Intro to Computer Architecture and Organization

Project 3: Non-Pipelined Control Unit

By Drew Lenz and Andrew Schmidlin

Contents:

Objectives

The principle objective here is to demonstrate that you understand the basic operation of the control unit within a simple processor.

Specifications

- A word size of 16-bits
- Memory address/Data bus size of 16-bits
- Byte addressable memory
- 64K byte main memory
- A 16-bit program status word (PSW) with status bits. The first two bits are the condition code bits Z and N; these bits are conditionally controlled and denote the results of comparisons of the instruction result to the values zero (Z=1-equality to zero; N=1- less than zero). In addition, a third bit denotes execution in either of privileged or user mode (some operations are prohibited in user mode). The remaining bits control operations/constraints in the memory space that are not addressed in this project.
- 16 instructions, 2 of which can be executed only in privileged mode. Attempt to execute these 2 instructions in user mode will cause a program check violation.
- 8 16-bit General Purpose Registers (Reg). Register 0 (Reg[0]) always holds the value 0 no matter what value is assigned to it.
- A 16-bit program counter (PC) which is also Reg[7]
- A 16-bit count-down timer that causes a timer interrupt when it reaches zero provided the machine is executing in user mode. Timer interrupts are ignored when executing in privileged mode.
- 2’s complement number representation

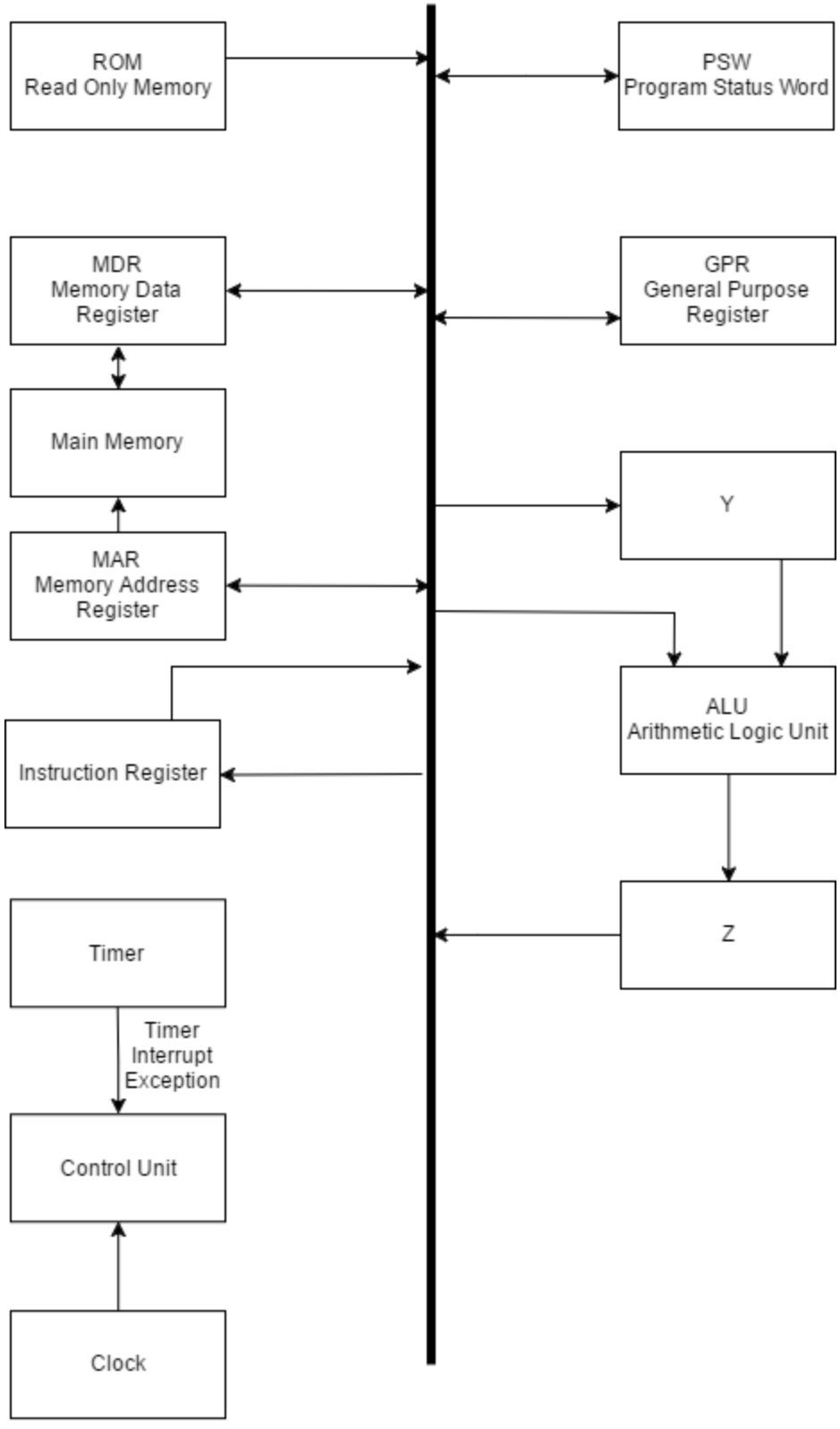

Figure 1: Data Path

ROM: Read Only Memory

The ROM is used to store constants that are used to point to specific points in memory used when reassigning and saving the PSW and PC when either a Program Check Violation or a Timeout exception occurs.

PSW: Program Status Word

The PSW is 16-bits including status bits. The first two bits are the conditionally controlled bits Z and N. They denote the results of comparisons of the instruction result to zero. Z=1 equality to zero. N=1 negative. The third bit is the P bit which controls whether execution is done in either privileged mode or user mode. Some operations cannot be executed while operating in user mode. Bits 4-16 control operations/constraints in memory space and will not be addressed in this report.

Clock

The clock outputs a pulse that causes states and sub-states to run.

Instruction Register & Sign Extender

The instruction register is a register that holds the instruction to be executed from the main memory. The sign extender is connected to the Instruction Register so the offsets do not need to be sent through the bus to be extended which allows for more efficient state equations. The sign extender extends Short_Offset from the 2-address instructions from 8 bits to 16 bits and extends Long_Offset from the 1-address instructions from 12 bits to 16 bits. The Short_Offset is extended by putting bit 8 of the instruction into bit 0 of the extended Short_Offset, and bits 9-14 of the extended Short_Offset correspond to 1-6 of the Short_Offset. Then bit 15 of the Short_Offset is extended to bits 7-15 of the extended Short_Offset. Long_Offset is extended in a similar manner with bits 4-14 corresponding to 0-10 then 11, 12, 13, 14, and 15 all taking on the value of the Long_Offsets 15th bit.

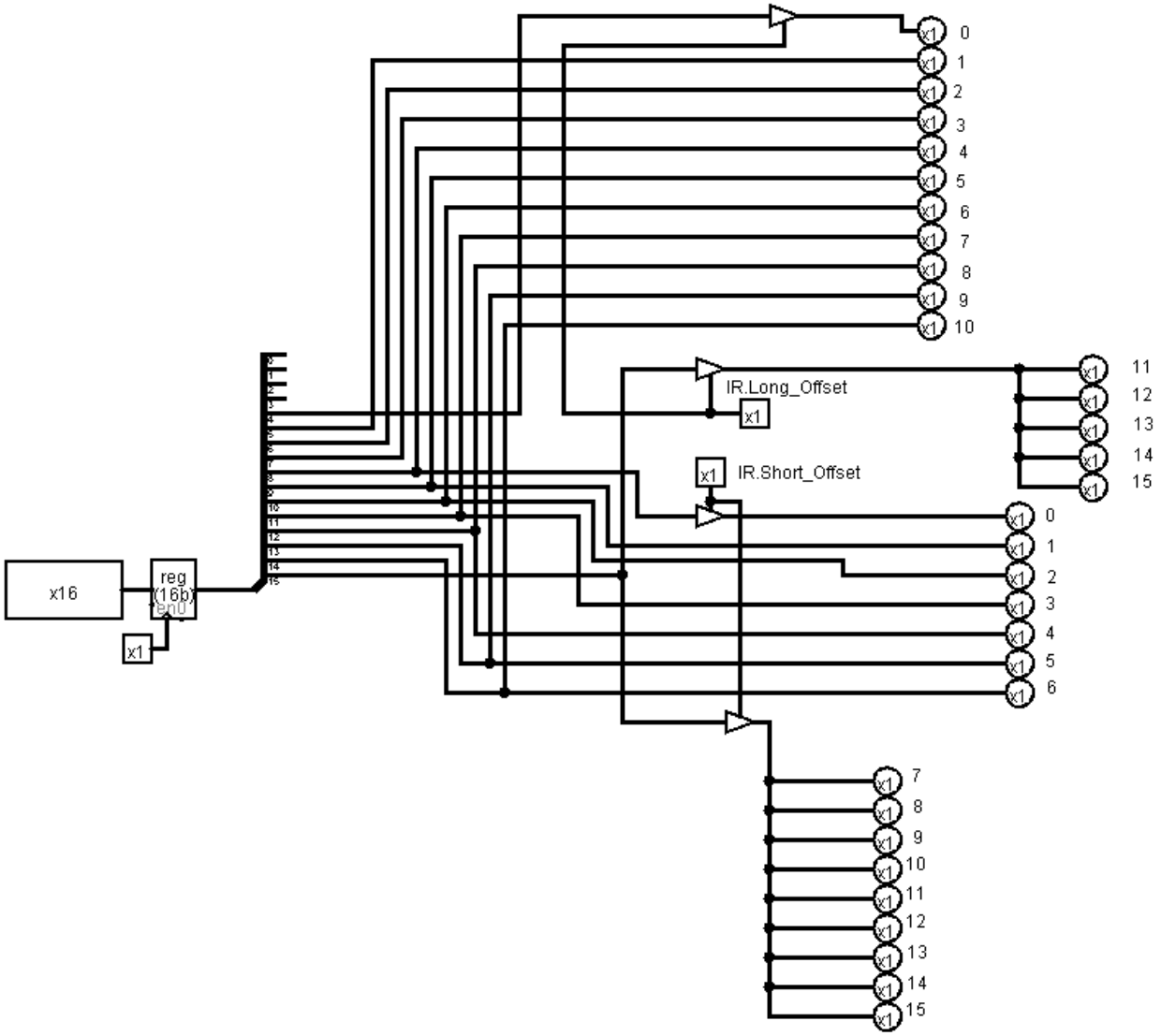

Figure 2: Sign Extender Circuit

Register

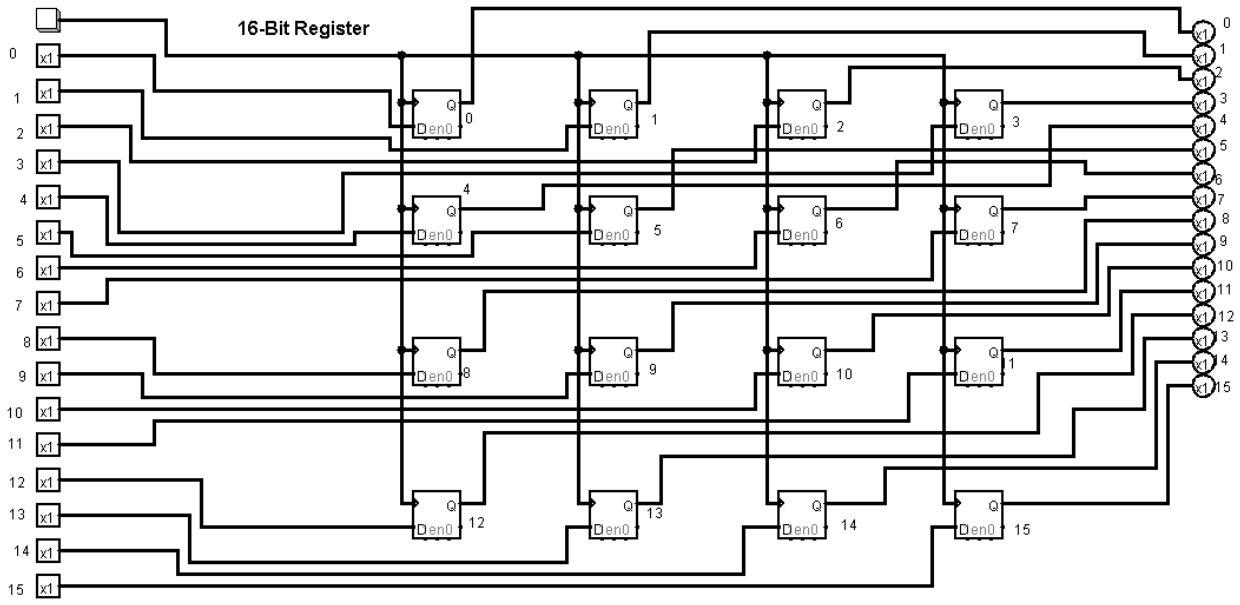

Figure 3: 16-Bit Register

GPR: General Purpose Register

The GPR contains 8 registers that hold 16 bits each. The registers are accessed through a demultiplexer with an enable coming from the control unit. When this enable coming from the control unit is off it allows for the PC which is held in register 7 to be incremented during each fetch cycle. The PC will be incremented on each clock cycle, and the registers will be updated on each clock cycle. The register being either written to or read from will depend on the selector bits going into the multiplexer from the control unit. The inputs going into the multiplexer that then signify which register to read from or write to via the come from the Instruction Register and correspond to Rd(3-Address), RS1, RS2, Rd2(2-Address), Reg7(PC), and Reg0(0).

