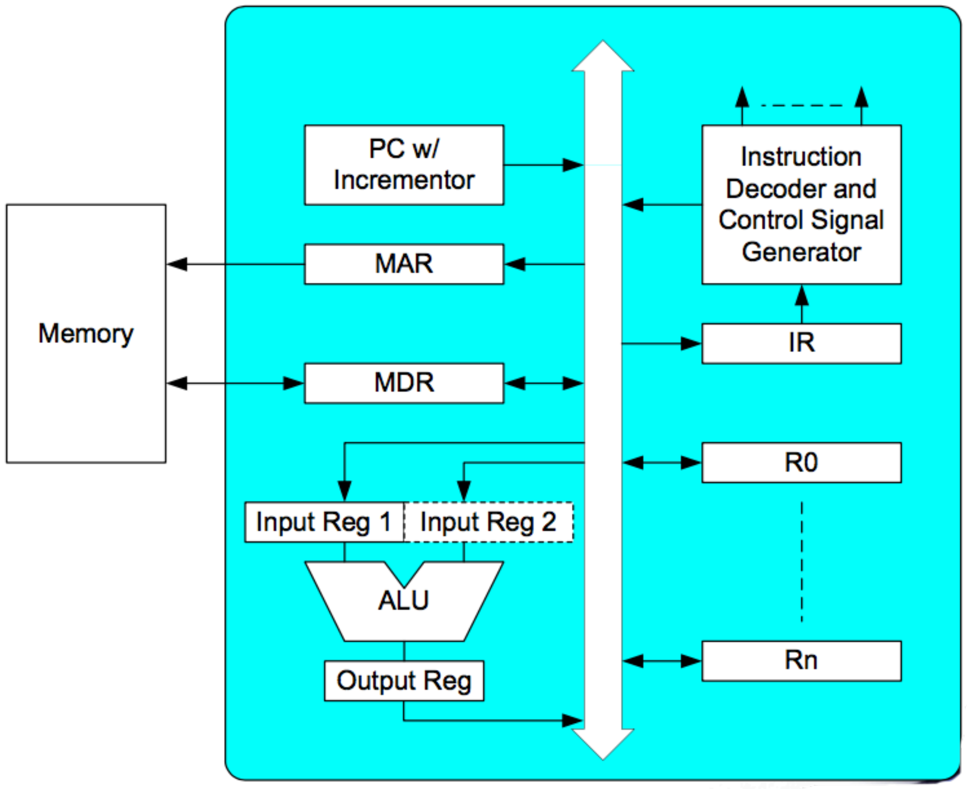
System Synthesis and Modeling (CSCE 3953)

RISC Microprocessor



Zack Fravel

010646947

12/8/16

zpfravel@uark.edu

Parent Objective

The parent objective of our final project in the course is to assemble our simple RISC microprocessor. All of us have been assigned a unique test program within the confines of our instruction set that will test the functionality of the design. We are to not only show the behavioral simulation works in ModelSim, but we are also assigned to make sure the simulation works synthesized through Design Vision as well. Other than the preliminary specifications of 16 bit-wide bus and various inputs and outputs (clk, reset, MFC, R_W, EN) for data/addressing along with program memory and data memory being mapped to a single address space, it was left to us to make a number of assumptions on exactly how to put all the pieces together.

Design Methodology

Registers and Logic Modules

- Program Counter (PC)

The program counter (PC) is designed to tell the MAR where to look for the next instruction on every fetch. Notable inputs are the tri enable/PC_out and the increment/PC_inc signal, which is set to increase the value of the counter by one if raised to ‘1’ on every clock edge. Like all modules that drive the bus, the only time PC’s data is transferred to the bus is when PC_out, or the tri enable, goes high. Reset sets the value back to 16’h0000.

- Memory Address Register (MAR) / Instruction Register (IR)

The memory address register (MAR) and instruction register (IR) are one in the same. Their functionality is identical, therefore I designed one module for both and instantiated it twice at top level. They both check for an input enable signal and latches data from the bus if high. Since neither of these registers drive the bus (MAR goes to memory and IR goes to ID), they're outputting at all times to their respective destinations. Reset sets the value back to 16’h0000 (same as all others).

- Memory Data Register (MDR)

The memory data register (MDR) is the most complex of all the registers in the system. It’s internal structure consists of two registers, one for reading data from memory to the bus and the other for writing data from the bus to memory. On the positive edge of the clock, both check for reset and their respective In-enable signals where they latch data from their respective inputs. The only difference between the two is the 'Read' register's output is dependent on a tri enable signal.

- General Purpose Registers (GPR)

The general purpose registers (GPR) are identical in form and functionality to the MAR and IR registers with one exception. Since they are meant to also be bus drivers, their output is dependent on a tri enable signal. I designed one module that is instantiated four times on the top level [GPR0 - GPR3]. Reset works as expected (16'h0000).

- Arithmetic Logic Unit (ALU)

The arithmetic logic unit (ALU) of our microprocessor needs is able to handle seven operations [add, sub, not, and, or, xor, xnor]. Internally, the ALU consists of three registers, two for operands and one for the result. On the positive edge of the clock, the ALU checks for the in-enable signals for these registers and latches data if high. Within the condition of the output register's in-enable signal going high, I have a case statement that sets the value of the result register to the result of one of the seven operations dependent on a 3-bit operation signal. Finally, like all other drivers of the bus, the result output is dependent on a tri enable signal.

- Instruction Decoder (ID)

The instruction decoder (ID) is a fairly simple logic block. The ID brings in data from the IR, at all times, and on the positive edge of the clock checks for an enable signal. Whenever this enable signal goes high, it kicks off the ID and a case statement is run on the top four bits (instruction_in[15:12] or opcode) and sends an output start signal to the corresponding finite state machine (FSM) to start execution of the instruction. The ID passes the separate outputs of Opcode, Ri, and Rj for all of the FSM's to work with.

Control Signal Generation (Moore-Style Finite State Machines)

All FSM's described in this report are Moore-Style FSM's with their outputs being solely dependent on their current state. They are all designed using three always blocks: transition, next state logic, and output logic. All of the operation FSM's send 'finish' signals that are | | together to the instruction fetch 'start' input. Also, the first stage of all the FSM's besides fetch is incrementing the PC to ensure execution keeps flowing.

- <u>Instruction Fetch</u>

My fetch FSM design is designed to be start whenever reset goes high or its start signal set by the other FSM's finish signal goes high. Whenever either of these conditions are met, the fetch process is kicked off. The FSM is responsible for sending out control signals that send the program counter's value to the MAR, waits MFC (memory sends data to MDR), and sends the MDR's data to the IR across the bus then finally kicks off the ID enable signal to decode the instruction. Once the instruction is decoded it kicks off the respective FSM for that instruction and thus the cycle continues.

- <u>R-type</u>

The R-Type FSM is kicked off whenever the ID receives an ADD, SUB, NOT, AND, OR, XOR, or XNOR instruction. The FSM also receives both operands from the ID. Once it starts, the FSM latches the first operand's register contents into the first ALU register, then does the same for the second operand into the second ALU register. Finally, it sends the signals to calculate the result and sets the first operand's register up to receive the data from the ALU across the bus.

- <u>Immediate-type</u>

The Immediate FSM is very similar to the R-Type FSM and is kicked off whenever the ID receives an ADDi or SUBi instruction. The Immediate FSM follows the same pattern of execution as the R-Type FSM, with the one exception and thats instead of latching register data to the second ALU register, the FSM drives the bus with an internal enable signal on a tri state output and sends the second operand to the ALU.

- <u>Move</u>

The Move FSM is extremely simple in its design. In three stages, the FSM increments the PC, and sets the corresponding signals to output data from the source register to the bus and latch it in the destination register.

- <u>Move Immediate</u>

The Move Immediate FSM does the exact same execution flow as the Move FSM except that instead of sending data from a source register, it sends data to the bus directly from the second operand much like the Immediate FSM with its internal tri enable signal.

- Store

The Store FSM sends the destination GPR’s value across the bus to the MAR then sends the source GPR’s value across the bus and latches to the MDR write register. The FSM then sets EN = 1 (memory enable), R_W = 0 (write) and waits for MFC. Once MFC goes high, we know the data is sent to memory and the FSM and finish execution.

- Load

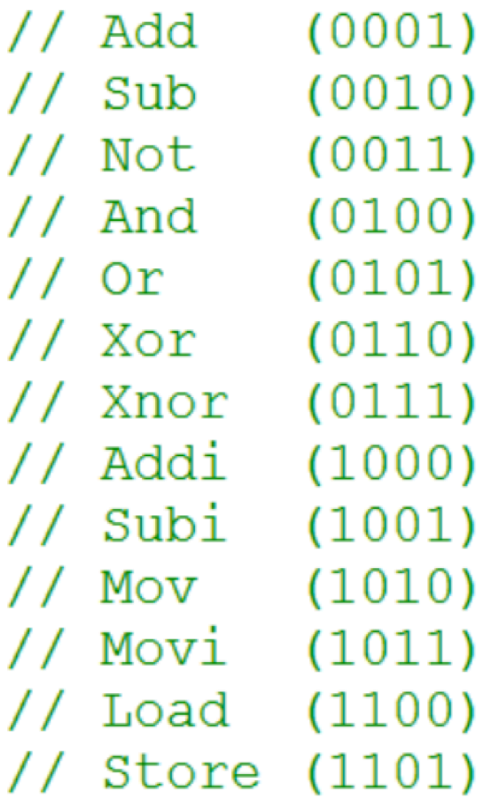
The Load FSM is much like the Store except a little more complicated. First off, the FSM sends the source GPR to MAR across the bus. After that, EN and R_W are both set to 1 to indicate a ‘read’ operation. Once MFC goes high, we know the memory has our data ready and the FSM latches it into the read register in the MDR which then sends it to the destination GPR.

Top Level Design

The top level design wasn't too hard to implement since I kept fairly good track of signals in the steps along the way. I declare all my modules and signals, the important part comes in making the wire connection. All of my GPR in/out signals are the OR outputs of all the FSM GPR signals. The same goes for PC increment signals, Memory R_W, EN, MDR/MAR latch signals, ALU signals, and finally sets the instruction fetch start signal to the output of the OR of all the other FSM finish signals.

Instruction Set and Other Assumptions

The figure on the right is the way I break down the instructions into machine opcode. “0000” I intentionally left blank to ensure the CPU would stop running at the end of the program execution. In our instruction set, the GPR registers are numbered 0, 1, 2, 3. In my design files, they are numbered 1, 2, 3, 4 respectively. This means, for example, ADD R0, R1 would translate to 0001000001000010. Other than that, everything else should appear to follow how I’ve described it elsewhere in the report. My testbench runs a 1 MHz clock speed to make sure any delay caused by synthesis is avoided.



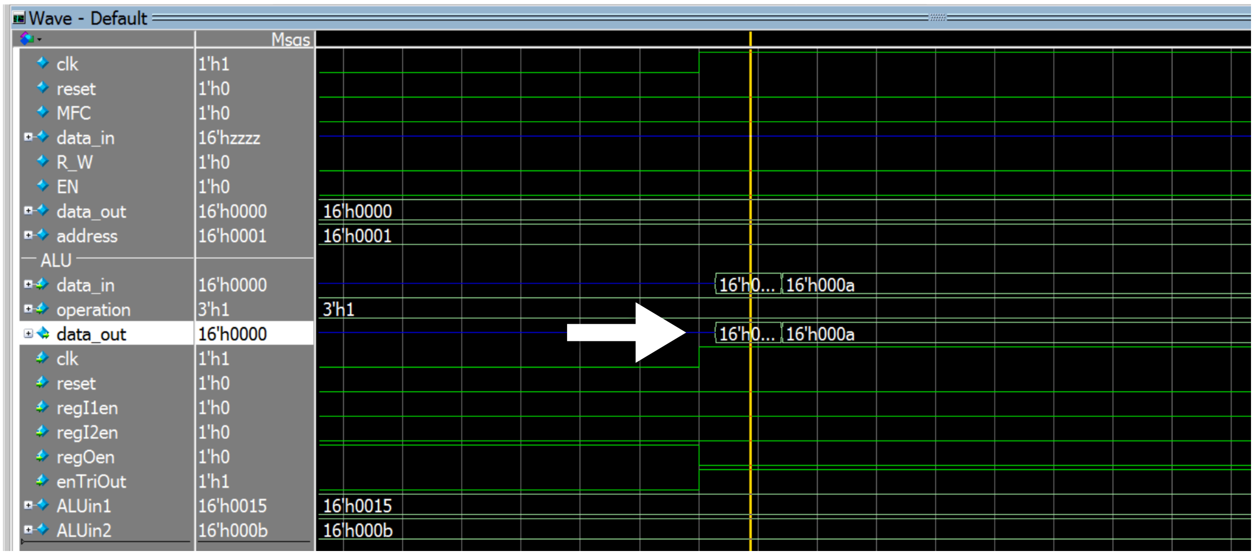
```
// Add    (0001)
// Sub    (0010)
// Not    (0011)
// And    (0100)
// Or     (0101)
// Xor    (0110)
// Xnor   (0111)
// Addi   (1000)
// Subi   (1001)
// Mov    (1010)
// Movi   (1011)
// Load   (1100)
// Store  (1101)
```

Synthesis Process

My strategy with synthesizing the components was to get the behavioral simulation working and synthesize blocks one at a time all under the same specifications. All of the blocks I synthesized I did with the following timing constraints: 50 ns clock cycle with input delay of 2 ns, output delay of 0.5 ns, and a clock uncertainty of 0.15 ns.

I was able to successfully synthesize and simulate with gate delay the entire processor, with one exception being the Instruction Fetch FSM. Other than the Instruction Fetch FSM, every module is simulated post-synthesis. The following are screenshots to show the gate delay in all synthesized modules. Positive edge of the clock is always at the top for delay reference.

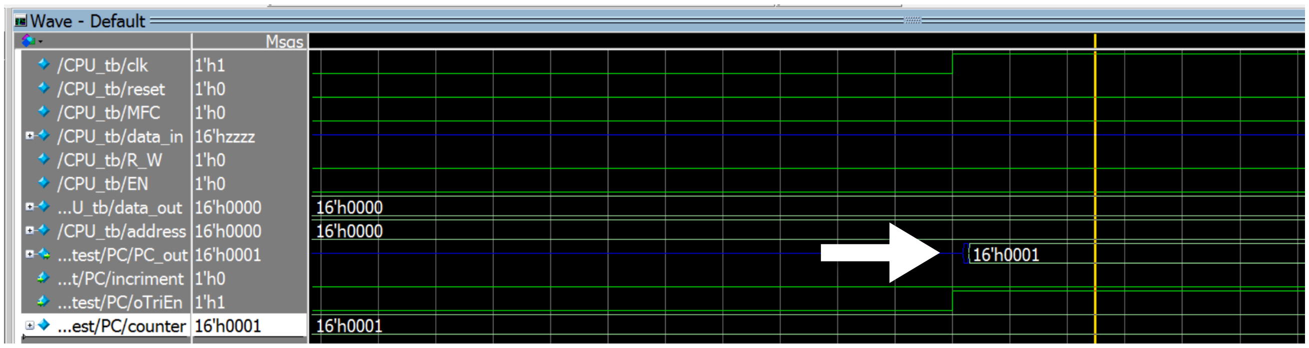
ALU



Wave - Default

	Msgs	
clk	1'h1	
reset	1'h0	
MFC	1'h0	
data_in	16'hzzzz	
R_W	1'h0	
EN	1'h0	
data_out	16'h0000	16'h0000
address	16'h0001	16'h0001
ALU		
data_in	16'h0000	16'h0... 16'h000a
operation	3'h1	3'h1
data_out	16'h0000	16'h0... 16'h000a
clk	1'h1	
reset	1'h0	
regI1en	1'h0	
regI2en	1'h0	
regOen	1'h0	
enTriOut	1'h1	
ALUin1	16'h0015	16'h0015
ALUin2	16'h000b	16'h000b

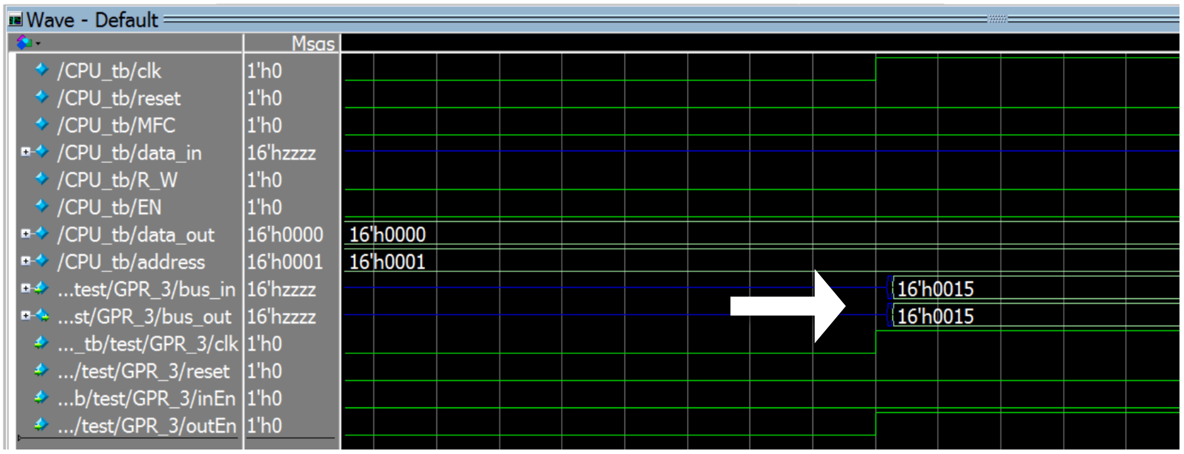
PC



Wave - Default

	Msgs	
/CPU_tb/clk	1'h1	
/CPU_tb/reset	1'h0	
/CPU_tb/MFC	1'h0	
/CPU_tb/data_in	16'hzzzz	
/CPU_tb/R_W	1'h0	
/CPU_tb/EN	1'h0	
...U_tb/data_out	16'h0000	16'h0000
/CPU_tb/address	16'h0000	16'h0000
...test/PC/PC_out	16'h0001	16'h0001
...t/PC/incriment	1'h0	
...test/PC/oTriEn	1'h1	
...est/PC/counter	16'h0001	16'h0001

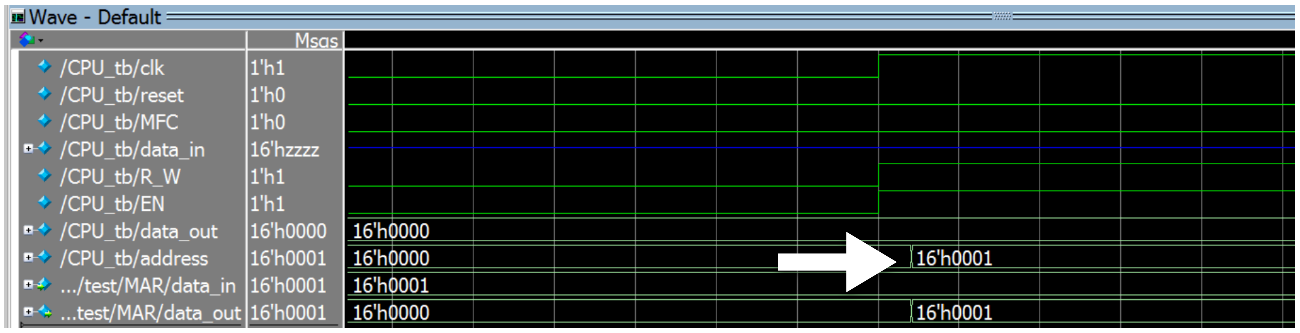
GPR



Wave - Default

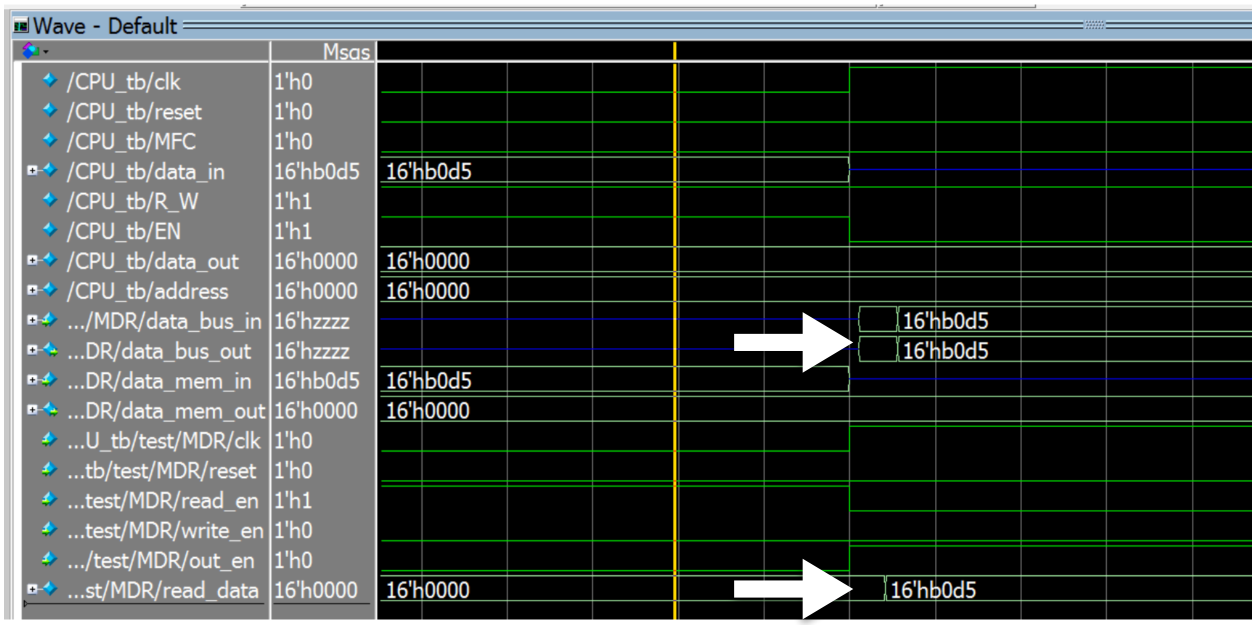
	Msgs	
/CPU_tb/clk	1'h0	
/CPU_tb/reset	1'h0	
/CPU_tb/MFC	1'h0	
/CPU_tb/data_in	16'hzzzz	
/CPU_tb/R_W	1'h0	
/CPU_tb/EN	1'h0	
/CPU_tb/data_out	16'h0000	16'h0000
/CPU_tb/address	16'h0001	16'h0001
...test/GPR_3/bus_in	16'hzzzz	16'h0015
...st/GPR_3/bus_out	16'hzzzz	16'h0015
..._tb/test/GPR_3/clk	1'h0	
.../test/GPR_3/reset	1'h0	
...b/test/GPR_3/inEn	1'h0	
.../test/GPR_3/outEn	1'h0	

MAR


Wave - Default

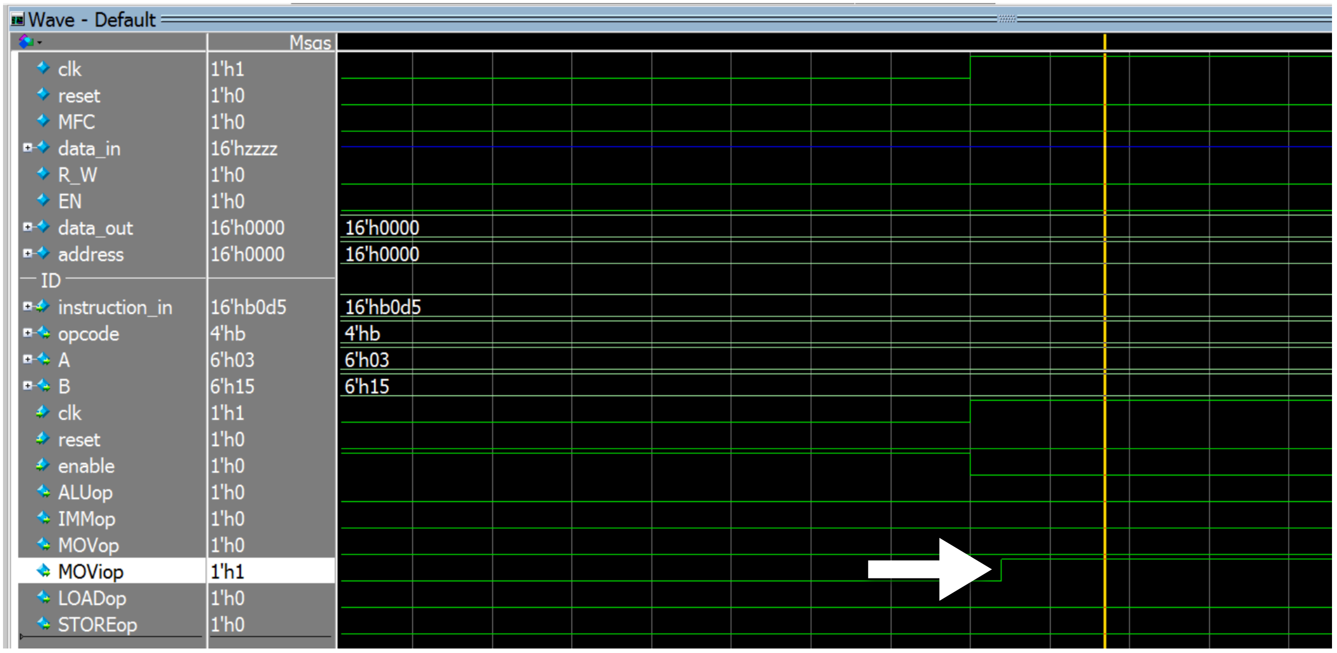
	Msgs	
/CPU_tb/clk	1'h1	
/CPU_tb/reset	1'h0	
/CPU_tb/MFC	1'h0	
/CPU_tb/data_in	16'hzzzz	
/CPU_tb/R_W	1'h1	
/CPU_tb/EN	1'h1	
/CPU_tb/data_out	16'h0000	16'h0000
/CPU_tb/address	16'h0001	16'h0000 → 16'h0001
.../test/MAR/data_in	16'h0001	16'h0001
...test/MAR/data_out	16'h0001	16'h0000 → 16'h0001

MDR


Wave - Default

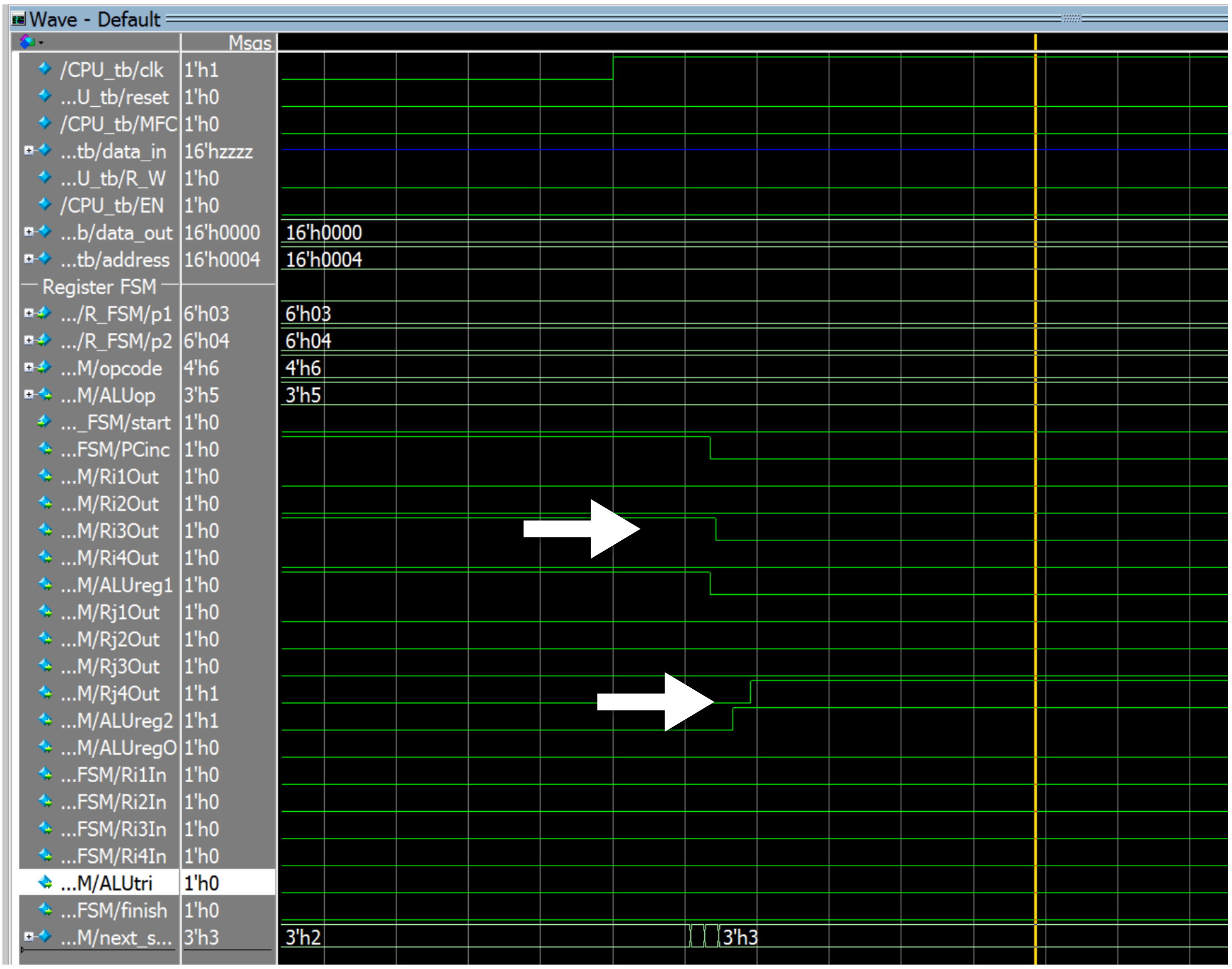
	Msgs	
/CPU_tb/clk	1'h0	
/CPU_tb/reset	1'h0	
/CPU_tb/MFC	1'h0	
/CPU_tb/data_in	16'hb0d5	16'hb0d5
/CPU_tb/R_W	1'h1	
/CPU_tb/EN	1'h1	
/CPU_tb/data_out	16'h0000	16'h0000
/CPU_tb/address	16'h0000	16'h0000
.../MDR/data_bus_in	16'hzzzz	16'hb0d5
...DR/data_bus_out	16'hzzzz	16'hb0d5
...DR/data_mem_in	16'hb0d5	16'hb0d5
...DR/data_mem_out	16'h0000	16'h0000
...U_tb/test/MDR/clk	1'h0	
...tb/test/MDR/reset	1'h0	
...test/MDR/read_en	1'h1	
...test/MDR/write_en	1'h0	
.../test/MDR/out_en	1'h0	
...st/MDR/read_data	16'h0000	16'h0000 → 16'hb0d5

ID


Wave - Default

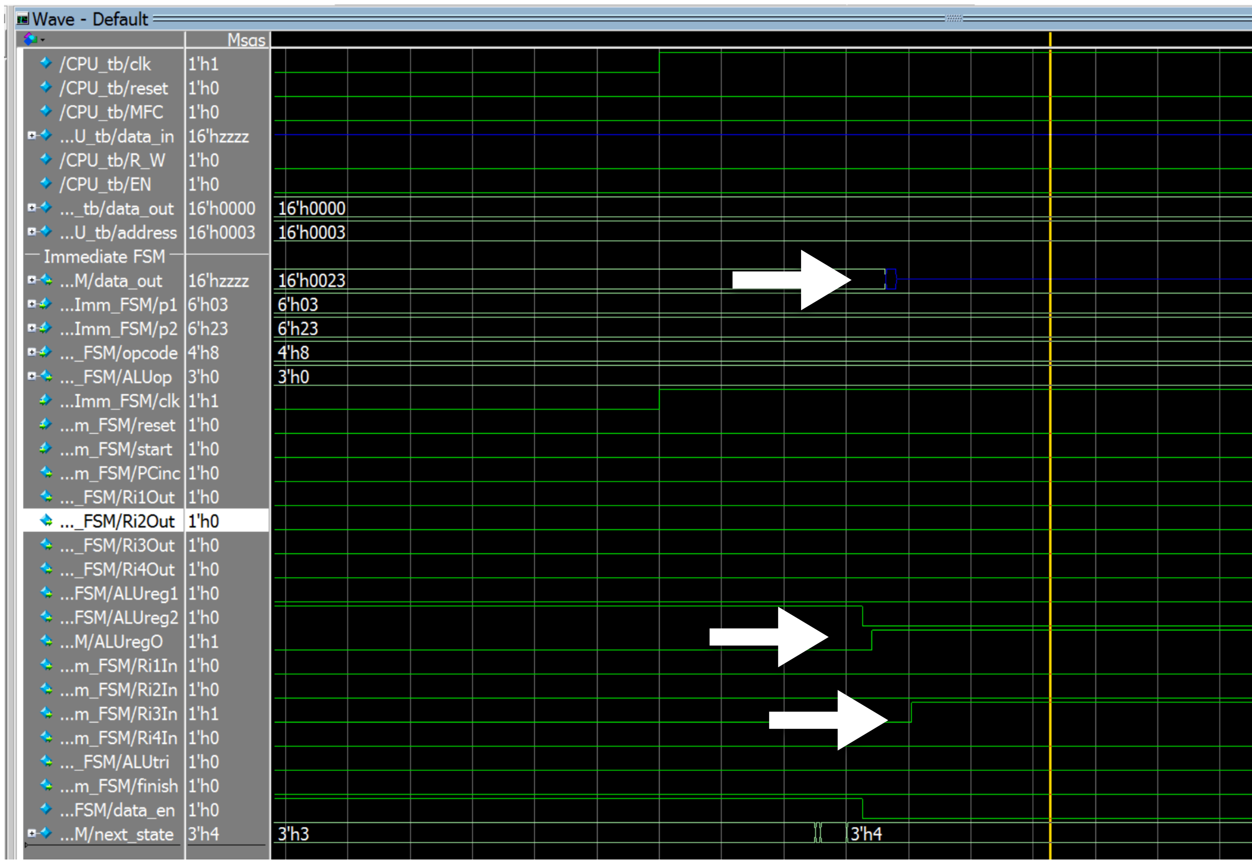
	Msgs	
clk	1'h1	
reset	1'h0	
MFC	1'h0	
data_in	16'hzzzz	
R_W	1'h0	
EN	1'h0	
data_out	16'h0000	16'h0000
address	16'h0000	16'h0000
ID		
instruction_in	16'hb0d5	16'hb0d5
opcode	4'hb	4'hb
A	6'h03	6'h03
B	6'h15	6'h15
clk	1'h1	
reset	1'h0	
enable	1'h0	
ALUop	1'h0	
IMMop	1'h0	
MOVop	1'h0	
MOViop	1'h1	
LOADop	1'h0	
STOREop	1'h0	

R-type



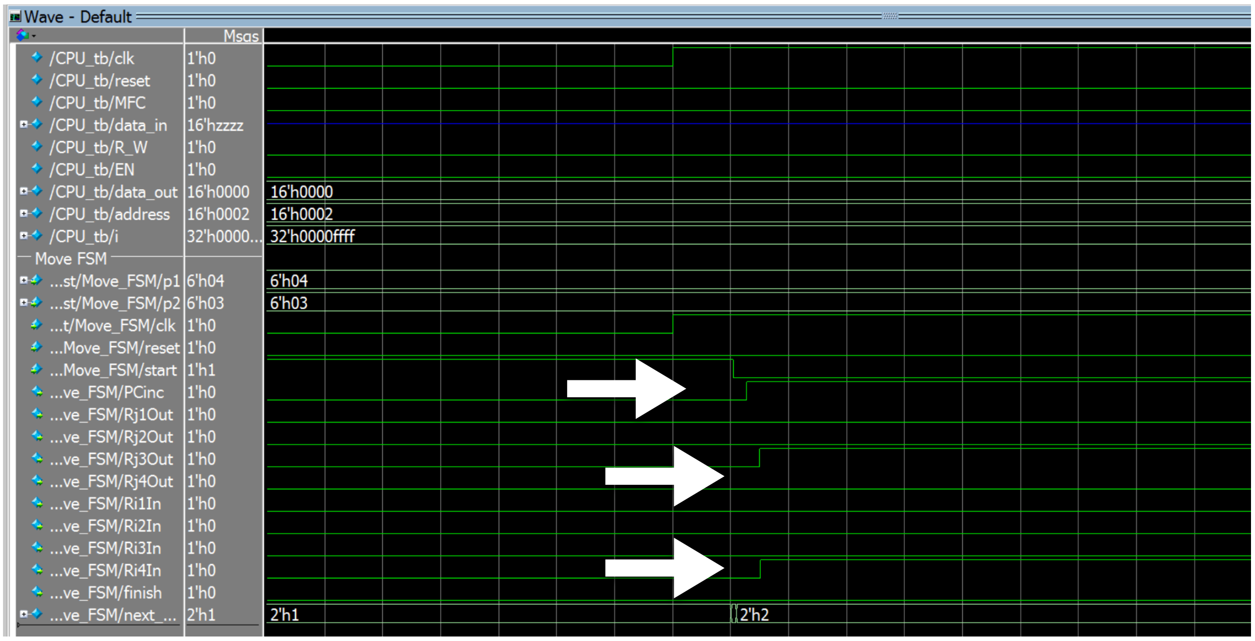
```
Wave - Default
                  Msgs
/CPU_tb/clk       1'h1
...U_tb/reset     1'h0
/CPU_tb/MFC       1'h0
...tb/data_in     16'hzzzz
...U_tb/R_W       1'h0
/CPU_tb/EN        1'h0
...b/data_out     16'h0000    16'h0000
...tb/address     16'h0004    16'h0004
Register FSM
.../R_FSM/p1      6'h03       6'h03
.../R_FSM/p2      6'h04       6'h04
...M/opcode       4'h6        4'h6
...M/ALUop        3'h5        3'h5
..._FSM/start     1'h0
...FSM/PCinc      1'h0
...M/Ri1Out       1'h0
...M/Ri2Out       1'h0
...M/Ri3Out       1'h0
...M/Ri4Out       1'h0
...M/ALUreg1      1'h0
...M/Rj1Out       1'h0
...M/Rj2Out       1'h0
...M/Rj3Out       1'h0
...M/Rj4Out       1'h1
...M/ALUreg2      1'h1
...M/ALUregO      1'h0
...FSM/Ri1In      1'h0
...FSM/Ri2In      1'h0
...FSM/Ri3In      1'h0
...FSM/Ri4In      1'h0
...M/ALUtri       1'h0
...FSM/finish     1'h0
...M/next_s...    3'h3        3'h2        3'h3
```

Immediate



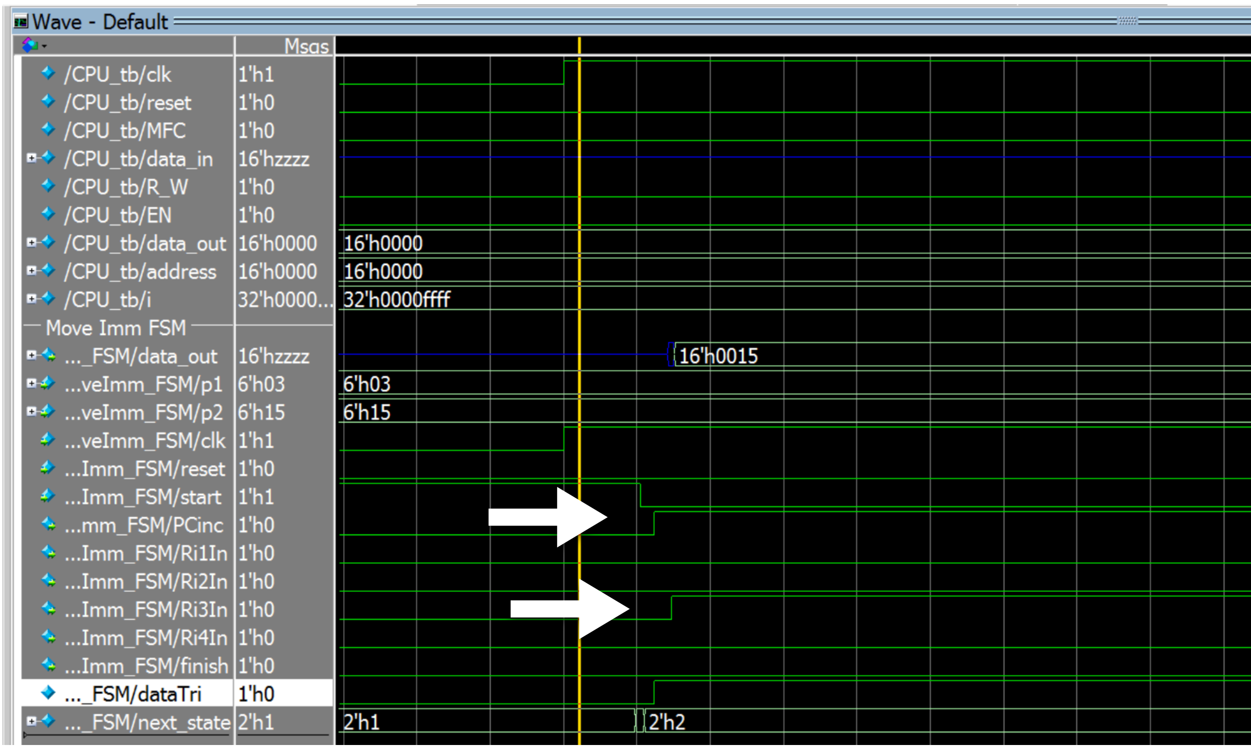
```
Wave - Default
                      Msgs
/CPU_tb/clk           1'h1
/CPU_tb/reset         1'h0
/CPU_tb/MFC           1'h0
...U_tb/data_in       16'hzzzz
/CPU_tb/R_W           1'h0
/CPU_tb/EN            1'h0
..._tb/data_out       16'h0000    16'h0000
...U_tb/address       16'h0003    16'h0003
Immediate FSM
...M/data_out         16'hzzzz    16'h0023
...Imm_FSM/p1         6'h03       6'h03
...Imm_FSM/p2         6'h23       6'h23
..._FSM/opcode        4'h8        4'h8
..._FSM/ALUop         3'h0        3'h0
...Imm_FSM/clk        1'h1
...m_FSM/reset        1'h0
...m_FSM/start        1'h0
...m_FSM/PCinc        1'h0
..._FSM/Ri1Out        1'h0
..._FSM/Ri2Out        1'h0
..._FSM/Ri3Out        1'h0
..._FSM/Ri4Out        1'h0
...FSM/ALUreg1        1'h0
...FSM/ALUreg2        1'h0
...M/ALUregO          1'h1
...m_FSM/Ri1In        1'h0
...m_FSM/Ri2In        1'h0
...m_FSM/Ri3In        1'h1
...m_FSM/Ri4In        1'h0
..._FSM/ALUtri        1'h0
...m_FSM/finish       1'h0
...FSM/data_en        1'h0
...M/next_state       3'h4        3'h3        3'h4
```

Move



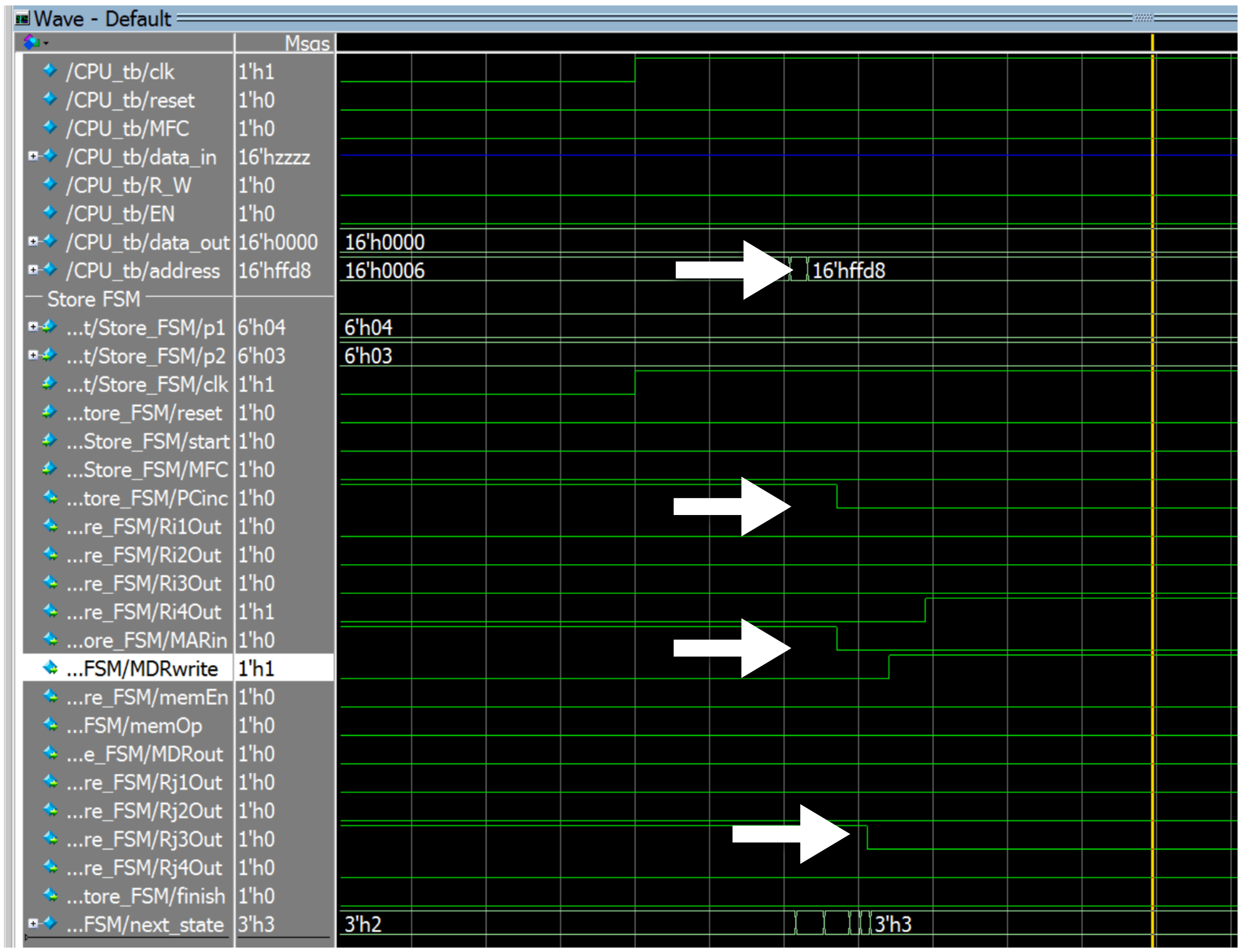
```
Wave - Default
                      Msgs
/CPU_tb/clk           1'h0
/CPU_tb/reset         1'h0
/CPU_tb/MFC           1'h0
/CPU_tb/data_in       16'hzzzz
/CPU_tb/R_W           1'h0
/CPU_tb/EN            1'h0
/CPU_tb/data_out      16'h0000      16'h0000
/CPU_tb/address       16'h0002      16'h0002
/CPU_tb/i             32'h0000...   32'h0000ffff
Move FSM
...st/Move_FSM/p1     6'h04         6'h04
...st/Move_FSM/p2     6'h03         6'h03
...t/Move_FSM/clk     1'h0
...Move_FSM/reset     1'h0
...Move_FSM/start     1'h1
...ve_FSM/PCinc       1'h0
...ve_FSM/Rj1Out      1'h0
...ve_FSM/Rj2Out      1'h0
...ve_FSM/Rj3Out      1'h0
...ve_FSM/Rj4Out      1'h0
...ve_FSM/Ri1In       1'h0
...ve_FSM/Ri2In       1'h0
...ve_FSM/Ri3In       1'h0
...ve_FSM/Ri4In       1'h0
...ve_FSM/finish      1'h0
...ve_FSM/next_...    2'h1          2'h1                2'h2
```

MoveImm



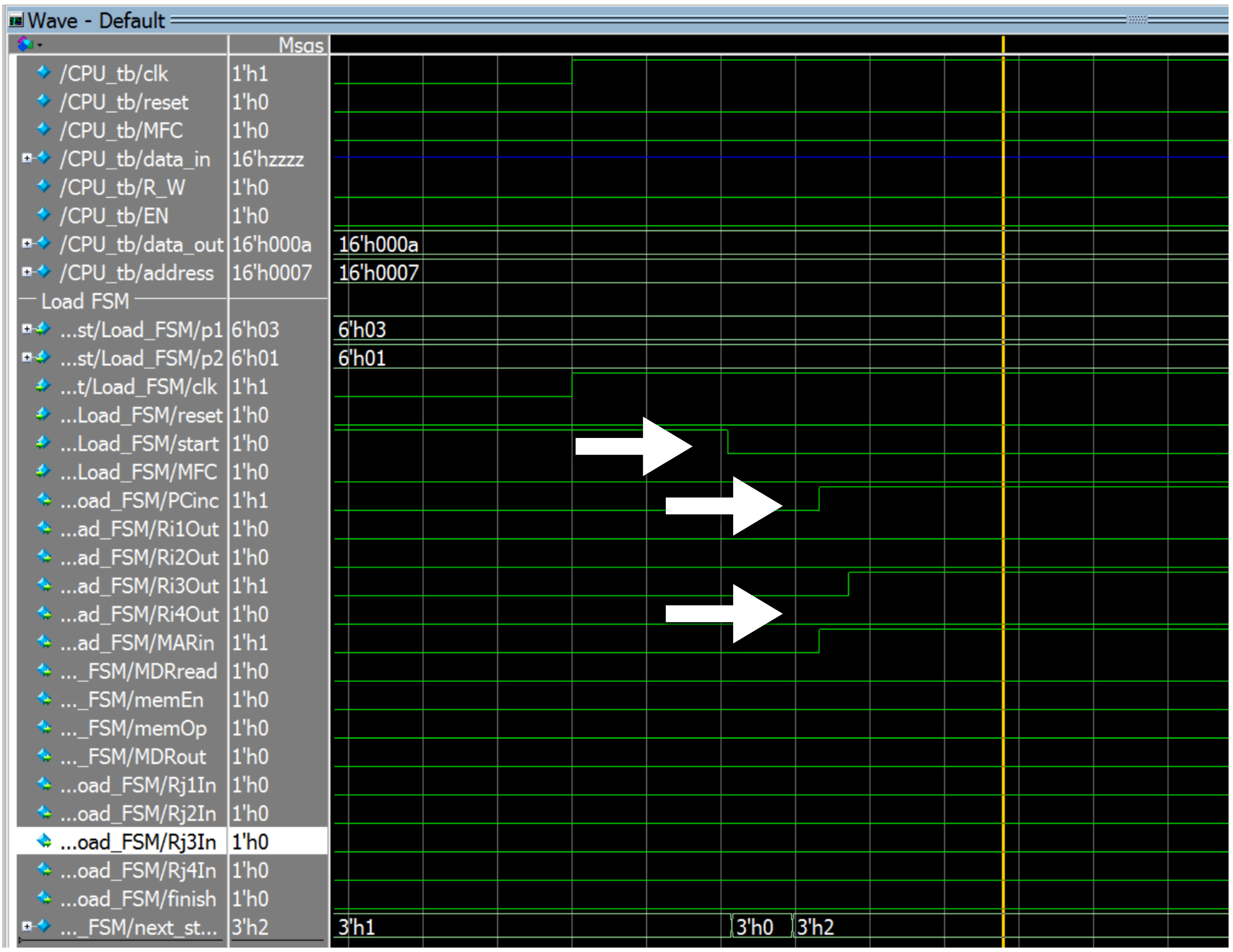
```
Wave - Default
                        Msgs
/CPU_tb/clk             1'h1
/CPU_tb/reset           1'h0
/CPU_tb/MFC             1'h0
/CPU_tb/data_in         16'hzzzz
/CPU_tb/R_W             1'h0
/CPU_tb/EN              1'h0
/CPU_tb/data_out        16'h0000      16'h0000
/CPU_tb/address         16'h0000      16'h0000
/CPU_tb/i               32'h0000...   32'h0000ffff
Move Imm FSM
..._FSM/data_out        16'hzzzz                          16'h0015
...veImm_FSM/p1         6'h03         6'h03
...veImm_FSM/p2         6'h15         6'h15
...veImm_FSM/clk        1'h1
...Imm_FSM/reset        1'h0
...Imm_FSM/start        1'h1
...mm_FSM/PCinc         1'h0
...Imm_FSM/Ri1In        1'h0
...Imm_FSM/Ri2In        1'h0
...Imm_FSM/Ri3In        1'h0
...Imm_FSM/Ri4In        1'h0
...Imm_FSM/finish       1'h0
..._FSM/dataTri         1'h0
..._FSM/next_state      2'h1          2'h1                2'h2
```

<u>Store</u>



```
Wave - Default
                          Msgs
/CPU_tb/clk               1'h1
/CPU_tb/reset             1'h0
/CPU_tb/MFC               1'h0
/CPU_tb/data_in           16'hzzzz
/CPU_tb/R_W               1'h0
/CPU_tb/EN                1'h0
/CPU_tb/data_out          16'h0000    16'h0000
/CPU_tb/address           16'hffd8    16'h0006          16'hffd8
Store FSM
...t/Store_FSM/p1         6'h04       6'h04
...t/Store_FSM/p2         6'h03       6'h03
...t/Store_FSM/clk        1'h1
...tore_FSM/reset         1'h0
...Store_FSM/start        1'h0
...Store_FSM/MFC          1'h0
...tore_FSM/PCinc         1'h0
...re_FSM/Ri1Out          1'h0
...re_FSM/Ri2Out          1'h0
...re_FSM/Ri3Out          1'h0
...re_FSM/Ri4Out          1'h1
...ore_FSM/MARin          1'h0
...FSM/MDRwrite           1'h1
...re_FSM/memEn           1'h0
...FSM/memOp              1'h0
...e_FSM/MDRout           1'h0
...re_FSM/Rj1Out          1'h0
...re_FSM/Rj2Out          1'h0
...re_FSM/Rj3Out          1'h0
...re_FSM/Rj4Out          1'h0
...tore_FSM/finish        1'h0
...FSM/next_state         3'h3        3'h2              3'h3
```

<u>Load</u>



```
Wave - Default
                          Msgs
/CPU_tb/clk               1'h1
/CPU_tb/reset             1'h0
/CPU_tb/MFC               1'h0
/CPU_tb/data_in           16'hzzzz
/CPU_tb/R_W               1'h0
/CPU_tb/EN                1'h0
/CPU_tb/data_out          16'h000a    16'h000a
/CPU_tb/address           16'h0007    16'h0007
Load FSM
...st/Load_FSM/p1         6'h03       6'h03
...st/Load_FSM/p2         6'h01       6'h01
...t/Load_FSM/clk         1'h1
...Load_FSM/reset         1'h0
...Load_FSM/start         1'h0
...Load_FSM/MFC           1'h0
...oad_FSM/PCinc          1'h1
...ad_FSM/Ri1Out          1'h0
...ad_FSM/Ri2Out          1'h0
...ad_FSM/Ri3Out          1'h1
...ad_FSM/Ri4Out          1'h0
...ad_FSM/MARin           1'h1
..._FSM/MDRread           1'h0
..._FSM/memEn             1'h0
..._FSM/memOp             1'h0
..._FSM/MDRout            1'h0
...oad_FSM/Rj1In          1'h0
...oad_FSM/Rj2In          1'h0
...oad_FSM/Rj3In          1'h0
...oad_FSM/Rj4In          1'h0
...oad_FSM/finish         1'h0
..._FSM/next_st...        3'h2        3'h1        3'h0  3'h2
```

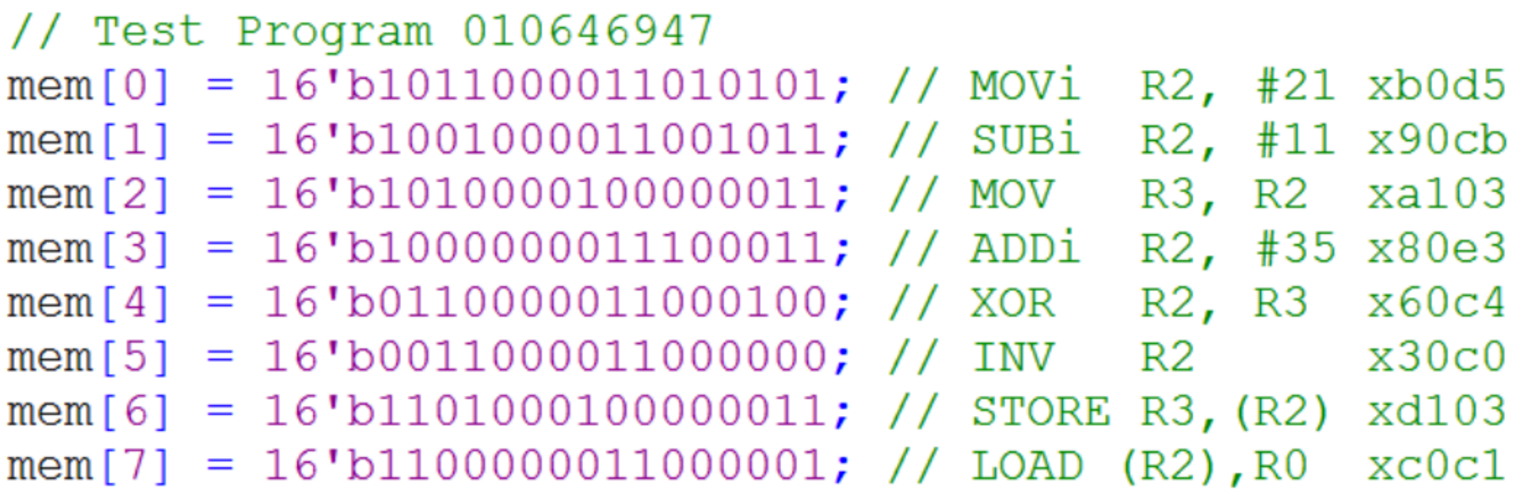
Simulation Method

The only real way to test if the processor works is to feed it a program! The figure on the right shows the program that I was assigned to show to its completion with my design. Since our top level modules do not need to be synthesized I chose to go the simplest route, at least in my opinion, and have my memory initialized in the testbench and have all the memory function be simulated in the testbench itself.

