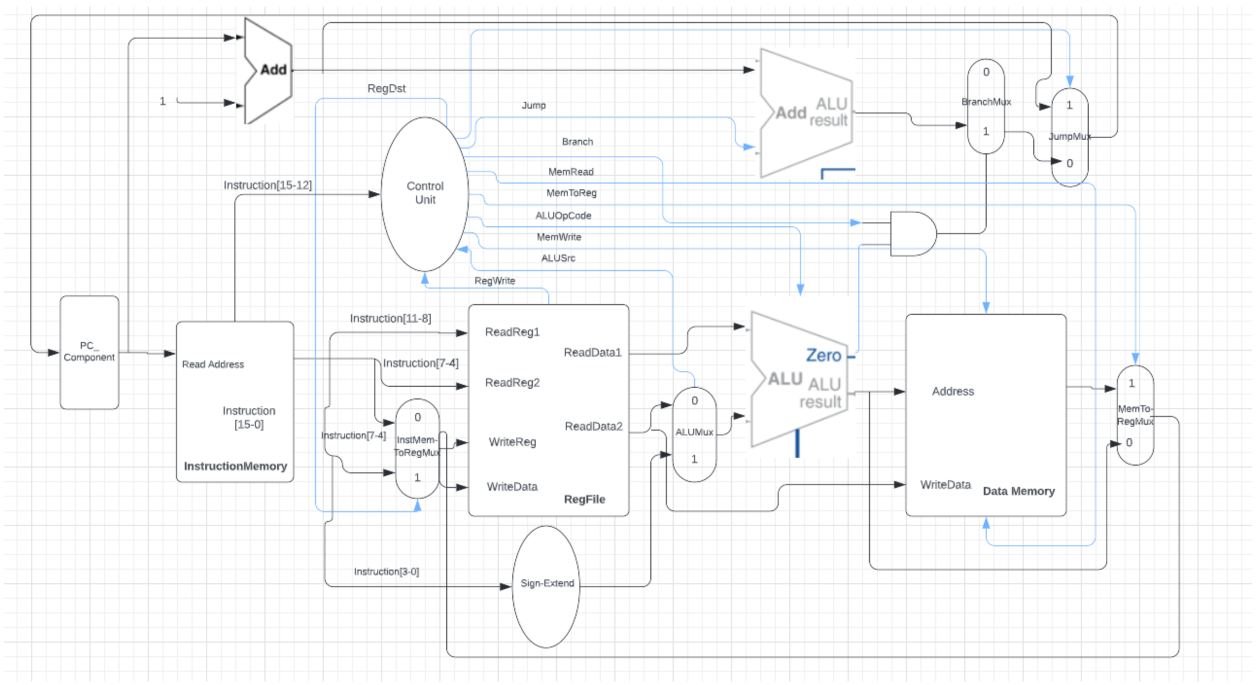
Lab 4 Final Report

1. Team Members

Team member: Matthew Coffey
Team member: Finn McGlothlin

2. Implementation Diagram

(Draw your design similar to the one shown below)



3. Datapath

The image above documents the connections between the units and the shows the overview of the whole code. Our code starts at the PC component and updates that on the falling edge of the clock. That new program counter then goes to the instruction memory and gets the new instruction that needs to be ran. Then, the instruction gets split up into the register file, the sign-extender, and the control unit. The control unit sends the correct signals to all the components. The register file pulls the contents out of the registers and sends them to the ALU. The sign-extender extends the sign bit and sends that to the ALU and the branch adder. The ALU takes the inputs and computes the necessary results. The zero flag dictates whether or not the branch operation is taken. The output of the ALU either goes into the Data Memory or the memory to reg mux. Finally, the data gets sent back to the register file and the next program counter is sent to the PC component.

4. ALU Control

As specified in the lab 4 design documents we were to design our ALU to compute at the very least 15 MIPS assembly instructions which were compiled and stored in our Instruction Memory component. As per our design document, we implemented 15 different ALU operations, all of which were one of the three Instructions formats: R-Type, I-Type, and J-Type.

R-Type instructions:

The add instruction was signified by the opcode “0000” and was computed by the ALU by adding the input signals ReadData1 and RegToALUMuxIn. The control signals associated with the add instruction were: RegDst = ‘1’, Jump = ‘0’, Branch = ‘0’, MemRead = ‘0’, MemToReg = ‘0’, MemWrite = ‘0’, ALUSrc = ‘0’, and RegWrite = ‘1’.

The next instruction our ALU supported was the sub operation. The subtract operation was signified by the opcode “0001” and was computed by the ALU by subtracting the input signals ReadData1 and RegToALUMuxIn. The control signals associated with the sub instruction were: RegDst = ‘1’, Jump = ‘0’, Branch = ‘0’, MemRead = ‘0’, MemToReg = ‘0’, MemWrite = ‘0’, ALUSrc = ‘0’, and RegWrite = ‘1’.

The next instruction our ALU supported was the bitwise or operation. The or operation was signified by the opcode “0010” and was computed by the ALU by using the built in VHDL or with both the input signals ReadData1 and RegToALUMuxIn. The control signals associated with the or instruction were: RegDst = ‘1’, Jump = ‘0’, Branch = ‘0’, MemRead = ‘0’, MemToReg = ‘0’, MemWrite = ‘0’, ALUSrc = ‘0’, and RegWrite = ‘1’.

The next instruction our ALU supported was the exclusive or operation(xor). The xor operation was signified by the opcode “0011” and was computed by the ALU by using the built in VHDL xor with both the input signals ReadData1 and RegToALUMuxIn. The control signals associated with the xor instruction were: RegDst = ‘1’, Jump = ‘0’, Branch = ‘0’, MemRead = ‘0’, MemToReg = ‘0’, MemWrite = ‘0’, ALUSrc = ‘0’, and RegWrite = ‘1’.

Of all the R-Type instructions listed above we ended up only using the or and xor operations as the adding and subtracting that was necessary for the sample program was done using constants.

I-Type Instructions:

The first I-Type instruction the ALU implemented was the multiply operation(mul). The mul operation was signified by the opcode “0100” and was computed by the ALU by using an addition operation inside a for loop. The operands used in the algorithm were ReadData1 from the register file and the sign extended constant RegToALUMuxIn received from the mux. The control signals associated with the mul instruction were: RegDst = ‘0’, Jump = ‘0’, Branch = ‘0’, MemRead = ‘0’, MemToReg = ‘0’, MemWrite = ‘0’, ALUSrc = ‘1’, and RegWrite = ‘1’.

The second I-Type instruction the ALU implemented was the divide operation(div). The div operation was signified by the opcode “0101” and was computed by the ALU by using the built in VHDL division operator. The operands used in the algorithm were ReadData1 from the register file and the sign extended constant RegToALUMuxIn received from the mux. The control signals associated with the mul instruction were: RegDst = ‘0’, Jump = ‘0’, Branch = ‘0’, MemRead = ‘0’, MemToReg = ‘0’, MemWrite = ‘0’, ALUSrc = ‘1’, and RegWrite = ‘1’.

The third I-Type instruction the ALU implemented was the add immediate operation(addi). The addi operation was signified by the opcode “0110” and was computed by the ALU by using the built in VHDL addition and concatenation operators. The operands used in the algorithm were ReadData1 from the register file and the sign extended constant RegToALUMuxIn received from the mux. The control signals associated with the addi instruction were: RegDst = ‘0’, Jump = ‘0’, Branch = ‘0’, MemRead = ‘0’, MemToReg = ‘0’, MemWrite = ‘0’, ALUSrc = ‘1’, and RegWrite = ‘1’.

The fourth I-Type instruction the ALU implemented was the subtract immediate operation(subi). The subi operation was signified by the opcode “0111” and was computed by the ALU by using the built in VHDL subtraction operator. The operands used in the algorithm were ReadData1 from the register file and the sign extended constant RegToALUMuxIn received from the mux. The control signals associated with the subi instruction were: RegDst = ‘0’, Jump = ‘0’, Branch = ‘0’, MemRead = ‘0’, MemToReg = ‘0’, MemWrite = ‘0’, ALUSrc = ‘1’, and RegWrite = ‘1’.

These four instructions were all used inside our implementation of the sample program. In implementing these operations, we were limited by the number of bits we could use for our constant operand inputs meaning we would need some other way to perform 16-bit constant operations. In order to accomplish this we split the addi instructions up and combined them with Shift logical left operation(sll) which in the end made 16-bit arithmetic possible.

The next I-Type instruction we implemented was the load word instruction (lw).The lw instruction is signified by the opcode “1000” and utitlizes a VHDL add operation inside of the ALU to add the sign-extended offset value with the Rt value. The resulting value is then passed to the Data Memory via the ALUResult signal which finds the data associated with that place in memory and passes that value to the DataMemToRegMux. The memory is then written back to the specified destination register inside of the Register file. The control signal associated with the lw instruction are as follows: RegDst = ‘0’, Jump = ‘0’, Branch = ‘0’, MemRead = ‘1’, MemToReg = ‘1’, MemWrite = ‘0’, ALUSrc = ‘1’, and RegWrite = ‘1’.

The next I-Type instruction we decided to implement was the store word instruction (sw). The sw instruction is represented by the opcode “1001” and utilizes a VHDL add operation inside of the ALU to add the sign-extended offset value with the Rt value. The resulting value is then passed to the Data Memory via the ALUResult signal and is used to in order to specify where the Rs register data value should be written to inside of the Data Memory. The control signals associated with the sw instruction are as follows: RegDst = ‘X’, Jump = ‘0’, Branch = ‘0’, MemRead = ‘0’, MemToReg = ‘X’, MemWrite = ‘1’, ALUSrc = ‘1’, and RegWrite = ‘0’.

The next three I-Type instructions that were implemented were all variations of Branch instructions. The first Branch instruction is the branch on equal (Beq) and represented by the opcode “1010”. The second Branch instruction is the branch on less than represented by the opcode “1011”. The last branch instruction we implemented is the branch on greater than which is represented by the opcode “1100. For all of these instructions we first subtract the Rs value by the rt value and output the result of the subtraction to the ALUResultOut signal. If the instruction is Beq we then test if the resulting subtraction value is equal to zero and if so, we set the zero-output signal to ‘1’. If the instruction is Blt we test if the Rs value is less than the Rt value and if so, we use the same zero-output signal and set it to ‘1’. If the instruction is Bgt we test if the Rs value is greater than the Rt value and if so, we use the same zero-output signal and set it to ‘1’. The outputs of these ALU operations are then used later in conjunction with the control unit signals, an and operation, a BranchMux component, and a JumpMux component to determine whether we will branch to a different point in our Instruction Memory. The control signals associated with the each of the branch instructions are as follows: RegDst = ‘X’, Jump = ‘0’, Branch = ‘1’, MemRead = ‘0’, MemToReg = ‘X’, MemWrite = ‘0’, ALUSrc = ‘0’, and RegWrite = ‘0’.

The final I-Type instruction that we implemented in our ALU is the shift left logical(sll) instruction. The sll instruction is represented by the opcode “1110” and utilized the VHDL sll operator to shift by a given constant value. The control signals associated with the sll instruction were as follows: RegDst = ‘0’, Jump = ‘0’, Branch = ‘0’, MemRead = ‘0’, MemToReg = ‘0’, MemWrite = ‘0’, ALUSrc = ‘1’, and RegWrite = ‘1’.

J-Type instructions:

The final two instructions that we implemented in our ALU were the J-Type instructions jump(j) and jump return (jr). The jump instruction was represented by the ALU by the opcode “1101” and the jump return instruction was represented by the instruction “1111”. These instructions utilize other components outside of the ALU such as the BranchJumpAdder and multiple other muxes to compute the next PC value which will specify the instruction that we are jumping to. Inside the ALU we will still set the ALUResultsignal to be all zeros. The control signal values associated for both the jump and jump return operation are as follows: RegDst = ‘X’, Jump = ‘1’, Branch = ‘0’, MemRead = ‘X’, MemToReg = ‘X’, MemWrite = ‘0’, ALUSrc = ‘X’, and RegWrite = ‘0’.

5. Control Unit

The control unit component we created in our 16-bit RISC processor design takes a 4-bit input from the InstructionMemory component. These bits represent the instructions Op code and are the 15-12 bits (MSB). Considering that our design only needed these 4-bits to represent all 15 instructions we chose to implement in our ALU we did not need a separate ALU control unit. The outputs of the control unit are the following signals: RegDst, Jump, Branch, MemRead, MemToReg, ALUOpcode, MemWrite, ALUSrc, and RegWrite.

The control unit implements a case statement as logic that determines the instruction is to be run. Based on the given instruction, the control unit output signals were set to values that would then help the processor know how to perform the specified instruction at various points inside the Datapath.

The first control signal used inside of the Datapath is the RegDst signal. This was used as an input into the InstMemToRegMux to specify what type of instruction i.e., R-Type, I-Type, and J-Type. If the instruction were to be a R-Type we would set the RegDst signal to be ‘1’ and if it was I-Type or J-type, then RegDst was set to ‘0’.

The next control signal used in the control unit was the RegWrite signal. The RegWrite signal was used to tell the register file component whether or not it should be writing a Data Memory value into a Register. The RegWrite signal was set to ‘1’ when the register file was needed to write a value received from the Data Memory to a register and was set to ‘0’ if it did not.

The next output signal the control unit implemented was the ALUSrc signal. This signal was used as an input to the ALUMux to determine whether the ALU should be using the value received from the sign-extension component or the ReadData2 signal coming from the Register file. If the instruction was I-type or J-type the ALUSrc value would be ‘1’ and if it was R-Type the value would be ‘0’.

The next output signal the control unit implemented was the MemWrite signal. The MemWrite signal is used as an input to the Data Memory component and determines when the Data Memory should be writing(storing) a value into the data memory. This control signal is explicitly used for the store word (sw) operation and is ‘1’ when we are writing a value into the Data Memory, and ‘0’ when we do not need to write to the Data Memory.

The next output control signal is the ALUOpCode. In our instruction design we decided to eliminate the bits that are normally reserved for the function and instead decoded all the possible instruction using just the 4 most significant bits of the instructions. In doing this, we eliminated the need for the ALU control unit and instead passed the ALUOpcode signal directly to the ALU. The next output signal we implemented was the MemToReg signal. This signal was used as an input into the DataMemToReg mux in order to specify if the Data Memory should be writing values into the register file. For example if a add instruction was specified we would need to write the result of the ALU operation to the destination register in the Register file. When it is necessary to write values back to a register the control signal is set to ‘1’ and is set to ‘0’ when unnecessary.

The next output signal from the control unit is the MemRead signal. This signal was set as an input to the Data Memory and was for the lw instruction. The signal is set to ‘1’ when a value needs to be read from the data memory and is set to ‘0’ when there is no value needed.
The next output signal is the Branch signal. The Branch signal was used alongside the zero signal coming from the ALU for the three I-Type branch instructions in order to determine when a branch to a different place in the Instruction Memory should occur. When a branch instruction is decoded the value is set to ‘1’ and is set to ‘0’ otherwise. The last output control signal we implemented in the control unit was the jump signal. The jump signal is set to ‘1’ whenever a J-Type instruction is decoded signifying that we need to jump to a different PC value. It is set to ‘0’ when the instruction type is not J-Type. Below are the control signal’s setups for each of our 16 possible instructions.

inst	RegDst	Jump	Branch	MemRead	MemToReg	MemWrite	ALUSrc	Reg-Write	ALUctrOp Code
add	1	0	0	0	0	0	0	1	0000
sub	1	0	0	0	0	0	0	1	0001
or	1	0	0	0	0	0	0	1	0010
xor	1	0	0	0	0	0	0	1	0011
mul	0	0	0	0	0	0	1	1	0100
div	0	0	0	0	0	0	1	1	0101
addi	0	0	0	0	0	0	1	1	0110
subi	0	0	0	0	0	0	1	1	0111
lw	0	0	0	1	1	0	1	1	1000
sw	X	0	0	0	X	1	1	0	1001
beq	X	0	1	0	X	0	0	0	1010

blt	X	0	1	0	X	0	0	0	1011
bgt	X	0	1	0	X	0	0	0	1100
sll	0	0	0	0	0	0	1	1	1110
j	X	1	0	X	X	0	X	0	1101
jr	X	1	0	X	X	0	X	0	1111

6. Simulation Results Memory and Register File at clock cycle 0

6.1. Memory and register file contents at clock cycle 0

Address 0x…	Hex Value	Binary Value 0b…
0	x"0101"	100000001
1	x"0101"	100000001
2	x"0110"	100010000
3	x"0011"	10001
4	x"00F0"	11110000
5	x"00FF"	11111111
6	x"0005"	101
7	x"0006"	110
8	x"0007"	111
9	x"0008"	1000
a	x"0009"	1001
b	x"000A"	1010
c	x"000B"	1011
d	x"000C"	1100
e	x"000D"	1101
f	x"000E"	1110
10	x"0101"	100000001
11	x"0101"	100000001
12	x"0110"	100010000
13	x"00F0"	11110000
14	x"0011"	10001
15	x"0000"	10110
16	x"00F0"	11110000
17	x"0016"	10110
18	x"0000"	0
19	x"0018"	11010

1a	x"0019"	11011
1b	x"001A"	11100
1c	x"001B"	11101
1d	x"001C"	11110
1e	x"001D"	11111
1f	x"001E"	100000
20	x"001F"	100001
21	x"0020"	100010
22	x"0021"	100011
23	x"0022"	100100
24	x"0023"	100101
25	x"0024"	100110
26	x"0025"	100111
27	x"0026"	101000
28	x"0027"	101001
29	x"0028"	101010
2a	x"0029"	101011
2b	x"002A"	101100
2c	x"002B"	101101
2d	x"002C"	101110
2e	x"002D"	101111
2f	x"002E"	110000
30	x"002F"	110001
31	x"0030"	110010
32	x"0031"	110011
33	x"0032"	110100
34	x"0033"	110101
35	x"0034"	110110
36	x"0035"	110111
37	x"0036"	111000
38	x"0037"	111001
39	x"0038"	111010
3a	x"0039"	111011
3b	x"003A"	111100
3c	x"003B"	111101
3d	x"003C"	111110
3e	x"003D"	111111
3f	x"003E"	1000000
40	x"003F"	1000001
41	x"0040"	1000010
42	x"0041"	1000011
43	x"0042"	1000100

44	x"0043"	1000101
45	x"0044"	1000110
46	x"0045"	1000111
47	x"0046"	1001000
48	x"0047"	1001001
49	x"0048"	1001010
4a	x"0049"	1001011
4b	x"004A"	1001100
4c	x"004B"	1001101
4d	x"004C"	1001110
4e	x"004D"	1001111
4f	x"004E"	1010000
50	x"004F"	1010001
51	x"0050"	1010010
52	x"0051"	1010011
53	x"0052"	1010100
54	x"0053"	1010101
55	x"0054"	1010110
56	x"0055"	1010111
57	x"0056"	1011000
58	x"0057"	1011001
59	x"0058"	1011010
5a	x"0059"	1011011
5b	x"005A"	1011100
5c	x"005B"	1011101
5d	x"005C"	1011110
5e	x"005D"	1011111
5f	x"005E"	1100000
60	x"005F"	1100001
61	x"0060"	1100010
62	x"0061"	1100011
63	x"0062"	1100100
64	x"0063"	1100101
65	x"0064"	1100110
66	x"0065"	1100111
67	x"0066"	1101000
68	x"0067"	1101001
69	x"0068"	1101010
6a	x"0069"	1101011
6b	x"006A"	1101100
6c	x"006B"	1101101
6d	x"006C"	1101110

6e	x"006D"	1101111
6f	x"006E"	1110000
70	x"006F"	1110001
71	x"0070"	1110010
72	x"0071"	1110011
73	x"0072"	1110100
74	x"0073"	1110101
75	x"0074"	1110110
76	x"0075"	1110111
77	x"0076"	1111000
78	x"0077"	1111001
79	x"0078"	1111010
7a	x"0079"	1111011
7b	x"007A"	1111100
7c	x"007B"	1111101
7d	x"007C"	1111110
7e	x"007D"	1111111
7f	x"007E"	10000000
80	x"007F"	10000001
81	x"0080"	10000010
82	x"0081"	10000011
83	x"0082"	10000100
84	x"0083"	10000101
85	x"0084"	10000110
86	x"0085"	10000111
87	x"0086"	10001000
88	x"0087"	10001001
89	x"0088"	10001010
8a	x"0089"	10001011
8b	x"008A"	10001100
8c	x"008B"	10001101
8d	x"008C"	10001110
8e	x"008D"	10001111
8f	x"008E"	10010000
90	x"008F"	10010001
91	x"0090"	10010010
92	x"0091"	10010011
93	x"0092"	10010100
94	x"0093"	10010101
95	x"0094"	10010110
96	x"0095"	10010111
97	x"0096"	10011000

98	x"0097"	10011001
99	x"0098"	10011010
9a	x"0099"	10011011
9b	x"009A"	10011100
9c	x"009B"	10011101
9d	x"009C"	10011110
9e	x"009D"	10011111
9f	x"009E"	10100000
a0	x"009F"	10100001
a1	x"00a0"	10100010
a2	x"00a1"	10100011
a3	x"00a2"	10100100
a4	x"00a3"	10100101
a5	x"00a4"	10100110
a6	x"00a5"	10100111
a7	x"00a6"	10101000
a8	x"00a7"	10101001
a9	x"00a8"	10101010
aa	x"00a9"	10101011
ab	x"00aa"	10101100
ac	x"00ab"	10101101
ad	x"00ac"	10101110
ae	x"00ad"	10101111
af	x"00ae"	10110000
b0	x"00af"	10110001
b1	x"00b0"	10110010
b2	x"00b1"	10110011
b3	x"00b2"	10110100
b4	x"00b3"	10110101
b5	x"00b4"	10110110
b6	x"00b5"	10110111
b7	x"00b6"	10111000
b8	x"00b7"	10111001
b9	x"00b8"	10111010
ba	x"00b9"	10111011
bb	x"00ba"	10111100
bc	x"00bb"	10111101
bd	x"00bc"	10111110
be	x"00bd"	10111111
bf	x"00be"	11000000
c0	x"00bf"	11000001
c1	x"00c0"	11000010

c2	x"00c1"	11000011
c3	x"00c2"	11000100
c4	x"00c3"	11000101
c5	x"00c4"	11000110
c6	x"00c5"	11000111
c7	x"00c6"	11001000
c8	x"00c7"	11001001
c9	x"00c8"	11001010
ca	x"00c9"	11001011
cb	x"00ca"	11001100
cc	x"00cb"	11001101
cd	x"00cc"	11001110
ce	x"00cd"	11001111
cf	x"00ce"	11010000
d0	x"00cf"	11010001
d1	x"00d0"	11010010
d2	x"00d1"	11010011
d3	x"00d2"	11010100
d4	x"00d3"	11010101
d5	x"00d4"	11010110
d6	x"00d5"	11010111
d7	x"00d6"	11011000
d8	x"00d7"	11011001
d9	x"00d8"	11011010
da	x"00d9"	11011011
db	x"00da"	11011100
dc	x"00db"	11011101
dd	x"00dc"	11011110
de	x"00dd"	11011111
df	x"00de"	11100000
e0	x"00df"	11100001
e1	x"00e0"	11100010
e2	x"00e1"	11100011
e3	x"00e2"	11100100
e4	x"00e3"	11100101
e5	x"00e4"	11100110
e6	x"00e5"	11100111
e7	x"00e6"	11101000
e8	x"00e7"	11101001
e9	x"00e8"	11101010
ea	x"00e9"	11101011
eb	x"00ea"	11101100

ec	x"00eb"	11101101
ed	x"00ec"	11101110
ee	x"00ed"	11101111
ef	x"00ee"	11110000
f0	x"00ef"	11110001
f1	x"00f0"	11110010
f2	x"00f1"	11110011
f3	x"00f2"	11110100
f4	x"00f3"	11110101
f5	x"00f4"	11110110
f6	x"00f5"	11110111
f7	x"00f6"	11111000
f8	x"00f7"	11111001
f9	x"00f8"	11111010
fa	x"00f9"	11111011
fb	x"00fa"	11111100
fc	x"00fb"	11111101
fd	x"00fc"	11111110
fe	x"00fd"	11111111
ff	x"00fe"	100000000

Register file (data):

Address 0x…	Hex Value	Binary Value 0b…
0	x"0000"	0
1	x"0040"	1000000
2	x"1010"	1000000010000
3	x"000F"	1111
4	x"00F0"	11110000
5	x"0010"	10000
6	x"0005"	101
7	x"0000"	0
8	x"0000"	0
9	x"0000"	0
a	x"0000"	0
b	x"0000"	0
c	x"0000"	0
d	x"0000"	0
e	x"0000"	0
f	x"0000"	0

6.2. Memory and register file contents for each loop:

You need to print out the data (not instructions) in your memory and register file after each loop. L, M, N should be the clock cycle numbers after each loop (e.g., L = 55 cycles after the first loop, N = 110 cycles after the second loop, etc). Your program should count the number of clock cycles after executing instructions.