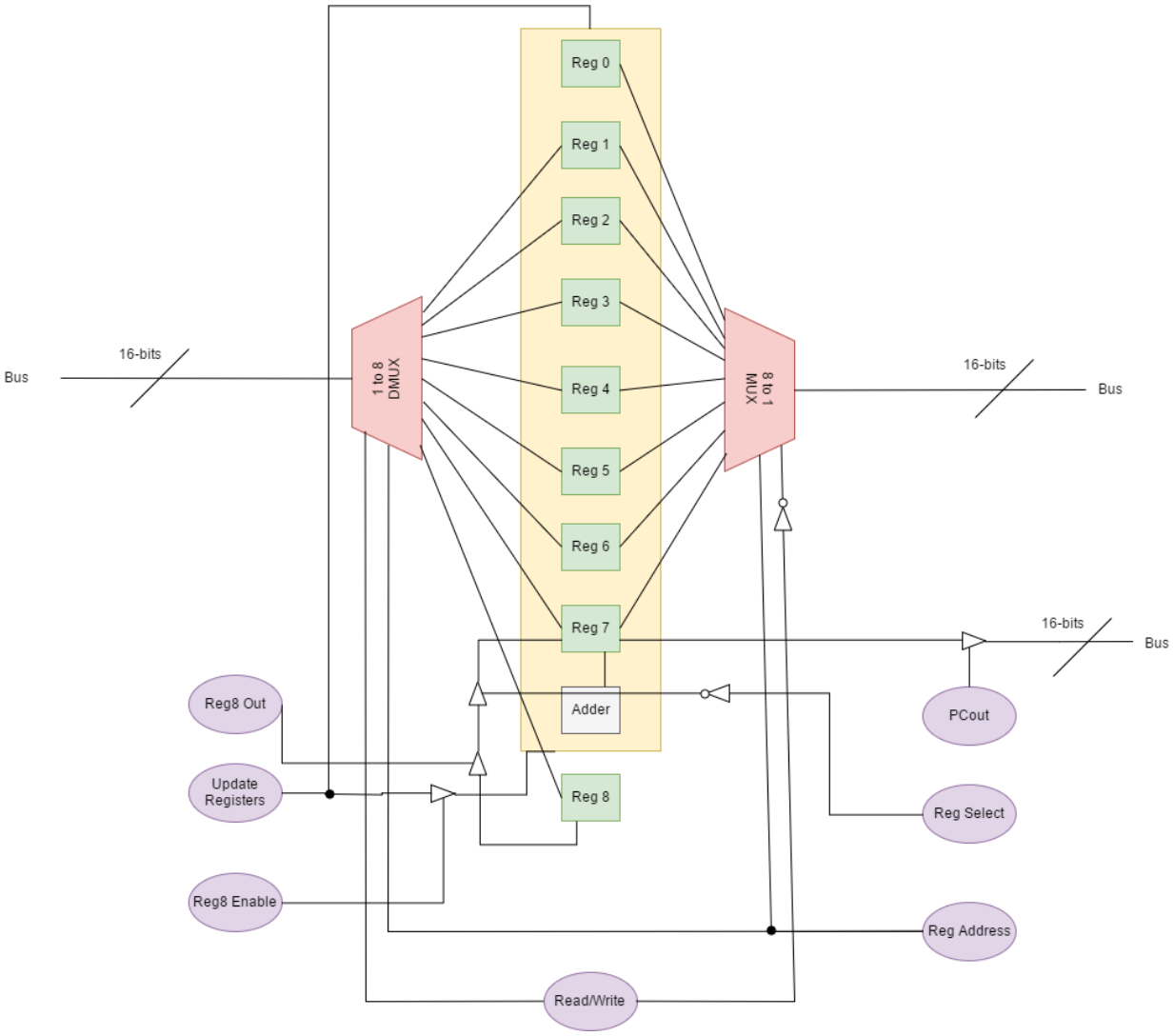

Figure 4 Register File Diagram

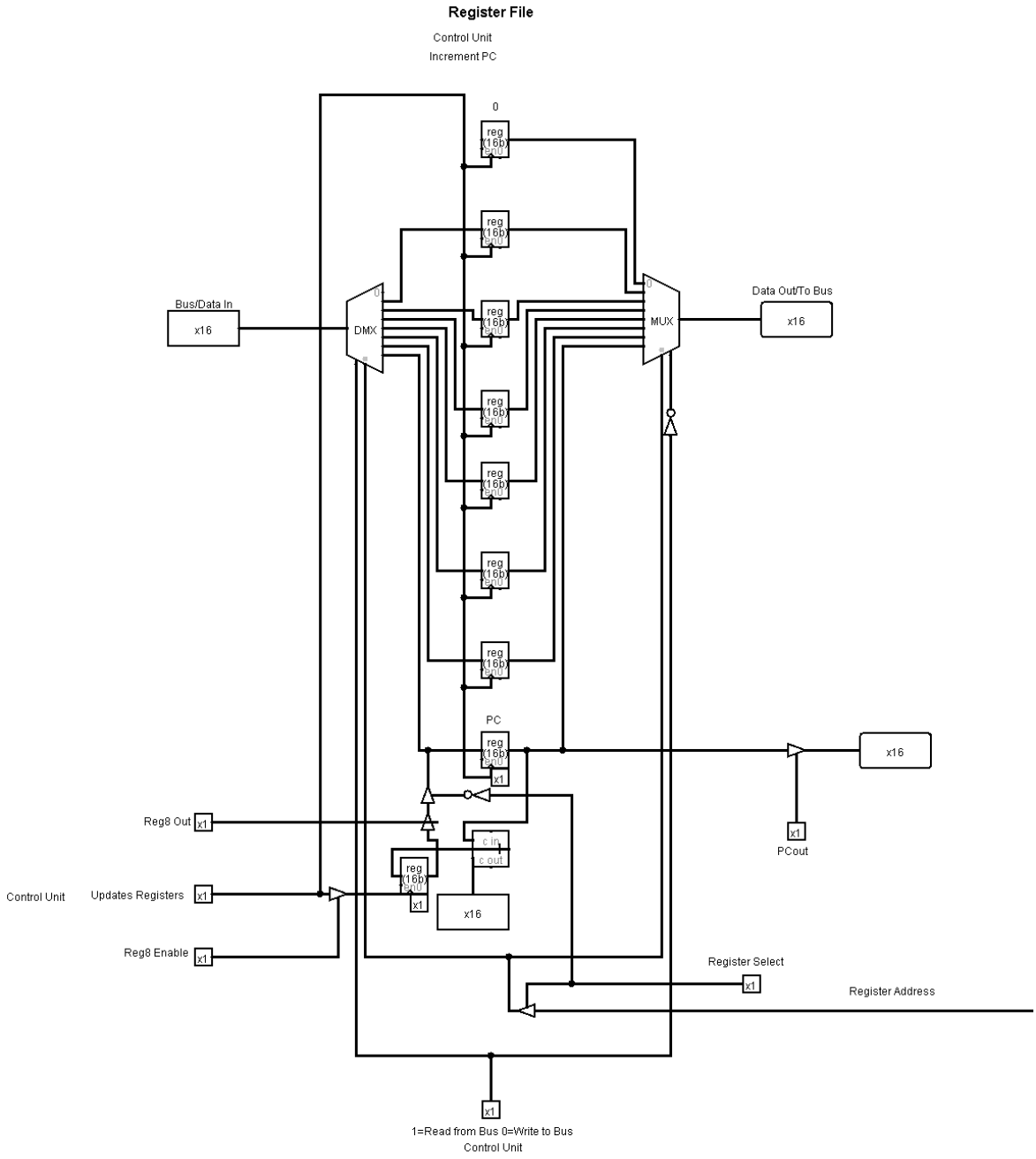

Figure 5 Register File Circuit

Main Memory

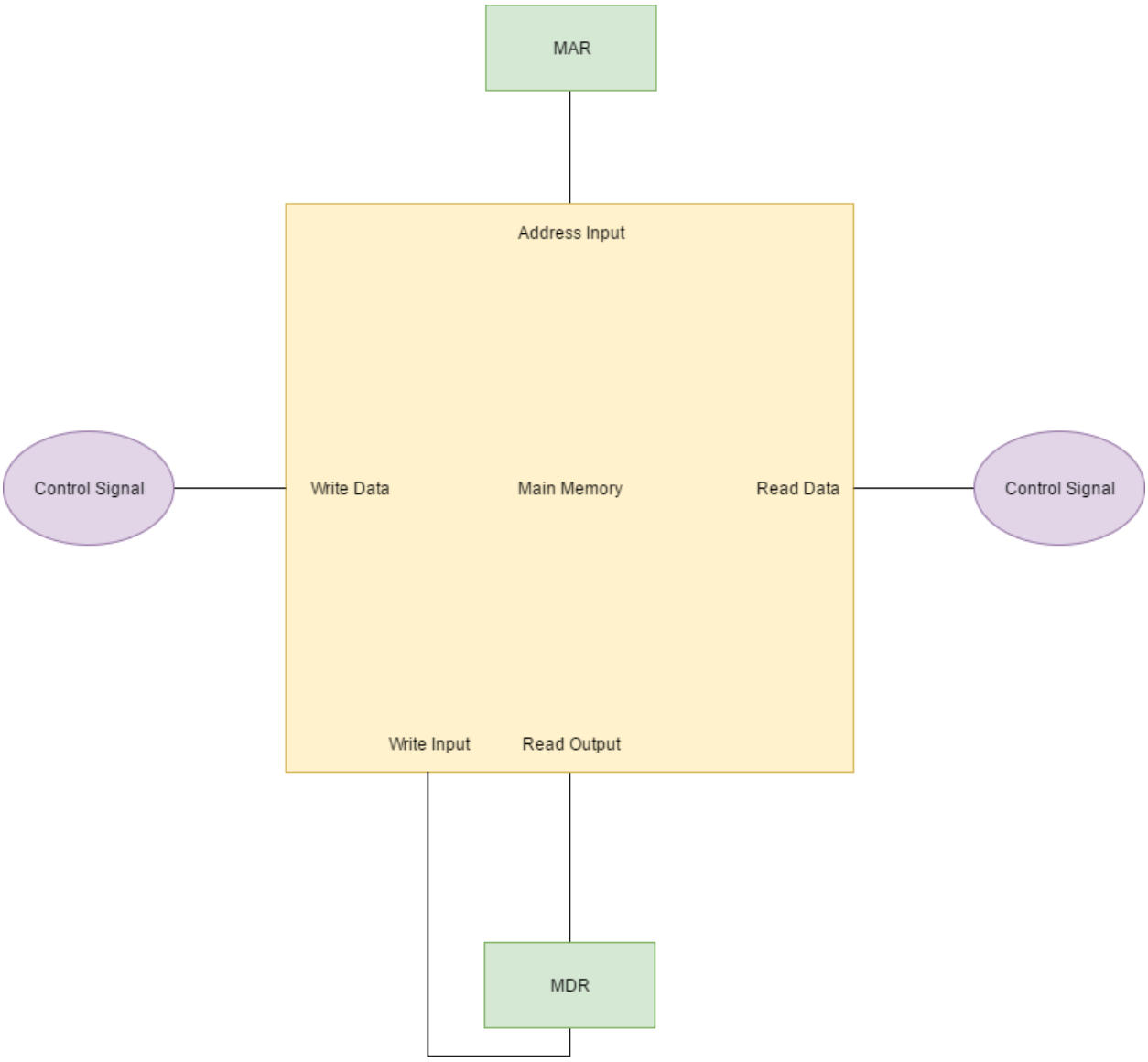

Figure 6 Main Memory Diagram

MAR: Memory Address Register & MDR: Memory Data Register

The Memory Address Register and Memory Data Register acts somewhat as an extension of main memory. The MAR holds the address of the desired data contained within Main Memory. The MDR holds the output of Main Memory after the memory function completes and holds data that is going to be written to Main Memory.

Y & Z

Y and Z act as registers for the ALU. Y can take inputs from the bus and send outputs to the bus and the ALU. Z can only take inputs from the ALU and send outputs to the bus.

ALU

The ALU consists of multiple smaller circuits that all go into a multiplexer that allows the control unit to decide which function it would like the result of. The ALU is capable of addition and subtraction via a carry lookahead adder/subtractor, logical AND, OR, NOT, Arithmetic Shift Right, and Logical Shift Left. The carry lookahead adder differentiates between addition and subtraction via 2's complement of the second input.

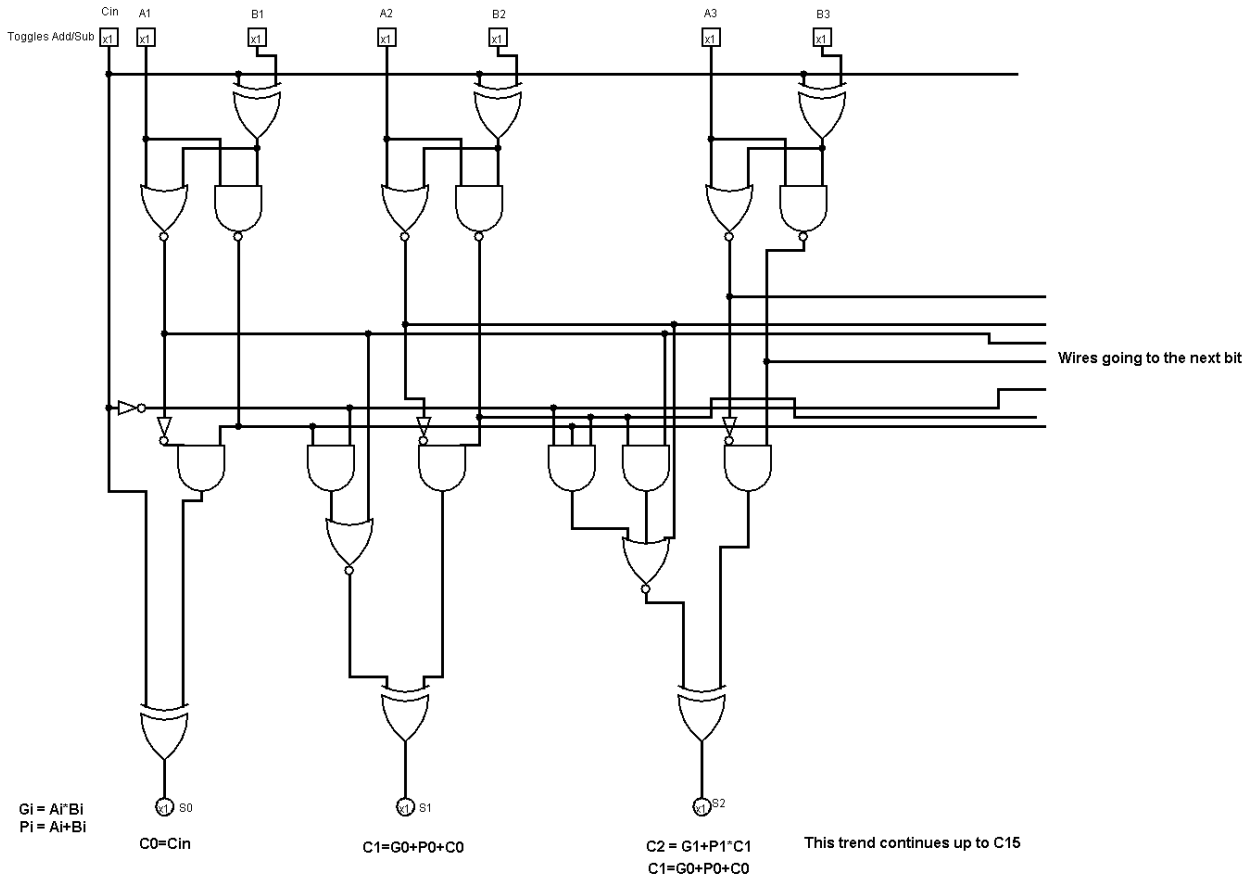

Figure 7 3-Bits of Carry Lookahead Adder

The carry lookahead adder works on an increasingly complex equation as the number of bits being computed increases. The equation is $C_{i+1} = G_i + P_i C_i$ where $G_i = x_i y_i$ and$P_i = x_i + y_i$. The adder does subtraction by having an input trigger all of the second inputs values XOR. This input also adds an extra one at the end to complete the 2's complement.

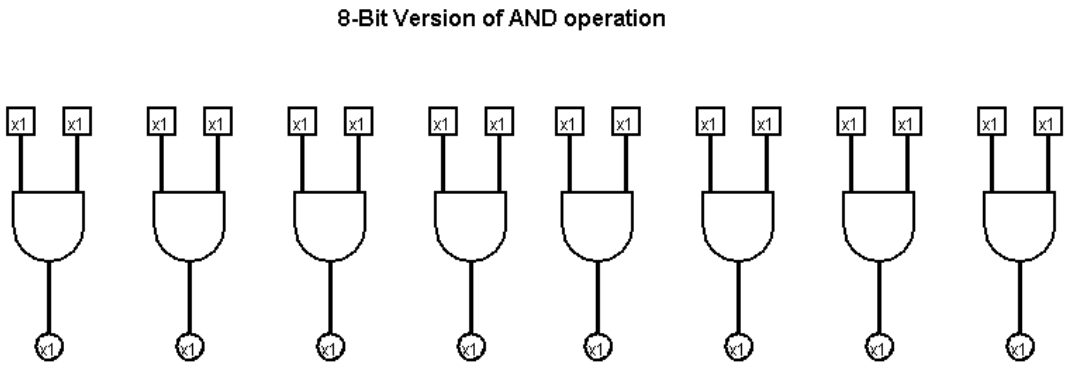

Figure 8 Logical AND

The version of logical AND displayed is a version simplified down to 8-bits instead of 16. The bits on the left side of each AND gate are from the first input and the bits on the right side of each AND gate are from the second input.

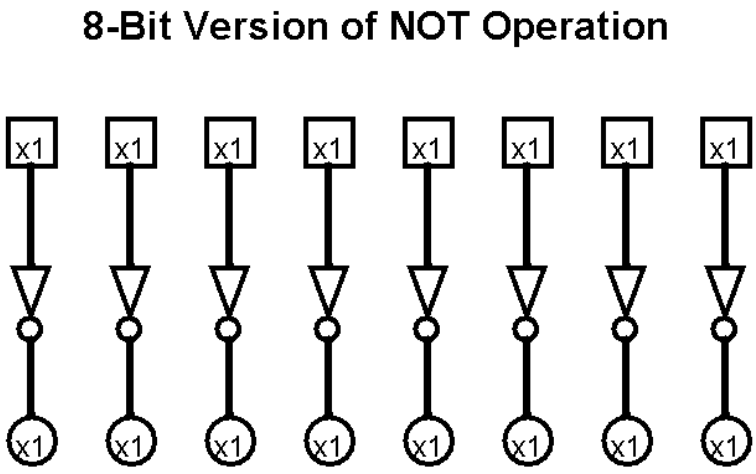

Figure 9 Logical NOT

The version of logical NOT displayed is a version simplified down to 8-bits instead of 16. The bits shown going into the NOT gates are from the first input only because the NOT operation does not use the second input in its computation.

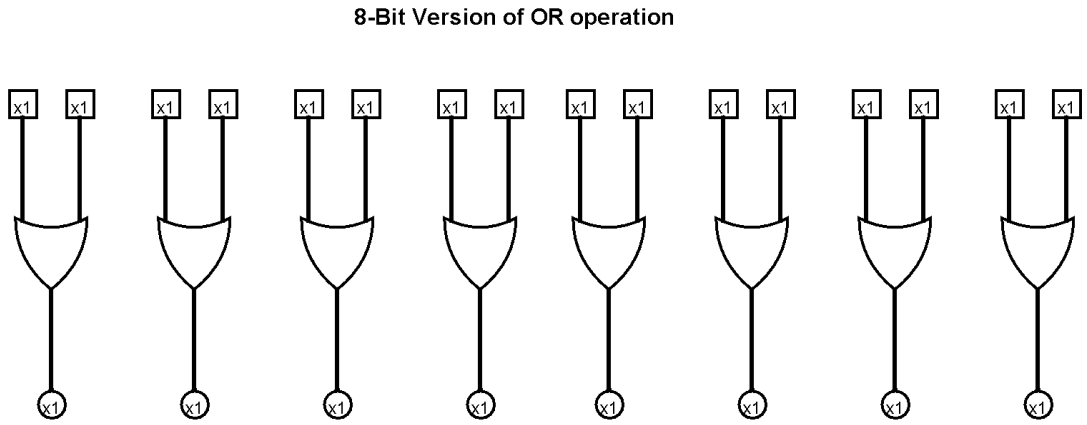

Figure 10 Logical OR

The version of logical OR displayed is a version simplified down to 8-bits instead of 16. The bits on the left side of each OR gate are from the first input and the bits on the right side of each OR gate are from the second input.

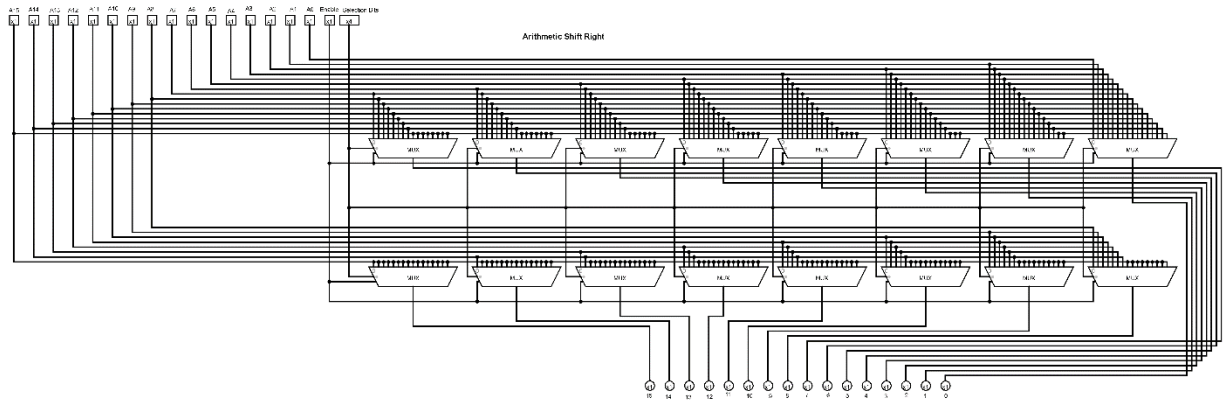

Figure 11 Arithmetic Shift Right

The Arithmetic Right Shift circuit shown above implements an Arithmetic Right Shift via 16 multiplexers. This allows the number of bits shifted to be selected. The selection bits decide which multiplexer to draw from. The most significant bit from the input will match the most significant bit of the output and that makes the shift arithmetic instead of logical. Input two is not used in this operation.

