes a carry lookahead adder-subtractor circuit in order to add or subtract two 16-bit values and output a single 16-bit value. The carry lookahead generator, while it utilizes more gates than the ripple-carry adder design, results in a significant decrease in propagation delay and performance time through the operations.

Optimization 6, Op3 and Op4 shift bits are in the Y Register

The input and output of the Y register is similar to most other blocks in the design. The unique part of the Y register however is the select bits, S0 to S3. The first four bits of the Y register determine the shifting bit quantity of the left shifter and right shifter, utilized by OP 3 (SHL) and 4 (SHRA) in the ALU when they are needed. The shifting registers are discussed further in depth later in the report. (Page 29)

Optimization 7, PC is put in GPR[7] instead of its own external register:

Since `GPR[7]` is the PC, it needs an additional control signal to separately enable it to load data from and unload data to the bus. `PC_Out` enables the buffer to output to the bus and `PC_In` can separately pulse the clock to load data from the bus into `GPR[7]`. A buffer is also placed on the buffer control lines to act as a diode for data flow. (Page 23)

ADD Op0 GPR[Rd] = GPR[Rs1] + left shifted(GPR[Rs2],IR.Shift)

Fetch)

1) PC_out, MAR_in, ALU/inc_right, Z_in, READ_MM
2) WMFC, Z_out, PC_in, Y_in
3) IR_in, MDR_out

Decode/Execute)

4) GPR[Rs2]out, ALU_in_Right, ALU.Leftshift_IR.shift, Z_in
1) Z_out, Y_in
2) GPR[Rs1]_out, ALU_in_Right, ALU_in_Left, ALU_ADD, Z_in
3) Z_out, GPR[Rd]_in

SUB Op1 GPR[Rd] = GPR[Rs1] - left shifted(GPR[Rs2],IR.Shift)

Fetch)

1) PC_out, MAR_in, ALU/inc_right, Z_in, READ_MM
2) WMFC, Z_out, PC_in, Y_in
3) IR_in, MDR_out

Decode/Execute)

4) GPR[Rs2]out, ALU_in_Right, ALU.Leftshift_IR.shift, Z_in
5) Z_out, Y_in
6) GPR[Rs1]_out, ALU_in_Right, ALU_in_Left, ALU_SUB, Z_in
7) Z_out, GPR[Rd]_in

AND Op2 GPR[Rd] = GPR[Rs1] and left shifted(GPR[Rs2],IR.Shift)

Fetch)

1) PC_out, MAR_in, ALU/inc_right, Z_in, READ_MM
2) WMFC, Z_out, PC_in, Y_in
3) IR_in, MDR_out

Decode/Execute)

4) GPR[Rs2]out, ALU_in_right, ALU.Leftshift_IR.shift, Z_in
5) Z_out, Y_in
6) GPR[Rs1]_out, ALU_in_Right, ALU_in_Left, ALU_AND, Z_in
7) Z_out, GPR[Rd]_in

SHL Op3 GPR[Rd] = shift left(GPR[Rs1]) by left shifted(GPR[Rs2],IR.Shift)3-0

Fetch)

1) PC_out, MAR_in, ALU/inc_right, Z_in, READ_MM
2) WMFC, Z_out, PC_in, Y_in
3) IR_in, MDR_out

Decode/Execute)

4) GPR[Rs2]out, ALU_in_right, ALU.Leftshift_IR.shift, Z_in
5) Z_out, Y_in
6) GPR[Rs1]_out, ALU_in_Right, ALU_in_Left, ALU.Leftshift(3 to 0 bits), Z_in
7) Z_out, GPR[Rd]_in

SHRA Op4 GPR[Rd] = shift right(GPR[Rs1]) by left shifted(GPR[Rs2],IR.Shift)3-0

Fetch)

1) PC_out, MAR_in, ALU/inc_right, Z_Lin, READ_MM
2) WMFC, Z_out, PC_in, Y_in
3) IR_in, MDR_out

Decode/Execute)

4) GPR[Rs2]out, ALU_in_right, ALU.Leftshift_IR.shift, Z_in
5) Z_out, Y_in
6) GPR[Rs1]_out, ALU_in_Right, ALU_in_Left, ALU.Rightshift(3 to 0 bits), Z_in
7) Z_out, GPR[Rd]_in

OR Op5 GPR[Rd] = GPR[Rs1] or left shifted(GPR[Rs2],IR.Shift)

Fetch)

1) PC_out, MAR_in, ALU/inc_right, Z_in, READ_MM
2) WMFC, Z_out, PC_in, Y_in
3) IR_in, MDR_out

Decode/Execute)

4) GPR[Rs2]out, ALU_in_right, ALU.Leftshift_IR.shift, Z_in
5) Z_out, Y_in
6) GPR[Rs1]_out, ALU_in_Right, ALU_in_Left, ALU_OR, Z_in
7) Z_out, GPR[Rd]_in

NOT Op6 GPR[Rd] = not MM[PC + Short Offset]

Fetch)

1) PC_out, MAR_in, ALU/inc_right, Z_in, READ_MM
2) Z_out, PC_in, Y_in
3) IR_in, MDR_out

Decode/Execute)

4) IR.Short_off_out, ALU_in_Right, ALU_in_Left, ALU_ADD, Z_in
5) READ_MM, Z_out, MAR_in
6) MDR_out, ALU_in_Right, ALU_NOT, Z_in
7) Z_out,GPR[Rd]_in

LD Op7 GPR[Rd] = MM[PC + Short Offset]

Fetch)

1) PC_out, MAR_in, ALU/inc_right, Z_in, READ_MM
2) Z_out, PC_in, Y_in
3) IR_in, MDR_out