The figure below shows the program instantiated in memory and translated into machine code within my testbench. The full testbench is included in the report at the end with the rest of the source code. In the next section I will provide a screenshot of the overall waveform of the program simulation as well as individual screenshots of each of the program instructions with their transactions detailed on the bus. In the following section I explain the waveforms and provide my conclusions. In my testbench on read/write operations I have the program wait two clock cycles then MFC gets raised to '1.'

010646947

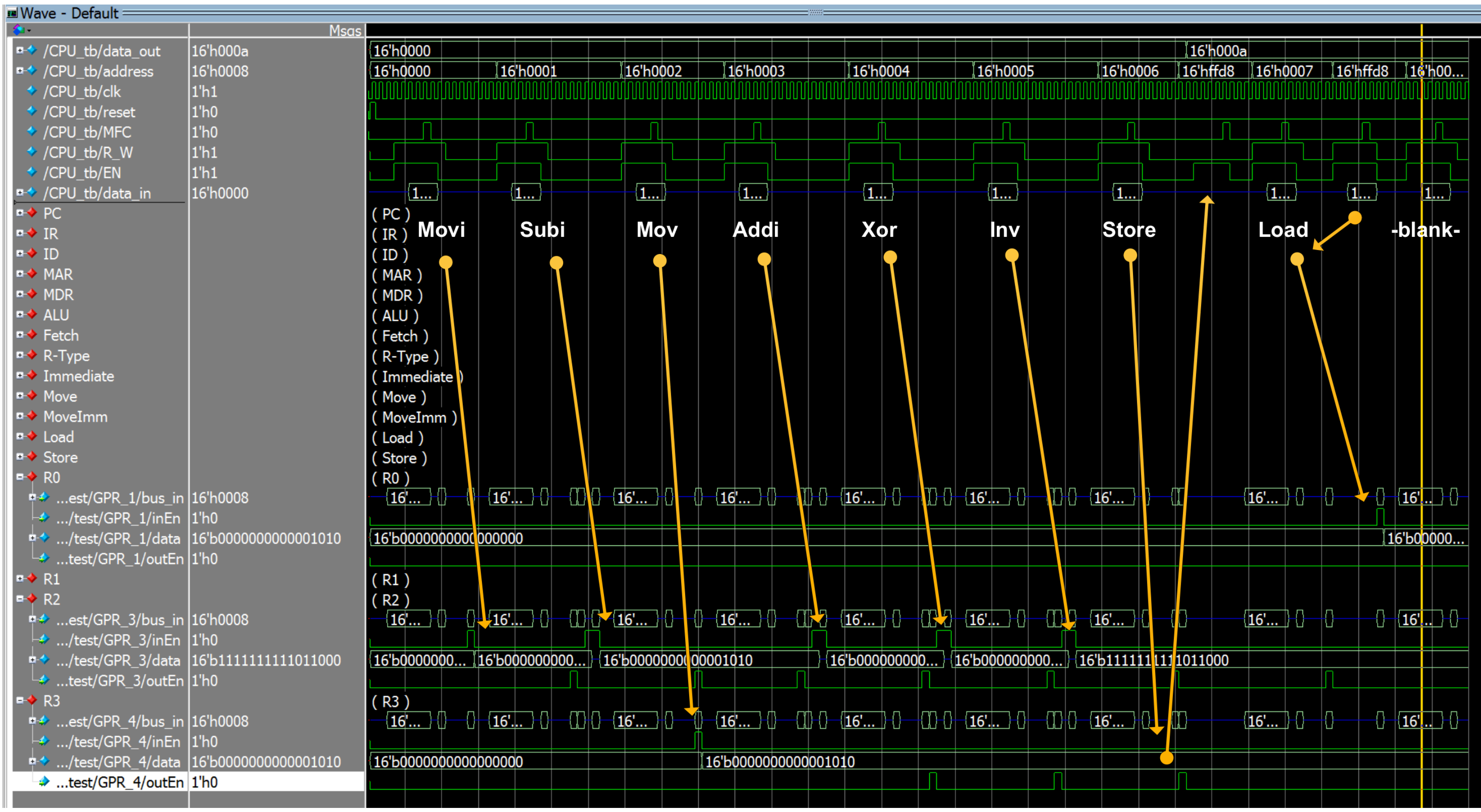
```
MOVI  R2, #21
SUBI  R2, #11
MOV   R3, R2
ADDI  R2, #35
XOR   R2, R3
INV   R2
STORE R3, (R2)
LOAD  (R2), R0
```



```
// Test Program 010646947
mem[0] = 16'b1011000011010101; // MOVi  R2, #21 xb0d5
mem[1] = 16'b1001000011001011; // SUBi  R2, #11 x90cb
mem[2] = 16'b1010000100000011; // MOV   R3, R2  xa103
mem[3] = 16'b1000000011100011; // ADDi  R2, #35 x80e3
mem[4] = 16'b0110000011000100; // XOR   R2, R3  x60c4
mem[5] = 16'b0011000011000000; // INV   R2      x30c0
mem[6] = 16'b1101000100000011; // STORE R3,(R2) xd103
mem[7] = 16'b1100000011000001; // LOAD  (R2),R0 xc0c1
```

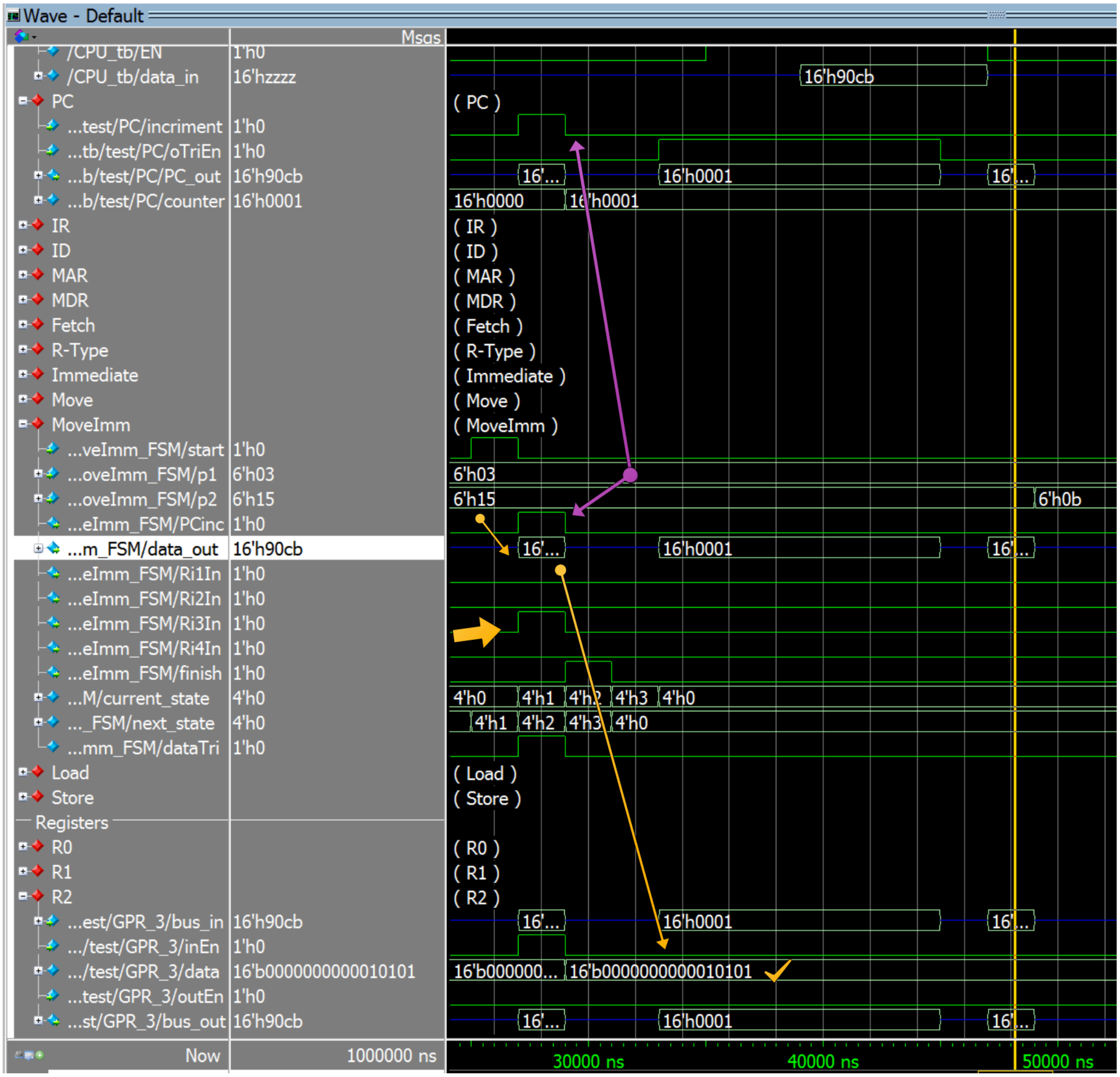
Results

Overall Program Waveform


Wave - Default

	Msgs
/CPU_tb/data_out	16'h000a
/CPU_tb/address	16'h0008
/CPU_tb/clk	1'h1
/CPU_tb/reset	1'h0
/CPU_tb/MFC	1'h0
/CPU_tb/R_W	1'h1
/CPU_tb/EN	1'h1
/CPU_tb/data_in	16'h0000
PC	
IR	
ID	
MAR	
MDR	
ALU	
Fetch	
R-Type	
Immediate	
Move	
MoveImm	
Load	
Store	
R0	
...est/GPR_1/bus_in	16'h0008
.../test/GPR_1/inEn	1'h0
.../test/GPR_1/data	16'b0000000000001010
...test/GPR_1/outEn	1'h0
R1	
R2	
...est/GPR_3/bus_in	16'h0008
.../test/GPR_3/inEn	1'h0
.../test/GPR_3/data	16'b1111111111011000
...test/GPR_3/outEn	1'h0
R3	
...est/GPR_4/bus_in	16'h0008
.../test/GPR_4/inEn	1'h0
.../test/GPR_4/data	16'b0000000000001010
...test/GPR_4/outEn	1'h0

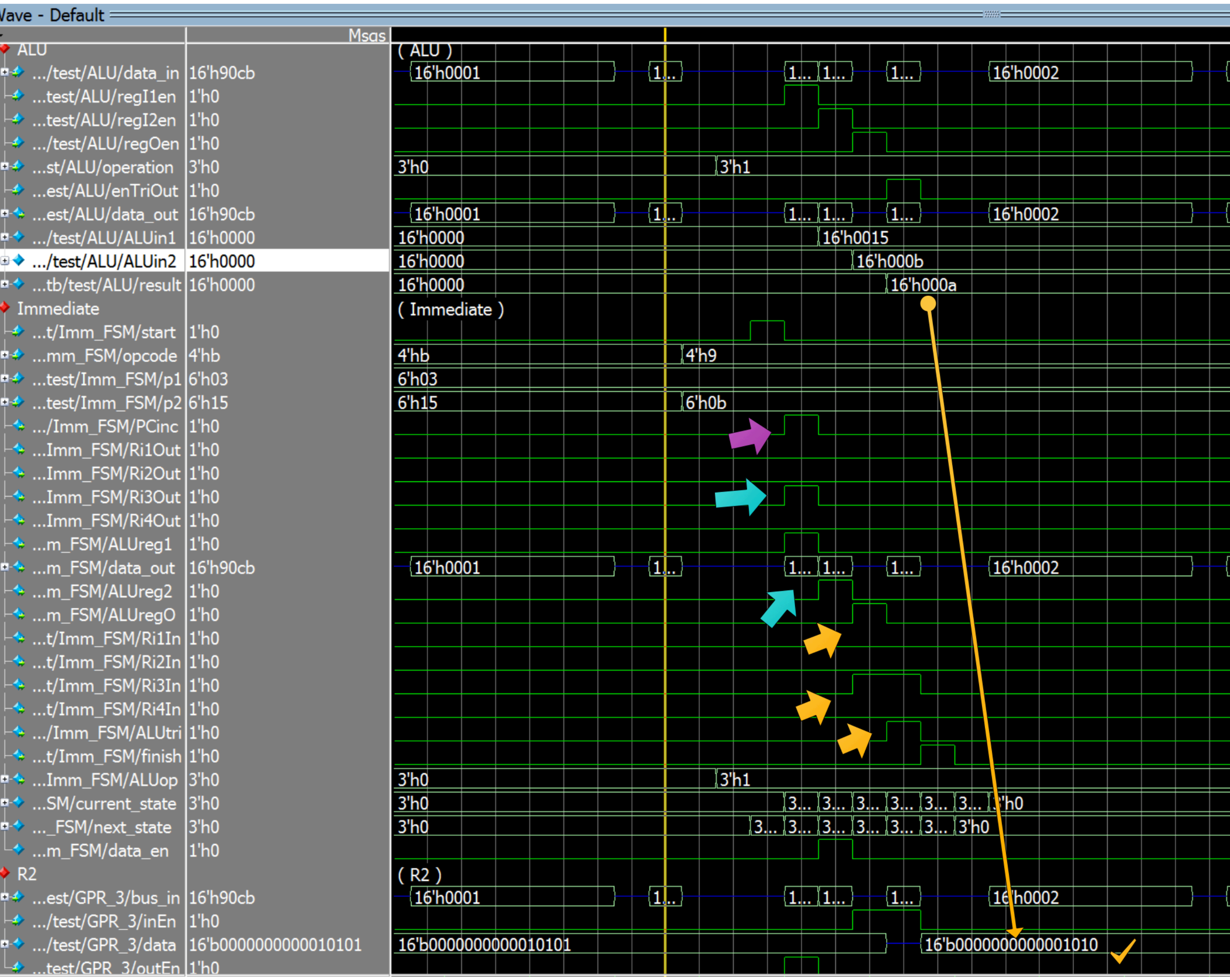
MOVI R2, #21



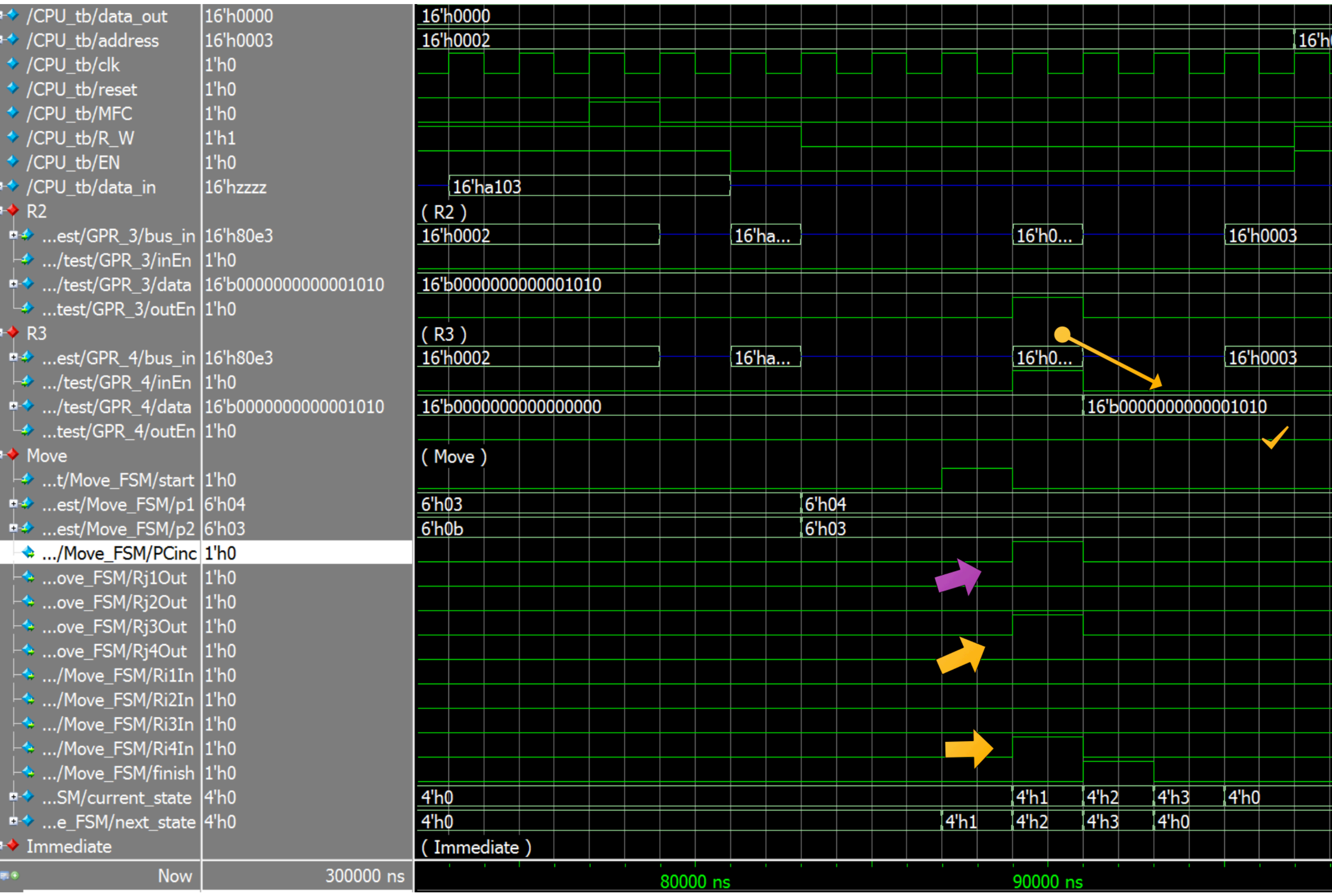
Wave - Default

	Msgs
/CPU_tb/EN	1'h0
/CPU_tb/data_in	16'hzzzz
PC	
...test/PC/incriment	1'h0
...tb/test/PC/oTriEn	1'h0
...b/test/PC/PC_out	16'h90cb
...b/test/PC/counter	16'h0001
IR	
ID	
MAR	
MDR	
Fetch	
R-Type	
Immediate	
Move	
MoveImm	
...veImm_FSM/start	1'h0
...oveImm_FSM/p1	6'h03
...oveImm_FSM/p2	6'h15
...eImm_FSM/PCinc	1'h0
...m_FSM/data_out	16'h90cb
...eImm_FSM/Ri1In	1'h0
...eImm_FSM/Ri2In	1'h0
...eImm_FSM/Ri3In	1'h0
...eImm_FSM/Ri4In	1'h0
...eImm_FSM/finish	1'h0
...M/current_state	4'h0
..._FSM/next_state	4'h0
...mm_FSM/dataTri	1'h0
Load	
Store	
Registers	
R0	
R1	
R2	
...est/GPR_3/bus_in	16'h90cb
.../test/GPR_3/inEn	1'h0
.../test/GPR_3/data	16'b0000000000010101
...test/GPR_3/outEn	1'h0
...st/GPR_3/bus_out	16'h90cb
Now	1000000 ns

SUBI R2, #11

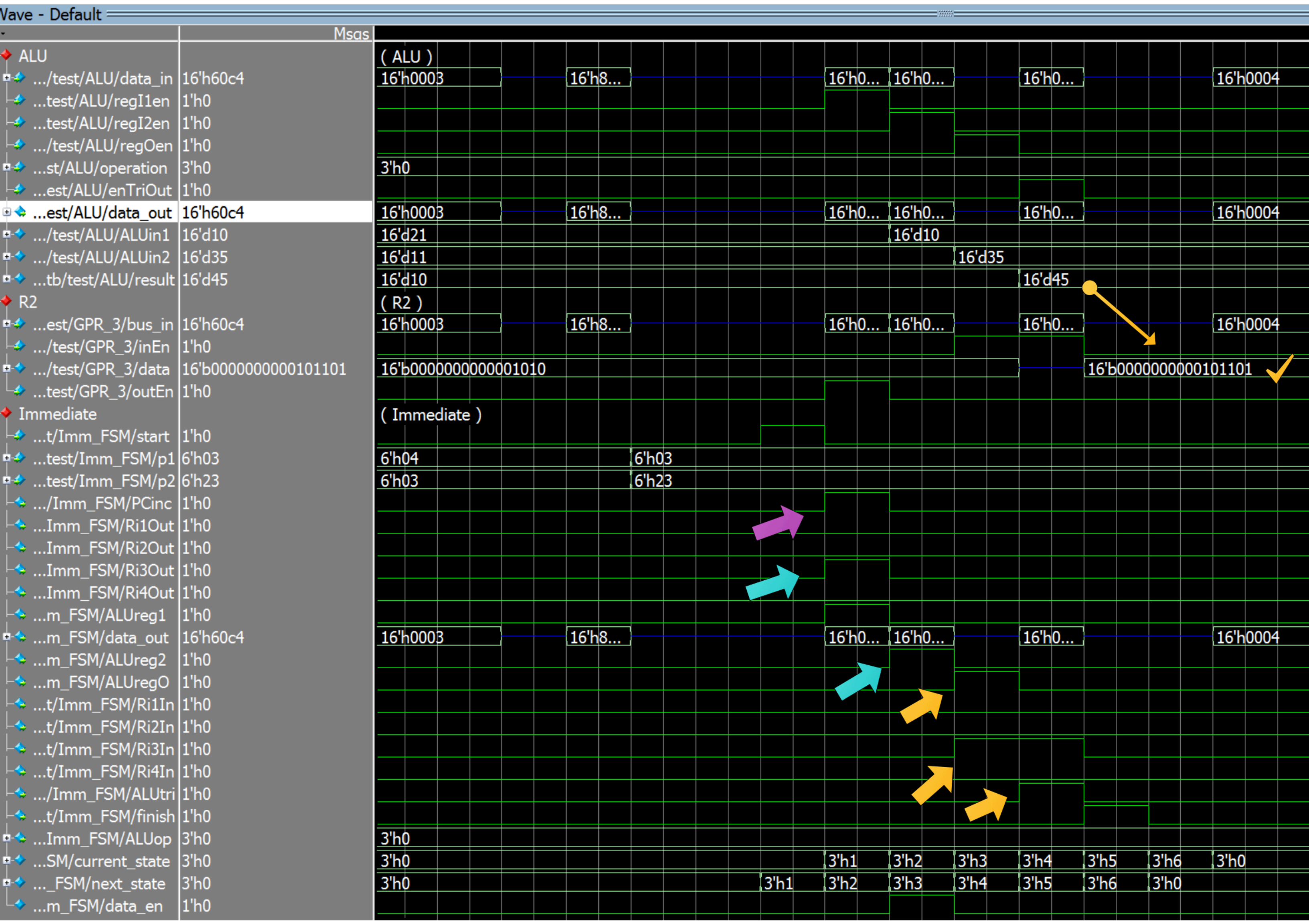

Wave - Default
Msgs
ALU
.../test/ALU/data_in 16'h90cb
...test/ALU/regI1en 1'h0
...test/ALU/regI2en 1'h0
.../test/ALU/regOen 1'h0
...st/ALU/operation 3'h0
...est/ALU/enTriOut 1'h0
...est/ALU/data_out 16'h90cb
.../test/ALU/ALUin1 16'h0000
.../test/ALU/ALUin2 16'h0000
...tb/test/ALU/result 16'h0000
Immediate
...t/Imm_FSM/start 1'h0
...mm_FSM/opcode 4'hb
...test/Imm_FSM/p1 6'h03
...test/Imm_FSM/p2 6'h15
.../Imm_FSM/PCinc 1'h0
...Imm_FSM/Ri1Out 1'h0
...Imm_FSM/Ri2Out 1'h0
...Imm_FSM/Ri3Out 1'h0
...Imm_FSM/Ri4Out 1'h0
...m_FSM/ALUreg1 1'h0
...m_FSM/data_out 16'h90cb
...m_FSM/ALUreg2 1'h0
...m_FSM/ALUregO 1'h0
...t/Imm_FSM/Ri1In 1'h0
...t/Imm_FSM/Ri2In 1'h0
...t/Imm_FSM/Ri3In 1'h0
...t/Imm_FSM/Ri4In 1'h0
.../Imm_FSM/ALUtri 1'h0
...t/Imm_FSM/finish 1'h0
...Imm_FSM/ALUop 3'h0
...SM/current_state 3'h0
..._FSM/next_state 3'h0
...m_FSM/data_en 1'h0
R2
...est/GPR_3/bus_in 16'h90cb
.../test/GPR_3/inEn 1'h0
.../test/GPR_3/data 16'b0000000000010101
...test/GPR_3/outEn 1'h0
(ALU)
16'h0001
16'h0002
3'h0
3'h1
16'h0000
16'h0015
16'h000b
16'h000a
(Immediate)
4'hb
4'h9
6'h03
6'h15
6'h0b
(R2)
16'b0000000000010101
16'b0000000000001010
✓

MOV R3, R2



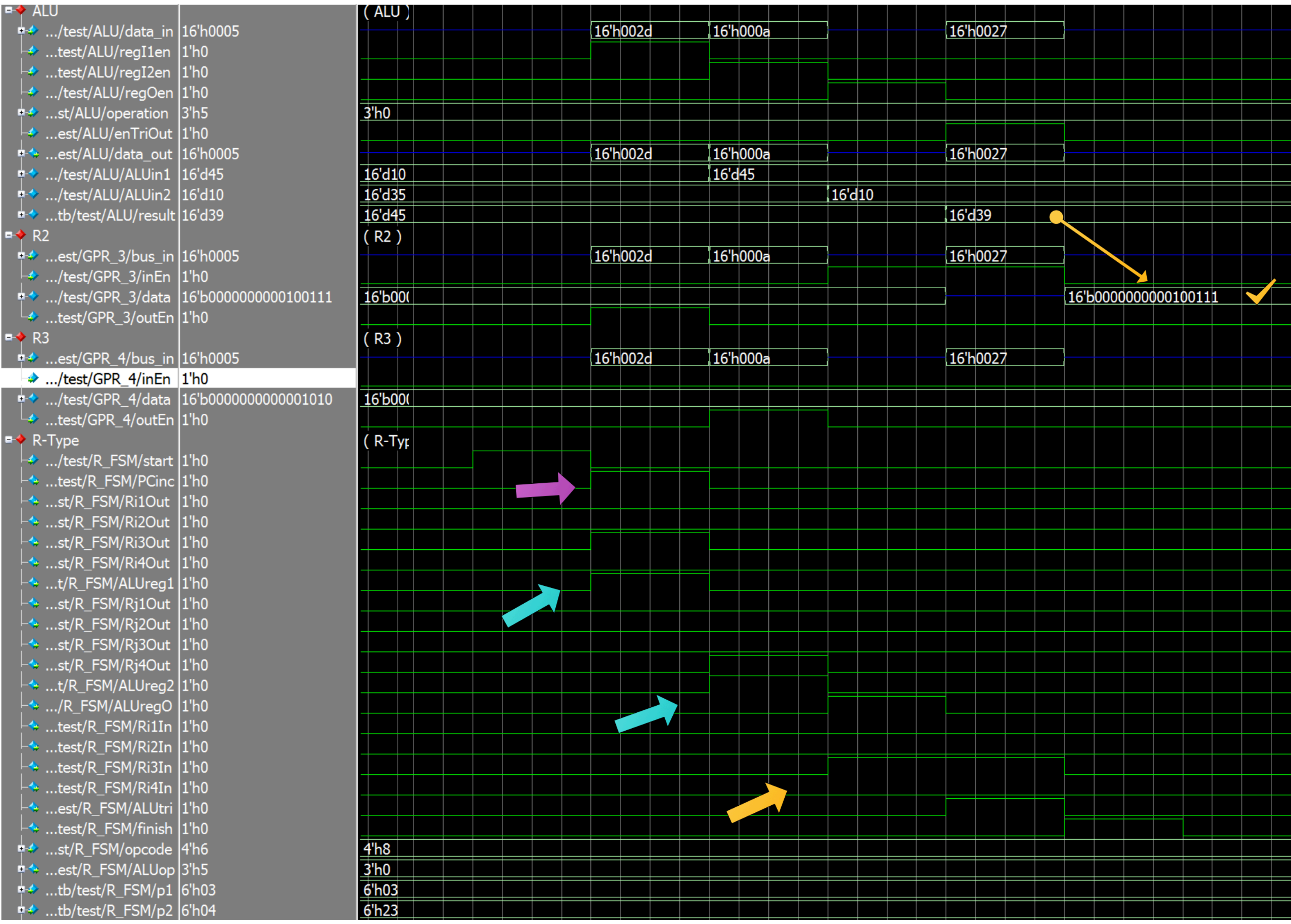
```
/CPU_tb/data_out            16'h0000
/CPU_tb/address             16'h0003
/CPU_tb/clk                 1'h0
/CPU_tb/reset               1'h0
/CPU_tb/MFC                 1'h0
/CPU_tb/R_W                 1'h1
/CPU_tb/EN                  1'h0
/CPU_tb/data_in             16'hzzzz
R2
...est/GPR_3/bus_in         16'h80e3
.../test/GPR_3/inEn         1'h0
.../test/GPR_3/data         16'b0000000000001010
...test/GPR_3/outEn         1'h0
R3
...est/GPR_4/bus_in         16'h80e3
.../test/GPR_4/inEn         1'h0
.../test/GPR_4/data         16'b0000000000001010
...test/GPR_4/outEn         1'h0
Move
...t/Move_FSM/start         1'h0
...est/Move_FSM/p1          6'h04
...est/Move_FSM/p2          6'h03
.../Move_FSM/PCinc          1'h0
...ove_FSM/Rj1Out           1'h0
...ove_FSM/Rj2Out           1'h0
...ove_FSM/Rj3Out           1'h0
...ove_FSM/Rj4Out           1'h0
.../Move_FSM/Ri1In          1'h0
.../Move_FSM/Ri2In          1'h0
.../Move_FSM/Ri3In          1'h0
.../Move_FSM/Ri4In          1'h0
.../Move_FSM/finish         1'h0
...SM/current_state         4'h0
...e_FSM/next_state         4'h0
Immediate
Now                         300000 ns
```

ADDI R2, #35



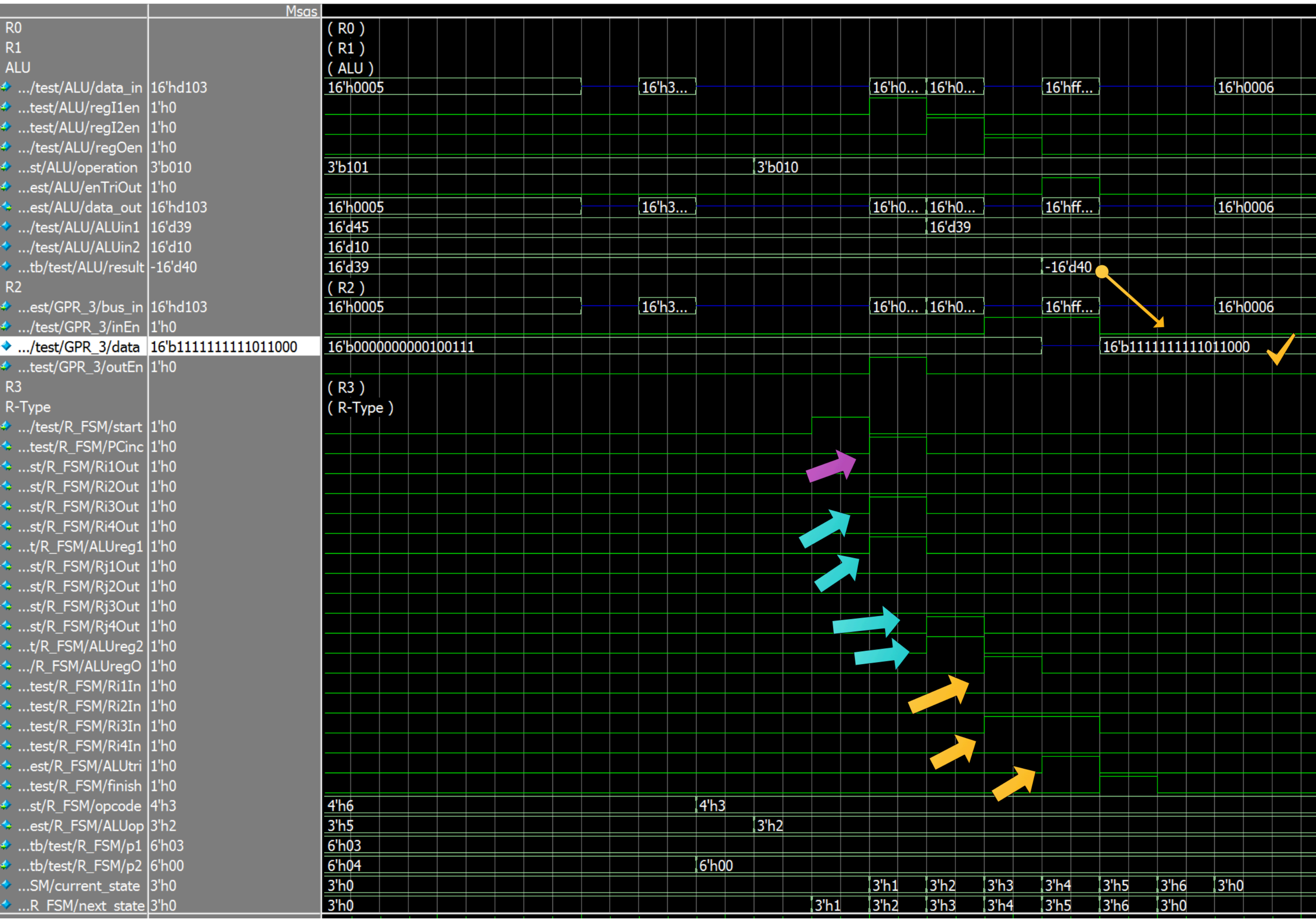
```
Vave - Default
                              Msgs
ALU
  .../test/ALU/data_in        16'h60c4
  ...test/ALU/regI1en         1'h0
  ...test/ALU/regI2en         1'h0
  .../test/ALU/regOen         1'h0
  ...st/ALU/operation         3'h0
  ...est/ALU/enTriOut         1'h0
  ...est/ALU/data_out         16'h60c4
  .../test/ALU/ALUin1         16'd10
  .../test/ALU/ALUin2         16'd35
  ...tb/test/ALU/result       16'd45
R2
  ...est/GPR_3/bus_in         16'h60c4
  .../test/GPR_3/inEn         1'h0
  .../test/GPR_3/data         16'b0000000000101101
  ...test/GPR_3/outEn         1'h0
Immediate
  ...t/Imm_FSM/start          1'h0
  ...test/Imm_FSM/p1          6'h03
  ...test/Imm_FSM/p2          6'h23
  .../Imm_FSM/PCinc           1'h0
  ...Imm_FSM/Ri1Out           1'h0
  ...Imm_FSM/Ri2Out           1'h0
  ...Imm_FSM/Ri3Out           1'h0
  ...Imm_FSM/Ri4Out           1'h0
  ...m_FSM/ALUreg1            1'h0
  ...m_FSM/data_out           16'h60c4
  ...m_FSM/ALUreg2            1'h0
  ...m_FSM/ALUregO            1'h0
  ...t/Imm_FSM/Ri1In          1'h0
  ...t/Imm_FSM/Ri2In          1'h0
  ...t/Imm_FSM/Ri3In          1'h0
  ...t/Imm_FSM/Ri4In          1'h0
  .../Imm_FSM/ALUtri          1'h0
  ...t/Imm_FSM/finish         1'h0
  ...Imm_FSM/ALUop            3'h0
  ...SM/current_state         3'h0
  ..._FSM/next_state          3'h0
  ...m_FSM/data_en            1'h0
```

XOR   R2, R3



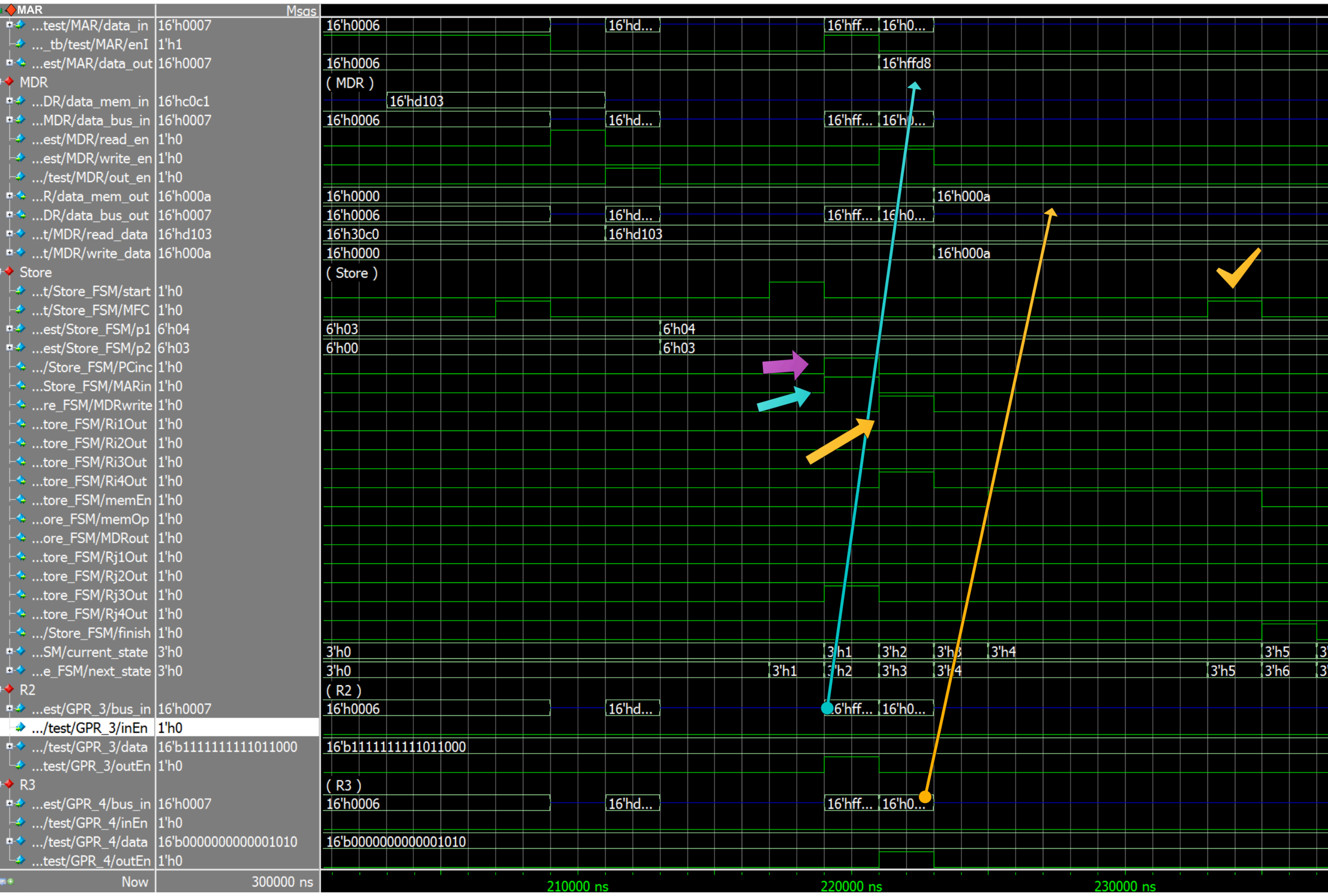
```
ALU
.../test/ALU/data_in      16'h0005
...test/ALU/regI1en       1'h0
...test/ALU/regI2en       1'h0
.../test/ALU/regOen       1'h0
...st/ALU/operation       3'h5
...est/ALU/enTriOut       1'h0
...est/ALU/data_out       16'h0005
.../test/ALU/ALUin1       16'd45
.../test/ALU/ALUin2       16'd10
...tb/test/ALU/result     16'd39
R2
...est/GPR_3/bus_in       16'h0005
.../test/GPR_3/inEn       1'h0
.../test/GPR_3/data       16'b0000000000100111
...test/GPR_3/outEn       1'h0
R3
...est/GPR_4/bus_in       16'h0005
.../test/GPR_4/inEn       1'h0
.../test/GPR_4/data       16'b0000000000001010
...test/GPR_4/outEn       1'h0
R-Type
.../test/R_FSM/start      1'h0
...test/R_FSM/PCinc       1'h0
...st/R_FSM/Ri1Out        1'h0
...st/R_FSM/Ri2Out        1'h0
...st/R_FSM/Ri3Out        1'h0
...st/R_FSM/Ri4Out        1'h0
...t/R_FSM/ALUreg1        1'h0
...st/R_FSM/Rj1Out        1'h0
...st/R_FSM/Rj2Out        1'h0
...st/R_FSM/Rj3Out        1'h0
...st/R_FSM/Rj4Out        1'h0
...t/R_FSM/ALUreg2        1'h0
.../R_FSM/ALUregO         1'h0
...test/R_FSM/Ri1In       1'h0
...test/R_FSM/Ri2In       1'h0
...test/R_FSM/Ri3In       1'h0
...test/R_FSM/Ri4In       1'h0
...est/R_FSM/ALUtri       1'h0
...test/R_FSM/finish      1'h0
...st/R_FSM/opcode        4'h6
...est/R_FSM/ALUop        3'h5
...tb/test/R_FSM/p1       6'h03
...tb/test/R_FSM/p2       6'h04
```

INV_R2



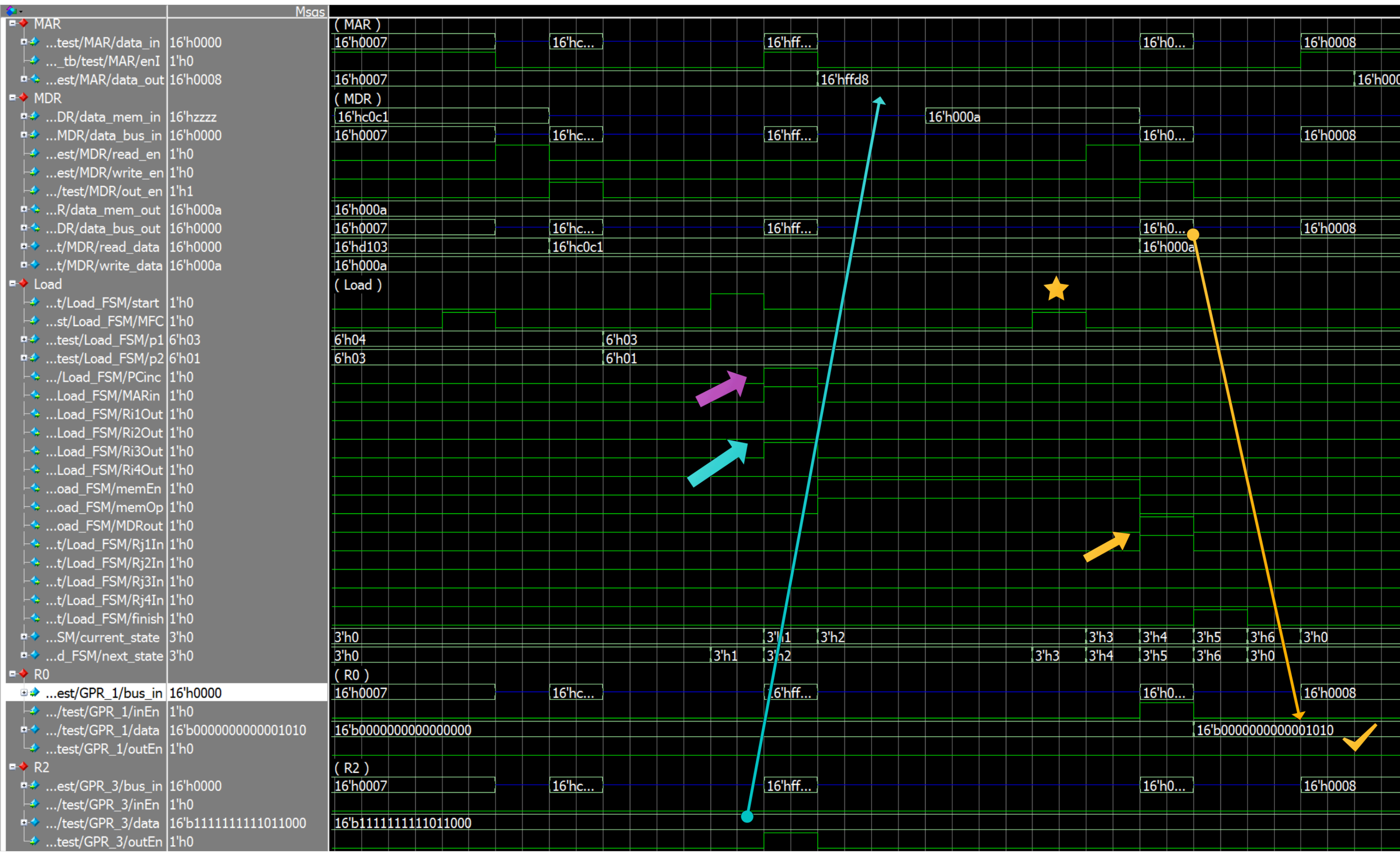
```
                              Msgs
R0                                      ( R0 )
R1                                      ( R1 )
ALU                                     ( ALU )
.../test/ALU/data_in    16'hd103
...test/ALU/regI1en     1'h0
...test/ALU/regI2en     1'h0
.../test/ALU/regOen     1'h0
...st/ALU/operation     3'b010
...est/ALU/enTriOut     1'h0
...est/ALU/data_out     16'hd103
.../test/ALU/ALUin1     16'd39
.../test/ALU/ALUin2     16'd10
...tb/test/ALU/result   -16'd40
R2                                      ( R2 )
...est/GPR_3/bus_in     16'hd103
.../test/GPR_3/inEn     1'h0
.../test/GPR_3/data     16'b1111111111011000
...test/GPR_3/outEn     1'h0
R3                                      ( R3 )
R-Type                                  ( R-Type )
.../test/R_FSM/start    1'h0
...test/R_FSM/PCinc     1'h0
...st/R_FSM/Ri1Out      1'h0
...st/R_FSM/Ri2Out      1'h0
...st/R_FSM/Ri3Out      1'h0
...st/R_FSM/Ri4Out      1'h0
...t/R_FSM/ALUreg1      1'h0
...st/R_FSM/Rj1Out      1'h0
...st/R_FSM/Rj2Out      1'h0
...st/R_FSM/Rj3Out      1'h0
...st/R_FSM/Rj4Out      1'h0
...t/R_FSM/ALUreg2      1'h0
.../R_FSM/ALUregO       1'h0
...test/R_FSM/Ri1In     1'h0
...test/R_FSM/Ri2In     1'h0
...test/R_FSM/Ri3In     1'h0
...test/R_FSM/Ri4In     1'h0
...est/R_FSM/ALUtri     1'h0
...test/R_FSM/finish    1'h0
...st/R_FSM/opcode      4'h3
...est/R_FSM/ALUop      3'h2
...tb/test/R_FSM/p1     6'h03
...tb/test/R_FSM/p2     6'h00
...SM/current_state     3'h0
...R_FSM/next_state     3'h0
```

STORE R3, (R2)



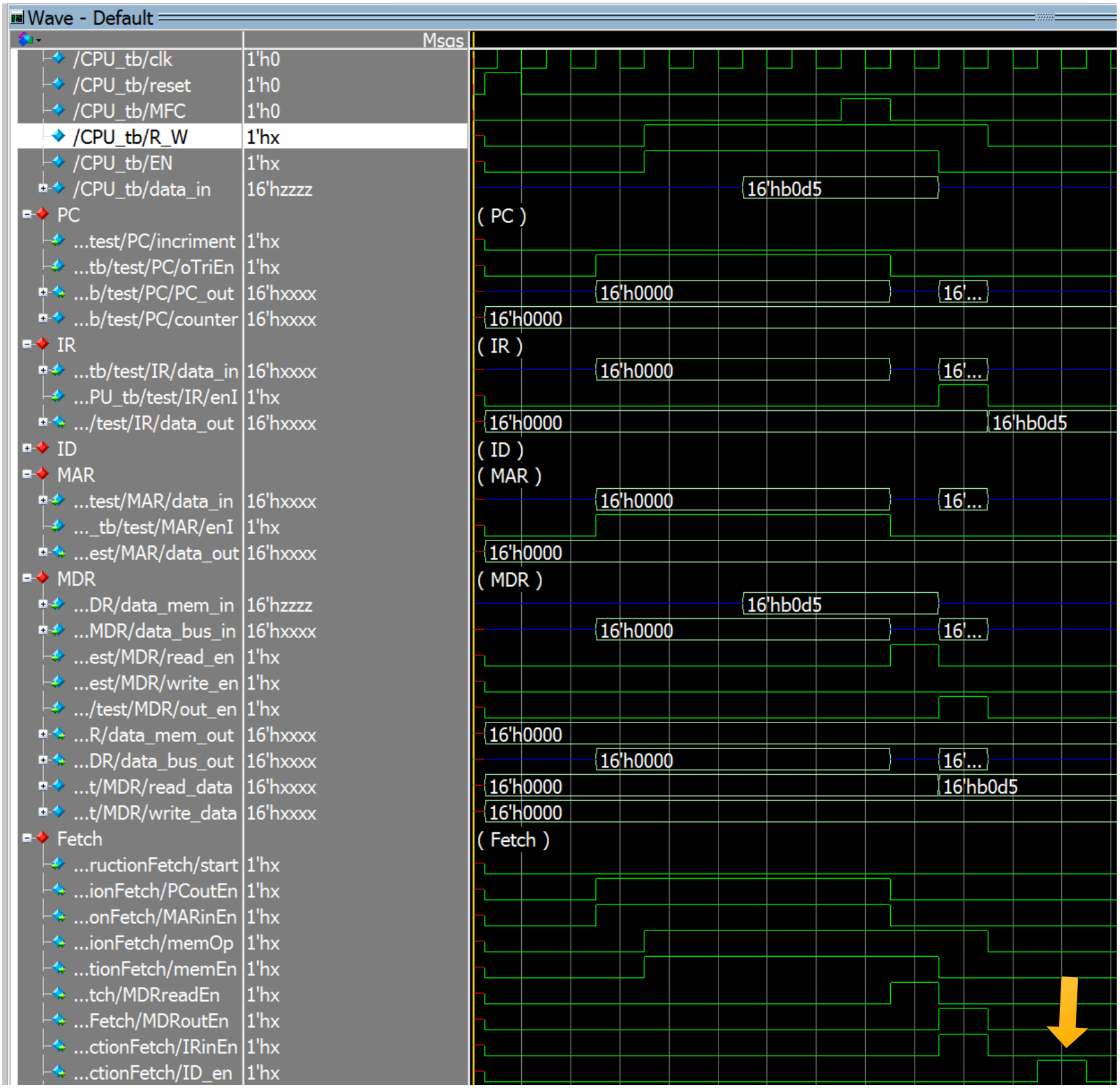
```
MAR                              Msgs
...test/MAR/data_in      16'h0007
..._tb/test/MAR/enI      1'h1
...est/MAR/data_out      16'h0007
MDR
...DR/data_mem_in        16'hc0c1
...MDR/data_bus_in       16'h0007
...est/MDR/read_en       1'h0
...est/MDR/write_en      1'h0
.../test/MDR/out_en      1'h0
...R/data_mem_out        16'h000a
...DR/data_bus_out       16'h0007
...t/MDR/read_data       16'hd103
...t/MDR/write_data      16'h000a
Store
...t/Store_FSM/start     1'h0
...t/Store_FSM/MFC       1'h0
...est/Store_FSM/p1      6'h04
...est/Store_FSM/p2      6'h03
.../Store_FSM/PCinc      1'h0
...Store_FSM/MARin       1'h0
...re_FSM/MDRwrite       1'h0
...tore_FSM/Ri1Out       1'h0
...tore_FSM/Ri2Out       1'h0
...tore_FSM/Ri3Out       1'h0
...tore_FSM/Ri4Out       1'h0
...tore_FSM/memEn        1'h0
...ore_FSM/memOp         1'h0
...ore_FSM/MDRout        1'h0
...tore_FSM/Rj1Out       1'h0
...tore_FSM/Rj2Out       1'h0
...tore_FSM/Rj3Out       1'h0
...tore_FSM/Rj4Out       1'h0
.../Store_FSM/finish     1'h0
...SM/current_state      3'h0
...e_FSM/next_state      3'h0
R2
...est/GPR_3/bus_in      16'h0007
.../test/GPR_3/inEn      1'h0
.../test/GPR_3/data      16'b1111111111011000
...test/GPR_3/outEn      1'h0
R3
...est/GPR_4/bus_in      16'h0007
.../test/GPR_4/inEn      1'h0
.../test/GPR_4/data      16'b0000000000001010
...test/GPR_4/outEn      1'h0
Now                      300000 ns
```