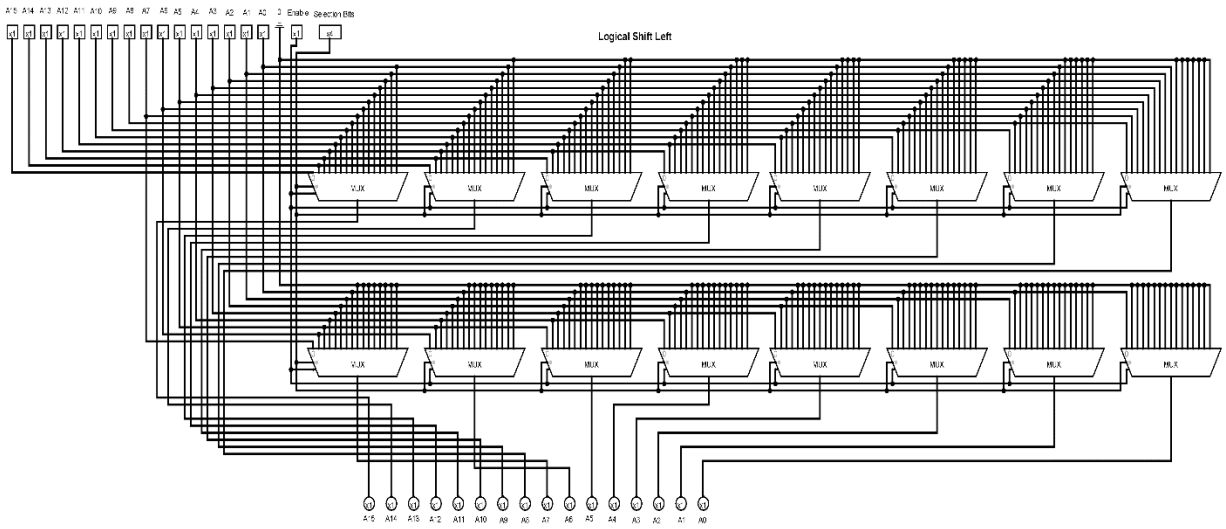

Figure 12 Logical Shift Left

The Logical Shift Left circuit implements a Logical Shift Left using 16 multiplexers with each possible output. A selection input then decides which output to use. The empty bits created by shifting the input left are filled in with zeros.

Finite State Automata

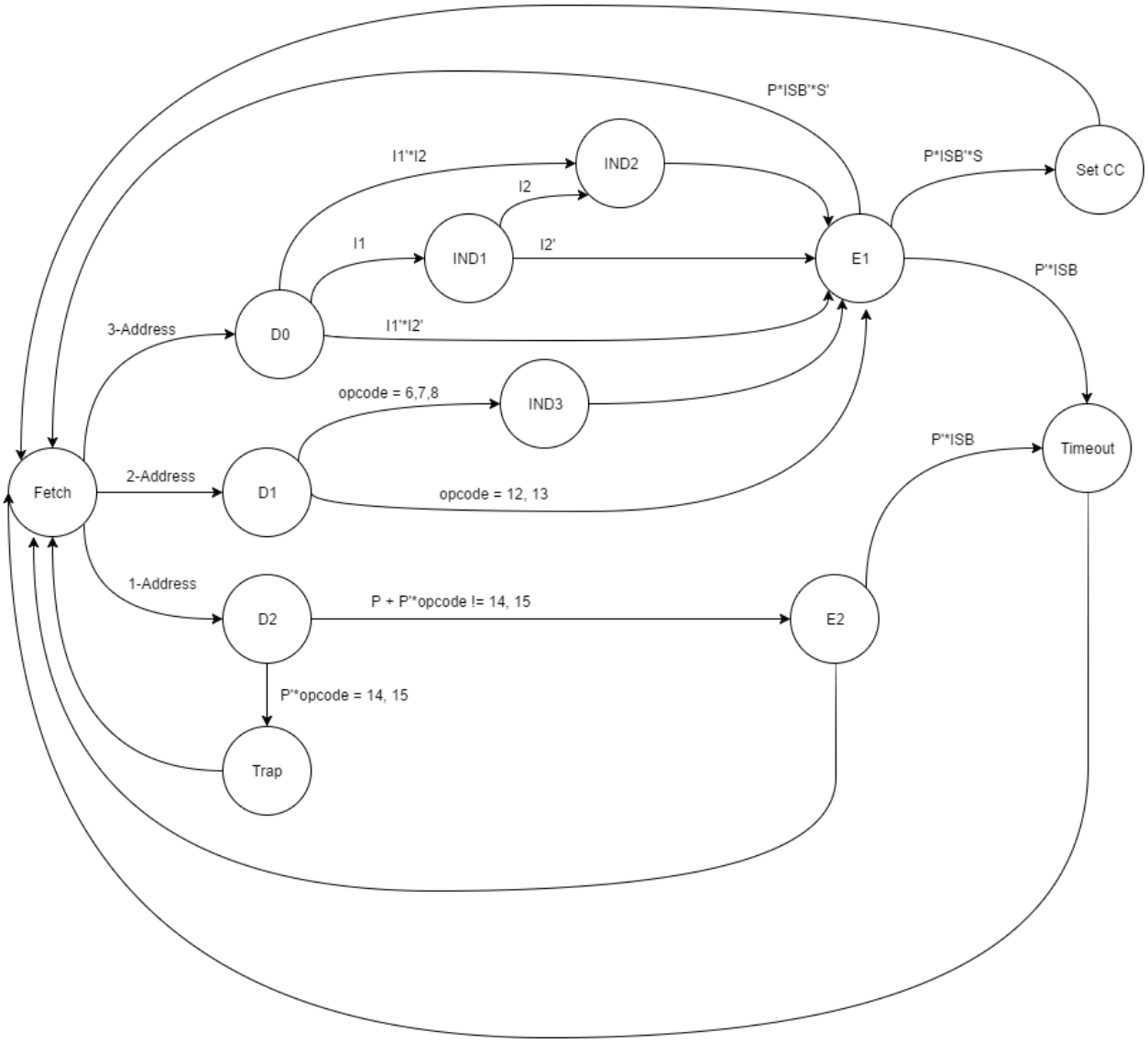

Figure 13: Finite State Machine

The finite state machine for the control unit designed in this lab is shown in Figure 13 above. The control unit starts in the fetch state. During the fetch state, an instruction is fetched from the main memory and placed in the instruction register. Next, the control unit enters one of 3 decode states depending on how many addresses exist in the fetched instruction. During the decode states, the instruction is prepared to be executed. After the decode states, some instructions require indirect states, as noted by the I1 and I2 bits in a 2 or 3-address instruction or an opcode of 6, 7, or 8 for a single address instruction. Once all decode and indirect states are finished, the control unit executes the instruction. After executing a 2 or 3-address instruction, the control unit can set condition code, timeout, or it can fetch the next instruction. The condition code is set if the instruction's S bit is 1 and the program did not timeout. A

timeout occurs when the system's countdown timer reaches 0 while executing in user mode (the P bit of the program status word is 0). This timeout state exists to interrupt the control unit and modify the main memory, program counter, and program status word. If neither of these conditions were met, the control unit fetches the next instruction. The execute state for a single address instruction does not have a set condition code state. The control unit also has a trap state. The trap state is activated if the privileged instructions CLK or LPSW are decoded and the program is in user mode. During the trap state, the PC and PSW are swapped.

Next State Table

The following tables show the state equations used to cycle through the states of the state machine. The equations used to set values for timeout (TO) and trap (pgm.chk) are shown above the tables.

$$\bar{P} * ISB = TO \qquad pgm.chk = \bar{P} * opcode = 14{,}15$$

Code $Q_3Q_2Q_1Q_0$	State $Q_3^*Q_2^*Q_1^*Q_0^*$	Equation
0000	Fetch	$E1 * \overline{TO} * \bar{S} + set.CC + Timeout + E2 + Trap$
0001	d0	$fetch * 3Address$
0010	d1	$fetch * 2Address$
0011	d2	$fetch * 1Address$
0100	Ind1	$d_0 * I_1$
0101	Ind2	$d_0 * \overline{I_1} * I_2 + Ind1 * I_2$
0110	Ind3	$d_1 * opcode = 6{,}7{,}8$
0111	E1	$Ind2 + IND1 * \overline{I_2} + d_0 * \overline{I_1} * \overline{I_2} + Ind3 + d_1 * opcode = 12{,}13$
1000	E2	$d2 * (P + (\bar{P} * opcode\ != 14{,}15)\ \text{->}\ d2 * P + d2 * \bar{P} * opcode 9{,}10{,}11$
1001	Set.CC	$E1 * \overline{TO} * S$
1010	Timeout	$(E1 + E2) * TO$
1011	Trap	$d2 * Pgm.ck$

Table 1: Previous State/Next State Table

Code $Q_3Q_2Q_1Q_0$	State $Q_3^*Q_2^*Q_1^*Q_0^*$	Equation
0000	Fetch	$\overline{Q_3} * Q_2 * Q_1 * Q_0 * \overline{TO} * \bar{S} + Q_3 * \overline{Q_2} * \overline{Q_1} * Q_0 + Q_3 * \overline{Q_2} * Q_1 * \overline{Q_0} + Q_3 * \overline{Q_2} * \overline{Q_1} * \overline{Q_0} + Q_3 * \overline{Q_2} * Q_1 * Q_0$
0001	d0	$\overline{Q_3} * \overline{Q_2} * \overline{Q_1} * \overline{Q_0} * 3Address$
0010	d1	$\overline{Q_3} * \overline{Q_2} * \overline{Q_1} * \overline{Q_0} * 2Address$
0011	d2	$\overline{Q_3} * \overline{Q_2} * \overline{Q_1} * \overline{Q_0} * 1Address$
0100	Ind1	$\overline{Q_3} * \overline{Q_2} * \overline{Q_1} * Q_0 * I_1$
0101	Ind2	$\overline{Q_3} * \overline{Q_2} * \overline{Q_1} * Q_0 * \overline{I_1} * I_2 + \overline{Q_3} * Q_2 * \overline{Q_1} * \overline{Q_0} * I_2$
0110	Ind3	$\overline{Q_3} * \overline{Q_2} * Q_1 * \overline{Q_0} * opcode = 6{,}7{,}8$
0111	E1	$\overline{Q_3} * Q_2 * \overline{Q_1} * Q_0 + \overline{Q_3} * Q_2 * \overline{Q_1} * \overline{Q_0} * \overline{I_2} + \overline{Q_3} * \overline{Q_2} * \overline{Q_1} * Q_0 * \overline{I_1} * \overline{I_2} + \overline{Q_3} * Q_2 * Q_1 * \overline{Q_0} + \overline{Q_3} * \overline{Q_2} * Q_1 * \overline{Q_0} * opcode = 12{,}13$
1000	E2	$\overline{Q_3} * \overline{Q_2} * Q_1 * Q_0 * (P + (\bar{P} + opcode\ != 14{,}15)\ \text{->}\ \overline{Q_3} * \overline{Q_2} * Q_1 * Q_0 * (P + \bar{P} * opcode 9{,}10{,}11)$
1001	Set.CC	$\overline{Q_3} * Q_2 * Q_1 * Q_0 * \overline{TO} * S$
1010	Timeout	$Q_3 * \overline{Q_2} * \overline{Q_1} * \overline{Q_0} * TO + \overline{Q_3} * Q_2 * Q_1 * Q_0 * TO$
1011	Trap	$\overline{Q_3} * \overline{Q_2} * Q_1 * Q_0 * Pgm.ck$

Table 2: Previous State Next State Table in terms of Q

Control Unit

The control unit keeps track of what state the processor is in and sends out control signals for each state. The control unit keeps track of where the processor is in its processing via a state machine. Our state machine is broken down hierarchically. It consists of 12 high level states; Fetch, Decode 0 (d0), Decode 1 (d1), Decode 2 (d2), Indirect 1 (Ind1), Indirect 2 (Ind2), Indirect 3 (Ind3), Execute 1 (E1), Execute 2 (E2), Set.CC, Timeout, and Trap. Each of these states has sub-states. It is during the substates that the control signals for each state are sent to the rest of the processor. For example Fetch's control signal is

1. PC_{out}, MAR_{in}, read_MM, adder_add, $Reg8_{in}$
2. WMFC
3. MDR_{out}, IR_{in}, $Reg8_{out}$, PC_{in}

So Fetch will have 3 sub-states to its high level state. When applied to all of the high level states we have 55 states.

Control Unit State Machine:

The control unit in this processor has 3 main parts. The first of these is a state machine. The state machine uses the current state's $Q_3Q_2Q_1Q_0$ values and some additional conditions to determine what the next state is. The logic used for determining each state is consistent with the state equations shown in Table 2.

The first portion of the state machine determines if the next state is fetch. It consists of multiple AND gates connected to an OR gate that check the current state as well as the S bit of an instruction and the timeout conditions to determine if fetch is the next state. The logic for the gates is consistent with the state equation described in Table 2. The gate level implementation of this state is shown below in Figure 14.

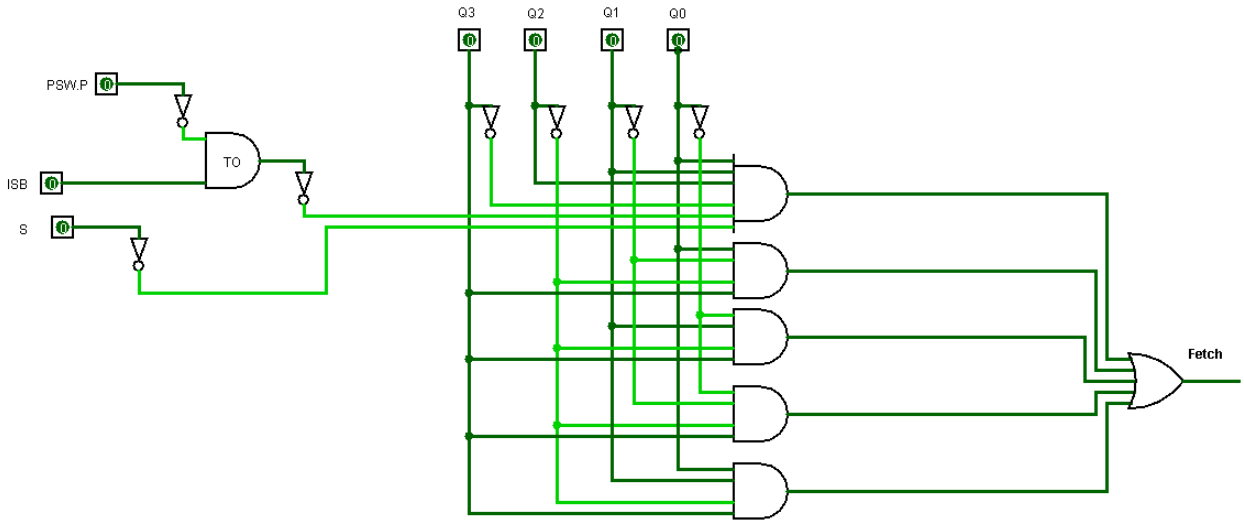

Figure 14: Fetch Portion of State Machine

The next portion of the state machine determines if the next state is one of the decode states. This portion first checks if the current state is fetch where $Q_3Q_2Q_1Q_0 = 0000$. If this is true, a positive value is passed to a demultiplexor that uses the opcode from the instruction register as a selection bit. If the opcode is 0, 1, 2, 3, 4, or 5, the instruction is a 3-address instruction and the next state is D0. If the opcode is 6, 7, 8, 12, or 13, the instruction is a 2-address instruction and the next state is D1. Lastly, if the opcode is 9, 10, 11, 14, or 15, the instruction is a 1-address instruction and the next state is D2. The gate level implementation of this state is shown below in Figure 15.

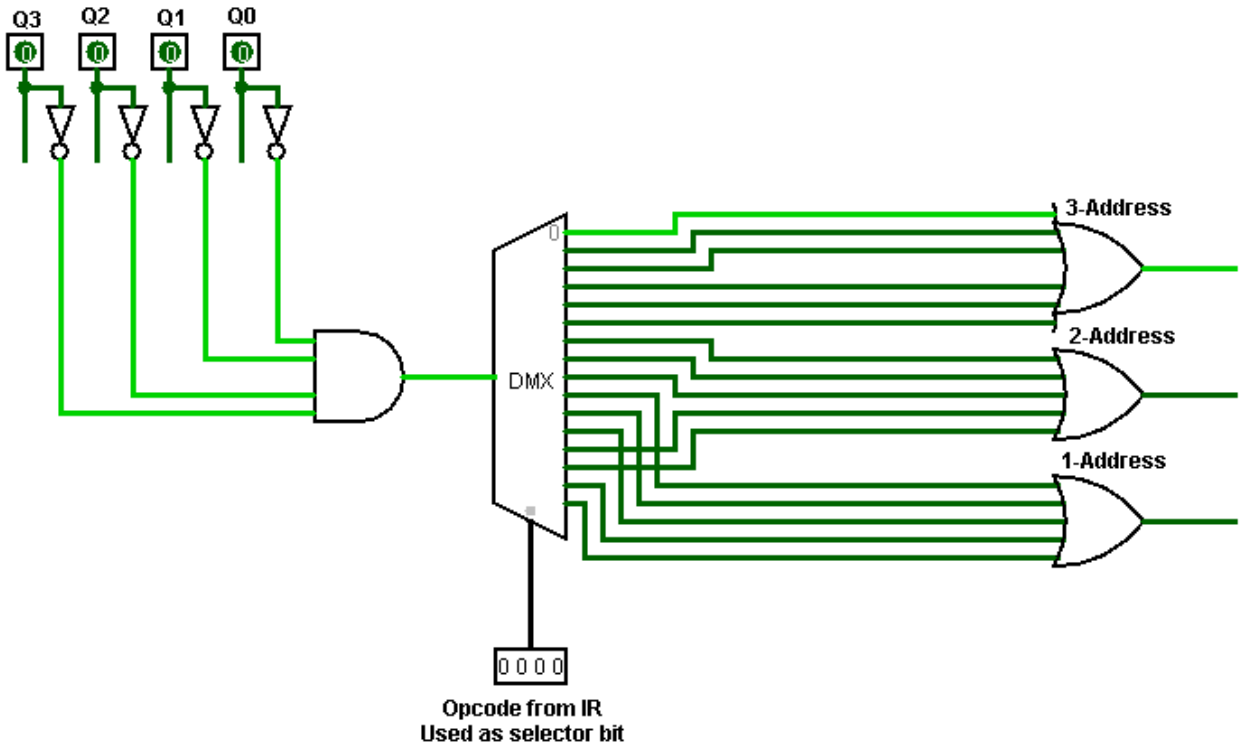

Figure 15: Decode Portion of State Machine

The next portion of the state machine determines if the next state is one of the indirect states. To determine if the next state is IND1, an AND gate is used to determine if the current state is D0 and the I1 bit of an instruction is 1. If this is true, the next state is IND1. To determine if the next state is IND2, one AND gate are used to determine if the current state is D0 and the I2 bit of an instruction is 1 and another AND gate is used to determine if the current state is IND1 and the I2 bit is 1. An OR gate is used to set the next state to IND2 if either of these results is true. Finally, this portion of the state machine can determine if the next state is IND3. An AND gate is used to check if the current state is D1 and the instruction's opcode is 6, 7, or 8. To determine if the opcode is 6, 7, or 8, the instruction's opcode is used as a selector bit in a demultiplexer that outputs 1 at those bits. If both conditions are met, the next state is set to IND3. The gate level implementation of this state is shown below in Figure 16.