Decode/Execute)

4) IR.Short_off_out, ALU_in_Right, ALU_in_Left, ALU_ADD, Z_in
5) READ_MM, Z_out, MAR_in
6) MDR_out, GPR[Rd]_in

ST Op8 MM[PC + Short Offset] = GPR[Rd]

Fetch)

1) PC_out, MAR_in, ALU/inc_right, Z_in, READ_MM
2) Z_out, PC_in, Y_in
3) IR_in, MDR_out

Decode/Execute)

4) IR.Short_off_out, ALU_in_Right, ALU_in_Left, ALU_ADD, Z_in
5) GPR[Rd]_out, MDR_in
6) WRITE_MM, Z_out, MAR_in

BRN Op9 if CC.N then PC = PC + Long Offset

Fetch)

1) PC_out, MAR_in, ALU/inc_right, Z_in, READ_MM
2) WMFC, Z_out, PC_in, Y_in
3) IR_in, MDR_out

Decode/Execute)

4) IR.Long_off_out, ALU_in_Right, ALU_in_Left, ALU_ADD, Z_In
5) Z_out, PC_in

BRZ Op10 if CC.Z then PC = PC + Long Offset

Fetch)

1) PC_out, MAR_in, ALU/inc_right, Z_in, READ_MM
2) WMFC, Z_out, PC_in, Y_in
3) IR_in, MDR_out

Decode/Execute)

4) IR.Long_off_out, ALU_in_Right, ALU_in_Left, ALU_ADD, Z_In
5) Z_out, PC_in

BR Op11 PC = PC + Long Offset

Fetch)

1) PC_out, MAR_in, ALU/inc_right, Z_in, READ_MM
2) WMFC, Z_out, PC_in, Y_in
3) IR_in, MDR_out

Decode/Execute)

4) IR.Long_off_out, ALU_in_Right, ALU_in_Left, ALU_Add, Z_In
5) Z_out, PC_in

JSR Op12 GPR[Rd] = PC; PC = PC + Short Offset

Fetch)

1) PC_out, MAR_in, ALU/inc_right, Z_in, READ_MM
2) WMFC, Z_out, PC_in, Y_in
3) IR_in, MDR_out

Decode/Execute)

4) GPR[Rd]_in, PC_out
5) IR.Short_off_out, ALU_in_Right, ALU_in_Left, ALU_ADD, Z_in
6) Z_out, PC_in

RTS Op13 PC = GPR[Rd] + Short Offset

Fetch)

1) PC_out, MAR_in, ALU/inc_right, Z_in, READ_MM
2) WMFC, Z_out, PC_in, Y_in
3) IR_in, MDR_out

Decode/Execute)

4) GPR[Rd]_out, ALU_in_One
5) IR.Short_off_out, ALU_in_Right, ALU_in_Left, ALU_ADD, Z_in

CLK Op14 Set timer to MM[PC + Long Offset]

Fetch)

1) PC_out, MAR_in, ALU/inc_right, Z_in, READ_MM
2) WMFC, Z_out, PC_in, Y_in
3) IR_in, MDR_out

Decode/Execute)

4) IR.Long_off_out, ALU_in_Right, ALU_in_Left, ALU_ADD, Z_In
5) READ_MM, Z_out, MAR_in
6) WMFC
7) MDR_out, Timer_in

LPSW Op15 PSW = MM[PC + Long Offset]

Fetch)

1) PC_out, MAR_in, ALU/inc_right, Z_in, READ_MM
2) WMFC, Z_out, PC_in, Y_in
3) IR_in, MDR_out

Decode/Execute)

4) IR.Long_off_out, ALU_in_Right, ALU_in_Left, ALU_Add, Z_In
5) READ_MM, Z_out, MAR_in
6) WMFC
7) MDR_out, PSW_in

Program Check Violation

(P in PSW isnt set and someone tries to do OP14 or OP15)
MM[0] = PSW
MM[2] = PC
PSW = MM[4]
PC = MM[6]

1) PC_out, MAR_in, ALU/inc_right, Z_in, READ_MM
2) Z_out, PC_in, Y_in
3) IR_in, MDR_out

Decode/Execute)

4) GPR[0]_out, MAR_in, Y_in
5) PSW_out, MDR_in, WRITE_MM, ALU_Inc_2, Z_in
6) Z_out, MAR_in, Y_in
7) GPR[7]_out, MDR_in, WRITE_MM, ALU_Inc_2, Z_in
8) Z_out, MAR_in, Y_in, READ_MM,
9) MDR_out, PSW_in, ALU_Inc_2, Z_in,
10) Z_out, MAR_in, READ_MM
11) MDR_out, PC_in

Timeout

(Timer = 0)
MM[8] = PSW
MM[10] = PC
PSW = MM[12]
PC = MM[14]

1) PC_out, MAR_in, ALU/inc_right, Z_in, READ_MM
2) Z_out, PC_in, Y_in
3) IR_in, MDR_out

Decode/Execute)

4) ROM[8]_out, MAR_in, Y_in
5) PSW_out, MDR_in, WRITE_MM, ALU_Inc_2, Z_in
6) Z_out, MAR_in, Y_in
7) GPR[7]_out, MDR_in, WRITE_MM, ALU_Inc_2, Z_in
8) Z_out, MAR_in, Y_in, READ_MM,
9) MDR_out, PSW_in, ALU_Inc_2, Z_in,
10) Z_out, MAR_in, READ_MM
11) MDR_out, PC_in

Figure 5-1 Original Unoptimized Control Signals

Note: Wait Memory For Complete (WMFC) was originally needed but rendered unnecessary do to asynchronous memory operation from the project requirements