After the first loop, clock cycle: 27 (end of 27)
Memory (data + instruction):

Address 0x…	Hex Value	Binary Value 0b…
0	x"0101"	100000001
1	x"0101"	100000001
2	x"0110"	100010000
3	x"0011"	10001
4	x"00F0"	11110000
5	x"00FF"	11111111
6	x"0005"	101
7	x"0006"	110
8	x"0007"	111
9	x"0008"	1000
a	x"0009"	1001
b	x"000A"	1010
c	x"000B"	1011
d	x"000C"	1100
e	x"000D"	1101
f	x"000E"	1110
10	x"FF00"	1111111100000000
11	x"0101"	100000001
12	x"0110"	100010000
13	x"00F0"	11110000
14	x"0011"	10001
15	x"0000"	10110
16	x"00F0"	11110000
17	x"0016"	10110
18	x"0000"	0
19	x"0018"	11010
1a	x"0019"	11011
1b	x"001A"	11100
1c	x"001B"	11101

1d	x"001C"	11110
1e	x"001D"	11111
1f	x"001E"	100000
20	x"001F"	100001
21	x"0020"	100010
22	x"0021"	100011
23	x"0022"	100100
24	x"0023"	100101
25	x"0024"	100110
26	x"0025"	100111
27	x"0026"	101000
28	x"0027"	101001
29	x"0028"	101010
2a	x"0029"	101011
2b	x"002A"	101100
2c	x"002B"	101101
2d	x"002C"	101110
2e	x"002D"	101111
2f	x"002E"	110000
30	x"002F"	110001
31	x"0030"	110010
32	x"0031"	110011
33	x"0032"	110100
34	x"0033"	110101
35	x"0034"	110110
36	x"0035"	110111
37	x"0036"	111000
38	x"0037"	111001
39	x"0038"	111010
3a	x"0039"	111011
3b	x"003A"	111100
3c	x"003B"	111101
3d	x"003C"	111110
3e	x"003D"	111111
3f	x"003E"	1000000
40	x"003F"	1000001
41	x"0040"	1000010
42	x"0041"	1000011
43	x"0042"	1000100
44	x"0043"	1000101
45	x"0044"	1000110
46	x"0045"	1000111

47	x"0046"	1001000
48	x"0047"	1001001
49	x"0048"	1001010
4a	x"0049"	1001011
4b	x"004A"	1001100
4c	x"004B"	1001101
4d	x"004C"	1001110
4e	x"004D"	1001111
4f	x"004E"	1010000
50	x"004F"	1010001
51	x"0050"	1010010
52	x"0051"	1010011
53	x"0052"	1010100
54	x"0053"	1010101
55	x"0054"	1010110
56	x"0055"	1010111
57	x"0056"	1011000
58	x"0057"	1011001
59	x"0058"	1011010
5a	x"0059"	1011011
5b	x"005A"	1011100
5c	x"005B"	1011101
5d	x"005C"	1011110
5e	x"005D"	1011111
5f	x"005E"	1100000
60	x"005F"	1100001
61	x"0060"	1100010
62	x"0061"	1100011
63	x"0062"	1100100
64	x"0063"	1100101
65	x"0064"	1100110
66	x"0065"	1100111
67	x"0066"	1101000
68	x"0067"	1101001
69	x"0068"	1101010
6a	x"0069"	1101011
6b	x"006A"	1101100
6c	x"006B"	1101101
6d	x"006C"	1101110
6e	x"006D"	1101111
6f	x"006E"	1110000
70	x"006F"	1110001

71	x"0070"	1110010
72	x"0071"	1110011
73	x"0072"	1110100
74	x"0073"	1110101
75	x"0074"	1110110
76	x"0075"	1110111
77	x"0076"	1111000
78	x"0077"	1111001
79	x"0078"	1111010
7a	x"0079"	1111011
7b	x"007A"	1111100
7c	x"007B"	1111101
7d	x"007C"	1111110
7e	x"007D"	1111111
7f	x"007E"	10000000
80	x"007F"	10000001
81	x"0080"	10000010
82	x"0081"	10000011
83	x"0082"	10000100
84	x"0083"	10000101
85	x"0084"	10000110
86	x"0085"	10000111
87	x"0086"	10001000
88	x"0087"	10001001
89	x"0088"	10001010
8a	x"0089"	10001011
8b	x"008A"	10001100
8c	x"008B"	10001101
8d	x"008C"	10001110
8e	x"008D"	10001111
8f	x"008E"	10010000
90	x"008F"	10010001
91	x"0090"	10010010
92	x"0091"	10010011
93	x"0092"	10010100
94	x"0093"	10010101
95	x"0094"	10010110
96	x"0095"	10010111
97	x"0096"	10011000
98	x"0097"	10011001
99	x"0098"	10011010
9a	x"0099"	10011011

9b	x"009A"	10011100
9c	x"009B"	10011101
9d	x"009C"	10011110
9e	x"009D"	10011111
9f	x"009E"	10100000
a0	x"009F"	10100001
a1	x"00a0"	10100010
a2	x"00a1"	10100011
a3	x"00a2"	10100100
a4	x"00a3"	10100101
a5	x"00a4"	10100110
a6	x"00a5"	10100111
a7	x"00a6"	10101000
a8	x"00a7"	10101001
a9	x"00a8"	10101010
aa	x"00a9"	10101011
ab	x"00aa"	10101100
ac	x"00ab"	10101101
ad	x"00ac"	10101110
ae	x"00ad"	10101111
af	x"00ae"	10110000
b0	x"00af"	10110001
b1	x"00b0"	10110010
b2	x"00b1"	10110011
b3	x"00b2"	10110100
b4	x"00b3"	10110101
b5	x"00b4"	10110110
b6	x"00b5"	10110111
b7	x"00b6"	10111000
b8	x"00b7"	10111001
b9	x"00b8"	10111010
ba	x"00b9"	10111011
bb	x"00ba"	10111100
bc	x"00bb"	10111101
bd	x"00bc"	10111110
be	x"00bd"	10111111
bf	x"00be"	11000000
c0	x"00bf"	11000001
c1	x"00c0"	11000010
c2	x"00c1"	11000011
c3	x"00c2"	11000100
c4	x"00c3"	11000101

c5	x"00c4"	11000110
c6	x"00c5"	11000111
c7	x"00c6"	11001000
c8	x"00c7"	11001001
c9	x"00c8"	11001010
ca	x"00c9"	11001011
cb	x"00ca"	11001100
cc	x"00cb"	11001101
cd	x"00cc"	11001110
ce	x"00cd"	11001111
cf	x"00ce"	11010000
d0	x"00cf"	11010001
d1	x"00d0"	11010010
d2	x"00d1"	11010011
d3	x"00d2"	11010100
d4	x"00d3"	11010101
d5	x"00d4"	11010110
d6	x"00d5"	11010111
d7	x"00d6"	11011000
d8	x"00d7"	11011001
d9	x"00d8"	11011010
da	x"00d9"	11011011
db	x"00da"	11011100
dc	x"00db"	11011101
dd	x"00dc"	11011110
de	x"00dd"	11011111
df	x"00de"	11100000
e0	x"00df"	11100001
e1	x"00e0"	11100010
e2	x"00e1"	11100011
e3	x"00e2"	11100100
e4	x"00e3"	11100101
e5	x"00e4"	11100110
e6	x"00e5"	11100111
e7	x"00e6"	11101000
e8	x"00e7"	11101001
e9	x"00e8"	11101010
ea	x"00e9"	11101011
eb	x"00ea"	11101100
ec	x"00eb"	11101101
ed	x"00ec"	11101110
ee	x"00ed"	11101111

ef	x"00ee"	11110000
f0	x"00ef"	11110001
f1	x"00f0"	11110010
f2	x"00f1"	11110011
f3	x"00f2"	11110100
f4	x"00f3"	11110101
f5	x"00f4"	11110110
f6	x"00f5"	11110111
f7	x"00f6"	11111000
f8	x"00f7"	11111001
f9	x"00f8"	11111010
fa	x"00f9"	11111011
fb	x"00fa"	11111100
fc	x"00fb"	11111101
fd	x"00fc"	11111110
fe	x"00fd"	11111111
ff	x"00fe"	100000000

Register file (data):

Address 0x…	Hex Value	Binary Value 0b…
0	x"0000"	0
1	x"0008"	1000
2	x"1018"	1000000011000
3	x"000F"	1111
4	x"00F0"	11110000
5	x"0010"	10000
6	x"0004"	100
7	x"0101"	100000001
8	x"0100"	100000000
9	x"FF00"	1111111100000000
a	x"0000"	0
b	x"0000"	0
c	x"0000"	0
d	x"0000"	0
e	x"0000"	0
f	x"0000"	0

After the second loop, clock cycle: 54
Memory Table: …

Address 0x…	Hex Value	Binary Value 0b…
0	x"0101"	100000001
1	x"0101"	100000001
2	x"0110"	100010000
3	x"0011"	10001
4	x"00F0"	11110000
5	x"00FF"	11111111
6	x"0005"	101
7	x"0006"	110
8	x"0007"	111
9	x"0008"	1000
a	x"0009"	1001
b	x"000A"	1010
c	x"000B"	1011
d	x"000C"	1100
e	x"000D"	1101
f	x"000E"	1110
10	x"FF00"	1111111100000000
11	x"0101"	100000001
12	x"FF00"	1111111100000000
13	x"00F0"	11110000
14	x"0011"	10001
15	x"0000"	10110
16	x"00F0"	11110000
17	x"0016"	10110
18	x"0000"	0
19	x"0018"	11010
1a	x"0019"	11011
1b	x"001A"	11100
1c	x"001B"	11101
1d	x"001C"	11110
1e	x"001D"	11111
1f	x"001E"	100000
20	x"001F"	100001
21	x"0020"	100010
22	x"0021"	100011
23	x"0022"	100100

24	x"0023"	100101
25	x"0024"	100110
26	x"0025"	100111
27	x"0026"	101000
28	x"0027"	101001
29	x"0028"	101010
2a	x"0029"	101011
2b	x"002A"	101100
2c	x"002B"	101101
2d	x"002C"	101110
2e	x"002D"	101111
2f	x"002E"	110000
30	x"002F"	110001
31	x"0030"	110010
32	x"0031"	110011
33	x"0032"	110100
34	x"0033"	110101
35	x"0034"	110110
36	x"0035"	110111
37	x"0036"	111000
38	x"0037"	111001
39	x"0038"	111010
3a	x"0039"	111011
3b	x"003A"	111100
3c	x"003B"	111101
3d	x"003C"	111110
3e	x"003D"	111111
3f	x"003E"	1000000
40	x"003F"	1000001
41	x"0040"	1000010
42	x"0041"	1000011
43	x"0042"	1000100
44	x"0043"	1000101
45	x"0044"	1000110
46	x"0045"	1000111
47	x"0046"	1001000
48	x"0047"	1001001
49	x"0048"	1001010
4a	x"0049"	1001011
4b	x"004A"	1001100
4c	x"004B"	1001101
4d	x"004C"	1001110

4e	x"004D"	1001111
4f	x"004E"	1010000
50	x"004F"	1010001
51	x"0050"	1010010
52	x"0051"	1010011
53	x"0052"	1010100
54	x"0053"	1010101
55	x"0054"	1010110
56	x"0055"	1010111
57	x"0056"	1011000
58	x"0057"	1011001
59	x"0058"	1011010
5a	x"0059"	1011011
5b	x"005A"	1011100
5c	x"005B"	1011101
5d	x"005C"	1011110
5e	x"005D"	1011111
5f	x"005E"	1100000
60	x"005F"	1100001
61	x"0060"	1100010
62	x"0061"	1100011
63	x"0062"	1100100
64	x"0063"	1100101
65	x"0064"	1100110
66	x"0065"	1100111
67	x"0066"	1101000
68	x"0067"	1101001
69	x"0068"	1101010
6a	x"0069"	1101011
6b	x"006A"	1101100
6c	x"006B"	1101101
6d	x"006C"	1101110
6e	x"006D"	1101111
6f	x"006E"	1110000
70	x"006F"	1110001
71	x"0070"	1110010
72	x"0071"	1110011
73	x"0072"	1110100
74	x"0073"	1110101
75	x"0074"	1110110
76	x"0075"	1110111
77	x"0076"	1111000

78	x"0077"	1111001
79	x"0078"	1111010
7a	x"0079"	1111011
7b	x"007A"	1111100
7c	x"007B"	1111101
7d	x"007C"	1111110
7e	x"007D"	1111111
7f	x"007E"	10000000
80	x"007F"	10000001
81	x"0080"	10000010
82	x"0081"	10000011
83	x"0082"	10000100
84	x"0083"	10000101
85	x"0084"	10000110
86	x"0085"	10000111
87	x"0086"	10001000
88	x"0087"	10001001
89	x"0088"	10001010
8a	x"0089"	10001011
8b	x"008A"	10001100
8c	x"008B"	10001101
8d	x"008C"	10001110
8e	x"008D"	10001111
8f	x"008E"	10010000
90	x"008F"	10010001
91	x"0090"	10010010
92	x"0091"	10010011
93	x"0092"	10010100
94	x"0093"	10010101
95	x"0094"	10010110
96	x"0095"	10010111
97	x"0096"	10011000
98	x"0097"	10011001
99	x"0098"	10011010
9a	x"0099"	10011011
9b	x"009A"	10011100
9c	x"009B"	10011101
9d	x"009C"	10011110
9e	x"009D"	10011111
9f	x"009E"	10100000
a0	x"009F"	10100001
a1	x"00a0"	10100010

a2	x"00a1"	10100011
a3	x"00a2"	10100100
a4	x"00a3"	10100101
a5	x"00a4"	10100110
a6	x"00a5"	10100111
a7	x"00a6"	10101000
a8	x"00a7"	10101001
a9	x"00a8"	10101010
aa	x"00a9"	10101011
ab	x"00aa"	10101100
ac	x"00ab"	10101101
ad	x"00ac"	10101110
ae	x"00ad"	10101111
af	x"00ae"	10110000
b0	x"00af"	10110001
b1	x"00b0"	10110010
b2	x"00b1"	10110011
b3	x"00b2"	10110100
b4	x"00b3"	10110101
b5	x"00b4"	10110110
b6	x"00b5"	10110111
b7	x"00b6"	10111000
b8	x"00b7"	10111001
b9	x"00b8"	10111010
ba	x"00b9"	10111011
bb	x"00ba"	10111100
bc	x"00bb"	10111101
bd	x"00bc"	10111110
be	x"00bd"	10111111
bf	x"00be"	11000000
c0	x"00bf"	11000001
c1	x"00c0"	11000010
c2	x"00c1"	11000011
c3	x"00c2"	11000100
c4	x"00c3"	11000101
c5	x"00c4"	11000110
c6	x"00c5"	11000111
c7	x"00c6"	11001000
c8	x"00c7"	11001001
c9	x"00c8"	11001010
ca	x"00c9"	11001011
cb	x"00ca"	11001100

cc	x"00cb"	11001101
cd	x"00cc"	11001110
ce	x"00cd"	11001111
cf	x"00ce"	11010000
d0	x"00cf"	11010001
d1	x"00d0"	11010010
d2	x"00d1"	11010011
d3	x"00d2"	11010100
d4	x"00d3"	11010101
d5	x"00d4"	11010110
d6	x"00d5"	11010111
d7	x"00d6"	11011000
d8	x"00d7"	11011001
d9	x"00d8"	11011010
da	x"00d9"	11011011
db	x"00da"	11011100
dc	x"00db"	11011101
dd	x"00dc"	11011110
de	x"00dd"	11011111
df	x"00de"	11100000
e0	x"00df"	11100001
e1	x"00e0"	11100010
e2	x"00e1"	11100011
e3	x"00e2"	11100100
e4	x"00e3"	11100101
e5	x"00e4"	11100110
e6	x"00e5"	11100111
e7	x"00e6"	11101000
e8	x"00e7"	11101001
e9	x"00e8"	11101010
ea	x"00e9"	11101011
eb	x"00ea"	11101100
ec	x"00eb"	11101101
ed	x"00ec"	11101110
ee	x"00ed"	11101111
ef	x"00ee"	11110000
f0	x"00ef"	11110001
f1	x"00f0"	11110010
f2	x"00f1"	11110011
f3	x"00f2"	11110100
f4	x"00f3"	11110101
f5	x"00f4"	11110110

f6	x"00f5"	11110111
f7	x"00f6"	11111000
f8	x"00f7"	11111001
f9	x"00f8"	11111010
fa	x"00f9"	11111011
fb	x"00fa"	11111100
fc	x"00fb"	11111101
fd	x"00fc"	11111110
fe	x"00fd"	11111111
ff	x"00fe"	100000000

Register File: **…**

Address 0x…	Hex Value	Binary Value 0b…
0	x"0000"	0
1	x"0001"	1
2	x"1019"	1000000011001
3	x"000F"	1111
4	x"00F0"	11110000
5	x"0012"	10010
6	x"0003"	11
7	x"0110"	100010000
8	x"0100"	100000000
9	x"FF00"	1111111100000000
a	x"0000"	0
b	x"0000"	0
c	x"0000"	0
d	x"0000"	0
e	x"0000"	0
f	x"0000"	0

After the 3rd loop, clock cycle: 81
Memory Table: …

Address 0x…	Hex Value	Binary Value 0b…
0	x"0101"	100000001
1	x"0101"	100000001
2	x"0110"	100010000

3	x"0011"	10001
4	x"00F0"	11110000
5	x"00FF"	11111111
6	x"0005"	101
7	x"0006"	110
8	x"0007"	111
9	x"0008"	1000
a	x"0009"	1001
b	x"000A"	1010
c	x"000B"	1011
d	x"000C"	1100
e	x"000D"	1101
f	x"000E"	1110
10	x"FF00"	1111111100000000
11	x"0101"	100000001
12	x"FF00"	1111111100000000
13	x"00F0"	11110000
14	x"00FF"	11111111
15	x"0000"	10110
16	x"00F0"	11110000
17	x"0016"	10110
18	x"0000"	0
19	x"0018"	11010
1a	x"0019"	11011
1b	x"001A"	11100
1c	x"001B"	11101
1d	x"001C"	11110
1e	x"001D"	11111
1f	x"001E"	100000
20	x"001F"	100001
21	x"0020"	100010
22	x"0021"	100011
23	x"0022"	100100
24	x"0023"	100101
25	x"0024"	100110
26	x"0025"	100111
27	x"0026"	101000
28	x"0027"	101001
29	x"0028"	101010
2a	x"0029"	101011
2b	x"002A"	101100
2c	x"002B"	101101

2d	x"002C"	101110
2e	x"002D"	101111
2f	x"002E"	110000
30	x"002F"	110001
31	x"0030"	110010
32	x"0031"	110011
33	x"0032"	110100
34	x"0033"	110101
35	x"0034"	110110
36	x"0035"	110111
37	x"0036"	111000
38	x"0037"	111001
39	x"0038"	111010
3a	x"0039"	111011
3b	x"003A"	111100
3c	x"003B"	111101
3d	x"003C"	111110
3e	x"003D"	111111
3f	x"003E"	1000000
40	x"003F"	1000001
41	x"0040"	1000010
42	x"0041"	1000011
43	x"0042"	1000100
44	x"0043"	1000101
45	x"0044"	1000110
46	x"0045"	1000111
47	x"0046"	1001000
48	x"0047"	1001001
49	x"0048"	1001010
4a	x"0049"	1001011
4b	x"004A"	1001100
4c	x"004B"	1001101
4d	x"004C"	1001110
4e	x"004D"	1001111
4f	x"004E"	1010000
50	x"004F"	1010001
51	x"0050"	1010010
52	x"0051"	1010011
53	x"0052"	1010100
54	x"0053"	1010101
55	x"0054"	1010110
56	x"0055"	1010111

57	x"0056"	1011000
58	x"0057"	1011001
59	x"0058"	1011010
5a	x"0059"	1011011
5b	x"005A"	1011100
5c	x"005B"	1011101
5d	x"005C"	1011110
5e	x"005D"	1011111
5f	x"005E"	1100000
60	x"005F"	1100001
61	x"0060"	1100010
62	x"0061"	1100011
63	x"0062"	1100100
64	x"0063"	1100101
65	x"0064"	1100110
66	x"0065"	1100111
67	x"0066"	1101000
68	x"0067"	1101001
69	x"0068"	1101010
6a	x"0069"	1101011
6b	x"006A"	1101100
6c	x"006B"	1101101
6d	x"006C"	1101110
6e	x"006D"	1101111
6f	x"006E"	1110000
70	x"006F"	1110001
71	x"0070"	1110010
72	x"0071"	1110011
73	x"0072"	1110100
74	x"0073"	1110101
75	x"0074"	1110110
76	x"0075"	1110111
77	x"0076"	1111000
78	x"0077"	1111001
79	x"0078"	1111010
7a	x"0079"	1111011
7b	x"007A"	1111100
7c	x"007B"	1111101
7d	x"007C"	1111110
7e	x"007D"	1111111
7f	x"007E"	10000000
80	x"007F"	10000001

81	x"0080"	10000010
82	x"0081"	10000011
83	x"0082"	10000100
84	x"0083"	10000101
85	x"0084"	10000110
86	x"0085"	10000111
87	x"0086"	10001000
88	x"0087"	10001001
89	x"0088"	10001010
8a	x"0089"	10001011
8b	x"008A"	10001100
8c	x"008B"	10001101
8d	x"008C"	10001110
8e	x"008D"	10001111
8f	x"008E"	10010000
90	x"008F"	10010001
91	x"0090"	10010010
92	x"0091"	10010011
93	x"0092"	10010100
94	x"0093"	10010101
95	x"0094"	10010110
96	x"0095"	10010111
97	x"0096"	10011000
98	x"0097"	10011001
99	x"0098"	10011010
9a	x"0099"	10011011
9b	x"009A"	10011100
9c	x"009B"	10011101
9d	x"009C"	10011110
9e	x"009D"	10011111
9f	x"009E"	10100000
a0	x"009F"	10100001
a1	x"00a0"	10100010
a2	x"00a1"	10100011
a3	x"00a2"	10100100
a4	x"00a3"	10100101
a5	x"00a4"	10100110
a6	x"00a5"	10100111
a7	x"00a6"	10101000
a8	x"00a7"	10101001
a9	x"00a8"	10101010
aa	x"00a9"	10101011

ab	x"00aa"	10101100
ac	x"00ab"	10101101
ad	x"00ac"	10101110
ae	x"00ad"	10101111
af	x"00ae"	10110000
b0	x"00af"	10110001
b1	x"00b0"	10110010
b2	x"00b1"	10110011
b3	x"00b2"	10110100
b4	x"00b3"	10110101
b5	x"00b4"	10110110
b6	x"00b5"	10110111
b7	x"00b6"	10111000
b8	x"00b7"	10111001
b9	x"00b8"	10111010
ba	x"00b9"	10111011
bb	x"00ba"	10111100
bc	x"00bb"	10111101
bd	x"00bc"	10111110
be	x"00bd"	10111111
bf	x"00be"	11000000
c0	x"00bf"	11000001
c1	x"00c0"	11000010
c2	x"00c1"	11000011
c3	x"00c2"	11000100
c4	x"00c3"	11000101
c5	x"00c4"	11000110
c6	x"00c5"	11000111
c7	x"00c6"	11001000
c8	x"00c7"	11001001
c9	x"00c8"	11001010
ca	x"00c9"	11001011
cb	x"00ca"	11001100
cc	x"00cb"	11001101
cd	x"00cc"	11001110
ce	x"00cd"	11001111
cf	x"00ce"	11010000
d0	x"00cf"	11010001
d1	x"00d0"	11010010
d2	x"00d1"	11010011
d3	x"00d2"	11010100
d4	x"00d3"	11010101

d5	x"00d4"	11010110
d6	x"00d5"	11010111
d7	x"00d6"	11011000
d8	x"00d7"	11011001
d9	x"00d8"	11011010
da	x"00d9"	11011011
db	x"00da"	11011100
dc	x"00db"	11011101
dd	x"00dc"	11011110
de	x"00dd"	11011111
df	x"00de"	11100000
e0	x"00df"	11100001
e1	x"00e0"	11100010
e2	x"00e1"	11100011
e3	x"00e2"	11100100
e4	x"00e3"	11100101
e5	x"00e4"	11100110
e6	x"00e5"	11100111
e7	x"00e6"	11101000
e8	x"00e7"	11101001
e9	x"00e8"	11101010
ea	x"00e9"	11101011
eb	x"00ea"	11101100
ec	x"00eb"	11101101
ed	x"00ec"	11101110
ee	x"00ed"	11101111
ef	x"00ee"	11110000
f0	x"00ef"	11110001
f1	x"00f0"	11110010
f2	x"00f1"	11110011
f3	x"00f2"	11110100
f4	x"00f3"	11110101
f5	x"00f4"	11110110
f6	x"00f5"	11110111
f7	x"00f6"	11111000
f8	x"00f7"	11111001
f9	x"00f8"	11111010
fa	x"00f9"	11111011
fb	x"00fa"	11111100
fc	x"00fb"	11111101
fd	x"00fc"	11111110
fe	x"00fd"	11111111

ff	x"00fe"	100000000

Register File: **…**

Address 0x…	Hex Value	Binary Value 0b…
0	x"0000"	0
1	x"0001"	1
2	x"1019"	1000000011001
3	x"003c"	111100
4	x"00cc"	11001100
5	x"0014"	10100
6	x"0002"	10
7	x"0011"	10001
8	x"0100"	100000000
9	x"00FF"	11111111
a	x"0000"	0
b	x"0000"	0
c	x"0000"	0
d	x"0000"	0
e	x"0000"	0
f	x"0000"	0

After the 4th loop, clock cycle: 108
Memory Table: …

Address 0x…	Hex Value	Binary Value 0b…
0	x"0101"	100000001
1	x"0101"	100000001
2	x"0110"	100010000
3	x"0011"	10001
4	x"00F0"	11110000
5	x"00FF"	11111111
6	x"0005"	101
7	x"0006"	110
8	x"0007"	111
9	x"0008"	1000
a	x"0009"	1001
b	x"000A"	1010

c	x"000B"	1011
d	x"000C"	1100
e	x"000D"	1101
f	x"000E"	1110
10	x"FF00"	1111111100000000
11	x"0101"	100000001
12	x"FF00"	1111111100000000
13	x"00F0"	11110000
14	x"00FF"	11111111
15	x"0000"	10110
16	x"00FF"	11111111
17	x"0016"	10110
18	x"0000"	0
19	x"0018"	11010
1a	x"0019"	11011
1b	x"001A"	11100
1c	x"001B"	11101
1d	x"001C"	11110
1e	x"001D"	11111
1f	x"001E"	100000
20	x"001F"	100001
21	x"0020"	100010
22	x"0021"	100011
23	x"0022"	100100
24	x"0023"	100101
25	x"0024"	100110
26	x"0025"	100111
27	x"0026"	101000
28	x"0027"	101001
29	x"0028"	101010
2a	x"0029"	101011
2b	x"002A"	101100
2c	x"002B"	101101
2d	x"002C"	101110
2e	x"002D"	101111
2f	x"002E"	110000
30	x"002F"	110001
31	x"0030"	110010
32	x"0031"	110011
33	x"0032"	110100
34	x"0033"	110101
35	x"0034"	110110

36	x"0035"	110111
37	x"0036"	111000
38	x"0037"	111001
39	x"0038"	111010
3a	x"0039"	111011
3b	x"003A"	111100
3c	x"003B"	111101
3d	x"003C"	111110
3e	x"003D"	111111
3f	x"003E"	1000000
40	x"003F"	1000001
41	x"0040"	1000010
42	x"0041"	1000011
43	x"0042"	1000100
44	x"0043"	1000101
45	x"0044"	1000110
46	x"0045"	1000111
47	x"0046"	1001000
48	x"0047"	1001001
49	x"0048"	1001010
4a	x"0049"	1001011
4b	x"004A"	1001100
4c	x"004B"	1001101
4d	x"004C"	1001110
4e	x"004D"	1001111
4f	x"004E"	1010000
50	x"004F"	1010001
51	x"0050"	1010010
52	x"0051"	1010011
53	x"0052"	1010100
54	x"0053"	1010101
55	x"0054"	1010110
56	x"0055"	1010111
57	x"0056"	1011000
58	x"0057"	1011001
59	x"0058"	1011010
5a	x"0059"	1011011
5b	x"005A"	1011100
5c	x"005B"	1011101
5d	x"005C"	1011110
5e	x"005D"	1011111
5f	x"005E"	1100000