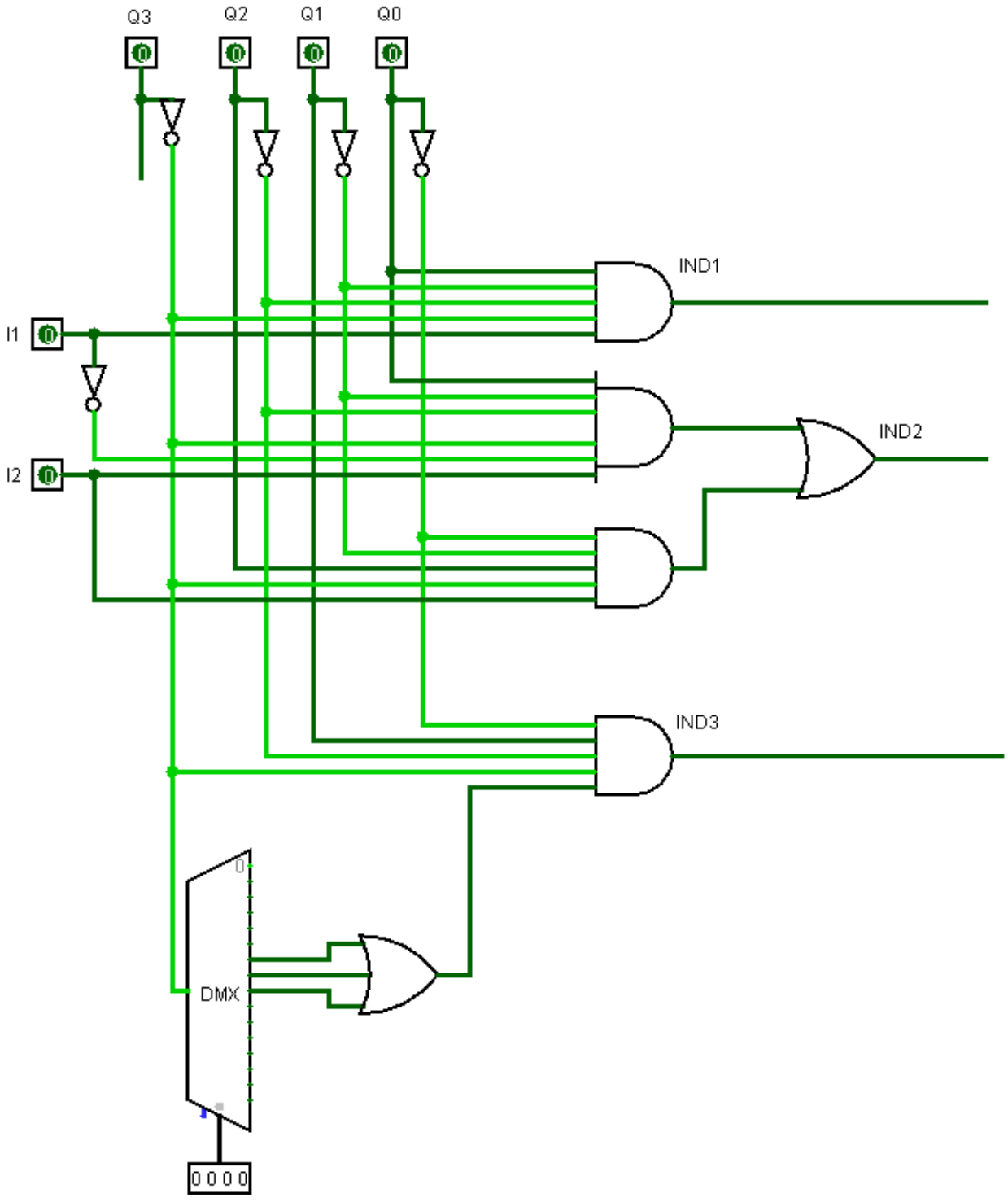

Figure 16: Indirect Portion of State Machine

The next portion of the state machine determines if the next state is E1. It consists of several AND gates connected to an OR gate to determine the current state as described in Table 2. The next state will be set to E1 if the current state is IND2, IND1 and IND2 = 0, D0 and IND1 and

IND2 = 0, IND3, or D1 and the opcode = 12 or 13. If any of these conditions is true, the next state is set to E1. The gate level implementation of this state is shown in Figure 17 below.

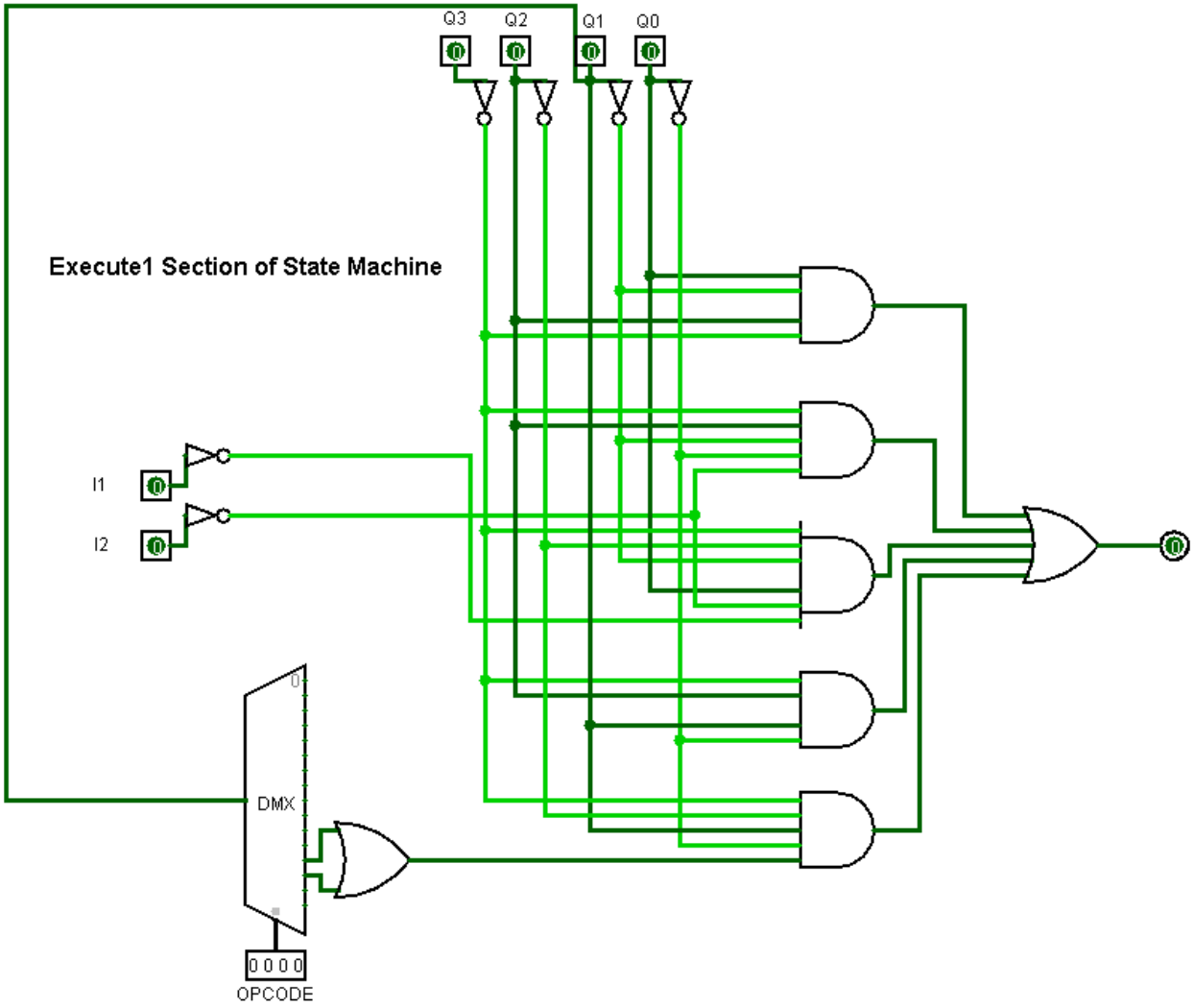

Figure 17: E1 Portion of State Machine

The next portion of the state machine determines if the next state is E2. This checks if the current state is D2 and a user isn't attempting to execute a privileged instruction when not in privileged mode. To determine the user is trying to execute a valid instruction, the P bit of the PSW is checked. If P = 1, the next state if E2. If P = 0 and the opcode is not 14 or 15, the next state is also E2. The gate level implementation of this state is shown in Figure 18 below.

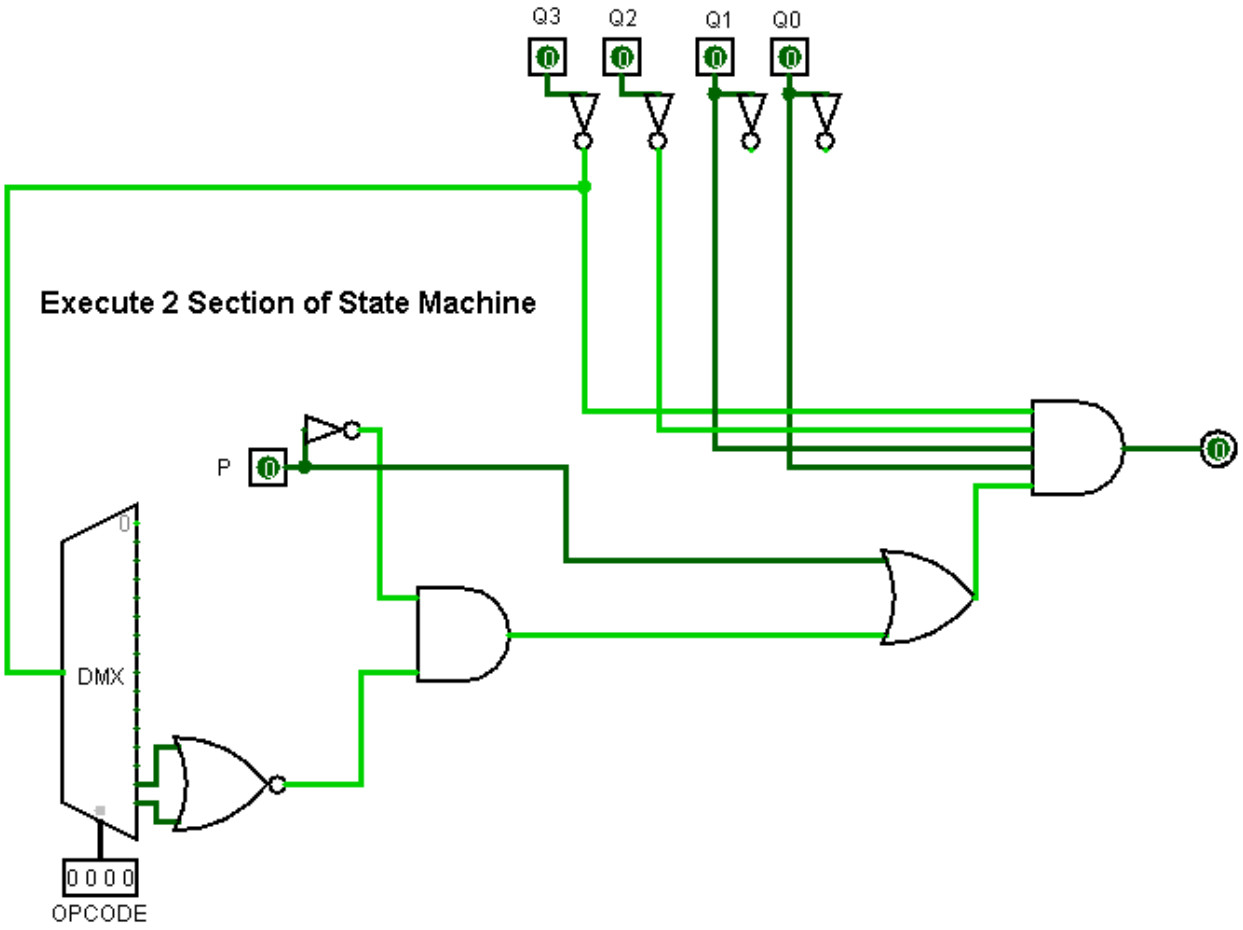

Figure 18: E2 Portion of State Machine

The final portion of the state machine can check if the next state should be set condition code, timeout, or trap. To determine if the next state is set condition code, an AND gate checks to see if the current state is E1, the processor's timeout conditions are not met, and the instruction's S bit is 1. To determine if the next state is timeout, an OR gate checks if the current state is E1 and the processor's timeout conditions are met or if the current state is E2 and the processor's timeout conditions are met. To determine if the next state is trap, an AND gate checks to see if the current state is D2 and a program check violation has occurred. A program check violation is triggered when the P bit of the program status word is 0 and the user is trying to execute the privileged instructions with opcodes 14 or 15. If both conditions are met, the next state is set to trap. The gate level implementation of this state is shown in Figure 19 below.

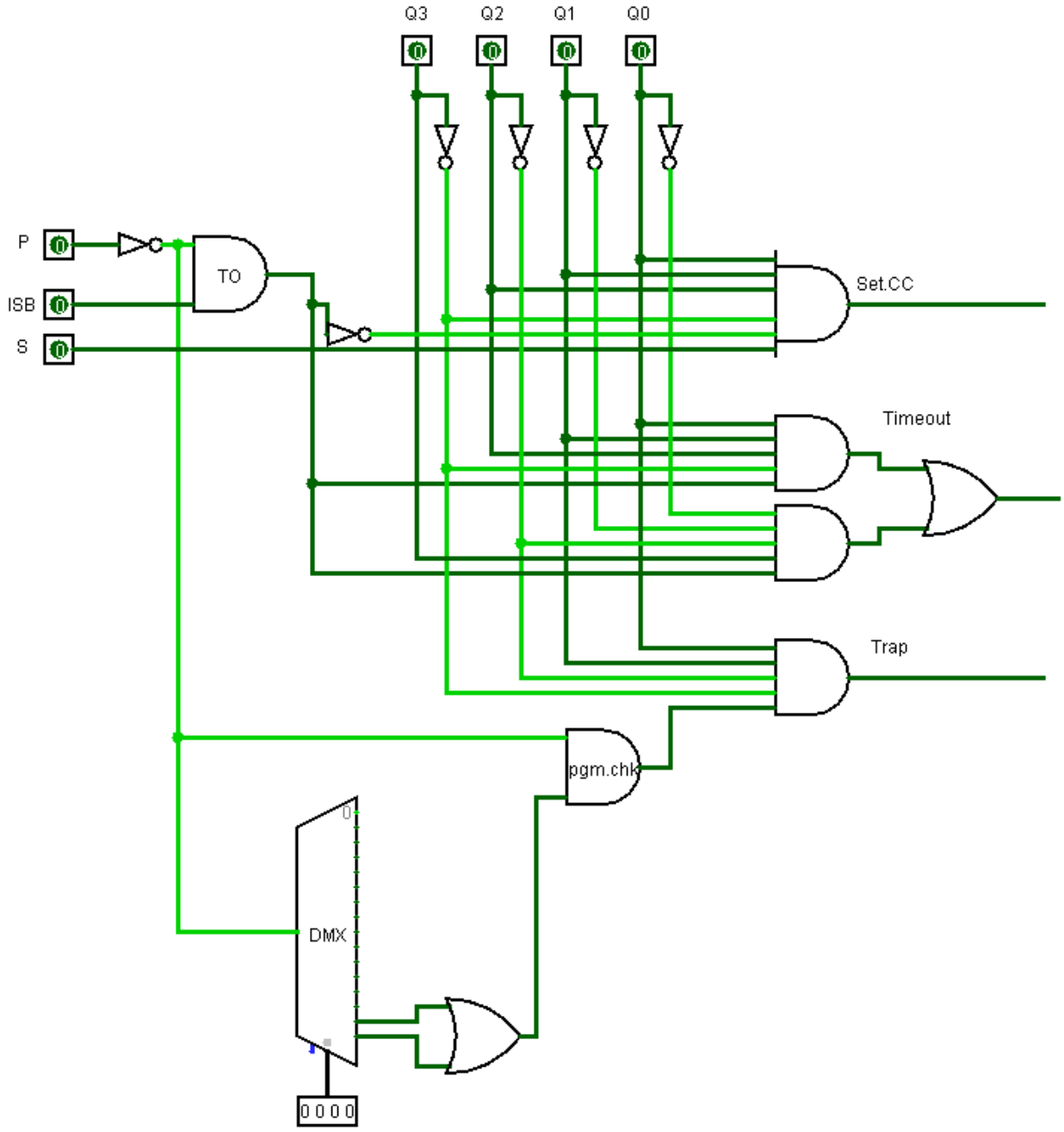

Figure 19: Set Condition Code, Timeout, and Trap Portions of State Machine

Control Unit Intermediate Stage:

The next part of the control unit is an intermediate stage between the state machine and the substates. The gate level implementation of this stage is shown in Figure 20.