<u>LOAD (R2), R0</u>



	Msgs
MAR	
...test/MAR/data_in	16'h0000
..._tb/test/MAR/enI	1'h0
...est/MAR/data_out	16'h0008
MDR	
...DR/data_mem_in	16'hzzzz
...MDR/data_bus_in	16'h0000
...est/MDR/read_en	1'h0
...est/MDR/write_en	1'h0
.../test/MDR/out_en	1'h1
...R/data_mem_out	16'h000a
...DR/data_bus_out	16'h0000
...t/MDR/read_data	16'h0000
...t/MDR/write_data	16'h000a
Load	
...t/Load_FSM/start	1'h0
...st/Load_FSM/MFC	1'h0
...test/Load_FSM/p1	6'h03
...test/Load_FSM/p2	6'h01
.../Load_FSM/PCinc	1'h0
...Load_FSM/MARin	1'h0
...Load_FSM/Ri1Out	1'h0
...Load_FSM/Ri2Out	1'h0
...Load_FSM/Ri3Out	1'h0
...Load_FSM/Ri4Out	1'h0
...oad_FSM/memEn	1'h0
...oad_FSM/memOp	1'h0
...oad_FSM/MDRout	1'h0
...t/Load_FSM/Rj1In	1'h0
...t/Load_FSM/Rj2In	1'h0
...t/Load_FSM/Rj3In	1'h0
...t/Load_FSM/Rj4In	1'h0
...t/Load_FSM/finish	1'h0
...SM/current_state	3'h0
...d_FSM/next_state	3'h0
R0	
...est/GPR_1/bus_in	16'h0000
.../test/GPR_1/inEn	1'h0
.../test/GPR_1/data	16'b0000000000001010
...test/GPR_1/outEn	1'h0
R2	
...est/GPR_3/bus_in	16'h0000
.../test/GPR_3/inEn	1'h0
.../test/GPR_3/data	16'b1111111111011000
...test/GPR_3/outEn	1'h0

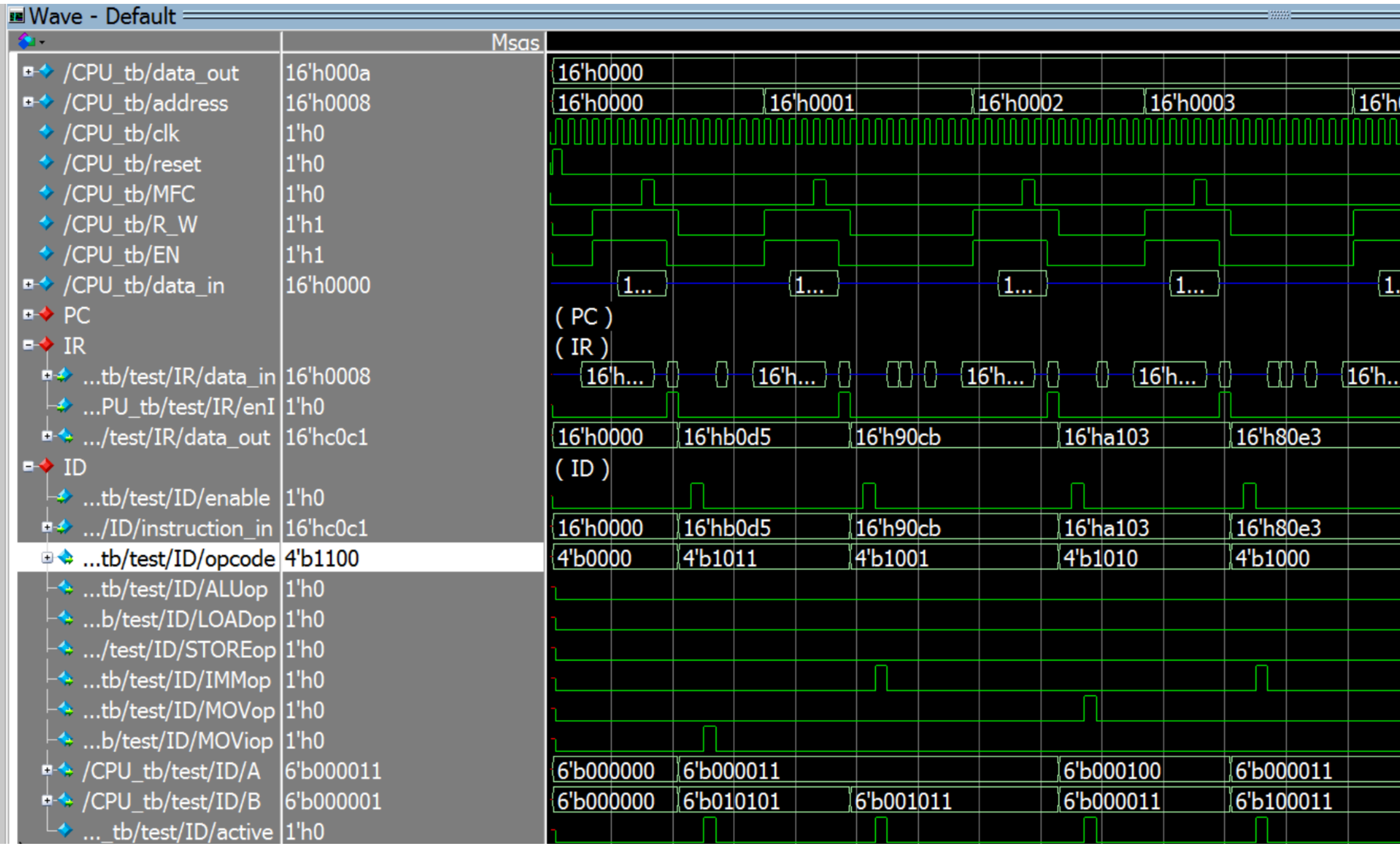
Instruction Fetch



Wave - Default

	Msgs
/CPU_tb/clk	1'h0
/CPU_tb/reset	1'h0
/CPU_tb/MFC	1'h0
/CPU_tb/R_W	1'hx
/CPU_tb/EN	1'hx
/CPU_tb/data_in	16'hzzzz
PC	
...test/PC/incriment	1'hx
...tb/test/PC/oTriEn	1'hx
...b/test/PC/PC_out	16'hxxxx
...b/test/PC/counter	16'hxxxx
IR	
...tb/test/IR/data_in	16'hxxxx
...PU_tb/test/IR/enI	1'hx
.../test/IR/data_out	16'hxxxx
ID	
MAR	
...test/MAR/data_in	16'hxxxx
..._tb/test/MAR/enI	1'hx
...est/MAR/data_out	16'hxxxx
MDR	
...DR/data_mem_in	16'hzzzz
...MDR/data_bus_in	16'hxxxx
...est/MDR/read_en	1'hx
...est/MDR/write_en	1'hx
.../test/MDR/out_en	1'hx
...R/data_mem_out	16'hxxxx
...DR/data_bus_out	16'hxxxx
...t/MDR/read_data	16'hxxxx
...t/MDR/write_data	16'hxxxx
Fetch	
...ructionFetch/start	1'hx
...ionFetch/PCoutEn	1'hx
...onFetch/MARinEn	1'hx
...ionFetch/memOp	1'hx
...tionFetch/memEn	1'hx
...tch/MDRreadEn	1'hx
...Fetch/MDRoutEn	1'hx
...ctionFetch/IRinEn	1'hx
...ctionFetch/ID_en	1'hx

Instruction Decode



	Msgs
/CPU_tb/data_out	16'h000a
/CPU_tb/address	16'h0008
/CPU_tb/clk	1'h0
/CPU_tb/reset	1'h0
/CPU_tb/MFC	1'h0
/CPU_tb/R_W	1'h1
/CPU_tb/EN	1'h1
/CPU_tb/data_in	16'h0000
PC	
IR	
...tb/test/IR/data_in	16'h0008
...PU_tb/test/IR/enI	1'h0
.../test/IR/data_out	16'hc0c1
ID	
...tb/test/ID/enable	1'h0
.../ID/instruction_in	16'hc0c1
...tb/test/ID/opcode	4'b1100
...tb/test/ID/ALUop	1'h0
...b/test/ID/LOADop	1'h0
.../test/ID/STOREop	1'h0
...tb/test/ID/IMMop	1'h0
...tb/test/ID/MOVop	1'h0
...b/test/ID/MOVIop	1'h0
/CPU_tb/test/ID/A	6'b000011
/CPU_tb/test/ID/B	6'b000001
..._tb/test/ID/active	1'h0

Analysis and Conclusion

Instruction Fetch

This screenshot is taken from right at the beginning of execution, showing that the instruction fetch is indeed triggered by the reset signal at the beginning. Once triggered, the FSM waits a clock cycle then sets the PC out-enable and the MAR in-enable to send the instruction address to memory. On the next clock cycle the FSM sets memEn = 1 and memOp = 1. It can be observed on the waveform that this sets the memory signals R_W = 1 and EN = 1 at the top. The memory process in my testbench waits two clock cycles and sends the correct data to the MDR then raises MFC. Once MFC is raised, all the signals relating to the memory operation are dropped and the instruction is latched into the IR from the MDR. Finally, the ID enable signal is raised to decode the instruction (yellow arrow).

Instruction Decode

This screenshot shows the functionality of the ID over the course of the first four instructions. It can be seen that each time a new instruction is fetched (indicated ultimately by the ID enable signal pointed to by the yellow arrow in the last picture). Each time this enable signal goes high, the ID decodes the instruction and sets the corresponding FSM to start. In this screenshot, you see the Movi, Imm, Mov, and Imm signal again get raised, which is what we expect.

Movi R2, #21

This screenshot shows the first instruction in the test program being executed. The purple arrows indicate the PC increment signal and shows its connection with the FSM. The bold arrow shows when the R2-in enable signal gets raised and the thin yellow arrows show the data from p2, the second operand (21) in the instruction, being passed to the bus and into R2. The finish signal initiates the next instruction fetch.

Subi R2, #11

This screenshot shows the second instruction being executed, starting with the purple arrow to indicate PC increment. The light blue arrows indicate where the FSM sends the first operand and second operand into the ALU. First GPR2's value gets transferred to the ALU, then the FSM sends the immediate value 11. After the ALU-out signal is raised it can be seen the correct result of 10 or 16'h000a is then sent into R2.

Mov R3, R2

This waveform of the third instruction is very simple, the purple arrow indicates the PC being incremented and the yellow arrows indicate the transfer of R2 into R3 over the bus. Again, like all other waveforms the finish signal initiates the next fetch.

Addi R2, #35

In this waveform of the fourth instruction, it starts like the rest of the others do with a PC increment. Following the PC increment, the blue arrows indicate the FSM sending the data from R2 into the ALU, then sending the immediate value 35 into the ALU. It's worth noting here the ALUop signal near the bottom of the screenshot is connected to the operation signal in the ALU, controlling the function. The yellow arrows indicate when the correct value of 45 is calculated and transferred into R2.

Xor R2, R3

This waveform shows functionality of the R-type FSM since all our calculations previously have been immediate operations. The blue arrows indicate where the FSM sends R2 into the ALU, then R3. Finally, the yellow arrow indicates when the result is calculated and sent back into R2.

Inv R2

This waveform shows another R-type FSM operation. The purple arrow indicates the PC being incremented, like always. The blue arrows show the FSM latching data into the ALU registers. The FSM sends data into the ALU's second register, however this wont affect our result since the answer is only dependent on the first register. The yellow arrows indicate where the correct inversion is calculated and sent back into R2.

Store R3, (R2)

In our second to last instruction, it begins like the rest with the purple arrow indicating a PC increment. The blue arrow indicates where the FSM sets the MAR to take in data from the bus, and below it we see the FSM set R2-out enable to drive the bus. This tells the memory where to look. The thin blue arrow shows this transfer. The yellow arrows indicate where the value stored in R3 is sent to the MDR. Then the FSM sets EN = 1 and R_W = 1 to indicate a write operation and the MFC signal indicates when the memory is done storing the data.

<u>Load (R2), R0</u>

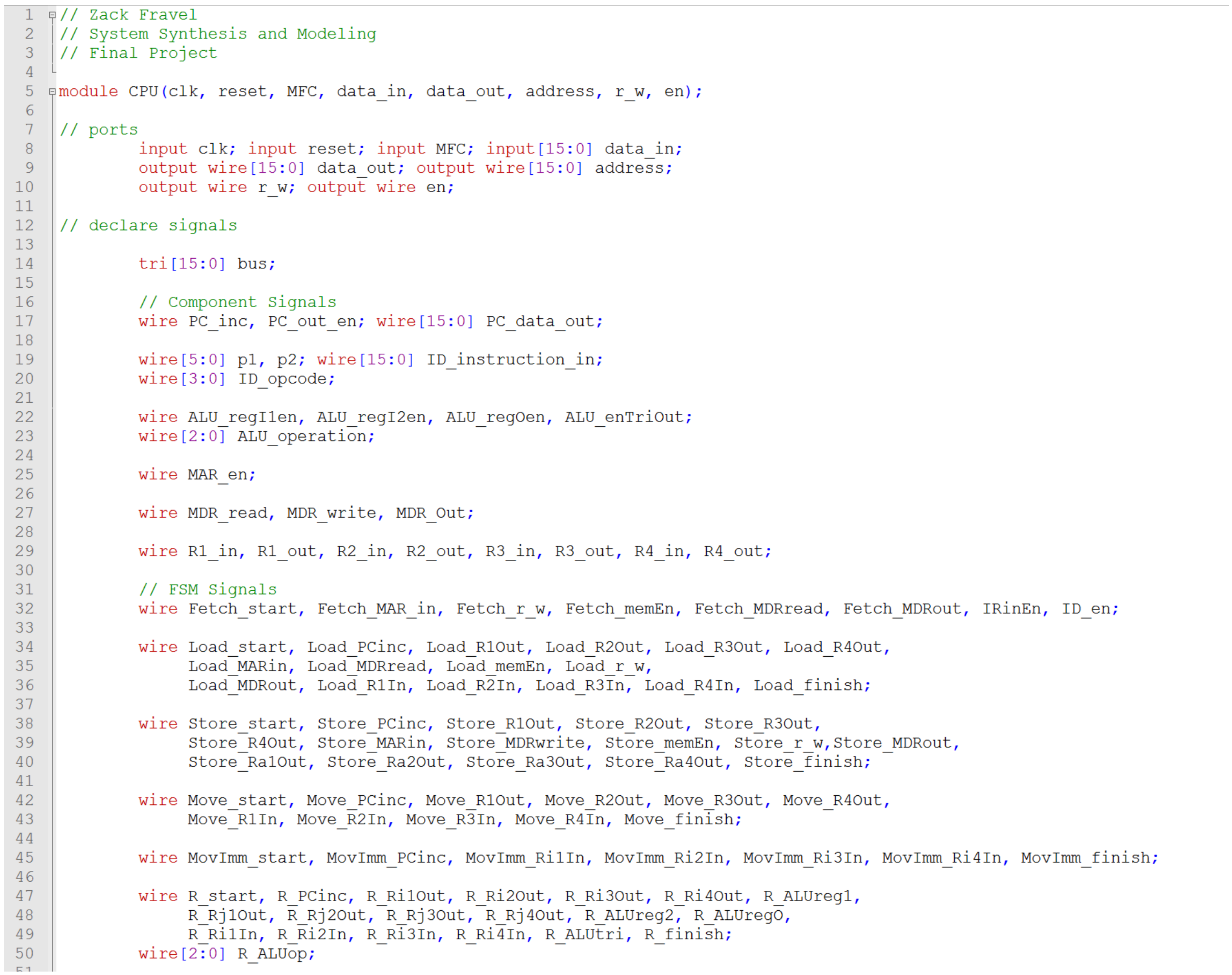
Finally, in our last waveform we have our last PC increment while at the same time sending the contents of R2 to the MAR to tell the memory where to look. The FSM then sets R_W = 1 and EN = 1 to indicate a read operation. The star on the picture indicates where MFC goes high and the FSM tells the MDR to latch its data from the memory. Finally, the FSM makes the MDR send its contents to R0, giving R0 the correct final value of 16’h000a or 10.

This concludes the description and analysis of my design for the RISC microprocessor. The following pages give the pre and post synthesis code for all modules, even the instruction fetch where I show the attempted simulation with it synthesized. Overall I feel I was very successful in executing the task set before me with a fully functioning design.

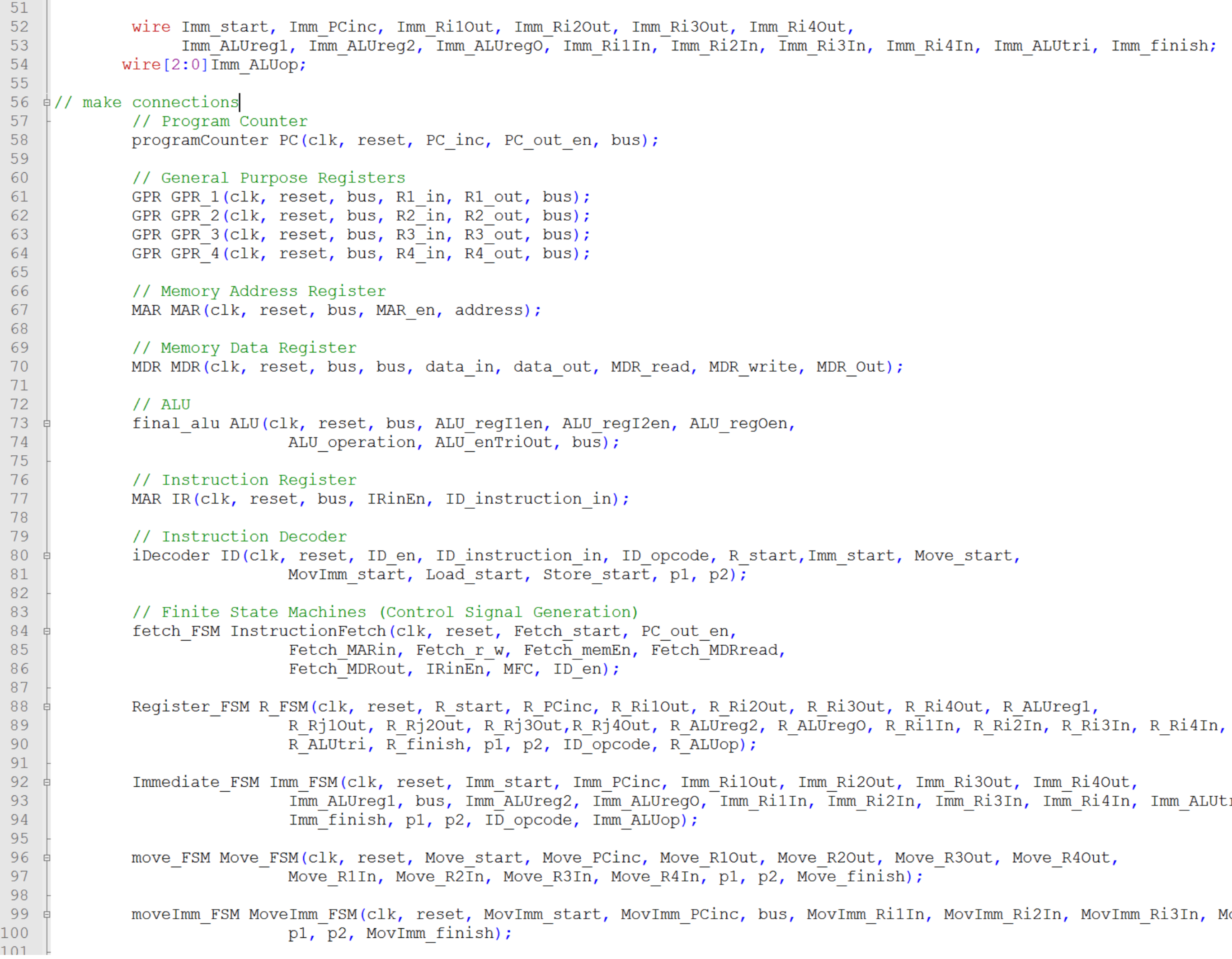
Source Code

Top Level / Testbench (Behavioral Code)

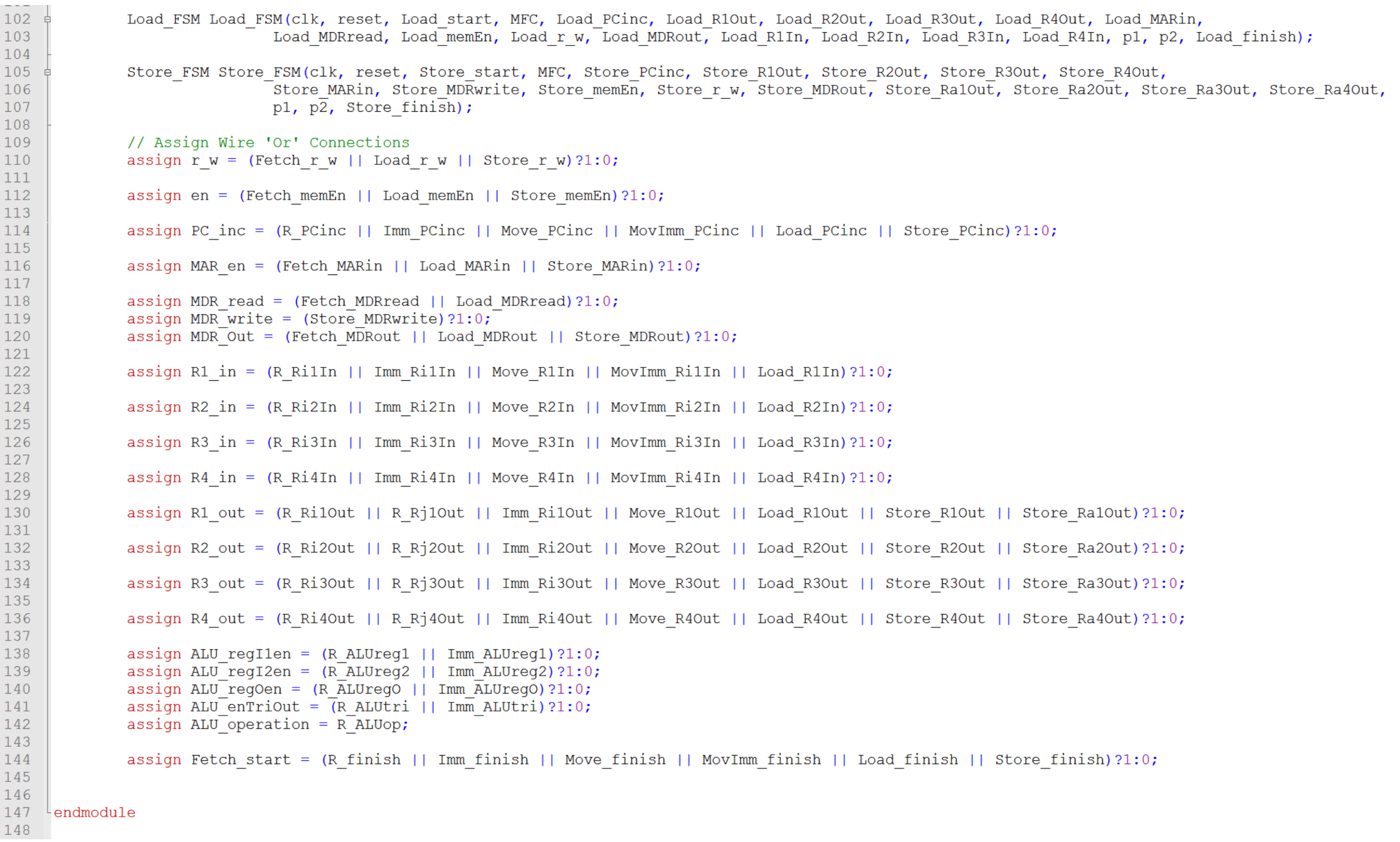
- CPU



```verilog
// Zack Fravel
// System Synthesis and Modeling
// Final Project

module CPU(clk, reset, MFC, data_in, data_out, address, r_w, en);

// ports
        input clk; input reset; input MFC; input[15:0] data_in;
        output wire[15:0] data_out; output wire[15:0] address;
        output wire r_w; output wire en;

// declare signals

        tri[15:0] bus;

        // Component Signals
        wire PC_inc, PC_out_en; wire[15:0] PC_data_out;

        wire[5:0] p1, p2; wire[15:0] ID_instruction_in;
        wire[3:0] ID_opcode;

        wire ALU_regI1en, ALU_regI2en, ALU_regOen, ALU_enTriOut;
        wire[2:0] ALU_operation;

        wire MAR_en;

        wire MDR_read, MDR_write, MDR_Out;

        wire R1_in, R1_out, R2_in, R2_out, R3_in, R3_out, R4_in, R4_out;

        // FSM Signals
        wire Fetch_start, Fetch_MAR_in, Fetch_r_w, Fetch_memEn, Fetch_MDRread, Fetch_MDRout, IRinEn, ID_en;

        wire Load_start, Load_PCinc, Load_R1Out, Load_R2Out, Load_R3Out, Load_R4Out,
             Load_MARin, Load_MDRread, Load_memEn, Load_r_w,
             Load_MDRout, Load_R1In, Load_R2In, Load_R3In, Load_R4In, Load_finish;

        wire Store_start, Store_PCinc, Store_R1Out, Store_R2Out, Store_R3Out,
             Store_R4Out, Store_MARin, Store_MDRwrite, Store_memEn, Store_r_w,Store_MDRout,
             Store_Ra1Out, Store_Ra2Out, Store_Ra3Out, Store_Ra4Out, Store_finish;

        wire Move_start, Move_PCinc, Move_R1Out, Move_R2Out, Move_R3Out, Move_R4Out,
             Move_R1In, Move_R2In, Move_R3In, Move_R4In, Move_finish;

        wire MovImm_start, MovImm_PCinc, MovImm_Ri1In, MovImm_Ri2In, MovImm_Ri3In, MovImm_Ri4In, MovImm_finish;

        wire R_start, R_PCinc, R_Ri1Out, R_Ri2Out, R_Ri3Out, R_Ri4Out, R_ALUreg1,
             R_Rj1Out, R_Rj2Out, R_Rj3Out, R_Rj4Out, R_ALUreg2, R_ALUregO,
             R_Ri1In, R_Ri2In, R_Ri3In, R_Ri4In, R_ALUtri, R_finish;
        wire[2:0] R_ALUop;
```

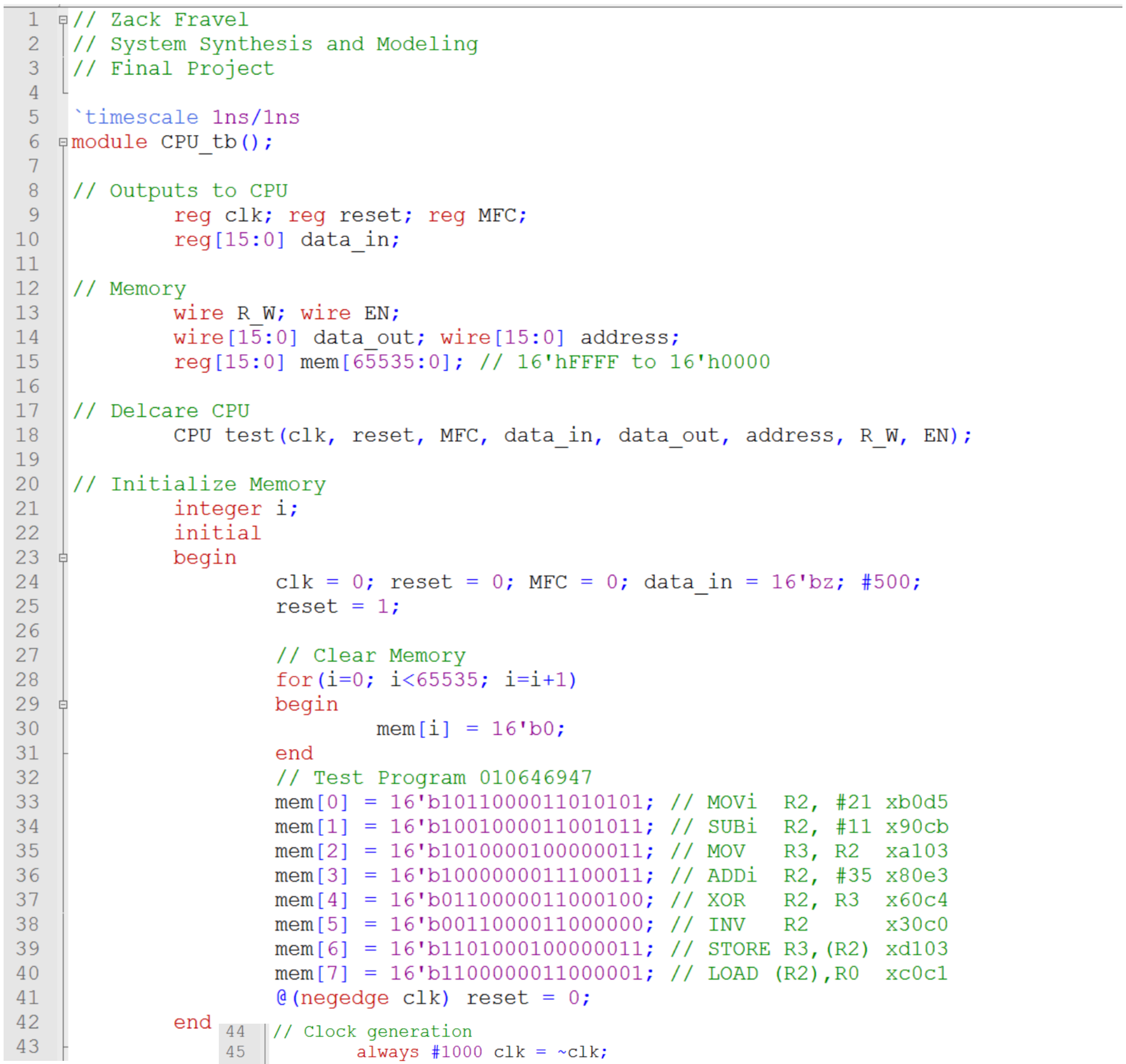


```
        wire Imm_start, Imm_PCinc, Imm_Ri1Out, Imm_Ri2Out, Imm_Ri3Out, Imm_Ri4Out,
             Imm_ALUreg1, Imm_ALUreg2, Imm_ALUregO, Imm_Ri1In, Imm_Ri2In, Imm_Ri3In, Imm_Ri4In, Imm_ALUtri, Imm_finish;
       wire[2:0]Imm_ALUop;

// make connections
        // Program Counter
        programCounter PC(clk, reset, PC_inc, PC_out_en, bus);

        // General Purpose Registers
        GPR GPR_1(clk, reset, bus, R1_in, R1_out, bus);
        GPR GPR_2(clk, reset, bus, R2_in, R2_out, bus);
        GPR GPR_3(clk, reset, bus, R3_in, R3_out, bus);
        GPR GPR_4(clk, reset, bus, R4_in, R4_out, bus);

        // Memory Address Register
        MAR MAR(clk, reset, bus, MAR_en, address);

        // Memory Data Register
        MDR MDR(clk, reset, bus, bus, data_in, data_out, MDR_read, MDR_write, MDR_Out);

        // ALU
        final_alu ALU(clk, reset, bus, ALU_regI1en, ALU_regI2en, ALU_regOen,
                        ALU_operation, ALU_enTriOut, bus);

        // Instruction Register
        MAR IR(clk, reset, bus, IRinEn, ID_instruction_in);

        // Instruction Decoder
        iDecoder ID(clk, reset, ID_en, ID_instruction_in, ID_opcode, R_start,Imm_start, Move_start,
                        MovImm_start, Load_start, Store_start, p1, p2);

        // Finite State Machines (Control Signal Generation)
        fetch_FSM InstructionFetch(clk, reset, Fetch_start, PC_out_en,
                        Fetch_MARin, Fetch_r_w, Fetch_memEn, Fetch_MDRread,
                        Fetch_MDRout, IRinEn, MFC, ID_en);

        Register_FSM R_FSM(clk, reset, R_start, R_PCinc, R_Ri1Out, R_Ri2Out, R_Ri3Out, R_Ri4Out, R_ALUreg1,
                        R_Rj1Out, R_Rj2Out, R_Rj3Out,R_Rj4Out, R_ALUreg2, R_ALUregO, R_Ri1In, R_Ri2In, R_Ri3In, R_Ri4In,
                        R_ALUtri, R_finish, p1, p2, ID_opcode, R_ALUop);

        Immediate_FSM Imm_FSM(clk, reset, Imm_start, Imm_PCinc, Imm_Ri1Out, Imm_Ri2Out, Imm_Ri3Out, Imm_Ri4Out,
                        Imm_ALUreg1, bus, Imm_ALUreg2, Imm_ALUregO, Imm_Ri1In, Imm_Ri2In, Imm_Ri3In, Imm_Ri4In, Imm_ALUtri,
                        Imm_finish, p1, p2, ID_opcode, Imm_ALUop);

        move_FSM Move_FSM(clk, reset, Move_start, Move_PCinc, Move_R1Out, Move_R2Out, Move_R3Out, Move_R4Out,
                        Move_R1In, Move_R2In, Move_R3In, Move_R4In, p1, p2, Move_finish);

        moveImm_FSM MoveImm_FSM(clk, reset, MovImm_start, MovImm_PCinc, bus, MovImm_Ri1In, MovImm_Ri2In, MovImm_Ri3In, MovImm_Ri4In,
                        p1, p2, MovImm_finish);
```

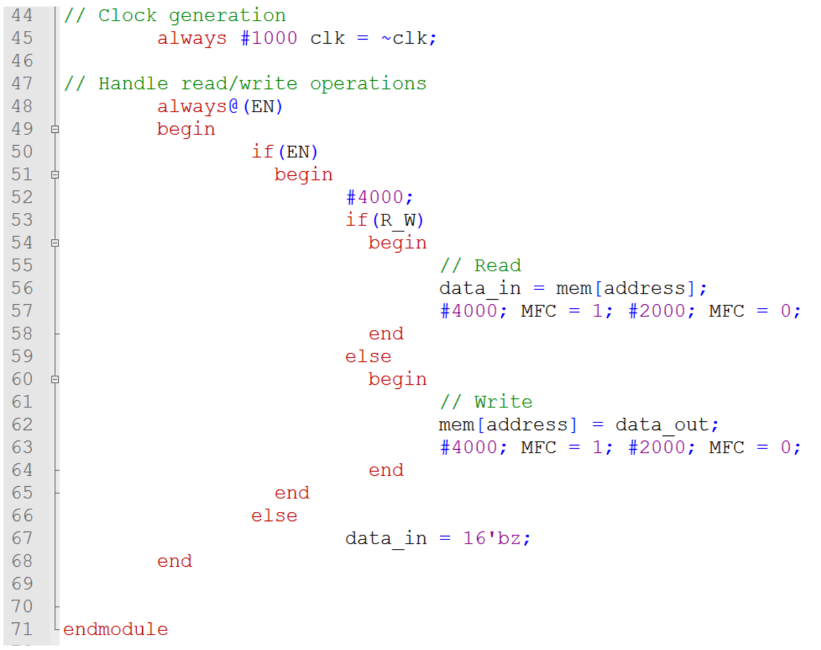


```
        Load_FSM Load_FSM(clk, reset, Load_start, MFC, Load_PCinc, Load_R1Out, Load_R2Out, Load_R3Out, Load_R4Out, Load_MARin,
                        Load_MDRread, Load_memEn, Load_r_w, Load_MDRout, Load_R1In, Load_R2In, Load_R3In, Load_R4In, p1, p2, Load_finish);

        Store_FSM Store_FSM(clk, reset, Store_start, MFC, Store_PCinc, Store_R1Out, Store_R2Out, Store_R3Out, Store_R4Out,
                        Store_MARin, Store_MDRwrite, Store_memEn, Store_r_w, Store_MDRout, Store_Ra1Out, Store_Ra2Out, Store_Ra3Out, Store_Ra4Out,
                        p1, p2, Store_finish);

        // Assign Wire 'Or' Connections
        assign r_w = (Fetch_r_w || Load_r_w || Store_r_w)?1:0;

        assign en = (Fetch_memEn || Load_memEn || Store_memEn)?1:0;

        assign PC_inc = (R_PCinc || Imm_PCinc || Move_PCinc || MovImm_PCinc || Load_PCinc || Store_PCinc)?1:0;

        assign MAR_en = (Fetch_MARin || Load_MARin || Store_MARin)?1:0;

        assign MDR_read = (Fetch_MDRread || Load_MDRread)?1:0;
        assign MDR_write = (Store_MDRwrite)?1:0;
        assign MDR_Out = (Fetch_MDRout || Load_MDRout || Store_MDRout)?1:0;

        assign R1_in = (R_Ri1In || Imm_Ri1In || Move_R1In || MovImm_Ri1In || Load_R1In)?1:0;

        assign R2_in = (R_Ri2In || Imm_Ri2In || Move_R2In || MovImm_Ri2In || Load_R2In)?1:0;

        assign R3_in = (R_Ri3In || Imm_Ri3In || Move_R3In || MovImm_Ri3In || Load_R3In)?1:0;

        assign R4_in = (R_Ri4In || Imm_Ri4In || Move_R4In || MovImm_Ri4In || Load_R4In)?1:0;

        assign R1_out = (R_Ri1Out || R_Rj1Out || Imm_Ri1Out || Move_R1Out || Load_R1Out || Store_R1Out || Store_Ra1Out)?1:0;

        assign R2_out = (R_Ri2Out || R_Rj2Out || Imm_Ri2Out || Move_R2Out || Load_R2Out || Store_R2Out || Store_Ra2Out)?1:0;

        assign R3_out = (R_Ri3Out || R_Rj3Out || Imm_Ri3Out || Move_R3Out || Load_R3Out || Store_R3Out || Store_Ra3Out)?1:0;

        assign R4_out = (R_Ri4Out || R_Rj4Out || Imm_Ri4Out || Move_R4Out || Load_R4Out || Store_R4Out || Store_Ra4Out)?1:0;

        assign ALU_regI1en = (R_ALUreg1 || Imm_ALUreg1)?1:0;
        assign ALU_regI2en = (R_ALUreg2 || Imm_ALUreg2)?1:0;
        assign ALU_regOen = (R_ALUregO || Imm_ALUregO)?1:0;
        assign ALU_enTriOut = (R_ALUtri || Imm_ALUtri)?1:0;
        assign ALU_operation = R_ALUop;

        assign Fetch_start = (R_finish || Imm_finish || Move_finish || MovImm_finish || Load_finish || Store_finish)?1:0;


endmodule
```

- Testbench



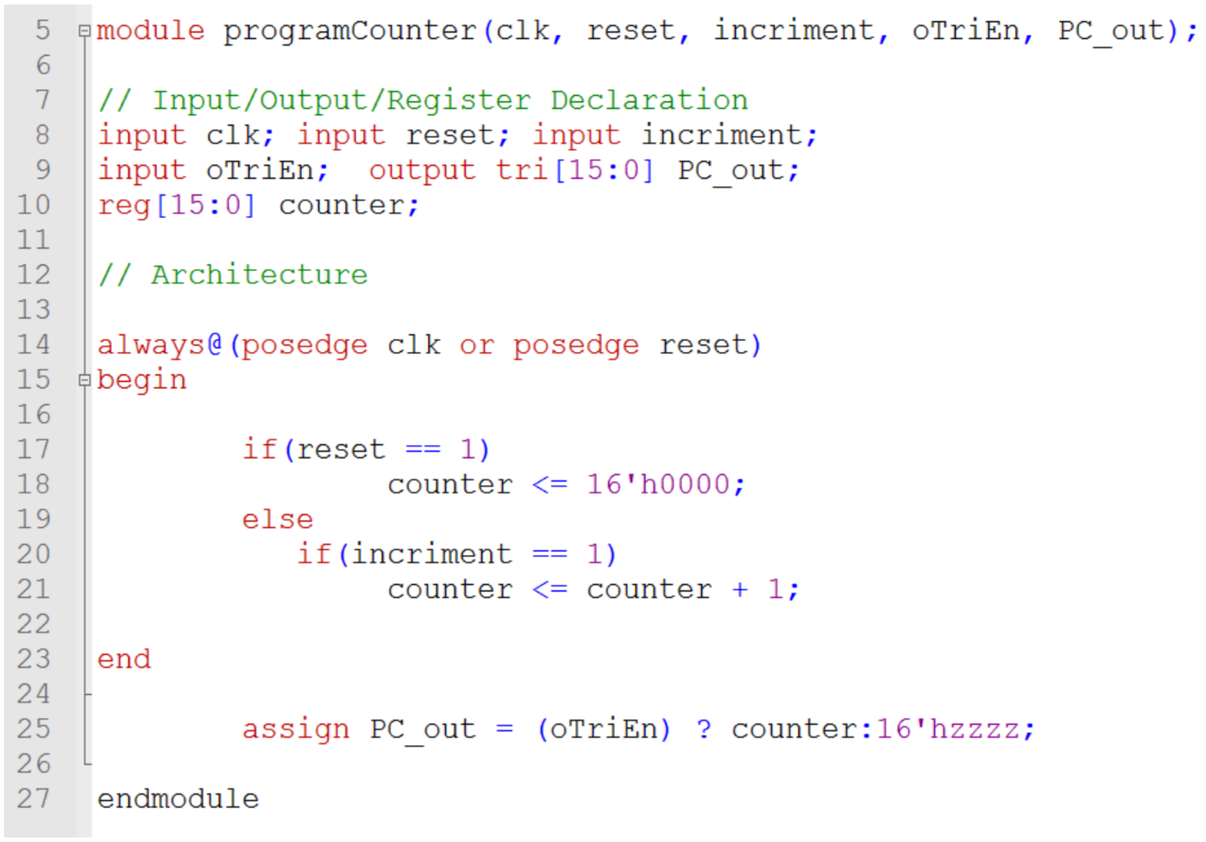
```
// Zack Fravel
// System Synthesis and Modeling
// Final Project

`timescale 1ns/1ns
module CPU_tb();

// Outputs to CPU
        reg clk; reg reset; reg MFC;
        reg[15:0] data_in;

// Memory
        wire R_W; wire EN;
        wire[15:0] data_out; wire[15:0] address;
        reg[15:0] mem[65535:0]; // 16'hFFFF to 16'h0000

// Delcare CPU
        CPU test(clk, reset, MFC, data_in, data_out, address, R_W, EN);

// Initialize Memory
        integer i;
        initial
        begin
                clk = 0; reset = 0; MFC = 0; data_in = 16'bz; #500;
                reset = 1;

                // Clear Memory
                for(i=0; i<65535; i=i+1)
                begin
                        mem[i] = 16'b0;
                end
                // Test Program 010646947
                mem[0] = 16'b1011000011010101; // MOVi  R2, #21 xb0d5
                mem[1] = 16'b1001000011001011; // SUBi  R2, #11 x90cb
                mem[2] = 16'b1010000100000011; // MOV   R3, R2  xa103
                mem[3] = 16'b1000000011100011; // ADDi  R2, #35 x80e3
                mem[4] = 16'b0110000011000100; // XOR   R2, R3  x60c4
                mem[5] = 16'b0011000011000000; // INV   R2      x30c0
                mem[6] = 16'b1101000100000011; // STORE R3,(R2) xd103
                mem[7] = 16'b1100000011000001; // LOAD (R2),R0  xc0c1
                @(negedge clk) reset = 0;
        end

```



```
// Clock generation
        always #1000 clk = ~clk;

// Handle read/write operations
        always@(EN)
        begin
                if(EN)
                  begin
                        #4000;
                        if(R_W)
                          begin
                                // Read
                                data_in = mem[address];
                                #4000; MFC = 1; #2000; MFC = 0;
                          end
                        else
                          begin
                                // Write
                                mem[address] = data_out;
                                #4000; MFC = 1; #2000; MFC = 0;
                          end
                  end
                else
                        data_in = 16'bz;
        end


endmodule
```

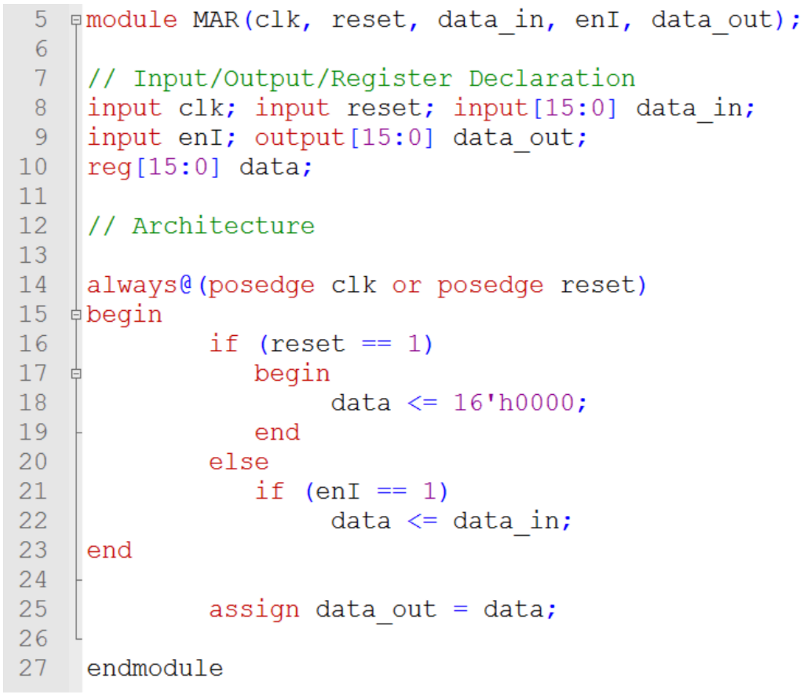
Modules (Pre-Synthesis Behavioral Code)

- Program Counter



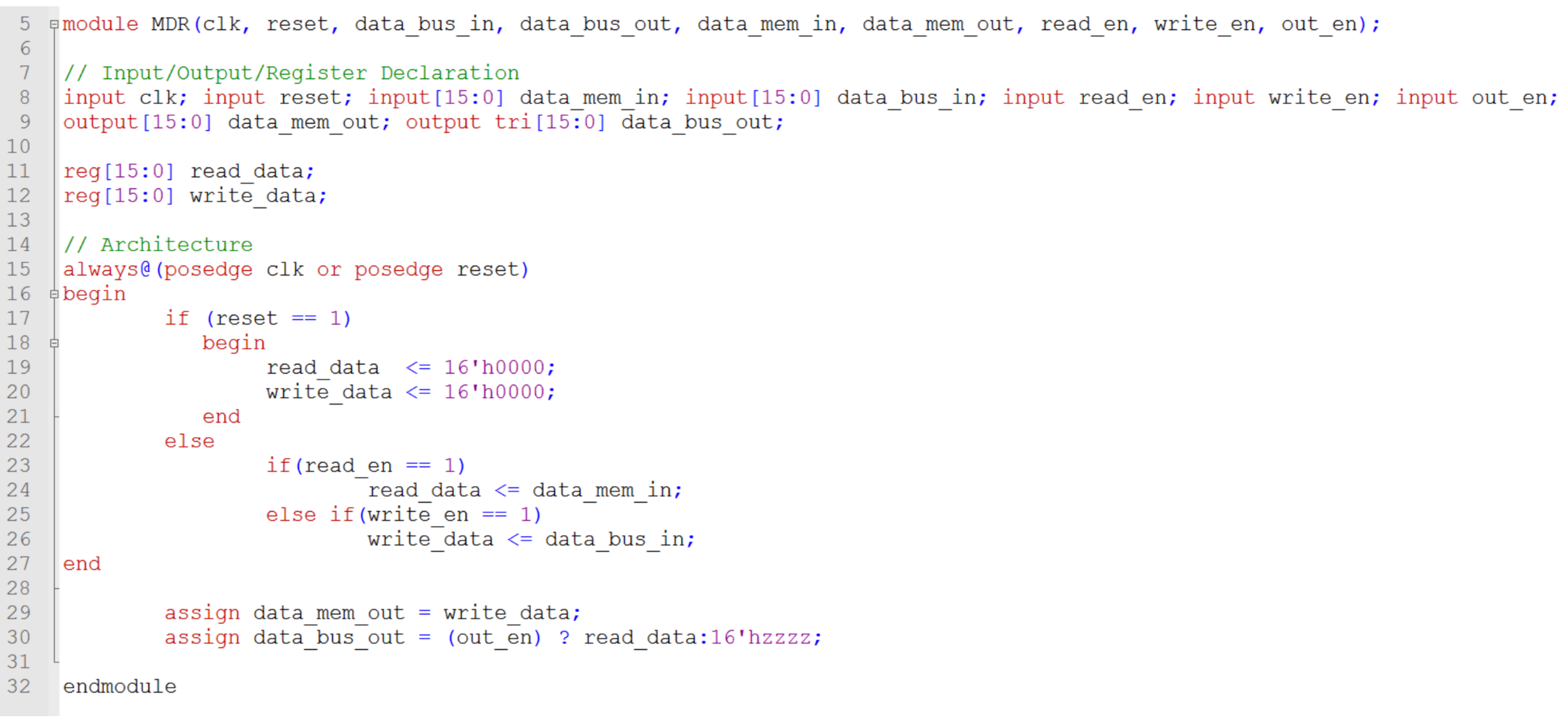
```verilog
module programCounter(clk, reset, incriment, oTriEn, PC_out);

// Input/Output/Register Declaration
input clk; input reset; input incriment;
input oTriEn;  output tri[15:0] PC_out;
reg[15:0] counter;

// Architecture

always@(posedge clk or posedge reset)
begin

        if(reset == 1)
                counter <= 16'h0000;
        else
           if(incriment == 1)
                counter <= counter + 1;

end

        assign PC_out = (oTriEn) ? counter:16'hzzzz;

endmodule
```

- MAR / IR



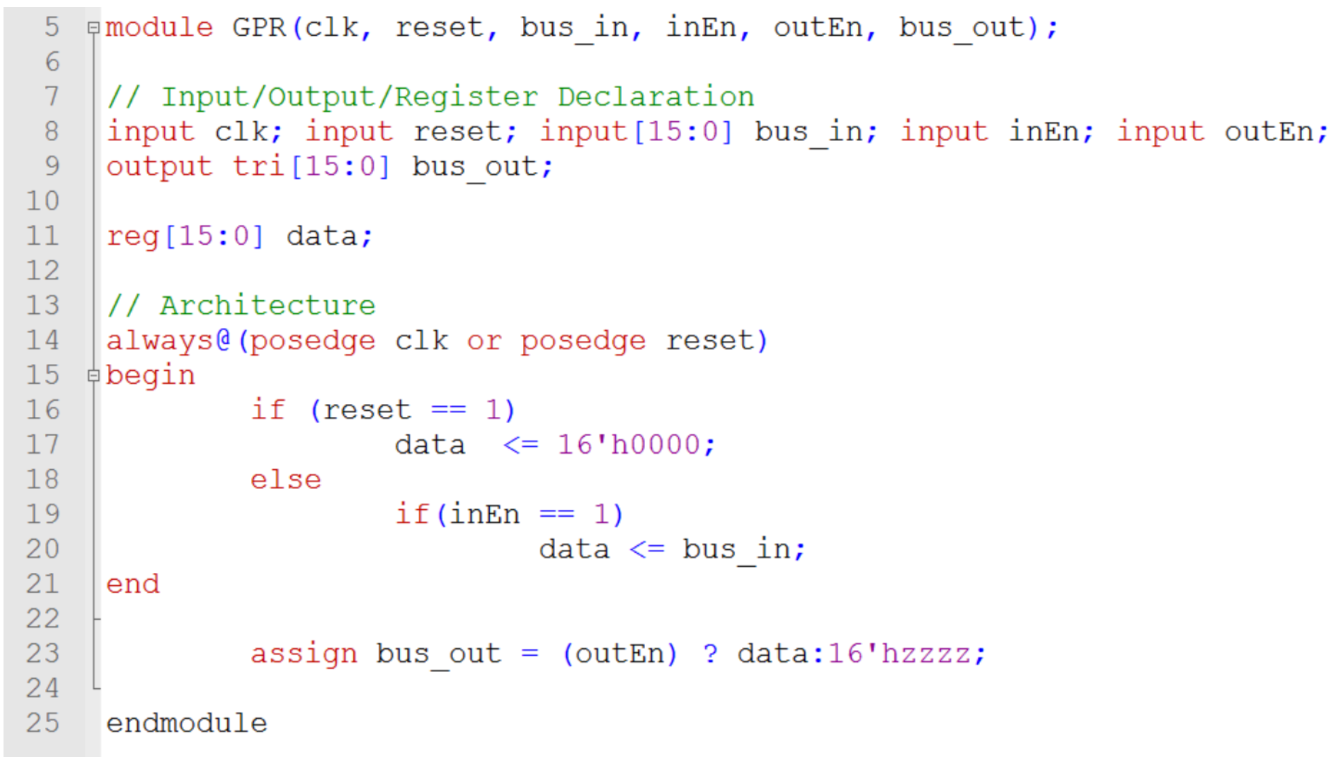
```verilog
module MAR(clk, reset, data_in, enI, data_out);

// Input/Output/Register Declaration
input clk; input reset; input[15:0] data_in;
input enI; output[15:0] data_out;
reg[15:0] data;

// Architecture

always@(posedge clk or posedge reset)
begin
        if (reset == 1)
           begin
                data <= 16'h0000;
           end
        else
           if (enI == 1)
                data <= data_in;
end

        assign data_out = data;

endmodule
```