60	x"005F"	1100001
61	x"0060"	1100010
62	x"0061"	1100011
63	x"0062"	1100100
64	x"0063"	1100101
65	x"0064"	1100110
66	x"0065"	1100111
67	x"0066"	1101000
68	x"0067"	1101001
69	x"0068"	1101010
6a	x"0069"	1101011
6b	x"006A"	1101100
6c	x"006B"	1101101
6d	x"006C"	1101110
6e	x"006D"	1101111
6f	x"006E"	1110000
70	x"006F"	1110001
71	x"0070"	1110010
72	x"0071"	1110011
73	x"0072"	1110100
74	x"0073"	1110101
75	x"0074"	1110110
76	x"0075"	1110111
77	x"0076"	1111000
78	x"0077"	1111001
79	x"0078"	1111010
7a	x"0079"	1111011
7b	x"007A"	1111100
7c	x"007B"	1111101
7d	x"007C"	1111110
7e	x"007D"	1111111
7f	x"007E"	10000000
80	x"007F"	10000001
81	x"0080"	10000010
82	x"0081"	10000011
83	x"0082"	10000100
84	x"0083"	10000101
85	x"0084"	10000110
86	x"0085"	10000111
87	x"0086"	10001000
88	x"0087"	10001001
89	x"0088"	10001010

8a	x"0089"	10001011
8b	x"008A"	10001100
8c	x"008B"	10001101
8d	x"008C"	10001110
8e	x"008D"	10001111
8f	x"008E"	10010000
90	x"008F"	10010001
91	x"0090"	10010010
92	x"0091"	10010011
93	x"0092"	10010100
94	x"0093"	10010101
95	x"0094"	10010110
96	x"0095"	10010111
97	x"0096"	10011000
98	x"0097"	10011001
99	x"0098"	10011010
9a	x"0099"	10011011
9b	x"009A"	10011100
9c	x"009B"	10011101
9d	x"009C"	10011110
9e	x"009D"	10011111
9f	x"009E"	10100000
a0	x"009F"	10100001
a1	x"00a0"	10100010
a2	x"00a1"	10100011
a3	x"00a2"	10100100
a4	x"00a3"	10100101
a5	x"00a4"	10100110
a6	x"00a5"	10100111
a7	x"00a6"	10101000
a8	x"00a7"	10101001
a9	x"00a8"	10101010
aa	x"00a9"	10101011
ab	x"00aa"	10101100
ac	x"00ab"	10101101
ad	x"00ac"	10101110
ae	x"00ad"	10101111
af	x"00ae"	10110000
b0	x"00af"	10110001
b1	x"00b0"	10110010
b2	x"00b1"	10110011
b3	x"00b2"	10110100

b4	x"00b3"	10110101
b5	x"00b4"	10110110
b6	x"00b5"	10110111
b7	x"00b6"	10111000
b8	x"00b7"	10111001
b9	x"00b8"	10111010
ba	x"00b9"	10111011
bb	x"00ba"	10111100
bc	x"00bb"	10111101
bd	x"00bc"	10111110
be	x"00bd"	10111111
bf	x"00be"	11000000
c0	x"00bf"	11000001
c1	x"00c0"	11000010
c2	x"00c1"	11000011
c3	x"00c2"	11000100
c4	x"00c3"	11000101
c5	x"00c4"	11000110
c6	x"00c5"	11000111
c7	x"00c6"	11001000
c8	x"00c7"	11001001
c9	x"00c8"	11001010
ca	x"00c9"	11001011
cb	x"00ca"	11001100
cc	x"00cb"	11001101
cd	x"00cc"	11001110
ce	x"00cd"	11001111
cf	x"00ce"	11010000
d0	x"00cf"	11010001
d1	x"00d0"	11010010
d2	x"00d1"	11010011
d3	x"00d2"	11010100
d4	x"00d3"	11010101
d5	x"00d4"	11010110
d6	x"00d5"	11010111
d7	x"00d6"	11011000
d8	x"00d7"	11011001
d9	x"00d8"	11011010
da	x"00d9"	11011011
db	x"00da"	11011100
dc	x"00db"	11011101
dd	x"00dc"	11011110

de	x"00dd"	11011111
df	x"00de"	11100000
e0	x"00df"	11100001
e1	x"00e0"	11100010
e2	x"00e1"	11100011
e3	x"00e2"	11100100
e4	x"00e3"	11100101
e5	x"00e4"	11100110
e6	x"00e5"	11100111
e7	x"00e6"	11101000
e8	x"00e7"	11101001
e9	x"00e8"	11101010
ea	x"00e9"	11101011
eb	x"00ea"	11101100
ec	x"00eb"	11101101
ed	x"00ec"	11101110
ee	x"00ed"	11101111
ef	x"00ee"	11110000
f0	x"00ef"	11110001
f1	x"00f0"	11110010
f2	x"00f1"	11110011
f3	x"00f2"	11110100
f4	x"00f3"	11110101
f5	x"00f4"	11110110
f6	x"00f5"	11110111
f7	x"00f6"	11111000
f8	x"00f7"	11111001
f9	x"00f8"	11111010
fa	x"00f9"	11111011
fb	x"00fa"	11111100
fc	x"00fb"	11111101
fd	x"00fc"	11111110
fe	x"00fd"	11111111
ff	x"00fe"	100000000

Register File: **…**

Address 0x…	Hex Value	Binary Value 0b…
0	x"0000"	0
1	x"0001"	1

2	x"1019"	1000000011001
3	x"00F0"	11110000
4	x"003C"	111100
5	x"0018"	11000
6	x"0001"	1
7	x"00F0"	11110000
8	x"0100"	100000000
9	x"00FF"	11111111
a	x"0000"	0
b	x"0000"	0
c	x"0000"	0
d	x"0000"	0
e	x"0000"	0
f	x"0000"	0

After the 5th loop, clock cycle: 135
Memory Table: …

Address 0x…	Hex Value	Binary Value 0b…
0	x"0101"	100000001
1	x"0101"	100000001
2	x"0110"	100010000
3	x"0011"	10001
4	x"00F0"	11110000
5	x"00FF"	11111111
6	x"0005"	101
7	x"0006"	110
8	x"0007"	111
9	x"0008"	1000
a	x"0009"	1001
b	x"000A"	1010
c	x"000B"	1011
d	x"000C"	1100
e	x"000D"	1101
f	x"000E"	1110
10	x"FF00"	1111111100000000
11	x"0101"	100000001
12	x"FF00"	1111111100000000
13	x"00F0"	11110000

14	x"00FF"	11111111
15	x"0000"	10110
16	x"00FF"	11111111
17	x"0016"	10110
18	x"00FF"	11111111
19	x"0018"	11010
1a	x"0019"	11011
1b	x"001A"	11100
1c	x"001B"	11101
1d	x"001C"	11110
1e	x"001D"	11111
1f	x"001E"	100000
20	x"001F"	100001
21	x"0020"	100010
22	x"0021"	100011
23	x"0022"	100100
24	x"0023"	100101
25	x"0024"	100110
26	x"0025"	100111
27	x"0026"	101000
28	x"0027"	101001
29	x"0028"	101010
2a	x"0029"	101011
2b	x"002A"	101100
2c	x"002B"	101101
2d	x"002C"	101110
2e	x"002D"	101111
2f	x"002E"	110000
30	x"002F"	110001
31	x"0030"	110010
32	x"0031"	110011
33	x"0032"	110100
34	x"0033"	110101
35	x"0034"	110110
36	x"0035"	110111
37	x"0036"	111000
38	x"0037"	111001
39	x"0038"	111010
3a	x"0039"	111011
3b	x"003A"	111100
3c	x"003B"	111101
3d	x"003C"	111110

3e	x"003D"	111111
3f	x"003E"	1000000
40	x"003F"	1000001
41	x"0040"	1000010
42	x"0041"	1000011
43	x"0042"	1000100
44	x"0043"	1000101
45	x"0044"	1000110
46	x"0045"	1000111
47	x"0046"	1001000
48	x"0047"	1001001
49	x"0048"	1001010
4a	x"0049"	1001011
4b	x"004A"	1001100
4c	x"004B"	1001101
4d	x"004C"	1001110
4e	x"004D"	1001111
4f	x"004E"	1010000
50	x"004F"	1010001
51	x"0050"	1010010
52	x"0051"	1010011
53	x"0052"	1010100
54	x"0053"	1010101
55	x"0054"	1010110
56	x"0055"	1010111
57	x"0056"	1011000
58	x"0057"	1011001
59	x"0058"	1011010
5a	x"0059"	1011011
5b	x"005A"	1011100
5c	x"005B"	1011101
5d	x"005C"	1011110
5e	x"005D"	1011111
5f	x"005E"	1100000
60	x"005F"	1100001
61	x"0060"	1100010
62	x"0061"	1100011
63	x"0062"	1100100
64	x"0063"	1100101
65	x"0064"	1100110
66	x"0065"	1100111
67	x"0066"	1101000

68	x"0067"	1101001
69	x"0068"	1101010
6a	x"0069"	1101011
6b	x"006A"	1101100
6c	x"006B"	1101101
6d	x"006C"	1101110
6e	x"006D"	1101111
6f	x"006E"	1110000
70	x"006F"	1110001
71	x"0070"	1110010
72	x"0071"	1110011
73	x"0072"	1110100
74	x"0073"	1110101
75	x"0074"	1110110
76	x"0075"	1110111
77	x"0076"	1111000
78	x"0077"	1111001
79	x"0078"	1111010
7a	x"0079"	1111011
7b	x"007A"	1111100
7c	x"007B"	1111101
7d	x"007C"	1111110
7e	x"007D"	1111111
7f	x"007E"	10000000
80	x"007F"	10000001
81	x"0080"	10000010
82	x"0081"	10000011
83	x"0082"	10000100
84	x"0083"	10000101
85	x"0084"	10000110
86	x"0085"	10000111
87	x"0086"	10001000
88	x"0087"	10001001
89	x"0088"	10001010
8a	x"0089"	10001011
8b	x"008A"	10001100
8c	x"008B"	10001101
8d	x"008C"	10001110
8e	x"008D"	10001111
8f	x"008E"	10010000
90	x"008F"	10010001
91	x"0090"	10010010

92	x"0091"	10010011
93	x"0092"	10010100
94	x"0093"	10010101
95	x"0094"	10010110
96	x"0095"	10010111
97	x"0096"	10011000
98	x"0097"	10011001
99	x"0098"	10011010
9a	x"0099"	10011011
9b	x"009A"	10011100
9c	x"009B"	10011101
9d	x"009C"	10011110
9e	x"009D"	10011111
9f	x"009E"	10100000
a0	x"009F"	10100001
a1	x"00a0"	10100010
a2	x"00a1"	10100011
a3	x"00a2"	10100100
a4	x"00a3"	10100101
a5	x"00a4"	10100110
a6	x"00a5"	10100111
a7	x"00a6"	10101000
a8	x"00a7"	10101001
a9	x"00a8"	10101010
aa	x"00a9"	10101011
ab	x"00aa"	10101100
ac	x"00ab"	10101101
ad	x"00ac"	10101110
ae	x"00ad"	10101111
af	x"00ae"	10110000
b0	x"00af"	10110001
b1	x"00b0"	10110010
b2	x"00b1"	10110011
b3	x"00b2"	10110100
b4	x"00b3"	10110101
b5	x"00b4"	10110110
b6	x"00b5"	10110111
b7	x"00b6"	10111000
b8	x"00b7"	10111001
b9	x"00b8"	10111010
ba	x"00b9"	10111011
bb	x"00ba"	10111100

bc	x"00bb"	10111101
bd	x"00bc"	10111110
be	x"00bd"	10111111
bf	x"00be"	11000000
c0	x"00bf"	11000001
c1	x"00c0"	11000010
c2	x"00c1"	11000011
c3	x"00c2"	11000100
c4	x"00c3"	11000101
c5	x"00c4"	11000110
c6	x"00c5"	11000111
c7	x"00c6"	11001000
c8	x"00c7"	11001001
c9	x"00c8"	11001010
ca	x"00c9"	11001011
cb	x"00ca"	11001100
cc	x"00cb"	11001101
cd	x"00cc"	11001110
ce	x"00cd"	11001111
cf	x"00ce"	11010000
d0	x"00cf"	11010001
d1	x"00d0"	11010010
d2	x"00d1"	11010011
d3	x"00d2"	11010100
d4	x"00d3"	11010101
d5	x"00d4"	11010110
d6	x"00d5"	11010111
d7	x"00d6"	11011000
d8	x"00d7"	11011001
d9	x"00d8"	11011010
da	x"00d9"	11011011
db	x"00da"	11011100
dc	x"00db"	11011101
dd	x"00dc"	11011110
de	x"00dd"	11011111
df	x"00de"	11100000
e0	x"00df"	11100001
e1	x"00e0"	11100010
e2	x"00e1"	11100011
e3	x"00e2"	11100100
e4	x"00e3"	11100101
e5	x"00e4"	11100110

e6	x"00e5"	11100111
e7	x"00e6"	11101000
e8	x"00e7"	11101001
e9	x"00e8"	11101010
ea	x"00e9"	11101011
eb	x"00ea"	11101100
ec	x"00eb"	11101101
ed	x"00ec"	11101110
ee	x"00ed"	11101111
ef	x"00ee"	11110000
f0	x"00ef"	11110001
f1	x"00f0"	11110010
f2	x"00f1"	11110011
f3	x"00f2"	11110100
f4	x"00f3"	11110101
f5	x"00f4"	11110110
f6	x"00f5"	11110111
f7	x"00f6"	11111000
f8	x"00f7"	11111001
f9	x"00f8"	11111010
fa	x"00f9"	11111011
fb	x"00fa"	11111100
fc	x"00fb"	11111101
fd	x"00fc"	11111110
fe	x"00fd"	11111111
ff	x"00fe"	100000000

Register File: **…**

Address 0x…	Hex Value	Binary Value 0b…
0	x"0000"	0
1	x"0001"	1
2	x"1019"	1000000011001
3	x"03C0"	1111000000
4	x"03FC"	1111111100
5	x"001A"	11010
6	x"0000"	0
7	x"0000"	0
8	x"0100"	100000000
9	x"00FF"	11111111
a	x"0000"	0

b	x"0000"	0
c	x"0000"	0
d	x"0000"	0
e	x"0000"	0
f	x"0000"	0

7. Integrated output data from the simulation results

Summarize the output **data (not instructions)** for the memory and register file for n loops of the test program.

Memory (data):

Address	Hex value after each loop					
	Initial	1st	2nd	3rd	4th	5th
0x10	x"0101"	x"FF00"	x"FF00"	x"FF00"	x"FF00"	x"FF00"
0x12	x"0110"	x"0110"	x"FF00"	x"FF00"	x"FF00"	x"FF00"
0x14	x"0011"	x"0011"	x"0011"	x"00FF"	x"00FF"	x"00FF"
0x16	x"00F0"	x"00F0"	x"00F0"	x"00F0"	x"00FF"	x"00FF"
0x18	x"0000"	x"0000"	x"0000"	x"0000"	x"0000"	x"00FF"

Register file (data):

Address	Hex value after each loop					
	Initial	1st	2nd	3rd	4th	5th
0	x"0000"	x"0000"	x"0000"	x"0000"	x"0000"	x"0000"
1	x"0040"	x"0008"	x"0001"	x"0001"	x"0001"	x"0001"
2	x"1010"	x"1018"	x"1019"	x"1019"	x"1019"	x"1019"

3	x"000F"	x"000F"	x"000F"	x"003c"	x"00F0"	x"03C0"
4	x"00F0"	x"00F0"	x"00F0"	x"00cc"	x"003C"	x"03FC"
5	x"0010"	x"0010"	x"0012"	x"0014"	x"0018"	x"001A"
6	x"0005"	x"0004"	x"0003"	x"0002"	x"0001"	x"0000"
7	x"0000"	x"0101"	x"0110"	x"0011"	x"00F0"	x"0000"
8	x"0000"	x"0100"	x"0100"	x"0100"	x"0100"	x"0100"
9	x"0000"	x"FF00"	x"FF00"	x"00FF"	x"00FF"	x"00FF"

8. Bonus Materials (if any)

Describe the modification to the design you made to implement the bonus parts of lab4

9. Discussion

We optimized the code by translating the C code to MIPS assembly and ordered the instructions in order to maximize the efficiency of the code. In the waveform, one optimization we made was making a nice waveform configuration file that has all of the possible signals organized for easy access and following.

The lab works correctly however, one part that we implemented that wasn't necessary was the shifters. We added them thinking we would grab half words instead of full words but then we went with the full word program counter. We left them in but the do nothing and don't effect the output.

Looking at the instruction memory, you will notice there are a few nops added at the beginning of the code. This was to give the signals time to stabilize before starting the first instruction. 5 definitely aren't necessary but they were added during testing and didn't affect the code at all.

10. Final Version of Simulator Code (Attached)

Here is a link directly to our github containing all data:
[Finnegan5095/16-bit-Risc-Lab4 (github.com)]

ALU:

```
----------------------------------------------------------------------------------
-
-- Company:
-- Engineer:
--
-- Create Date: 10/09/2022 05:09:09 PM
-- Design Name:
-- Module Name: ALU - Behavioral
-- Project Name:
-- Target Devices:
-- Tool Versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------
-


library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
--use IEEE.STD_LOGIC_UNSIGNED.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
use IEEE.NUMERIC_STD.ALL;
--use IEEE.STD_LOGIC_ARITH.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity ALU is
  Port (clk : in STD_LOGIC;
        ReadData1   : in STD_LOGIC_VECTOR(15 downto 0);
        RegToALUMuxIn   : in STD_LOGIC_VECTOR(15 downto 0);
        ALUctrOpCode  : in STD_LOGIC_VECTOR(3 downto 0);
        Zero    : out std_logic;
        Overflow    : out std_logic;
        Carryout    : out std_logic;
        ALUResultOut  : out STD_LOGIC_VECTOR(15 downto 0));
end ALU;

architecture Behavioral of ALU is

begin

    process(clk)
    Variable Result_Adder1 : signed(31 downto 0);
    Variable Adder_Zero1 : std_logic := '0';
    Variable Adder_Overflow1 : std_logic := '0';
    Variable Adder_Carryout1 : std_logic := '0';
    Variable Divisor : Integer := 0;
    Variable Mult1 : Integer := 0;
    Variable Mult2 : Integer := 0;
    Variable MultTotal : Integer := 0;
    Variable QuotientToInt : Integer := 0;
    Variable TempDiv : Integer := 0;
    Variable TempInv : signed(15 downto 0);
    Variable iTypeConstant : std_logic_vector(15 downto 0) := "0000000000000000";
    begin
        if rising_edge(clk) then
        --Case statements to determine ALUs operation and set the output with
correct output signals.
         iTypeConstant(3 downto 0) := RegToALUMuxIn(3 downto 0);
        case(ALUctrOpCode) IS
            --iTypeConstant(3 downto 0) := RegToALUMuxIn(3 downto 0);
            -- Adder case
            When "0000" =>
                  ALUResultOut <= STD_LOGIC_VECTOR(SIGNED(ReadData1) +
SIGNED(RegToALUMuxIn));
                  if (STD_LOGIC_VECTOR(SIGNED(ReadData1) + SIGNED(RegToALU-
MuxIn))) = "0000000000000000" then
                    Zero <= '1';
                else
                    Zero <= '0';
                end if;

            -- Subtraction case
            When "0001" =>
                  ALUResultOut <= STD_LOGIC_VECTOR(SIGNED(ReadData1) -
SIGNED(RegToALUMuxIn));
                  if (STD_LOGIC_VECTOR(SIGNED(ReadData1) - SIGNED(RegToALU-
MuxIn))) = "0000000000000000" then
                    Zero <= '1';
                 else
                    Zero <= '0';
                end if;

            -- Bitwise or case
            When "0010" =>
                  ALUResultOut <= (ReadData1 or RegToALUMuxIn);
                  if (ReadData1 or RegToALUMuxIn) = "0000000000000000" then -- If
less than zero goes high meaning we will branch.
                        Zero <= '1';
                  else
                        Zero <= '0';
                  end if;

            -- Exclusive or case
            When "0011" =>
                  ALUResultOut <= ReadData1 xor RegToALUMuxIn;
                  if (ReadData1 xor RegToALUMuxIn) = "0000000000000000" then --
If less than zero goes high meaning we will branch.
                        Zero <= '1';
                  else
                        Zero <= '0';
                  end if;


            -- Multiply
            When "0100" =>
                Mult1 := to_integer(signed(ReadData1));
                Mult2 := to_integer(signed(iTypeConstant));
                MultTotal := 0;
                for i in 1 to Mult2 loop
                    MultTotal := MultTotal + Mult1;
                end loop;

                ALUResultOut <= STD_LOGIC_VECTOR(to_signed(MultTotal, Read-
Data1'length));

                if (STD_LOGIC_VECTOR(to_signed(MultTotal, ReadData1'length))) =
"0000000000000000" then
```

```
                Zero <= '1';
            else
                Zero <= '0';
            end if;

        -- Divide
        When "0101" =>
            QuotientToInt := to_integer(signed(ReadData1));
            Divisor := to_integer(signed(iTypeConstant));
            TempDiv := 0;
            TempInv := "0000000000000000";
            --While TempDiv < QuotientToInt loop
            --       TempDiv := TempDiv + Divisor;
            --        if (TempDiv >= QuotientToInt) then
            --            exit;
            --        end if;
            --end loop;

            --if ((QuotientToInt < 0) or  (Divisor < 0)) then
            --    TempInv := not(to_signed(TempDiv, TempInv'length)) +
"0000000000000001";
            --end if;
            --ALUResultOut <= STD_LOGIC_VECTOR(TempInv);
            --R <= std_logic_vector(to_signed(to_integer(signed(X) /
signed(Y)),32));
            ALUResultOut <= std_logic_vector(to_signed(TO_INTEGER(signed(Re-
adData1) / signed(iTypeConstant)) , 16));
            --ALUResultOut <= (signed(ReadData1)) / (signed(RegToALUMuxIn));

            if (STD_LOGIC_VECTOR(TempINV)) = "0000000000000000" then
                Zero <= '1';
            else
                Zero <= '0';
            end if;



        -- Add Immediate
        When "0110" =>
            ALUResultOut <= STD_LOGIC_VECTOR(SIGNED(ReadData1) +
SIGNED(iTypeConstant));
            if (STD_LOGIC_VECTOR(SIGNED(ReadData1) + SIGNED(iTypeConstant)))
= "0000000000000000" then
                Zero <= '1';
            else
```

```
                        Zero <= '0';
                    end if;

                -- Sub Immediate
                When "0111" =>
                    ALUResultOut <= STD_LOGIC_VECTOR(SIGNED(ReadData1) -
SIGNED(iTypeConstant));
                    if (STD_LOGIC_VECTOR(SIGNED(ReadData1) - SIGNED(iTypeConstant)))
= "0000000000000000" then
                        Zero <= '1';
                    else
                        Zero <= '0';
                    end if;

                  -- Load Word
                When "1000" =>
                    ALUResultOut <= STD_LOGIC_VECTOR(SIGNED(ReadData1) +
SIGNED(iTypeConstant));
                    if (STD_LOGIC_VECTOR(SIGNED(ReadData1) + SIGNED(iTypeConstant)))
= "0000000000000000" then
                        Zero <= '1';
                    else
                        Zero <= '0';
                    end if;

                      -- Store word
                When "1001" =>
                    ALUResultOut <= STD_LOGIC_VECTOR(SIGNED(ReadData1) +
SIGNED(iTypeConstant));
                    if (STD_LOGIC_VECTOR(SIGNED(ReadData1) + SIGNED(iTypeConstant)))
= "0000000000000000" then
                        Zero <= '1';
                    else
                        Zero <= '0';
                    end if;

                      -- Branch on equal
                When "1010" =>
                    ALUResultOut <= STD_LOGIC_VECTOR(SIGNED(ReadData1) -
SIGNED(RegToALUMuxIn));
                      if ((SIGNED(ReadData1) - SIGNED(RegToALUMuxIn)) =
"0000000000000000") then
                        Zero <= '1';
                      else
                        Zero <= '0';
```

```
                   end if;

                  -- Branch on less than
              When "1011" =>
                  ALUResultOut <= STD_LOGIC_VECTOR(SIGNED(ReadData1) -
SIGNED(RegToALUMuxIn));
                  if ((SIGNED(ReadData1) < SIGNED(RegToALUMuxIn))) then -- If less
than zero goes high meaning we will branch.
                      Zero <= '1';
                  else
                      Zero <= '0';
                  end if;

                   -- Branch on greater than
              When "1100" =>
                  ALUResultOut <= STD_LOGIC_VECTOR(SIGNED(ReadData1) -
SIGNED(RegToALUMuxIn));
                  if ((SIGNED(ReadData1) > SIGNED(RegToALUMuxIn))) then -- If less
than zero goes high meaning we will branch.
                      Zero <= '1';
                  else
                      Zero <= '0';
                  end if;

                   -- Jump
              When "1101" =>
                  -- We don't need these values so we just set to zeros.
                  ALUResultOut <= "0000000000000000";
                  Zero <= '0';

                   -- Shift Left Logical
              When "1110" =>
                  ALUResultOut <= std_logic_vector(unsigned(ReadData1) sll to_inte-
ger(unsigned(iTypeConstant)));
                  --ALUResultOut <= std_logic_vector(shift_left(unsigned(Read-
Data1), to_integer(unsigned(iTypeConstant))));

                   -- Jump Return
              When "1111" =>
                   -- We don't need these values so we just set to zeros.
                  ALUResultOut <= "0000000000000000";
                  Zero <= '0';

              When others =>
                  ALUResultOut <= "0000000000000000";
```

```
                Zero <= '0';

        end case;
        end if;
    end process;
end Behavioral;
```

ALUMux:

```
----------------------------------------------------------------------------------
-
-- Company: 
-- Engineer: 
-- 
-- Create Date: 10/27/2022 06:45:00 PM
-- Design Name: 
-- Module Name: ALUMux - Behavioral
-- Project Name: 
-- Target Devices: 
-- Tool Versions: 
-- Description: 
-- 
-- Dependencies: 
-- 
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
-- 
----------------------------------------------------------------------------------
-


library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity ALUMux is
  Port (ReadData2 : in STD_LOGIC_VECTOR(15 downto 0);
        SignExtend : in STD_LOGIC_VECTOR(15 downto 0);
        ALUSrc  : in STD_LOGIC;
        ALUMuxOut   : out STD_LOGIC_VECTOR(15 downto 0));
end ALUMux;

architecture Behavioral of ALUMux is
```

```vhdl
begin

    process (ReadData2, ALUSrc, SignExtend)
   begin
    if (ALUSrc = '0') then
        ALUMuxOut <= ReadData2;
    elsif (ALUSrc = '1') then
        ALUMuxOut <= SignExtend;
    else
        ALUMuxOut <= "HHHHHHHHHHHHHHHH";
    end if;
    end process;

end Behavioral;
```