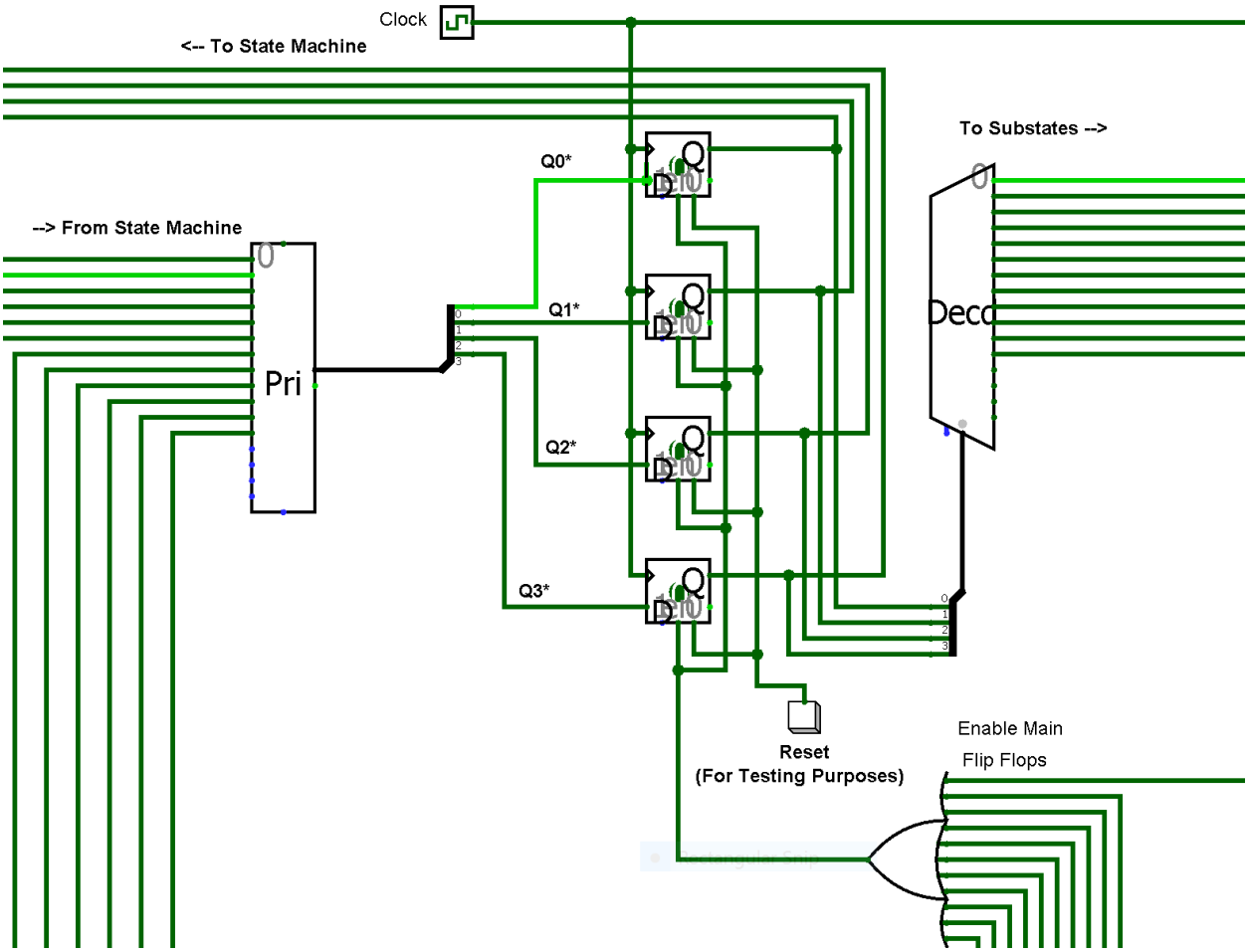

Figure 20: Intermediate Stage of Control Unit

The intermediate stage begins with a priority encoder which expresses the next state as the 4-bit value $Q_3^* Q_2^* Q_1^* Q_0^*$. The single bit output from each portion of the state machine is connected to this encoder. The next state value will be stored in D flip flops that are enabled only when a substate has finished. The signal used to enable the D flip flops is very important because it prevents the control unit from entering the next state too early and sending conflicting control signals to the rest of the processor. The enable flip flops signal is sent during the last set of control signals of the active substate. When the D flip flops are activated, the Q values for the now current state are sent back to the state machine to determine the next state. The Q values are also sent to 4-bit to 16 single bit output decoder where each output bit corresponds to a high-level state. When a bit is sent out of this decoder, it is used to activate the appropriate substate circuit which will send control signals to the rest of the processor.

Control Signals

Before describing the substate portion of the control unit, it is important to explain what a control signal is and to explain the control signals that are sent by this processor. Control signals are sent by the control unit to various components of a processor. These control signals are used to execute instructions. The various control signals of this processor are as follows.

In Signals – These signals are wired to the clock/enable bits of various registers in the processor. They are used to store the value on the bus in the specified register.

Out Signals – These signals are wired to tristate buffers coming off various registers in the processor. They are used to put the value stored in the register on the bus.

GPR Control signals:

1. $GPR[address]_{in}$ – This control signal is used to write the value on the bus to the GPR register noted by the address in brackets. When this control signal is sent, a couple smaller control signals are sent to the GPR. The first signal turns register select on. This activates a tri-state buffer that allows an address to be sent to the GPR's write demultiplexer. The next signal is the 3-bit value of the address that data will be written to. This 3-bit value will be determined in the control unit and it changes depending on which bits in the instruction describe are used for addressing. The next signal sends a read signal. This signal enables the demultiplexer to pass the value coming in on the bus to the desired register. Lastly, an update register signal is passed to the clock of each register. This allows each register to read a new value. These control signals occur simultaneously.
2. $GPR[address]_{out}$ – This control signal is used to place the value at the passed address to the bus. When this control signal is sent, two control signals are sent to the GPR. The first signal is turns register select on. The other signal is the address as a 3-bit value. These control signals are sent simultaneously.
3. PC_{out} – This control signal activates a tri-state buffer attached to the program counter. This signal is used to put the value stored in the PC directly on the bus.
4. $Reg8_{enable}$ – This control signal activates the ninth register, which is connected to an adder coming off the PC which is used to increment the PC during the fetch cycle. This signal must be sent when a value is to be stored or read from this register.
5. $Reg8_{out}$ – This signal is used to activate a tri-state buffer that outputs an incremented PC value into the PC.

Main Memory Control Signals:

1. MAR_{in} – This control signal activates the memory address register to store the value currently on the bus.
2. MDR_{in} – This control signal activates the memory data register to store the value currently on the bus.

3. MDR_{out} – This control signal activates a tri-state buffer to place the value in the MDR on the bus.
4. Read_MM – This control signal tells main memory to read the value stored at the address indicated by the MAR. The value is then stored in the MDR.
5. Write_MM – This control signal tells main memory to write the value stored in the MDR into the address indicated by the MAR.
6. WMFC – Wait for memory function to complete. This signal is not actually sent to any component in the processor. However, it will be included in instructions to indicate the pause necessary when reading to or writing from main memory. It is assumed that the memory function in this processor lasts for only one clock cycle.

ALU Signals – These signals send 3-bit instructions to the ALU to indicate what operation the ALU should perform. The ALU signals are wired to an 8 input to 3-bit priority encoder as described in Table 3 below.

Signal	3-bit Output
ALU/add	000
ALU/subtract	001
ALU/and	010
ALU/or	011
ALU/not	100
ALU/shift_right	101
ALU/shift_left	110

Table 3: ALU Control Signals

Control Signal Instructions:

The following control signals are sent from the control unit during the substates of a high-level state. Each line of instructions is executed during one clock cycle. The instructions detailed below have been optimized.

Fetch Control Signal Instructions:

1. PC_{out}, MAR_{in}, Read_MM, $Reg8_{enable}$, Y_{in}
2. WMFC
3. MDR_{out}, IR_{in}, $Reg8_{enable}$, $Reg8_{out}$, Update Registers

D0 Control Signal instructions:

1. If $I_1 = 0$:
    a. $GPR[IR.Rs1]_{out}$, $OP1_{in}$
2. If $I_2 = 0$:
    a. $GPR[IR.Rs2]_{out}$, $OP2_{in}$

D1 Control Signal Instructions:

1. IR.Short_Offset$_{out}$, ALU/add, Z$_{in}$

D2 Control Signal Instructions:

1. IR.Long_Offset$_{out}$, ALU/add, Z$_{in}$

IND1 Control Signal Instructions:

1. GPR[IR.Rs1]$_{out}$, MAR$_{in}$, Read_MM
2. WMFC
3. MDR$_{out}$, OP1$_{in}$
4. If I2 = 0:
    a. GPR[IR.RS2]$_{out}$, OP2$_{in}$

IND2 Control Signal Instructions:

1. GPR[IR.RS2]$_{out}$, MAR$_{in}$, Read_MM
2. WMFC
3. MDR$_{out}$, OP2$_{in}$

IND3 Control Signal Instructions:

1. Z$_{out}$, MAR$_{in}$

E1 Control Signal Instructions:

This state executes the 3-address and 2-address instructions. The control signals for these instructions are as follows.

3-address instructions:

ADD opcode = 0, GPR[Rd]=OP1+OP2:

1. OP1$_{out}$, Y$_{in}$
2. OP2$_{out}$, ALU/add, Z$_{in}$
3. Z$_{out}$, GPR[IR.Rd]$_{in}$

SUB opcode = 1, GPR[Rd]=OP1-OP2:

1. OP1$_{out}$, Y$_{in}$
2. OP2$_{out}$, ALU/subtract, Z$_{in}$
3. Z$_{out}$, GPR[IR.Rd]$_{in}$

AND opcode = 2, GPR[Rd]=OP1 **and** OP2:

1. OP1$_{out}$, Y$_{in}$
2. OP2$_{out}$, ALU/and, Z$_{in}$

3. Z_{out}, GPR[IR.Rd]$_{in}$

SHL opcode = 3, GPR[Rd]=shift_left(OP1) by $OP2_{3-0}$:

1. OP2[3-0]$_{out}$, Y_{in}
2. $OP1_{out}$, ALU/shift_left, Z_{in}
3. Z_{out}, GPR[IR.Rd]$_{in}$

SHRA opcode = 4, GPR[Rd]=shift_right(OP1) by $OP2_{3-0}$:

1. OP2[3-0]$_{out}$, Y_{in}
2. $OP1_{out}$, ALU/shift_right, Z_{in}
3. Z_{out}, GPR[IR.Rd]$_{in}$

OR opcode = 5, GPR[Rd] = OP1 **or** OP2:

1. $OP1_{out}$, Y_{in}
2. $OP2_{out}$, ALU/or, Z_{in}
3. Z_{out}, GPR[IR.Rd]$_{in}$

2-address instructions:

NOT opcode = 6, GPR[Rd] = **not** MM[PC + Short_Offset]:

1. read_MM
2. WMFC
3. MDR_{out}, ALU/not, Z_{in}
4. Z_{out}, GPR[IR.Rd]$_{in}$

LD opcode = 7, GPR[Rd] = MM[PC + Short_Offset]:

1. read_MM
2. WMFC
3. MDR_{out}, GPR[IR.Rd]$_{in}$

ST opcode = 8, MM[PC + Short_Offset] = GPR[Rd]:

1. GPR[IR.Rd]$_{out}$, MDR_{in}, Write_MM

JSR opcode = 12, GPR[Rd] = PC; PC = PC + Short_Offset:

1. GPR[IR.Rd]$_{in}$, Y_{out}
2. Z_{out}, PC_{in}

RTS opcode = 13, PC = GPR[Rd] + Short_Offset:

1. GPR[IR.Rd]$_{out}$, Y_{in}
2. IR.Short_Offset, ALU/add, Z_{in}
3. Z_{out}, PC_{in}

E2 Control Signal Instructions:

This state executes the 1-address instructions. The control signals for these instructions are as follows.

1-address

BRN opcode = 9, if PSW.N then PC = PC + Long_Offset:

1. If PSW.N:
   a. Z_{out}, PC_{in}

BRZ opcode = 10, if PSW.Z then PC = PC + Long_Offset:

1. If PSW.Z:
   a. Z_{out}, PC_{in}

BR opcode = 11, PC = PC + Long_Offset:

1. Z_{out}, PC_{in}

CLK opcode = 14, set timer to MM[PC + Long_Offset]:

1. Z_{out}, MAR_{in}, Read_MM
2. WMFC
3. MDR_{out}, $Timer_{in}$

LPSW opcode = 15, PSW = MM[PC + Long_Offset]:

1. Z_{out}, MAR_{in}, Read_MM
2. WMFC
3. MDR_{out}, PSW_{in}

Set.CC Control Signal Instructions:

This state sends out a single set condition code signal. This control signal activates two tri-state buffers that go into the Z and N bits of the PSW. The Z bit indicates that the executed instruction resulted in 0. To check this, a 16 input NOR gate reads the value stored in the Z register and it outputs 1 if all 16 bits are 0. The N bit indicates that the executed instruction resulted in a value less than 0. An AND gate checks if the overflow coming out of the look ahead adder is 0. It also checks if the instruction's opcode was 1 which indicates that a subtraction was performed. This check is useful because subtraction is the only instruction that could result in a negative number.

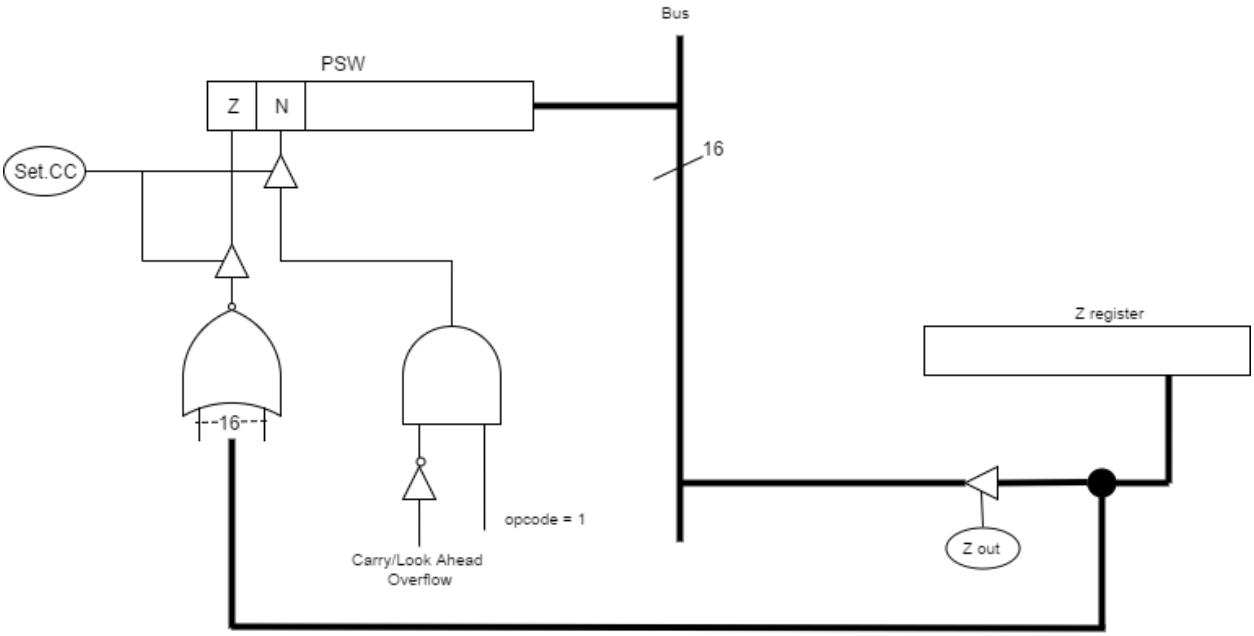

Figure 21: Set.CC Signal and Implementation

Timeout:

1. $ROM[8]_{out}$, MAR_{in}
2. PSW_{out}, MDR_{in}, Write_MM
3. WMFC
4. $ROM[10]_{out}$, MAR_{in}
5. PC_{out}, MDR_{in}, Write_MM
6. WMFC
7. $ROM[12]_{out}$, MAR_{in}, Read_MM
8. WMFC
9. MDR_{out}, PSW_{in}
10. $ROM[14]_{out}$, MAR_{in}, Read_MM
11. WMFC
12. MDR_{out}, PC_{in}

Trap:

1. $ROM[0]_{out}$, MAR_{in}
2. PSW_{out}, MDR_{in}, Write_MM
3. WMFC
4. $ROM[2]_{out}$, MAR_{in}
5. PC_{out}, MDR_{in}, Write_MM
6. WMFC
7. $ROM[4]_{out}$, MAR_{in}, Read_MM
8. WMFC

9. MDR_{out}, PSW_{in}
10. $ROM[6]_{out}$, MAR_{in}, Read_MM
11. WMFC
12. MDR_{out}, PC_{in}

Control Unit Substates:

The final stage of the control unit consists of substates for each high-level state. It is during the substates that control signals are sent to the rest of the processor. Each substate is built off a 2-bit counter like the one shown in Figure 22 below.