- MDR



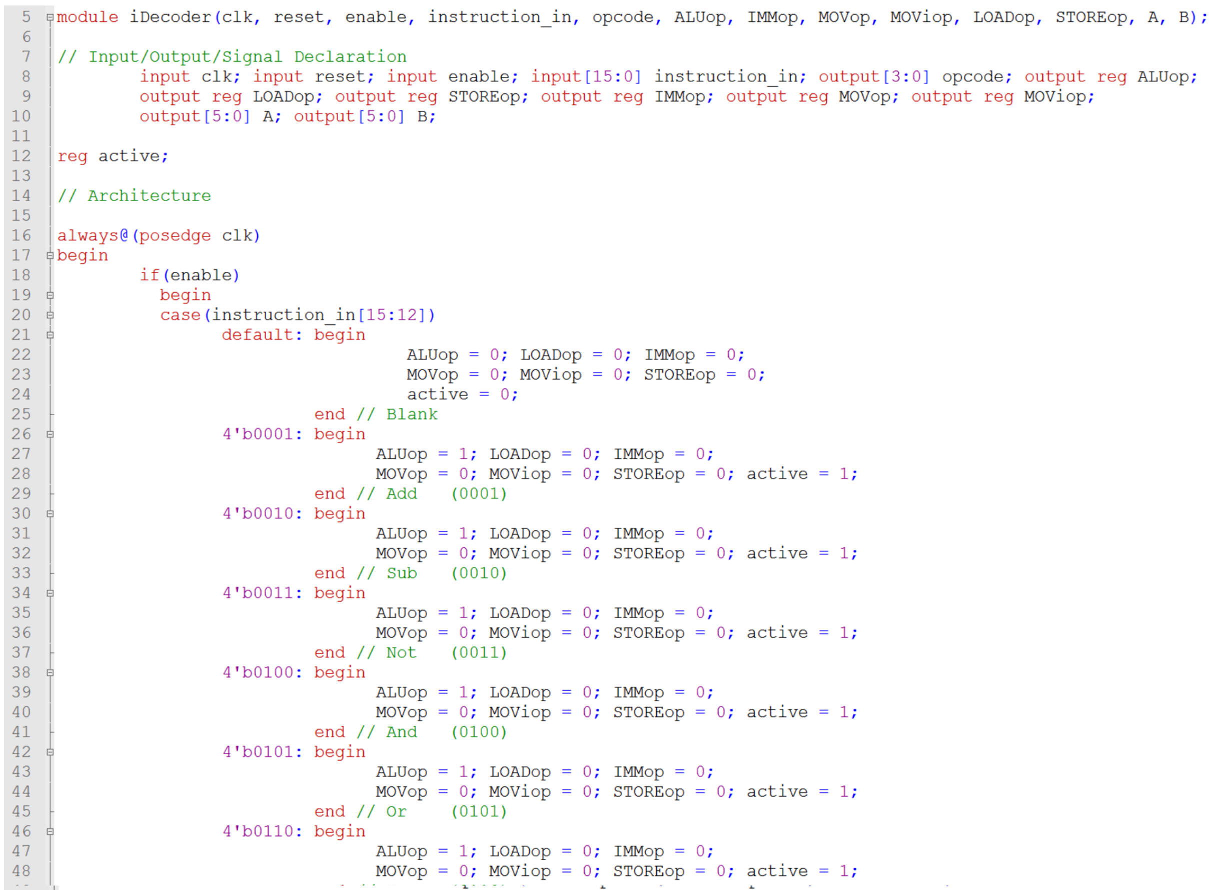
```verilog
module MDR(clk, reset, data_bus_in, data_bus_out, data_mem_in, data_mem_out, read_en, write_en, out_en);

// Input/Output/Register Declaration
input clk; input reset; input[15:0] data_mem_in; input[15:0] data_bus_in; input read_en; input write_en; input out_en;
output[15:0] data_mem_out; output tri[15:0] data_bus_out;

reg[15:0] read_data;
reg[15:0] write_data;

// Architecture
always@(posedge clk or posedge reset)
begin
        if (reset == 1)
           begin
                read_data  <= 16'h0000;
                write_data <= 16'h0000;
           end
        else
                if(read_en == 1)
                        read_data <= data_mem_in;
                else if(write_en == 1)
                        write_data <= data_bus_in;
end

        assign data_mem_out = write_data;
        assign data_bus_out = (out_en) ? read_data:16'hzzzz;

endmodule
```

- GPR

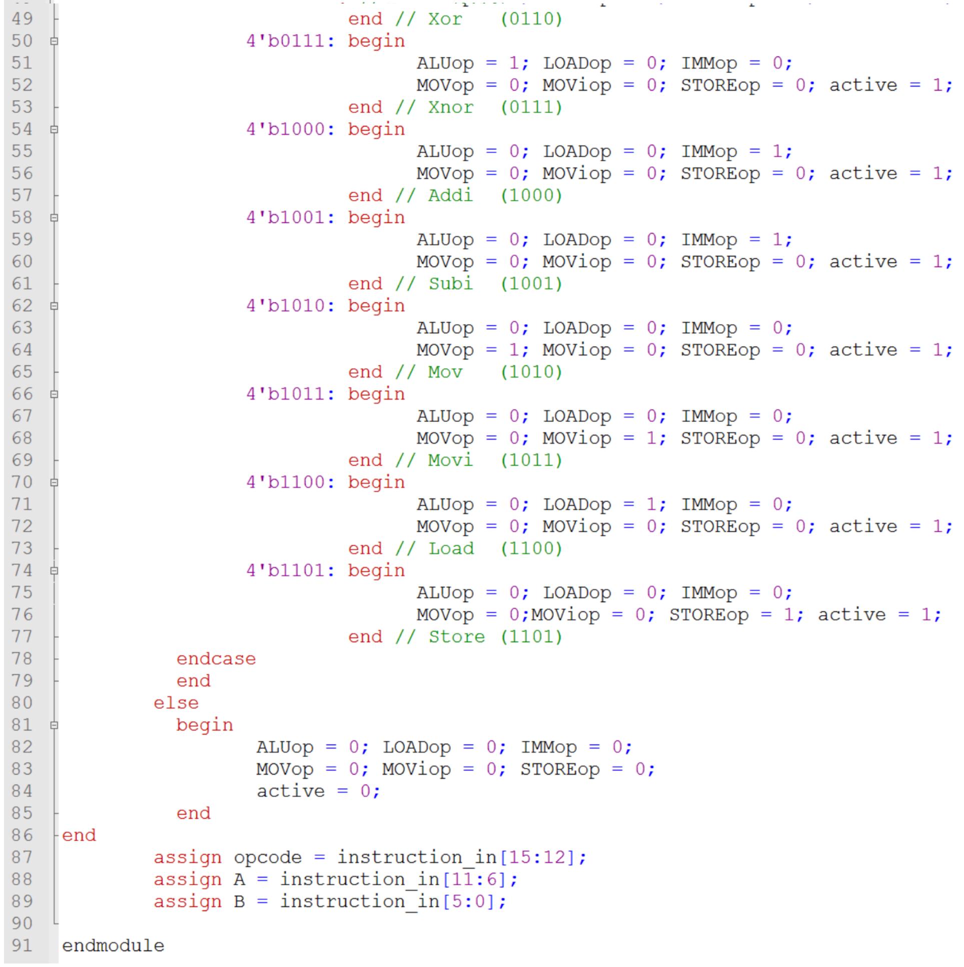


```verilog
module GPR(clk, reset, bus_in, inEn, outEn, bus_out);

// Input/Output/Register Declaration
input clk; input reset; input[15:0] bus_in; input inEn; input outEn;
output tri[15:0] bus_out;

reg[15:0] data;

// Architecture
always@(posedge clk or posedge reset)
begin
        if (reset == 1)
                data  <= 16'h0000;
        else
                if(inEn == 1)
                        data <= bus_in;
end

        assign bus_out = (outEn) ? data:16'hzzzz;

endmodule
```

- ID

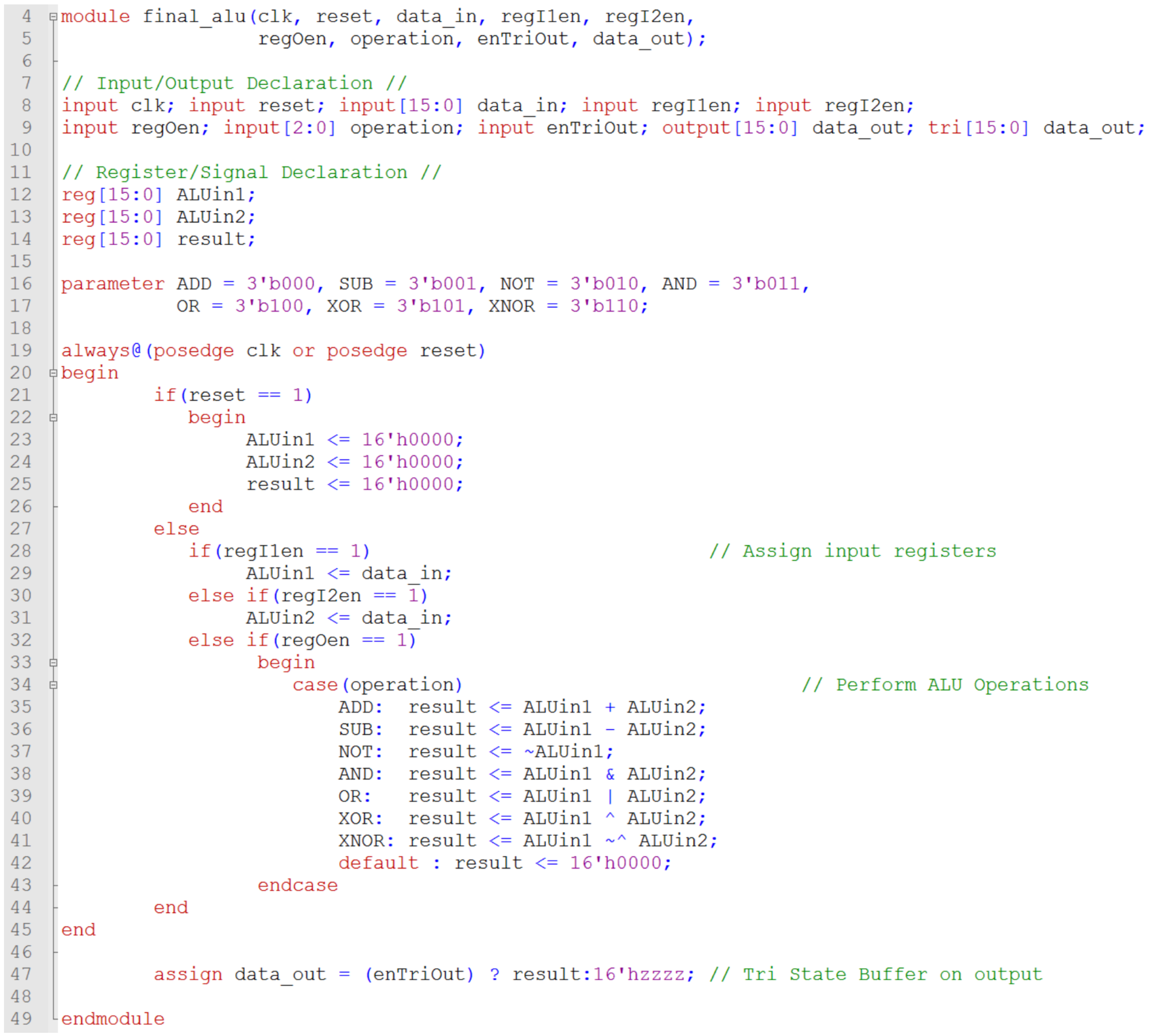


```
module iDecoder(clk, reset, enable, instruction_in, opcode, ALUop, IMMop, MOVop, MOViop, LOADop, STOREop, A, B);

// Input/Output/Signal Declaration
        input clk; input reset; input enable; input[15:0] instruction_in; output[3:0] opcode; output reg ALUop;
        output reg LOADop; output reg STOREop; output reg IMMop; output reg MOVop; output reg MOViop;
        output[5:0] A; output[5:0] B;

reg active;

// Architecture

always@(posedge clk)
begin
        if(enable)
          begin
          case(instruction_in[15:12])
                default: begin
                                  ALUop = 0; LOADop = 0; IMMop = 0;
                                  MOVop = 0; MOViop = 0; STOREop = 0;
                                  active = 0;
                         end // Blank
                4'b0001: begin
                               ALUop = 1; LOADop = 0; IMMop = 0;
                               MOVop = 0; MOViop = 0; STOREop = 0; active = 1;
                         end // Add   (0001)
                4'b0010: begin
                               ALUop = 1; LOADop = 0; IMMop = 0;
                               MOVop = 0; MOViop = 0; STOREop = 0; active = 1;
                         end // Sub   (0010)
                4'b0011: begin
                               ALUop = 1; LOADop = 0; IMMop = 0;
                               MOVop = 0; MOViop = 0; STOREop = 0; active = 1;
                         end // Not   (0011)
                4'b0100: begin
                               ALUop = 1; LOADop = 0; IMMop = 0;
                               MOVop = 0; MOViop = 0; STOREop = 0; active = 1;
                         end // And   (0100)
                4'b0101: begin
                               ALUop = 1; LOADop = 0; IMMop = 0;
                               MOVop = 0; MOViop = 0; STOREop = 0; active = 1;
                         end // Or    (0101)
                4'b0110: begin
                               ALUop = 1; LOADop = 0; IMMop = 0;
                               MOVop = 0; MOViop = 0; STOREop = 0; active = 1;
```



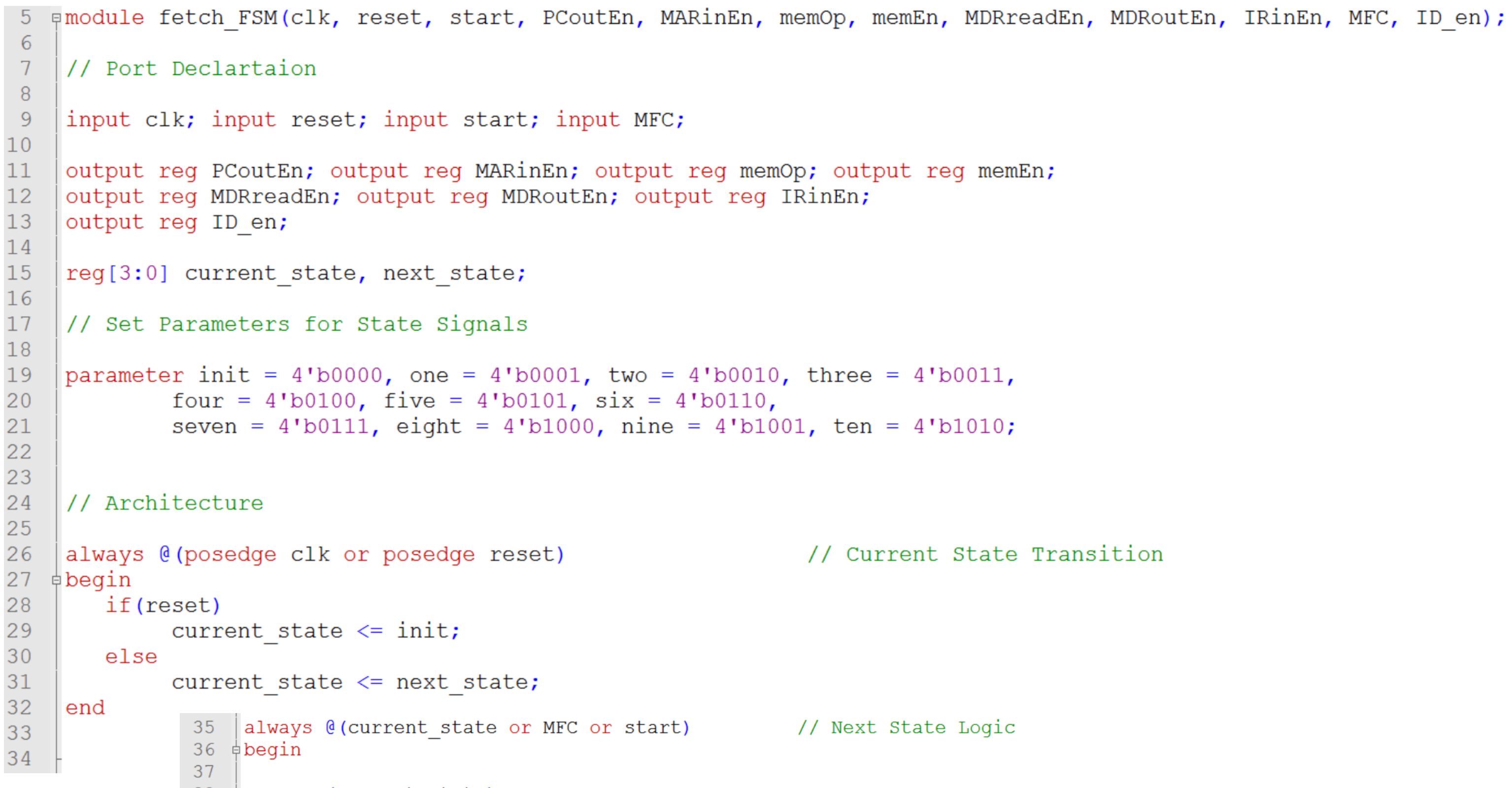
```
                         end // Xor   (0110)
                4'b0111: begin
                               ALUop = 1; LOADop = 0; IMMop = 0;
                               MOVop = 0; MOViop = 0; STOREop = 0; active = 1;
                         end // Xnor  (0111)
                4'b1000: begin
                               ALUop = 0; LOADop = 0; IMMop = 1;
                               MOVop = 0; MOViop = 0; STOREop = 0; active = 1;
                         end // Addi  (1000)
                4'b1001: begin
                               ALUop = 0; LOADop = 0; IMMop = 1;
                               MOVop = 0; MOViop = 0; STOREop = 0; active = 1;
                         end // Subi  (1001)
                4'b1010: begin
                               ALUop = 0; LOADop = 0; IMMop = 0;
                               MOVop = 1; MOViop = 0; STOREop = 0; active = 1;
                         end // Mov   (1010)
                4'b1011: begin
                               ALUop = 0; LOADop = 0; IMMop = 0;
                               MOVop = 0; MOViop = 1; STOREop = 0; active = 1;
                         end // Movi  (1011)
                4'b1100: begin
                               ALUop = 0; LOADop = 1; IMMop = 0;
                               MOVop = 0; MOViop = 0; STOREop = 0; active = 1;
                         end // Load  (1100)
                4'b1101: begin
                               ALUop = 0; LOADop = 0; IMMop = 0;
                               MOVop = 0;MOViop = 0; STOREop = 1; active = 1;
                         end // Store (1101)
          endcase
          end
        else
          begin
                ALUop = 0; LOADop = 0; IMMop = 0;
                MOVop = 0; MOViop = 0; STOREop = 0;
                active = 0;
          end
end
        assign opcode = instruction_in[15:12];
        assign A = instruction_in[11:6];
        assign B = instruction_in[5:0];

endmodule
```

- ALU

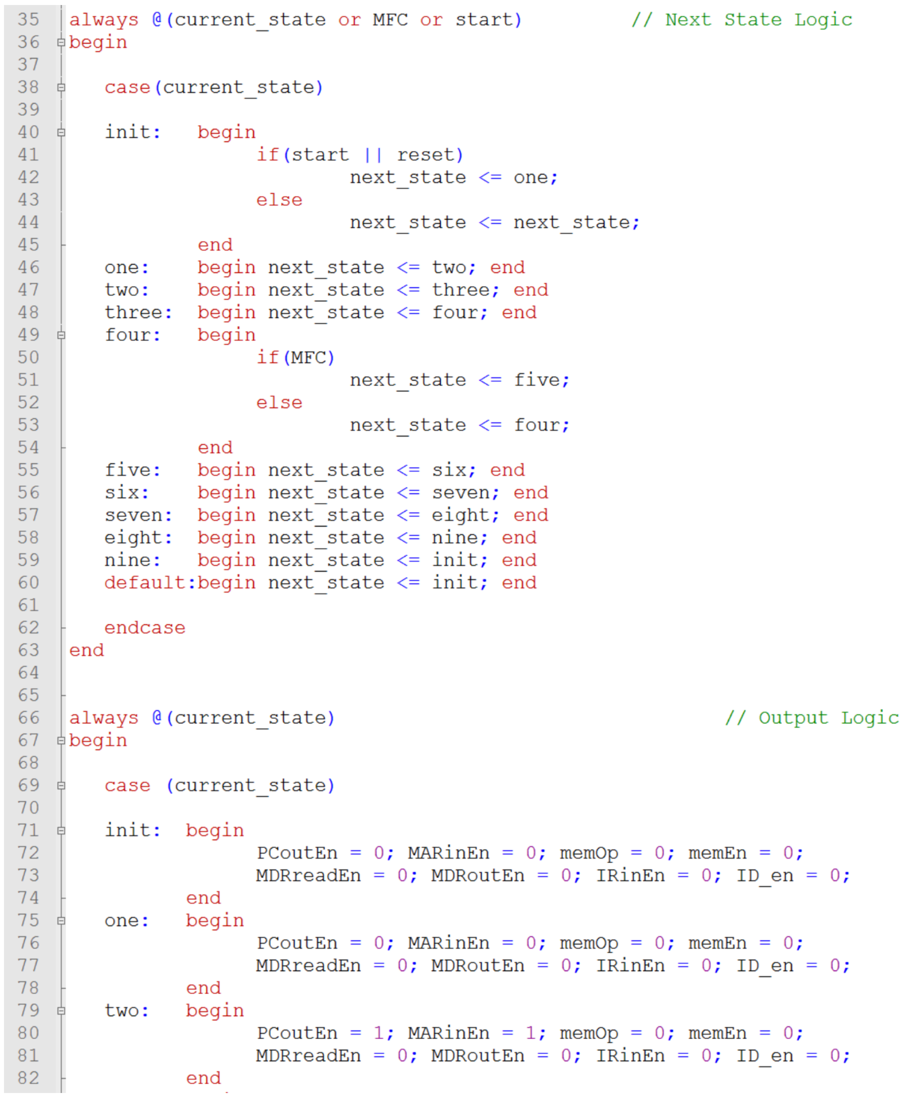


```verilog
module final_alu(clk, reset, data_in, regI1en, regI2en,
                 regOen, operation, enTriOut, data_out);

// Input/Output Declaration //
input clk; input reset; input[15:0] data_in; input regI1en; input regI2en;
input regOen; input[2:0] operation; input enTriOut; output[15:0] data_out; tri[15:0] data_out;

// Register/Signal Declaration //
reg[15:0] ALUin1;
reg[15:0] ALUin2;
reg[15:0] result;

parameter ADD = 3'b000, SUB = 3'b001, NOT = 3'b010, AND = 3'b011,
          OR = 3'b100, XOR = 3'b101, XNOR = 3'b110;

always@(posedge clk or posedge reset)
begin
        if(reset == 1)
           begin
                ALUin1 <= 16'h0000;
                ALUin2 <= 16'h0000;
                result <= 16'h0000;
           end
        else
           if(regI1en == 1)                             // Assign input registers
                ALUin1 <= data_in;
           else if(regI2en == 1)
                ALUin2 <= data_in;
           else if(regOen == 1)
                 begin
                    case(operation)                            // Perform ALU Operations
                        ADD:  result <= ALUin1 + ALUin2;
                        SUB:  result <= ALUin1 - ALUin2;
                        NOT:  result <= ~ALUin1;
                        AND:  result <= ALUin1 & ALUin2;
                        OR:   result <= ALUin1 | ALUin2;
                        XOR:  result <= ALUin1 ^ ALUin2;
                        XNOR: result <= ALUin1 ~^ ALUin2;
                        default : result <= 16'h0000;
                 endcase
        end
end

        assign data_out = (enTriOut) ? result:16'hzzzz; // Tri State Buffer on output

endmodule
```

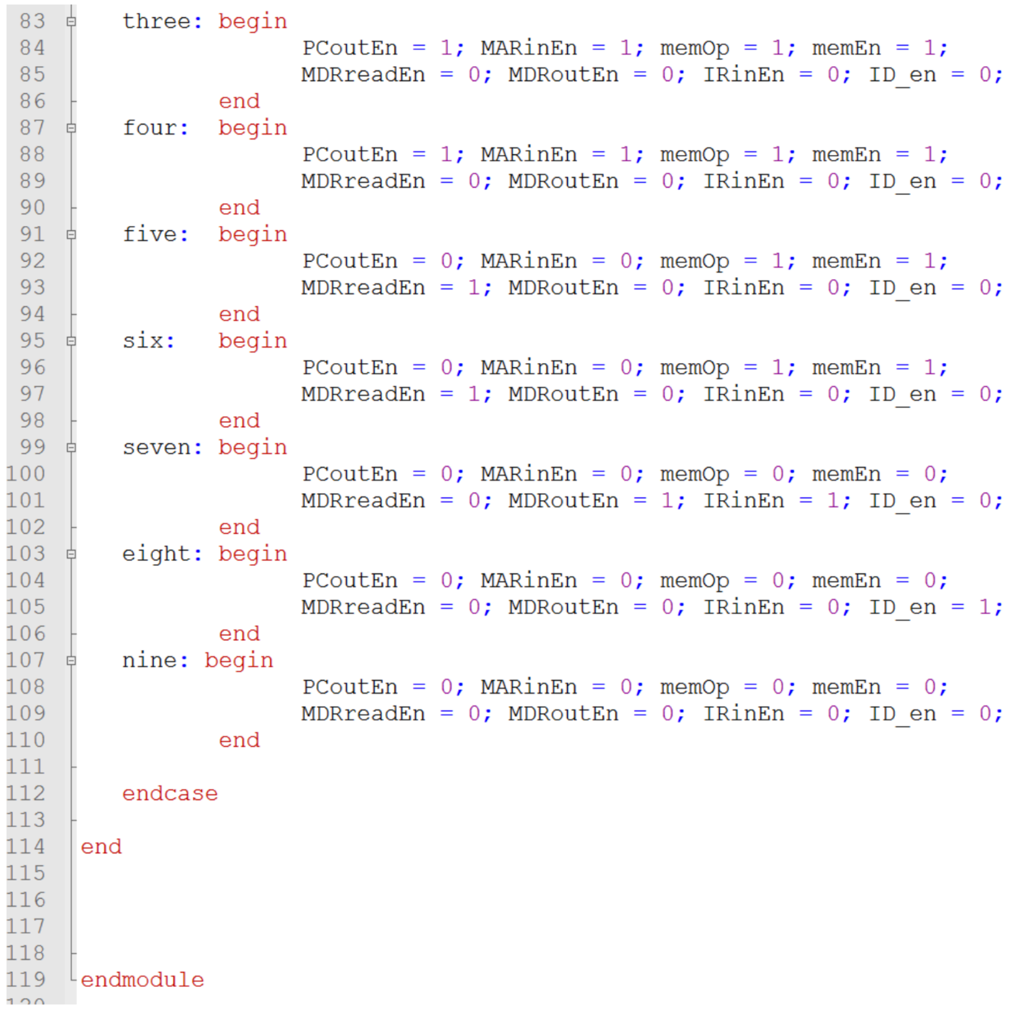
- Instruction Fetch FSM



```
module fetch_FSM(clk, reset, start, PCoutEn, MARinEn, memOp, memEn, MDRreadEn, MDRoutEn, IRinEn, MFC, ID_en);

// Port Declartaion

input clk; input reset; input start; input MFC;

output reg PCoutEn; output reg MARinEn; output reg memOp; output reg memEn;
output reg MDRreadEn; output reg MDRoutEn; output reg IRinEn;
output reg ID_en;

reg[3:0] current_state, next_state;

// Set Parameters for State Signals

parameter init = 4'b0000, one = 4'b0001, two = 4'b0010, three = 4'b0011,
        four = 4'b0100, five = 4'b0101, six = 4'b0110,
        seven = 4'b0111, eight = 4'b1000, nine = 4'b1001, ten = 4'b1010;


// Architecture

always @(posedge clk or posedge reset)                  // Current State Transition
begin
    if(reset)
        current_state <= init;
    else
        current_state <= next_state;
end
```

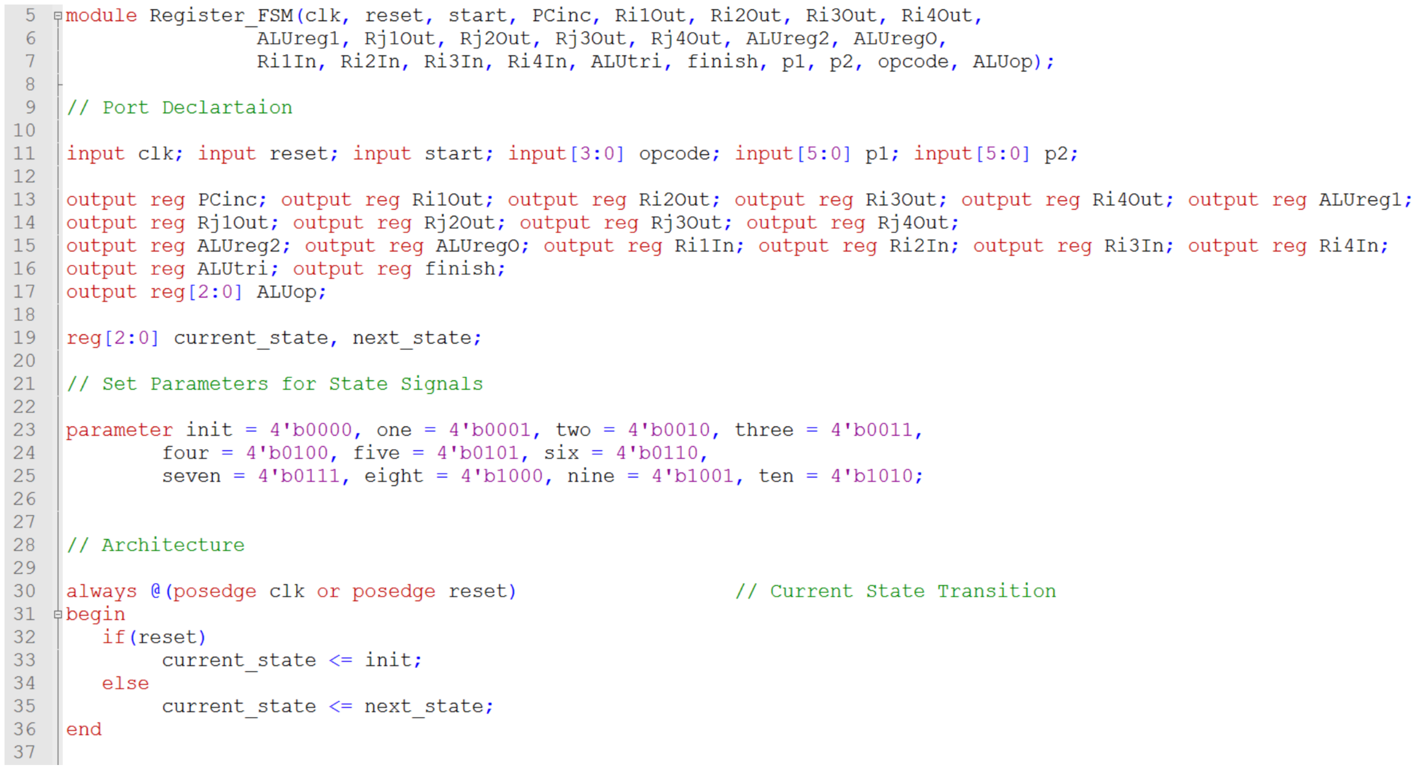


```
always @(current_state or MFC or start)         // Next State Logic
begin

    case(current_state)

    init:   begin
                if(start || reset)
                        next_state <= one;
                else
                        next_state <= next_state;
            end
    one:    begin next_state <= two; end
    two:    begin next_state <= three; end
    three:  begin next_state <= four; end
    four:   begin
                if(MFC)
                        next_state <= five;
                else
                        next_state <= four;
            end
    five:   begin next_state <= six; end
    six:    begin next_state <= seven; end
    seven:  begin next_state <= eight; end
    eight:  begin next_state <= nine; end
    nine:   begin next_state <= init; end
    default:begin next_state <= init; end

    endcase
end


always @(current_state)                                 // Output Logic
begin

    case (current_state)

    init:  begin
                PCoutEn = 0; MARinEn = 0; memOp = 0; memEn = 0;
                MDRreadEn = 0; MDRoutEn = 0; IRinEn = 0; ID_en = 0;
           end
    one:   begin
                PCoutEn = 0; MARinEn = 0; memOp = 0; memEn = 0;
                MDRreadEn = 0; MDRoutEn = 0; IRinEn = 0; ID_en = 0;
           end
    two:   begin
                PCoutEn = 1; MARinEn = 1; memOp = 0; memEn = 0;
                MDRreadEn = 0; MDRoutEn = 0; IRinEn = 0; ID_en = 0;
           end
```



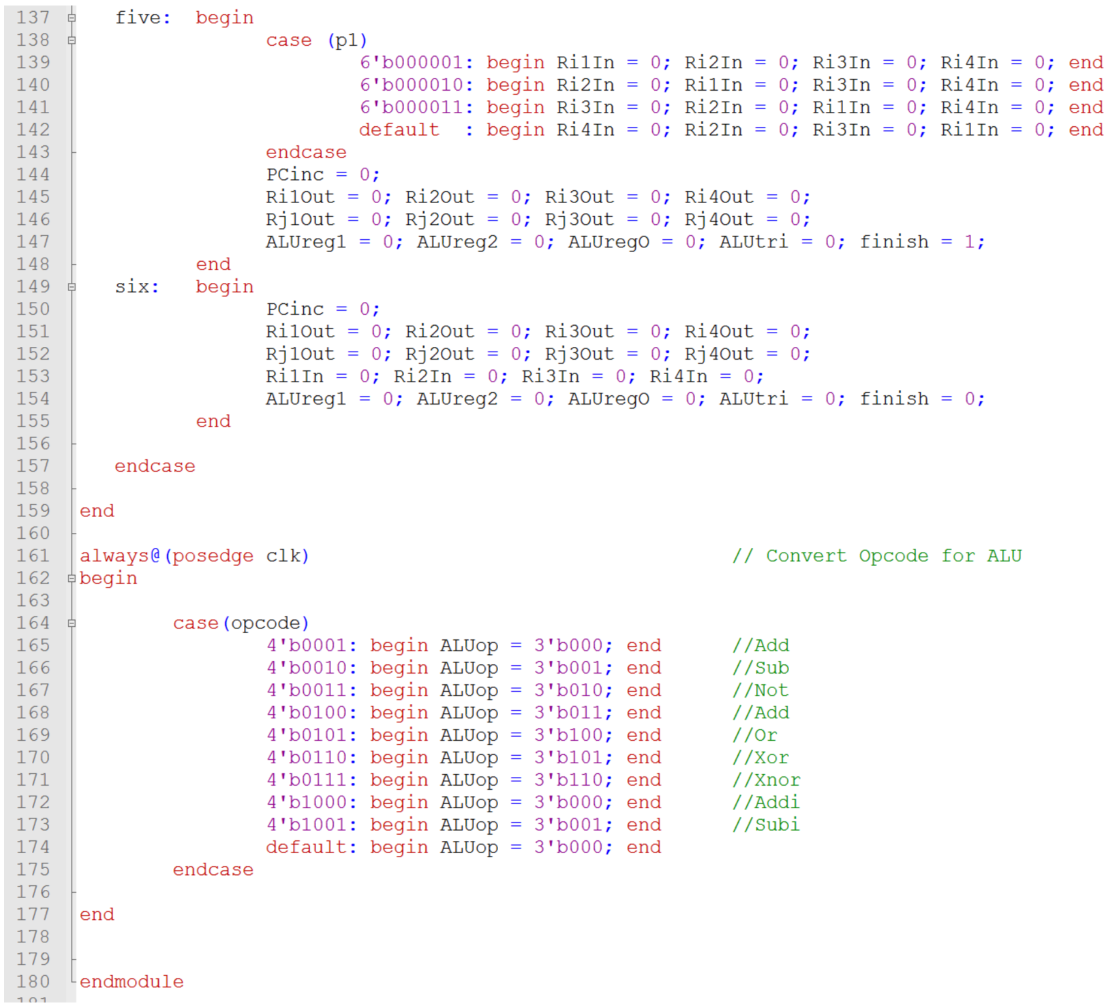
```
    three: begin
                    PCoutEn = 1; MARinEn = 1; memOp = 1; memEn = 1;
                    MDRreadEn = 0; MDRoutEn = 0; IRinEn = 0; ID_en = 0;
           end
    four:  begin
                    PCoutEn = 1; MARinEn = 1; memOp = 1; memEn = 1;
                    MDRreadEn = 0; MDRoutEn = 0; IRinEn = 0; ID_en = 0;
           end
    five:  begin
                    PCoutEn = 0; MARinEn = 0; memOp = 1; memEn = 1;
                    MDRreadEn = 1; MDRoutEn = 0; IRinEn = 0; ID_en = 0;
           end
    six:   begin
                    PCoutEn = 0; MARinEn = 0; memOp = 1; memEn = 1;
                    MDRreadEn = 1; MDRoutEn = 0; IRinEn = 0; ID_en = 0;
           end
    seven: begin
                    PCoutEn = 0; MARinEn = 0; memOp = 0; memEn = 0;
                    MDRreadEn = 0; MDRoutEn = 1; IRinEn = 1; ID_en = 0;
           end
    eight: begin
                    PCoutEn = 0; MARinEn = 0; memOp = 0; memEn = 0;
                    MDRreadEn = 0; MDRoutEn = 0; IRinEn = 0; ID_en = 1;
           end
    nine: begin
                    PCoutEn = 0; MARinEn = 0; memOp = 0; memEn = 0;
                    MDRreadEn = 0; MDRoutEn = 0; IRinEn = 0; ID_en = 0;
           end

    endcase

end




endmodule
```

- R-type FSM



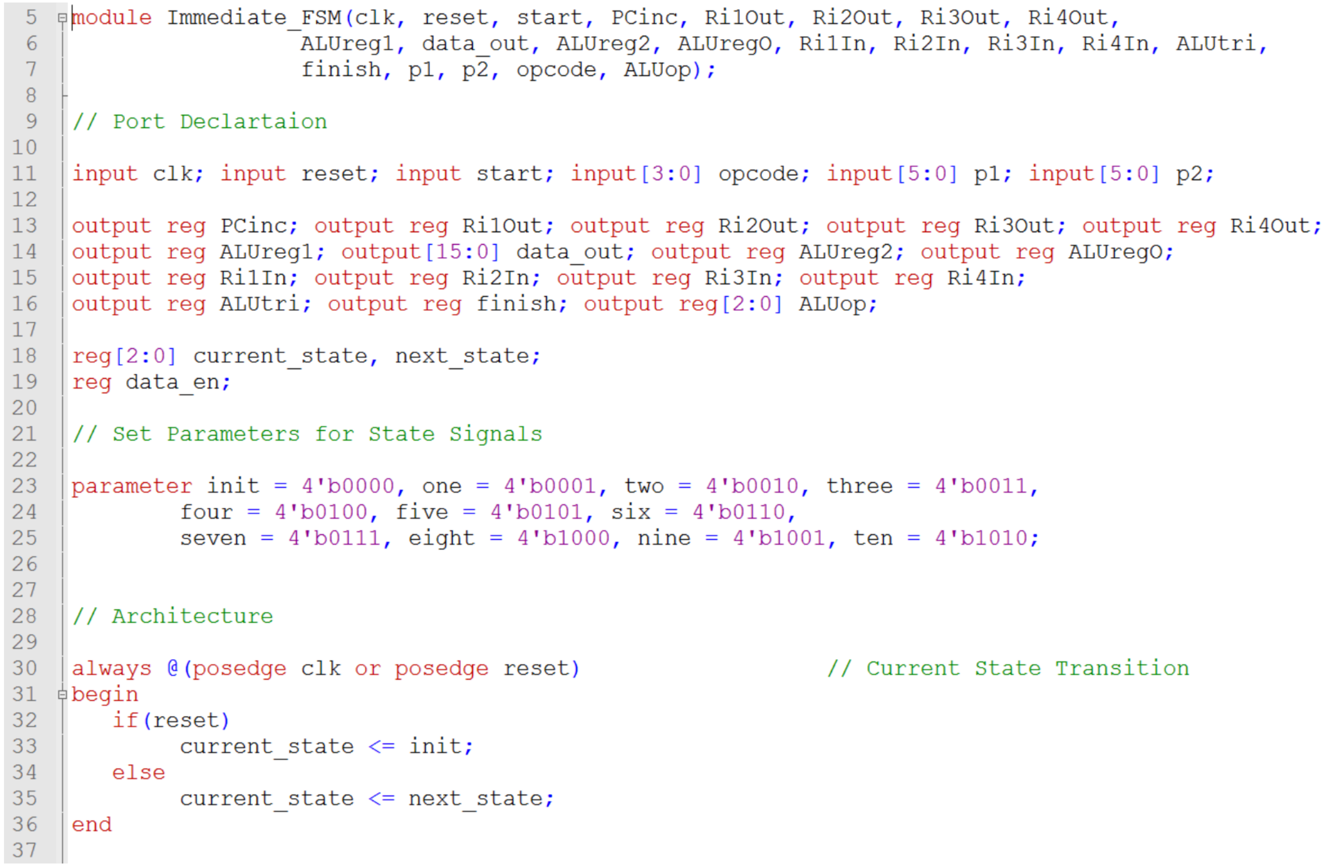
```
module Register_FSM(clk, reset, start, PCinc, Ri1Out, Ri2Out, Ri3Out, Ri4Out,
                ALUreg1, Rj1Out, Rj2Out, Rj3Out, Rj4Out, ALUreg2, ALUregO,
                Ri1In, Ri2In, Ri3In, Ri4In, ALUtri, finish, p1, p2, opcode, ALUop);

// Port Declartaion

input clk; input reset; input start; input[3:0] opcode; input[5:0] p1; input[5:0] p2;

output reg PCinc; output reg Ri1Out; output reg Ri2Out; output reg Ri3Out; output reg Ri4Out; output reg ALUreg1;
output reg Rj1Out; output reg Rj2Out; output reg Rj3Out; output reg Rj4Out;
output reg ALUreg2; output reg ALUregO; output reg Ri1In; output reg Ri2In; output reg Ri3In; output reg Ri4In;
output reg ALUtri; output reg finish;
output reg[2:0] ALUop;

reg[2:0] current_state, next_state;

// Set Parameters for State Signals

parameter init = 4'b0000, one = 4'b0001, two = 4'b0010, three = 4'b0011,
        four = 4'b0100, five = 4'b0101, six = 4'b0110,
        seven = 4'b0111, eight = 4'b1000, nine = 4'b1001, ten = 4'b1010;


// Architecture

always @(posedge clk or posedge reset)                    // Current State Transition
begin
   if(reset)
        current_state <= init;
   else
        current_state <= next_state;
end

```

```
always @(current_state or start)                        // Next State Logic
begin

   case(current_state)

   init: begin
                if(start)
                        next_state <= one;
                else
                        next_state <= init;
         end

   one:    begin next_state <= two; end
   two:    begin next_state <= three; end
   three:  begin next_state <= four; end
   four:   begin next_state <= five; end
   five:   begin next_state <= six; end
   six:    begin next_state <= init; end
   default:begin next_state <= init; end

   endcase
end


always @(current_state)                                 // Output Logic
begin

   case (current_state)

   init:  begin
                PCinc = 0;
                Ri1Out = 0; Ri2Out = 0; Ri3Out = 0; Ri4Out = 0;
                Rj1Out = 0; Rj2Out = 0; Rj3Out = 0; Rj4Out = 0;
                Ri1In = 0; Ri2In = 0; Ri3In = 0; Ri4In = 0;
                ALUreg1 = 0; ALUreg2 = 0; ALUreg0 = 0; ALUtri = 0; finish = 0;
          end
   one:   begin
                PCinc = 1; ALUreg1 = 1;
                case (p1)
                        6'b000001: begin Ri1Out = 1; Ri2Out = 0; Ri3Out = 0; Ri4Out = 0; end
                        6'b000010: begin Ri2Out = 1; Ri1Out = 0; Ri3Out = 0; Ri4Out = 0; end
                        6'b000011: begin Ri3Out = 1; Ri2Out = 0; Ri1Out = 0; Ri4Out = 0; end
                        default :  begin Ri4Out = 1; Ri2Out = 0; Ri3Out = 0; Ri1Out = 0; end
                endcase
                Rj1Out = 0; Rj2Out = 0; Rj3Out = 0; Rj4Out = 0;
                Ri1In = 0; Ri2In = 0; Ri3In = 0; Ri4In = 0;
                ALUreg2 = 0; ALUreg0 = 0; ALUtri = 0; finish = 0;
          end
   two:   begin
                ALUreg2 = 1;
                case (p1)
                        6'b000001: begin Ri1Out = 0; Ri2Out = 0; Ri3Out = 0; Ri4Out = 0; end
                        6'b000010: begin Ri2Out = 0; Ri1Out = 0; Ri3Out = 0; Ri4Out = 0; end
                        6'b000011: begin Ri3Out = 0; Ri2Out = 0; Ri1Out = 0; Ri4Out = 0; end
                        default  : begin Ri4Out = 0; Ri2Out = 0; Ri3Out = 0; Ri1Out = 0; end
                endcase

                case (p2)
                        6'b000001: begin Rj1Out = 1; Rj2Out = 0; Rj3Out = 0; Rj4Out = 0; end
                        6'b000010: begin Rj2Out = 1; Rj1Out = 0; Rj3Out = 0; Rj4Out = 0; end
                        6'b000011: begin Rj3Out = 1; Rj2Out = 0; Rj1Out = 0; Rj4Out = 0; end
                        default  : begin Rj4Out = 1; Rj2Out = 0; Rj3Out = 0; Rj1Out = 0; end
                endcase
                PCinc = 0;
                Ri1In = 0; Ri2In = 0; Ri3In = 0; Ri4In = 0;
                ALUreg1 = 0; ALUreg0 = 0; ALUtri = 0; finish = 0;
          end
   three: begin
                ALUreg0 = 1;
                case (p2)
                        6'b000001: begin Rj1Out = 0; Rj2Out = 0; Rj3Out = 0; Rj4Out = 0; end
                        6'b000010: begin Rj2Out = 0; Rj1Out = 0; Rj3Out = 0; Rj4Out = 0; end
                        6'b000011: begin Rj3Out = 0; Rj2Out = 0; Rj1Out = 0; Rj4Out = 0; end
                        default  : begin Rj4Out = 0; Rj2Out = 0; Rj3Out = 0; Rj1Out = 0; end
                endcase

                case (p1)
                        6'b000001: begin Ri1In = 1; Ri2In = 0; Ri3In = 0; Ri4In = 0; end
                        6'b000010: begin Ri2In = 1; Ri1In = 0; Ri3In = 0; Ri4In = 0; end
                        6'b000011: begin Ri3In = 1; Ri2In = 0; Ri1In = 0; Ri4In = 0; end
                        default  : begin Ri4In = 1; Ri2In = 0; Ri3In = 0; Ri1In = 0; end
                endcase
                PCinc = 0;
                Ri1Out = 0; Ri2Out = 0; Ri3Out = 0; Ri4Out = 0;
                ALUreg1 = 0; ALUreg2 = 0;ALUtri = 0; finish = 0;
          end
   four:  begin
                PCinc = 0;
                Ri1Out = 0; Ri2Out = 0; Ri3Out = 0; Ri4Out = 0;
                Rj1Out = 0; Rj2Out = 0; Rj3Out = 0; Rj4Out = 0;
                case (p1)
                        6'b000001: begin Ri1In = 1; Ri2In = 0; Ri3In = 0; Ri4In = 0; end
                        6'b000010: begin Ri2In = 1; Ri1In = 0; Ri3In = 0; Ri4In = 0; end
                        6'b000011: begin Ri3In = 1; Ri2In = 0; Ri1In = 0; Ri4In = 0; end
                        default  : begin Ri4In = 1; Ri2In = 0; Ri3In = 0; Ri1In = 0; end
                endcase
                ALUreg1 = 0; ALUreg2 = 0; ALUreg0 = 0; ALUtri = 1; finish = 0;
          end
```

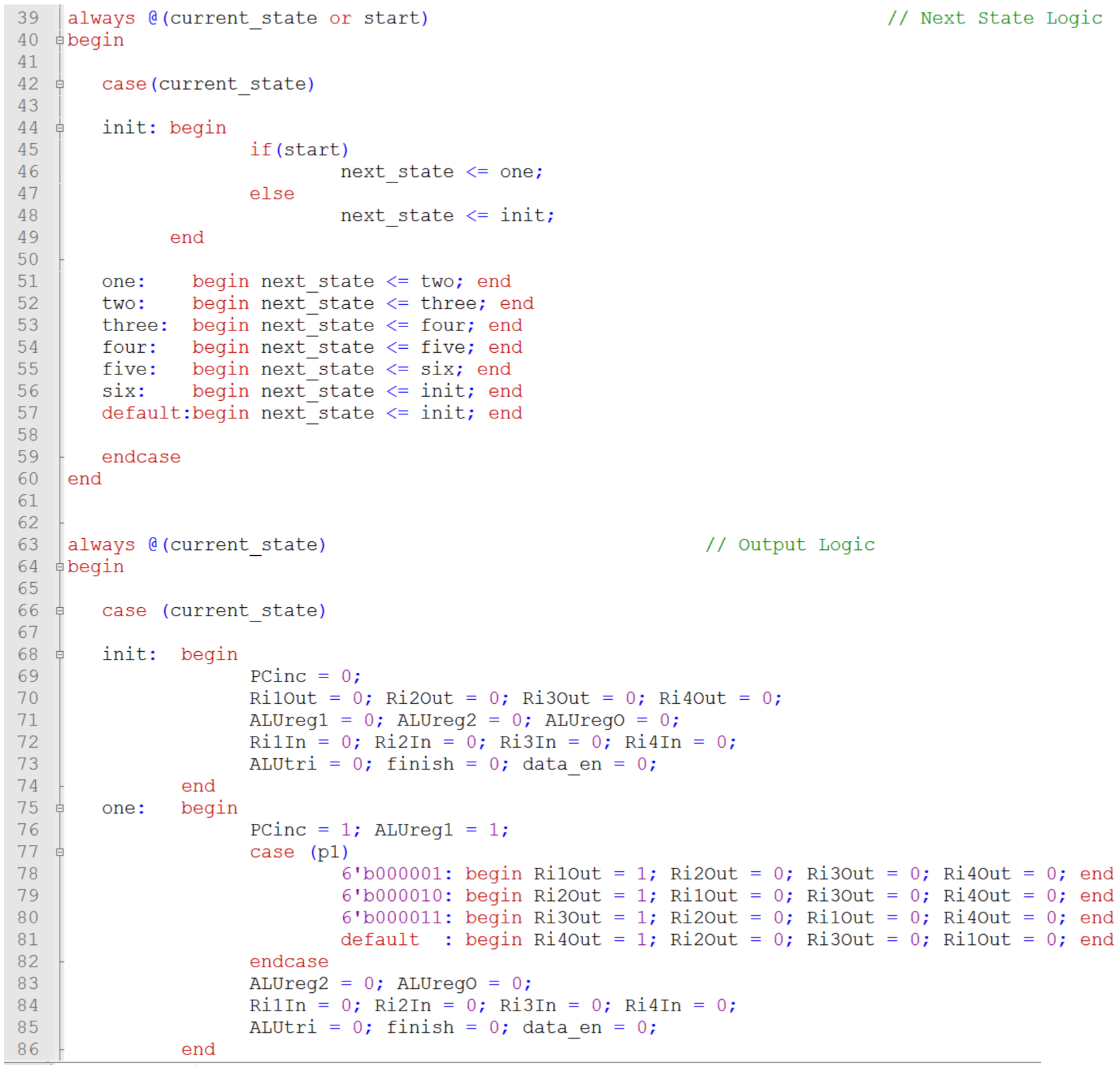


```
    five:  begin
                case (p1)
                        6'b000001: begin Ri1In = 0; Ri2In = 0; Ri3In = 0; Ri4In = 0; end
                        6'b000010: begin Ri2In = 0; Ri1In = 0; Ri3In = 0; Ri4In = 0; end
                        6'b000011: begin Ri3In = 0; Ri2In = 0; Ri1In = 0; Ri4In = 0; end
                        default  : begin Ri4In = 0; Ri2In = 0; Ri3In = 0; Ri1In = 0; end
                endcase
                PCinc = 0;
                Ri1Out = 0; Ri2Out = 0; Ri3Out = 0; Ri4Out = 0;
                Rj1Out = 0; Rj2Out = 0; Rj3Out = 0; Rj4Out = 0;
                ALUreg1 = 0; ALUreg2 = 0; ALUregO = 0; ALUtri = 0; finish = 1;
          end
    six:   begin
                PCinc = 0;
                Ri1Out = 0; Ri2Out = 0; Ri3Out = 0; Ri4Out = 0;
                Rj1Out = 0; Rj2Out = 0; Rj3Out = 0; Rj4Out = 0;
                Ri1In = 0; Ri2In = 0; Ri3In = 0; Ri4In = 0;
                ALUreg1 = 0; ALUreg2 = 0; ALUregO = 0; ALUtri = 0; finish = 0;
          end

    endcase

end

always@(posedge clk)                                    // Convert Opcode for ALU
begin

        case(opcode)
                4'b0001: begin ALUop = 3'b000; end      //Add
                4'b0010: begin ALUop = 3'b001; end      //Sub
                4'b0011: begin ALUop = 3'b010; end      //Not
                4'b0100: begin ALUop = 3'b011; end      //Add
                4'b0101: begin ALUop = 3'b100; end      //Or
                4'b0110: begin ALUop = 3'b101; end      //Xor
                4'b0111: begin ALUop = 3'b110; end      //Xnor
                4'b1000: begin ALUop = 3'b000; end      //Addi
                4'b1001: begin ALUop = 3'b001; end      //Subi
                default: begin ALUop = 3'b000; end
        endcase

end


endmodule
```

