For this problem, we are going to make use of this simple datapath that consists of a four register register file, a relatively simple arithmetic logic unit that can perform ADD, SUB, MUL, and NAND operations. In addition, it can compare two inputs and determine whether or not they are equal. The result of the comparison, Z, can then be used to control what happens next. There are multiple control signals in this datapath. The first two are Asel and Bsel.

These are used to select which register drives the corresponding multiplexer output.

The value stored in the register selected by Asel becomes the A input to the arithmetic operations and is passed to the arithmetic units along the red wire.

The value stored in the register selected by Bsel becomes the B input to the arithmetic operations and is passed to the arithmetic units along the blue wire.

The next control signal is Opsel. It selects which of the four operation outputs should be selected by the Opsel multiplexer as the result of our operation.

This result is fed back to the register file along the purple wire.

The Wen is a write enable for the register file which specifies whether or not the result of our operation should be written back into the register file.

If it is supposed to be written back to the register file, then the Wsel control signal selects to which of the registers the result should be written.

The yellow box is the control FSM. It generates the control signals for the rest of the datapath based on the operations that you want to perform.

Suppose the initial value of our 4 registers is: R0 = 1, R1 = 0, R2 = -1, and R3 = N. We want to evaluate the result of the function 3 * N – 2 and store the result into R3. Our job is to design the control FSM so that it produces the correct signals to achieve what we want.

To help us get started, here is an incomplete listing of the code that will achieve what we want. The Sx labels are the names of the states corresponding to each instruction in our program. Our first job is to figure out the values of RX, RY, and RZ so that our code behaves as expected.

Let's begin by looking at state S0. We want to end up with the value -2 in R2 by adding R2 which currently holds -1 to some register.

In order to produce -2, we need to add -1 which means that RX = R2.

Next, we look at state S1. Here we want to end up with the value 2 in R1 by adding Ry to R0 which currently holds a 1.

In order to produce 2, we need to add 1 which means RY = R0.

State S2 adds R0 to R1 and stores the result in R1.

Since R0 still equals 1 and R1 = 2, then we produce R1 = 3.

Now, let's look at state S3. Our goal is to multiply 3*N and store the result into R3. To achieve this, we multiply RZ by R3 and store the result in R3. Since R3 currently = N, that means that we want to multiply it by R1 which equals 3. So RZ = R1.

Finally, we add R3 and R2 to produce 3*N-2 and store that result back into R3.

S5 just executes a HALT() instruction to indicate that we are done.

Now that we have working code, our next goal is to determine the settings for the control FSM that will make the correct operations be executed by our datapath.

Since we have 6 states, we will need 3 state bits to encode the value of the current and next state. We begin with state S0.

In order to encode that we are in state zero using 3 bits, we set our current state, S[2:0] to 000. In this operation we don't care about the Z signal, so Z = X which means don't care. The instruction that we want to execute after this first ADD, is the next ADD in state S1. This means that our next state is 001.

Note that the notation S' is often used to represent the next state.

Our register select signals each need to select one of 4 registers.

This means that these signals must each be 2 bits wide.

Our Asel control signal identifies the register that should be used as input A.

This register is R2, so Asel is 10. Bsel identifies the second source operand.

In this case it is also R2, so Bsel = 10 as well.

The Opsel signal identifies which operation we want to perform.

Since we have 4 distinct operations, we would need two bits to distinguish amongst them and we would make each operation be associated with one of the 4 encodings.

For simplicity, let's just label Opsel as ADD to indicate that we selected the encoding for the ADD. The register we want to write our result to, also known as the destination register, is R2 for this operation.

This means that Wsel = 10 and Wen = 1. Wen is a signal that enables writing to the register file. If it is set to 0, then regardless of the value of Wsel, no value will be written into the register file.

Now let's quickly run through the rest of our instructions.

Our current state is state S1, or 001. Once again Z is a don't care.

Since the instruction that will be executed next is the one in S2, our next state is 010.

Our Asel = 00 and Bsel = 00. Opsel = ADD and Wsel = 01 and Wen = 1.

State 2 follows the same model, so current state is 010 and next state is 011.

Here Asel = 00, Bsel = 01 and Wsel = 01 and Wen = 1.

Once again our Opsel is an ADD. We move on to state 3 whose current state is 011 and next state is 100. Asel = 01, Bsel = 11, Wsel = 11, and Wen = 1.

Here our Opsel is MUL to indicate that the operation to be executed here is a multiply.

For state four, we have current state set to 100 and next state to 101.

Asel = 11, Bsel = 10, Wsel = 11, and Wen = 1. Our Opsel is once again ADD.

Finally, we reach state 5. This state looks a little different from the previous states so lets examine it a little more closely.

The first thing to note, is that when we get to state 5 we want to stay there because we are done with our execution, so both the current state and the next state are 101.

Most of the other control bits can be set to don't care because at this point we mostly don't care about what the rest of the datapath is doing.

The only other signal that we do need to worry about is Wen.

Since we are allowing the rest of our datapath to run in whatever way, we need to ensure that nothing produced on the datapath at this stage gets written back to any of the registers.

In order to guarantee that, we set Wen = 0. Here is the complete control ROM that will execute the function 3*N-2 and store its result into R3.