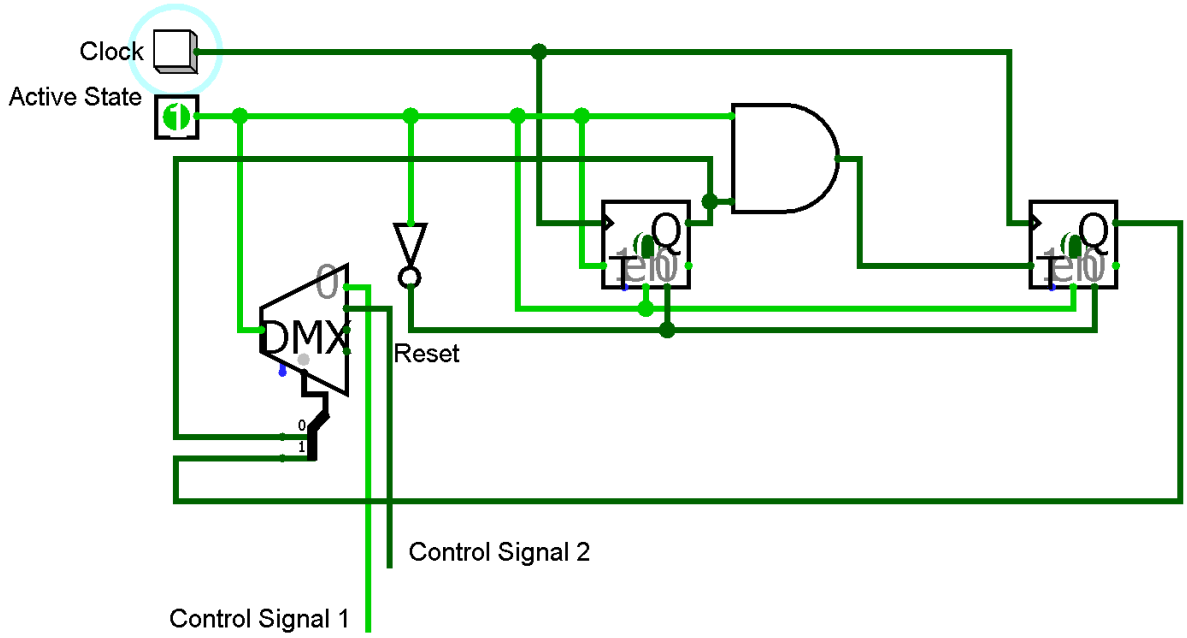

Figure 22: Example 2-Bit Counter

This counter is activated by the "Active State" input. This input is the output bit from the intermediate stage's decoder that corresponds to the current $Q_3Q_2Q_1Q_0$ value. This active state input enables two T flip-flops that count the number of clock cycles from 0 to 3. An AND gate is placed between the two flip flops to increment the second bit of the clock count if the first bit is 1. The outputs from the T flip-flops are sent to the selector bit of demultiplexer that outputs a signal corresponding to the current clock cycle count. This signal activates the control signals for that step of the instruction. On the last step of the substate, a signal is also output to the enable bit of the intermediate stage's D flip flops. This will switch to the first substate of the next state. When the active state signal is turned off, a not gate is used to reset both T flip flops.

A 4-bit counter was also created for this control unit. Its function is identical to the 2-bit counter described above except this counter counts 4 bits instead of 2 bits. This 4-bit counter is

used as a substate that combines the Timeout and Trap states. A gate level implementation of the 4-bit counter is shown in Figure 23 below.

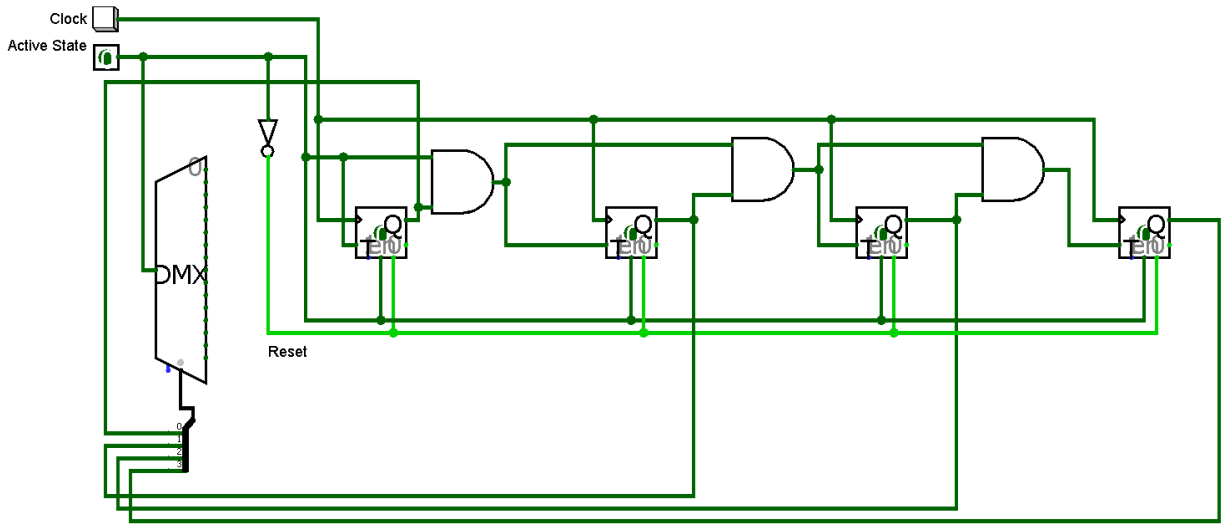

Figure 23: 4-Bit Counter

The first substate of the control unit handles all the control signals for fetch. On the first clock cycle, it places the value of the PC on the bus and stores that value in the memory address register and the Y register of the ALU. It also reads main memory at the location specified by the PC. Lastly, it enables register 8 which stores an incremented PC value in the extra register. On the second clock cycle, no signals are sent but this clock cycle is used to wait for the memory function to complete. On the final clock cycle, the value stored in the MDR is placed on the bus and stored in the instruction register. Signals are also sent to enable register 8, activate the tri-state buffer coming out of register 8 and update all general purpose registers. These three control signals are responsible for placing the incremented PC value back in the PC register. The gate level implementation of this substate is shown in Figure 24.

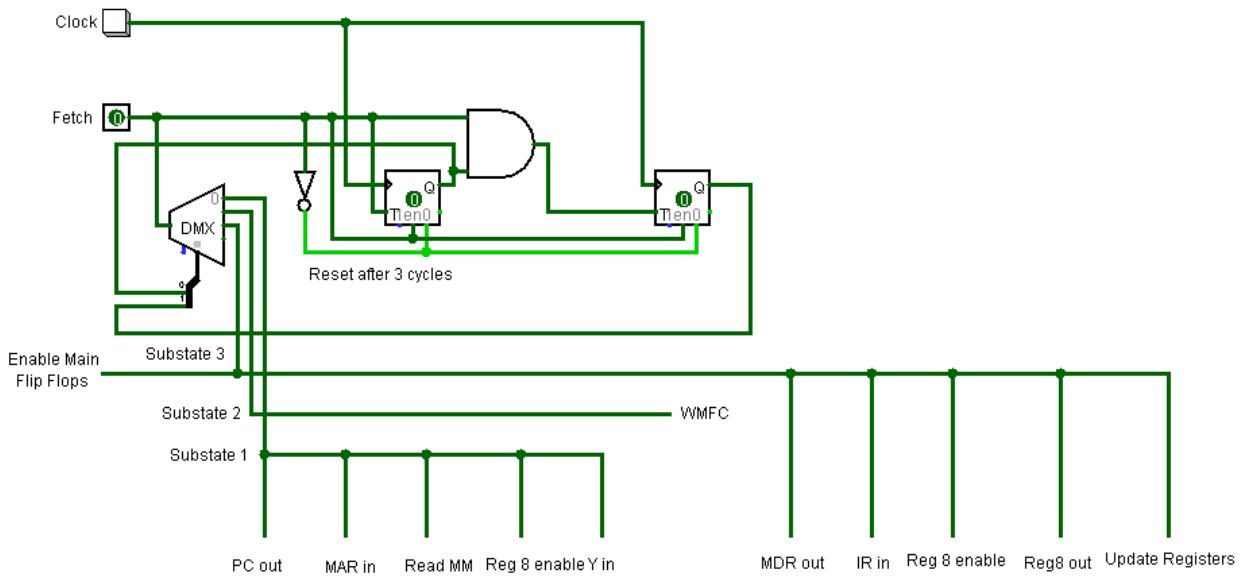

Figure 24: Fetch Substate

The next substate handles all the control signals for D0. The first clock cycle check the I1 and I2 bits of the instruction. If the value of I1 is 1, no control signals are sent, the intermediate flip-flops are enabled and the state machine progresses to the next state. If the value of I1 is 0, the GPR value at the address denoted by the RS1 bits of the instruction is output onto the bus and stored in the OP1 register. If I2 = 1, the intermediate flip flops are enabled on the first clock cycle and the state machine progresses to the next state. If I2 = 0, the GPR value at the address denoted by the RS2 bits of the instruction is output onto the bus and stored in the OP2 register. The gate level implementation of this substate is shown in Figure 25.

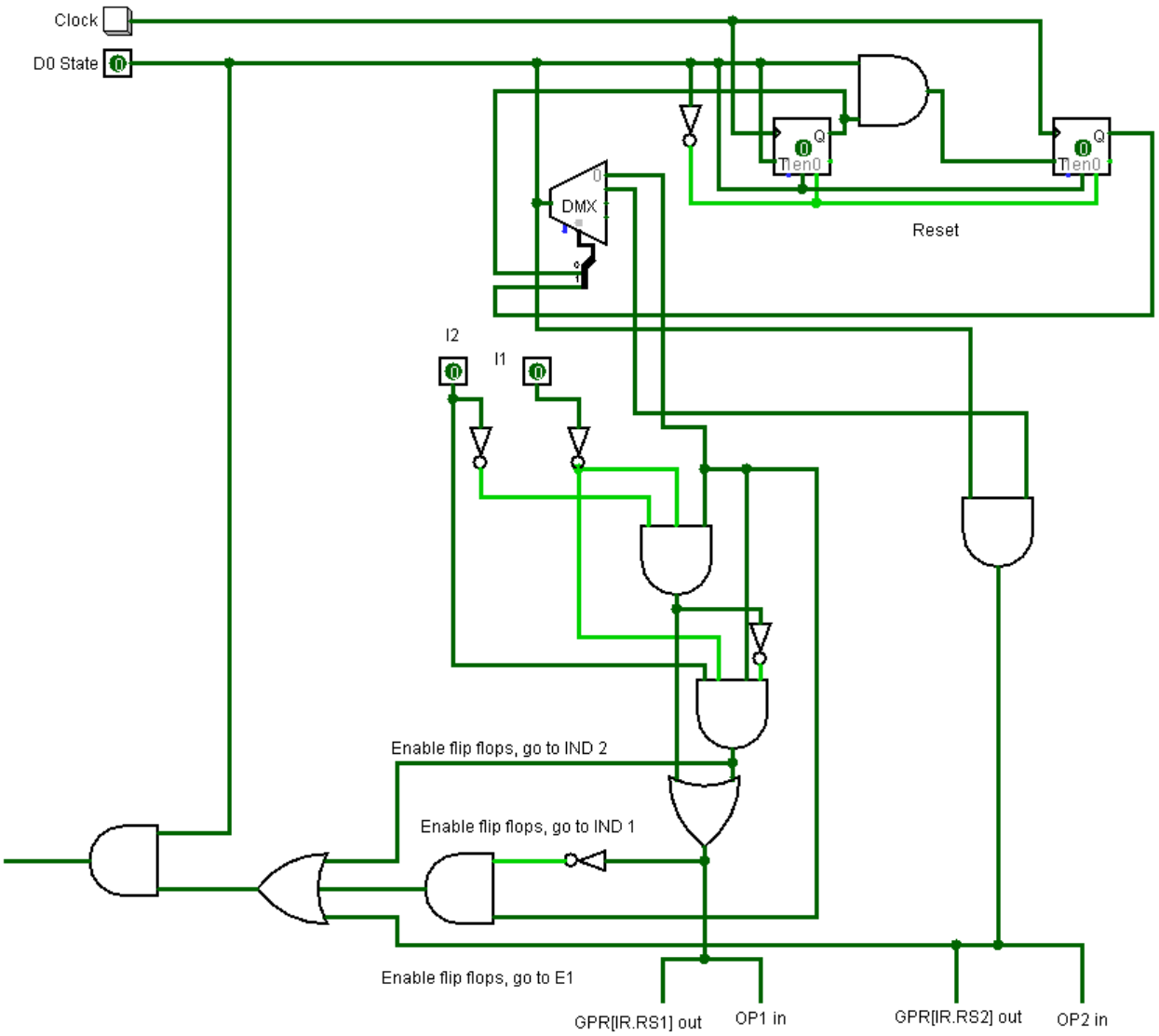

Figure 25: D0 Substate

The next substate handles all control signals for D1. This substate only sends one set of control signals. It sends control signals to place the data stored in the instruction register's short offset bits on the bus and add them to the PC value that was stored in Y during the fetch cycle. This sum is stored in the Z register. These signals are an optimization that reduces the number of clock cycles needed to execute some of the 2-address instructions. The gate level implementation of this substate is shown in Figure 26.

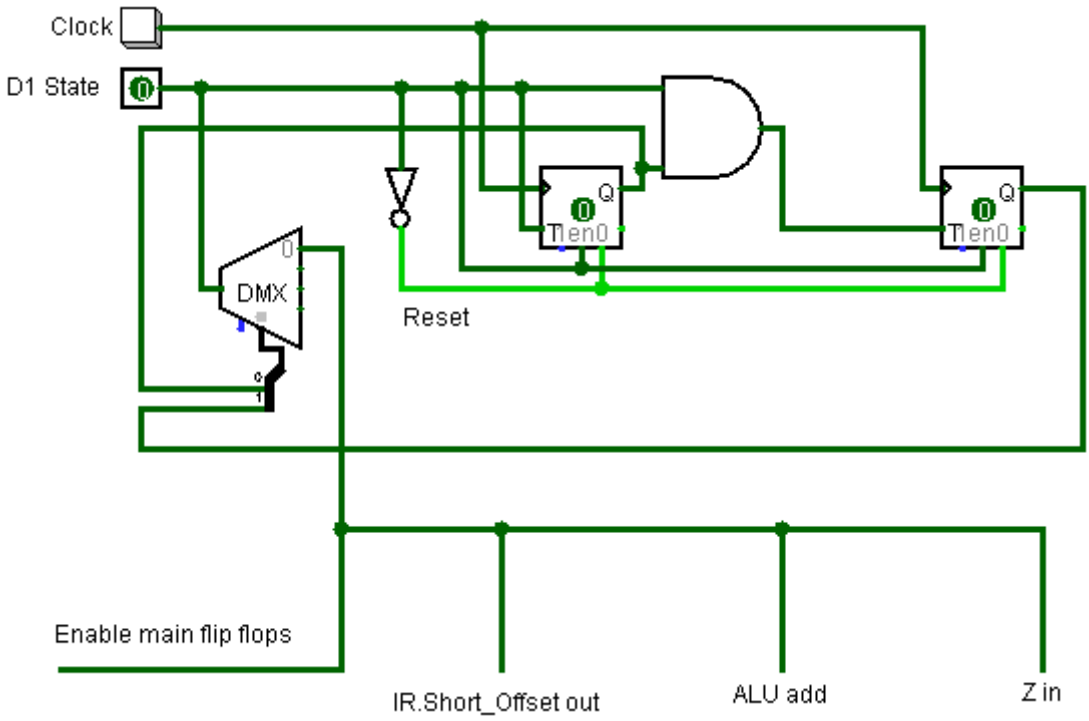

Figure 26: D1 Substate

The next substate handles all control signals for D2. This substate only sends out one set of control signals. It sends control signals to output data stored instruction's long offset bits and add them to the PC value that was stored in Y during the fetch cycle. A control signal is also sent to the Z register coming off the ALU to store the added value. These signals are an optimization that reduces the number of clock cycles needed to execute 1-address instructions. The gate level implementation of this substate is shown in Figure 27.

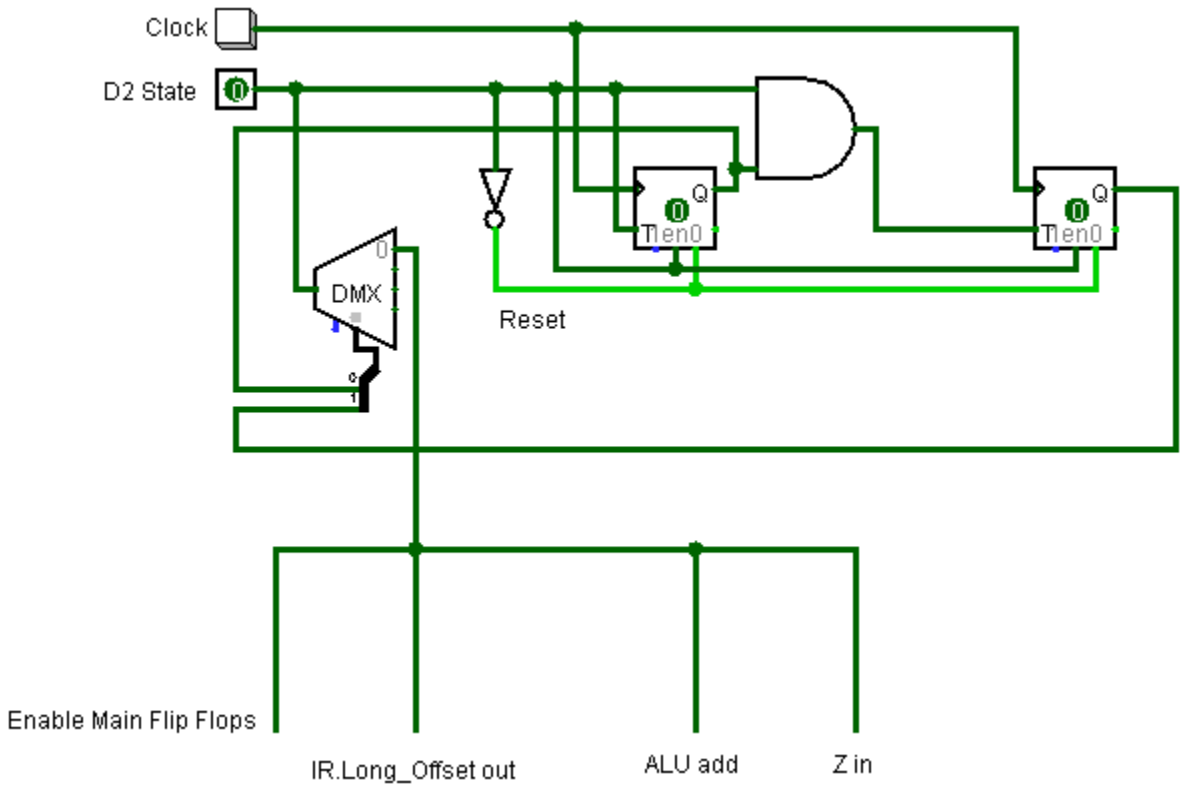

Figure 27: D2 Substate

The next substate handles all control signals for IND1. This substate is used to store a value from main memory in the OP1 register. During the first clock cycle, the GPR value at the address denoted by the RS1 bits of the instruction is output onto the bus, stored in MAR, and the main memory is instructed to read at that address. During the next clock cycle, nothing happens since the processor is waiting for the memory function to complete. On the third clock cycle, the data from the MDR is put on the bus and stored in OP1. For the final clock cycle, the GPR value at the address denoted by the RS2 bits of the instruction is output onto the bus and stored in the OP2 register if I2 = 0. The D flip flops in the intermediate stage are also enabled during this final clock cycle. If I2 = 1, the final set of control signals are not sent. The gate level implementation of this substate is shown in Figure 28.