BranchJumpAdder:

```
----------------------------------------------------------------------------------
-
-- Company: 
-- Engineer: 
-- 
-- Create Date: 10/28/2022 12:46:54 AM
-- Design Name: 
-- Module Name: BranchJumpAdder - Behavioral
-- Project Name: 
-- Target Devices: 
-- Tool Versions: 
-- Description: 
-- 
-- Dependencies: 
-- 
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
-- 
----------------------------------------------------------------------------------
-


library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity BranchJumpAdder is
   Port (ShiftInput : in STD_LOGIC_VECTOR(15 downto 0);
         PCAddress : in STD_LOGIC_VECTOR(15 downto 0);
         ALUResult  : out STD_LOGIC_VECTOR(15 downto 0));
end BranchJumpAdder;

architecture Behavioral of BranchJumpAdder is

begin
```

```
    process(PCAddress)
    begin
         ALUResult <= std_logic_vector(signed(ShiftInput) + signed(PCAddress)); -
-Adding the two 16-bit addresses.
    end process;

end Behavioral;
```

BranchMux:

```
----------------------------------------------------------------------------------
-
-- Company: 
-- Engineer: 
-- 
-- Create Date: 10/27/2022 06:55:12 PM
-- Design Name: 
-- Module Name: BranchMux - Behavioral
-- Project Name: 
-- Target Devices: 
-- Tool Versions: 
-- Description: 
-- 
-- Dependencies: 
-- 
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
-- 
----------------------------------------------------------------------------------
-


library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity BranchMux is
  Port (PCPlusFour : in STD_LOGIC_VECTOR(15 downto 0);
        BranchALUResult : in STD_LOGIC_VECTOR(15 downto 0);
        --PCSrc   : in STD_LOGIC;
        Zero : in std_logic;
        Branch : in std_logic;
        NextPC   : out STD_LOGIC_VECTOR(15 downto 0));
end BranchMux;
```

```
architecture Behavioral of BranchMux is
signal PCSrc : std_logic;
begin
PCSrc <= (Zero and Branch);
process (PCPlusFour, BranchALUResult)
   begin
    if (PCSrc = '0') then
        NextPC <= PCPlusFour;
    elsif (PCSrc = '1') then
        NextPC <= BranchALUResult;
    else
        NextPC <= "HHHHHHHHHHHHHHHH";
    end if;

    end process;

end Behavioral;
```

ControlUnit:
```
----------------------------------------------------------------------------------
-
-- Company: 
-- Engineer: 
-- 
-- Create Date: 11/03/2022 05:02:03 PM
-- Design Name: 
-- Module Name: ControlUnit - Behavioral
-- Project Name: 
-- Target Devices: 
-- Tool Versions: 
-- Description: 
-- 
-- Dependencies: 
-- 
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
-- 
----------------------------------------------------------------------------------
-


library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity ControlUnit is
  Port (InstOpCode : in STD_LOGIC_VECTOR(15 downto 0);
        RegDst  : out STD_LOGIC;
        Jump  : out STD_LOGIC;
        Branch  : out STD_LOGIC;
        MemRead  : out STD_LOGIC;
        MemToReg  : out STD_LOGIC;
        ALUOpCode  : out STD_LOGIC_VECTOR(3 downto 0);
        MemWrite  : out STD_LOGIC;
        ALUSrc   : out STD_LOGIC;
        RegWrite   : out STD_LOGIC);
end ControlUnit;

architecture Behavioral of ControlUnit is

begin

Process(InstOpCode) is
begin

    CASE InstOpCode(15 downto 12) is
        When "0000" => -- Add (R-type)
            RegDst <= '1';
            Jump <= '0';
            Branch <= '0';
            MemRead <= '0';
            MemToReg <= '0';
            MemWrite <= '0';
            ALUSrc <= '0';
            RegWrite <= '1';
            ALUOpCode <= "0000";

        When "0001" => -- Sub (R-type)
            RegDst <= '1';
            Jump <= '0';
            Branch <= '0';
            MemRead <= '0';
            MemToReg <= '0';
            MemWrite <= '0';
            ALUSrc <= '0';
            RegWrite <= '1';
            ALUOpCode <= "0001";

        When "0010" => -- Or (R-type)
             RegDst <= '1';
            Jump <= '0';
            Branch <= '0';
            MemRead <= '0';
            MemToReg <= '0';
            MemWrite <= '0';
            ALUSrc <= '0';
            RegWrite <= '1';
            ALUOpCode <= "0010";
```

```
When "0011" => -- Xor (R-type)
    RegDst <= '1';
    Jump <= '0';
    Branch <= '0';
    MemRead <= '0';
    MemToReg <= '0';
    MemWrite <= '0';
    ALUSrc <= '0';
    RegWrite <= '1';
    ALUOpCode <= "0011";

When "0100" => -- mul (I-type)
    RegDst <= '0';
    Jump <= '0';
    Branch <= '0';
    MemRead <= '0';
    MemToReg <= '0';
    MemWrite <= '0';
    ALUSrc <= '1';
    RegWrite <= '1';
    ALUOpCode <= "0100";

When "0101" => -- div (I-type)
    RegDst <= '0';
    Jump <= '0';
    Branch <= '0';
    MemRead <= '0';
    MemToReg <= '0';
    MemWrite <= '0';
    ALUSrc <= '1';
    RegWrite <= '1';
    ALUOpCode <= "0101";

When "0110" => -- addi (I-type)
    RegDst <= '0';
    Jump <= '0';
    Branch <= '0';
    MemRead <= '0';
    MemToReg <= '0';
    MemWrite <= '0';
    ALUSrc <= '1';
    RegWrite <= '1';
    ALUOpCode <= "0110";

When "0111" => -- subi (I-type)
```

```vhdl
        RegDst <= '0';
        Jump <= '0';
        Branch <= '0';
        MemRead <= '0';
        MemToReg <= '0';
        MemWrite <= '0';
        ALUSrc <= '1';
        RegWrite <= '1';
        ALUOpCode <= "0111";

    When "1000" => -- lw (I-type)
        RegDst <= '0';
        Jump <= '0';
        Branch <= '0';
        MemRead <= '1';
        MemToReg <= '1';
        MemWrite <= '0';
        ALUSrc <= '1';
        RegWrite <= '1';
        ALUOpCode <= "1000";

    When "1001" => -- sw (I-type)
        RegDst <= '0';
        Jump <= '0';
        Branch <= '0';
        MemRead <= '0';
        MemToReg <= '0';
        MemWrite <= '1';
        ALUSrc <= '1';
        RegWrite <= '0';
        ALUOpCode <= "1001";

    When "1010" => -- beq (I-type)
        RegDst <= '0';
        Jump <= '0';
        Branch <= '1';
        MemRead <= '0';
        MemToReg <= '0';
        MemWrite <= '0';
        ALUSrc <= '0';
        RegWrite <= '0';
        ALUOpCode <= "1010";

    When "1011" => --blt (I-type)
         RegDst <= '0';
        Jump <= '0';
        Branch <= '1';
        MemRead <= '0';
        MemToReg <= '0';
        MemWrite <= '0';
        ALUSrc <= '0';
        RegWrite <= '0';
        ALUOpCode <= "1011";

    When "1100" => --bgt (I-type)
        RegDst <= '0';
        Jump <= '0';
        Branch <= '1';
        MemRead <= '0';
        MemToReg <= '0';
        MemWrite <= '0';
        ALUSrc <= '0';
        RegWrite <= '0';
        ALUOpCode <= "1100";

    When "1110" => -- sll (I-type)
        RegDst <= '0';
        Jump <= '0';
        Branch <= '0';
        MemRead <= '0';
        MemToReg <= '0';
        MemWrite <= '0';
        ALUSrc <= '1';
        RegWrite <= '1';
        ALUOpCode <= "1110";

    When "1101" => -- j (J-type)
        RegDst <= '0';
        Jump <= '1';
        Branch <= '0';
        MemRead <= '0';
        MemToReg <= '0';
        MemWrite <= '0';
        ALUSrc <= '0';
        RegWrite <= '0';
        ALUOpCode <= "1101";

    When "1111" => -- jr (J-type)
        RegDst <= '0';
        Jump <= '1';
```

```
            Branch <= '0';
            MemRead <= '0';
            MemToReg <= '0';
            MemWrite <= '0';
            ALUSrc <= '0';
            RegWrite <= '0';
            ALUOpCode <= "1111";

        When Others =>
            RegDst <= 'X';
            Jump <= 'X';
            Branch <= 'X';
            MemRead <= 'X';
            MemToReg <= 'X';
            MemWrite <= 'X';
            ALUSrc <= 'X';
            RegWrite <= 'X';
            ALUOpCode <= "XXXX";

    end CASE;
end process;

end Behavioral;
```

DataMemory:

```vhdl
----------------------------------------------------------------------------------
-
-- Company:
-- Engineer:
--
-- Create Date: 10/27/2022 09:05:16 PM
-- Design Name:
-- Module Name: DataMemory - Behavioral
-- Project Name:
-- Target Devices:
-- Tool Versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------
-


library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity DataMemory is
  Port (Zero : in STD_LOGIC;
        clk  : in STD_LOGIC;
        WriteData : in STD_LOGIC_VECTOR(15 downto 0);
        ALUResult : in STD_LOGIC_VECTOR(15 downto 0);
        MemWrite    : in STD_LOGIC;
        MemRead     : in STD_LOGIC;
        ReadData    : out STD_LOGIC_VECTOR(15 downto 0));
end DataMemory;
```

```vhdl
architecture Behavioral of DataMemory is

type DataMem is array (0 to 255) of STD_LOGIC_VECTOR(15 downto 0);
signal LongAssMemorySignal : DataMem := (
    x"0101",
    x"0101",
    x"0110",
    x"0011",
    x"00F0",
    x"00FF",
    x"0005",
    x"0006",
    x"0007",
    x"0008",
    x"0009",
    x"000A",
    x"000B",
    x"000C",
    x"000D",
    x"000E",
    x"0101", --0000000100000001
    x"0101",
    x"0110", --
    x"00F0",
    x"0011", --
    x"0000",
    x"00F0", --
    x"0016",
    x"0000", --
    x"0018",
    x"0019",
    x"001A",
    x"001B",
    x"001C",
    x"001D",
    x"001E",
    x"001F",
    x"0020",
    x"0021",
    x"0022",
    x"0023",
    x"0024",
    x"0025",
    x"0026",
    x"0027",
    x"0028",
    x"0029",
    x"002A",
    x"002B",
    x"002C",
    x"002D",
    x"002E",
    x"002F",
    x"0030",
    x"0031",
    x"0032",
    x"0033",
    x"0034",
    x"0035",
    x"0036",
    x"0037",
    x"0038",
    x"0039",
    x"003A",
    x"003B",
    x"003C",
    x"003D",
    x"003E",
    x"003F",
    x"0040",
    x"0041",
    x"0042",
    x"0043",
    x"0044",
    x"0045",
    x"0046",
    x"0047",
    x"0048",
    x"0049",
    x"004A",
    x"004B",
    x"004C",
    x"004D",
    x"004E",
    x"004F",
    x"0050",
    x"0051",
    x"0052",
    x"0053",
    x"0054",
    x"0055",
    x"0056",
    x"0057",
    x"0058",
    x"0059",
    x"005A",
    x"005B",
    x"005C",
    x"005D",
    x"005E",
    x"005F",
    x"0060",
    x"0061",
    x"0062",
    x"0063",
    x"0064",
    x"0065",
    x"0066",
    x"0067",
    x"0068",
    x"0069",
    x"006A",
    x"006B",
    x"006C",
    x"006D",
    x"006E",
    x"006F",
    x"0070",
    x"0071",
    x"0072",
    x"0073",
    x"0074",
    x"0075",
    x"0076",
    x"0077",
    x"0078",
    x"0079",
    x"007A",
    x"007B",
    x"007C",
    x"007D",
    x"007E",
    x"007F",
    x"0080",
    x"0081",
    x"0082",
    x"0083",
    x"0084",
    x"0085",
    x"0086",
    x"0087",
    x"0088",
    x"0089",
    x"008A",
    x"008B",
    x"008C",
    x"008D",
    x"008E",
    x"008F",
    x"0090",
    x"0091",
    x"0092",
    x"0093",
    x"0094",
    x"0095",
    x"0096",
    x"0097",
    x"0098",
    x"0099",
    x"009A",
    x"009B",
    x"009C",
    x"009D",
    x"009E",
    x"009F",
    x"00a0",
    x"00a1",
    x"00a2",
    x"00a3",
    x"00a4",
    x"00a5",
    x"00a6",
    x"00a7",
    x"00a8",
    x"00a9",
    x"00aa",
    x"00ab",
    x"00ac",
    x"00ad",
    x"00ae",
    x"00af",
    x"00b0",
    x"00b1",
    x"00b2",
    x"00b3",
    x"00b4",
    x"00b5",
    x"00b6",
    x"00b7",
    x"00b8",
    x"00b9",
    x"00ba",
    x"00bb",
    x"00bc",
    x"00bd",
    x"00be",
    x"00bf",
    x"00c0",
    x"00c1",
    x"00c2",
    x"00c3",
    x"00c4",
    x"00c5",
    x"00c6",
    x"00c7",
    x"00c8",
    x"00c9",
    x"00ca",
    x"00cb",
    x"00cc",
    x"00cd",
    x"00ce",
    x"00cf",
    x"00d0",
    x"00d1",
    x"00d2",
    x"00d3",
    x"00d4",
    x"00d5",
    x"00d6",
    x"00d7",
    x"00d8",
    x"00d9",
    x"00da",
    x"00db",
    x"00dc",
    x"00dd",
    x"00de",
    x"00df",
    x"00e0",
    x"00e1",
    x"00e2",
    x"00e3",
    x"00e4",
    x"00e5",
    x"00e6",
    x"00e7",
    x"00e8",
    x"00e9",
    x"00ea",
    x"00eb",
    x"00ec",
    x"00ed",
    x"00ee",
    x"00ef",
    x"00f0",
    x"00f1",
    x"00f2",
    x"00f3",
    x"00f4",
    x"00f5",
    x"00f6",
    x"00f7",
    x"00f8",
    x"00f9",
    x"00fa",
    x"00fb",
    x"00fc",
    x"00fd",
    x"00fe"
    --ffx"060F",
    --100x"060F"

    );
begin


process(clk)
begin
```

```
-- Write and read on the falling edge.
if (MemRead = '1') then
    ReadData <= LongAssMemorySignal(to_integer(unsigned(ALUResult)));

elsif (MemWrite = '1' and Rising_edge(clk)) then
    LongAssMemorySignal(to_integer(unsigned(WriteData))) <= ALUResult;

else --?
    --ReadData <= x"0001";
end if;


end process;
end Behavioral;
```

Datapath:

```
----------------------------------------------------------------------------------
-
-- Company:
-- Engineer:
--
-- Create Date: 10/28/2022 06:15:38 PM
-- Design Name:
-- Module Name: Datapath - Structural
-- Project Name:
-- Target Devices:
-- Tool Versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------
-


library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity Datapath is
  Port (clk : in STD_LOGIC);
end Datapath;

architecture Structural of Datapath is

    Component ALU
        PORT (clk : in STD_LOGIC;
              ReadData1   : in STD_LOGIC_VECTOR(15 downto 0);
```

```
          RegToALUMuxIn   : in STD_LOGIC_VECTOR(15 downto 0);
          ALUctrOpCode  : in STD_LOGIC_VECTOR(3 downto 0);
          Zero    : out std_logic;
          ALUResultOut  : out STD_LOGIC_VECTOR(15 downto 0));
END Component;

Component ALUMux
    Port (ReadData2 : in STD_LOGIC_VECTOR(15 downto 0);
          SignExtend : in STD_LOGIC_VECTOR(15 downto 0);
          ALUSrc  : in STD_LOGIC;
          ALUMuxOut   : out STD_LOGIC_VECTOR(15 downto 0));
End Component;

Component BranchJumpAdder
    Port (ShiftInput : in STD_LOGIC_VECTOR(15 downto 0);
          PCAddress : in STD_LOGIC_VECTOR(15 downto 0);
          ALUResult  : out STD_LOGIC_VECTOR(15 downto 0));
End Component;

Component BranchMux
    Port (PCPlusFour : in STD_LOGIC_VECTOR(15 downto 0);
          BranchALUResult : in STD_LOGIC_VECTOR(15 downto 0);
          Zero : in STD_LOGIC;
          Branch : in STD_LOGIC;
          --PCSrc  : in STD_LOGIC;
          NextPC   : out STD_LOGIC_VECTOR(15 downto 0));
End Component;

Component DataMemory
    Port (Zero : in STD_LOGIC;
          clk  : in STD_LOGIC;
          WriteData : in STD_LOGIC_VECTOR(15 downto 0);
          ALUResult : in STD_LOGIC_VECTOR(15 downto 0);
          MemWrite    : in STD_LOGIC;
          MemRead     : in STD_LOGIC;
          ReadData    : out STD_LOGIC_VECTOR(15 downto 0));
End Component;

Component InstMemToRegMux
     Port (RegDst : in STD_LOGIC;
           Inst1Rs    : in STD_LOGIC_VECTOR(3 downto 0);
           Inst2Rd    : in STD_LOGIC_VECTOR(3 downto 0);
           MuxOut   : out STD_LOGIC_VECTOR(3 downto 0));
End Component;
```

```
Component InstructionMemory
     Port (InstAddress : in STD_LOGIC_VECTOR(15 downto 0);
           InstData    : out STD_LOGIC_VECTOR(15 downto 0));
End Component;

Component Instruction_Adder_Component
    Port (clk : in  STD_LOGIC;
          InstIn : in STD_LOGIC_VECTOR(15 downto 0);
          InstOut : out STD_LOGIC_VECTOR(15 downto 0));
End Component;

Component MemToRegMux
    Port (ReadDataMem : in STD_LOGIC_VECTOR(15 downto 0);
          ALUResult : in STD_LOGIC_VECTOR(15 downto 0);
          MemToReg  : in STD_LOGIC;
          MemToRegMuxOut   : out STD_LOGIC_VECTOR(15 downto 0));
End Component;

Component PC_Component
    Port (clk : in  STD_LOGIC;
          pc_in : in  STD_LOGIC_VECTOR (15 downto 0);
          pc_out : inout  STD_LOGIC_VECTOR (15 downto 0));
End Component;

Component RegFile
    Port (clk : in STD_LOGIC;
          InstData : in STD_LOGIC_VECTOR(15 downto 0);
          ReadReg1 : in STD_LOGIC_VECTOR(3 downto 0);
          ReadReg2 : in STD_LOGIC_VECTOR(3 downto 0);
          WriteReg : in STD_LOGIC_VECTOR(3 downto 0);
          WriteData : in STD_LOGIC_VECTOR(15 downto 0);
          RegWriteCtrl : in STD_LOGIC;
          ReadData1 : out STD_LOGIC_VECTOR(15 downto 0);
          ReadData2 : out STD_LOGIC_VECTOR(15 downto 0));
End Component;

Component ShiftBranch
    Port (SignExtend : in STD_LOGIC_VECTOR(15 downto 0);
          ShiftOutBranch    : out STD_LOGIC_VECTOR(15 downto 0));
End Component;

Component ShiftJump
    Port (ShiftAddress : in STD_LOGIC_VECTOR(11 downto 0);
          ShiftOut    : out STD_LOGIC_VECTOR(12 downto 0));
End Component;
```

```
    Component SignExtension
        Port (DataIn : in STD_LOGIC_VECTOR(3 downto 0);
              ExtendedData : out STD_LOGIC_VECTOR(15 downto 0));
    End Component;

    Component JumpMux
        Port (Jump : in STD_LOGIC;
              --JumpAddressIn : in STD_LOGIC_VECTOR(15 downto 0);
              PCMostSig : in std_logic_vector(2 downto 0);
              ShiftJumpSignalOut : in std_logic_vector(12 downto 0);
              BranchMuxIn : in STD_LOGIC_VECTOR(15 downto 0);
              PCOut   : inout STD_LOGIC_VECTOR(15 downto 0));
    End Component;

    Component ControlUnit
        Port (InstOpCode : in STD_LOGIC_VECTOR(15 downto 0);
              RegDst  : out STD_LOGIC;
              Jump  : out STD_LOGIC;
              Branch  : out STD_LOGIC;
              MemRead  : out STD_LOGIC;
              MemToReg  : out STD_LOGIC;
              ALUOpCode  : out STD_LOGIC_VECTOR(3 downto 0);
              MemWrite  : out STD_LOGIC;
              ALUSrc  : out STD_LOGIC;
              RegWrite  : out STD_LOGIC);
    End Component;

    FOR ALL : ALU use ENTITY work.ALU(Behavioral);
    FOR ALL : ALUMux use ENTITY work.ALUMux(Behavioral);
    FOR ALL : BranchJumpAdder use ENTITY work.BranchJumpAdder(Behavioral);
    FOR ALL : BranchMux use ENTITY work.BranchMux(Behavioral);
    FOR ALL : DataMemory use ENTITY work.DataMemory(Behavioral);
    FOR ALL : InstMemToRegMux use ENTITY work.InstMemToRegMux(Behavioral);
    FOR ALL : InstructionMemory use ENTITY work.InstructionMemory(Behavioral);
    FOR ALL : Instruction_Adder_Component use ENTITY work.Instruction_Adder_Com-
ponent(Behavioral);
    FOR ALL : MemToRegMux use ENTITY work.MemToRegMux(Behavioral);
    FOR ALL : PC_Component use ENTITY work.PC_Component(Behavioral);
    FOR ALL : RegFile use ENTITY work.RegFile(Behavioral);
    FOR ALL : ShiftBranch use ENTITY work.ShiftBranch(Behavioral);
    FOR ALL : ShiftJump use ENTITY work.ShiftJump(Behavioral);
    FOR ALL : SignExtension use ENTITY work.SignExtension(Behavioral);
    FOR ALL : JumpMux use ENTITY work.JumpMux(Behavioral);
    FOR ALL : ControlUnit use ENTITY work.ControlUnit(Behavioral);
    signal InitialPCSignal : STD_LOGIC_VECTOR(15 downto
0)                        := "0000000000000000";
    signal pc_out_signal : STD_LOGIC_VECTOR(15 downto
0)                          := "1111111111111111";
    signal InstDataSignal : STD_LOGIC_VECTOR(15 downto
0)                         := "0000000000000000";
    --signal InstInAdderSignal : STD_LOGIC_VECTOR(15 downto
0)                    := "0000000000000000";
    --signal InstOutAdderSignal : STD_LOGIC_VECTOR(15 downto
0)                   := "0000000000000000";
    --
    signal InstRsMuxSignal : STD_LOGIC_VECTOR(3 downto
0)                         := "0000";
    signal InstRtMuxSignal : STD_LOGIC_VECTOR(3 downto
0)                         := "0000";
    signal InstRdMuxSignal : STD_LOGIC_VECTOR(3 downto
0)                         := "0000";
    --
    signal WriteRegisterSignalReg : STD_LOGIC_VECTOR(3 downto
0)                  := "0000";
    signal Instruction_Adder_Component_Signal : STD_LOGIC_VECTOR(15 downto
0)   := "0000000000000000";
    signal SignExtendedSignal : STD_LOGIC_VECTOR(15 downto
0)                     := "0000000000000000";
    signal JumpShifterSignal : STD_LOGIC_VECTOR(11 downto
0)                      := "000000000000";
    signal ReadReg1OutSignal : STD_LOGIC_VECTOR(15 downto
0)                      := "0000000000000000";
    signal ReadReg2OutSignal : STD_LOGIC_VECTOR(15 downto
0)                      := "0000000000000000";
    signal ALUMuxOutSignal : STD_LOGIC_VECTOR(15 downto
0)                        := "0000000000000000";
    signal ZeroSignal :
STD_LOGIC                                              := '0';
    signal ALUResultsigOut : STD_LOGIC_VECTOR(15 downto
0)                        := "0000000000000000";
    signal DataMemorySignalOut : STD_LOGIC_VECTOR(15 downto
0)                    := "0000000000000000";
    signal MemToRegMuxOutSignal : STD_LOGIC_VECTOR(15 downto
0)                   := "0000000000000000";
    signal ShiftOutBranchSignal : STD_LOGIC_VECTOR(15 downto
0)                   := "0000000000000000";
    signal PCMostSig : STD_LOGIC_VECTOR(2 downto
0)                               := "000";
```

```
    signal BranchJumpAdderALUResult : STD_LOGIC_VECTOR(15 downto
0)              := "0000000000000000";
    signal ShiftJumpSignalOut : STD_LOGIC_VECTOR(12 downto
0)                    := "0000000000000";
    signal ZeroAndBranchSignal :
STD_LOGIC                                       := '0';
    signal BranchMuxSignalOut : STD_LOGIC_VECTOR(15 downto
0)                    := "0000000000000000";
    signal JumpFullSignal : STD_LOGIC_VECTOR(15 downto
0)                        := "0000000000000000";
    signal PCOutSignal : STD_LOGIC_VECTOR(15 downto
0)                           := "1111111111111111";
    --
    --Control Unit signals.
    signal OpCodeSignal : STD_LOGIC_VECTOR(3 downto
0)                           := "0000";
    signal ALUOpCodeSignal : STD_LOGIC_VECTOR(3 downto
0)                        := "0000";
    signal RegDstSignal :
STD_LOGIC                                              := '0';
    signal JumpSignal :
STD_LOGIC                                                := '0';
    signal BranchSignal :
STD_LOGIC                                              := '0';
    signal MemReadSignal :
STD_LOGIC                                             := '0';
    signal MemToRegSignal :
STD_LOGIC                                            := '0';
    signal MemWriteSignal :
STD_LOGIC                                            := '0';
    signal ALUSrcSignal :
STD_LOGIC                                              := '0';
    signal RegWriteSignal :
STD_LOGIC                                            := '0';


begin


    PC_ComponentCall : PC_Component
    PORT MAP(
        clk => clk,
        pc_in  => PCOutSignal,
        pc_out  => pc_out_signal
    );
```

```
InstructionMemoryCall : InstructionMemory
 PORT MAP(
    InstAddress => pc_out_signal,
    InstData  => InstDataSignal
);

Instruction_Adder_ComponentCall : Instruction_Adder_Component
 PORT MAP(
    clk => clk,
    InstIn => pc_out_signal,
    InstOut  => Instruction_Adder_Component_Signal
);

ControlUnitCall : ControlUnit
 PORT MAP(
    InstOpCode => InstDataSignal,
    RegDst => RegDstSignal,
    Jump  => JumpSignal,
    Branch  => BranchSignal,
    MemRead => MemReadSignal,
    MemToReg => MemToRegSignal,
    ALUOpCode => ALUOpCodeSignal,
    MemWrite => MemWriteSignal,
    ALUSrc => ALUSrcSignal,
    RegWrite => RegWriteSignal
);

InstMemToRegMuxCall : InstMemToRegMux
 PORT MAP(
    RegDst => RegDstSignal,
    Inst1Rs  => InstDataSignal(7 downto 4),
    Inst2Rd  => InstDataSignal(3 downto 0),
    MuxOut => WriteRegisterSignalReg
);

SignExtensionCall : SignExtension
 PORT MAP(
    DataIn => InstDataSignal(3 downto 0), --Rd
    ExtendedData  => SignExtendedSignal
);

ShiftBranchCall : ShiftBranch
 PORT MAP(
    SignExtend => SignExtendedSignal,
    ShiftOutBranch  => ShiftOutBranchSignal
);

BranchJumpAdderCall : BranchJumpAdder
 PORT MAP(
    ShiftInput => ShiftOutBranchSignal,
    PCAddress  => Instruction_Adder_Component_Signal,
    ALUResult  => BranchJumpAdderALUResult
);

ShiftJumpCall : ShiftJump
 PORT MAP(
    ShiftAddress => InstDataSignal(11 downto 0),
    ShiftOut  => ShiftJumpSignalOut
);

RegFileCall : RegFile
 PORT MAP(
    clk => clk,
    InstData => InstDataSignal,
    ReadReg1  => InstDataSignal(11 downto 8),
    ReadReg2  => InstDataSignal(7 downto 4),
    WriteReg  => WriteRegisterSignalReg,
    WriteData => MemToRegMuxOutSignal,
    RegWriteCtrl => RegWriteSignal,
    ReadData1 => ReadReg1OutSignal,
    ReadData2 => ReadReg2OutSignal
);

ALUMuxCall  : ALUMux
 PORT MAP(
    ReadData2 => ReadReg2OutSignal,
    SignExtend  => SignExtendedSignal,
    ALUSrc  => ALUSrcSignal,
    ALUMuxOut => ALUMuxOutSignal
);

ALUCall : ALU
PORT MAP(
    clk => clk,
    ReadData1 => ReadReg1OutSignal,
    RegToALUMuxIn  => ALUMuxOutSignal,
    ALUctrOpCode  => ALUOpCodeSignal,
    Zero  => ZeroSignal,
    ALUResultOut => ALUResultsigOut
```

```vhdl
    );

    DataMemoryCall : DataMemory
    PORT MAP(
        Zero => ZeroSignal,
        clk  => clk,
        WriteData  => ReadReg2OutSignal,
        ALUResult  => ALUResultsigOut,
        MemWrite => MemWriteSignal,
        MemRead => MemReadSignal,
        ReadData => DataMemorySignalOut
    );

    MemToRegMuxCall : MemToRegMux
    PORT MAP(
        ReadDataMem => DataMemorySignalOut,
        ALUResult  => ALUResultsigOut,
        MemToReg  => MemToRegSignal,
        MemToRegMuxOut  => MemToRegMuxOutSignal
    );

    BranchMuxCall : BranchMux
    PORT MAP(
        PCPlusFour => Instruction_Adder_Component_Signal,
        BranchALUResult  => BranchJumpAdderALUResult,
        Zero => ZeroSignal,
        Branch => BranchSignal,
        --PCSrc => ZeroAndBranchSignal ,
        NextPC => BranchMuxSignalOut
    );

    JumpMuxCall : JumpMux
    PORT MAP(
        Jump => JumpSignal,
        PCMostSig => Instruction_Adder_Component_Signal(15 downto 13),
        ShiftJumpSignalOut => ShiftJumpSignalOut,
        --JumpAddressIn  => PCMostSig & ShiftJumpSignalOut,
        BranchMuxIn  => BranchMuxSignalOut,
        PCOut => PCOutSignal
    );

    --process (clk)
    --process (clk, PCOutSignal, InstDataSignal, Instruction_Adder_Component_Sig-
nal, ShiftJumpSignalOut, ZeroSignal, BranchSignal )
    --begin
    --if falling_edge(clk) then
    --      InitialPCSignal <= PCOutSignal;
    --end if;
    --if rising_edge(clk) then
    --if falling_edge(clk) then

           --InitialPCSignal <= PCOutSignal;
           --OpCodeSignal <= InstDataSignal(15 downto 12);
           --PCMostSig <= Instruction_Adder_Component_Signal(15 downto 13);
           --InstRsMuxSignal <= InstDataSignal(11 downto 8);
           --InstRtMuxSignal <= InstDataSignal(7 downto 4);
           --InstRdMuxSignal <= InstDataSignal(3 downto 0);
           --JumpShifterSignal <= InstDataSignal(11 downto 0);
           --JumpFullSignal <= PCMostSig & ShiftJumpSignalOut;
           --ZeroAndBranchSignal <= ZeroSignal and BranchSignal;


    --end if;
    --end process;


end Structural;
```