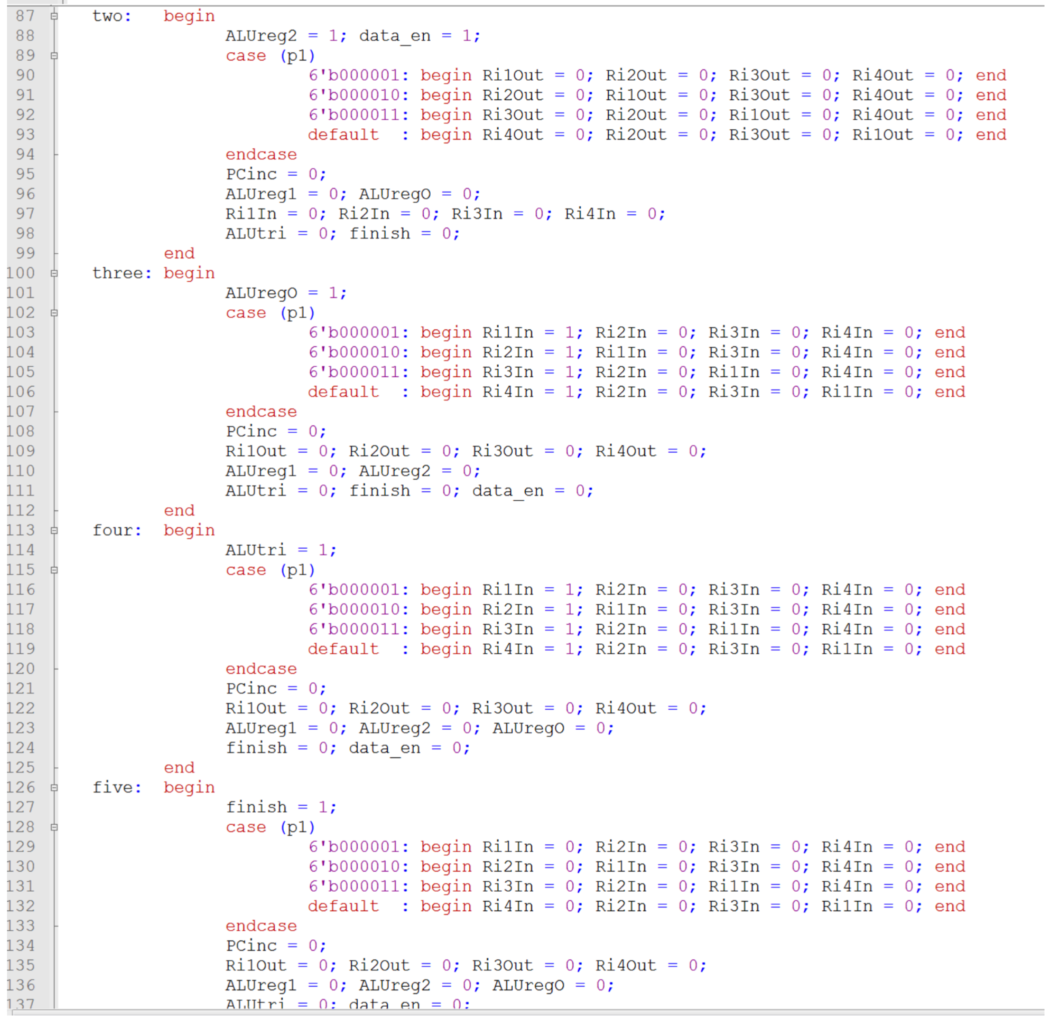
- Immediate FSM



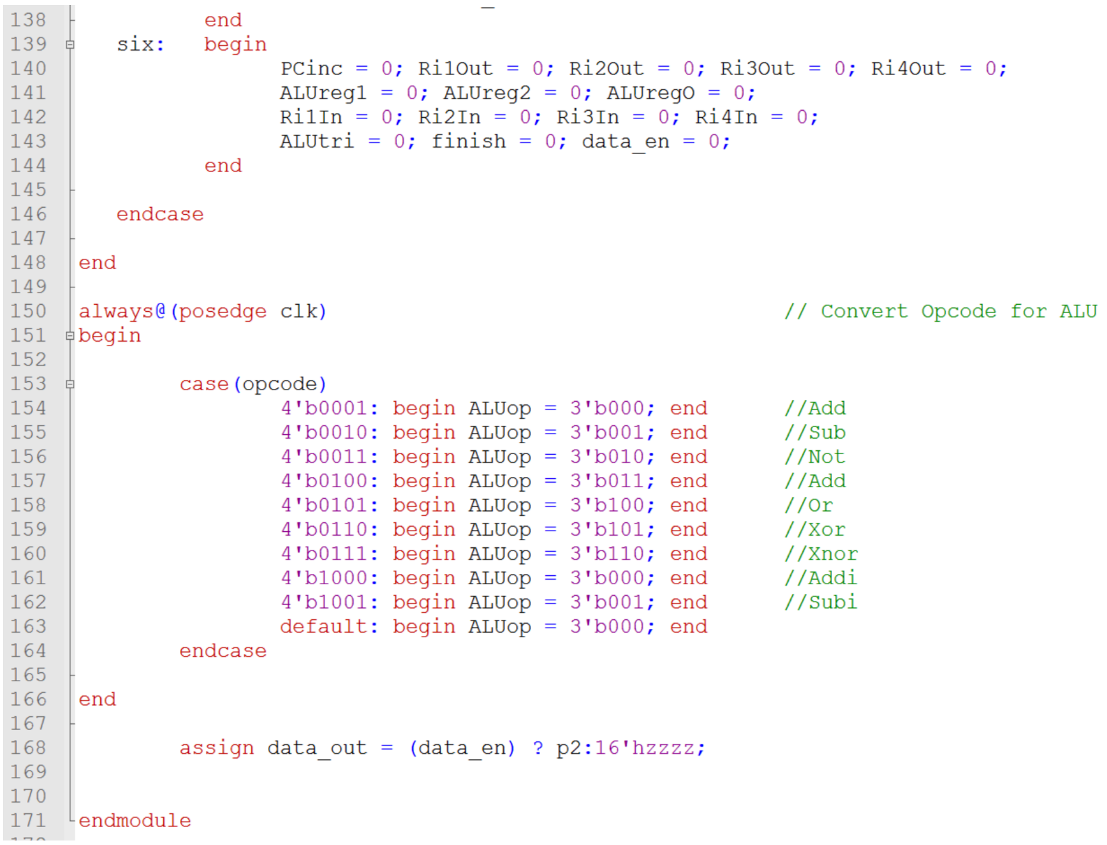
```
module Immediate_FSM(clk, reset, start, PCinc, Ri1Out, Ri2Out, Ri3Out, Ri4Out,
                     ALUreg1, data_out, ALUreg2, ALUregO, Ri1In, Ri2In, Ri3In, Ri4In, ALUtri,
                     finish, p1, p2, opcode, ALUop);

// Port Declartaion

input clk; input reset; input start; input[3:0] opcode; input[5:0] p1; input[5:0] p2;

output reg PCinc; output reg Ri1Out; output reg Ri2Out; output reg Ri3Out; output reg Ri4Out;
output reg ALUreg1; output[15:0] data_out; output reg ALUreg2; output reg ALUregO;
output reg Ri1In; output reg Ri2In; output reg Ri3In; output reg Ri4In;
output reg ALUtri; output reg finish; output reg[2:0] ALUop;

reg[2:0] current_state, next_state;
reg data_en;

// Set Parameters for State Signals

parameter init = 4'b0000, one = 4'b0001, two = 4'b0010, three = 4'b0011,
          four = 4'b0100, five = 4'b0101, six = 4'b0110,
          seven = 4'b0111, eight = 4'b1000, nine = 4'b1001, ten = 4'b1010;


// Architecture

always @(posedge clk or posedge reset)                      // Current State Transition
begin
   if(reset)
         current_state <= init;
   else
         current_state <= next_state;
end
```



```
always @(current_state or start)                                    // Next State Logic
begin

   case(current_state)

   init: begin
              if(start)
                      next_state <= one;
              else
                      next_state <= init;
         end

   one:    begin next_state <= two; end
   two:    begin next_state <= three; end
   three:  begin next_state <= four; end
   four:   begin next_state <= five; end
   five:   begin next_state <= six; end
   six:    begin next_state <= init; end
   default:begin next_state <= init; end

   endcase
end


always @(current_state)                                // Output Logic
begin

   case (current_state)

   init:  begin
              PCinc = 0;
              Ri1Out = 0; Ri2Out = 0; Ri3Out = 0; Ri4Out = 0;
              ALUreg1 = 0; ALUreg2 = 0; ALUregO = 0;
              Ri1In = 0; Ri2In = 0; Ri3In = 0; Ri4In = 0;
              ALUtri = 0; finish = 0; data_en = 0;
          end
   one:   begin
              PCinc = 1; ALUreg1 = 1;
              case (p1)
                      6'b000001: begin Ri1Out = 1; Ri2Out = 0; Ri3Out = 0; Ri4Out = 0; end
                      6'b000010: begin Ri2Out = 1; Ri1Out = 0; Ri3Out = 0; Ri4Out = 0; end
                      6'b000011: begin Ri3Out = 1; Ri2Out = 0; Ri1Out = 0; Ri4Out = 0; end
                      default  : begin Ri4Out = 1; Ri2Out = 0; Ri3Out = 0; Ri1Out = 0; end
              endcase
              ALUreg2 = 0; ALUregO = 0;
              Ri1In = 0; Ri2In = 0; Ri3In = 0; Ri4In = 0;
              ALUtri = 0; finish = 0; data_en = 0;
          end
```

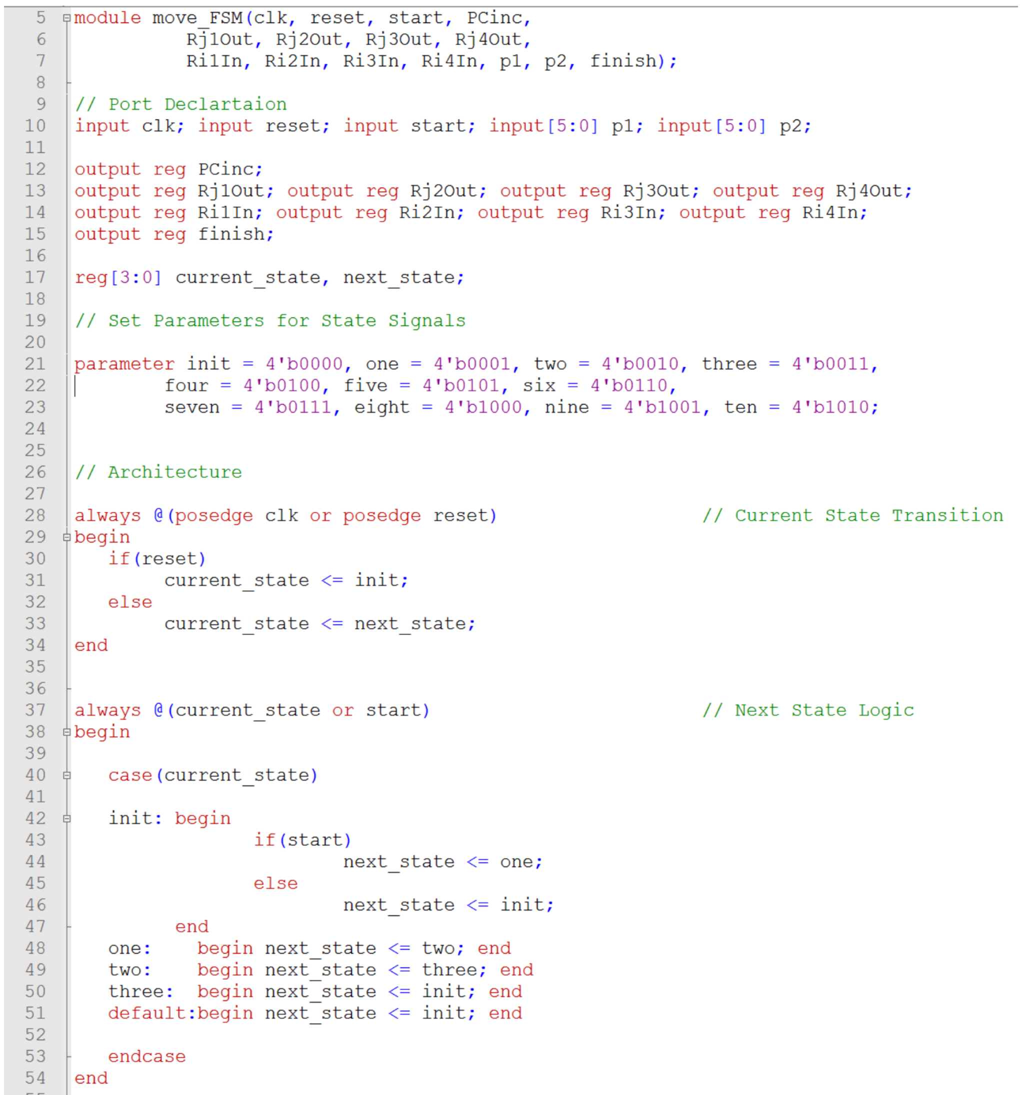


```
   two:   begin
              ALUreg2 = 1; data_en = 1;
              case (p1)
                      6'b000001: begin Ri1Out = 0; Ri2Out = 0; Ri3Out = 0; Ri4Out = 0; end
                      6'b000010: begin Ri2Out = 0; Ri1Out = 0; Ri3Out = 0; Ri4Out = 0; end
                      6'b000011: begin Ri3Out = 0; Ri2Out = 0; Ri1Out = 0; Ri4Out = 0; end
                      default  : begin Ri4Out = 0; Ri2Out = 0; Ri3Out = 0; Ri1Out = 0; end
              endcase
              PCinc = 0;
              ALUreg1 = 0; ALUregO = 0;
              Ri1In = 0; Ri2In = 0; Ri3In = 0; Ri4In = 0;
              ALUtri = 0; finish = 0;
          end
   three: begin
              ALUregO = 1;
              case (p1)
                      6'b000001: begin Ri1In = 1; Ri2In = 0; Ri3In = 0; Ri4In = 0; end
                      6'b000010: begin Ri2In = 1; Ri1In = 0; Ri3In = 0; Ri4In = 0; end
                      6'b000011: begin Ri3In = 1; Ri2In = 0; Ri1In = 0; Ri4In = 0; end
                      default  : begin Ri4In = 1; Ri2In = 0; Ri3In = 0; Ri1In = 0; end
              endcase
              PCinc = 0;
              Ri1Out = 0; Ri2Out = 0; Ri3Out = 0; Ri4Out = 0;
              ALUreg1 = 0; ALUreg2 = 0;
              ALUtri = 0; finish = 0; data_en = 0;
          end
   four:  begin
              ALUtri = 1;
              case (p1)
                      6'b000001: begin Ri1In = 1; Ri2In = 0; Ri3In = 0; Ri4In = 0; end
                      6'b000010: begin Ri2In = 1; Ri1In = 0; Ri3In = 0; Ri4In = 0; end
                      6'b000011: begin Ri3In = 1; Ri2In = 0; Ri1In = 0; Ri4In = 0; end
                      default  : begin Ri4In = 1; Ri2In = 0; Ri3In = 0; Ri1In = 0; end
              endcase
              PCinc = 0;
              Ri1Out = 0; Ri2Out = 0; Ri3Out = 0; Ri4Out = 0;
              ALUreg1 = 0; ALUreg2 = 0; ALUregO = 0;
              finish = 0; data_en = 0;
          end
   five:  begin
              finish = 1;
              case (p1)
                      6'b000001: begin Ri1In = 0; Ri2In = 0; Ri3In = 0; Ri4In = 0; end
                      6'b000010: begin Ri2In = 0; Ri1In = 0; Ri3In = 0; Ri4In = 0; end
                      6'b000011: begin Ri3In = 0; Ri2In = 0; Ri1In = 0; Ri4In = 0; end
                      default  : begin Ri4In = 0; Ri2In = 0; Ri3In = 0; Ri1In = 0; end
              endcase
              PCinc = 0;
              Ri1Out = 0; Ri2Out = 0; Ri3Out = 0; Ri4Out = 0;
              ALUreg1 = 0; ALUreg2 = 0; ALUregO = 0;
              ALUtri = 0; data_en = 0;
```

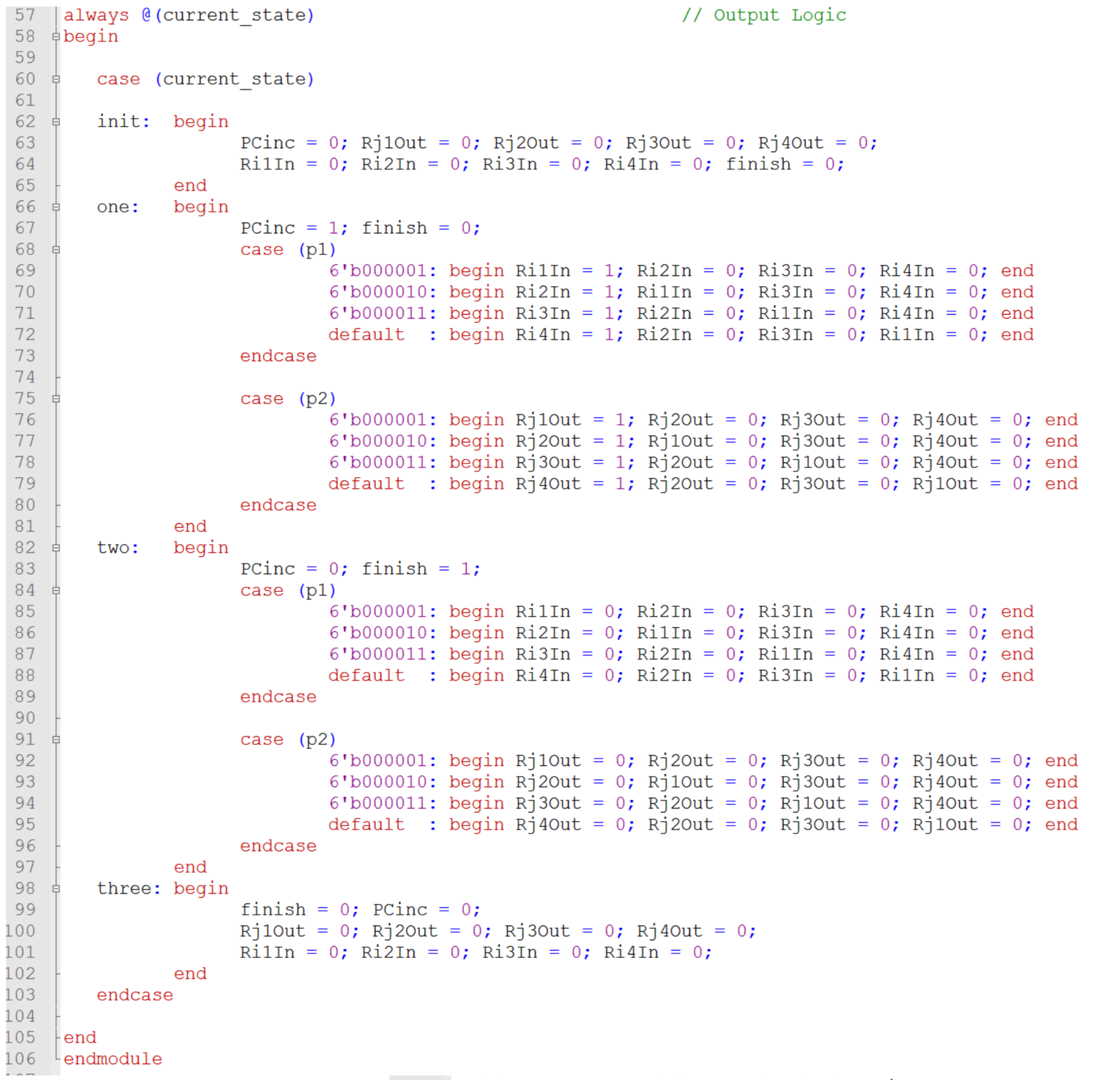


```
          end
   six:   begin
                PCinc = 0; Ri1Out = 0; Ri2Out = 0; Ri3Out = 0; Ri4Out = 0;
                ALUreg1 = 0; ALUreg2 = 0; ALUregO = 0;
                Ri1In = 0; Ri2In = 0; Ri3In = 0; Ri4In = 0;
                ALUtri = 0; finish = 0; data_en = 0;
          end

   endcase

end

always@(posedge clk)                                    // Convert Opcode for ALU
begin

        case(opcode)
                4'b0001: begin ALUop = 3'b000; end      //Add
                4'b0010: begin ALUop = 3'b001; end      //Sub
                4'b0011: begin ALUop = 3'b010; end      //Not
                4'b0100: begin ALUop = 3'b011; end      //Add
                4'b0101: begin ALUop = 3'b100; end      //Or
                4'b0110: begin ALUop = 3'b101; end      //Xor
                4'b0111: begin ALUop = 3'b110; end      //Xnor
                4'b1000: begin ALUop = 3'b000; end      //Addi
                4'b1001: begin ALUop = 3'b001; end      //Subi
                default: begin ALUop = 3'b000; end
        endcase

end

        assign data_out = (data_en) ? p2:16'hzzzz;


endmodule
```

- Move FSM

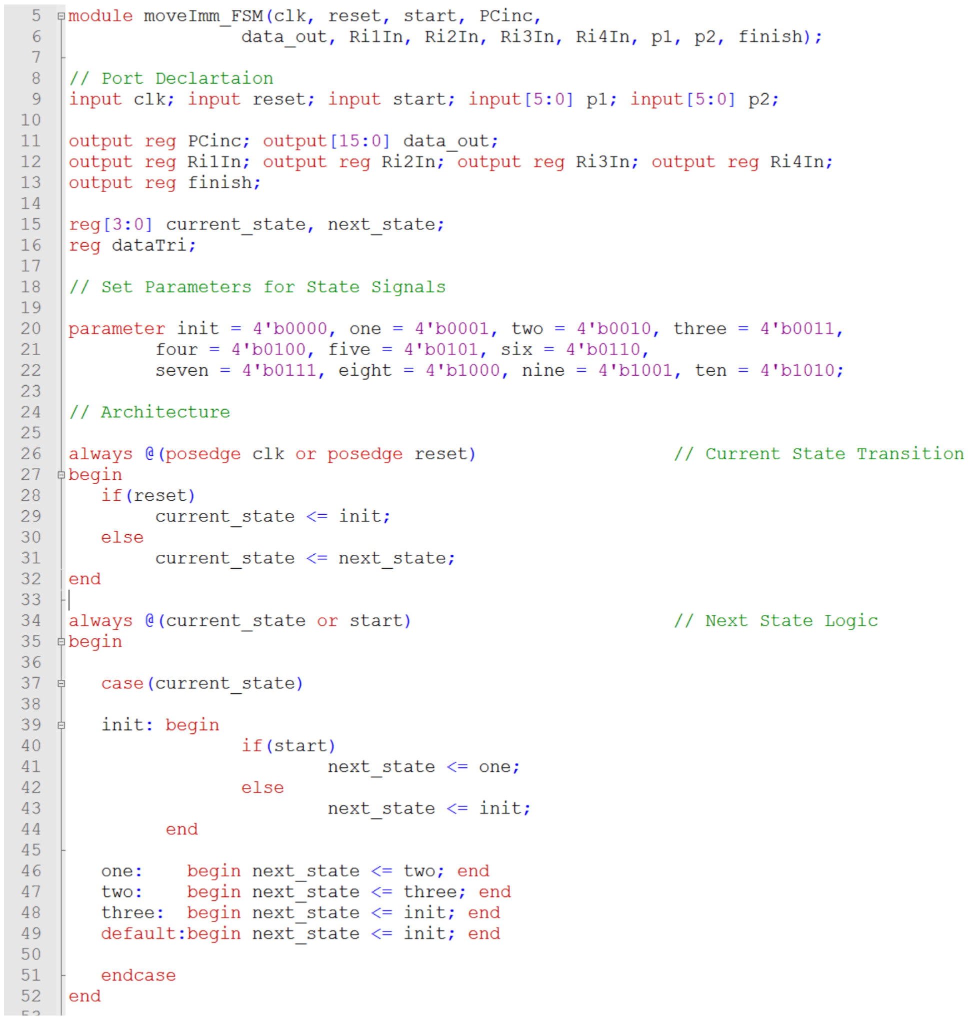


```
module move_FSM(clk, reset, start, PCinc,
          Rj1Out, Rj2Out, Rj3Out, Rj4Out,
          Ri1In, Ri2In, Ri3In, Ri4In, p1, p2, finish);

// Port Declartaion
input clk; input reset; input start; input[5:0] p1; input[5:0] p2;

output reg PCinc;
output reg Rj1Out; output reg Rj2Out; output reg Rj3Out; output reg Rj4Out;
output reg Ri1In; output reg Ri2In; output reg Ri3In; output reg Ri4In;
output reg finish;

reg[3:0] current_state, next_state;

// Set Parameters for State Signals

parameter init = 4'b0000, one = 4'b0001, two = 4'b0010, three = 4'b0011,
        four = 4'b0100, five = 4'b0101, six = 4'b0110,
        seven = 4'b0111, eight = 4'b1000, nine = 4'b1001, ten = 4'b1010;


// Architecture

always @(posedge clk or posedge reset)                  // Current State Transition
begin
   if(reset)
        current_state <= init;
   else
        current_state <= next_state;
end


always @(current_state or start)                        // Next State Logic
begin

   case(current_state)

   init: begin
                if(start)
                        next_state <= one;
                else
                        next_state <= init;
         end
   one:    begin next_state <= two; end
   two:    begin next_state <= three; end
   three:  begin next_state <= init; end
   default:begin next_state <= init; end

   endcase
end
```

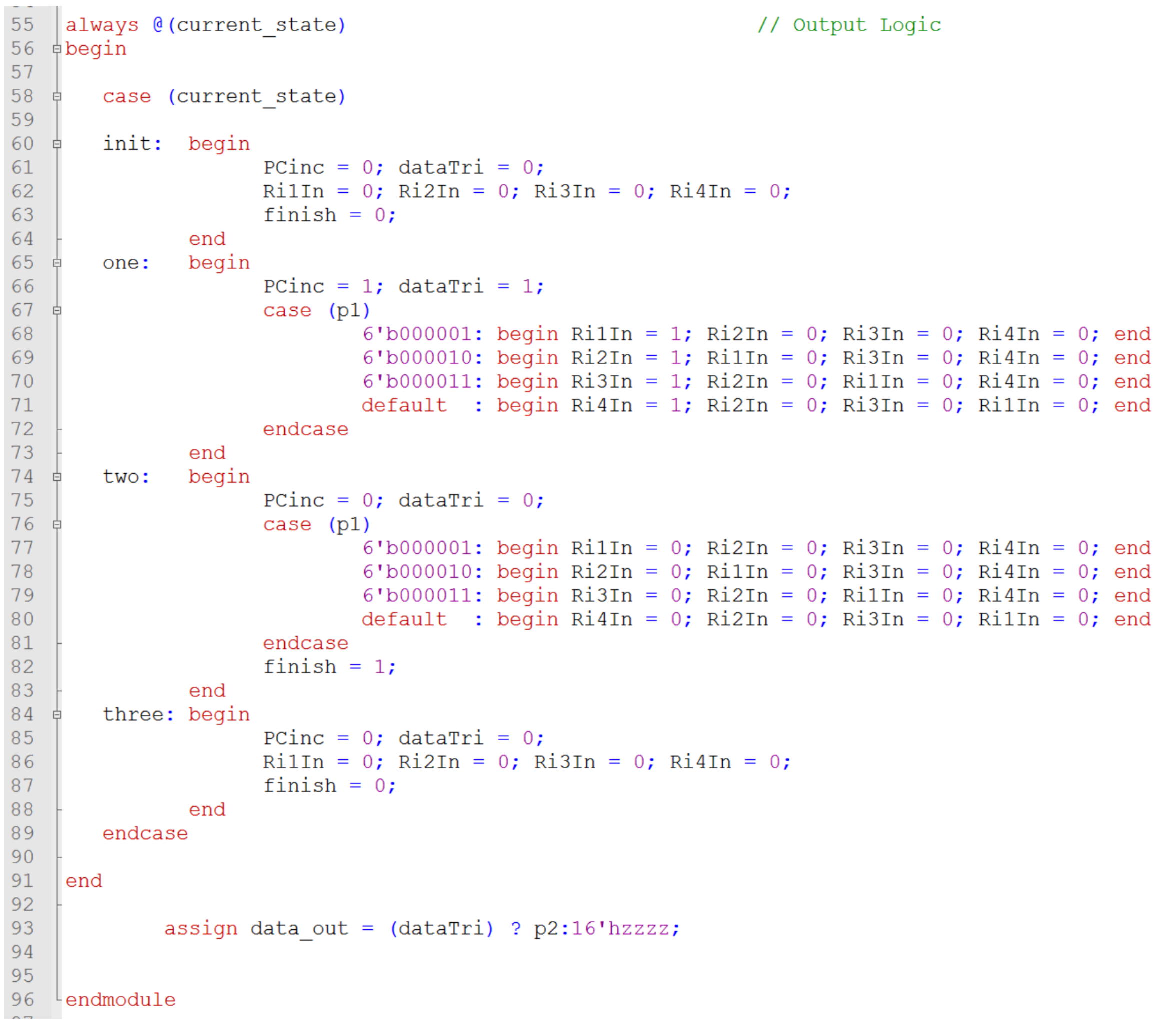


```verilog
always @(current_state)                                 // Output Logic
begin

   case (current_state)

   init:  begin
                PCinc = 0; Rj1Out = 0; Rj2Out = 0; Rj3Out = 0; Rj4Out = 0;
                Ri1In = 0; Ri2In = 0; Ri3In = 0; Ri4In = 0; finish = 0;
          end
   one:   begin
                PCinc = 1; finish = 0;
                case (p1)
                        6'b000001: begin Ri1In = 1; Ri2In = 0; Ri3In = 0; Ri4In = 0; end
                        6'b000010: begin Ri2In = 1; Ri1In = 0; Ri3In = 0; Ri4In = 0; end
                        6'b000011: begin Ri3In = 1; Ri2In = 0; Ri1In = 0; Ri4In = 0; end
                        default  : begin Ri4In = 1; Ri2In = 0; Ri3In = 0; Ri1In = 0; end
                endcase

                case (p2)
                        6'b000001: begin Rj1Out = 1; Rj2Out = 0; Rj3Out = 0; Rj4Out = 0; end
                        6'b000010: begin Rj2Out = 1; Rj1Out = 0; Rj3Out = 0; Rj4Out = 0; end
                        6'b000011: begin Rj3Out = 1; Rj2Out = 0; Rj1Out = 0; Rj4Out = 0; end
                        default  : begin Rj4Out = 1; Rj2Out = 0; Rj3Out = 0; Rj1Out = 0; end
                endcase
          end
   two:   begin
                PCinc = 0; finish = 1;
                case (p1)
                        6'b000001: begin Ri1In = 0; Ri2In = 0; Ri3In = 0; Ri4In = 0; end
                        6'b000010: begin Ri2In = 0; Ri1In = 0; Ri3In = 0; Ri4In = 0; end
                        6'b000011: begin Ri3In = 0; Ri2In = 0; Ri1In = 0; Ri4In = 0; end
                        default  : begin Ri4In = 0; Ri2In = 0; Ri3In = 0; Ri1In = 0; end
                endcase

                case (p2)
                        6'b000001: begin Rj1Out = 0; Rj2Out = 0; Rj3Out = 0; Rj4Out = 0; end
                        6'b000010: begin Rj2Out = 0; Rj1Out = 0; Rj3Out = 0; Rj4Out = 0; end
                        6'b000011: begin Rj3Out = 0; Rj2Out = 0; Rj1Out = 0; Rj4Out = 0; end
                        default  : begin Rj4Out = 0; Rj2Out = 0; Rj3Out = 0; Rj1Out = 0; end
                endcase
          end
   three: begin
                finish = 0; PCinc = 0;
                Rj1Out = 0; Rj2Out = 0; Rj3Out = 0; Rj4Out = 0;
                Ri1In = 0; Ri2In = 0; Ri3In = 0; Ri4In = 0;
          end
   endcase

end
endmodule
```

- MoveImm FSM

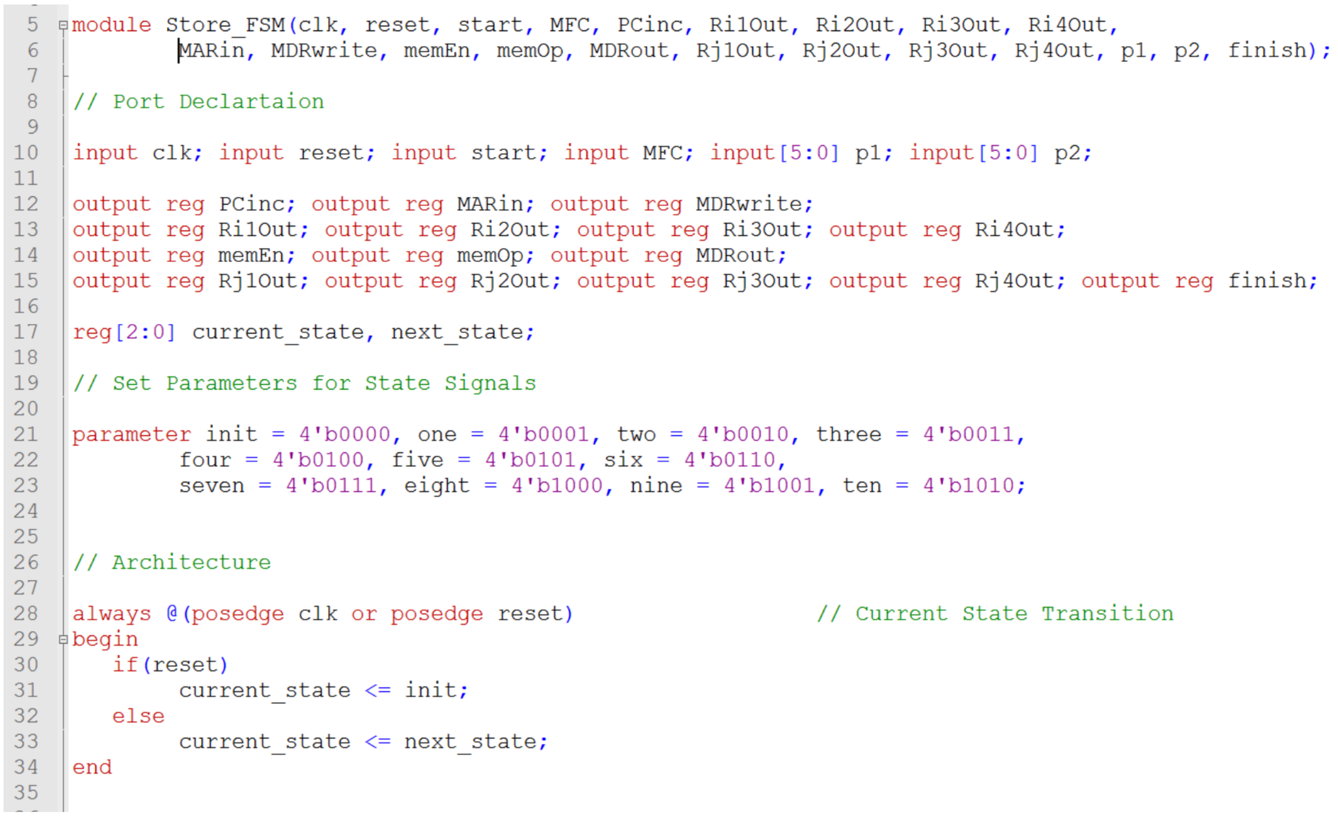


```verilog
module moveImm_FSM(clk, reset, start, PCinc,
                data_out, Ri1In, Ri2In, Ri3In, Ri4In, p1, p2, finish);

// Port Declartaion
input clk; input reset; input start; input[5:0] p1; input[5:0] p2;

output reg PCinc; output[15:0] data_out;
output reg Ri1In; output reg Ri2In; output reg Ri3In; output reg Ri4In;
output reg finish;

reg[3:0] current_state, next_state;
reg dataTri;

// Set Parameters for State Signals

parameter init = 4'b0000, one = 4'b0001, two = 4'b0010, three = 4'b0011,
        four = 4'b0100, five = 4'b0101, six = 4'b0110,
        seven = 4'b0111, eight = 4'b1000, nine = 4'b1001, ten = 4'b1010;

// Architecture

always @(posedge clk or posedge reset)                  // Current State Transition
begin
   if(reset)
        current_state <= init;
   else
        current_state <= next_state;
end

always @(current_state or start)                        // Next State Logic
begin

   case(current_state)

   init: begin
                if(start)
                        next_state <= one;
                else
                        next_state <= init;
         end

   one:     begin next_state <= two; end
   two:     begin next_state <= three; end
   three:   begin next_state <= init; end
   default:begin next_state <= init; end

   endcase
end
```

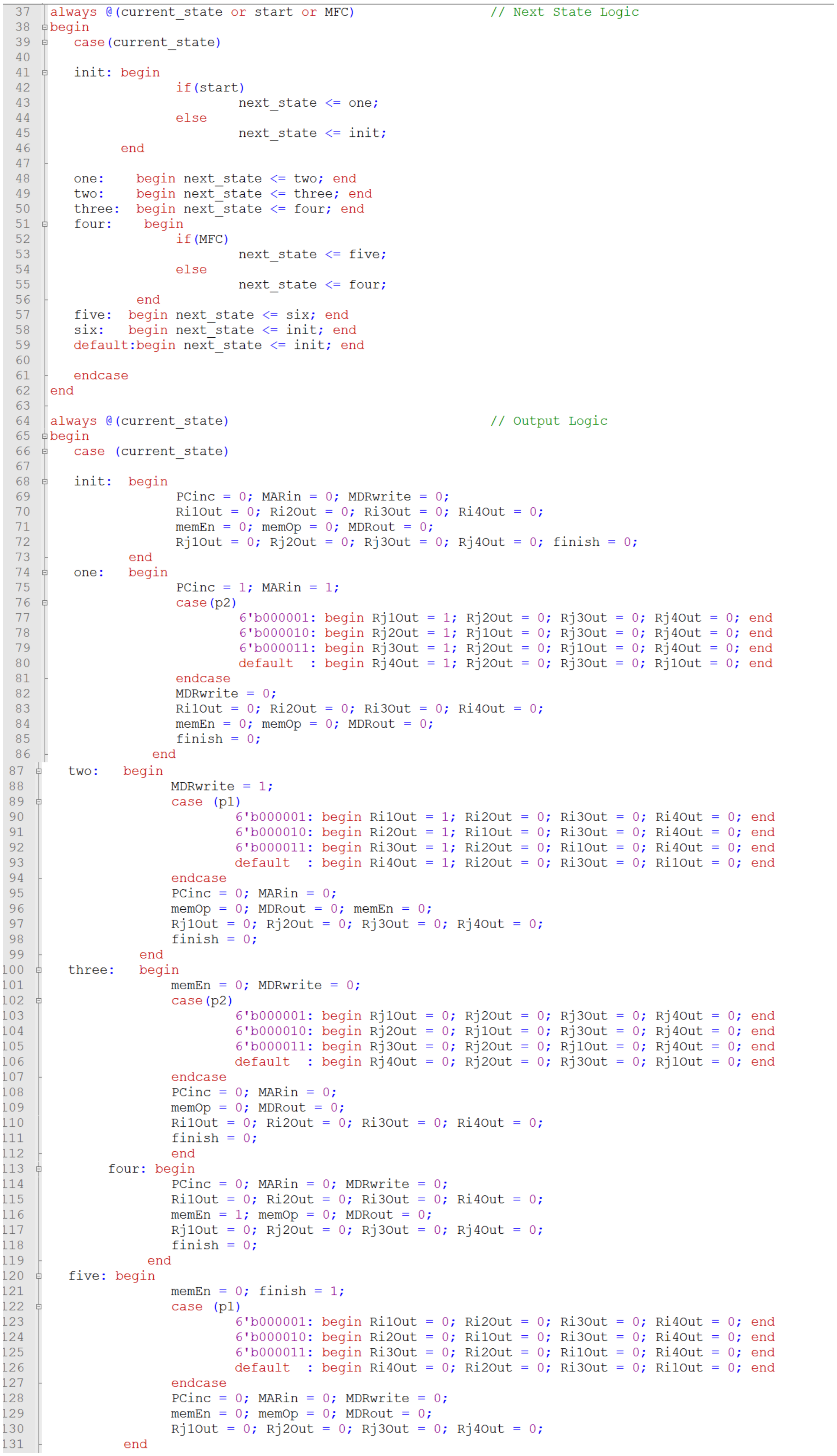


```
always @(current_state)                              // Output Logic
begin

   case (current_state)

   init:  begin
                PCinc = 0; dataTri = 0;
                Ri1In = 0; Ri2In = 0; Ri3In = 0; Ri4In = 0;
                finish = 0;
          end
   one:   begin
                PCinc = 1; dataTri = 1;
                case (p1)
                        6'b000001: begin Ri1In = 1; Ri2In = 0; Ri3In = 0; Ri4In = 0; end
                        6'b000010: begin Ri2In = 1; Ri1In = 0; Ri3In = 0; Ri4In = 0; end
                        6'b000011: begin Ri3In = 1; Ri2In = 0; Ri1In = 0; Ri4In = 0; end
                        default  : begin Ri4In = 1; Ri2In = 0; Ri3In = 0; Ri1In = 0; end
                endcase
          end
   two:   begin
                PCinc = 0; dataTri = 0;
                case (p1)
                        6'b000001: begin Ri1In = 0; Ri2In = 0; Ri3In = 0; Ri4In = 0; end
                        6'b000010: begin Ri2In = 0; Ri1In = 0; Ri3In = 0; Ri4In = 0; end
                        6'b000011: begin Ri3In = 0; Ri2In = 0; Ri1In = 0; Ri4In = 0; end
                        default  : begin Ri4In = 0; Ri2In = 0; Ri3In = 0; Ri1In = 0; end
                endcase
                finish = 1;
          end
   three: begin
                PCinc = 0; dataTri = 0;
                Ri1In = 0; Ri2In = 0; Ri3In = 0; Ri4In = 0;
                finish = 0;
          end
   endcase

end

        assign data_out = (dataTri) ? p2:16'hzzzz;


endmodule
```

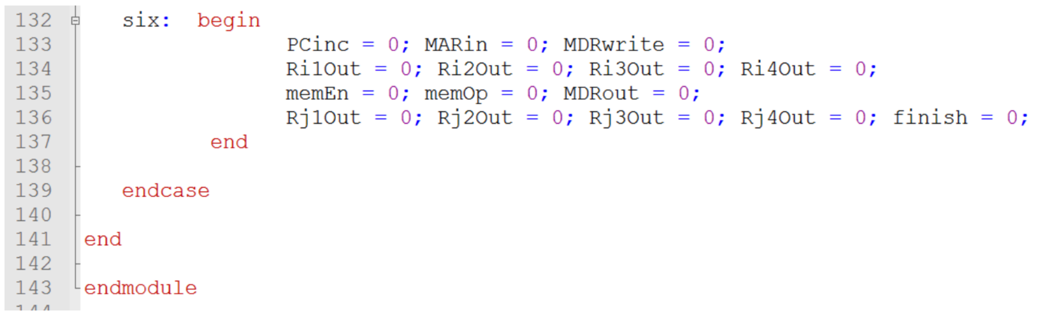
- Store FSM



```
module Store_FSM(clk, reset, start, MFC, PCinc, Ri1Out, Ri2Out, Ri3Out, Ri4Out,
        MARin, MDRwrite, memEn, memOp, MDRout, Rj1Out, Rj2Out, Rj3Out, Rj4Out, p1, p2, finish);

// Port Declartaion

input clk; input reset; input start; input MFC; input[5:0] p1; input[5:0] p2;

output reg PCinc; output reg MARin; output reg MDRwrite;
output reg Ri1Out; output reg Ri2Out; output reg Ri3Out; output reg Ri4Out;
output reg memEn; output reg memOp; output reg MDRout;
output reg Rj1Out; output reg Rj2Out; output reg Rj3Out; output reg Rj4Out; output reg finish;

reg[2:0] current_state, next_state;

// Set Parameters for State Signals

parameter init = 4'b0000, one = 4'b0001, two = 4'b0010, three = 4'b0011,
        four = 4'b0100, five = 4'b0101, six = 4'b0110,
        seven = 4'b0111, eight = 4'b1000, nine = 4'b1001, ten = 4'b1010;


// Architecture

always @(posedge clk or posedge reset)                    // Current State Transition
begin
   if(reset)
        current_state <= init;
   else
        current_state <= next_state;
end

```

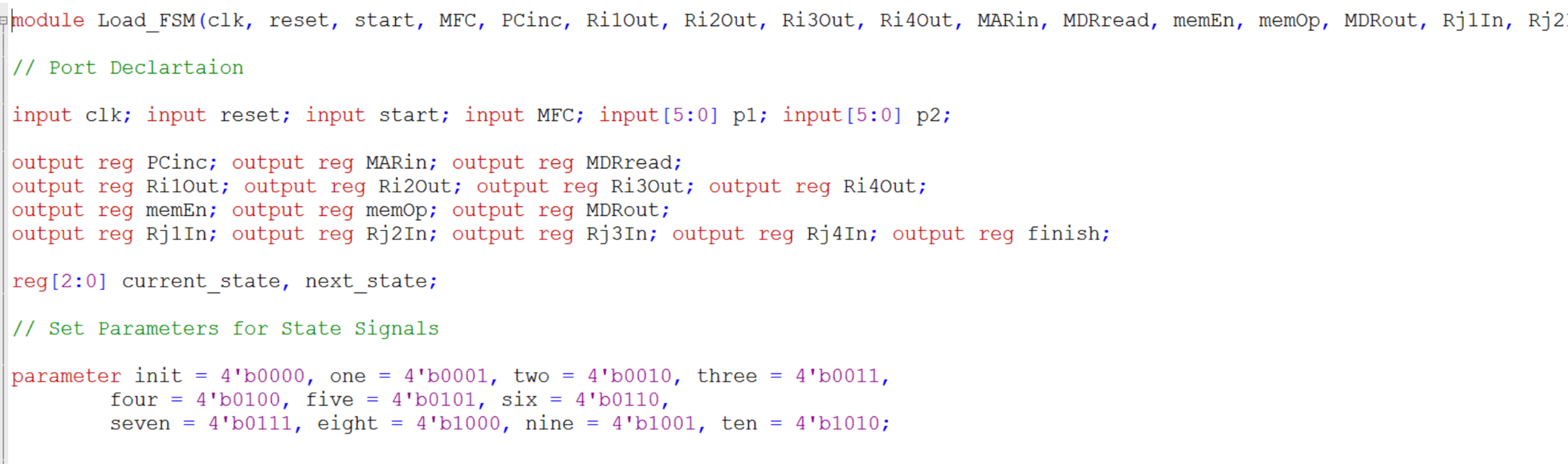


```
always @(current_state or start or MFC)                 // Next State Logic
begin
   case(current_state)

   init: begin
                if(start)
                        next_state <= one;
                else
                        next_state <= init;
         end

   one:    begin next_state <= two; end
   two:    begin next_state <= three; end
   three:  begin next_state <= four; end
   four:    begin
                if(MFC)
                        next_state <= five;
                else
                        next_state <= four;
           end
   five:  begin next_state <= six; end
   six:   begin next_state <= init; end
   default:begin next_state <= init; end

   endcase
end

always @(current_state)                                 // Output Logic
begin
   case (current_state)

   init:  begin
                PCinc = 0; MARin = 0; MDRwrite = 0;
                RilOut = 0; Ri2Out = 0; Ri3Out = 0; Ri4Out = 0;
                memEn = 0; memOp = 0; MDRout = 0;
                Rj1Out = 0; Rj2Out = 0; Rj3Out = 0; Rj4Out = 0; finish = 0;
          end
   one:   begin
                PCinc = 1; MARin = 1;
                case(p2)
                        6'b000001: begin Rj1Out = 1; Rj2Out = 0; Rj3Out = 0; Rj4Out = 0; end
                        6'b000010: begin Rj2Out = 1; Rj1Out = 0; Rj3Out = 0; Rj4Out = 0; end
                        6'b000011: begin Rj3Out = 1; Rj2Out = 0; Rj1Out = 0; Rj4Out = 0; end
                        default  : begin Rj4Out = 1; Rj2Out = 0; Rj3Out = 0; Rj1Out = 0; end
                endcase
                MDRwrite = 0;
                Ri1Out = 0; Ri2Out = 0; Ri3Out = 0; Ri4Out = 0;
                memEn = 0; memOp = 0; MDRout = 0;
                finish = 0;
             end
   two:   begin
                MDRwrite = 1;
                case (p1)
                        6'b000001: begin Ri1Out = 1; Ri2Out = 0; Ri3Out = 0; Ri4Out = 0; end
                        6'b000010: begin Ri2Out = 1; Ri1Out = 0; Ri3Out = 0; Ri4Out = 0; end
                        6'b000011: begin Ri3Out = 1; Ri2Out = 0; Ri1Out = 0; Ri4Out = 0; end
                        default  : begin Ri4Out = 1; Ri2Out = 0; Ri3Out = 0; Ri1Out = 0; end
                endcase
                PCinc = 0; MARin = 0;
                memOp = 0; MDRout = 0; memEn = 0;
                Rj1Out = 0; Rj2Out = 0; Rj3Out = 0; Rj4Out = 0;
                finish = 0;
           end
   three:  begin
                memEn = 0; MDRwrite = 0;
                case(p2)
                        6'b000001: begin Rj1Out = 0; Rj2Out = 0; Rj3Out = 0; Rj4Out = 0; end
                        6'b000010: begin Rj2Out = 0; Rj1Out = 0; Rj3Out = 0; Rj4Out = 0; end
                        6'b000011: begin Rj3Out = 0; Rj2Out = 0; Rj1Out = 0; Rj4Out = 0; end
                        default  : begin Rj4Out = 0; Rj2Out = 0; Rj3Out = 0; Rj1Out = 0; end
                endcase
                PCinc = 0; MARin = 0;
                memOp = 0; MDRout = 0;
                Ri1Out = 0; Ri2Out = 0; Ri3Out = 0; Ri4Out = 0;
                finish = 0;
                end
        four: begin
                PCinc = 0; MARin = 0; MDRwrite = 0;
                Ri1Out = 0; Ri2Out = 0; Ri3Out = 0; Ri4Out = 0;
                memEn = 1; memOp = 0; MDRout = 0;
                Rj1Out = 0; Rj2Out = 0; Rj3Out = 0; Rj4Out = 0;
                finish = 0;
            end
   five: begin
                memEn = 0; finish = 1;
                case (p1)
                        6'b000001: begin Ri1Out = 0; Ri2Out = 0; Ri3Out = 0; Ri4Out = 0; end
                        6'b000010: begin Ri2Out = 0; Ri1Out = 0; Ri3Out = 0; Ri4Out = 0; end
                        6'b000011: begin Ri3Out = 0; Ri2Out = 0; Ri1Out = 0; Ri4Out = 0; end
                        default  : begin Ri4Out = 0; Ri2Out = 0; Ri3Out = 0; Ri1Out = 0; end
                endcase
                PCinc = 0; MARin = 0; MDRwrite = 0;
                memEn = 0; memOp = 0; MDRout = 0;
                Rj1Out = 0; Rj2Out = 0; Rj3Out = 0; Rj4Out = 0;
         end
```

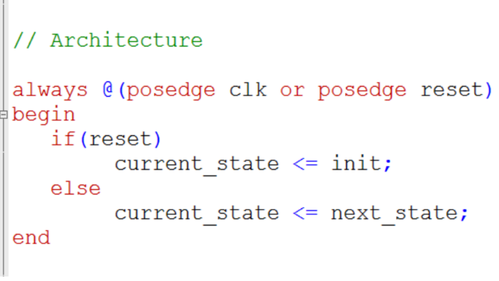


```
    six:  begin
                PCinc = 0; MARin = 0; MDRwrite = 0;
                Ri1Out = 0; Ri2Out = 0; Ri3Out = 0; Ri4Out = 0;
                memEn = 0; memOp = 0; MDRout = 0;
                Rj1Out = 0; Rj2Out = 0; Rj3Out = 0; Rj4Out = 0; finish = 0;
          end

    endcase

end

endmodule
```