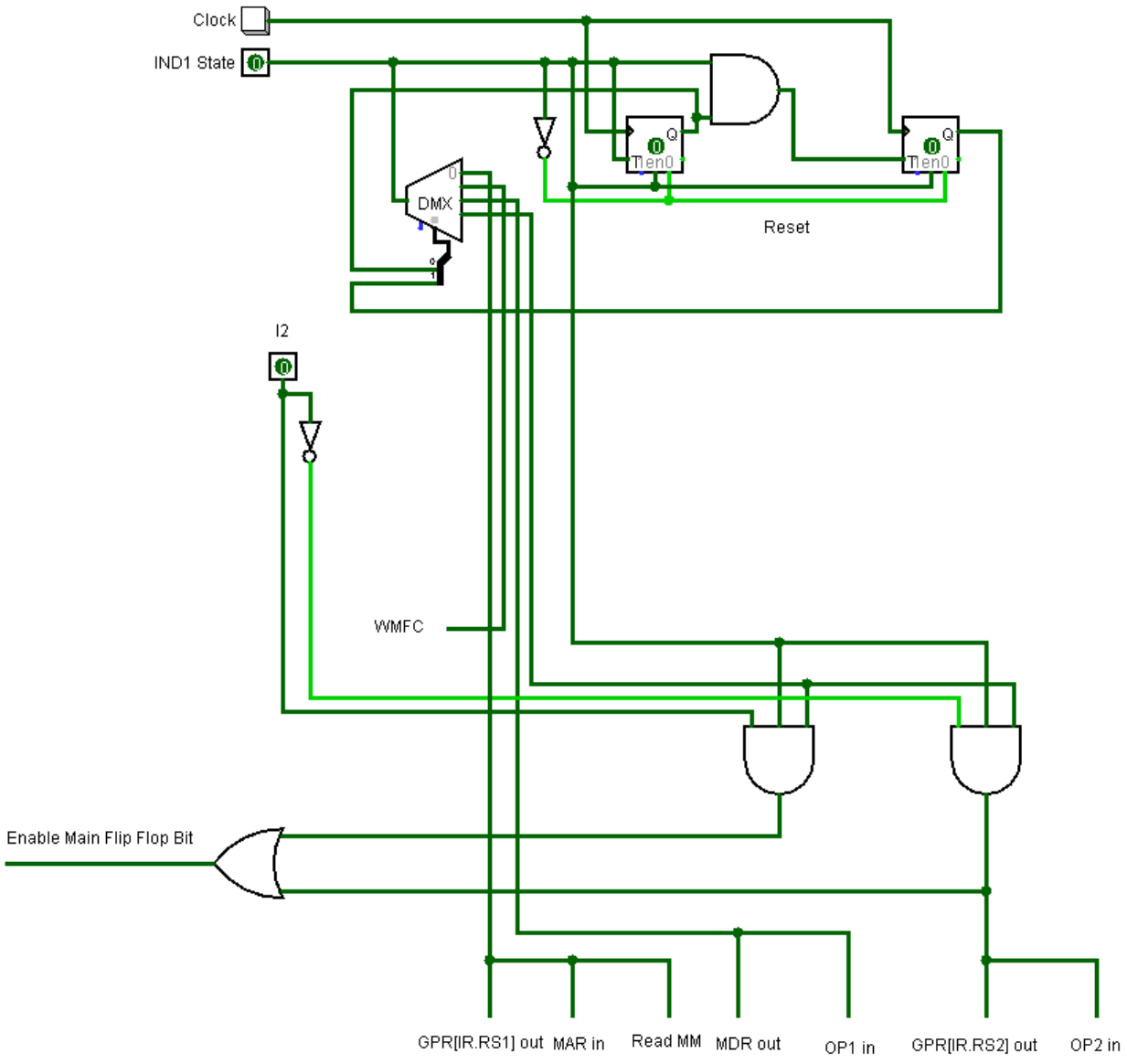

Figure 28: IND1 Substate

The next substate handles all control signals for IND2. This substate is used to store a value from main memory in the OP2 register. During the first clock cycle, signals are sent to place the value stored in the GPR address denoted by the data stored in the RS2 bits of the instruction on the bus and store it in MAR. An instruction is also sent telling main memory to read at that address. The next clock cycle waits for the memory function to complete, no control signals are sent during this time. The final clock cycle outputs the value in MDR and stores it in the OP2 register. The gate level implementation of this substate is shown in Figure 29.

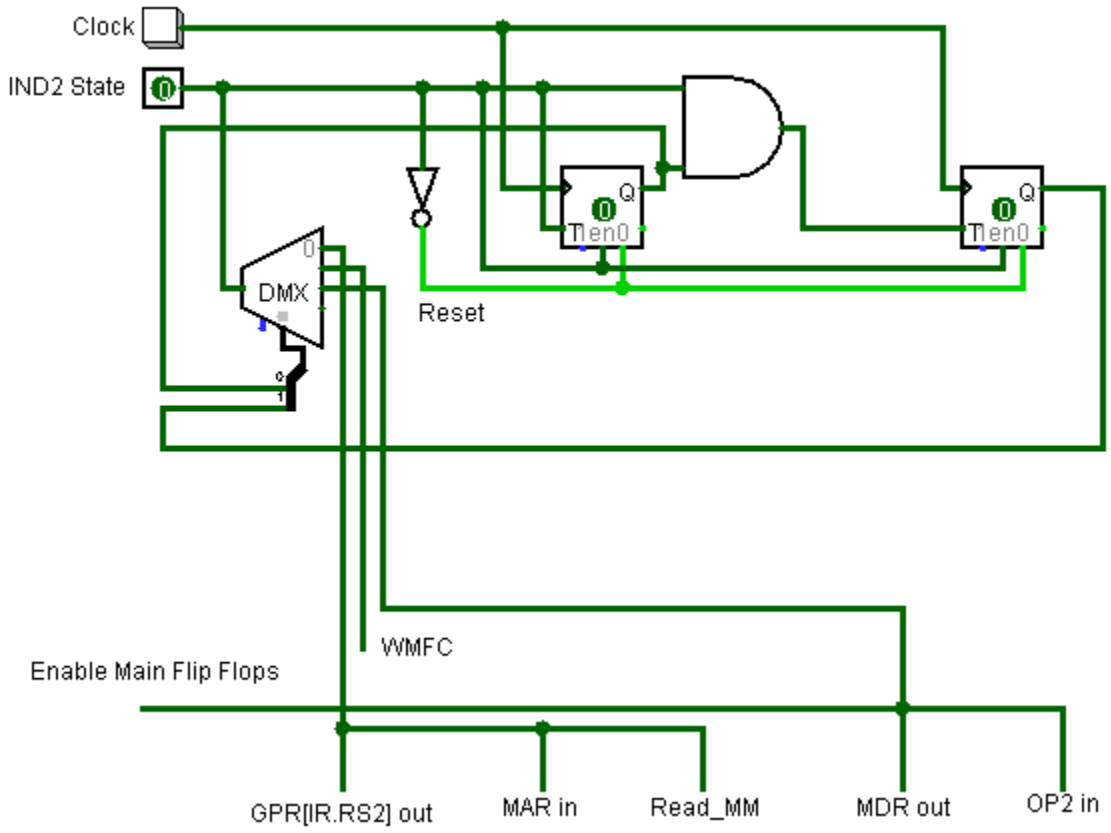

Figure 29: IND2 Substate

The next substate handles all control signals for IND3. This substate is used to send control signals that will optimize the circuit by simplifying the execution signals for opcodes 6, 7, and 8. This substate sends control signals to output the PC + short offset result stored in Z during D1 on the bus and store it in the MAR. Sending these control signals during this state simplifies the instructions for opcodes 6, 7, and 8 and reduces the number of clock cycles they require to execute. The gate level implementation of this substate is shown in Figure 30.

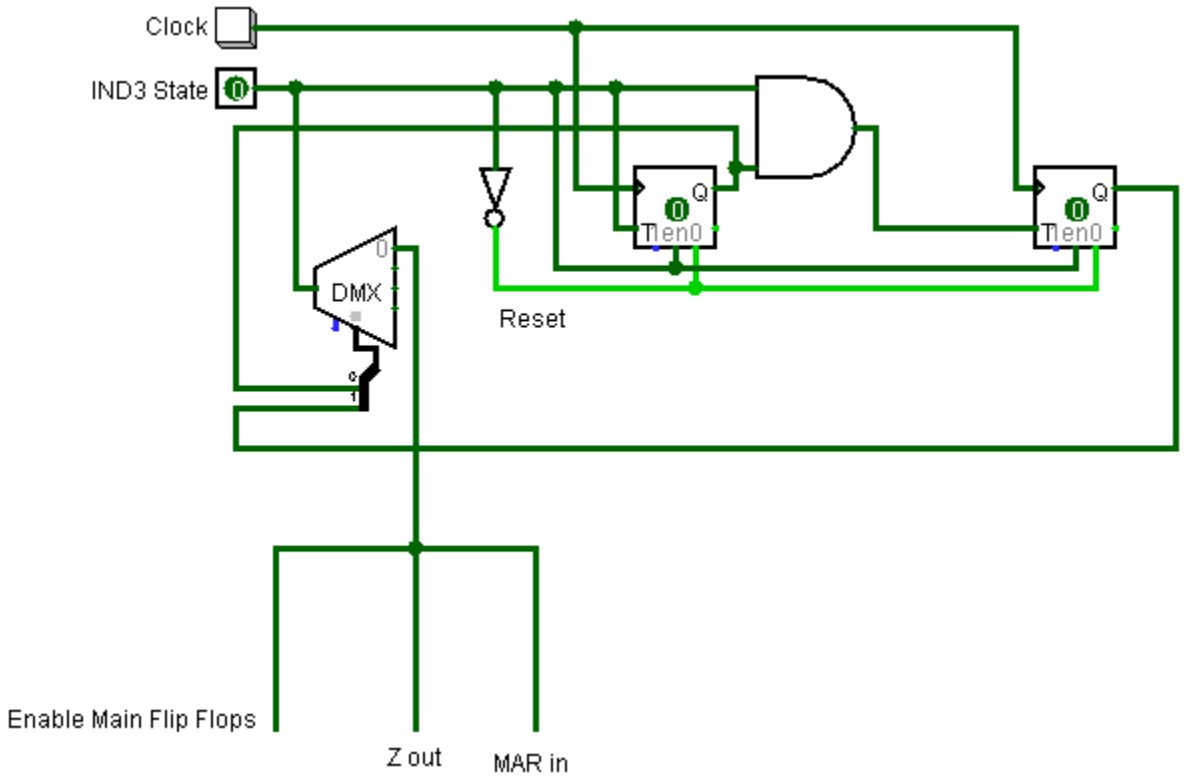

Figure 30: IND3 Substate

The next substate, E1, sends control signals to execute 3-address and 2-address instructions. This substate uses a demultiplexor for each clock cycle and uses the instruction register's opcode bits to determine which control signals to send. The control signals correspond with the instructions described above for each 3-address and 2-address opcode. A couple optimizations were made in this substate. In instances where multiple opcodes have the same instructions during a clock cycle, an OR gate is used to reduce the number control signals sent. A decoder is used to send the 3-bit selector bits to the ALU for instructions that use the ALU. The ALU control signals decoder is shown in Figure 32. This helps by formatting the control signal in a way that is readily accepted by the ALU. A tradeoff here is that additional hardware is used but it is worth it to simplify the control signals. Another optimization is that the intermediate D flip flops are activated early if an opcode's instructions do not require all 4 clock cycles. This moves the control unit to the next state and saves time by avoiding unnecessary clock cycles. Lastly, the hardware used in this substate is reduced by not using a demultiplexer for the fourth clock cycle. Since opcode 6 is the only instruction set that needs a fourth clock cycle, a demultiplexer to select this opcode's control signals is unnecessary. This control signals can be wired directly to the counter's fourth output. The gate level implementation of this substate is shown in Figure 31.

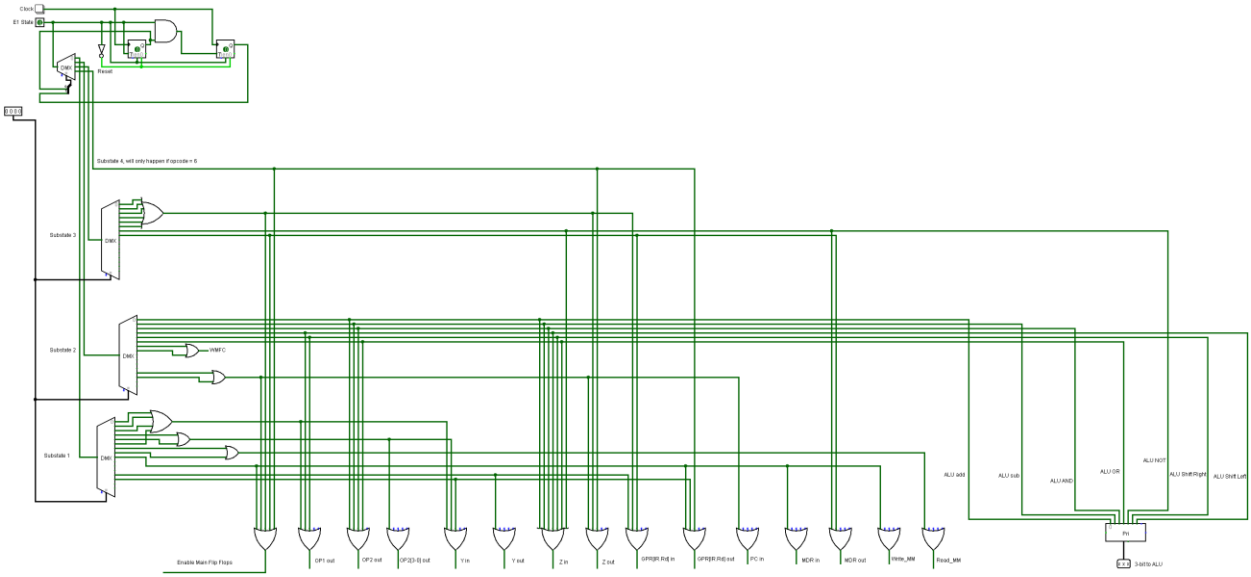

Figure 31: E1 Substate

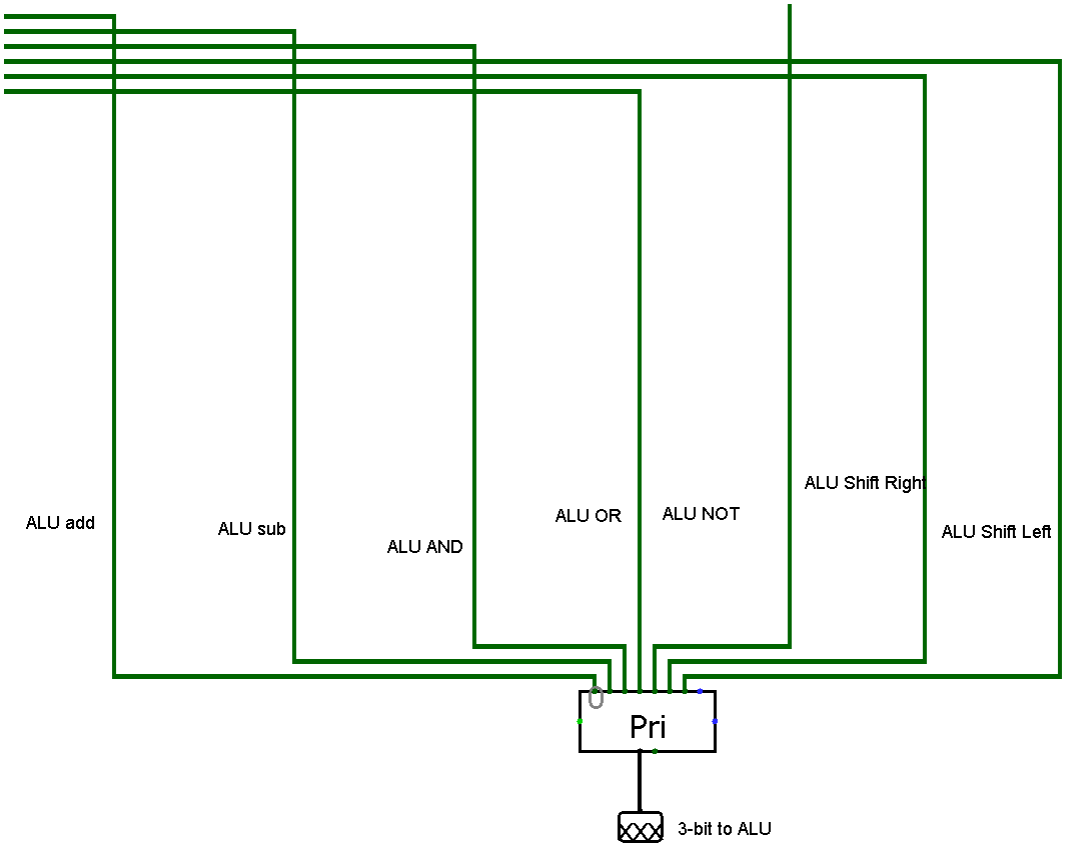

Figure 32: E1 ALU Control Signals

The next substate, E2, sends control signals to execute the 1-address instructions. This substate uses a demultiplexer that uses the instruction's opcode bits as selector bits for each clock cycle to determine which control signals to send. The control signals sent in this step are consistent with the instruction sets described above. AND gates are used for instructions 9 and 10 because they only send control signals if the PSW has 1 values for the N or Z bits respectively. An OR gate connects the results of these gates and the demultiplexor output at 11 because the same control signals are sent if either of these conditions are met. Additional OR gates are used to combine instructions that send the same control signals during a clock cycle to simplify the number of control signals. The second clock cycle of the counter does not do anything as that step requires waiting for the memory function to complete. As with the substate for E1, the intermediate D flip flops are enabled on the final step of the selected instruction to avoid waiting through unnecessary clock cycles. The gate level implementation of this substate is shown below in Figure 33.