InstMemToRegMux:

```
----------------------------------------------------------------------------------
-
-- Company:
-- Engineer:
--
-- Create Date: 10/28/2022 06:16:24 PM
-- Design Name:
-- Module Name: InstMemToRegMux - Behavioral
-- Project Name:
-- Target Devices:
-- Tool Versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------
-


library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity InstMemToRegMux is
  Port (RegDst : in STD_LOGIC;
        Inst1Rs   : in STD_LOGIC_VECTOR(3 downto 0);
        Inst2Rd   : in STD_LOGIC_VECTOR(3 downto 0);
        MuxOut  : out STD_LOGIC_VECTOR(3 downto 0));

end InstMemToRegMux;

architecture Behavioral of InstMemToRegMux is
```

```
begin

   process (RegDst, Inst2Rd, Inst1Rs)
   begin
    if (RegDst = '0') then
        MuxOut <= Inst1Rs;
    elsif (RegDst = '1') then
        MuxOut <= Inst2Rd;
    else
        MuxOut <= "HHHH";
    end if;
    end process;


end Behavioral;
```

Instruction_Adder_Component:

```
----------------------------------------------------------------------------------
-
-- Company:
-- Engineer:
--
-- Create Date: 10/22/2022 04:12:25 PM
-- Design Name:
-- Module Name: Instruction_Adder_Component - Behavioral
-- Project Name:
-- Target Devices:
-- Tool Versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------
-


library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
use IEEE.NUMERIC_STD.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity Instruction_Adder_Component is
  Port (clk : in  STD_LOGIC;
        InstIn : in STD_LOGIC_VECTOR(15 downto 0);
        InstOut : out STD_LOGIC_VECTOR(15 downto 0));
end Instruction_Adder_Component;

architecture Behavioral of Instruction_Adder_Component is
```

```
begin
process(clk, InstIn)
    variable temp : std_logic_vector(15 downto 0) := "0000000000000000";
    begin
        if rising_edge(clk) then
            temp := InstIn + "0000000000000001";
        --InstOut <= InstIn + "0000000000000001"; --Incrementing by 1.
        else
            temp := temp;
        end if;
        InstOut <= temp;
 end process;

end Behavioral;
```

InstructionMemory:

```vhdl
----------------------------------------------------------------------------------
-
-- Company: 
-- Engineer: 
-- 
-- Create Date: 10/27/2022 04:16:17 PM
-- Design Name: 
-- Module Name: InstructionMemory - Behavioral
-- Project Name: 
-- Target Devices: 
-- Tool Versions: 
-- Description: 
-- 
-- Dependencies: 
-- 
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
-- 
----------------------------------------------------------------------------------
-


library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
use IEEE.NUMERIC_STD.ALL;
--use IEEE.STD_LOGIC_ARITH.ALL;
--use IEEE.STD_LOGIC_UNSIGNED.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity InstructionMemory is
  Port (InstAddress : in STD_LOGIC_VECTOR(15 downto 0);
        InstData    : out STD_LOGIC_VECTOR(15 downto 0));
end InstructionMemory;

architecture Behavioral of InstructionMemory is
```

```
type InstMem is array (0 to 43) of STD_LOGIC_VECTOR(15 downto 0); --Declaring ar-
ray type that will store all 16-bit instructions vectors.

signal Memory : InstMem := (
        "1110000000000000",   --pc = 0  0000   Nop insertion
        "1110000000000000",   --pc = 1  0001   Nop insertion
        "1110000000000000",   --pc = 2  0010   Nop insertion
        "1110000000000000",   --pc = 3  0011   Nop insertion
        "1110000000000000",   --pc = 4  0100   Nop insertion
        "1100011000000001",   --pc = 5  0101   bgt a0 zero ZeroLine  branch 1
line (pc incremented by 2*1)
        "1101000001111111",   --pc = 6  0110   j Exit                jump 40
lines (pc incremented by 2*40)
        "0111011001100001",   --pc = 7  0111   subi a1 a1 1          subtract 1
from a1
        "1000010101110000",   --pc = 8  1000   lw t0 0(a0)           put contents
of a0 into t0
        "1110100010000100",   --pc = 9  1001   sll t1 t1 4           shift t1
left four
        "0110100010000000",   --pc = 10 1010   addi t1 t1 0x0        add 0000 to
t1
        "1110100010000100",   --pc = 11 1011   sll t1 t1 4           shift t1
left four
        "0110100010000001",   --pc = 12 1100   addi t1 t1 0x1        add 0001 to
t1
        "1110100010000100",   --pc = 13 1101   sll t1 t1 4           shift t1
left four
        "0110100010000000",   --pc = 14 1110   addi t1 t1 0x0        add 0000 to
t1
        "1110100010000100",   --pc = 15 1110   sll t1 t1 4           shift t1
left four
        "0110100010000000",   --pc = 16 10000  addi t1 t1 0x0        add 0000 to
t1
        "1100011110000001",   --pc = 17 10001  bgt t0 t1 SkipIfJump  branch to
SkipIfJump if t0 > t1, skip 1 lines (pc incremented by 2*1)
        "1101000000011111",   --pc = 18 10010  j Else                jump to else
to avoid if statement
        "0101000100011000",   --pc = 19 10011  div v0 v0 0b1000      divide v0 by
8
        "0010001000010010",   --pc = 20 10100  or v1 v0 v1           or v1 and v0
        "1110100110010100",   --pc = 21 10101  sll t2 t2 4           shift t2
left four
        "0110100110011111",   --pc = 22 10110  addi t2 t2 0xF        add 1111 to
t2
```

```
        "1110100110010100",   --pc = 23 10111  sll t2 t2 4          shift t2
left four
        "0110100110011111",   --pc = 24 11000  addi t2 t2 0xF        add 1111 to
t2
        "1110100110010100",   --pc = 25 11001  sll t2 t2 4          shift t2
left four
        "0110100110010000",   --pc = 26 11010  addi t2 t2 0x0        add 0000 to
t2
        "1110100110010100",   --pc = 27 11011  sll t2 t2 4          shift t2
left four
        "0110100110010000",   --pc = 28 11100  addi t2 t2 0x0        add 0000 to
t2
        "1001100101010000",   --pc = 29 11101  sw t2 0(a0)          store t2
into a0
        "1101000000101010",   --pc = 30 11110  j Endif              jump endif
to avoid else statement
        "0100001100110100",   --pc = 31 11111  mul v2 v2 4          multiply v2
by four
        "0011010000110100",   --pc = 32 100000  xor v3 v2 v3        xor v3 and
v2
        "1110100110010100",   --pc = 33 100001  sll t2 t2 4         shift l2
left four
        "0110100110010000",   --pc = 34 100010  addi t2 t2 0x0       add 0000 to
t2
        "1110100110010100",   --pc = 35 100011  sll t2 t2 4         shift l2
left four
        "0110100110010000",   --pc = 36 100100  addi t2 t2 0x0       add 0000 to
t2
        "1110100110010100",   --pc = 37 100101  sll t2 t2 4         shift l2
left four
        "0110100110011111",   --pc = 38 100110  addi t2 t2 0xF       add 1111 to
t2
        "1110100110010100",   --pc = 39 100111  sll t2 t2 4         shift l2
left four
        "0110100110011111",   --pc = 40 101000  addi t2 t2 0xF       add 1111 to
t2
        "1001100101010000",   --pc = 41 101001  sw t2 0(a0)         store t2
into ao
        "0110010101010010",   --pc = 42 101010  addi a0 a0 2        add 2 to ao
        "1101000000000101"    --pc = 43 101011  j Loop              jump to Loop
-40 lines
         );


begin
```

```vhdl
   process (InstAddress)
   begin

        if (to_integer(unsigned(InstAddress)) < 44) then
            InstData <= Memory(to_integer(unsigned(InstAddress)));
        else
--              InstData <= Memory(0);
              InstData <= "1110000000000000"; --Nop insertion.
        end if;

--          InstData <= Memory(to_integer(unsigned(InstAddress)))
--          WHEN to_integer(unsigned(InstAddress)) < 64
--          ELSE Memory(0);

   end process;

end Behavioral;
```

JumpMux:

```vhdl
----------------------------------------------------------------------------------
-
-- Company: 
-- Engineer: 
-- 
-- Create Date: 10/28/2022 07:13:55 PM
-- Design Name: 
-- Module Name: JumpMux - Behavioral
-- Project Name: 
-- Target Devices: 
-- Tool Versions: 
-- Description: 
-- 
-- Dependencies: 
-- 
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
-- 
----------------------------------------------------------------------------------
-


library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity JumpMux is
  Port (Jump : in STD_LOGIC;
        ShiftJumpSignalOut : in std_logic_vector(12 downto 0);
        PCMostSig : in std_logic_vector(2 downto 0);
        --JumpAddressIn : in STD_LOGIC_VECTOR(15 downto 0);
        BranchMuxIn : in STD_LOGIC_VECTOR(15 downto 0);
        PCOut   : out STD_LOGIC_VECTOR(15 downto 0));
end JumpMux;
```

```vhdl
architecture Behavioral of JumpMux is
signal JumpAddressIn : std_logic_vector(15 downto 0);
begin
JumpAddressIn <= PCMostSig & ShiftJumpSignalOut;
process (JumpAddressIn, BranchMuxIn)
--Variable TempInv : std_logic_vector(15 downto 0);
   begin
    if (Jump = '0') then
        PCOut <= BranchMuxIn;
    elsif (Jump = '1') then
        PCOut <= JumpAddressIn;
    else
        --PCOut <= "HHHHHHHHHHHHHHHH";
        PCOut <= "0000000000000000";
    end if;
    end process;

end Behavioral;
```

MemToRegMux:

```
----------------------------------------------------------------------------------
-
-- Company: 
-- Engineer: 
-- 
-- Create Date: 10/27/2022 06:52:22 PM
-- Design Name: 
-- Module Name: MemToRegMux - Behavioral
-- Project Name: 
-- Target Devices: 
-- Tool Versions: 
-- Description: 
-- 
-- Dependencies: 
-- 
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
-- 
----------------------------------------------------------------------------------
-


library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity MemToRegMux is
  Port (ReadDataMem : in STD_LOGIC_VECTOR(15 downto 0);
        ALUResult : in STD_LOGIC_VECTOR(15 downto 0);
        MemToReg  : in STD_LOGIC;
        MemToRegMuxOut   : out STD_LOGIC_VECTOR(15 downto 0));
end MemToRegMux;

architecture Behavioral of MemToRegMux is
```

```
begin

   process (ReadDataMem, MemToReg, ALUResult)
   begin
    if (MemToReg = '0') then
        MemToRegMuxOut <= ALUResult;
    elsif (MemToReg = '1') then
        MemToRegMuxOut <= ReadDataMem;
    else
        MemToRegMuxOut <= "HHHHHHHHHHHHHHHH";
    end if;
    --MemToRegMuxOut <= ReadDataMem WHEN MemToReg = '0' ELSE ALUResult;
    end process;

end Behavioral;
```

PC_Component:

```
----------------------------------------------------------------------------------
-- Company:
-- Engineer:

-- Create Date: 10/22/2022 04:02:45 PM
-- Design Name:
-- Module Name: PC_Component - Behavioral
-- Project Name:
-- Target Devices:
-- Tool Versions:
-- Description:

-- Dependencies:

-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;


entity PC_Component is
    Port ( clk : in  STD_LOGIC;
           pc_in : in  STD_LOGIC_VECTOR (15 downto 0);
           pc_out : inout  STD_LOGIC_VECTOR (15 downto 0));
end PC_Component;

architecture Behavioral of PC_Component is

begin

process (clk, pc_in)
    variable temp : std_logic_vector(15 downto 0) := "0000000000000000";
    begin
    --if rising_edge(clk) then
    if falling_edge(clk) then
       temp := pc_in;
    else
       temp := temp;
       --pc_out <= pc_in ;--after 49ns;
    end if;
    pc_out <= temp;-- after 49ns;
end process;
end Behavioral;
```

RegFile:

```
----------------------------------------------------------------------------------
-
-- Company:
-- Engineer:
--
-- Create Date: 10/27/2022 04:51:27 PM
-- Design Name:
-- Module Name: RegFile - Behavioral
-- Project Name:
-- Target Devices:
-- Tool Versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------
-


library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity RegFile is
  Port (clk : in STD_LOGIC;
        InstData : in STD_LOGIC_VECTOR(15 downto 0);
        ReadReg1 : in STD_LOGIC_VECTOR(3 downto 0);
        ReadReg2 : in STD_LOGIC_VECTOR(3 downto 0);
        WriteReg : in STD_LOGIC_VECTOR(3 downto 0);
        WriteData : in STD_LOGIC_VECTOR(15 downto 0);
        RegWriteCtrl : in STD_LOGIC;
        ReadData1 : out STD_LOGIC_VECTOR(15 downto 0);
        ReadData2 : out STD_LOGIC_VECTOR(15 downto 0));
end RegFile;

architecture Behavioral of RegFile is

 type RegMemArray is array(0 to 15) of STD_LOGIC_VECTOR(15 downto 0);
 signal Reg_File : RegMemArray := (
    "0000000000000000", --$Zero 0 0000
    "0000000001000000", --$v0   1 0001
    "0001000000010000", --$v1   2 0010
    "0000000000001111", --$v2   3 0011
    "0000000011110000", --$v3   4 0100
    "0000000000010000", --$a0   5 0101
    "0000000000000101", --$a1   6 0110
    "0000000000000000", --$t0   7 0111
    "0000000000000000", --$t1   8 1000
    "0000000000000000", --$t2   9 1001
    "0000000000000000", --$t3   10
    "0000000000000000", --$t4   11
    "0000000000000000", --$t5   12
    "0000000000000000", --$s0   13
    "0000000000000000", --$s1   14
    "0000000000000000" -- $ra   15
 );


begin

    -- Outputting the Register data by converting the ReadReg 4 bits to an inte-
ger index.
    process (InstData)
    begin
        --if (rising_edge(clk)) then
            ReadData1 <= Reg_File(to_integer(unsigned(ReadReg1)));
            ReadData2 <= Reg_File(to_integer(unsigned(ReadReg2)));
        --end if;
    end process;

process(WriteData)
    begin
    --Check if we are at the rising edge of the clock and that the Reg write ena-
ble signal is '1'.
    --if (rising_edge(clk) and RegWriteCtrl = '1' and not(WriteReg = "0000"))
then
    --    Reg_File(to_integer(unsigned(WriteReg))) <= WriteData after 49ns;
    --else
    --    Reg_File(to_integer(unsigned(WriteReg))) <= Reg_File(to_integer(un-
signed(WriteReg)));
    --end if;
    if (RegWriteCtrl = '1' and not(WriteReg = "0000")) then
        Reg_File(to_integer(unsigned(WriteReg))) <= WriteData after
49ns;
    else

        Reg_File(to_integer(unsigned(WriteReg))) <= Reg_File(to_integer(un-
signed(WriteReg)));
end
if;

end process;
end Behavioral;
```

ShiftBranch:

```
----------------------------------------------------------------------------------
-
-- Company: 
-- Engineer: 
-- 
-- Create Date: 10/28/2022 12:28:59 AM
-- Design Name: 
-- Module Name: ShiftBranch - Behavioral
-- Project Name: 
-- Target Devices: 
-- Tool Versions: 
-- Description: 
-- 
-- Dependencies: 
-- 
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
-- 
----------------------------------------------------------------------------------
-


library IEEE;
use IEEE.STD_LOGIC_1164.ALL;


-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity ShiftBranch is
  Port (SignExtend : in STD_LOGIC_VECTOR(15 downto 0);
        ShiftOutBranch    : out STD_LOGIC_VECTOR(15 downto 0));
end ShiftBranch;

architecture Behavioral of ShiftBranch is

begin
```

```
--ShiftOutBranch <= SignExtend(14 downto 0) & '0'; --Move bits over one and con-
catenate with 0 on LSB.
process (SignExtend)
begin
    ShiftOutBranch <= SignExtend(15 downto 0);

end process;
end Behavioral;
```

ShiftJump:

```
----------------------------------------------------------------------------------
-- Company: 
-- Engineer: 
-- 
-- Create Date: 10/28/2022 12:24:19 AM
-- Design Name: 
-- Module Name: ShiftJump - Behavioral
-- Project Name: 
-- Target Devices: 
-- Tool Versions: 
-- Description: 
-- 
-- Dependencies: 
-- 
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
-- 
----------------------------------------------------------------------------------


library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity ShiftJump is
  Port (ShiftAddress : in STD_LOGIC_VECTOR(11 downto 0);
        ShiftOut    : out STD_LOGIC_VECTOR(12 downto 0));
end ShiftJump;

architecture Behavioral of ShiftJump is

begin
```

```vhdl
--ShiftOut <= ShiftAddress & '0'; --Move bits over one and concatenate with 0 on
LSB.
process(ShiftAddress)
begin
    ShiftOut <= '0' & ShiftAddress; -- Shouldn't need to shift.

end process;

end Behavioral;
```

SignExtension:

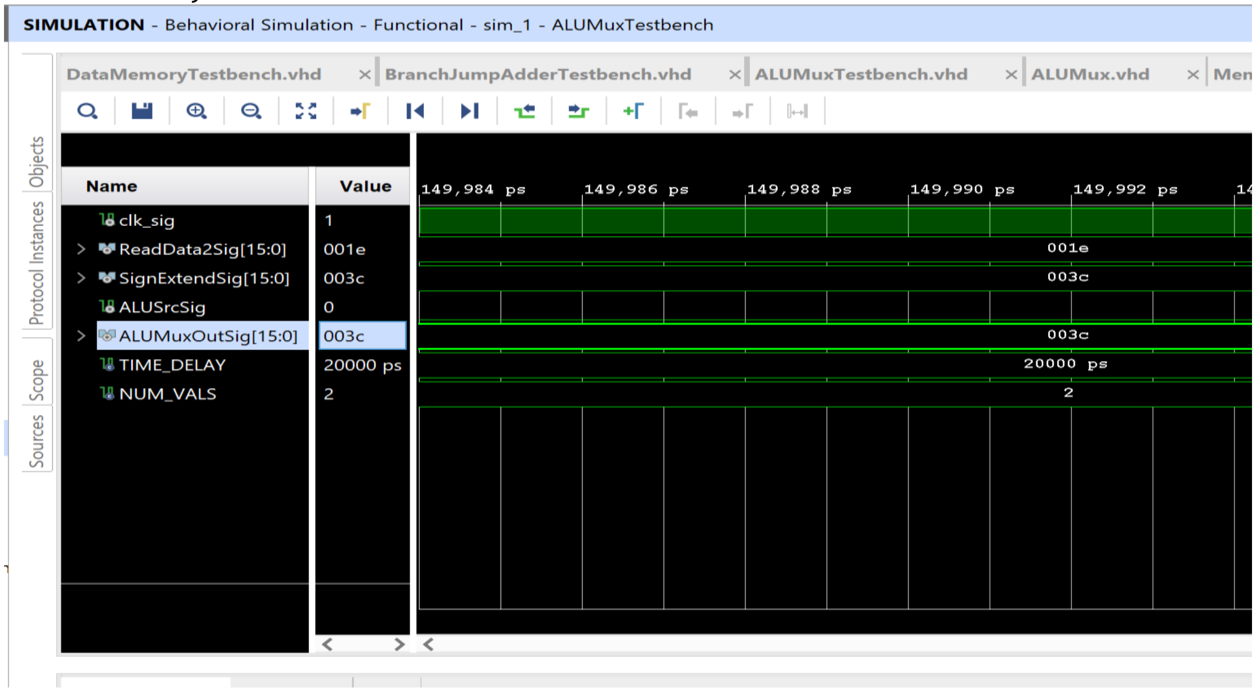
```
---------------------------------------------------------------------------------
-
-- Company:
-- Engineer:
--
-- Create Date: 10/27/2022 06:29:49 PM
-- Design Name:
-- Module Name: SignExtension - Behavioral
-- Project Name:
-- Target Devices:
-- Tool Versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
---------------------------------------------------------------------------------
-


library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity SignExtension is
  Port (DataIn : in STD_LOGIC_VECTOR(3 downto 0);
        ExtendedData : out STD_LOGIC_VECTOR(15 downto 0));
end SignExtension;

architecture Behavioral of SignExtension is

begin
```

```
process(DataIn)
begin

    if (DataIn(3) = '1') then
        ExtendedData <= "111111111111" & DataIn;
    else
        ExtendedData <= "000000000000" & DataIn;

    end if;
-- ExtendedData <= "111111111111" & DataIn WHEN DataIn(3) = '1' ELSE
"000000000000" & DataIn;

end process;


end Behavioral;
```

Testbenches:

ALUTestbench:

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity ALUTestbench is
end ALUTestbench;

architecture behavior of ALUTestbench is
    constant TIME_DELAY : time := 20 ns;
    constant NUM_VALS : integer := 2;


    type ALUOpCode_array is array(0 to (NUM_VALS - 1)) of std_logic_vector(3
downto 0);
    constant ALUOpCode_vals : ALUOpCode_array := ("0110", "0111");

    type ReadData1_array is array(0 to (NUM_VALS - 1)) of std_logic_vector(15
downto 0);
    constant ReadData1_vals : ReadData1_array :=
("0000000011110000","0000000000001111");

    type RegToALUMuxIn_array is array(0 to (NUM_VALS - 1)) of std_logic_vector(15
downto 0);
    constant RegToALUMuxIn_vals : RegToALUMuxIn_array :=
("0000000011110000","0000000000001111");

    signal ReadData1Sig : STD_LOGIC_VECTOR(15 downto 0) := "0000000000000000";
    signal RegToALUMuxInSig : STD_LOGIC_VECTOR(15 downto 0) :=
"0000000000000000";
    signal ALUResultOutSig : STD_LOGIC_VECTOR(15 downto 0) := "0000000000000000";
    signal ALUctrOpCodeSig : std_logic_vector (3 downto 0) := "0010";
    signal ZeroSig : std_logic := '0';
    signal clk_sig : std_logic := '0';


    begin
    DUT : entity work.ALU(behavior) port map (
                                          ReadData1 => ReadData1Sig,
                                          RegToALUMuxIn => RegToALUMuxInSig,
                                          ALUResultOut => ALUResultOutSig,
                                          ALUctrOpCode => ALUctrOpCodeSig,
                                                        Zero => ZeroSig);

        clock : process
          begin
            for i in 0 to 2 * (NUM_VALS) loop
              clk_sig <= NOT clk_sig;
              wait for TIME_DELAY/2;
            end loop;
            wait;
          end process clock;

         stimulus : process
          begin
            for i in 0 to (NUM_VALS - 1) loop
              ReadData1Sig <= ReadData1_vals(i);
              RegToALUMuxInSig <= RegToALUMuxIn_vals(i);
              ALUctrOpCodeSig <= ALUOpCode_vals(i);
              wait for TIME_DELAY;
            end loop;
            wait;
      end process stimulus;


end behavior;
```


SIMULATION - Behavioral Simulation - Functional - sim_1 - ALUMuxTestbench

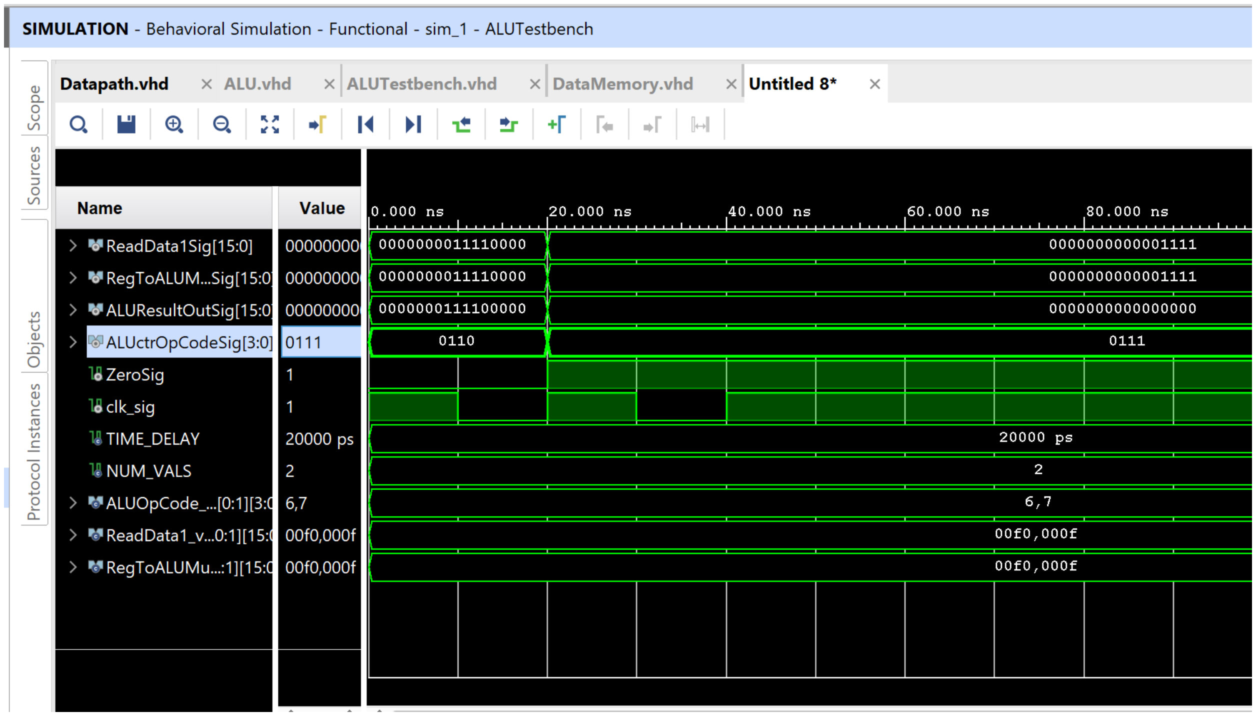
DataMemoryTestbench.vhd × BranchJumpAdderTestbench.vhd × ALUMuxTestbench.vhd × ALUMux.vhd × Men

Name	Value
clk_sig	1
ReadData2Sig[15:0]	001e
SignExtendSig[15:0]	003c
ALUSrcSig	0
ALUMuxOutSig[15:0]	003c
TIME_DELAY	20000 ps
NUM_VALS	2

ALUMuxTestbench:

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity ALUMuxTestbench is
end ALUMuxTestbench;

architecture behavior of ALUMuxTestbench is
    constant TIME_DELAY : time := 20 ns;
    constant NUM_VALS : integer := 2;


    --type InstAddress_array is array(0 to (NUM_VALS - 1)) of std_logic_vector(15
downto 0);
    --type pc_out_array is array(0 to (NUM_VALS - 1)) of std_logic_vector(15
downto 0);

    --constant InstAddress_vals : InstAddress_array :=
("0000000000000000","0000000000000001");



    signal clk_sig : std_logic := '0';
    signal ReadData2Sig : std_logic_vector (15 downto 0) := "0000000000011110";
    signal SignExtendSig : std_logic_vector (15 downto 0) := "0000000000111100";
    signal ALUSrcSig : std_logic := '0';
    signal ALUMuxOutSig : std_logic_vector (15 downto 0) := "0000000000111100";

    begin
    DUT : entity work.ALUMux(behavioral) port map (
                                                ReadData2  => ReadData2Sig,
                                                SignExtend => SignExtendSig,
                                                ALUSrc     => ALUSrcSig,
                                                ALUMuxOut  => ALUMuxOutSig);


        clock : process
          begin
            for i in 0 to 2 * (NUM_VALS) loop
              clk_sig <= NOT clk_sig;
              wait for TIME_DELAY/2;
            end loop;
            wait;
          end process clock;
```

```
         stimulus : process
          begin
            for i in 0 to (NUM_VALS - 1) loop
                ReadData2Sig <= ReadData2Sig;
                SignExtendSig <= SignExtendSig;
                ALUMuxOutSig  <=  ALUMuxOutSig;
              wait for TIME_DELAY;
            end loop;
            wait;
  end process stimulus;


end behavior;
```



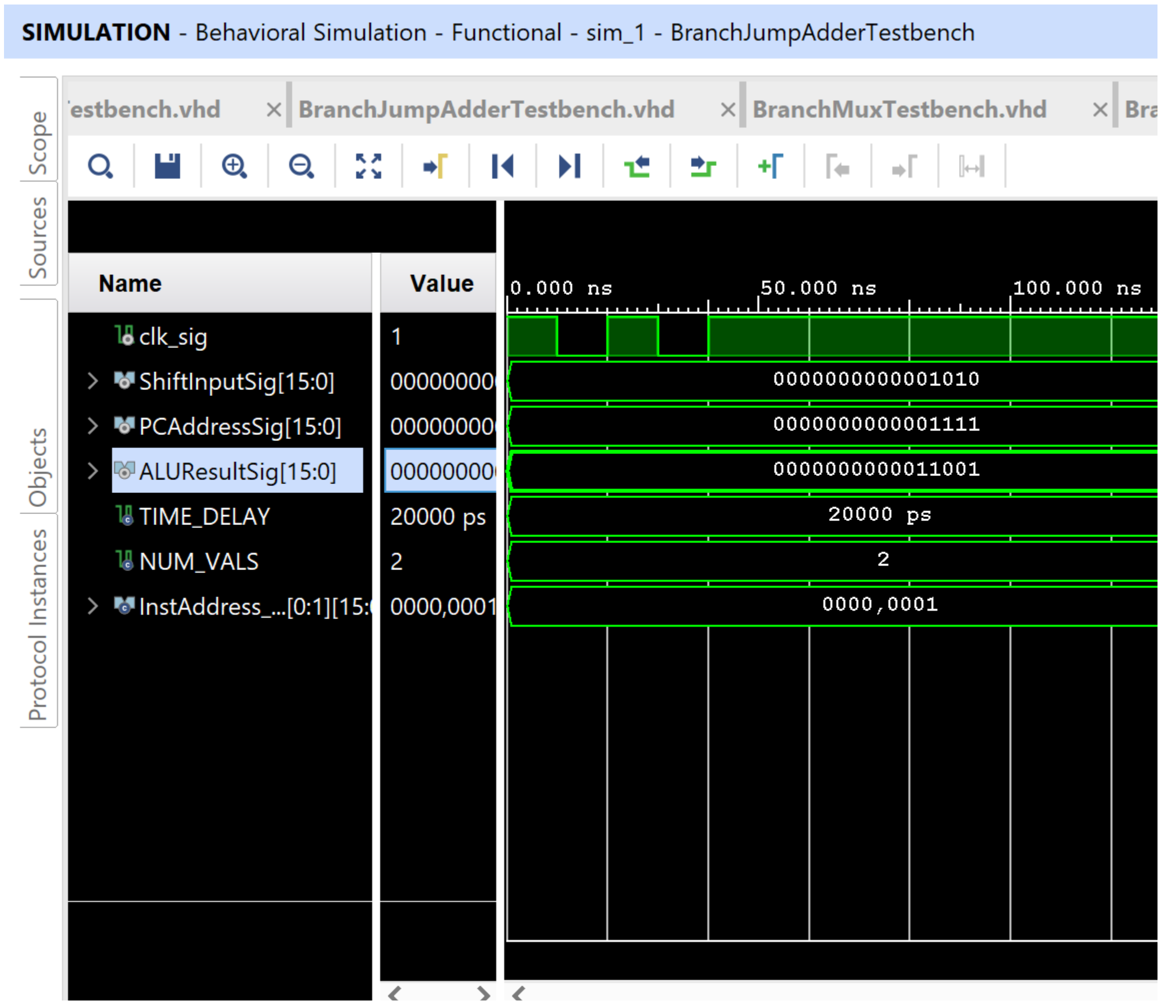
SIMULATION - Behavioral Simulation - Functional - sim_1 - ALUTestbench

Datapath.vhd | ALU.vhd | ALUTestbench.vhd | DataMemory.vhd | Untitled 8*

```
Name                      Value       0.000 ns    20.000 ns   40.000 ns   60.000 ns   80.000 ns
ReadData1Sig[15:0]        00000000    0000000011110000 | 0000000000001111
RegToALUM...Sig[15:0]     00000000    0000000011110000 | 0000000000001111
ALUResultOutSig[15:0]     00000000    0000000111100000 | 0000000000000000
ALUctrOpCodeSig[3:0]      0111        0110 | 0111
ZeroSig                   1
clk_sig                   1
TIME_DELAY                20000 ps    20000 ps
NUM_VALS                  2           2
ALUOpCode_...[0:1][3:0]   6,7         6,7
ReadData1_v...0:1][15:0]  00f0,000f   00f0,000f
RegToALUMu...:1][15:0]    00f0,000f   00f0,000f
```

BranchJumpAdderTestbench:
```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity BranchJumpAdderTestbench is
end BranchJumpAdderTestbench;

architecture behavior of BranchJumpAdderTestbench is
    constant TIME_DELAY : time := 20 ns;
    constant NUM_VALS : integer := 2;


    type InstAddress_array is array(0 to (NUM_VALS - 1)) of std_logic_vector(15
downto 0);
    --type pc_out_array is array(0 to (NUM_VALS - 1)) of std_logic_vector(15
downto 0);

    constant InstAddress_vals : InstAddress_array :=
("0000000000000000","0000000000000001");



    signal clk_sig : std_logic := '0';
    signal ShiftInputSig : std_logic_vector (15 downto 0) := "0000000000001010";
    signal PCAddressSig : std_logic_vector (15 downto 0) := "0000000000001111";
    signal ALUResultSig : std_logic_vector (15 downto 0) := "0000000000000000";

    begin
    DUT : entity work.BranchJumpAdder(behavioral) port map (
                                         ShiftInput =>  ShiftInputSig,
                                         PCAddress => PCAddressSig,
                                         ALUResult => ALUResultSig);


        clock : process
          begin
            for i in 0 to 2 * (NUM_VALS) loop
              clk_sig <= NOT clk_sig;
              wait for TIME_DELAY/2;
            end loop;
            wait;
          end process clock;

         stimulus : process
```

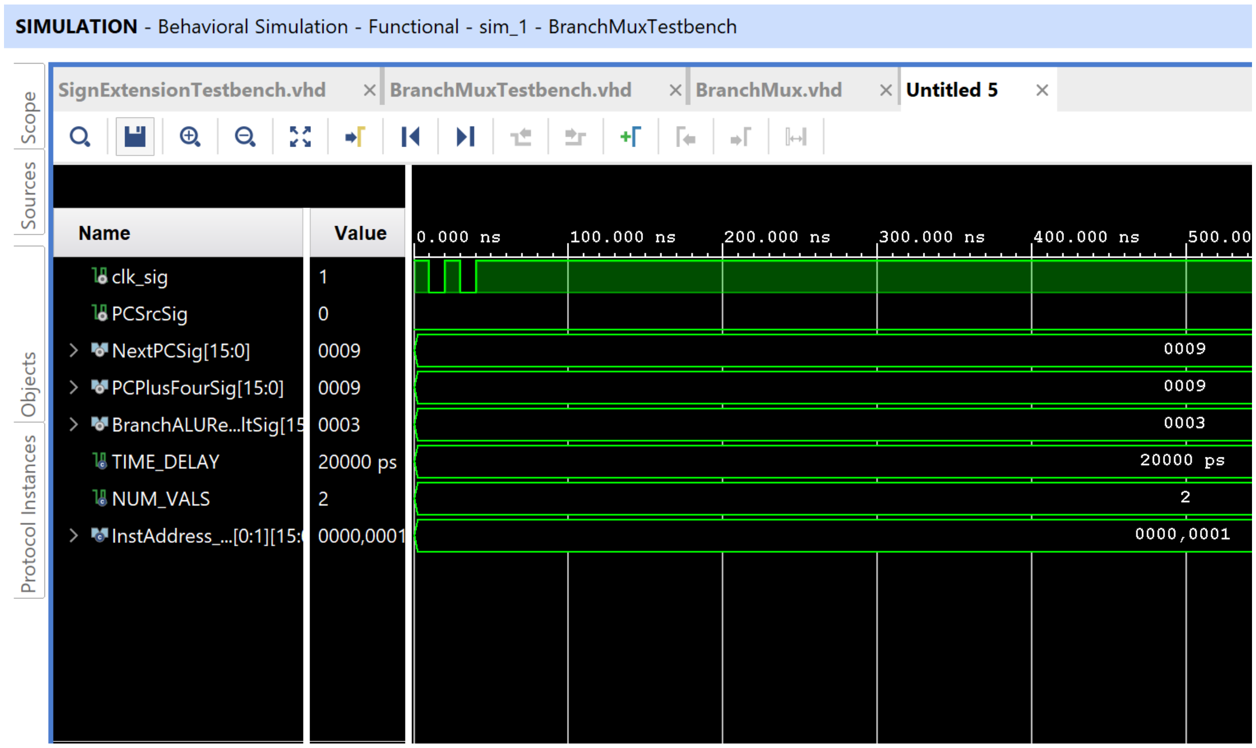
```
            begin
              for i in 0 to (NUM_VALS - 1) loop
                ShiftInputSig <= ShiftInputSig;
                PCAddressSig <= PCAddressSig;
                wait for TIME_DELAY;
              end loop;
              wait;
      end process stimulus;



  end behavior;
```


SIMULATION - Behavioral Simulation - Functional - sim_1 - BranchJumpAdderTestbench

Name	Value	
clk_sig	1	
ShiftInputSig[15:0]	00000000	0000000000001010
PCAddressSig[15:0]	00000000	0000000000001111
ALUResultSig[15:0]	00000000	0000000000011001
TIME_DELAY	20000 ps	20000 ps
NUM_VALS	2	2
InstAddress_...[0:1][15:	0000,0001	0000,0001

BranchMuxTestbench:

```vhdl
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity BranchMuxTestbench is
end BranchMuxTestbench;

architecture behavior of BranchMuxTestbench is
    constant TIME_DELAY : time := 20 ns;
    constant NUM_VALS : integer := 2;


    type InstAddress_array is array(0 to (NUM_VALS - 1)) of std_logic_vector(15
downto 0);
    --type pc_out_array is array(0 to (NUM_VALS - 1)) of std_logic_vector(15
downto 0);

    constant InstAddress_vals : InstAddress_array :=
("0000000000000000","0000000000000001");



    signal clk_sig : std_logic := '0';
    signal PCSrcSig : std_logic := '0';
    signal NextPCSig : std_logic_vector (15 downto 0) := "0000000000000000";
    signal PCPlusFourSig : std_logic_vector (15 downto 0) := "0000000000001001";
    signal BranchALUResultSig : std_logic_vector (15 downto 0) :=
"0000000000000011";
    begin
    DUT : entity work.BranchMux(behavioral) port map (
                                             PCPlusFour => PCPlusFourSig,
                                             BranchALUResult => BranchALURe-
sultSig,
                                             PCSrc => PCSrcSig,
                                             NextPC => NextPCSig);


        clock : process
          begin
            for i in 0 to 2 * (NUM_VALS) loop
              clk_sig <= NOT clk_sig;
              wait for TIME_DELAY/2;
            end loop;
            wait;
          end process clock;

         stimulus : process
          begin
            for i in 0 to (NUM_VALS - 1) loop
              PCPlusFourSig <= PCPlusFourSig;
              BranchALUResultSig <= BranchALUResultSig;
              wait for TIME_DELAY;
            end loop;
            wait;
  end process stimulus;



end behavior;
```

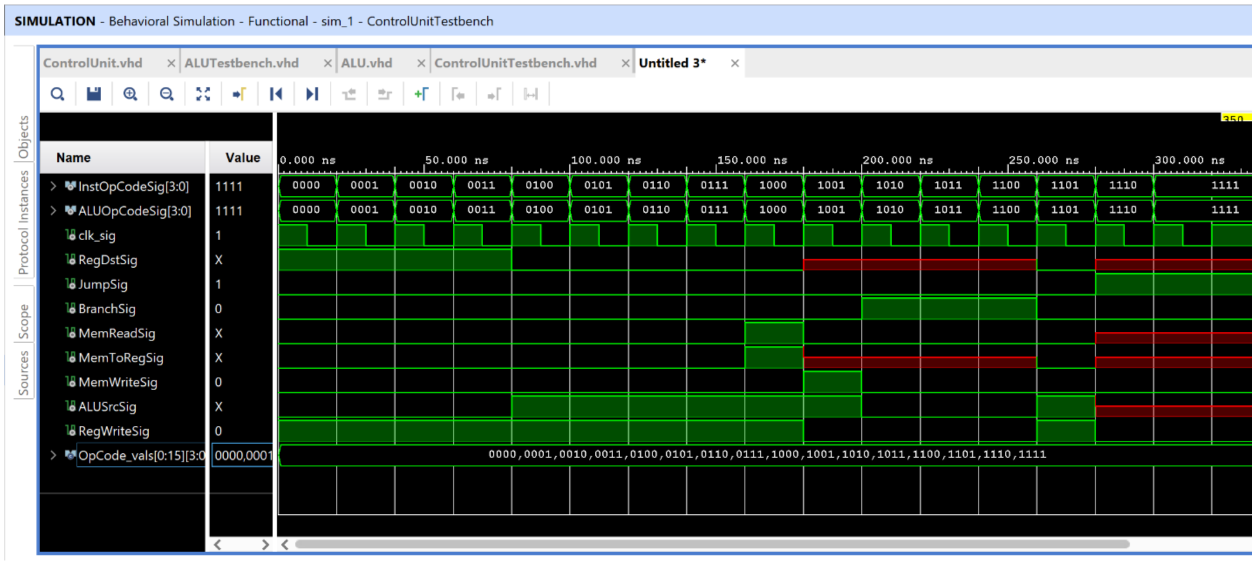


SIMULATION - Behavioral Simulation - Functional - sim_1 - BranchMuxTestbench

SignExtensionTestbench.vhd | BranchMuxTestbench.vhd | BranchMux.vhd | Untitled 5

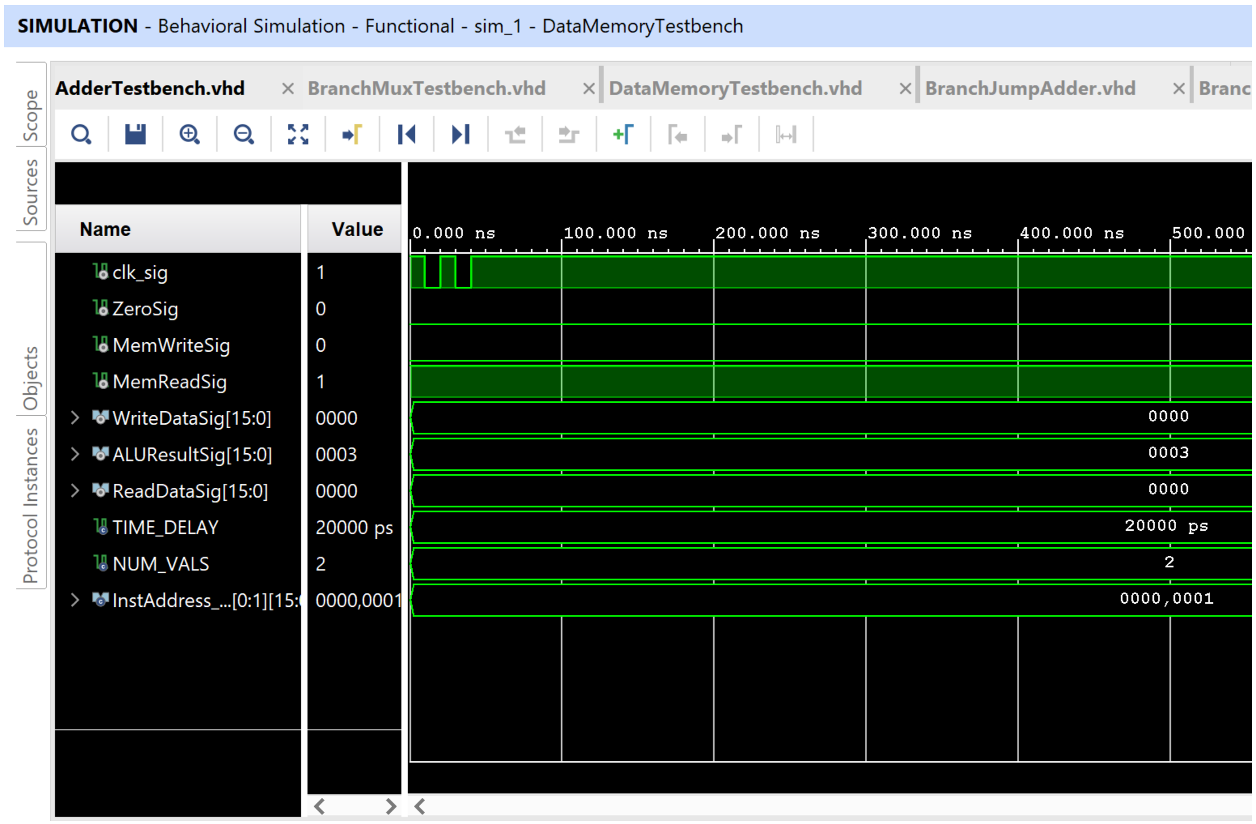
Name	Value
clk_sig	1
PCSrcSig	0
NextPCSig[15:0]	0009
PCPlusFourSig[15:0]	0009
BranchALURe...ltSig[15	0003
TIME_DELAY	20000 ps
NUM_VALS	2
InstAddress_...[0:1][15:	0000,0001

ControlUnitTestbench:
```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity ControlUnitTestbench is
end ControlUnitTestbench;

architecture behavioral of ControlUnitTestbench is
    constant TIME_DELAY : time := 20 ns;
    constant NUM_VALS : integer := 16;


    type OpCode_array is array(0 to (NUM_VALS - 1)) of std_logic_vector(3 downto
0);
    constant OpCode_vals : OpCode_array := ("0000", "0001", "0010", "0011",
"0100", "0101", "0110", "0111", "1000", "1001", "1010", "1011", "1100", "1101",
"1110", "1111");

--    type ReadData1_array is array(0 to (NUM_VALS - 1)) of std_logic_vector(15
downto 0);
--    constant ReadData1_vals : ReadData1_array :=
("0000000011110000","0000000000001111");

--    type RegToALUMuxIn_array is array(0 to (NUM_VALS - 1)) of std_logic_vec-
tor(15 downto 0);
--    constant RegToALUMuxIn_vals : RegToALUMuxIn_array :=
("0000000011110000","0000000000001111");
    signal InstOpCodeSig : std_logic_vector (3 downto 0) := "0000";
    signal ALUOpCodeSig : std_logic_vector (3 downto 0) := "0000";
    signal clk_sig : std_logic := '0';
    signal RegDstSig : std_logic := '0';
    signal JumpSig : std_logic := '0';
    signal BranchSig : std_logic := '0';
    signal MemReadSig : std_logic := '0';
    signal MemToRegSig : std_logic := '0';
    signal MemWriteSig : std_logic := '0';
    signal ALUSrcSig : std_logic := '0';
    signal RegWriteSig : std_logic := '0';


    begin
    DUT : entity work.ControlUnit(behavioral) port map (
                                                InstOpCode => InstOpCodeSig,
                                                ALUOpCode => ALUOpCodeSig,
                                             RegDst => RegDstSig,
                                             Jump => JumpSig,
                                             Branch => BranchSig,
                                             MemRead => MemReadSig,
                                             MemToReg => MemToRegSig,
                                             MemWrite => MemWriteSig,
                                             ALUSrc => ALUSrcSig,
                                             RegWrite => RegWriteSig
                                             );



        clock : process
          begin
            for i in 0 to 2 * (NUM_VALS) loop
              clk_sig <= NOT clk_sig;
              wait for TIME_DELAY/2;
            end loop;
            wait;
          end process clock;

         stimulus : process
          begin
            for i in 0 to (NUM_VALS - 1) loop
              InstOpCodeSig <= OpCode_vals(i);

              wait for TIME_DELAY;
            end loop;
            wait;
  end process stimulus;



end behavioral;
```


SIMULATION - Behavioral Simulation - Functional - sim_1 - ControlUnitTestbench
ControlUnit.vhd
ALUTestbench.vhd
ALU.vhd
ControlUnitTestbench.vhd
Untitled 3*
Objects
Protocol Instances
Scope
Sources
Name
Value
InstOpCodeSig[3:0] 1111
ALUOpCodeSig[3:0] 1111
clk_sig 1
RegDstSig X
JumpSig 1
BranchSig 0
MemReadSig X
MemToRegSig X
MemWriteSig 0
ALUSrcSig X
RegWriteSig 0
OpCode_vals[0:15][3:0] 0000,0001
0.000 ns
50.000 ns
100.000 ns
150.000 ns
200.000 ns
250.000 ns
300.000 ns
350
0000 0001 0010 0011 0100 0101 0110 0111 1000 1001 1010 1011 1100 1101 1110 1111
0000 0001 0010 0011 0100 0101 0110 0111 1000 1001 1010 1011 1100 1101 1110 1111
0000,0001,0010,0011,0100,0101,0110,0111,1000,1001,1010,1011,1100,1101,1110,1111