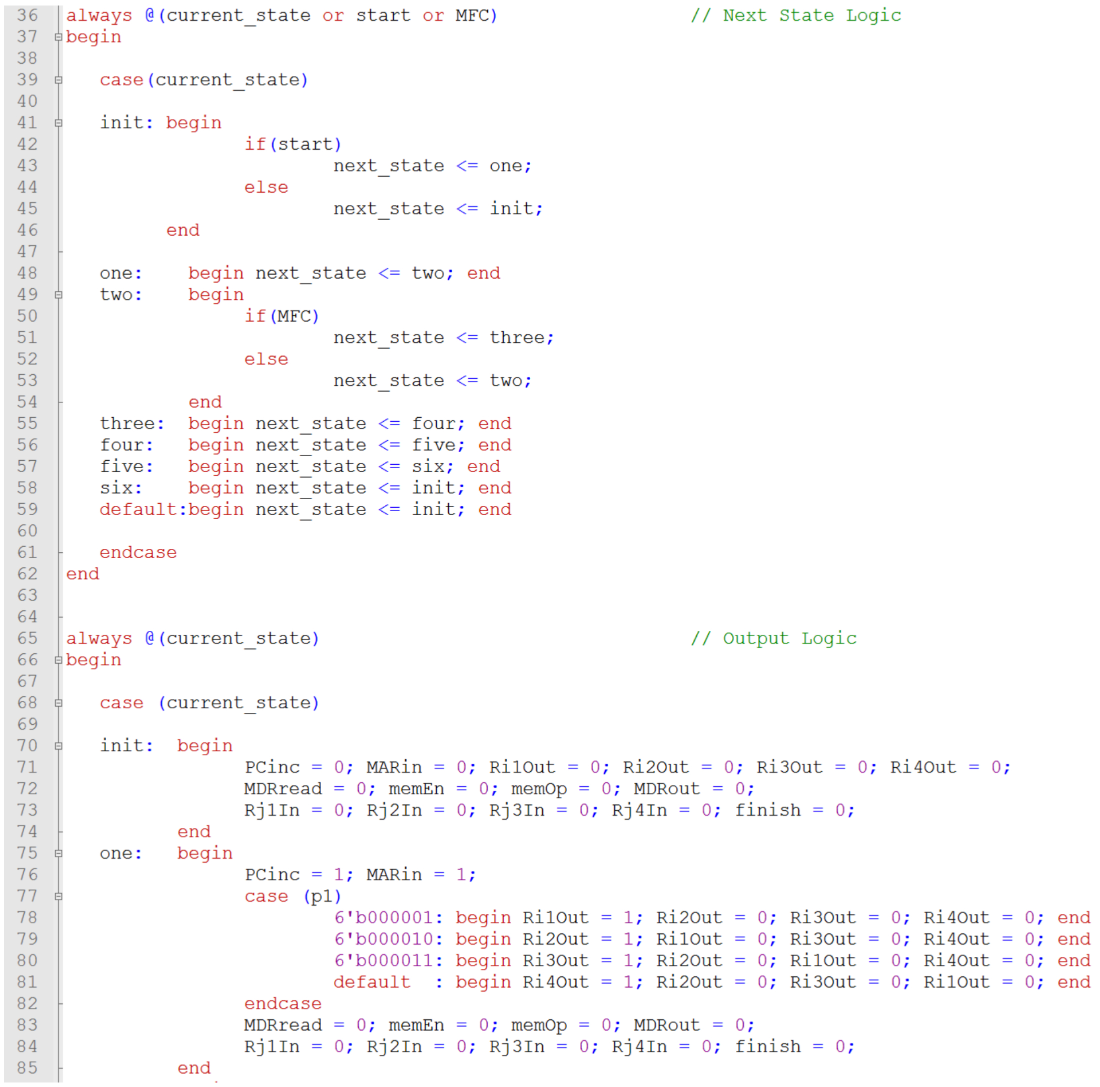
- Load FSM



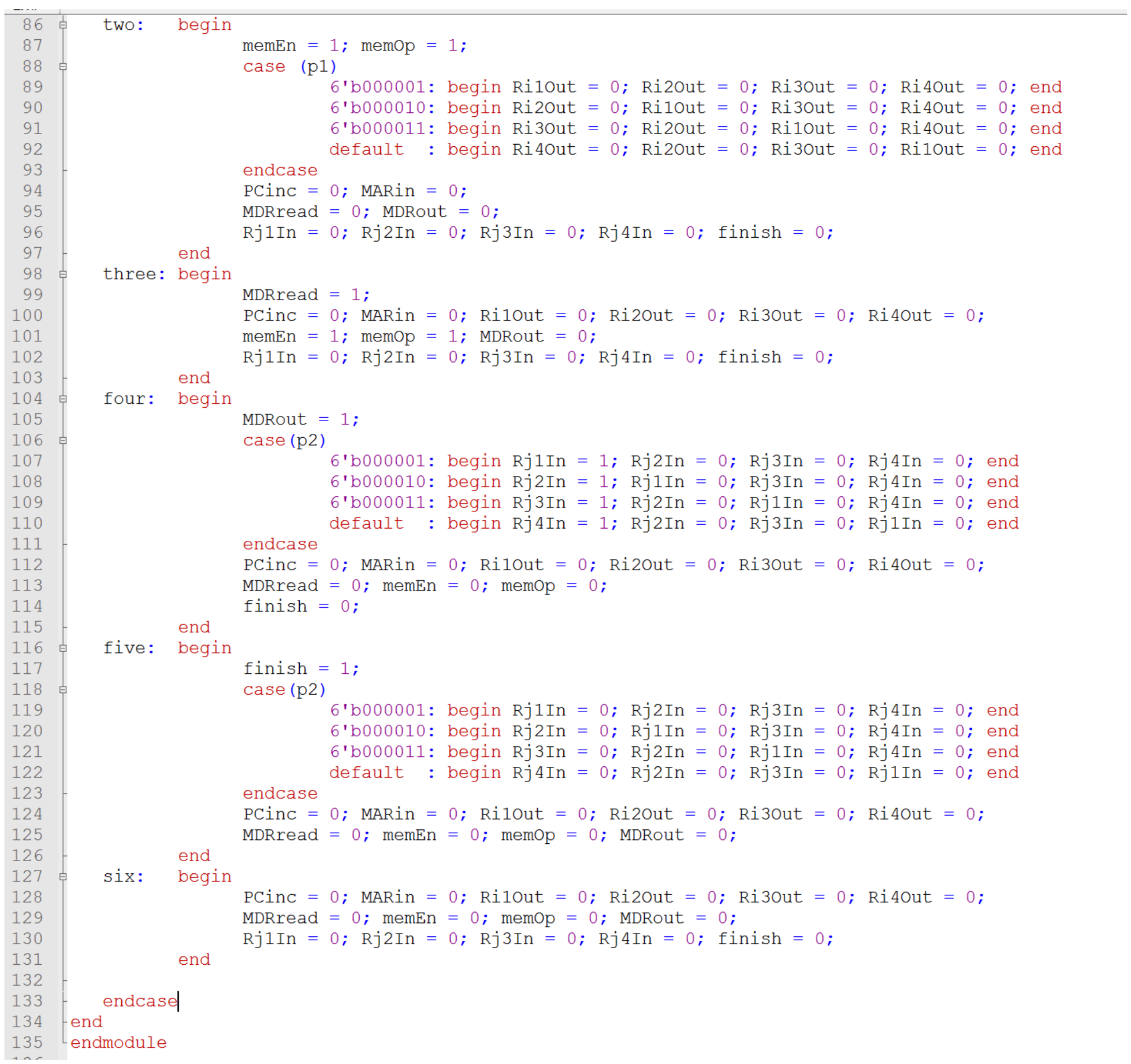
```
module Load_FSM(clk, reset, start, MFC, PCinc, Ri1Out, Ri2Out, Ri3Out, Ri4Out, MARin, MDRread, memEn, memOp, MDRout, Rj1In, Rj2

// Port Declartaion

input clk; input reset; input start; input MFC; input[5:0] p1; input[5:0] p2;

output reg PCinc; output reg MARin; output reg MDRread;
output reg Ri1Out; output reg Ri2Out; output reg Ri3Out; output reg Ri4Out;
output reg memEn; output reg memOp; output reg MDRout;
output reg Rj1In; output reg Rj2In; output reg Rj3In; output reg Rj4In; output reg finish;

reg[2:0] current_state, next_state;

// Set Parameters for State Signals

parameter init = 4'b0000, one = 4'b0001, two = 4'b0010, three = 4'b0011,
        four = 4'b0100, five = 4'b0101, six = 4'b0110,
        seven = 4'b0111, eight = 4'b1000, nine = 4'b1001, ten = 4'b1010;
```



```
// Architecture

always @(posedge clk or posedge reset)
begin
   if(reset)
        current_state <= init;
   else
        current_state <= next_state;
end
```



```
always @(current_state or start or MFC)                 // Next State Logic
begin

   case(current_state)

   init: begin
                 if(start)
                        next_state <= one;
                 else
                        next_state <= init;
         end

   one:     begin next_state <= two; end
   two:     begin
                 if(MFC)
                        next_state <= three;
                 else
                        next_state <= two;
            end
   three:   begin next_state <= four; end
   four:    begin next_state <= five; end
   five:    begin next_state <= six; end
   six:     begin next_state <= init; end
   default:begin next_state <= init; end

   endcase
end


always @(current_state)                                 // Output Logic
begin

   case (current_state)

   init:  begin
                PCinc = 0; MARin = 0; Ri1Out = 0; Ri2Out = 0; Ri3Out = 0; Ri4Out = 0;
                MDRread = 0; memEn = 0; memOp = 0; MDRout = 0;
                Rj1In = 0; Rj2In = 0; Rj3In = 0; Rj4In = 0; finish = 0;
          end
   one:   begin
                PCinc = 1; MARin = 1;
                case (p1)
                        6'b000001: begin Ri1Out = 1; Ri2Out = 0; Ri3Out = 0; Ri4Out = 0; end
                        6'b000010: begin Ri2Out = 1; Ri1Out = 0; Ri3Out = 0; Ri4Out = 0; end
                        6'b000011: begin Ri3Out = 1; Ri2Out = 0; Ri1Out = 0; Ri4Out = 0; end
                        default  : begin Ri4Out = 1; Ri2Out = 0; Ri3Out = 0; Ri1Out = 0; end
                endcase
                MDRread = 0; memEn = 0; memOp = 0; MDRout = 0;
                Rj1In = 0; Rj2In = 0; Rj3In = 0; Rj4In = 0; finish = 0;
          end
```



```
   two:   begin
                memEn = 1; memOp = 1;
                case (p1)
                        6'b000001: begin Ri1Out = 0; Ri2Out = 0; Ri3Out = 0; Ri4Out = 0; end
                        6'b000010: begin Ri2Out = 0; Ri1Out = 0; Ri3Out = 0; Ri4Out = 0; end
                        6'b000011: begin Ri3Out = 0; Ri2Out = 0; Ri1Out = 0; Ri4Out = 0; end
                        default  : begin Ri4Out = 0; Ri2Out = 0; Ri3Out = 0; Ri1Out = 0; end
                endcase
                PCinc = 0; MARin = 0;
                MDRread = 0; MDRout = 0;
                Rj1In = 0; Rj2In = 0; Rj3In = 0; Rj4In = 0; finish = 0;
          end
   three: begin
                MDRread = 1;
                PCinc = 0; MARin = 0; Ri1Out = 0; Ri2Out = 0; Ri3Out = 0; Ri4Out = 0;
                memEn = 1; memOp = 1; MDRout = 0;
                Rj1In = 0; Rj2In = 0; Rj3In = 0; Rj4In = 0; finish = 0;
          end
   four:  begin
                MDRout = 1;
                case(p2)
                        6'b000001: begin Rj1In = 1; Rj2In = 0; Rj3In = 0; Rj4In = 0; end
                        6'b000010: begin Rj2In = 1; Rj1In = 0; Rj3In = 0; Rj4In = 0; end
                        6'b000011: begin Rj3In = 1; Rj2In = 0; Rj1In = 0; Rj4In = 0; end
                        default  : begin Rj4In = 1; Rj2In = 0; Rj3In = 0; Rj1In = 0; end
                endcase
                PCinc = 0; MARin = 0; Ri1Out = 0; Ri2Out = 0; Ri3Out = 0; Ri4Out = 0;
                MDRread = 0; memEn = 0; memOp = 0;
                finish = 0;
          end
   five:  begin
                finish = 1;
                case(p2)
                        6'b000001: begin Rj1In = 0; Rj2In = 0; Rj3In = 0; Rj4In = 0; end
                        6'b000010: begin Rj2In = 0; Rj1In = 0; Rj3In = 0; Rj4In = 0; end
                        6'b000011: begin Rj3In = 0; Rj2In = 0; Rj1In = 0; Rj4In = 0; end
                        default  : begin Rj4In = 0; Rj2In = 0; Rj3In = 0; Rj1In = 0; end
                endcase
                PCinc = 0; MARin = 0; Ri1Out = 0; Ri2Out = 0; Ri3Out = 0; Ri4Out = 0;
                MDRread = 0; memEn = 0; memOp = 0; MDRout = 0;
          end
   six:   begin
                PCinc = 0; MARin = 0; Ri1Out = 0; Ri2Out = 0; Ri3Out = 0; Ri4Out = 0;
                MDRread = 0; memEn = 0; memOp = 0; MDRout = 0;
                Rj1In = 0; Rj2In = 0; Rj3In = 0; Rj4In = 0; finish = 0;
          end

   endcase
end
endmodule
```

Modules (Post-Synthesis Netlist)

- Program Counter

```
module programCounter_DW01_inc_0 ( A, SUM );
  input [15:0] A;
  output [15:0] SUM;

  wire   [15:2] carry;

  HADDX1 U1_1_14 ( .A0(A[14]), .B0(carry[14]), .C1(carry[15]), .SO(SUM[14]) );
  HADDX1 U1_1_13 ( .A0(A[13]), .B0(carry[13]), .C1(carry[14]), .SO(SUM[13]) );
  HADDX1 U1_1_12 ( .A0(A[12]), .B0(carry[12]), .C1(carry[13]), .SO(SUM[12]) );
  HADDX1 U1_1_11 ( .A0(A[11]), .B0(carry[11]), .C1(carry[12]), .SO(SUM[11]) );
  HADDX1 U1_1_10 ( .A0(A[10]), .B0(carry[10]), .C1(carry[11]), .SO(SUM[10]) );
  HADDX1 U1_1_9 ( .A0(A[9]), .B0(carry[9]), .C1(carry[10]), .SO(SUM[9]) );
  HADDX1 U1_1_8 ( .A0(A[8]), .B0(carry[8]), .C1(carry[9]), .SO(SUM[8]) );
  HADDX1 U1_1_7 ( .A0(A[7]), .B0(carry[7]), .C1(carry[8]), .SO(SUM[7]) );
  HADDX1 U1_1_6 ( .A0(A[6]), .B0(carry[6]), .C1(carry[7]), .SO(SUM[6]) );
  HADDX1 U1_1_5 ( .A0(A[5]), .B0(carry[5]), .C1(carry[6]), .SO(SUM[5]) );
  HADDX1 U1_1_4 ( .A0(A[4]), .B0(carry[4]), .C1(carry[5]), .SO(SUM[4]) );
  HADDX1 U1_1_3 ( .A0(A[3]), .B0(carry[3]), .C1(carry[4]), .SO(SUM[3]) );
  HADDX1 U1_1_2 ( .A0(A[2]), .B0(carry[2]), .C1(carry[3]), .SO(SUM[2]) );
  HADDX1 U1_1_1 ( .A0(A[1]), .B0(A[0]), .C1(carry[2]), .SO(SUM[1]) );
  INVX0 U1 ( .IN(A[0]), .QN(SUM[0]) );
  XOR2X1 U2 ( .IN1(carry[15]), .IN2(A[15]), .Q(SUM[15]) );
endmodule


module programCounter ( clk, reset, incriment, oTriEn, PC_out );
  output [15:0] PC_out;
  input clk, reset, incriment, oTriEn;
  wire   N18, N19, N20, N21, N22, N23, N24, N25, N26, N27, N28, N29, N30, N31,
         N32, N33, n66, n67, n68, n69, n70, n71, n72, n73, n74, n75, n76, n77,
         n78, n79, n80, n81, n117, n118;
  wire   [15:0] counter;
  tri   \PC_out[15] ;
  tri   \PC_out[14] ;
  tri   \PC_out[13] ;
  tri   \PC_out[12] ;
  tri   \PC_out[11] ;
  tri   \PC_out[10] ;
  tri   \PC_out[9] ;
  tri   \PC_out[8] ;
  tri   \PC_out[7] ;
  tri   \PC_out[6] ;
  tri   \PC_out[5] ;
  tri   \PC_out[4] ;
  tri   \PC_out[3] ;
  tri   \PC_out[2] ;
  tri   \PC_out[1] ;
  tri   \PC_out[0] ;

  TNBUFFHX1 \PC_out_tri[0]  ( .IN(counter[0]), .ENB(oTriEn), .Q(PC_out[0]) );
  TNBUFFHX1 \PC_out_tri[1]  ( .IN(counter[1]), .ENB(oTriEn), .Q(PC_out[1]) );
  TNBUFFHX1 \PC_out_tri[2]  ( .IN(counter[2]), .ENB(oTriEn), .Q(PC_out[2]) );
  TNBUFFHX1 \PC_out_tri[3]  ( .IN(counter[3]), .ENB(oTriEn), .Q(PC_out[3]) );
  TNBUFFHX1 \PC_out_tri[4]  ( .IN(counter[4]), .ENB(oTriEn), .Q(PC_out[4]) );
  TNBUFFHX1 \PC_out_tri[5]  ( .IN(counter[5]), .ENB(oTriEn), .Q(PC_out[5]) );
  TNBUFFHX1 \PC_out_tri[6]  ( .IN(counter[6]), .ENB(oTriEn), .Q(PC_out[6]) );
  TNBUFFHX1 \PC_out_tri[7]  ( .IN(counter[7]), .ENB(oTriEn), .Q(PC_out[7]) );
  TNBUFFHX1 \PC_out_tri[8]  ( .IN(counter[8]), .ENB(oTriEn), .Q(PC_out[8]) );
  TNBUFFHX1 \PC_out_tri[9]  ( .IN(counter[9]), .ENB(oTriEn), .Q(PC_out[9]) );
  TNBUFFHX1 \PC_out_tri[10]  ( .IN(counter[10]), .ENB(oTriEn), .Q(PC_out[10])
        );
  TNBUFFHX1 \PC_out_tri[11]  ( .IN(counter[11]), .ENB(oTriEn), .Q(PC_out[11])
        );
  TNBUFFHX1 \PC_out_tri[12]  ( .IN(counter[12]), .ENB(oTriEn), .Q(PC_out[12])
        );
  TNBUFFHX1 \PC_out_tri[13]  ( .IN(counter[13]), .ENB(oTriEn), .Q(PC_out[13])
        );
  TNBUFFHX1 \PC_out_tri[14]  ( .IN(counter[14]), .ENB(oTriEn), .Q(PC_out[14])
        );
  TNBUFFHX1 \PC_out_tri[15]  ( .IN(counter[15]), .ENB(oTriEn), .Q(PC_out[15])
        );
  programCounter_DW01_inc_0 add_21 ( .A(counter), .SUM({N33, N32, N31, N30,
        N29, N28, N27, N26, N25, N24, N23, N22, N21, N20, N19, N18}) );
  DFFARX1 \counter_reg[15]  ( .D(n66), .CLK(clk), .RSTB(n117), .Q(counter[15])
        );
  DFFARX1 \counter_reg[6]  ( .D(n75), .CLK(clk), .RSTB(n117), .Q(counter[6])
        );
  DFFARX1 \counter_reg[5]  ( .D(n76), .CLK(clk), .RSTB(n117), .Q(counter[5])
        );
  DFFARX1 \counter_reg[4]  ( .D(n77), .CLK(clk), .RSTB(n117), .Q(counter[4])
        );
  DFFARX1 \counter_reg[3]  ( .D(n78), .CLK(clk), .RSTB(n117), .Q(counter[3])
        );
  DFFARX1 \counter_reg[2]  ( .D(n79), .CLK(clk), .RSTB(n117), .Q(counter[2])
        );
  DFFARX1 \counter_reg[1]  ( .D(n80), .CLK(clk), .RSTB(n117), .Q(counter[1])
        );
  DFFARX1 \counter_reg[7]  ( .D(n74), .CLK(clk), .RSTB(n117), .Q(counter[7])
        );
  DFFARX1 \counter_reg[8]  ( .D(n73), .CLK(clk), .RSTB(n117), .Q(counter[8])
        );
  DFFARX1 \counter_reg[9]  ( .D(n72), .CLK(clk), .RSTB(n117), .Q(counter[9])
        );
  DFFARX1 \counter_reg[10]  ( .D(n71), .CLK(clk), .RSTB(n117), .Q(counter[10])
        );
  DFFARX1 \counter_reg[11]  ( .D(n70), .CLK(clk), .RSTB(n117), .Q(counter[11])
        );
  DFFARX1 \counter_reg[12]  ( .D(n69), .CLK(clk), .RSTB(n117), .Q(counter[12])
        );
  DFFARX1 \counter_reg[13]  ( .D(n68), .CLK(clk), .RSTB(n117), .Q(counter[13])
        );
  DFFARX1 \counter_reg[14]  ( .D(n67), .CLK(clk), .RSTB(n117), .Q(counter[14])
        );
  DFFARX1 \counter_reg[0]  ( .D(n81), .CLK(clk), .RSTB(n117), .Q(counter[0])
        );
  AO22X1 U55 ( .IN1(n118), .IN2(counter[0]), .IN3(incriment), .IN4(N18), .Q(
        n81) );
  AO22X1 U56 ( .IN1(n118), .IN2(counter[15]), .IN3(N33), .IN4(incriment), .Q(
        n66) );
  AO22X1 U57 ( .IN1(n118), .IN2(counter[14]), .IN3(N32), .IN4(incriment), .Q(
        n67) );
  AO22X1 U58 ( .IN1(n118), .IN2(counter[13]), .IN3(N31), .IN4(incriment), .Q(
        n68) );
  AO22X1 U59 ( .IN1(n118), .IN2(counter[12]), .IN3(N30), .IN4(incriment), .Q(
        n69) );
  AO22X1 U60 ( .IN1(n118), .IN2(counter[11]), .IN3(N29), .IN4(incriment), .Q(
        n70) );
  AO22X1 U61 ( .IN1(n118), .IN2(counter[10]), .IN3(N28), .IN4(incriment), .Q(
        n71) );
  AO22X1 U62 ( .IN1(n118), .IN2(counter[9]), .IN3(N27), .IN4(incriment), .Q(
        n72) );
  AO22X1 U63 ( .IN1(n118), .IN2(counter[8]), .IN3(N26), .IN4(incriment), .Q(
        n73) );
  AO22X1 U64 ( .IN1(n118), .IN2(counter[7]), .IN3(N25), .IN4(incriment), .Q(
        n74) );
  AO22X1 U65 ( .IN1(n118), .IN2(counter[1]), .IN3(N19), .IN4(incriment), .Q(
        n80) );
  AO22X1 U66 ( .IN1(n118), .IN2(counter[2]), .IN3(N20), .IN4(incriment), .Q(
        n79) );
  AO22X1 U67 ( .IN1(n118), .IN2(counter[3]), .IN3(N21), .IN4(incriment), .Q(
        n78) );
  AO22X1 U68 ( .IN1(n118), .IN2(counter[4]), .IN3(N22), .IN4(incriment), .Q(
        n77) );
  AO22X1 U69 ( .IN1(n118), .IN2(counter[5]), .IN3(N23), .IN4(incriment), .Q(
        n76) );
  AO22X1 U70 ( .IN1(n118), .IN2(counter[6]), .IN3(N24), .IN4(incriment), .Q(
        n75) );
  INVX2 U71 ( .IN(incriment), .QN(n118) );
  INVX2 U72 ( .IN(reset), .QN(n117) );
endmodule
```

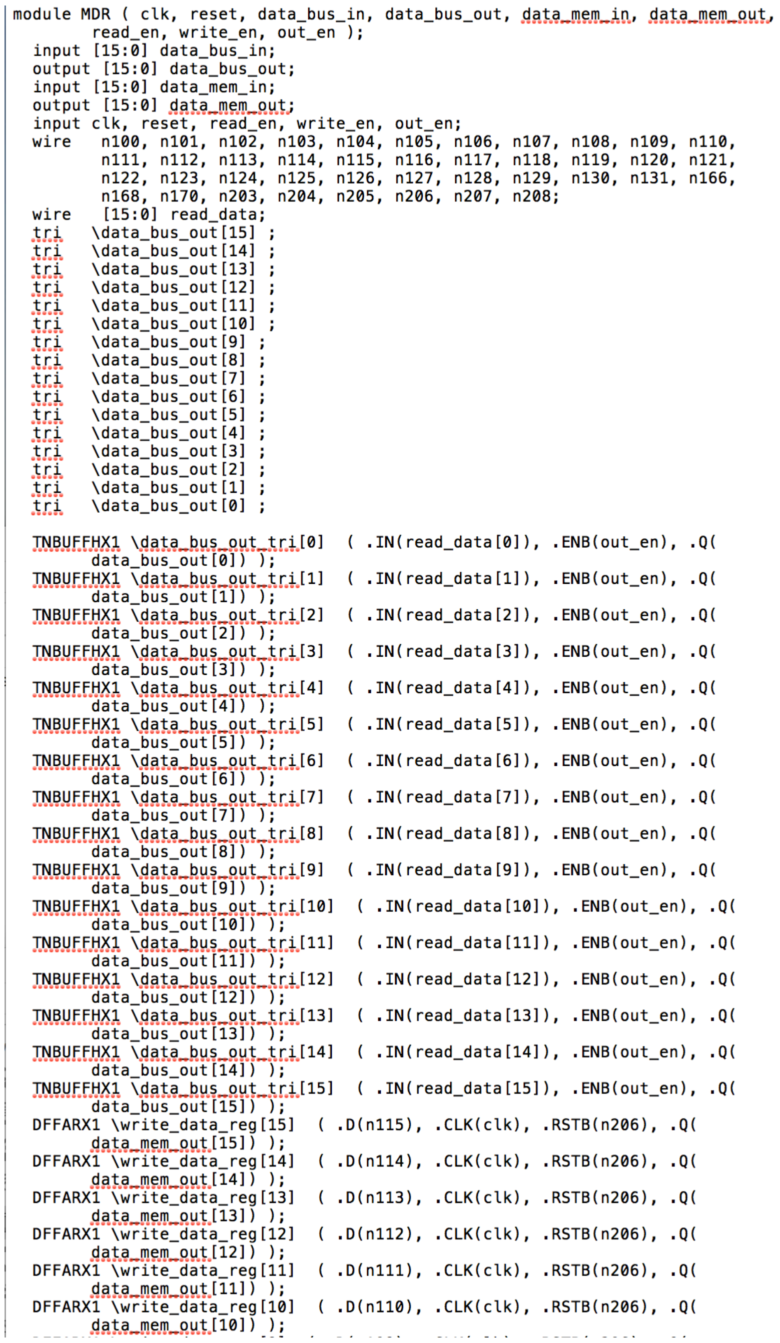
- MAR / IR

```
module MAR ( clk, reset, data_in, enI, data_out );
  input [15:0] data_in;
  output [15:0] data_out;
  input clk, reset, enI;
  wire   n34, n35, n36, n37, n38, n39, n40, n41, n42, n43, n44, n45, n46, n47,
         n48, n49, n66, n67;

  DFFARX1 \data_reg[15]  ( .D(n49), .CLK(clk), .RSTB(n66), .Q(data_out[15]) );
  DFFARX1 \data_reg[14]  ( .D(n48), .CLK(clk), .RSTB(n66), .Q(data_out[14]) );
  DFFARX1 \data_reg[13]  ( .D(n47), .CLK(clk), .RSTB(n66), .Q(data_out[13]) );
  DFFARX1 \data_reg[12]  ( .D(n46), .CLK(clk), .RSTB(n66), .Q(data_out[12]) );
  DFFARX1 \data_reg[11]  ( .D(n45), .CLK(clk), .RSTB(n66), .Q(data_out[11]) );
  DFFARX1 \data_reg[10]  ( .D(n44), .CLK(clk), .RSTB(n66), .Q(data_out[10]) );
  DFFARX1 \data_reg[9]  ( .D(n43), .CLK(clk), .RSTB(n66), .Q(data_out[9]) );
  DFFARX1 \data_reg[8]  ( .D(n42), .CLK(clk), .RSTB(n66), .Q(data_out[8]) );
  DFFARX1 \data_reg[7]  ( .D(n41), .CLK(clk), .RSTB(n66), .Q(data_out[7]) );
  DFFARX1 \data_reg[6]  ( .D(n40), .CLK(clk), .RSTB(n66), .Q(data_out[6]) );
  DFFARX1 \data_reg[5]  ( .D(n39), .CLK(clk), .RSTB(n66), .Q(data_out[5]) );
  DFFARX1 \data_reg[4]  ( .D(n38), .CLK(clk), .RSTB(n66), .Q(data_out[4]) );
  DFFARX1 \data_reg[3]  ( .D(n37), .CLK(clk), .RSTB(n66), .Q(data_out[3]) );
  DFFARX1 \data_reg[2]  ( .D(n36), .CLK(clk), .RSTB(n66), .Q(data_out[2]) );
  DFFARX1 \data_reg[1]  ( .D(n35), .CLK(clk), .RSTB(n66), .Q(data_out[1]) );
  DFFARX1 \data_reg[0]  ( .D(n34), .CLK(clk), .RSTB(n66), .Q(data_out[0]) );
  AO22X1 U38 ( .IN1(n67), .IN2(data_out[0]), .IN3(data_in[0]), .IN4(enI), .Q(
        n34) );
  AO22X1 U39 ( .IN1(n67), .IN2(data_out[1]), .IN3(data_in[1]), .IN4(enI), .Q(
        n35) );
  AO22X1 U40 ( .IN1(n67), .IN2(data_out[2]), .IN3(data_in[2]), .IN4(enI), .Q(
        n36) );
  AO22X1 U41 ( .IN1(n67), .IN2(data_out[3]), .IN3(data_in[3]), .IN4(enI), .Q(
        n37) );
  AO22X1 U42 ( .IN1(n67), .IN2(data_out[4]), .IN3(data_in[4]), .IN4(enI), .Q(
        n38) );
  AO22X1 U43 ( .IN1(n67), .IN2(data_out[5]), .IN3(data_in[5]), .IN4(enI), .Q(
        n39) );
  AO22X1 U44 ( .IN1(n67), .IN2(data_out[6]), .IN3(data_in[6]), .IN4(enI), .Q(
        n40) );
  AO22X1 U45 ( .IN1(n67), .IN2(data_out[7]), .IN3(data_in[7]), .IN4(enI), .Q(
        n41) );
  AO22X1 U46 ( .IN1(n67), .IN2(data_out[8]), .IN3(data_in[8]), .IN4(enI), .Q(
        n42) );
  AO22X1 U47 ( .IN1(n67), .IN2(data_out[9]), .IN3(data_in[9]), .IN4(enI), .Q(
        n43) );
  AO22X1 U48 ( .IN1(n67), .IN2(data_out[10]), .IN3(data_in[10]), .IN4(enI),
        .Q(n44) );
  AO22X1 U49 ( .IN1(n67), .IN2(data_out[11]), .IN3(data_in[11]), .IN4(enI),
        .Q(n45) );
  AO22X1 U50 ( .IN1(n67), .IN2(data_out[12]), .IN3(data_in[12]), .IN4(enI),
        .Q(n46) );
  AO22X1 U51 ( .IN1(n67), .IN2(data_out[13]), .IN3(data_in[13]), .IN4(enI),
        .Q(n47) );
  AO22X1 U52 ( .IN1(n67), .IN2(data_out[14]), .IN3(data_in[14]), .IN4(enI),
        .Q(n48) );
  AO22X1 U53 ( .IN1(n67), .IN2(data_out[15]), .IN3(enI), .IN4(data_in[15]),
        .Q(n49) );
  INVX2 U54 ( .IN(enI), .QN(n67) );
  INVX2 U55 ( .IN(reset), .QN(n66) );
endmodule
```

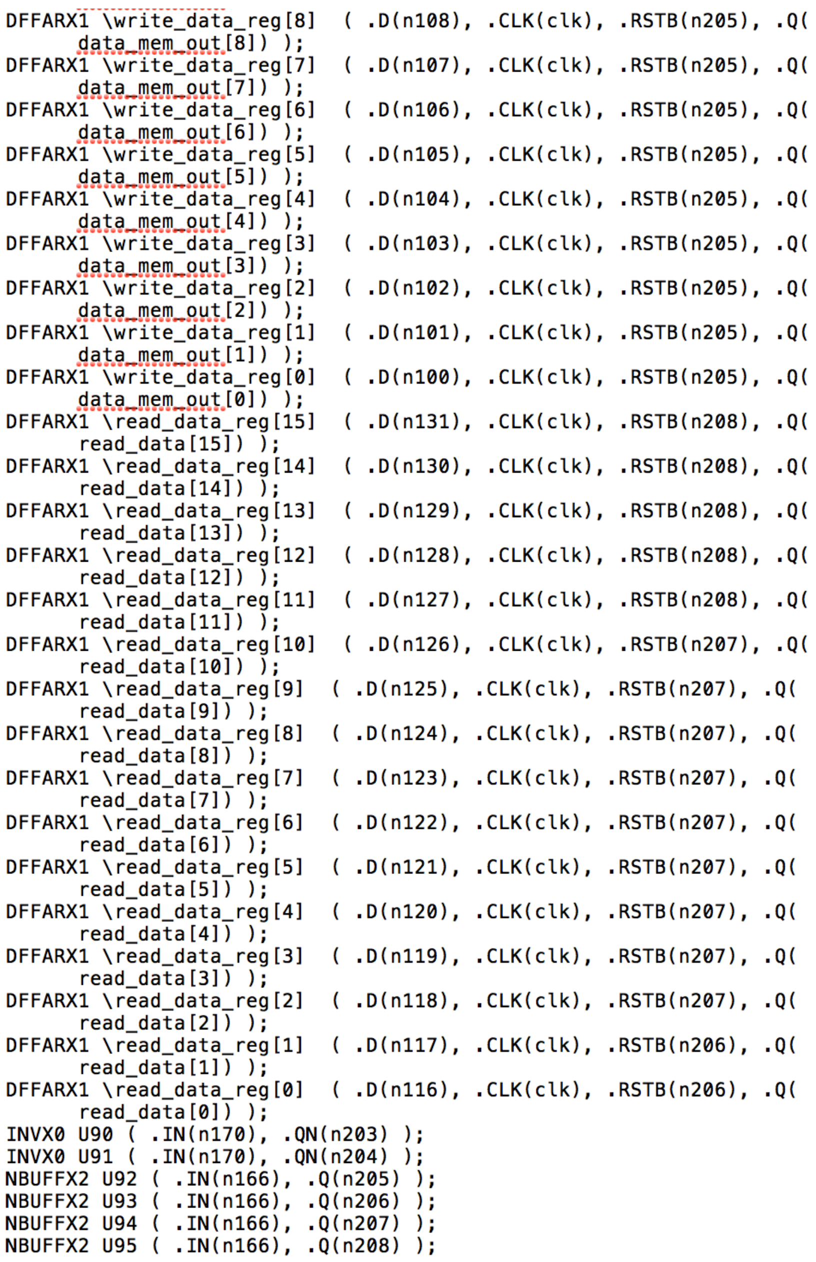
- GPR

```
module GPR ( clk, reset, bus_in, inEn, outEn, bus_out );
  input [15:0] bus_in;
  output [15:0] bus_out;
  input clk, reset, inEn, outEn;
  wire   n66, n67, n68, n69, n70, n71, n72, n73, n74, n75, n76, n77, n78, n79,
         n80, n81, n82, n83, n84, n85, n86, n87, n88, n89, n90, n91, n92, n93,
         n94, n95, n96, n97, n114, n115;
  tri   \bus_out[15] ;
  tri   \bus_out[14] ;
  tri   \bus_out[13] ;
  tri   \bus_out[12] ;
  tri   \bus_out[11] ;
  tri   \bus_out[10] ;
  tri   \bus_out[9] ;
  tri   \bus_out[8] ;
  tri   \bus_out[7] ;
  tri   \bus_out[6] ;
  tri   \bus_out[5] ;
  tri   \bus_out[4] ;
  tri   \bus_out[3] ;
  tri   \bus_out[2] ;
  tri   \bus_out[1] ;
  tri   \bus_out[0] ;

  TNBUFFHX1 \bus_out_tri[0]  ( .IN(n97), .ENB(outEn), .Q(bus_out[0]) );
  TNBUFFHX1 \bus_out_tri[1]  ( .IN(n96), .ENB(outEn), .Q(bus_out[1]) );
  TNBUFFHX1 \bus_out_tri[2]  ( .IN(n95), .ENB(outEn), .Q(bus_out[2]) );
  TNBUFFHX1 \bus_out_tri[3]  ( .IN(n94), .ENB(outEn), .Q(bus_out[3]) );
  TNBUFFHX1 \bus_out_tri[4]  ( .IN(n93), .ENB(outEn), .Q(bus_out[4]) );
  TNBUFFHX1 \bus_out_tri[5]  ( .IN(n92), .ENB(outEn), .Q(bus_out[5]) );
  TNBUFFHX1 \bus_out_tri[6]  ( .IN(n91), .ENB(outEn), .Q(bus_out[6]) );
  TNBUFFHX1 \bus_out_tri[7]  ( .IN(n90), .ENB(outEn), .Q(bus_out[7]) );
  TNBUFFHX1 \bus_out_tri[8]  ( .IN(n89), .ENB(outEn), .Q(bus_out[8]) );
  TNBUFFHX1 \bus_out_tri[9]  ( .IN(n88), .ENB(outEn), .Q(bus_out[9]) );
  TNBUFFHX1 \bus_out_tri[10]  ( .IN(n87), .ENB(outEn), .Q(bus_out[10]) );
  TNBUFFHX1 \bus_out_tri[11]  ( .IN(n86), .ENB(outEn), .Q(bus_out[11]) );
  TNBUFFHX1 \bus_out_tri[12]  ( .IN(n85), .ENB(outEn), .Q(bus_out[12]) );
  TNBUFFHX1 \bus_out_tri[13]  ( .IN(n84), .ENB(outEn), .Q(bus_out[13]) );
  TNBUFFHX1 \bus_out_tri[14]  ( .IN(n83), .ENB(outEn), .Q(bus_out[14]) );
  TNBUFFHX1 \bus_out_tri[15]  ( .IN(n82), .ENB(outEn), .Q(bus_out[15]) );
  DFFARX1 \data_reg[15]  ( .D(n81), .CLK(clk), .RSTB(n114), .Q(n82) );
  DFFARX1 \data_reg[14]  ( .D(n80), .CLK(clk), .RSTB(n114), .Q(n83) );
  DFFARX1 \data_reg[13]  ( .D(n79), .CLK(clk), .RSTB(n114), .Q(n84) );
  DFFARX1 \data_reg[12]  ( .D(n78), .CLK(clk), .RSTB(n114), .Q(n85) );
  DFFARX1 \data_reg[11]  ( .D(n77), .CLK(clk), .RSTB(n114), .Q(n86) );
  DFFARX1 \data_reg[10]  ( .D(n76), .CLK(clk), .RSTB(n114), .Q(n87) );
  DFFARX1 \data_reg[9]  ( .D(n75), .CLK(clk), .RSTB(n114), .Q(n88) );
  DFFARX1 \data_reg[8]  ( .D(n74), .CLK(clk), .RSTB(n114), .Q(n89) );
  DFFARX1 \data_reg[7]  ( .D(n73), .CLK(clk), .RSTB(n114), .Q(n90) );
  DFFARX1 \data_reg[6]  ( .D(n72), .CLK(clk), .RSTB(n114), .Q(n91) );
  DFFARX1 \data_reg[5]  ( .D(n71), .CLK(clk), .RSTB(n114), .Q(n92) );
  DFFARX1 \data_reg[4]  ( .D(n70), .CLK(clk), .RSTB(n114), .Q(n93) );
  DFFARX1 \data_reg[3]  ( .D(n69), .CLK(clk), .RSTB(n114), .Q(n94) );
  DFFARX1 \data_reg[2]  ( .D(n68), .CLK(clk), .RSTB(n114), .Q(n95) );
  DFFARX1 \data_reg[1]  ( .D(n67), .CLK(clk), .RSTB(n114), .Q(n96) );
  DFFARX1 \data_reg[0]  ( .D(n66), .CLK(clk), .RSTB(n114), .Q(n97) );
  AO22X1 U54 ( .IN1(n97), .IN2(n115), .IN3(bus_in[0]), .IN4(inEn), .Q(n66) );
  AO22X1 U55 ( .IN1(n96), .IN2(n115), .IN3(bus_in[1]), .IN4(inEn), .Q(n67) );
  AO22X1 U56 ( .IN1(n95), .IN2(n115), .IN3(bus_in[2]), .IN4(inEn), .Q(n68) );
  AO22X1 U57 ( .IN1(n94), .IN2(n115), .IN3(bus_in[3]), .IN4(inEn), .Q(n69) );
  AO22X1 U58 ( .IN1(n93), .IN2(n115), .IN3(bus_in[4]), .IN4(inEn), .Q(n70) );
  AO22X1 U59 ( .IN1(n92), .IN2(n115), .IN3(bus_in[5]), .IN4(inEn), .Q(n71) );
  AO22X1 U60 ( .IN1(n91), .IN2(n115), .IN3(bus_in[6]), .IN4(inEn), .Q(n72) );
  AO22X1 U61 ( .IN1(n90), .IN2(n115), .IN3(bus_in[7]), .IN4(inEn), .Q(n73) );
  AO22X1 U62 ( .IN1(n89), .IN2(n115), .IN3(bus_in[8]), .IN4(inEn), .Q(n74) );
  AO22X1 U63 ( .IN1(n88), .IN2(n115), .IN3(bus_in[9]), .IN4(inEn), .Q(n75) );
  AO22X1 U64 ( .IN1(n87), .IN2(n115), .IN3(bus_in[10]), .IN4(inEn), .Q(n76) );
  AO22X1 U65 ( .IN1(n86), .IN2(n115), .IN3(bus_in[11]), .IN4(inEn), .Q(n77) );
  AO22X1 U66 ( .IN1(n85), .IN2(n115), .IN3(bus_in[12]), .IN4(inEn), .Q(n78) );
  AO22X1 U67 ( .IN1(n84), .IN2(n115), .IN3(bus_in[13]), .IN4(inEn), .Q(n79) );
  AO22X1 U68 ( .IN1(n83), .IN2(n115), .IN3(bus_in[14]), .IN4(inEn), .Q(n80) );
  AO22X1 U69 ( .IN1(n82), .IN2(n115), .IN3(inEn), .IN4(bus_in[15]), .Q(n81) );
  INVX2 U70 ( .IN(inEn), .QN(n115) );
  INVX2 U71 ( .IN(reset), .QN(n114) );
endmodule
```

- MDR



```
module MDR ( clk, reset, data_bus_in, data_bus_out, data_mem_in, data_mem_out,
        read_en, write_en, out_en );
  input [15:0] data_bus_in;
  output [15:0] data_bus_out;
  input [15:0] data_mem_in;
  output [15:0] data_mem_out;
  input clk, reset, read_en, write_en, out_en;
  wire   n100, n101, n102, n103, n104, n105, n106, n107, n108, n109, n110,
         n111, n112, n113, n114, n115, n116, n117, n118, n119, n120, n121,
         n122, n123, n124, n125, n126, n127, n128, n129, n130, n131, n166,
         n168, n170, n203, n204, n205, n206, n207, n208;
  wire   [15:0] read_data;
  tri   \data_bus_out[15] ;
  tri   \data_bus_out[14] ;
  tri   \data_bus_out[13] ;
  tri   \data_bus_out[12] ;
  tri   \data_bus_out[11] ;
  tri   \data_bus_out[10] ;
  tri   \data_bus_out[9] ;
  tri   \data_bus_out[8] ;
  tri   \data_bus_out[7] ;
  tri   \data_bus_out[6] ;
  tri   \data_bus_out[5] ;
  tri   \data_bus_out[4] ;
  tri   \data_bus_out[3] ;
  tri   \data_bus_out[2] ;
  tri   \data_bus_out[1] ;
  tri   \data_bus_out[0] ;

  TNBUFFHX1 \data_bus_out_tri[0]  ( .IN(read_data[0]), .ENB(out_en), .Q(
        data_bus_out[0]) );
  TNBUFFHX1 \data_bus_out_tri[1]  ( .IN(read_data[1]), .ENB(out_en), .Q(
        data_bus_out[1]) );
  TNBUFFHX1 \data_bus_out_tri[2]  ( .IN(read_data[2]), .ENB(out_en), .Q(
        data_bus_out[2]) );
  TNBUFFHX1 \data_bus_out_tri[3]  ( .IN(read_data[3]), .ENB(out_en), .Q(
        data_bus_out[3]) );
  TNBUFFHX1 \data_bus_out_tri[4]  ( .IN(read_data[4]), .ENB(out_en), .Q(
        data_bus_out[4]) );
  TNBUFFHX1 \data_bus_out_tri[5]  ( .IN(read_data[5]), .ENB(out_en), .Q(
        data_bus_out[5]) );
  TNBUFFHX1 \data_bus_out_tri[6]  ( .IN(read_data[6]), .ENB(out_en), .Q(
        data_bus_out[6]) );
  TNBUFFHX1 \data_bus_out_tri[7]  ( .IN(read_data[7]), .ENB(out_en), .Q(
        data_bus_out[7]) );
  TNBUFFHX1 \data_bus_out_tri[8]  ( .IN(read_data[8]), .ENB(out_en), .Q(
        data_bus_out[8]) );
  TNBUFFHX1 \data_bus_out_tri[9]  ( .IN(read_data[9]), .ENB(out_en), .Q(
        data_bus_out[9]) );
  TNBUFFHX1 \data_bus_out_tri[10]  ( .IN(read_data[10]), .ENB(out_en), .Q(
        data_bus_out[10]) );
  TNBUFFHX1 \data_bus_out_tri[11]  ( .IN(read_data[11]), .ENB(out_en), .Q(
        data_bus_out[11]) );
  TNBUFFHX1 \data_bus_out_tri[12]  ( .IN(read_data[12]), .ENB(out_en), .Q(
        data_bus_out[12]) );
  TNBUFFHX1 \data_bus_out_tri[13]  ( .IN(read_data[13]), .ENB(out_en), .Q(
        data_bus_out[13]) );
  TNBUFFHX1 \data_bus_out_tri[14]  ( .IN(read_data[14]), .ENB(out_en), .Q(
        data_bus_out[14]) );
  TNBUFFHX1 \data_bus_out_tri[15]  ( .IN(read_data[15]), .ENB(out_en), .Q(
        data_bus_out[15]) );
  DFFARX1 \write_data_reg[15]  ( .D(n115), .CLK(clk), .RSTB(n206), .Q(
        data_mem_out[15]) );
  DFFARX1 \write_data_reg[14]  ( .D(n114), .CLK(clk), .RSTB(n206), .Q(
        data_mem_out[14]) );
  DFFARX1 \write_data_reg[13]  ( .D(n113), .CLK(clk), .RSTB(n206), .Q(
        data_mem_out[13]) );
  DFFARX1 \write_data_reg[12]  ( .D(n112), .CLK(clk), .RSTB(n206), .Q(
        data_mem_out[12]) );
  DFFARX1 \write_data_reg[11]  ( .D(n111), .CLK(clk), .RSTB(n206), .Q(
        data_mem_out[11]) );
  DFFARX1 \write_data_reg[10]  ( .D(n110), .CLK(clk), .RSTB(n206), .Q(
        data_mem_out[10]) );
```



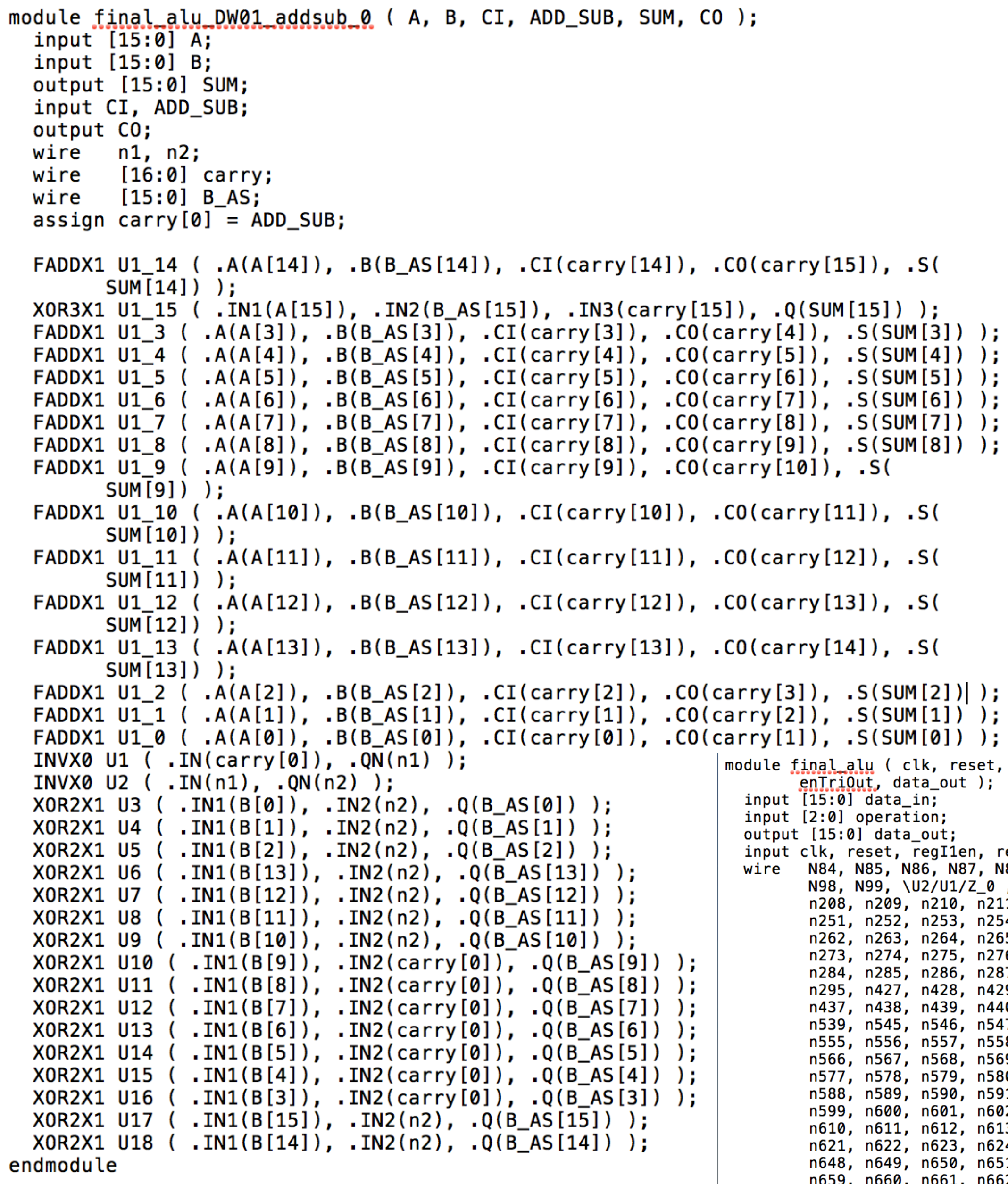
```
DFFARX1 \write_data_reg[8]  ( .D(n108), .CLK(clk), .RSTB(n205), .Q(
      data_mem_out[8]) );
DFFARX1 \write_data_reg[7]  ( .D(n107), .CLK(clk), .RSTB(n205), .Q(
      data_mem_out[7]) );
DFFARX1 \write_data_reg[6]  ( .D(n106), .CLK(clk), .RSTB(n205), .Q(
      data_mem_out[6]) );
DFFARX1 \write_data_reg[5]  ( .D(n105), .CLK(clk), .RSTB(n205), .Q(
      data_mem_out[5]) );
DFFARX1 \write_data_reg[4]  ( .D(n104), .CLK(clk), .RSTB(n205), .Q(
      data_mem_out[4]) );
DFFARX1 \write_data_reg[3]  ( .D(n103), .CLK(clk), .RSTB(n205), .Q(
      data_mem_out[3]) );
DFFARX1 \write_data_reg[2]  ( .D(n102), .CLK(clk), .RSTB(n205), .Q(
      data_mem_out[2]) );
DFFARX1 \write_data_reg[1]  ( .D(n101), .CLK(clk), .RSTB(n205), .Q(
      data_mem_out[1]) );
DFFARX1 \write_data_reg[0]  ( .D(n100), .CLK(clk), .RSTB(n205), .Q(
      data_mem_out[0]) );
DFFARX1 \read_data_reg[15]  ( .D(n131), .CLK(clk), .RSTB(n208), .Q(
      read_data[15]) );
DFFARX1 \read_data_reg[14]  ( .D(n130), .CLK(clk), .RSTB(n208), .Q(
      read_data[14]) );
DFFARX1 \read_data_reg[13]  ( .D(n129), .CLK(clk), .RSTB(n208), .Q(
      read_data[13]) );
DFFARX1 \read_data_reg[12]  ( .D(n128), .CLK(clk), .RSTB(n208), .Q(
      read_data[12]) );
DFFARX1 \read_data_reg[11]  ( .D(n127), .CLK(clk), .RSTB(n208), .Q(
      read_data[11]) );
DFFARX1 \read_data_reg[10]  ( .D(n126), .CLK(clk), .RSTB(n207), .Q(
      read_data[10]) );
DFFARX1 \read_data_reg[9]  ( .D(n125), .CLK(clk), .RSTB(n207), .Q(
      read_data[9]) );
DFFARX1 \read_data_reg[8]  ( .D(n124), .CLK(clk), .RSTB(n207), .Q(
      read_data[8]) );
DFFARX1 \read_data_reg[7]  ( .D(n123), .CLK(clk), .RSTB(n207), .Q(
      read_data[7]) );
DFFARX1 \read_data_reg[6]  ( .D(n122), .CLK(clk), .RSTB(n207), .Q(
      read_data[6]) );
DFFARX1 \read_data_reg[5]  ( .D(n121), .CLK(clk), .RSTB(n207), .Q(
      read_data[5]) );
DFFARX1 \read_data_reg[4]  ( .D(n120), .CLK(clk), .RSTB(n207), .Q(
      read_data[4]) );
DFFARX1 \read_data_reg[3]  ( .D(n119), .CLK(clk), .RSTB(n207), .Q(
      read_data[3]) );
DFFARX1 \read_data_reg[2]  ( .D(n118), .CLK(clk), .RSTB(n207), .Q(
      read_data[2]) );
DFFARX1 \read_data_reg[1]  ( .D(n117), .CLK(clk), .RSTB(n206), .Q(
      read_data[1]) );
DFFARX1 \read_data_reg[0]  ( .D(n116), .CLK(clk), .RSTB(n206), .Q(
      read_data[0]) );
INVX0 U90 ( .IN(n170), .QN(n203) );
INVX0 U91 ( .IN(n170), .QN(n204) );
NBUFFX2 U92 ( .IN(n166), .Q(n205) );
NBUFFX2 U93 ( .IN(n166), .Q(n206) );
NBUFFX2 U94 ( .IN(n166), .Q(n207) );
NBUFFX2 U95 ( .IN(n166), .Q(n208) );
```