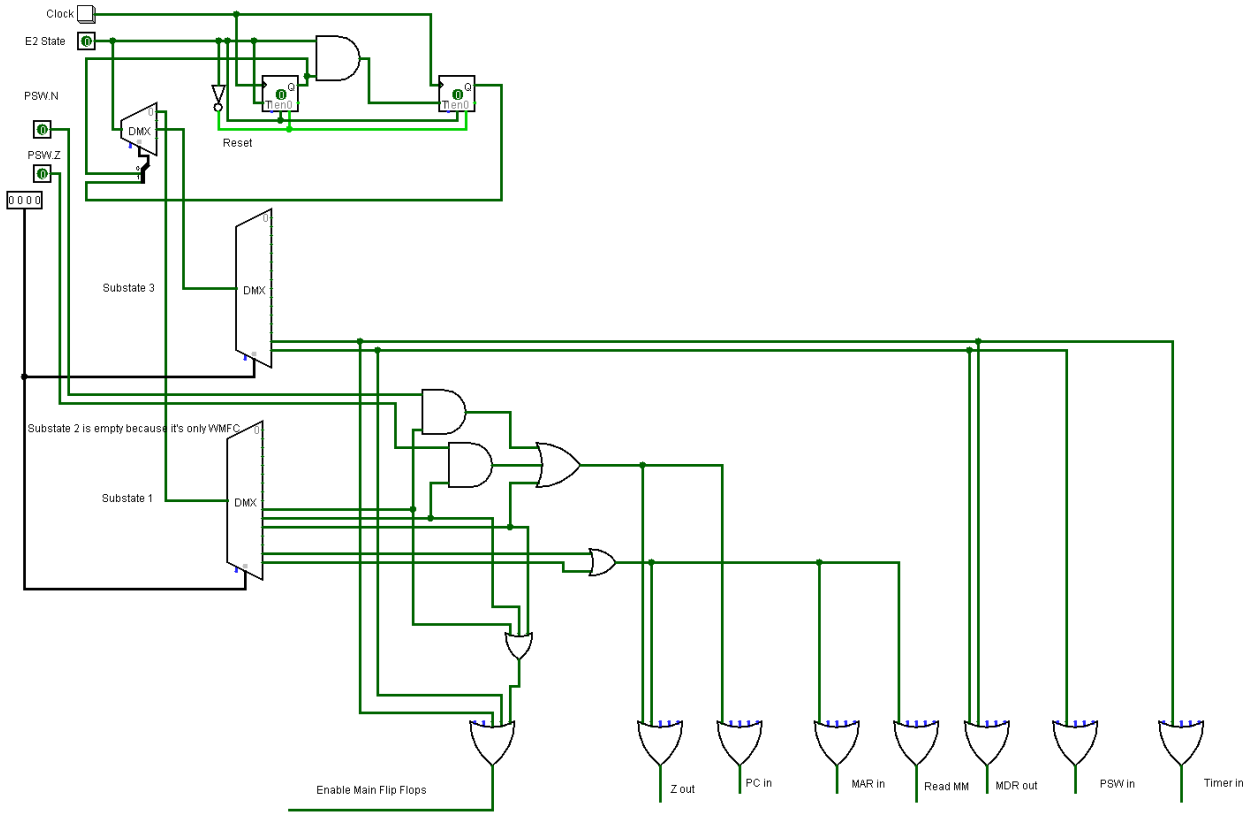

Figure 33: E2 Substate

The next substate, Set.CC, does not have a gate level implementation. This is because this state only sends out one signal and does not require multiple clock cycles. The set.CC control signal is wired directly to the intermediate state's decoder. This signal will also enable the D flip flops in the intermediate stage to advance the state machine to its next state. The use of set.CC was previously outlined in Figure 21.

The final two substates, Timeout and Trap, are handled by the same substate gate level implementation. The combination of these substates is an optimization because it reduces hardware and control signal wiring. This substate uses a 4-bit counter because both instruction sets require 12 clock cycles. The instructions are identical aside from the ROM addresses they read from. To solve this, the intermediate decoder's timeout bit is wired to tri-state buffers used to send signals to the ROM addresses used during timeout. The trap bit is similarly wired to tri-state buffers used to send signals to the ROM addresses used during the trap state. The gate level implementation of these substates is shown below in Figure 34.

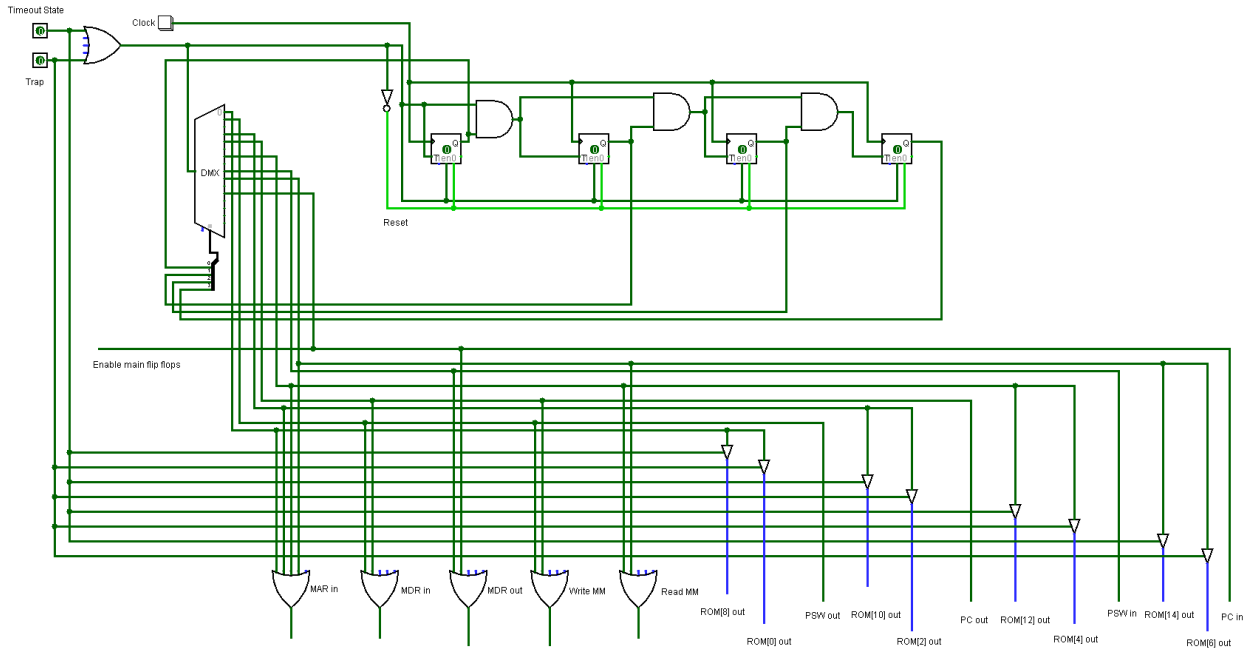

Figure 34: Timeout and Trap Substate

A screenshot of our entire control unit is shown in Figure 35 on the next page.

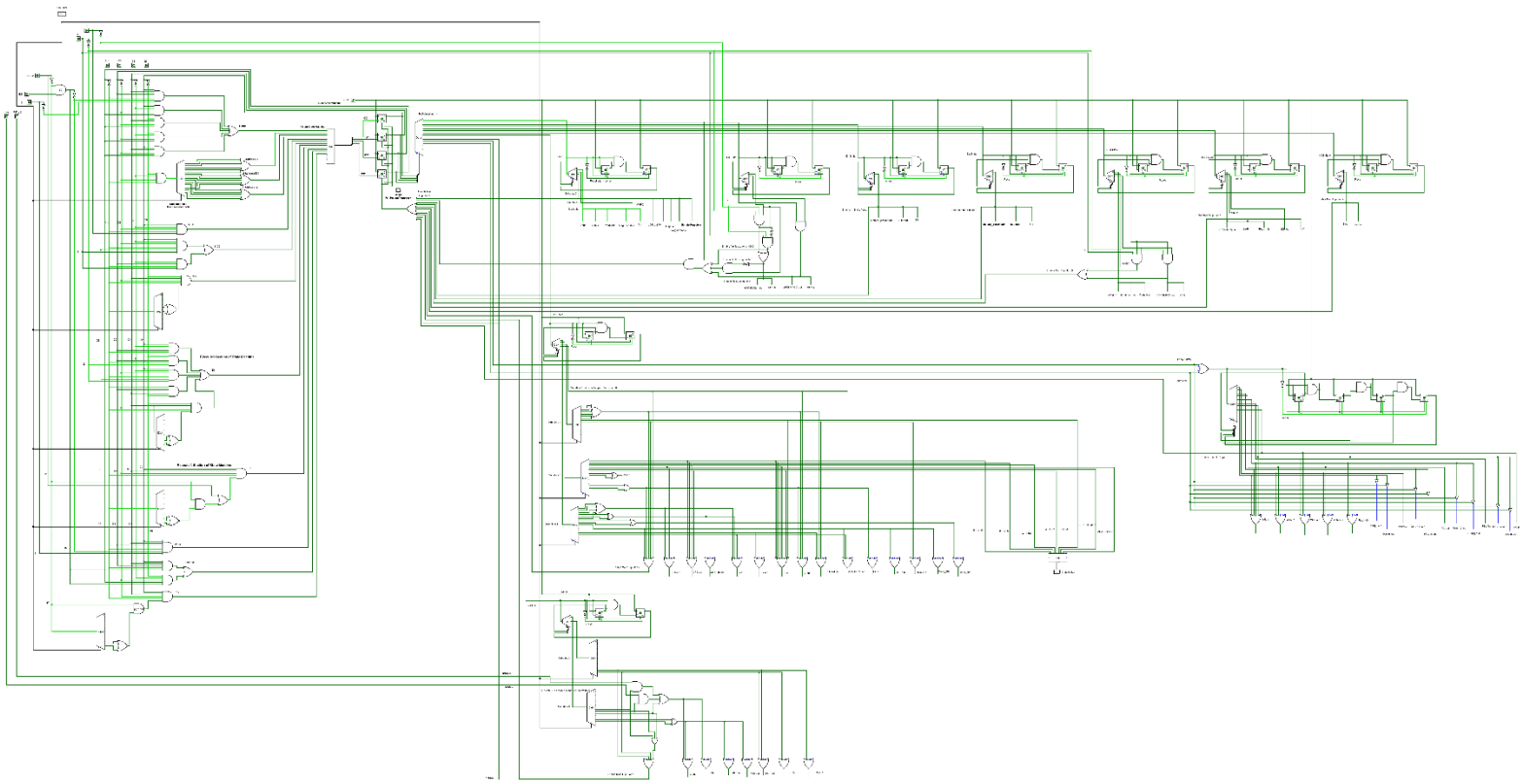

Figure 35: Control Unit

Optimizations

Ripple Carry Adder to Carry Look Ahead Adder

The ripple carry adder is a much simpler circuit hardware wise than the carry look ahead adder but it is also much slower. The carry look ahead adder reduces calculation time by calculating all of the carry bits at the same time instead of having to wait for each bit to finish computing to know what the carry is.

Single Level State Machine to Hierarchical Structure of States

In our hierarchical structure there are 12 high level states that break off into lower level sub-states in which control signals are sent to the components of the processor. This is an improvement from the disorderly structure of having all of the states represented equally. This allowed us to minimize our state machine from 64 states to 12 states and 55 substates. Which means we do not have to check for every single next state equation since most substates can be executed linearly.

Comparator to Current Circuitry

Originally, we planned on using a comparator to define the Z and N bits of the PSW. We later realized with some simple circuits we could compute it in less time. We use an NOR gate with 16 inputs taking all the inputs from the Z register to compute the Z bit. To compute the N bit we send a signal to an AND gate when opcode 1 is sent to the ALU then the NOT of the Carry/Look Ahead Overflow into the other input of the AND gate. This works because there will only ever be a negative output when subtraction is being done (opcode 1) and a negative number is represented by the Carry/Look Ahead Overflow being 0. All of this can be seen in Figure 21.

The following 2-address and 1-address instructions were optimized using decode states. The benefit of making these optimizations is that it reduces the number of clock cycles needed to execute these instructions.

2 address instructions:

NOT opcode = 6, GPR[Rd] = **not** MM[PC + Short_Offset]:

Old Instruction	Optimization
1. PC_{out}, Y_{in}	This was removed by adding Y_{in} to the first line of Fetch.
2. IR.Short_Offset$_{out}$, ALU/add, Z_{in}	This was removed by sending these signals during D1.
3. Z_{out}, MAR_{in}, read_MM	Z_{out}, MAR_{in} were removed by sending these signals in the IND3 state.
4. WMFC	
5. MDR_{out}, ALU/not, Z_{in}	
6. Z_{out}, GPR[IR.Rd]$_{in}$	

LD opcode = 7, GPR[Rd] = MM[PC + Short_Offset]:

Old Instruction	Optimization
1. PC_{out}, Y_{in}	This was removed by adding Y_{in} to the first line of Fetch.
2. IR.Short_Offset$_{out}$, ALU/add, Z_{in}	This was removed by sending these signals during D1.
3. Z_{out}, MAR_{in}, read_MM	Z_{out}, MAR_{in} were removed by sending these signals in the IND3 state.
4. WMFC	
5. MDR_{out}, GPR[IR.Rd]$_{in}$	

ST opcode = 8, MM[PC + Short_Offset] = GPR[Rd]:

Old Instruction	Optimization
1. GPR[IR.Rd]$_{out}$, MDR_{in}	
2. PC_{out}, Y_{in}	This was removed by adding Y_{in} to the first line of fetch.
3. IR.Short_Offset$_{out}$, ALU/add, Z_{in}	This was removed by sending these signals during D1.
4. Z_{out}, MAR_{in}	This step was removed by sending these signals in the IND3 state.
5. Write_MM	Lines 1 and 5 could be combined into a single step after the optimizations.

JSR opcode = 12, GPR[Rd] = PC; PC = PC + Short_Offset:

Old Instruction	Optimization
1. PC_{out}, $GPR[IR.Rd]_{in}$, Y_{in}	Since the PC_{out} value was stored in Y_{in} during fetch, this line was revised to: $GPR[IR.Rd]_{in}$, Y_{out}
2. $IR.Short\_Offset_{out}$, ALU/Add, Z_{in}	This step was removed by sending these signals during D1.
3. Z_{out}, PC_{in}	

1-Address:

BRN opcode = 9, if CC.N then PC = PC + Long_Offset:

Old Instruction	Optimization
1. If CC.N → 2	
2. $IR.Long\_Offset_{out}$, Y_{in}	$IR.Long\_Offset_{out}$, ALU/add, Z_{in} were sent during D2.
3. PC_{out}, ALU/add, Z_{in}	PC_{out} was stored in Y_{in} during fetch
4. Z_{out}, PC_{in}	

BRZ opcode = 10, if CC.Z then PC = PC + Long_Offset:

Old Instruction	Optimization
1. If CC.Z → 2	
2. $IR.Long\_Offset_{out}$, Y_{in}	$IR.Long\_Offset_{out}$, ALU/add, Z_{in} were sent during D2.
3. PC_{out}, ALU/add, Z_{in}	PC_{out} was stored in Y_{in} during fetch
4. Z_{out}, PC_{in}	

BR opcode = 11, PC = PC + Long_Offset:

Old Instruction	Optimization
1. $IR.Long\_Offset_{out}$, Y_{in}	$IR.Long\_Offset_{out}$, ALU/add, Z_{in} were sent during D2.
2. PC_{out}, ALU/add, Z_{in}	PC_{out} was stored in Y_{in} during fetch
3. Z_{out}, PC_{in}	

CLK opcode = 14, set timer to MM[PC + Long_Offset]:

Old Instruction	Optimization
1. $IR.Long\_Offset_{out}$, Y_{in}	$IR.Long\_Offset_{out}$, ALU/add, Z_{in} were sent during D2.
2. PC_{out}, ALU/add, Z_{in}	PC_{out} was stored in Y_{in} during fetch
3. Z_{out}, MAR_{in}, Read_MM	
4. WMFC	
5. MDR_{out}, $Timer_{in}$	

LPSW opcode = 15, PSW = MM[PC + Long_Offset]:

Old Instruction	Optimization
1. $IR.Long\_Offset_{out}$, Y_{in}	$IR.Long\_Offset_{out}$, ALU/add, Z_{in} were sent during D2.
2. PC_{out}, ALU/add, Z_{in}	PC_{out} was stored in Y_{in} during fetch
3. Z_{out}, MAR_{in}, Read_MM	
4. WMFC	
5. MDR_{out}, PSW_{in}	