DataMemoryTestbench:

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity DataMemoryTestbench is
end DataMemoryTestbench;

architecture behavior of DataMemoryTestbench is
    constant TIME_DELAY : time := 20 ns;
    constant NUM_VALS : integer := 2;


    type InstAddress_array is array(0 to (NUM_VALS - 1)) of std_logic_vector(15
downto 0);
    --type pc_out_array is array(0 to (NUM_VALS - 1)) of std_logic_vector(15
downto 0);

    constant InstAddress_vals : InstAddress_array :=
("0000000000000000","0000000000000001");



    signal clk_sig : std_logic := '0';
    signal ZeroSig : std_logic := '0';
    signal MemWriteSig : std_logic := '0';
    signal MemReadSig : std_logic := '1';
    signal WriteDataSig : std_logic_vector (15 downto 0) := "0000000000000000";
    signal ALUResultSig : std_logic_vector (15 downto 0) := "0000000000000011";
    signal ReadDataSig : std_logic_vector (15 downto 0) := "0000000000000000";
    begin
    DUT : entity work.DataMemory(behavioral) port map (
                                             Zero => ZeroSig,
                                             WriteData => WriteDataSig,
                                             ALUResult => ALUResultSig,
                                             MemWrite => MemWriteSig,
                                             MemRead => MemReadSig);


        clock : process
          begin
            for i in 0 to 2 * (NUM_VALS) loop
              clk_sig <= NOT clk_sig;
              wait for TIME_DELAY/2;
            end loop;
            wait;
          end process clock;

         stimulus : process
          begin
            for i in 0 to (NUM_VALS - 1) loop
              ZeroSig <= ZeroSig;
              WriteDataSig <= WriteDataSig;
              ALUResultSig <= ALUResultSig;
              ReadDataSig <= ReadDataSig;
              wait for TIME_DELAY;
            end loop;
            wait;
  end process stimulus;



end behavior;
```

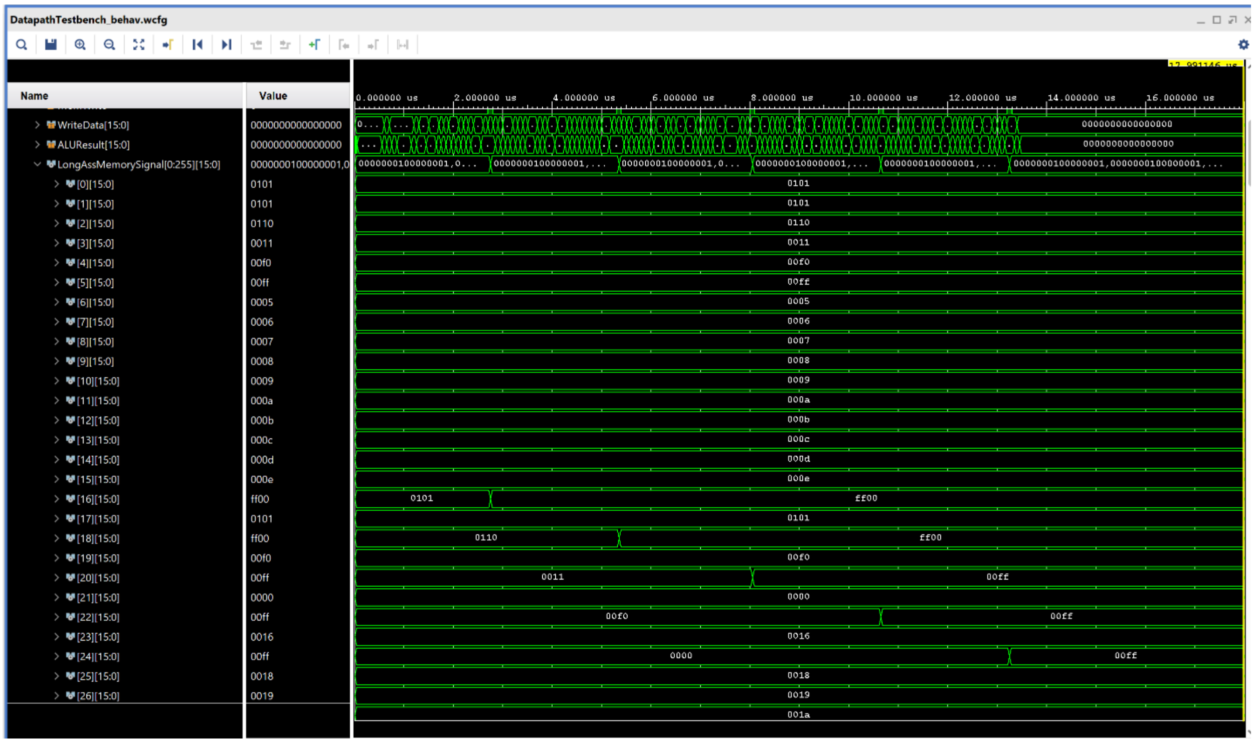


SIMULATION - Behavioral Simulation - Functional - sim_1 - DataMemoryTestbench

Name	Value
clk_sig	1
ZeroSig	0
MemWriteSig	0
MemReadSig	1
WriteDataSig[15:0]	0000
ALUResultSig[15:0]	0003
ReadDataSig[15:0]	0000
TIME_DELAY	20000 ps
NUM_VALS	2
InstAddress_...[0:1][15:	0000,0001

DatapathTestbench:

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity DatapathTestbench is
end DatapathTestbench;

architecture structural of DatapathTestbench is
    constant TIME_DELAY : time := 100 ns;
    constant NUM_VALS : integer := 150;

    signal clk_sig : std_logic := '1';

    begin
    DUT : entity work.Datapath(structural) port map (
                                            clk => clk_sig
                                            );


        clock : process
          begin
            for i in 0 to 2 * (NUM_VALS) loop
              clk_sig <= NOT clk_sig;
              wait for TIME_DELAY/2;
            end loop;
            wait;
          end process clock;

--          stimulus : process
--            begin
--              for i in 0 to (NUM_VALS - 1) loop
--                --ReadData1Sig <= ReadData1_vals(i);
--                --RegToALUMuxInSig <= RegToALUMuxIn_vals(i);
--                --ALUctrOpCodeSig <= ALUOpCode_vals(i);
--                wait for TIME_DELAY;
--              end loop;
--              wait;
--          end process stimulus;



end structural;
```

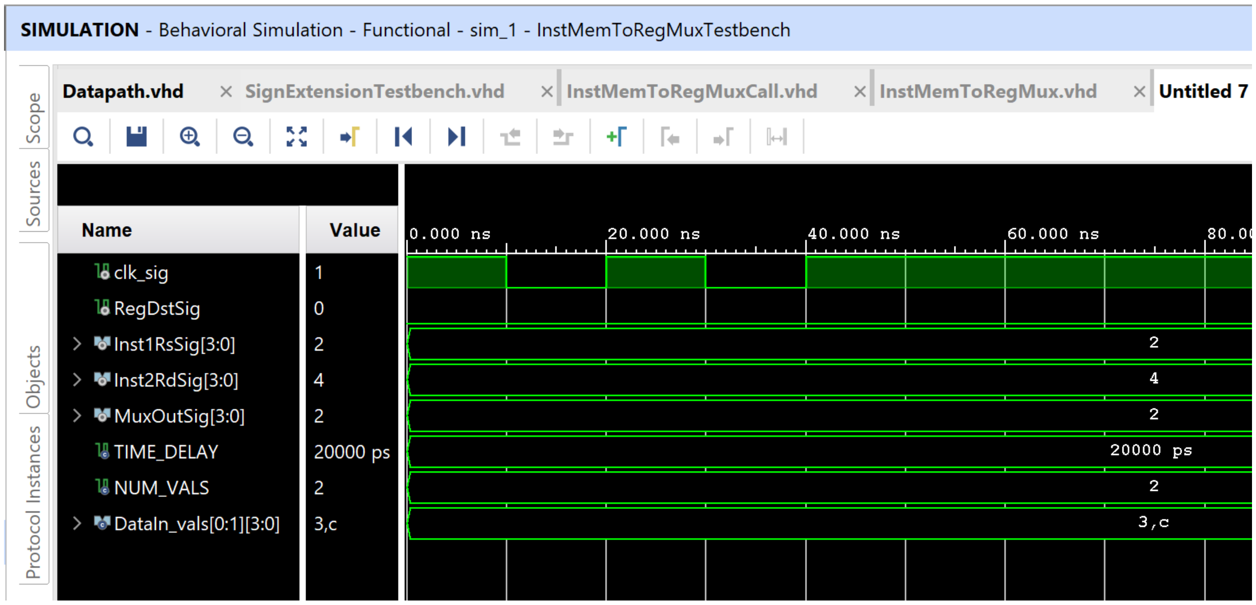


DatapathTestbench_behav.wcfg

```
Name                                  Value
WriteData[15:0]                       0000000000000000
ALUResult[15:0]                       0000000000000000
LongAssMemorySignal[0:255][15:0]      0000000100000001,0
  [0][15:0]                           0101
  [1][15:0]                           0101
  [2][15:0]                           0110
  [3][15:0]                           0011
  [4][15:0]                           00f0
  [5][15:0]                           00ff
  [6][15:0]                           0005
  [7][15:0]                           0006
  [8][15:0]                           0007
  [9][15:0]                           0008
  [10][15:0]                          0009
  [11][15:0]                          000a
  [12][15:0]                          000b
  [13][15:0]                          000c
  [14][15:0]                          000d
  [15][15:0]                          000e
  [16][15:0]                          ff00
  [17][15:0]                          0101
  [18][15:0]                          ff00
  [19][15:0]                          00f0
  [20][15:0]                          00ff
  [21][15:0]                          0000
  [22][15:0]                          00ff
  [23][15:0]                          0016
  [24][15:0]                          00ff
  [25][15:0]                          0018
  [26][15:0]                          0019
```

InstMemToRegMuxCall:
```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity InstMemToRegMuxTestbench is
end InstMemToRegMuxTestbench;

architecture behavior of InstMemToRegMuxTestbench is
    constant TIME_DELAY : time := 20 ns;
    constant NUM_VALS : integer := 2;


    type DataIn_array is array(0 to (NUM_VALS - 1)) of std_logic_vector(3 downto
0);
    --type pc_out_array is array(0 to (NUM_VALS - 1)) of std_logic_vector(15
downto 0);

    constant DataIn_vals : DataIn_array := ("0011","1100");



    signal clk_sig : std_logic := '0';
    signal RegDstSig : std_logic := '0';
    signal Inst1RsSig : std_logic_vector (3 downto 0) := "0010";
    signal Inst2RdSig : std_logic_vector (3 downto 0) := "0100";
    signal MuxOutSig : std_logic_vector (3 downto 0) := "0000";
    begin
    DUT : entity work.InstMemToRegMux(behavioral) port map (
                                        RegDst => RegDstSig,
                                        Inst1Rs => Inst1RsSig,
                                        Inst2Rd => Inst2RdSig,
                                        MuxOut => MuxOutSig);


        clock : process
          begin
            for i in 0 to 2 * (NUM_VALS) loop
              clk_sig <= NOT clk_sig;
              wait for TIME_DELAY/2;
            end loop;
            wait;
          end process clock;

         stimulus : process
```

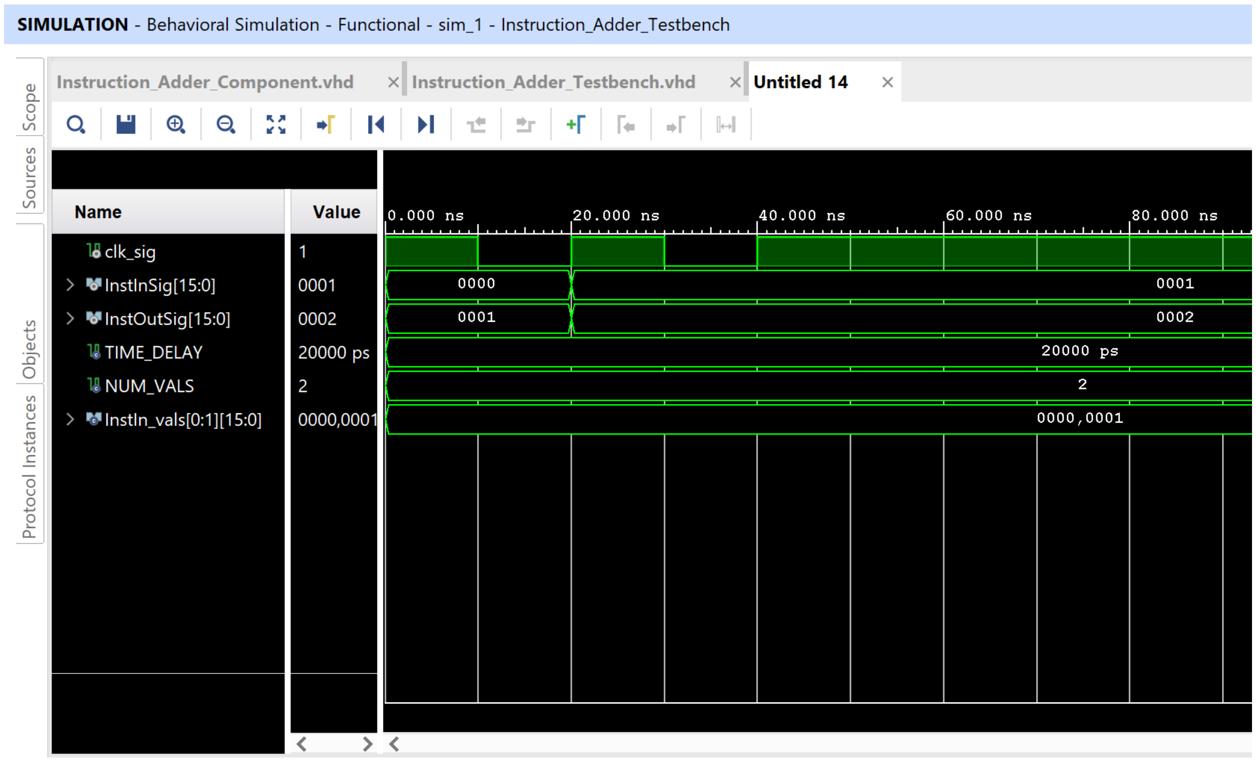
```
          begin
            for i in 0 to (NUM_VALS - 1) loop
              Inst1RsSig <= Inst1RsSig;
              Inst2RdSig <= Inst2RdSig;
              wait for TIME_DELAY;
            end loop;
            wait;
  end process stimulus;



end behavior;
```


SIMULATION - Behavioral Simulation - Functional - sim_1 - InstMemToRegMuxTestbench

Name	Value
clk_sig	1
RegDstSig	0
Inst1RsSig[3:0]	2
Inst2RdSig[3:0]	4
MuxOutSig[3:0]	2
TIME_DELAY	20000 ps
NUM_VALS	2
DataIn_vals[0:1][3:0]	3,c

Instruction_Adder_Testbench:
```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity Instruction_Adder_Testbench is
end Instruction_Adder_Testbench;

architecture behavior of Instruction_Adder_Testbench is
    constant TIME_DELAY : time := 20 ns;
    constant NUM_VALS : integer := 2;


    type InstIn_array is array(0 to (NUM_VALS - 1)) of std_logic_vector(15 downto
0);
    --type pc_out_array is array(0 to (NUM_VALS - 1)) of std_logic_vector(15
downto 0);

    constant InstIn_vals : InstIn_array :=
("0000000000000000","0000000000000001");



    signal clk_sig : std_logic := '0';
    signal InstInSig : std_logic_vector (15 downto 0) := "0000000000000000";
    signal InstOutSig : std_logic_vector (15 downto 0) := "0000000000000000";
    begin
    DUT : entity work.Instruction_Adder_Component(behavioral) port map (
                                         InstIn => InstInSig,
                                         InstOut => InstOutSig);


        clock : process
          begin
            for i in 0 to 2 * (NUM_VALS) loop
              clk_sig <= NOT clk_sig;
              wait for TIME_DELAY/2;
            end loop;
            wait;
          end process clock;

         stimulus : process
          begin
            for i in 0 to (NUM_VALS - 1) loop
              InstInSig <= InstIn_vals(i);
```

```
                wait for TIME_DELAY;
            end loop;
            wait;
    end process stimulus;


end behavior;
```


SIMULATION - Behavioral Simulation - Functional - sim_1 - Instruction_Adder_Testbench

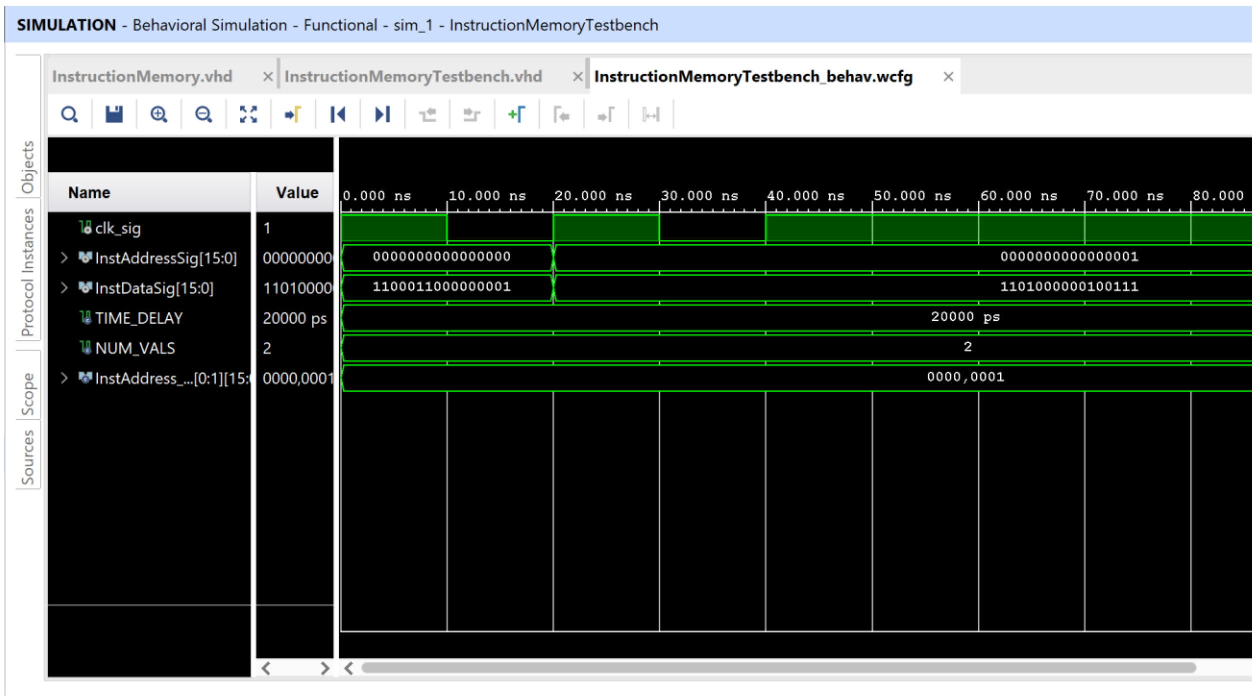
Instruction_Adder_Component.vhd | Instruction_Adder_Testbench.vhd | Untitled 14

Name	Value
clk_sig	1
InstInSig[15:0]	0001
InstOutSig[15:0]	0002
TIME_DELAY	20000 ps
NUM_VALS	2
InstIn_vals[0:1][15:0]	0000,0001

InstructionMemoryTestbench:

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity InstructionMemoryTestbench is
end InstructionMemoryTestbench;

architecture behavior of InstructionMemoryTestbench is
    constant TIME_DELAY : time := 20 ns;
    constant NUM_VALS : integer := 2;


    type InstAddress_array is array(0 to (NUM_VALS - 1)) of std_logic_vector(15
downto 0);
    --type pc_out_array is array(0 to (NUM_VALS - 1)) of std_logic_vector(15
downto 0);

    constant InstAddress_vals : InstAddress_array :=
("0000000000000000","0000000000000001");



    signal clk_sig : std_logic := '0';
    signal InstAddressSig : std_logic_vector (15 downto 0) := "0000000000000000";
    signal InstDataSig : std_logic_vector (15 downto 0) := "0000000000000000";
    begin
    DUT : entity work.InstructionMemory(behavioral) port map (
                                         InstAddress => InstAddressSig,
                                         InstData => InstDataSig);


        clock : process
          begin
            for i in 0 to 2 * (NUM_VALS) loop
              clk_sig <= NOT clk_sig;
              wait for TIME_DELAY/2;
            end loop;
            wait;
          end process clock;

         stimulus : process
          begin
            for i in 0 to (NUM_VALS - 1) loop
              InstAddressSig <= InstAddress_vals(i);
```

```
                wait for TIME_DELAY;
              end loop;
              wait;
   end process stimulus;


end behavior;
```



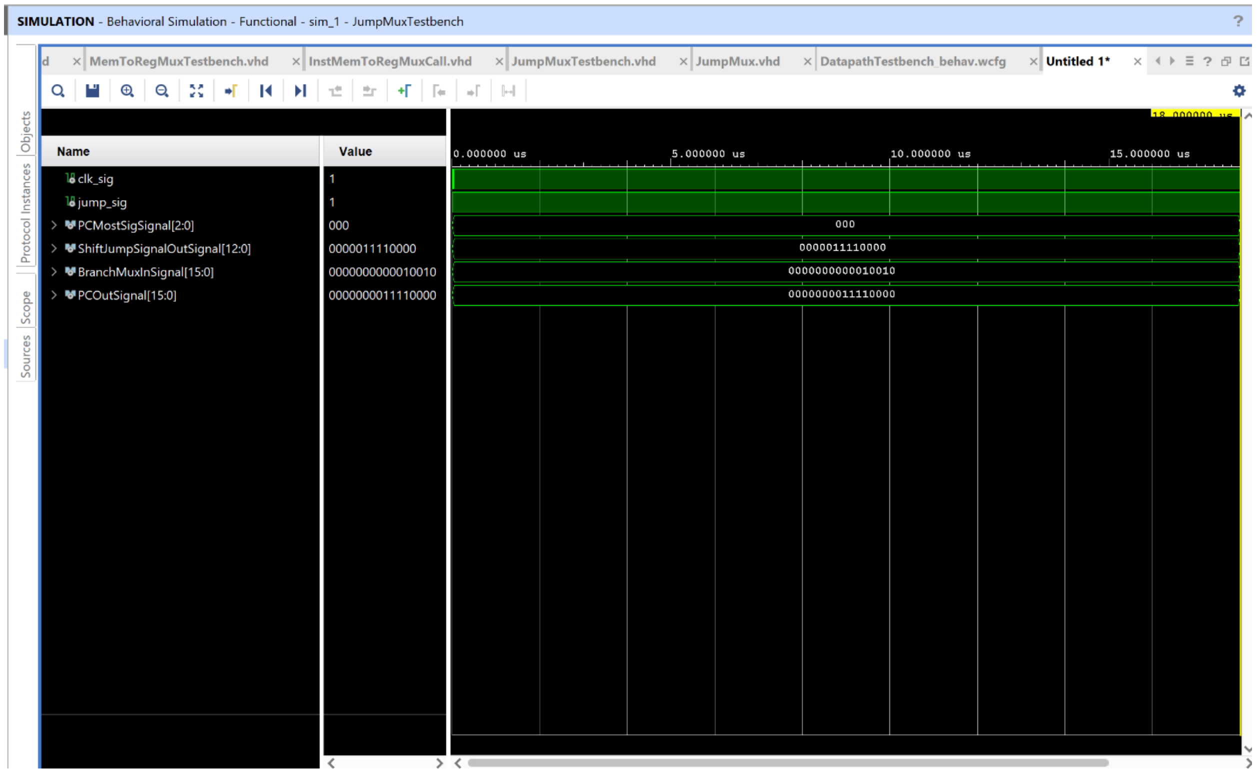
SIMULATION - Behavioral Simulation - Functional - sim_1 - InstructionMemoryTestbench

InstructionMemory.vhd | InstructionMemoryTestbench.vhd | InstructionMemoryTestbench_behav.wcfg

```
Name                      Value        0.000 ns  10.000 ns  20.000 ns  30.000 ns  40.000 ns  50.000 ns  60.000 ns  70.000 ns  80.000
clk_sig                   1
InstAddressSig[15:0]      00000000     0000000000000000      0000000000000001
InstDataSig[15:0]         11010000     1100011000000001      1101000000100111
TIME_DELAY                20000 ps     20000 ps
NUM_VALS                  2            2
InstAddress_...[0:1][15:  0000,0001    0000,0001
```

JumpMux:

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity JumpMuxTestbench is
end JumpMuxTestbench;

architecture behavior of JumpMuxTestbench is
    constant TIME_DELAY : time := 20 ns;
    constant NUM_VALS : integer := 2;


    type DataIn_array is array(0 to (NUM_VALS - 1)) of std_logic_vector(3 downto
0);
    --type pc_out_array is array(0 to (NUM_VALS - 1)) of std_logic_vector(15
downto 0);

    constant DataIn_vals : DataIn_array := ("0011","1100");


    signal jump_sig : std_logic := '1';
    signal clk_sig : std_logic := '0';
    signal ShiftJumpSignalOutSignal : std_logic_vector(12 downto 0) :=
"0000011110000";
    signal PCMostSigSignal : std_logic_vector(2 downto 0) := "000";
    signal BranchMuxInSignal : std_logic_vector(15 downto 0) :=
"0000000000010010";
    signal PCOutSignal : std_logic_vector(15 downto 0) := "0000000000000000";

    begin
    DUT : entity work.JumpMux(behavioral) port map (
                                            Jump => jump_sig,
                                            ShiftJumpSignalOut =>
ShiftJumpSignalOutSignal,
                                            PCMostSig => PCMostSigSignal,
                                            BranchMuxIn => BranchMuxInSignal,
                                            PCOut => PCOutSignal);


        clock : process
          begin
            for i in 0 to 2 * (NUM_VALS) loop
              clk_sig <= NOT clk_sig;
              wait for TIME_DELAY/2;
            end loop;
            wait;
          end process clock;

         stimulus : process
          begin
            for i in 0 to (NUM_VALS - 1) loop
              --Inst1RsSig <= Inst1RsSig;
              --Inst2RdSig <= Inst2RdSig;
              wait for TIME_DELAY;
            end loop;
            wait;
      end process stimulus;



  end behavior;
```