- MDR (cont)

```
AO22X1 U96 ( .IN1(data_bus_in[0]), .IN2(n170), .IN3(n203), .IN4(
       data_mem_out[0]), .Q(n100) );
AO22X1 U97 ( .IN1(data_bus_in[1]), .IN2(n170), .IN3(n203), .IN4(
       data_mem_out[1]), .Q(n101) );
AO22X1 U98 ( .IN1(data_bus_in[2]), .IN2(n170), .IN3(n203), .IN4(
       data_mem_out[2]), .Q(n102) );
AO22X1 U99 ( .IN1(data_bus_in[3]), .IN2(n170), .IN3(n203), .IN4(
       data_mem_out[3]), .Q(n103) );
AO22X1 U100 ( .IN1(data_bus_in[4]), .IN2(n170), .IN3(n203), .IN4(
       data_mem_out[4]), .Q(n104) );
AO22X1 U101 ( .IN1(data_bus_in[5]), .IN2(n170), .IN3(n203), .IN4(
       data_mem_out[5]), .Q(n105) );
AO22X1 U102 ( .IN1(data_bus_in[6]), .IN2(n170), .IN3(n204), .IN4(
       data_mem_out[6]), .Q(n106) );
AO22X1 U103 ( .IN1(data_bus_in[7]), .IN2(n170), .IN3(n204), .IN4(
       data_mem_out[7]), .Q(n107) );
AO22X1 U104 ( .IN1(data_bus_in[8]), .IN2(n170), .IN3(n204), .IN4(
       data_mem_out[8]), .Q(n108) );
AO22X1 U105 ( .IN1(data_bus_in[9]), .IN2(n170), .IN3(n204), .IN4(
       data_mem_out[9]), .Q(n109) );
AO22X1 U106 ( .IN1(data_bus_in[10]), .IN2(n170), .IN3(n204), .IN4(
       data_mem_out[10]), .Q(n110) );
AO22X1 U107 ( .IN1(data_bus_in[11]), .IN2(n170), .IN3(n204), .IN4(
       data_mem_out[11]), .Q(n111) );
AO22X1 U108 ( .IN1(data_bus_in[12]), .IN2(n170), .IN3(n204), .IN4(
       data_mem_out[12]), .Q(n112) );
AO22X1 U109 ( .IN1(data_bus_in[13]), .IN2(n170), .IN3(n204), .IN4(
       data_mem_out[13]), .Q(n113) );
AO22X1 U110 ( .IN1(data_bus_in[14]), .IN2(n170), .IN3(n204), .IN4(
       data_mem_out[14]), .Q(n114) );
AO22X1 U111 ( .IN1(data_bus_in[15]), .IN2(n170), .IN3(n204), .IN4(
       data_mem_out[15]), .Q(n115) );
AO22X1 U112 ( .IN1(read_data[0]), .IN2(n168), .IN3(data_mem_in[0]), .IN4(
       read_en), .Q(n116) );
AO22X1 U113 ( .IN1(read_data[1]), .IN2(n168), .IN3(data_mem_in[1]), .IN4(
       read_en), .Q(n117) );
AO22X1 U114 ( .IN1(read_data[2]), .IN2(n168), .IN3(data_mem_in[2]), .IN4(
       read_en), .Q(n118) );
AO22X1 U115 ( .IN1(read_data[3]), .IN2(n168), .IN3(data_mem_in[3]), .IN4(
       read_en), .Q(n119) );
AO22X1 U116 ( .IN1(read_data[4]), .IN2(n168), .IN3(data_mem_in[4]), .IN4(
       read_en), .Q(n120) );
AO22X1 U117 ( .IN1(read_data[5]), .IN2(n168), .IN3(data_mem_in[5]), .IN4(
       read_en), .Q(n121) );
AO22X1 U118 ( .IN1(read_data[6]), .IN2(n168), .IN3(data_mem_in[6]), .IN4(
       read_en), .Q(n122) );
AO22X1 U119 ( .IN1(read_data[7]), .IN2(n168), .IN3(data_mem_in[7]), .IN4(
       read_en), .Q(n123) );
AO22X1 U120 ( .IN1(read_data[8]), .IN2(n168), .IN3(data_mem_in[8]), .IN4(
       read_en), .Q(n124) );
AO22X1 U121 ( .IN1(read_data[9]), .IN2(n168), .IN3(data_mem_in[9]), .IN4(
       read_en), .Q(n125) );
AO22X1 U122 ( .IN1(read_data[10]), .IN2(n168), .IN3(data_mem_in[10]), .IN4(
       read_en), .Q(n126) );
AO22X1 U123 ( .IN1(read_data[11]), .IN2(n168), .IN3(data_mem_in[11]), .IN4(
       read_en), .Q(n127) );
AO22X1 U124 ( .IN1(read_data[12]), .IN2(n168), .IN3(data_mem_in[12]), .IN4(
       read_en), .Q(n128) );
AO22X1 U125 ( .IN1(read_data[13]), .IN2(n168), .IN3(data_mem_in[13]), .IN4(
       read_en), .Q(n129) );
AO22X1 U126 ( .IN1(read_data[14]), .IN2(n168), .IN3(data_mem_in[14]), .IN4(
       read_en), .Q(n130) );
AO22X1 U127 ( .IN1(read_data[15]), .IN2(n168), .IN3(read_en), .IN4(
       data_mem_in[15]), .Q(n131) );
AND2X1 U128 ( .IN1(write_en), .IN2(n168), .Q(n170) );
INVX2 U129 ( .IN(read_en), .QN(n168) );
INVX0 U130 ( .IN(reset), .QN(n166) );
endmodule
```

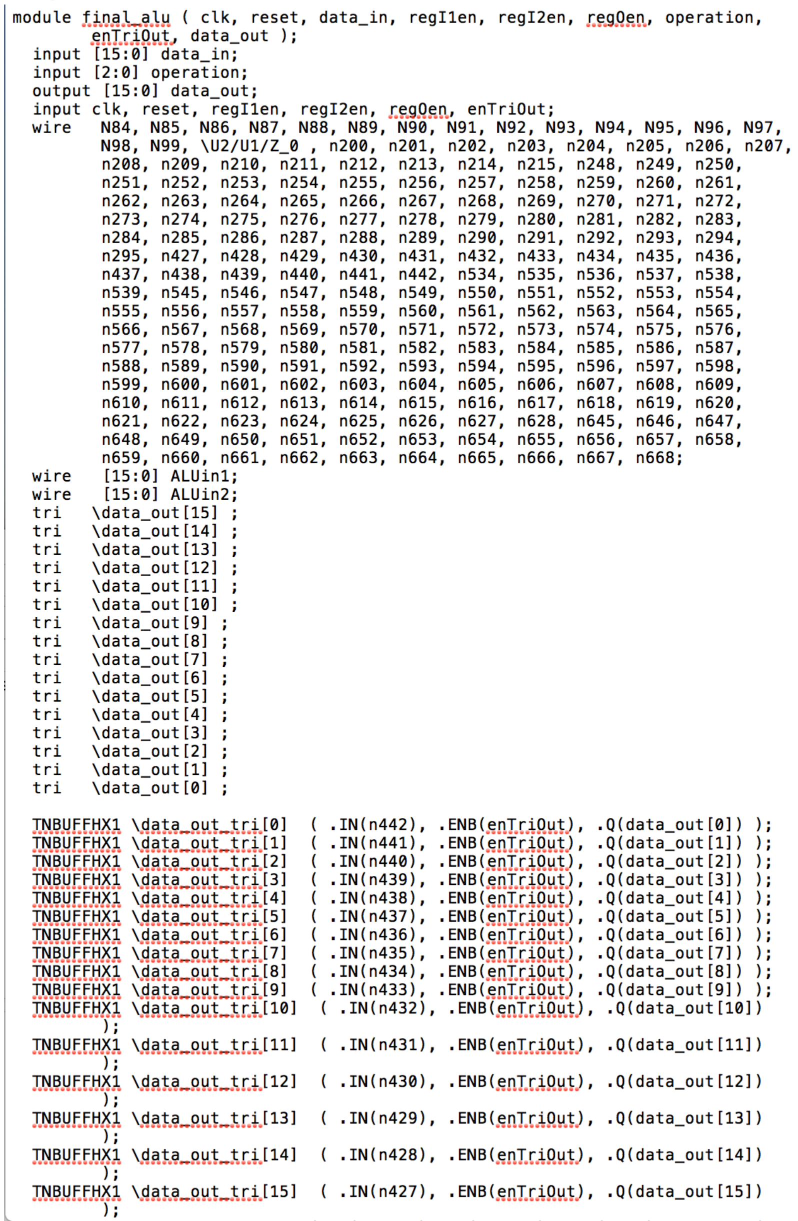
- ID

```
module iDecoder ( clk, reset, enable, instruction_in, opcode, ALUop, IMMop,
        MOVop, MOViop, LOADop, STOREop, A, B );
  input [15:0] instruction_in;
  output [3:0] opcode;
  output [5:0] A;
  output [5:0] B;
  input clk, reset, enable;
  output ALUop, IMMop, MOVop, MOViop, LOADop, STOREop;
  wire   N73, N74, N75, N76, N77, N78, n31, n32, n33, n34, n35, n36, n37, n38
  assign opcode[3] = instruction_in[15];
  assign opcode[2] = instruction_in[14];
  assign opcode[1] = instruction_in[13];
  assign opcode[0] = instruction_in[12];
  assign A[5] = instruction_in[11];
  assign A[4] = instruction_in[10];
  assign A[3] = instruction_in[9];
  assign A[2] = instruction_in[8];
  assign A[1] = instruction_in[7];
  assign A[0] = instruction_in[6];
  assign B[5] = instruction_in[5];
  assign B[4] = instruction_in[4];
  assign B[3] = instruction_in[3];
  assign B[2] = instruction_in[2];
  assign B[1] = instruction_in[1];
  assign B[0] = instruction_in[0];

  DFFX1 IMMop_reg ( .D(N77), .CLK(clk), .Q(IMMop) );
  DFFX1 LOADop_reg ( .D(N76), .CLK(clk), .Q(LOADop) );
  DFFX1 MOVop_reg ( .D(N78), .CLK(clk), .Q(MOVop) );
  DFFX1 ALUop_reg ( .D(N75), .CLK(clk), .Q(ALUop) );
  DFFX1 MOViop_reg ( .D(N73), .CLK(clk), .Q(MOViop) );
  DFFX1 STOREop_reg ( .D(N74), .CLK(clk), .Q(STOREop) );
  NAND2X0 U31 ( .IN1(n38), .IN2(n32), .QN(n35) );
  NOR2X0 U32 ( .IN1(n36), .IN2(n34), .QN(N74) );
  NOR3X0 U33 ( .IN1(n34), .IN2(n35), .IN3(n33), .QN(N73) );
  AND3X1 U34 ( .IN1(n37), .IN2(n31), .IN3(enable), .Q(N75) );
  NAND3X0 U35 ( .IN1(n33), .IN2(n32), .IN3(n34), .QN(n37) );
  NAND3X0 U36 ( .IN1(n38), .IN2(n33), .IN3(opcode[2]), .QN(n36) );
  ISOLANDX1 U37 ( .D(enable), .ISO(n31), .Q(n38) );
  NOR3X0 U38 ( .IN1(n35), .IN2(opcode[0]), .IN3(n33), .QN(N78) );
  NOR2X0 U39 ( .IN1(opcode[0]), .IN2(n36), .QN(N76) );
  NOR2X0 U40 ( .IN1(opcode[1]), .IN2(n35), .QN(N77) );
  INVX0 U41 ( .IN(opcode[1]), .QN(n33) );
  INVX0 U42 ( .IN(opcode[0]), .QN(n34) );
  INVX0 U43 ( .IN(opcode[2]), .QN(n32) );
  INVX0 U44 ( .IN(opcode[3]), .QN(n31) );
endmodule
```



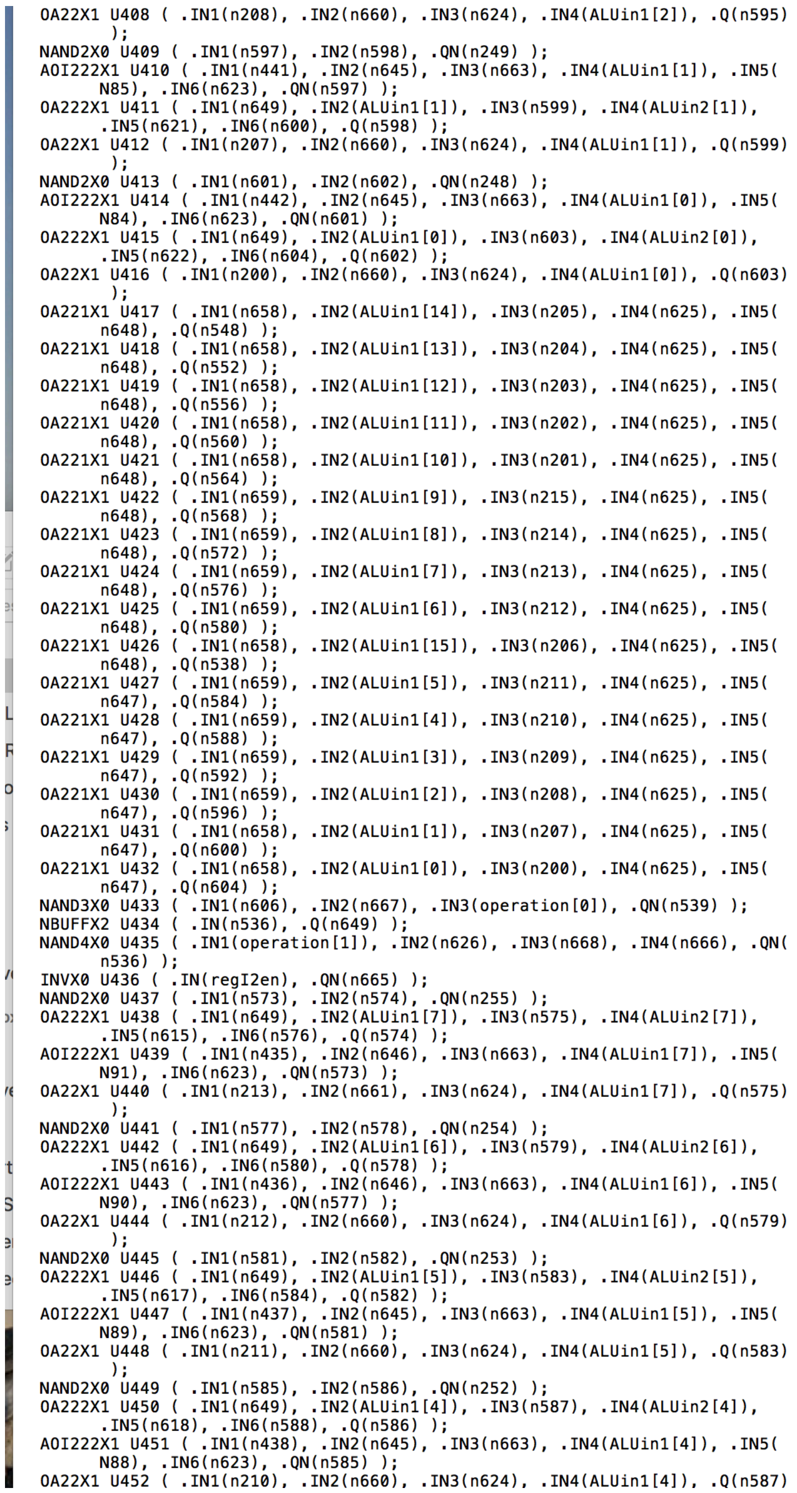
```
module final_alu_DW01_addsub_0 ( A, B, CI, ADD_SUB, SUM, CO );
  input [15:0] A;
  input [15:0] B;
  output [15:0] SUM;
  input CI, ADD_SUB;
  output CO;
  wire   n1, n2;
  wire   [16:0] carry;
  wire   [15:0] B_AS;
  assign carry[0] = ADD_SUB;

  FADDX1 U1_14 ( .A(A[14]), .B(B_AS[14]), .CI(carry[14]), .CO(carry[15]), .S(
        SUM[14]) );
  XOR3X1 U1_15 ( .IN1(A[15]), .IN2(B_AS[15]), .IN3(carry[15]), .Q(SUM[15]) );
  FADDX1 U1_3 ( .A(A[3]), .B(B_AS[3]), .CI(carry[3]), .CO(carry[4]), .S(SUM[3]) );
  FADDX1 U1_4 ( .A(A[4]), .B(B_AS[4]), .CI(carry[4]), .CO(carry[5]), .S(SUM[4]) );
  FADDX1 U1_5 ( .A(A[5]), .B(B_AS[5]), .CI(carry[5]), .CO(carry[6]), .S(SUM[5]) );
  FADDX1 U1_6 ( .A(A[6]), .B(B_AS[6]), .CI(carry[6]), .CO(carry[7]), .S(SUM[6]) );
  FADDX1 U1_7 ( .A(A[7]), .B(B_AS[7]), .CI(carry[7]), .CO(carry[8]), .S(SUM[7]) );
  FADDX1 U1_8 ( .A(A[8]), .B(B_AS[8]), .CI(carry[8]), .CO(carry[9]), .S(SUM[8]) );
  FADDX1 U1_9 ( .A(A[9]), .B(B_AS[9]), .CI(carry[9]), .CO(carry[10]), .S(
        SUM[9]) );
  FADDX1 U1_10 ( .A(A[10]), .B(B_AS[10]), .CI(carry[10]), .CO(carry[11]), .S(
        SUM[10]) );
  FADDX1 U1_11 ( .A(A[11]), .B(B_AS[11]), .CI(carry[11]), .CO(carry[12]), .S(
        SUM[11]) );
  FADDX1 U1_12 ( .A(A[12]), .B(B_AS[12]), .CI(carry[12]), .CO(carry[13]), .S(
        SUM[12]) );
  FADDX1 U1_13 ( .A(A[13]), .B(B_AS[13]), .CI(carry[13]), .CO(carry[14]), .S(
        SUM[13]) );
  FADDX1 U1_2 ( .A(A[2]), .B(B_AS[2]), .CI(carry[2]), .CO(carry[3]), .S(SUM[2]) );
  FADDX1 U1_1 ( .A(A[1]), .B(B_AS[1]), .CI(carry[1]), .CO(carry[2]), .S(SUM[1]) );
  FADDX1 U1_0 ( .A(A[0]), .B(B_AS[0]), .CI(carry[0]), .CO(carry[1]), .S(SUM[0]) );
  INVX0 U1 ( .IN(carry[0]), .QN(n1) );
  INVX0 U2 ( .IN(n1), .QN(n2) );
  XOR2X1 U3 ( .IN1(B[0]), .IN2(n2), .Q(B_AS[0]) );
  XOR2X1 U4 ( .IN1(B[1]), .IN2(n2), .Q(B_AS[1]) );
  XOR2X1 U5 ( .IN1(B[2]), .IN2(n2), .Q(B_AS[2]) );
  XOR2X1 U6 ( .IN1(B[13]), .IN2(n2), .Q(B_AS[13]) );
  XOR2X1 U7 ( .IN1(B[12]), .IN2(n2), .Q(B_AS[12]) );
  XOR2X1 U8 ( .IN1(B[11]), .IN2(n2), .Q(B_AS[11]) );
  XOR2X1 U9 ( .IN1(B[10]), .IN2(n2), .Q(B_AS[10]) );
  XOR2X1 U10 ( .IN1(B[9]), .IN2(carry[0]), .Q(B_AS[9]) );
  XOR2X1 U11 ( .IN1(B[8]), .IN2(carry[0]), .Q(B_AS[8]) );
  XOR2X1 U12 ( .IN1(B[7]), .IN2(carry[0]), .Q(B_AS[7]) );
  XOR2X1 U13 ( .IN1(B[6]), .IN2(carry[0]), .Q(B_AS[6]) );
  XOR2X1 U14 ( .IN1(B[5]), .IN2(carry[0]), .Q(B_AS[5]) );
  XOR2X1 U15 ( .IN1(B[4]), .IN2(carry[0]), .Q(B_AS[4]) );
  XOR2X1 U16 ( .IN1(B[3]), .IN2(carry[0]), .Q(B_AS[3]) );
  XOR2X1 U17 ( .IN1(B[15]), .IN2(n2), .Q(B_AS[15]) );
  XOR2X1 U18 ( .IN1(B[14]), .IN2(n2), .Q(B_AS[14]) );
endmodule
```

- ALU



```
module final_alu ( clk, reset, data_in, regI1en, regI2en, regOen, operation,
       enTriOut, data_out );
  input [15:0] data_in;
  input [2:0] operation;
  output [15:0] data_out;
  input clk, reset, regI1en, regI2en, regOen, enTriOut;
  wire   N84, N85, N86, N87, N88, N89, N90, N91, N92, N93, N94, N95, N96, N97,
         N98, N99, \U2/U1/Z_0 , n200, n201, n202, n203, n204, n205, n206, n207,
         n208, n209, n210, n211, n212, n213, n214, n215, n248, n249, n250,
         n251, n252, n253, n254, n255, n256, n257, n258, n259, n260, n261,
         n262, n263, n264, n265, n266, n267, n268, n269, n270, n271, n272,
         n273, n274, n275, n276, n277, n278, n279, n280, n281, n282, n283,
         n284, n285, n286, n287, n288, n289, n290, n291, n292, n293, n294,
         n295, n427, n428, n429, n430, n431, n432, n433, n434, n435, n436,
         n437, n438, n439, n440, n441, n442, n534, n535, n536, n537, n538,
         n539, n545, n546, n547, n548, n549, n550, n551, n552, n553, n554,
         n555, n556, n557, n558, n559, n560, n561, n562, n563, n564, n565,
         n566, n567, n568, n569, n570, n571, n572, n573, n574, n575, n576,
         n577, n578, n579, n580, n581, n582, n583, n584, n585, n586, n587,
         n588, n589, n590, n591, n592, n593, n594, n595, n596, n597, n598,
         n599, n600, n601, n602, n603, n604, n605, n606, n607, n608, n609,
         n610, n611, n612, n613, n614, n615, n616, n617, n618, n619, n620,
         n621, n622, n623, n624, n625, n626, n627, n628, n645, n646, n647,
         n648, n649, n650, n651, n652, n653, n654, n655, n656, n657, n658,
         n659, n660, n661, n662, n663, n664, n665, n666, n667, n668;
  wire   [15:0] ALUin1;
  wire   [15:0] ALUin2;
  tri   \data_out[15] ;
  tri   \data_out[14] ;
  tri   \data_out[13] ;
  tri   \data_out[12] ;
  tri   \data_out[11] ;
  tri   \data_out[10] ;
  tri   \data_out[9] ;
  tri   \data_out[8] ;
  tri   \data_out[7] ;
  tri   \data_out[6] ;
  tri   \data_out[5] ;
  tri   \data_out[4] ;
  tri   \data_out[3] ;
  tri   \data_out[2] ;
  tri   \data_out[1] ;
  tri   \data_out[0] ;

  TNBUFFHX1 \data_out_tri[0]  ( .IN(n442), .ENB(enTriOut), .Q(data_out[0]) );
  TNBUFFHX1 \data_out_tri[1]  ( .IN(n441), .ENB(enTriOut), .Q(data_out[1]) );
  TNBUFFHX1 \data_out_tri[2]  ( .IN(n440), .ENB(enTriOut), .Q(data_out[2]) );
  TNBUFFHX1 \data_out_tri[3]  ( .IN(n439), .ENB(enTriOut), .Q(data_out[3]) );
  TNBUFFHX1 \data_out_tri[4]  ( .IN(n438), .ENB(enTriOut), .Q(data_out[4]) );
  TNBUFFHX1 \data_out_tri[5]  ( .IN(n437), .ENB(enTriOut), .Q(data_out[5]) );
  TNBUFFHX1 \data_out_tri[6]  ( .IN(n436), .ENB(enTriOut), .Q(data_out[6]) );
  TNBUFFHX1 \data_out_tri[7]  ( .IN(n435), .ENB(enTriOut), .Q(data_out[7]) );
  TNBUFFHX1 \data_out_tri[8]  ( .IN(n434), .ENB(enTriOut), .Q(data_out[8]) );
  TNBUFFHX1 \data_out_tri[9]  ( .IN(n433), .ENB(enTriOut), .Q(data_out[9]) );
  TNBUFFHX1 \data_out_tri[10]  ( .IN(n432), .ENB(enTriOut), .Q(data_out[10])
        );
  TNBUFFHX1 \data_out_tri[11]  ( .IN(n431), .ENB(enTriOut), .Q(data_out[11])
        );
  TNBUFFHX1 \data_out_tri[12]  ( .IN(n430), .ENB(enTriOut), .Q(data_out[12])
        );
  TNBUFFHX1 \data_out_tri[13]  ( .IN(n429), .ENB(enTriOut), .Q(data_out[13])
        );
  TNBUFFHX1 \data_out_tri[14]  ( .IN(n428), .ENB(enTriOut), .Q(data_out[14])
        );
  TNBUFFHX1 \data_out_tri[15]  ( .IN(n427), .ENB(enTriOut), .Q(data_out[15])
        );
```

```
final_alu_DW01_addsub_0 r56 ( .A(ALUin1), .B(ALUin2), .CI(1'b0), .ADD_SUB(
        \U2/U1/Z_0 ), .SUM({N99, N98, N97, N96, N95, N94, N93, N92, N91, N90,
        N89, N88, N87, N86, N85, N84}) );
DFFARX1 \ALUin1_reg[15]  ( .D(n295), .CLK(clk), .RSTB(n653), .Q(ALUin1[15]),
        .QN(n206) );
DFFARX1 \ALUin1_reg[14]  ( .D(n294), .CLK(clk), .RSTB(n652), .Q(ALUin1[14]),
        .QN(n205) );
DFFARX1 \ALUin2_reg[7]  ( .D(n271), .CLK(clk), .RSTB(n654), .Q(ALUin2[7]),
        .QN(n615) );
DFFARX1 \ALUin2_reg[8]  ( .D(n272), .CLK(clk), .RSTB(n654), .Q(ALUin2[8]),
        .QN(n614) );
DFFARX1 \ALUin2_reg[9]  ( .D(n273), .CLK(clk), .RSTB(n654), .Q(ALUin2[9]),
        .QN(n613) );
DFFARX1 \ALUin2_reg[10]  ( .D(n274), .CLK(clk), .RSTB(n654), .Q(ALUin2[10]),
        .QN(n612) );
DFFARX1 \ALUin2_reg[11]  ( .D(n275), .CLK(clk), .RSTB(n654), .Q(ALUin2[11]),
        .QN(n611) );
DFFARX1 \ALUin2_reg[12]  ( .D(n276), .CLK(clk), .RSTB(n654), .Q(ALUin2[12]),
        .QN(n610) );
DFFARX1 \ALUin2_reg[13]  ( .D(n277), .CLK(clk), .RSTB(n654), .Q(ALUin2[13]),
        .QN(n609) );
DFFARX1 \ALUin2_reg[14]  ( .D(n278), .CLK(clk), .RSTB(n653), .Q(ALUin2[14]),
        .QN(n608) );
DFFARX1 \ALUin2_reg[15]  ( .D(n279), .CLK(clk), .RSTB(n653), .Q(ALUin2[15]),
        .QN(n607) );
DFFARX1 \ALUin1_reg[6]  ( .D(n286), .CLK(clk), .RSTB(n652), .Q(ALUin1[6]),
        .QN(n212) );
DFFARX1 \ALUin1_reg[7]  ( .D(n287), .CLK(clk), .RSTB(n652), .Q(ALUin1[7]),
        .QN(n213) );
DFFARX1 \ALUin1_reg[8]  ( .D(n288), .CLK(clk), .RSTB(n652), .Q(ALUin1[8]),
        .QN(n214) );
DFFARX1 \ALUin1_reg[9]  ( .D(n289), .CLK(clk), .RSTB(n652), .Q(ALUin1[9]),
        .QN(n215) );
DFFARX1 \ALUin1_reg[10]  ( .D(n290), .CLK(clk), .RSTB(n652), .Q(ALUin1[10]),
        .QN(n201) );
DFFARX1 \ALUin1_reg[11]  ( .D(n291), .CLK(clk), .RSTB(n652), .Q(ALUin1[11]),
        .QN(n202) );
DFFARX1 \ALUin1_reg[12]  ( .D(n292), .CLK(clk), .RSTB(n652), .Q(ALUin1[12]),
        .QN(n203) );
DFFARX1 \ALUin1_reg[13]  ( .D(n293), .CLK(clk), .RSTB(n652), .Q(ALUin1[13]),
        .QN(n204) );
DFFARX1 \ALUin2_reg[0]  ( .D(n264), .CLK(clk), .RSTB(n655), .Q(ALUin2[0]),
        .QN(n622) );
DFFARX1 \ALUin2_reg[1]  ( .D(n265), .CLK(clk), .RSTB(n655), .Q(ALUin2[1]),
        .QN(n621) );
DFFARX1 \ALUin2_reg[2]  ( .D(n266), .CLK(clk), .RSTB(n655), .Q(ALUin2[2]),
        .QN(n620) );
DFFARX1 \ALUin2_reg[3]  ( .D(n267), .CLK(clk), .RSTB(n655), .Q(ALUin2[3]),
        .QN(n619) );
DFFARX1 \ALUin2_reg[4]  ( .D(n268), .CLK(clk), .RSTB(n655), .Q(ALUin2[4]),
        .QN(n618) );
DFFARX1 \ALUin2_reg[5]  ( .D(n269), .CLK(clk), .RSTB(n654), .Q(ALUin2[5]),
        .QN(n617) );
DFFARX1 \ALUin2_reg[6]  ( .D(n270), .CLK(clk), .RSTB(n654), .Q(ALUin2[6]),
        .QN(n616) );
DFFARX1 \ALUin1_reg[0]  ( .D(n280), .CLK(clk), .RSTB(n653), .Q(ALUin1[0]),
        .QN(n200) );
DFFARX1 \ALUin1_reg[1]  ( .D(n281), .CLK(clk), .RSTB(n653), .Q(ALUin1[1]),
        .QN(n207) );
DFFARX1 \ALUin1_reg[2]  ( .D(n282), .CLK(clk), .RSTB(n653), .Q(ALUin1[2]),
        .QN(n208) );
DFFARX1 \ALUin1_reg[3]  ( .D(n283), .CLK(clk), .RSTB(n653), .Q(ALUin1[3]),
        .QN(n209) );
DFFARX1 \ALUin1_reg[4]  ( .D(n284), .CLK(clk), .RSTB(n653), .Q(ALUin1[4]),
        .QN(n210) );
```

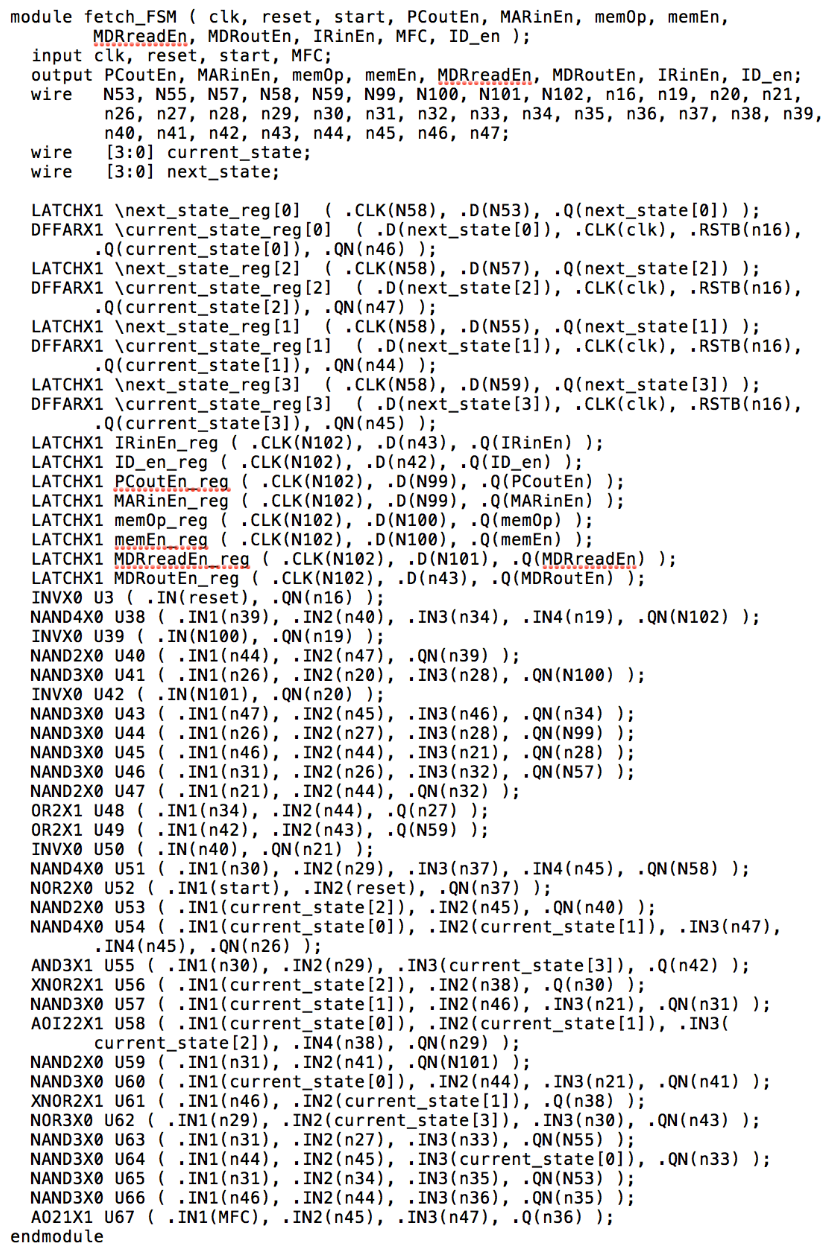


```
OA22X1 U408 ( .IN1(n208), .IN2(n660), .IN3(n624), .IN4(ALUin1[2]), .Q(n595)
        );
NAND2X0 U409 ( .IN1(n597), .IN2(n598), .QN(n249) );
AOI222X1 U410 ( .IN1(n441), .IN2(n645), .IN3(n663), .IN4(ALUin1[1]), .IN5(
        N85), .IN6(n623), .QN(n597) );
OA222X1 U411 ( .IN1(n649), .IN2(ALUin1[1]), .IN3(n599), .IN4(ALUin2[1]),
        .IN5(n621), .IN6(n600), .Q(n598) );
OA22X1 U412 ( .IN1(n207), .IN2(n660), .IN3(n624), .IN4(ALUin1[1]), .Q(n599)
        );
NAND2X0 U413 ( .IN1(n601), .IN2(n602), .QN(n248) );
AOI222X1 U414 ( .IN1(n442), .IN2(n645), .IN3(n663), .IN4(ALUin1[0]), .IN5(
        N84), .IN6(n623), .QN(n601) );
OA222X1 U415 ( .IN1(n649), .IN2(ALUin1[0]), .IN3(n603), .IN4(ALUin2[0]),
        .IN5(n622), .IN6(n604), .Q(n602) );
OA22X1 U416 ( .IN1(n200), .IN2(n660), .IN3(n624), .IN4(ALUin1[0]), .Q(n603)
        );
OA221X1 U417 ( .IN1(n658), .IN2(ALUin1[14]), .IN3(n205), .IN4(n625), .IN5(
        n648), .Q(n548) );
OA221X1 U418 ( .IN1(n658), .IN2(ALUin1[13]), .IN3(n204), .IN4(n625), .IN5(
        n648), .Q(n552) );
OA221X1 U419 ( .IN1(n658), .IN2(ALUin1[12]), .IN3(n203), .IN4(n625), .IN5(
        n648), .Q(n556) );
OA221X1 U420 ( .IN1(n658), .IN2(ALUin1[11]), .IN3(n202), .IN4(n625), .IN5(
        n648), .Q(n560) );
OA221X1 U421 ( .IN1(n658), .IN2(ALUin1[10]), .IN3(n201), .IN4(n625), .IN5(
        n648), .Q(n564) );
OA221X1 U422 ( .IN1(n659), .IN2(ALUin1[9]), .IN3(n215), .IN4(n625), .IN5(
        n648), .Q(n568) );
OA221X1 U423 ( .IN1(n659), .IN2(ALUin1[8]), .IN3(n214), .IN4(n625), .IN5(
        n648), .Q(n572) );
OA221X1 U424 ( .IN1(n659), .IN2(ALUin1[7]), .IN3(n213), .IN4(n625), .IN5(
        n648), .Q(n576) );
OA221X1 U425 ( .IN1(n659), .IN2(ALUin1[6]), .IN3(n212), .IN4(n625), .IN5(
        n648), .Q(n580) );
OA221X1 U426 ( .IN1(n658), .IN2(ALUin1[15]), .IN3(n206), .IN4(n625), .IN5(
        n648), .Q(n538) );
OA221X1 U427 ( .IN1(n659), .IN2(ALUin1[5]), .IN3(n211), .IN4(n625), .IN5(
        n647), .Q(n584) );
OA221X1 U428 ( .IN1(n659), .IN2(ALUin1[4]), .IN3(n210), .IN4(n625), .IN5(
        n647), .Q(n588) );
OA221X1 U429 ( .IN1(n659), .IN2(ALUin1[3]), .IN3(n209), .IN4(n625), .IN5(
        n647), .Q(n592) );
OA221X1 U430 ( .IN1(n659), .IN2(ALUin1[2]), .IN3(n208), .IN4(n625), .IN5(
        n647), .Q(n596) );
OA221X1 U431 ( .IN1(n658), .IN2(ALUin1[1]), .IN3(n207), .IN4(n625), .IN5(
        n647), .Q(n600) );
OA221X1 U432 ( .IN1(n658), .IN2(ALUin1[0]), .IN3(n200), .IN4(n625), .IN5(
        n647), .Q(n604) );
NAND3X0 U433 ( .IN1(n606), .IN2(n667), .IN3(operation[0]), .QN(n539) );
NBUFFX2 U434 ( .IN(n536), .Q(n649) );
NAND4X0 U435 ( .IN1(operation[1]), .IN2(n626), .IN3(n668), .IN4(n666), .QN(
        n536) );
INVX0 U436 ( .IN(regI2en), .QN(n665) );
NAND2X0 U437 ( .IN1(n573), .IN2(n574), .QN(n255) );
OA222X1 U438 ( .IN1(n649), .IN2(ALUin1[7]), .IN3(n575), .IN4(ALUin2[7]),
        .IN5(n615), .IN6(n576), .Q(n574) );
AOI222X1 U439 ( .IN1(n435), .IN2(n646), .IN3(n663), .IN4(ALUin1[7]), .IN5(
        N91), .IN6(n623), .QN(n573) );
OA22X1 U440 ( .IN1(n213), .IN2(n661), .IN3(n624), .IN4(ALUin1[7]), .Q(n575)
        );
NAND2X0 U441 ( .IN1(n577), .IN2(n578), .QN(n254) );
OA222X1 U442 ( .IN1(n649), .IN2(ALUin1[6]), .IN3(n579), .IN4(ALUin2[6]),
        .IN5(n616), .IN6(n580), .Q(n578) );
AOI222X1 U443 ( .IN1(n436), .IN2(n646), .IN3(n663), .IN4(ALUin1[6]), .IN5(
        N90), .IN6(n623), .QN(n577) );
OA22X1 U444 ( .IN1(n212), .IN2(n660), .IN3(n624), .IN4(ALUin1[6]), .Q(n579)
        );
NAND2X0 U445 ( .IN1(n581), .IN2(n582), .QN(n253) );
OA222X1 U446 ( .IN1(n649), .IN2(ALUin1[5]), .IN3(n583), .IN4(ALUin2[5]),
        .IN5(n617), .IN6(n584), .Q(n582) );
AOI222X1 U447 ( .IN1(n437), .IN2(n645), .IN3(n663), .IN4(ALUin1[5]), .IN5(
        N89), .IN6(n623), .QN(n581) );
OA22X1 U448 ( .IN1(n211), .IN2(n660), .IN3(n624), .IN4(ALUin1[5]), .Q(n583)
        );
NAND2X0 U449 ( .IN1(n585), .IN2(n586), .QN(n252) );
OA222X1 U450 ( .IN1(n649), .IN2(ALUin1[4]), .IN3(n587), .IN4(ALUin2[4]),
        .IN5(n618), .IN6(n588), .Q(n586) );
AOI222X1 U451 ( .IN1(n438), .IN2(n645), .IN3(n663), .IN4(ALUin1[4]), .IN5(
        N88), .IN6(n623), .QN(n585) );
OA22X1 U452 ( .IN1(n210), .IN2(n660), .IN3(n624), .IN4(ALUin1[4]), .Q(n587)
DFFARX1 \ALUin1_reg[5]  ( .D(n285), .CLK(clk), .RSTB(n653), .Q(ALUin1[5]),
        .QN(n211) );
DFFARX1 \result_reg[15]  ( .D(n263), .CLK(clk), .RSTB(n657), .Q(n427) );
DFFARX1 \result_reg[0]  ( .D(n248), .CLK(clk), .RSTB(n657), .Q(n442) );
DFFARX1 \result_reg[1]  ( .D(n249), .CLK(clk), .RSTB(n657), .Q(n441) );
DFFARX1 \result_reg[2]  ( .D(n250), .CLK(clk), .RSTB(n656), .Q(n440) );
DFFARX1 \result_reg[3]  ( .D(n251), .CLK(clk), .RSTB(n656), .Q(n439) );
DFFARX1 \result_reg[4]  ( .D(n252), .CLK(clk), .RSTB(n656), .Q(n438) );
DFFARX1 \result_reg[5]  ( .D(n253), .CLK(clk), .RSTB(n656), .Q(n437) );
DFFARX1 \result_reg[6]  ( .D(n254), .CLK(clk), .RSTB(n656), .Q(n436) );
DFFARX1 \result_reg[7]  ( .D(n255), .CLK(clk), .RSTB(n656), .Q(n435) );
DFFARX1 \result_reg[8]  ( .D(n256), .CLK(clk), .RSTB(n656), .Q(n434) );
DFFARX1 \result_reg[9]  ( .D(n257), .CLK(clk), .RSTB(n656), .Q(n433) );
DFFARX1 \result_reg[10]  ( .D(n258), .CLK(clk), .RSTB(n656), .Q(n432) );
DFFARX1 \result_reg[11]  ( .D(n259), .CLK(clk), .RSTB(n655), .Q(n431) );
DFFARX1 \result_reg[12]  ( .D(n260), .CLK(clk), .RSTB(n655), .Q(n430) );
DFFARX1 \result_reg[13]  ( .D(n261), .CLK(clk), .RSTB(n655), .Q(n429) );
DFFARX1 \result_reg[14]  ( .D(n262), .CLK(clk), .RSTB(n655), .Q(n428) );
AND3X1 U347 ( .IN1(n667), .IN2(n666), .IN3(n626), .Q(n623) );
NAND3X1 U348 ( .IN1(n606), .IN2(n668), .IN3(operation[1]), .QN(n624) );
AND2X1 U349 ( .IN1(n624), .IN2(n605), .Q(n625) );
INVX0 U350 ( .IN(n627), .QN(n647) );
INVX0 U351 ( .IN(n628), .QN(n650) );
AND3X1 U352 ( .IN1(n664), .IN2(n665), .IN3(regOen), .Q(n626) );
INVX2 U353 ( .IN(n647), .QN(n663) );
INVX0 U354 ( .IN(n627), .QN(n648) );
NOR2X0 U355 ( .IN1(n666), .IN2(n645), .QN(n606) );
AND3X1 U356 ( .IN1(n668), .IN2(n667), .IN3(n606), .Q(n627) );
NBUFFX2 U357 ( .IN(n539), .Q(n659) );
NBUFFX2 U358 ( .IN(n539), .Q(n658) );
NBUFFX2 U359 ( .IN(n539), .Q(n661) );
NBUFFX2 U360 ( .IN(n539), .Q(n660) );
INVX0 U361 ( .IN(n626), .QN(n645) );
INVX0 U362 ( .IN(n628), .QN(n651) );
INVX0 U363 ( .IN(n626), .QN(n646) );
NBUFFX2 U364 ( .IN(n662), .Q(n652) );
NBUFFX2 U365 ( .IN(n662), .Q(n653) );
NBUFFX2 U366 ( .IN(n662), .Q(n654) );
NBUFFX2 U367 ( .IN(n662), .Q(n655) );
NBUFFX2 U368 ( .IN(n662), .Q(n656) );
NBUFFX2 U369 ( .IN(n662), .Q(n657) );
AND3X1 U370 ( .IN1(operation[0]), .IN2(n666), .IN3(n667), .Q(\U2/U1/Z_0 ) );
NAND2X0 U371 ( .IN1(n545), .IN2(n546), .QN(n262) );
OA222X1 U372 ( .IN1(n649), .IN2(ALUin1[14]), .IN3(n547), .IN4(ALUin2[14]),
        .IN5(n608), .IN6(n548), .Q(n546) );
AOI222X1 U373 ( .IN1(n428), .IN2(n646), .IN3(n663), .IN4(ALUin1[14]), .IN5(
        N98), .IN6(n623), .QN(n545) );
OA22X1 U374 ( .IN1(n205), .IN2(n661), .IN3(n624), .IN4(ALUin1[14]), .Q(n547)
         );
NAND2X0 U375 ( .IN1(n549), .IN2(n550), .QN(n261) );
OA222X1 U376 ( .IN1(n649), .IN2(ALUin1[13]), .IN3(n551), .IN4(ALUin2[13]),
        .IN5(n609), .IN6(n552), .Q(n550) );
AOI222X1 U377 ( .IN1(n429), .IN2(n646), .IN3(n663), .IN4(ALUin1[13]), .IN5(
        N97), .IN6(n623), .QN(n549) );
OA22X1 U378 ( .IN1(n204), .IN2(n661), .IN3(n624), .IN4(ALUin1[13]), .Q(n551)
         );
NAND2X0 U379 ( .IN1(n553), .IN2(n554), .QN(n260) );
OA222X1 U380 ( .IN1(n649), .IN2(ALUin1[12]), .IN3(n555), .IN4(ALUin2[12]),
        .IN5(n610), .IN6(n556), .Q(n554) );
AOI222X1 U381 ( .IN1(n430), .IN2(n646), .IN3(n663), .IN4(ALUin1[12]), .IN5(
        N96), .IN6(n623), .QN(n553) );
OA22X1 U382 ( .IN1(n203), .IN2(n661), .IN3(n624), .IN4(ALUin1[12]), .Q(n555)
         );
NAND2X0 U383 ( .IN1(n557), .IN2(n558), .QN(n259) );
OA222X1 U384 ( .IN1(n649), .IN2(ALUin1[11]), .IN3(n559), .IN4(ALUin2[11]),
        .IN5(n611), .IN6(n560), .Q(n558) );
AOI222X1 U385 ( .IN1(n431), .IN2(n646), .IN3(n663), .IN4(ALUin1[11]), .IN5(
        N95), .IN6(n623), .QN(n557) );
OA22X1 U386 ( .IN1(n202), .IN2(n661), .IN3(n624), .IN4(ALUin1[11]), .Q(n559)
         );
```

```
  NAND2X0 U453 ( .IN1(n589), .IN2(n590), .QN(n251) );
  OA222X1 U454 ( .IN1(n649), .IN2(ALUin1[3]), .IN3(n591), .IN4(ALUin2[3]),
        .IN5(n619), .IN6(n592), .Q(n590) );
  AOI222X1 U455 ( .IN1(n439), .IN2(n645), .IN3(n663), .IN4(ALUin1[3]), .IN5(
        N87), .IN6(n623), .QN(n589) );
  OA22X1 U456 ( .IN1(n209), .IN2(n660), .IN3(n624), .IN4(ALUin1[3]), .Q(n591)
         );
  NAND4X0 U457 ( .IN1(operation[1]), .IN2(operation[0]), .IN3(n626), .IN4(n666), .QN(n605) );
  INVX2 U458 ( .IN(regI1en), .QN(n664) );
  INVX0 U459 ( .IN(operation[0]), .QN(n668) );
  AND2X1 U460 ( .IN1(regI2en), .IN2(n664), .Q(n628) );
  AO22X1 U461 ( .IN1(n628), .IN2(data_in[15]), .IN3(n651), .IN4(ALUin2[15]),
        .Q(n279) );
  AO22X1 U462 ( .IN1(n628), .IN2(data_in[14]), .IN3(n651), .IN4(ALUin2[14]),
        .Q(n278) );
  AO22X1 U463 ( .IN1(n628), .IN2(data_in[13]), .IN3(n651), .IN4(ALUin2[13]),
        .Q(n277) );
  AO22X1 U464 ( .IN1(n628), .IN2(data_in[12]), .IN3(n651), .IN4(ALUin2[12]),
        .Q(n276) );
  AO22X1 U465 ( .IN1(n628), .IN2(data_in[11]), .IN3(n651), .IN4(ALUin2[11]),
        .Q(n275) );
  AO22X1 U466 ( .IN1(n628), .IN2(data_in[10]), .IN3(n651), .IN4(ALUin2[10]),
        .Q(n274) );
  AO22X1 U467 ( .IN1(n628), .IN2(data_in[9]), .IN3(n651), .IN4(ALUin2[9]), .Q(
        n273) );
  AO22X1 U468 ( .IN1(n628), .IN2(data_in[8]), .IN3(n651), .IN4(ALUin2[8]), .Q(
        n272) );
  AO22X1 U469 ( .IN1(n628), .IN2(data_in[7]), .IN3(n651), .IN4(ALUin2[7]), .Q(
        n271) );
  AO22X1 U470 ( .IN1(n628), .IN2(data_in[6]), .IN3(n651), .IN4(ALUin2[6]), .Q(
        n270) );
  AO22X1 U471 ( .IN1(n628), .IN2(data_in[5]), .IN3(n650), .IN4(ALUin2[5]), .Q(
        n269) );
  AO22X1 U472 ( .IN1(n628), .IN2(data_in[4]), .IN3(n650), .IN4(ALUin2[4]), .Q(
        n268) );
  AO22X1 U473 ( .IN1(n628), .IN2(data_in[3]), .IN3(n650), .IN4(ALUin2[3]), .Q(
        n267) );
  AO22X1 U474 ( .IN1(n628), .IN2(data_in[2]), .IN3(n650), .IN4(ALUin2[2]), .Q(
        n266) );
  AO22X1 U475 ( .IN1(n628), .IN2(data_in[1]), .IN3(n650), .IN4(ALUin2[1]), .Q(
        n265) );
  AO22X1 U476 ( .IN1(n628), .IN2(data_in[0]), .IN3(n650), .IN4(ALUin2[0]), .Q(
        n264) );
  AO22X1 U477 ( .IN1(ALUin1[14]), .IN2(n664), .IN3(data_in[14]), .IN4(regI1en),
        .Q(n294) );
  AO22X1 U478 ( .IN1(ALUin1[13]), .IN2(n664), .IN3(data_in[13]), .IN4(regI1en),
        .Q(n293) );
  AO22X1 U479 ( .IN1(ALUin1[12]), .IN2(n664), .IN3(data_in[12]), .IN4(regI1en),
        .Q(n292) );
  AO22X1 U480 ( .IN1(ALUin1[11]), .IN2(n664), .IN3(data_in[11]), .IN4(regI1en),
        .Q(n291) );
  AO22X1 U481 ( .IN1(ALUin1[10]), .IN2(n664), .IN3(data_in[10]), .IN4(regI1en),
        .Q(n290) );
  AO22X1 U482 ( .IN1(ALUin1[9]), .IN2(n664), .IN3(data_in[9]), .IN4(regI1en),
        .Q(n289) );
  AO22X1 U483 ( .IN1(ALUin1[8]), .IN2(n664), .IN3(data_in[8]), .IN4(regI1en),
        .Q(n288) );
  AO22X1 U484 ( .IN1(ALUin1[7]), .IN2(n664), .IN3(data_in[7]), .IN4(regI1en),
        .Q(n287) );
  AO22X1 U485 ( .IN1(ALUin1[6]), .IN2(n664), .IN3(data_in[6]), .IN4(regI1en),
        .Q(n286) );
  AO22X1 U486 ( .IN1(ALUin1[5]), .IN2(n664), .IN3(data_in[5]), .IN4(regI1en),
        .Q(n285) );
  AO22X1 U487 ( .IN1(ALUin1[4]), .IN2(n664), .IN3(data_in[4]), .IN4(regI1en),
        .Q(n284) );
  AO22X1 U488 ( .IN1(ALUin1[3]), .IN2(n664), .IN3(data_in[3]), .IN4(regI1en),
        .Q(n283) );
  AO22X1 U489 ( .IN1(ALUin1[2]), .IN2(n664), .IN3(data_in[2]), .IN4(regI1en),
        .Q(n282) );
  AO22X1 U490 ( .IN1(ALUin1[1]), .IN2(n664), .IN3(data_in[1]), .IN4(regI1en),
        .Q(n281) );
  AO22X1 U491 ( .IN1(ALUin1[0]), .IN2(n664), .IN3(data_in[0]), .IN4(regI1en),
        .Q(n280) );
  AO22X1 U492 ( .IN1(ALUin1[15]), .IN2(n664), .IN3(regI1en), .IN4(data_in[15]),
        .Q(n295) );
  INVX0 U493 ( .IN(reset), .QN(n662) );
endmodule
```

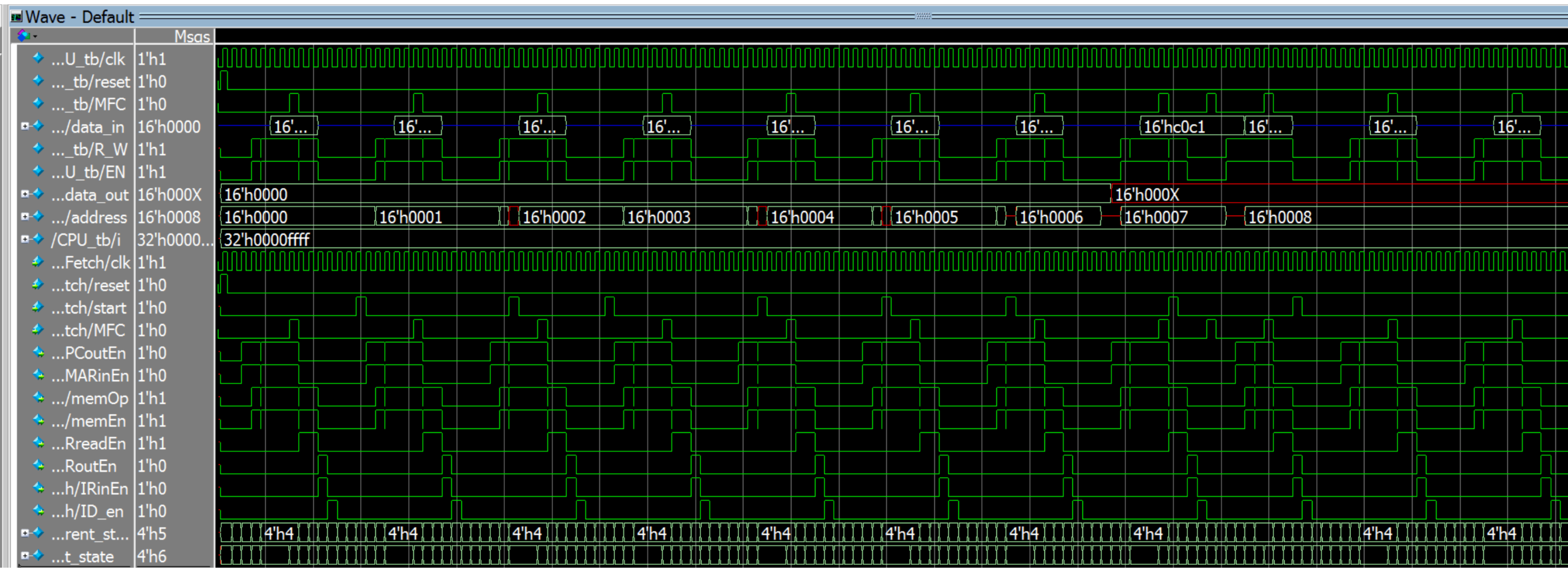
- Instruction Fetch FSM

- Netlist



```
module fetch_FSM ( clk, reset, start, PCoutEn, MARinEn, memOp, memEn,
        MDRreadEn, MDRoutEn, IRinEn, MFC, ID_en );
  input clk, reset, start, MFC;
  output PCoutEn, MARinEn, memOp, memEn, MDRreadEn, MDRoutEn, IRinEn, ID_en;
  wire   N53, N55, N57, N58, N59, N99, N100, N101, N102, n16, n19, n20, n21,
         n26, n27, n28, n29, n30, n31, n32, n33, n34, n35, n36, n37, n38, n39,
         n40, n41, n42, n43, n44, n45, n46, n47;
  wire   [3:0] current_state;
  wire   [3:0] next_state;

  LATCHX1 \next_state_reg[0]  ( .CLK(N58), .D(N53), .Q(next_state[0]) );
  DFFARX1 \current_state_reg[0]  ( .D(next_state[0]), .CLK(clk), .RSTB(n16),
        .Q(current_state[0]), .QN(n46) );
  LATCHX1 \next_state_reg[2]  ( .CLK(N58), .D(N57), .Q(next_state[2]) );
  DFFARX1 \current_state_reg[2]  ( .D(next_state[2]), .CLK(clk), .RSTB(n16),
        .Q(current_state[2]), .QN(n47) );
  LATCHX1 \next_state_reg[1]  ( .CLK(N58), .D(N55), .Q(next_state[1]) );
  DFFARX1 \current_state_reg[1]  ( .D(next_state[1]), .CLK(clk), .RSTB(n16),
        .Q(current_state[1]), .QN(n44) );
  LATCHX1 \next_state_reg[3]  ( .CLK(N58), .D(N59), .Q(next_state[3]) );
  DFFARX1 \current_state_reg[3]  ( .D(next_state[3]), .CLK(clk), .RSTB(n16),
        .Q(current_state[3]), .QN(n45) );
  LATCHX1 IRinEn_reg ( .CLK(N102), .D(n43), .Q(IRinEn) );
  LATCHX1 ID_en_reg ( .CLK(N102), .D(n42), .Q(ID_en) );
  LATCHX1 PCoutEn_reg ( .CLK(N102), .D(N99), .Q(PCoutEn) );
  LATCHX1 MARinEn_reg ( .CLK(N102), .D(N99), .Q(MARinEn) );
  LATCHX1 memOp_reg ( .CLK(N102), .D(N100), .Q(memOp) );
  LATCHX1 memEn_reg ( .CLK(N102), .D(N100), .Q(memEn) );
  LATCHX1 MDRreadEn_reg ( .CLK(N102), .D(N101), .Q(MDRreadEn) );
  LATCHX1 MDRoutEn_reg ( .CLK(N102), .D(n43), .Q(MDRoutEn) );
  INVX0 U3 ( .IN(reset), .QN(n16) );
  NAND4X0 U38 ( .IN1(n39), .IN2(n40), .IN3(n34), .IN4(n19), .QN(N102) );
  INVX0 U39 ( .IN(N100), .QN(n19) );
  NAND2X0 U40 ( .IN1(n44), .IN2(n47), .QN(n39) );
  NAND3X0 U41 ( .IN1(n26), .IN2(n20), .IN3(n28), .QN(N100) );
  INVX0 U42 ( .IN(N101), .QN(n20) );
  NAND3X0 U43 ( .IN1(n47), .IN2(n45), .IN3(n46), .QN(n34) );
  NAND3X0 U44 ( .IN1(n26), .IN2(n27), .IN3(n28), .QN(N99) );
  NAND3X0 U45 ( .IN1(n46), .IN2(n44), .IN3(n21), .QN(n28) );
  NAND3X0 U46 ( .IN1(n31), .IN2(n26), .IN3(n32), .QN(N57) );
  NAND2X0 U47 ( .IN1(n21), .IN2(n44), .QN(n32) );
  OR2X1 U48 ( .IN1(n34), .IN2(n44), .Q(n27) );
  OR2X1 U49 ( .IN1(n42), .IN2(n43), .Q(N59) );
  INVX0 U50 ( .IN(n40), .QN(n21) );
  NAND4X0 U51 ( .IN1(n30), .IN2(n29), .IN3(n37), .IN4(n45), .QN(N58) );
  NOR2X0 U52 ( .IN1(start), .IN2(reset), .QN(n37) );
  NAND2X0 U53 ( .IN1(current_state[2]), .IN2(n45), .QN(n40) );
  NAND4X0 U54 ( .IN1(current_state[0]), .IN2(current_state[1]), .IN3(n47),
        .IN4(n45), .QN(n26) );
  AND3X1 U55 ( .IN1(n30), .IN2(n29), .IN3(current_state[3]), .Q(n42) );
  XNOR2X1 U56 ( .IN1(current_state[2]), .IN2(n38), .Q(n30) );
  NAND3X0 U57 ( .IN1(current_state[1]), .IN2(n46), .IN3(n21), .QN(n31) );
  AOI22X1 U58 ( .IN1(current_state[0]), .IN2(current_state[1]), .IN3(
        current_state[2]), .IN4(n38), .QN(n29) );
  NAND2X0 U59 ( .IN1(n31), .IN2(n41), .QN(N101) );
  NAND3X0 U60 ( .IN1(current_state[0]), .IN2(n44), .IN3(n21), .QN(n41) );
  XNOR2X1 U61 ( .IN1(n46), .IN2(current_state[1]), .Q(n38) );
  NOR3X0 U62 ( .IN1(n29), .IN2(current_state[3]), .IN3(n30), .QN(n43) );
  NAND3X0 U63 ( .IN1(n31), .IN2(n27), .IN3(n33), .QN(N55) );
  NAND3X0 U64 ( .IN1(n44), .IN2(n45), .IN3(current_state[0]), .QN(n33) );
  NAND3X0 U65 ( .IN1(n31), .IN2(n34), .IN3(n35), .QN(N53) );
  NAND3X0 U66 ( .IN1(n46), .IN2(n44), .IN3(n36), .QN(n35) );
  AO21X1 U67 ( .IN1(MFC), .IN2(n45), .IN3(n47), .Q(n36) );
endmodule
```

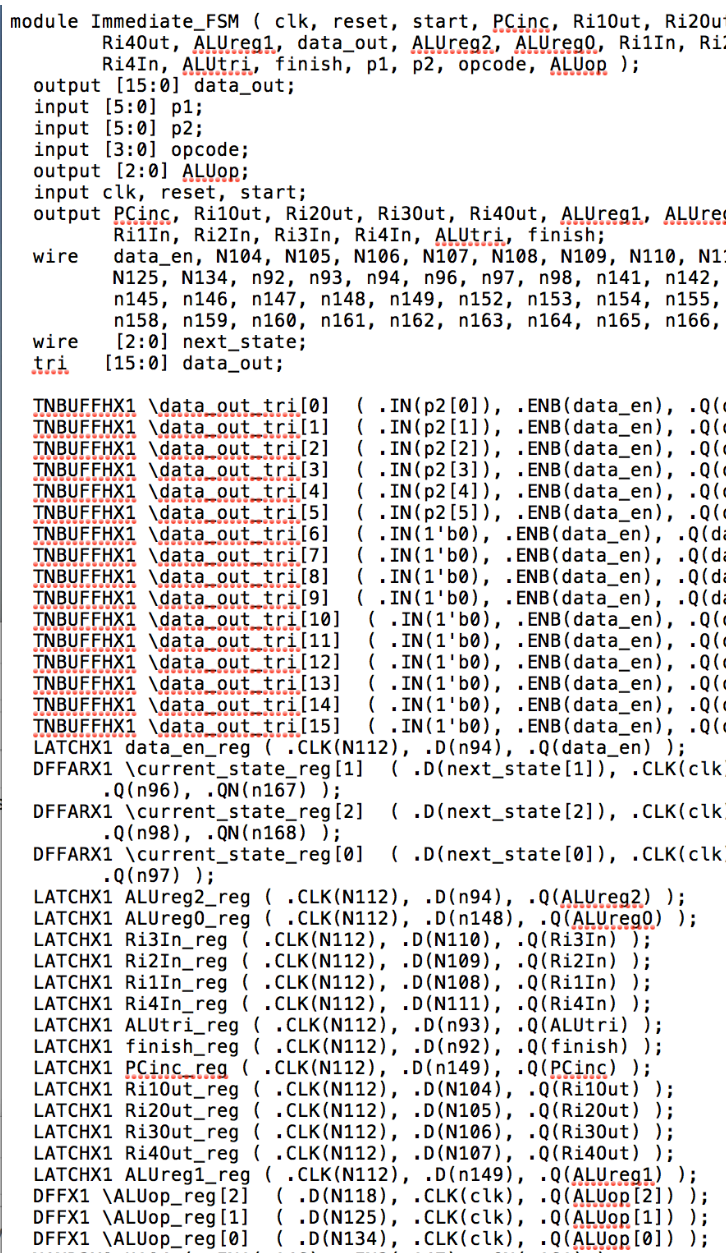
- Attempted Simulation



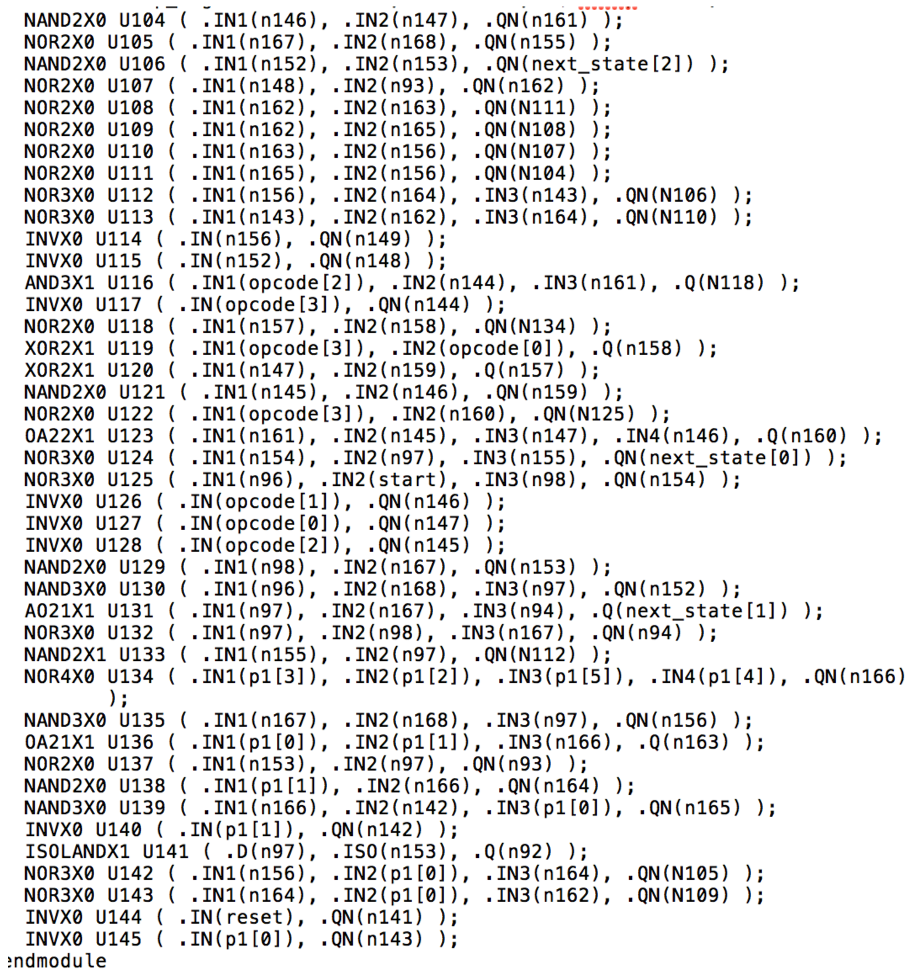
- R-type FSM

```
module Register_FSM ( clk, reset, start, PCinc, Ri1Out, Ri2Out, Ri3Out, Ri4Out,
        ALUreg1, Rj1Out, Rj2Out, Rj3Out, Rj4Out, ALUreg2, ALUreg0, Ri1In,
        Ri2In, Ri3In, Ri4In, ALUtri, finish, p1, p2, opcode, ALUop );
  input [5:0] p1;
  input [5:0] p2;
  input [3:0] opcode;
  output [2:0] ALUop;
  input clk, reset, start;
  output PCinc, Ri1Out, Ri2Out, Ri3Out, Ri4Out, ALUreg1, Rj1Out, Rj2Out,
         Rj3Out, Rj4Out, ALUreg2, ALUreg0, Ri1In, Ri2In, Ri3In, Ri4In, ALUtri,
         finish;
  wire   N105, N106, N107, N108, N109, N110, N111, N112, N113, N114, N115,
         N116, N117, N123, N130, N139, n27, n63, n65, n118, n119, n120, n121,
         n122, n123, n124, n125, n126, n127, n128, n129, n130, n134, n135,
         n136, n137, n138, n139, n140, n141, n142, n143, n144, n145, n146,
         n147, n148, n149, n150, n151, n152, n153, n154, n155, n156;
  wire   [2:0] next_state;

  NAND4X1 U70 ( .IN1(n144), .IN2(n138), .IN3(n135), .IN4(n154), .QN(N117) );
  DFFARX1 \current_state_reg[2]  ( .D(next_state[2]), .CLK(clk), .RSTB(n118),
        .Q(n152), .QN(n155) );
  DFFARX1 \current_state_reg[0]  ( .D(next_state[0]), .CLK(clk), .RSTB(n118),
        .Q(n154), .QN(n65) );
  DFFARX1 \current_state_reg[1]  ( .D(next_state[1]), .CLK(clk), .RSTB(n118),
        .Q(n156), .QN(n63) );
  LATCHX1 Rj2Out_reg ( .CLK(N117), .D(N110), .Q(Rj2Out) );
  LATCHX1 Ri2In_reg ( .CLK(N117), .D(N114), .Q(Ri2In) );
  LATCHX1 Rj4Out_reg ( .CLK(N117), .D(N112), .Q(Rj4Out) );
  LATCHX1 finish_reg ( .CLK(N117), .D(n27), .Q(finish) );
  LATCHX1 Ri2Out_reg ( .CLK(N117), .D(N106), .Q(Ri2Out) );
  LATCHX1 Rj1Out_reg ( .CLK(N117), .D(N109), .Q(Rj1Out) );
  DFFX1 \ALUop_reg[2]  ( .D(N123), .CLK(clk), .Q(ALUop[2]) );
  DFFX1 \ALUop_reg[1]  ( .D(N130), .CLK(clk), .Q(ALUop[1]) );
  DFFX1 \ALUop_reg[0]  ( .D(N139), .CLK(clk), .Q(ALUop[0]) );
  LATCHX1 Rj3Out_reg ( .CLK(N117), .D(N111), .Q(Rj3Out) );
  LATCHX1 Ri1In_reg ( .CLK(N117), .D(N113), .Q(Ri1In) );
  LATCHX1 ALUreg1_reg ( .CLK(N117), .D(n128), .Q(ALUreg1) );
  LATCHX1 Ri4In_reg ( .CLK(N117), .D(N116), .Q(Ri4In) );
  LATCHX1 Ri3In_reg ( .CLK(N117), .D(N115), .Q(Ri3In) );
  LATCHX1 ALUreg2_reg ( .CLK(N117), .D(n127), .Q(ALUreg2) );
  LATCHX1 ALUreg0_reg ( .CLK(N117), .D(n130), .Q(ALUreg0) );
  LATCHX1 ALUtri_reg ( .CLK(N117), .D(n153), .Q(ALUtri) );
  LATCHX1 PCinc_reg ( .CLK(N117), .D(n128), .Q(PCinc) );
  LATCHX1 Ri1Out_reg ( .CLK(N117), .D(N105), .Q(Ri1Out) );
  LATCHX1 Ri3Out_reg ( .CLK(N117), .D(N107), .Q(Ri3Out) );
  LATCHX1 Ri4Out_reg ( .CLK(N117), .D(N108), .Q(Ri4Out) );
  NOR2X0 U112 ( .IN1(n153), .IN2(n130), .QN(n144) );
  INVX0 U113 ( .IN(n134), .QN(n130) );
  NAND2X0 U114 ( .IN1(n125), .IN2(n126), .QN(n143) );
  NOR2X0 U115 ( .IN1(n135), .IN2(n154), .QN(n153) );
  NAND3X0 U116 ( .IN1(n154), .IN2(n155), .IN3(n156), .QN(n134) );
  NAND2X0 U117 ( .IN1(n134), .IN2(n135), .QN(next_state[2]) );
  INVX0 U118 ( .IN(n136), .QN(n127) );
  NOR2X0 U119 ( .IN1(n144), .IN2(n145), .QN(N116) );
  NOR2X0 U120 ( .IN1(n144), .IN2(n147), .QN(N113) );
  NOR2X0 U121 ( .IN1(n145), .IN2(n138), .QN(N108) );
  NOR2X0 U122 ( .IN1(n147), .IN2(n138), .QN(N105) );
  NOR2X0 U123 ( .IN1(n149), .IN2(n122), .QN(N111) );
  NOR3X0 U124 ( .IN1(n138), .IN2(n146), .IN3(n120), .QN(N107) );
  NOR3X0 U125 ( .IN1(n120), .IN2(n144), .IN3(n146), .QN(N115) );
  INVX0 U126 ( .IN(n150), .QN(n121) );
  INVX0 U127 ( .IN(n138), .QN(n128) );
  NAND3X0 U128 ( .IN1(n129), .IN2(n136), .IN3(n137), .QN(next_state[0])
  INVX0 U129 ( .IN(n153), .QN(n129) );
  NAND3X0 U130 ( .IN1(n65), .IN2(n63), .IN3(start), .QN(n137) );
  NOR2X0 U131 ( .IN1(n139), .IN2(n140), .QN(N139) );
  XNOR2X1 U132 ( .IN1(opcode[3]), .IN2(n126), .Q(n140) );
  XNOR2X1 U133 ( .IN1(opcode[0]), .IN2(n141), .Q(n139) );
  NAND2X0 U134 ( .IN1(n123), .IN2(n125), .QN(n141) );
  NOR2X0 U135 ( .IN1(opcode[3]), .IN2(n142), .QN(N130) );
  OA22X1 U136 ( .IN1(n143), .IN2(n123), .IN3(n126), .IN4(n125), .Q(n142) );
  NOR3X0 U137 ( .IN1(n123), .IN2(opcode[3]), .IN3(n124), .QN(N123) );
  INVX0 U138 ( .IN(n143), .QN(n124) );
  INVX0 U139 ( .IN(opcode[1]), .QN(n125) );
  INVX0 U140 ( .IN(opcode[0]), .QN(n126) );
  INVX0 U141 ( .IN(opcode[2]), .QN(n123) );
  NAND2X0 U142 ( .IN1(n152), .IN2(n63), .QN(n135) );
  NAND3X0 U143 ( .IN1(n156), .IN2(n155), .IN3(n65), .QN(n136) );
  AO21X1 U144 ( .IN1(n63), .IN2(n154), .IN3(n127), .Q(next_state[1]) );
  NOR4X0 U145 ( .IN1(p2[3]), .IN2(p2[2]), .IN3(p2[5]), .IN4(p2[4]), .QN(n150)
        );
  NOR4X0 U146 ( .IN1(p1[3]), .IN2(p1[2]), .IN3(p1[5]), .IN4(p1[4]), .QN(n151)
        );
  NAND3X0 U147 ( .IN1(n154), .IN2(n155), .IN3(n63), .QN(n138) );
  OA21X1 U148 ( .IN1(p1[0]), .IN2(p1[1]), .IN3(n151), .Q(n145) );
  NAND2X0 U149 ( .IN1(p1[1]), .IN2(n151), .QN(n146) );
  NAND3X0 U150 ( .IN1(n151), .IN2(n119), .IN3(p1[0]), .QN(n147) );
  INVX0 U151 ( .IN(p1[1]), .QN(n119) );
  NAND3X0 U152 ( .IN1(n127), .IN2(n150), .IN3(p2[1]), .QN(n149) );
  NOR2X0 U153 ( .IN1(n65), .IN2(n135), .QN(n27) );
  OA21X1 U154 ( .IN1(n148), .IN2(n121), .IN3(n127), .Q(N112) );
  NOR2X0 U155 ( .IN1(p2[1]), .IN2(p2[0]), .QN(n148) );
  NOR4X0 U156 ( .IN1(p2[1]), .IN2(n121), .IN3(n136), .IN4(n122), .QN(N109) );
  NOR3X0 U157 ( .IN1(n138), .IN2(p1[0]), .IN3(n146), .QN(N106) );
  NOR3X0 U158 ( .IN1(n146), .IN2(p1[0]), .IN3(n144), .QN(N114) );
  NOR2X0 U159 ( .IN1(p2[0]), .IN2(n149), .QN(N110) );
  INVX0 U160 ( .IN(reset), .QN(n118) );
  INVX0 U161 ( .IN(p2[0]), .QN(n122) );
  INVX0 U162 ( .IN(p1[0]), .QN(n120) );
endmodule
```

- Immediate FSM



```
module Immediate_FSM ( clk, reset, start, PCinc, Ri1Out, Ri2Out, Ri3Out,
        Ri4Out, ALUreg1, data_out, ALUreg2, ALUreg0, Ri1In, Ri2In, Ri3In,
        Ri4In, ALUtri, finish, p1, p2, opcode, ALUop );
  output [15:0] data_out;
  input [5:0] p1;
  input [5:0] p2;
  input [3:0] opcode;
  output [2:0] ALUop;
  input clk, reset, start;
  output PCinc, Ri1Out, Ri2Out, Ri3Out, Ri4Out, ALUreg1, ALUreg2, ALUreg0,
         Ri1In, Ri2In, Ri3In, Ri4In, ALUtri, finish;
  wire   data_en, N104, N105, N106, N107, N108, N109, N110, N111, N112, N118,
         N125, N134, n92, n93, n94, n96, n97, n98, n141, n142, n143, n144,
         n145, n146, n147, n148, n149, n152, n153, n154, n155, n156, n157,
         n158, n159, n160, n161, n162, n163, n164, n165, n166, n167, n168;
  wire   [2:0] next_state;
  tri   [15:0] data_out;

  TNBUFFHX1 \data_out_tri[0]  ( .IN(p2[0]), .ENB(data_en), .Q(data_out[0]) );
  TNBUFFHX1 \data_out_tri[1]  ( .IN(p2[1]), .ENB(data_en), .Q(data_out[1]) );
  TNBUFFHX1 \data_out_tri[2]  ( .IN(p2[2]), .ENB(data_en), .Q(data_out[2]) );
  TNBUFFHX1 \data_out_tri[3]  ( .IN(p2[3]), .ENB(data_en), .Q(data_out[3]) );
  TNBUFFHX1 \data_out_tri[4]  ( .IN(p2[4]), .ENB(data_en), .Q(data_out[4]) );
  TNBUFFHX1 \data_out_tri[5]  ( .IN(p2[5]), .ENB(data_en), .Q(data_out[5]) );
  TNBUFFHX1 \data_out_tri[6]  ( .IN(1'b0), .ENB(data_en), .Q(data_out[6]) );
  TNBUFFHX1 \data_out_tri[7]  ( .IN(1'b0), .ENB(data_en), .Q(data_out[7]) );
  TNBUFFHX1 \data_out_tri[8]  ( .IN(1'b0), .ENB(data_en), .Q(data_out[8]) );
  TNBUFFHX1 \data_out_tri[9]  ( .IN(1'b0), .ENB(data_en), .Q(data_out[9]) );
  TNBUFFHX1 \data_out_tri[10]  ( .IN(1'b0), .ENB(data_en), .Q(data_out[10]) );
  TNBUFFHX1 \data_out_tri[11]  ( .IN(1'b0), .ENB(data_en), .Q(data_out[11]) );
  TNBUFFHX1 \data_out_tri[12]  ( .IN(1'b0), .ENB(data_en), .Q(data_out[12]) );
  TNBUFFHX1 \data_out_tri[13]  ( .IN(1'b0), .ENB(data_en), .Q(data_out[13]) );
  TNBUFFHX1 \data_out_tri[14]  ( .IN(1'b0), .ENB(data_en), .Q(data_out[14]) );
  TNBUFFHX1 \data_out_tri[15]  ( .IN(1'b0), .ENB(data_en), .Q(data_out[15]) );
  LATCHX1 data_en_reg ( .CLK(N112), .D(n94), .Q(data_en) );
  DFFARX1 \current_state_reg[1]  ( .D(next_state[1]), .CLK(clk), .RSTB(n141),
        .Q(n96), .QN(n167) );
  DFFARX1 \current_state_reg[2]  ( .D(next_state[2]), .CLK(clk), .RSTB(n141),
        .Q(n98), .QN(n168) );
  DFFARX1 \current_state_reg[0]  ( .D(next_state[0]), .CLK(clk), .RSTB(n141),
        .Q(n97) );
  LATCHX1 ALUreg2_reg ( .CLK(N112), .D(n94), .Q(ALUreg2) );
  LATCHX1 ALUreg0_reg ( .CLK(N112), .D(n148), .Q(ALUreg0) );
  LATCHX1 Ri3In_reg ( .CLK(N112), .D(N110), .Q(Ri3In) );
  LATCHX1 Ri2In_reg ( .CLK(N112), .D(N109), .Q(Ri2In) );
  LATCHX1 Ri1In_reg ( .CLK(N112), .D(N108), .Q(Ri1In) );
  LATCHX1 Ri4In_reg ( .CLK(N112), .D(N111), .Q(Ri4In) );
  LATCHX1 ALUtri_reg ( .CLK(N112), .D(n93), .Q(ALUtri) );
  LATCHX1 finish_reg ( .CLK(N112), .D(n92), .Q(finish) );
  LATCHX1 PCinc_reg ( .CLK(N112), .D(n149), .Q(PCinc) );
  LATCHX1 Ri1Out_reg ( .CLK(N112), .D(N104), .Q(Ri1Out) );
  LATCHX1 Ri2Out_reg ( .CLK(N112), .D(N105), .Q(Ri2Out) );
  LATCHX1 Ri3Out_reg ( .CLK(N112), .D(N106), .Q(Ri3Out) );
  LATCHX1 Ri4Out_reg ( .CLK(N112), .D(N107), .Q(Ri4Out) );
  LATCHX1 ALUreg1_reg ( .CLK(N112), .D(n149), .Q(ALUreg1) );
  DFFX1 \ALUop_reg[2]  ( .D(N118), .CLK(clk), .Q(ALUop[2]) );
  DFFX1 \ALUop_reg[1]  ( .D(N125), .CLK(clk), .Q(ALUop[1]) );
  DFFX1 \ALUop_reg[0]  ( .D(N134), .CLK(clk), .Q(ALUop[0]) );
```



```
  NAND2X0 U104 ( .IN1(n146), .IN2(n147), .QN(n161) );
  NOR2X0 U105 ( .IN1(n167), .IN2(n168), .QN(n155) );
  NAND2X0 U106 ( .IN1(n152), .IN2(n153), .QN(next_state[2]) );
  NOR2X0 U107 ( .IN1(n148), .IN2(n93), .QN(n162) );
  NOR2X0 U108 ( .IN1(n162), .IN2(n163), .QN(N111) );
  NOR2X0 U109 ( .IN1(n162), .IN2(n165), .QN(N108) );
  NOR2X0 U110 ( .IN1(n163), .IN2(n156), .QN(N107) );
  NOR2X0 U111 ( .IN1(n165), .IN2(n156), .QN(N104) );
  NOR3X0 U112 ( .IN1(n156), .IN2(n164), .IN3(n143), .QN(N106) );
  NOR3X0 U113 ( .IN1(n143), .IN2(n162), .IN3(n164), .QN(N110) );
  INVX0 U114 ( .IN(n156), .QN(n149) );
  INVX0 U115 ( .IN(n152), .QN(n148) );
  AND3X1 U116 ( .IN1(opcode[2]), .IN2(n144), .IN3(n161), .Q(N118) );
  INVX0 U117 ( .IN(opcode[3]), .QN(n144) );
  NOR2X0 U118 ( .IN1(n157), .IN2(n158), .QN(N134) );
  XOR2X1 U119 ( .IN1(opcode[3]), .IN2(opcode[0]), .Q(n158) );
  XOR2X1 U120 ( .IN1(n147), .IN2(n159), .Q(n157) );
  NAND2X0 U121 ( .IN1(n145), .IN2(n146), .QN(n159) );
  NOR2X0 U122 ( .IN1(opcode[3]), .IN2(n160), .QN(N125) );
  OA22X1 U123 ( .IN1(n161), .IN2(n145), .IN3(n147), .IN4(n146), .Q(n160) );
  NOR3X0 U124 ( .IN1(n154), .IN2(n97), .IN3(n155), .QN(next_state[0]) );
  NOR3X0 U125 ( .IN1(n96), .IN2(start), .IN3(n98), .QN(n154) );
  INVX0 U126 ( .IN(opcode[1]), .QN(n146) );
  INVX0 U127 ( .IN(opcode[0]), .QN(n147) );
  INVX0 U128 ( .IN(opcode[2]), .QN(n145) );
  NAND2X0 U129 ( .IN1(n98), .IN2(n167), .QN(n153) );
  NAND3X0 U130 ( .IN1(n96), .IN2(n168), .IN3(n97), .QN(n152) );
  AO21X1 U131 ( .IN1(n97), .IN2(n167), .IN3(n94), .Q(next_state[1]) );
  NOR3X0 U132 ( .IN1(n97), .IN2(n98), .IN3(n167), .QN(n94) );
  NAND2X1 U133 ( .IN1(n155), .IN2(n97), .QN(N112) );
  NOR4X0 U134 ( .IN1(p1[3]), .IN2(p1[2]), .IN3(p1[5]), .IN4(p1[4]), .QN(n166)
        );
  NAND3X0 U135 ( .IN1(n167), .IN2(n168), .IN3(n97), .QN(n156) );
  OA21X1 U136 ( .IN1(p1[0]), .IN2(p1[1]), .IN3(n166), .Q(n163) );
  NOR2X0 U137 ( .IN1(n153), .IN2(n97), .QN(n93) );
  NAND2X0 U138 ( .IN1(p1[1]), .IN2(n166), .QN(n164) );
  NAND3X0 U139 ( .IN1(n166), .IN2(n142), .IN3(p1[0]), .QN(n165) );
  INVX0 U140 ( .IN(p1[1]), .QN(n142) );
  ISOLANDX1 U141 ( .D(n97), .ISO(n153), .Q(n92) );
  NOR3X0 U142 ( .IN1(n156), .IN2(p1[0]), .IN3(n164), .QN(N105) );
  NOR3X0 U143 ( .IN1(n164), .IN2(p1[0]), .IN3(n162), .QN(N109) );
  INVX0 U144 ( .IN(reset), .QN(n141) );
  INVX0 U145 ( .IN(p1[0]), .QN(n143) );
endmodule
```

- Move FSM

```
module move_FSM ( clk, reset, start, PCinc, Rj1Out, Rj2Out, Rj3Out, Rj4Out,
        Ri1In, Ri2In, Ri3In, Ri4In, p1, p2, finish );
  input [5:0] p1;
  input [5:0] p2;
  input clk, reset, start;
  output PCinc, Rj1Out, Rj2Out, Rj3Out, Rj4Out, Ri1In, Ri2In, Ri3In, Ri4In,
         finish;
  wire   N57, N58, N59, N60, N61, N62, N63, N64, n32, n57, n58, n59, n60, n61,
         n62, n63, n64, n65, n66, n67, n68, n69, n70, n71, n72;
  wire   [1:0] next_state;

  DFFARX1 \current_state_reg[1]  ( .D(next_state[1]), .CLK(clk), .RSTB(n57),
        .Q(n71) );
  DFFARX1 \current_state_reg[0]  ( .D(next_state[0]), .CLK(clk), .RSTB(n57),
        .QN(n32) );
  LATCHX1 PCinc_reg ( .CLK(1'b1), .D(n72), .Q(PCinc) );
  LATCHX1 Rj2Out_reg ( .CLK(1'b1), .D(N58), .Q(Rj2Out) );
  LATCHX1 Ri2In_reg ( .CLK(1'b1), .D(N62), .Q(Ri2In) );
  LATCHX1 Ri4In_reg ( .CLK(1'b1), .D(N64), .Q(Ri4In) );
  LATCHX1 Rj4Out_reg ( .CLK(1'b1), .D(N60), .Q(Rj4Out) );
  LATCHX1 Ri1In_reg ( .CLK(1'b1), .D(N61), .Q(Ri1In) );
  LATCHX1 Rj1Out_reg ( .CLK(1'b1), .D(N57), .Q(Rj1Out) );
  LATCHX1 finish_reg ( .CLK(1'b1), .D(n62), .Q(finish) );
  LATCHX1 Rj3Out_reg ( .CLK(1'b1), .D(N59), .Q(Rj3Out) );
  LATCHX1 Ri3In_reg ( .CLK(1'b1), .D(N63), .Q(Ri3In) );
  NOR2X0 U58 ( .IN1(n66), .IN2(n59), .QN(N63) );
  NOR2X0 U59 ( .IN1(n69), .IN2(n61), .QN(N59) );
  INVX0 U60 ( .IN(n70), .QN(n60) );
  INVX0 U61 ( .IN(n67), .QN(n58) );
  INVX0 U62 ( .IN(n72), .QN(n63) );
  INVX0 U63 ( .IN(n64), .QN(n62) );
  NAND2X0 U64 ( .IN1(n64), .IN2(n63), .QN(next_state[1]) );
  NOR4X0 U65 ( .IN1(p1[3]), .IN2(p1[2]), .IN3(p1[5]), .IN4(p1[4]), .QN(n67) );
  NOR4X0 U66 ( .IN1(p2[3]), .IN2(p2[2]), .IN3(p2[5]), .IN4(p2[4]), .QN(n70) );
  NAND3X0 U67 ( .IN1(n72), .IN2(n67), .IN3(p1[1]), .QN(n66) );
  NAND3X0 U68 ( .IN1(n70), .IN2(n72), .IN3(p2[1]), .QN(n69) );
  OA21X1 U69 ( .IN1(n68), .IN2(n60), .IN3(n72), .Q(N60) );
  NOR2X0 U70 ( .IN1(p2[1]), .IN2(p2[0]), .QN(n68) );
  OA21X1 U71 ( .IN1(n65), .IN2(n58), .IN3(n72), .Q(N64) );
  NOR2X0 U72 ( .IN1(p1[1]), .IN2(p1[0]), .QN(n65) );
  NOR4X0 U73 ( .IN1(p1[1]), .IN2(n58), .IN3(n63), .IN4(n59), .QN(N61) );
  NOR4X0 U74 ( .IN1(p2[1]), .IN2(n63), .IN3(n60), .IN4(n61), .QN(N57) );
  NOR2X0 U75 ( .IN1(p1[0]), .IN2(n66), .QN(N62) );
  NOR2X0 U76 ( .IN1(p2[0]), .IN2(n69), .QN(N58) );
  AO21X1 U77 ( .IN1(start), .IN2(n32), .IN3(n62), .Q(next_state[0]) );
  INVX0 U78 ( .IN(p2[0]), .QN(n61) );
  INVX0 U79 ( .IN(p1[0]), .QN(n59) );
  NOR2X0 U80 ( .IN1(n71), .IN2(n32), .QN(n72) );
  NAND2X0 U81 ( .IN1(n71), .IN2(n32), .QN(n64) );
  INVX0 U82 ( .IN(reset), .QN(n57) );
endmodule
```

- MoveImm FSM

```
module moveImm_FSM ( clk, reset, start, PCinc, data_out, Ri1In, Ri2In, Ri3In,
        Ri4In, p1, p2, finish );
  output [15:0] data_out;
  input [5:0] p1;
  input [5:0] p2;
  input clk, reset, start;
  output PCinc, Ri1In, Ri2In, Ri3In, Ri4In, finish;
  wire   dataTri, N59, N60, N61, N62, n65, n80, n81, n82, n83, n84, n86, n87,
         n88, n89, n90, n91, n92;
  wire   [1:0] next_state;
  tri   [15:0] data_out;

  TNBUFFHX1 \data_out_tri[0]  ( .IN(p2[0]), .ENB(dataTri), .Q(data_out[0]) );
  TNBUFFHX1 \data_out_tri[1]  ( .IN(p2[1]), .ENB(dataTri), .Q(data_out[1]) );
  TNBUFFHX1 \data_out_tri[2]  ( .IN(p2[2]), .ENB(dataTri), .Q(data_out[2]) );
  TNBUFFHX1 \data_out_tri[3]  ( .IN(p2[3]), .ENB(dataTri), .Q(data_out[3]) );
  TNBUFFHX1 \data_out_tri[4]  ( .IN(p2[4]), .ENB(dataTri), .Q(data_out[4]) );
  TNBUFFHX1 \data_out_tri[5]  ( .IN(p2[5]), .ENB(dataTri), .Q(data_out[5]) );
  TNBUFFHX1 \data_out_tri[6]  ( .IN(1'b0), .ENB(dataTri), .Q(data_out[6]) );
  TNBUFFHX1 \data_out_tri[7]  ( .IN(1'b0), .ENB(dataTri), .Q(data_out[7]) );
  TNBUFFHX1 \data_out_tri[8]  ( .IN(1'b0), .ENB(dataTri), .Q(data_out[8]) );
  TNBUFFHX1 \data_out_tri[9]  ( .IN(1'b0), .ENB(dataTri), .Q(data_out[9]) );
  TNBUFFHX1 \data_out_tri[10]  ( .IN(1'b0), .ENB(dataTri), .Q(data_out[10]) );
  TNBUFFHX1 \data_out_tri[11]  ( .IN(1'b0), .ENB(dataTri), .Q(data_out[11]) );
  TNBUFFHX1 \data_out_tri[12]  ( .IN(1'b0), .ENB(dataTri), .Q(data_out[12]) );
  TNBUFFHX1 \data_out_tri[13]  ( .IN(1'b0), .ENB(dataTri), .Q(data_out[13]) );
  TNBUFFHX1 \data_out_tri[14]  ( .IN(1'b0), .ENB(dataTri), .Q(data_out[14]) );
  TNBUFFHX1 \data_out_tri[15]  ( .IN(1'b0), .ENB(dataTri), .Q(data_out[15]) );
  DFFARX1 \current_state_reg[0]  ( .D(next_state[0]), .CLK(clk), .RSTB(n80),
        .Q(n65), .QN(n92) );
  DFFARX1 \current_state_reg[1]  ( .D(next_state[1]), .CLK(clk), .RSTB(n80),
        .Q(n90) );
  LATCHX1 dataTri_reg ( .CLK(1'b1), .D(n91), .Q(dataTri) );
  LATCHX1 PCinc_reg ( .CLK(1'b1), .D(n91), .Q(PCinc) );
  LATCHX1 Ri2In_reg ( .CLK(1'b1), .D(N60), .Q(Ri2In) );
  LATCHX1 Ri4In_reg ( .CLK(1'b1), .D(N62), .Q(Ri4In) );
  LATCHX1 Ri1In_reg ( .CLK(1'b1), .D(N59), .Q(Ri1In) );
  LATCHX1 finish_reg ( .CLK(n83), .D(n84), .Q(finish) );
  LATCHX1 Ri3In_reg ( .CLK(1'b1), .D(N61), .Q(Ri3In) );
  NOR2X0 U51 ( .IN1(n88), .IN2(n82), .QN(N61) );
  INVX0 U52 ( .IN(n89), .QN(n81) );
  INVX0 U53 ( .IN(n91), .QN(n83) );
  NAND2X0 U54 ( .IN1(n86), .IN2(n83), .QN(next_state[1]) );
  INVX0 U55 ( .IN(n86), .QN(n84) );
  NOR2X0 U56 ( .IN1(n92), .IN2(n90), .QN(n91) );
  NOR4X0 U57 ( .IN1(p1[3]), .IN2(p1[2]), .IN3(p1[5]), .IN4(p1[4]), .QN(n89) );
  NAND3X0 U58 ( .IN1(n89), .IN2(n91), .IN3(p1[1]), .QN(n88) );
  OA21X1 U59 ( .IN1(n87), .IN2(n81), .IN3(n91), .Q(N62) );
  NOR2X0 U60 ( .IN1(p1[1]), .IN2(p1[0]), .QN(n87) );
  NOR4X0 U61 ( .IN1(p1[1]), .IN2(n83), .IN3(n81), .IN4(n82), .QN(N59) );
  NOR2X0 U62 ( .IN1(p1[0]), .IN2(n88), .QN(N60) );
  AO21X1 U63 ( .IN1(start), .IN2(n92), .IN3(n84), .Q(next_state[0]) );
  INVX0 U64 ( .IN(p1[0]), .QN(n82) );
  NAND2X0 U65 ( .IN1(n90), .IN2(n92), .QN(n86) );
  INVX0 U66 ( .IN(reset), .QN(n80) );
endmodule
```

- Store FSM

```
module Store_FSM ( clk, reset, start, MFC, PCinc, Ri1Out, Ri2Out, Ri3Out,
        Ri4Out, MARin, MDRwrite, memEn, memOp, MDRout, Rj1Out, Rj2Out, Rj3Out,
        Rj4Out, p1, p2, finish );
  input [5:0] p1;
  input [5:0] p2;
  input clk, reset, start, MFC;
  output PCinc, Ri1Out, Ri2Out, Ri3Out, Ri4Out, MARin, MDRwrite, memEn, memOp,
         MDRout, Rj1Out, Rj2Out, Rj3Out, Rj4Out, finish;
  wire   N83, N84, N85, N86, N87, N88, N89, N90, N91, n39, n40, n45, n46, n47,
         n80, n81, n82, n83, n84, n85, n87, n90, n91, n92, n93, n94, n95, n96,
         n97, n98, n99, n100, n101, n102, n103;
  wire   [2:0] next_state;
  assign memOp = 1'b0;
  assign MDRout = 1'b0;

  DFFARX1 \current_state_reg[0]  ( .D(next_state[0]), .CLK(clk), .RSTB(n80),
        .Q(n101), .QN(n47) );
  DFFARX1 \current_state_reg[2]  ( .D(next_state[2]), .CLK(clk), .RSTB(n80),
        .Q(n103), .QN(n45) );
  DFFARX1 \current_state_reg[1]  ( .D(next_state[1]), .CLK(clk), .RSTB(n80),
        .Q(n102), .QN(n46) );
  LATCHX1 Ri1Out_reg ( .CLK(N91), .D(N83), .Q(Ri1Out) );
  LATCHX1 Ri2Out_reg ( .CLK(N91), .D(N84), .Q(Ri2Out) );
  LATCHX1 Rj1Out_reg ( .CLK(N91), .D(N87), .Q(Rj1Out) );
  LATCHX1 Ri4Out_reg ( .CLK(N91), .D(N86), .Q(Ri4Out) );
  LATCHX1 Rj2Out_reg ( .CLK(N91), .D(N88), .Q(Rj2Out) );
  LATCHX1 Rj4Out_reg ( .CLK(N91), .D(N90), .Q(Rj4Out) );
  LATCHX1 finish_reg ( .CLK(N91), .D(n39), .Q(finish) );
  LATCHX1 memEn_reg ( .CLK(N91), .D(n40), .Q(memEn) );
  LATCHX1 Ri3Out_reg ( .CLK(N91), .D(N85), .Q(Ri3Out) );
  LATCHX1 Rj3Out_reg ( .CLK(N91), .D(N89), .Q(Rj3Out) );
  LATCHX1 PCinc_reg ( .CLK(N91), .D(n85), .Q(PCinc) );
  LATCHX1 MARin_reg ( .CLK(N91), .D(n85), .Q(MARin) );
  LATCHX1 MDRwrite_reg ( .CLK(N91), .D(n87), .Q(MDRwrite) );
  NOR2X0 U76 ( .IN1(n90), .IN2(n101), .QN(n40) );
  INVX0 U77 ( .IN(n100), .QN(n87) );
  NAND3X1 U78 ( .IN1(n102), .IN2(n101), .IN3(n103), .QN(N91) );
  NOR2X0 U79 ( .IN1(n94), .IN2(n84), .QN(N89) );
  NOR2X0 U80 ( .IN1(n98), .IN2(n82), .QN(N85) );
  INVX0 U81 ( .IN(n95), .QN(n83) );
  INVX0 U82 ( .IN(n99), .QN(n81) );
  INVX0 U83 ( .IN(n96), .QN(n85) );
  AO221X1 U84 ( .IN1(n92), .IN2(start), .IN3(MFC), .IN4(n40), .IN5(n87), .Q(
        next_state[0]) );
  NOR2X0 U85 ( .IN1(n103), .IN2(n101), .QN(n92) );
  NAND2X0 U86 ( .IN1(n46), .IN2(n103), .QN(n90) );
  NAND3X0 U87 ( .IN1(n45), .IN2(n102), .IN3(n47), .QN(n100) );
  AO21X1 U88 ( .IN1(n46), .IN2(n101), .IN3(n87), .Q(next_state[1]) );
  NAND2X0 U89 ( .IN1(n90), .IN2(n91), .QN(next_state[2]) );
  NAND3X0 U90 ( .IN1(n102), .IN2(n101), .IN3(n45), .QN(n91) );
  NOR4X0 U91 ( .IN1(p2[3]), .IN2(p2[2]), .IN3(p2[5]), .IN4(p2[4]), .QN(n95) );
  NOR4X0 U92 ( .IN1(p1[3]), .IN2(p1[2]), .IN3(p1[5]), .IN4(p1[4]), .QN(n99) );
  NAND3X0 U93 ( .IN1(n45), .IN2(n101), .IN3(n46), .QN(n96) );
  NAND3X0 U94 ( .IN1(n85), .IN2(n95), .IN3(p2[1]), .QN(n94) );
  NAND3X0 U95 ( .IN1(n87), .IN2(n99), .IN3(p1[1]), .QN(n98) );
  NOR2X0 U96 ( .IN1(n47), .IN2(n90), .QN(n39) );
  OA21X1 U97 ( .IN1(n93), .IN2(n83), .IN3(n85), .Q(N90) );
  NOR2X0 U98 ( .IN1(p2[1]), .IN2(p2[0]), .QN(n93) );
  OA21X1 U99 ( .IN1(n97), .IN2(n81), .IN3(n87), .Q(N86) );
  NOR2X0 U100 ( .IN1(p1[1]), .IN2(p1[0]), .QN(n97) );
  NOR4X0 U101 ( .IN1(p2[1]), .IN2(n83), .IN3(n96), .IN4(n84), .QN(N87) );
  NOR4X0 U102 ( .IN1(p1[1]), .IN2(n81), .IN3(n100), .IN4(n82), .QN(N83) );
  NOR2X0 U103 ( .IN1(p2[0]), .IN2(n94), .QN(N88) );
  NOR2X0 U104 ( .IN1(p1[0]), .IN2(n98), .QN(N84) );
  INVX0 U105 ( .IN(reset), .QN(n80) );
  INVX0 U106 ( .IN(p2[0]), .QN(n84) );
  INVX0 U107 ( .IN(p1[0]), .QN(n82) );
endmodule
```

- Load FSM

```
module Load_FSM ( clk, reset, start, MFC, PCinc, Ri1Out, Ri2Out, Ri3Out,
        Ri4Out, MARin, MDRread, memEn, memOp, MDRout, Rj1In, Rj2In, Rj3In,
        Rj4In, p1, p2, finish );
  input [5:0] p1;
  input [5:0] p2;
  input clk, reset, start, MFC;
  output PCinc, Ri1Out, Ri2Out, Ri3Out, Ri4Out, MARin, MDRread, memEn, memOp,
         MDRout, Rj1In, Rj2In, Rj3In, Rj4In, finish;
  wire   N83, N84, N85, N86, N87, N88, N89, N90, N91, N92, n37, n38, n42, n44,
         n78, n79, n80, n81, n82, n83, n84, n88, n89, n90, n91, n92, n93, n94,
         n95, n96, n97, n98, n99, n100;
  wire   [2:0] next_state;

  DFFARX1 \current_state_reg[1]  ( .D(next_state[1]), .CLK(clk), .RSTB(n78),
        .Q(n97), .QN(n98) );
  DFFARX1 \current_state_reg[2]  ( .D(next_state[2]), .CLK(clk), .RSTB(n78),
        .Q(n99), .QN(n42) );
  DFFARX1 \current_state_reg[0]  ( .D(next_state[0]), .CLK(clk), .RSTB(n78),
        .Q(n100), .QN(n44) );
  LATCHX1 Ri2Out_reg ( .CLK(N92), .D(N84), .Q(Ri2Out) );
  LATCHX1 Ri3Out_reg ( .CLK(N92), .D(N85), .Q(Ri3Out) );
  LATCHX1 Rj1In_reg ( .CLK(N92), .D(N88), .Q(Rj1In) );
  LATCHX1 MDRout_reg ( .CLK(N92), .D(n83), .Q(MDRout) );
  LATCHX1 memOp_reg ( .CLK(N92), .D(N87), .Q(memOp) );
  LATCHX1 memEn_reg ( .CLK(N92), .D(N87), .Q(memEn) );
  LATCHX1 MDRread_reg ( .CLK(N92), .D(n38), .Q(MDRread) );
  LATCHX1 Ri4Out_reg ( .CLK(N92), .D(N86), .Q(Ri4Out) );
  LATCHX1 Rj2In_reg ( .CLK(N92), .D(N89), .Q(Rj2In) );
  LATCHX1 Rj3In_reg ( .CLK(N92), .D(N90), .Q(Rj3In) );
  LATCHX1 Rj4In_reg ( .CLK(N92), .D(N91), .Q(Rj4In) );
  LATCHX1 finish_reg ( .CLK(N92), .D(n37), .Q(finish) );
  LATCHX1 PCinc_reg ( .CLK(N92), .D(n84), .Q(PCinc) );
  LATCHX1 MARin_reg ( .CLK(N92), .D(n84), .Q(MARin) );
  LATCHX1 Ri1Out_reg ( .CLK(N92), .D(N83), .Q(Ri1Out) );
  NOR2X0 U70 ( .IN1(n98), .IN2(n99), .QN(N87) );
  AO21X1 U71 ( .IN1(n99), .IN2(n98), .IN3(n38), .Q(next_state[2]) );
  INVX0 U72 ( .IN(n92), .QN(n83) );
  AND3X1 U73 ( .IN1(n99), .IN2(n98), .IN3(n100), .Q(n37) );
  NOR2X0 U74 ( .IN1(n90), .IN2(n82), .QN(N90) );
  NOR2X0 U75 ( .IN1(n94), .IN2(n80), .QN(N85) );
  INVX0 U76 ( .IN(n95), .QN(n79) );
  INVX0 U77 ( .IN(n91), .QN(n81) );
  INVX0 U78 ( .IN(n96), .QN(n84) );
  AO21X1 U79 ( .IN1(n44), .IN2(n88), .IN3(n83), .Q(next_state[0]) );
  AO22X1 U80 ( .IN1(start), .IN2(n98), .IN3(MFC), .IN4(N87), .Q(n88) );
  AO22X1 U81 ( .IN1(n100), .IN2(n98), .IN3(n44), .IN4(N87), .Q(next_state[1])
        );
  NAND3X0 U82 ( .IN1(n99), .IN2(n98), .IN3(n44), .QN(n92) );
  ISOLANDX1 U83 ( .D(N87), .ISO(n44), .Q(n38) );
  NAND3X1 U84 ( .IN1(n99), .IN2(n100), .IN3(n97), .QN(N92) );
  NOR4X0 U85 ( .IN1(p2[3]), .IN2(p2[2]), .IN3(p2[5]), .IN4(p2[4]), .QN(n91) );
  NOR4X0 U86 ( .IN1(p1[3]), .IN2(p1[2]), .IN3(p1[5]), .IN4(p1[4]), .QN(n95) );
  NAND3X0 U87 ( .IN1(n100), .IN2(n98), .IN3(n42), .QN(n96) );
  NAND3X0 U88 ( .IN1(n83), .IN2(n91), .IN3(p2[1]), .QN(n90) );
  NAND3X0 U89 ( .IN1(n84), .IN2(n95), .IN3(p1[1]), .QN(n94) );
  OA21X1 U90 ( .IN1(n89), .IN2(n81), .IN3(n83), .Q(N91) );
  NOR2X0 U91 ( .IN1(p2[1]), .IN2(p2[0]), .QN(n89) );
  OA21X1 U92 ( .IN1(n93), .IN2(n79), .IN3(n84), .Q(N86) );
  NOR2X0 U93 ( .IN1(p1[1]), .IN2(p1[0]), .QN(n93) );
  NOR4X0 U94 ( .IN1(p1[1]), .IN2(n79), .IN3(n96), .IN4(n80), .QN(N83) );
  NOR4X0 U95 ( .IN1(p2[1]), .IN2(n81), .IN3(n92), .IN4(n82), .QN(N88) );
  NOR2X0 U96 ( .IN1(p2[0]), .IN2(n90), .QN(N89) );
  NOR2X0 U97 ( .IN1(p1[0]), .IN2(n94), .QN(N84) );
  INVX0 U98 ( .IN(reset), .QN(n78) );
  INVX0 U99 ( .IN(p2[0]), .QN(n82) );
  INVX0 U100 ( .IN(p1[0]), .QN(n80) );
endmodule
```