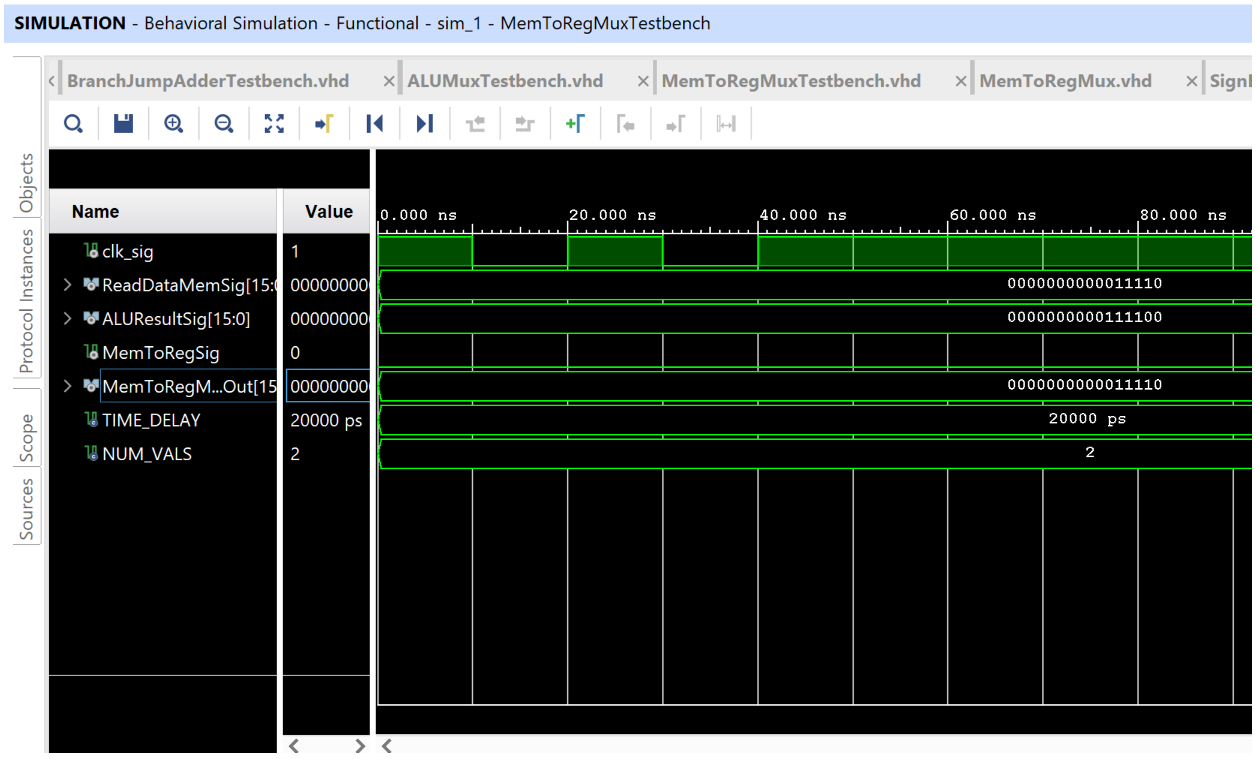
SIMULATION - Behavioral Simulation - Functional - sim_1 - JumpMuxTestbench

Name	Value
clk_sig	1
jump_sig	1
PCMostSigSignal[2:0]	000
ShiftJumpSignalOutSignal[12:0]	0000011110000
BranchMuxInSignal[15:0]	0000000000010010
PCOutSignal[15:0]	0000000011110000

MemToRegMuxTestbench:

```vhdl
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity MemToRegMuxTestbench is
end MemToRegMuxTestbench;

architecture behavior of MemToRegMuxTestbench is
    constant TIME_DELAY : time := 20 ns;
    constant NUM_VALS : integer := 2;


    --type InstAddress_array is array(0 to (NUM_VALS - 1)) of std_logic_vector(15
downto 0);
    --type pc_out_array is array(0 to (NUM_VALS - 1)) of std_logic_vector(15
downto 0);

    --constant InstAddress_vals : InstAddress_array :=
("0000000000000000","0000000000000001");



    signal clk_sig : std_logic := '0';
    signal ReadDataMemSig : std_logic_vector (15 downto 0) := "0000000000011110";
    signal ALUResultSig : std_logic_vector (15 downto 0) := "0000000000111100";
    signal MemToRegSig : std_logic := '0';
    signal MemToRegMuxOut : std_logic_vector (15 downto 0) := "0000000000000000";

    begin
    DUT : entity work.MemToRegMux(behavioral) port map (
                                           ReadDataMem  => ReadDataMemSig,
                                           ALUResult => ALUResultSig,
                                           MemToReg => MemToRegSig,
                                           MemToRegMuxOut => MemToRegMuxOut);
        clock : process
          begin
            for i in 0 to 2 * (NUM_VALS) loop
              clk_sig <= NOT clk_sig;
              wait for TIME_DELAY/2;
            end loop;
            wait;
          end process clock;

         stimulus : process
```

```
          begin
            for i in 0 to (NUM_VALS - 1) loop
              ReadDataMemSig <= ReadDataMemSig;
              ALUResultSig <= ALUResultSig;
              MemToRegSig <= MemToRegSig;
              wait for TIME_DELAY;
            end loop;
            wait;
  end process stimulus;


end behavior;
```



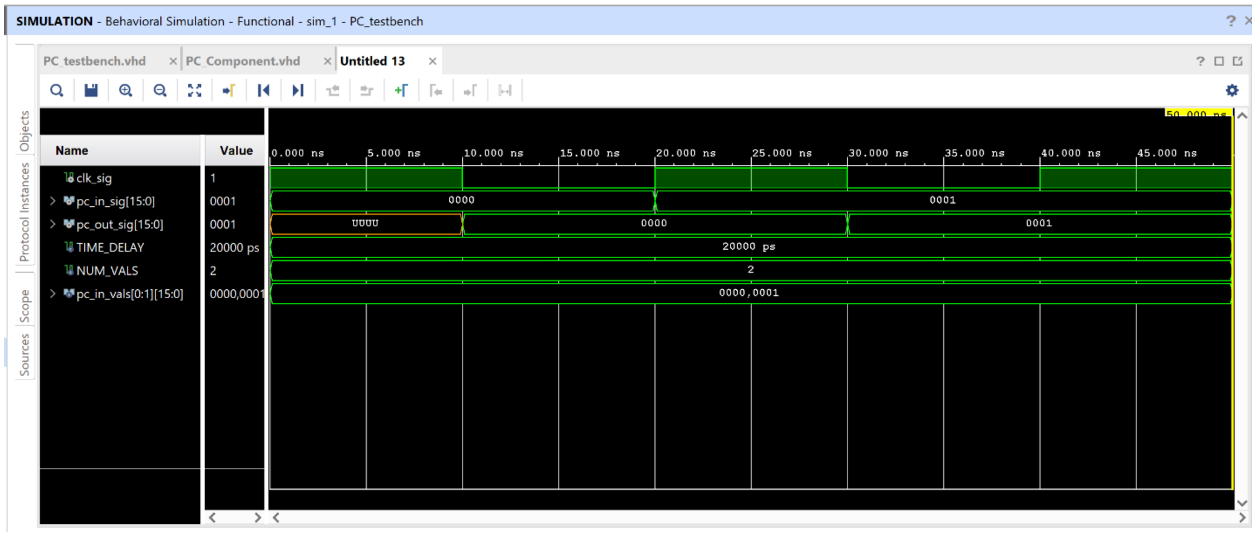
SIMULATION - Behavioral Simulation - Functional - sim_1 - MemToRegMuxTestbench

```
Name                    Value
clk_sig                 1
ReadDataMemSig[15:0]    00000000    0000000000011110
ALUResultSig[15:0]      00000000    0000000000111100
MemToRegSig             0
MemToRegM...Out[15      00000000    0000000000011110
TIME_DELAY              20000 ps    20000 ps
NUM_VALS                2           2
```

PC_testbench:

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity PC_testbench is
end PC_testbench;

architecture behavior of PC_testbench is
    constant TIME_DELAY : time := 20 ns;
    constant NUM_VALS : integer := 2;


    type pc_in_array is array(0 to (NUM_VALS - 1)) of std_logic_vector(15 downto
0);
    --type pc_out_array is array(0 to (NUM_VALS - 1)) of std_logic_vector(15
downto 0);

    constant pc_in_vals : pc_in_array := ("0000000000000000","0000000000000001");



    signal clk_sig : std_logic := '0';
    signal pc_in_sig : std_logic_vector (15 downto 0) := "0000000000000000";
    signal pc_out_sig : std_logic_vector (15 downto 0) := "0000000000000000";
    begin
    DUT : entity work.PC_Component(behavioral) port map (
                                            clk => clk_sig,
                                            pc_in => pc_in_sig,
                                            pc_out => pc_out_sig);


        clock : process
          begin
            for i in 0 to 2 * (NUM_VALS) loop
              clk_sig <= NOT clk_sig;
              wait for TIME_DELAY/2;
            end loop;
            wait;
          end process clock;

         stimulus : process
          begin
            for i in 0 to (NUM_VALS - 1) loop
              pc_in_sig <= pc_in_vals(i);
```

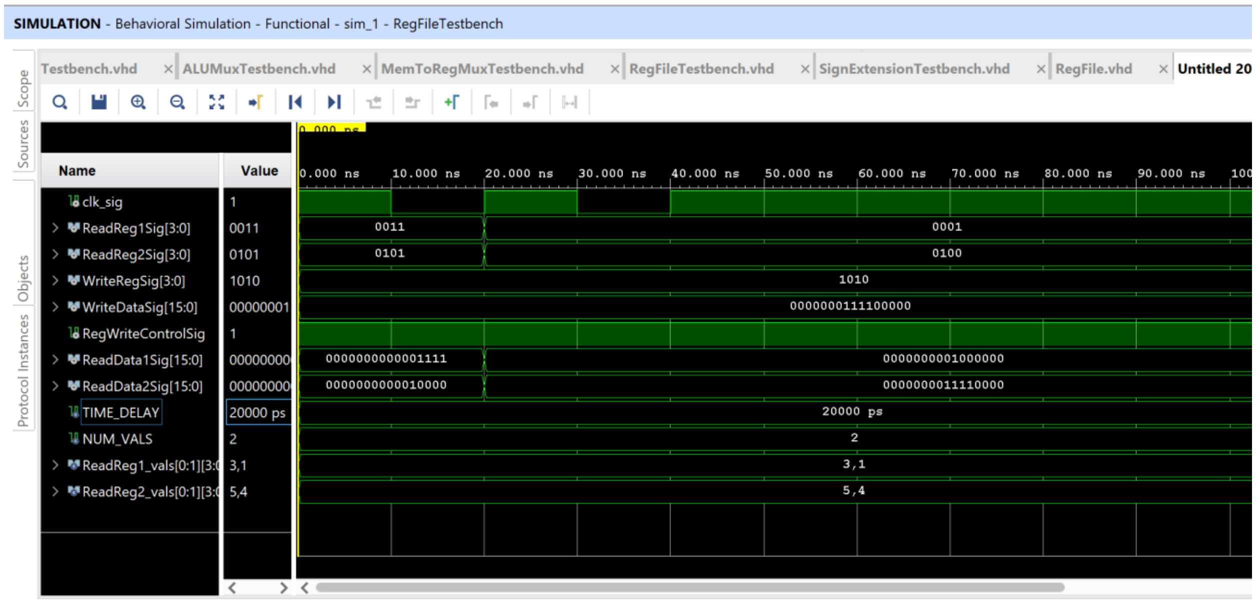
```
                wait for TIME_DELAY;
            end loop;
            wait;
    end process stimulus;


end behavior;
```



Name	Value
clk_sig	1
pc_in_sig[15:0]	0001
pc_out_sig[15:0]	0001
TIME_DELAY	20000 ps
NUM_VALS	2
pc_in_vals[0:1][15:0]	0000,0001

RegFileTestbench:

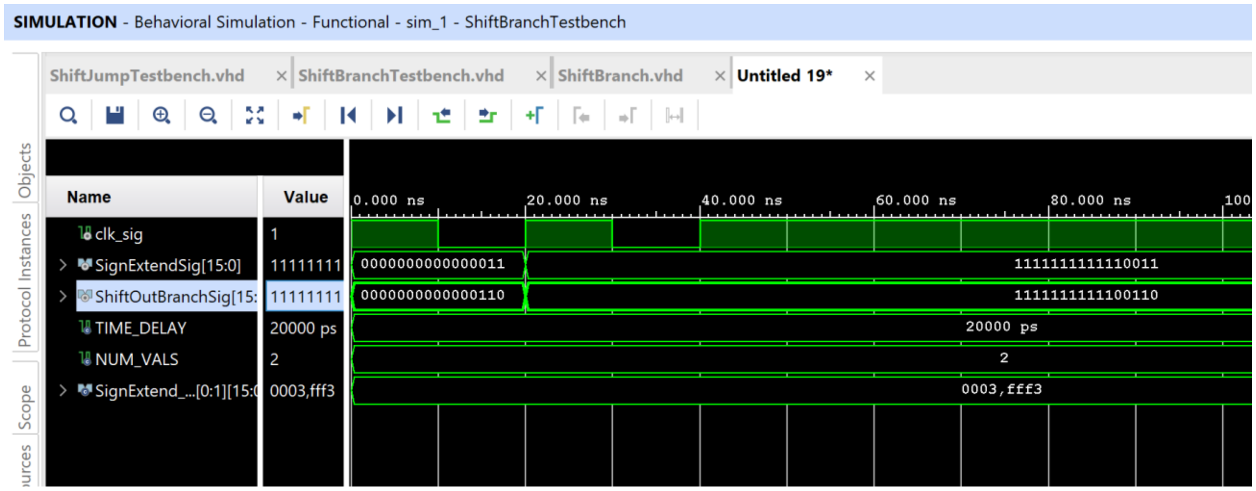
```vhdl
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity RegFileTestbench is
end RegFileTestbench;

architecture behavior of RegFileTestbench is
    constant TIME_DELAY : time := 20 ns;
    constant NUM_VALS : integer := 2;


    type ReadReg1_array is array(0 to (NUM_VALS - 1)) of std_logic_vector(3
downto 0);
    type ReadReg2_array is array(0 to (NUM_VALS - 1)) of std_logic_vector(3
downto 0);
    --type pc_out_array is array(0 to (NUM_VALS - 1)) of std_logic_vector(15
downto 0);

    constant ReadReg1_vals : ReadReg1_array := ("0011","0001");
    constant ReadReg2_vals : ReadReg2_array := ("0101","0100");



    signal clk_sig : std_logic := '0';
    signal ReadReg1Sig : std_logic_vector (3 downto 0) := "0000";
    signal ReadReg2Sig : std_logic_vector (3 downto 0) := "0000";
    signal WriteRegSig : std_logic_vector (3 downto 0) := "1010";
    signal WriteDataSig : std_logic_vector (15 downto 0) := "0000000111100000";
    signal RegWriteControlSig : std_logic := '1';
    signal ReadData1Sig : std_logic_vector (15 downto 0) := "0000000111100000";
    signal ReadData2Sig : std_logic_vector (15 downto 0) := "0000000111100000";


    begin
    DUT : entity work.RegFile(behavioral) port map (
                                        clk => clk_sig,
                                        ReadReg1 => ReadReg1Sig,
                                        ReadReg2 => ReadReg2Sig,
                                        WriteReg => WriteRegSig,
                                        WriteData => WriteDataSig,
                                        RegWriteCtrl => RegWriteControlSig,
                                        ReadData1 => ReadData1Sig,
                                        ReadData2 => ReadData2Sig);
    clock : process
     begin
      for i in 0 to 2 * (NUM_VALS) loop
       clk_sig <= NOT clk_sig;
       wait for TIME_DELAY/2;
      end loop;
      wait;
     end process clock;

    stimulus : process
     begin
      for i in 0 to (NUM_VALS - 1) loop
       ReadReg1Sig <= ReadReg1_vals(i);
       ReadReg2Sig <= ReadReg2_vals(i);
       wait for TIME_DELAY;
      end loop;
      wait;
 end process stimulus;


end behavior;
```



```
Name                      Value
clk_sig                   1
ReadReg1Sig[3:0]          0011        0011 | 0001
ReadReg2Sig[3:0]          0101        0101 | 0100
WriteRegSig[3:0]          1010        1010
WriteDataSig[15:0]        00000001    0000000111100000
RegWriteControlSig        1
ReadData1Sig[15:0]        00000000    0000000000001111 | 0000000001000000
ReadData2Sig[15:0]        00000000    0000000000010000 | 0000000011110000
TIME_DELAY                20000 ps    20000 ps
NUM_VALS                  2           2
ReadReg1_vals[0:1][3:0]   3,1         3,1
ReadReg2_vals[0:1][3:0]   5,4         5,4
```

ShiftBranchTestbench:
```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity ShiftBranchTestbench is
end ShiftBranchTestbench;

architecture behavior of ShiftBranchTestbench is
    constant TIME_DELAY : time := 20 ns;
    constant NUM_VALS : integer := 2;


    type SignExtend_array is array(0 to (NUM_VALS - 1)) of std_logic_vector(15
downto 0);
    --type pc_out_array is array(0 to (NUM_VALS - 1)) of std_logic_vector(15
downto 0);

    constant SignExtend_vals : SignExtend_array :=
("0000000000000011","1111111111110011");



    signal clk_sig : std_logic := '0';
    signal SignExtendSig : std_logic_vector (15 downto 0) := "0000000000000000";
    signal ShiftOutBranchSig : std_logic_vector (15 downto 0) :=
"0000000000000000";
    begin
    DUT : entity work.ShiftBranch(behavioral) port map (
                                             SignExtend => SignExtendSig,
                                             ShiftOutBranch => ShiftOut-
BranchSig);


        clock : process
          begin
            for i in 0 to 2 * (NUM_VALS) loop
              clk_sig <= NOT clk_sig;
              wait for TIME_DELAY/2;
            end loop;
            wait;
          end process clock;

         stimulus : process
          begin
```

```
            for i in 0 to (NUM_VALS - 1) loop
              SignExtendSig <= SignExtend_vals(i);
              wait for TIME_DELAY;
            end loop;
            wait;
  end process stimulus;



end behavior;
```

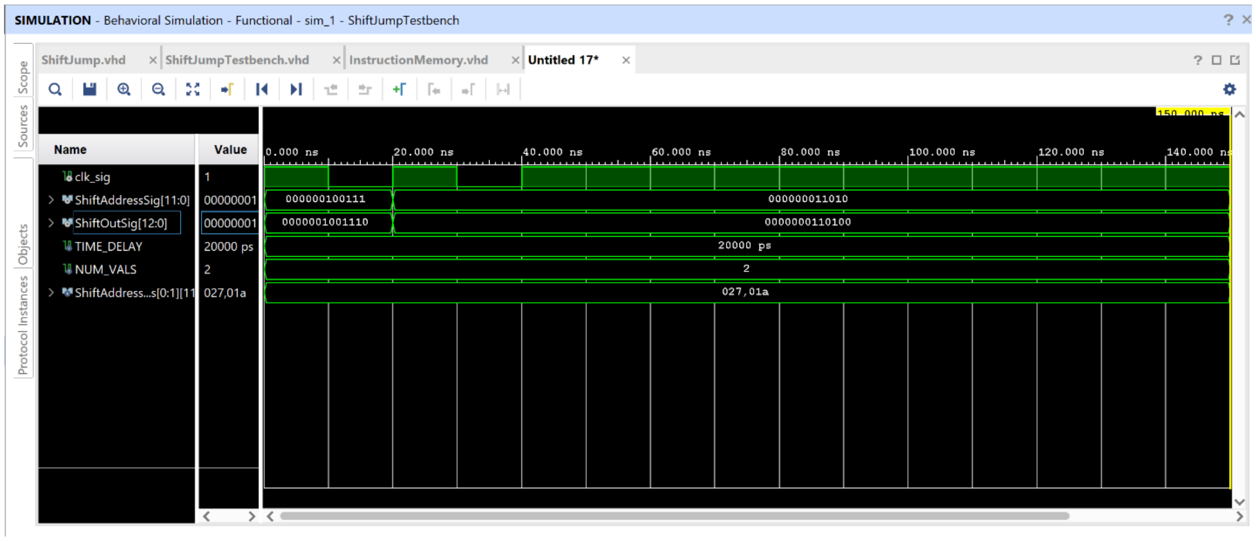


```
Name                      Value
clk_sig                   1
SignExtendSig[15:0]       11111111    0000000000000011 | 1111111111110011
ShiftOutBranchSig[15:     11111111    0000000000000110 | 1111111111100110
TIME_DELAY                20000 ps    20000 ps
NUM_VALS                  2           2
SignExtend_...[0:1][15:(  0003,fff3   0003,fff3
```

ShiftJumpTestbench:

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity ShiftJumpTestbench is
end ShiftJumpTestbench;

architecture behavior of ShiftJumpTestbench is
    constant TIME_DELAY : time := 20 ns;
    constant NUM_VALS : integer := 2;


    type ShiftAddress_array is array(0 to (NUM_VALS - 1)) of std_logic_vector(11
downto 0);
    --type pc_out_array is array(0 to (NUM_VALS - 1)) of std_logic_vector(15
downto 0);

    constant ShiftAddress_vals : ShiftAddress_array :=
("000000100111","000000011010");



    signal clk_sig : std_logic := '0';
    signal ShiftAddressSig : std_logic_vector (11 downto 0) := "000000000000";
    signal ShiftOutSig : std_logic_vector (12 downto 0) := "0000000000000";
    begin
    DUT : entity work.ShiftJump(behavioral) port map (
                                             ShiftAddress => ShiftAddressSig,
                                             ShiftOut => ShiftOutSig);


        clock : process
          begin
            for i in 0 to 2 * (NUM_VALS) loop
              clk_sig <= NOT clk_sig;
              wait for TIME_DELAY/2;
            end loop;
            wait;
          end process clock;

         stimulus : process
          begin
            for i in 0 to (NUM_VALS - 1) loop
              ShiftAddressSig <= ShiftAddress_vals(i);
```

```
                wait for TIME_DELAY;
            end loop;
            wait;
    end process stimulus;


end behavior;
```

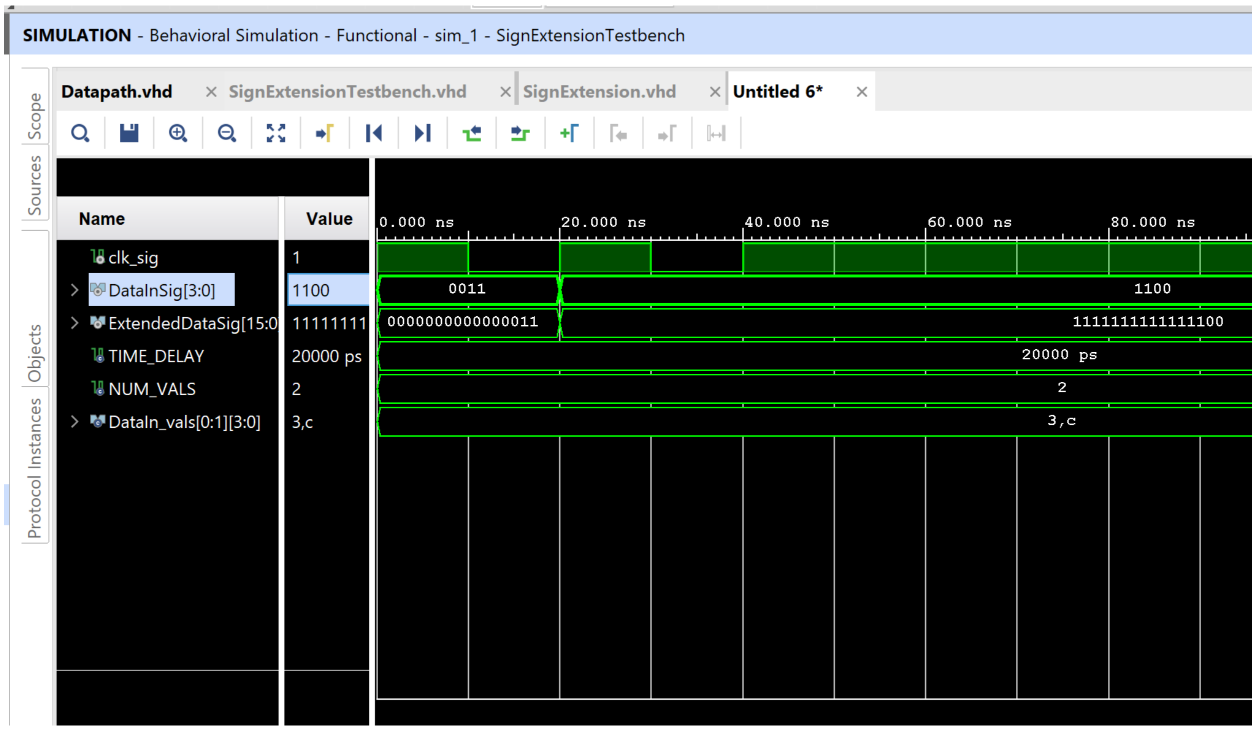


SIMULATION - Behavioral Simulation - Functional - sim_1 - ShiftJumpTestbench

Name	Value		
clk_sig	1		
ShiftAddressSig[11:0]	00000001	000000100111	000000011010
ShiftOutSig[12:0]	00000001	0000001001110	0000000110100
TIME_DELAY	20000 ps	20000 ps	
NUM_VALS	2	2	
ShiftAddress...s[0:1][11	027,01a	027,01a	

SignExtensionTestbench:
```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity SignExtensionTestbench is
end SignExtensionTestbench;

architecture behavior of SignExtensionTestbench is
    constant TIME_DELAY : time := 20 ns;
    constant NUM_VALS : integer := 2;


    type DataIn_array is array(0 to (NUM_VALS - 1)) of std_logic_vector(3 downto
0);
    --type pc_out_array is array(0 to (NUM_VALS - 1)) of std_logic_vector(15
downto 0);

    constant DataIn_vals : DataIn_array := ("0011","1100");



    signal clk_sig : std_logic := '0';
    signal DataInSig : std_logic_vector (3 downto 0) := "0000";
    signal ExtendedDataSig : std_logic_vector (15 downto 0) :=
"0000000000000000";
    begin
    DUT : entity work.SignExtension(behavioral) port map (
                                             DataIn => DataInSig,
                                             ExtendedData => ExtendedDataSig);


        clock : process
          begin
            for i in 0 to 2 * (NUM_VALS) loop
              clk_sig <= NOT clk_sig;
              wait for TIME_DELAY/2;
            end loop;
            wait;
          end process clock;

         stimulus : process
          begin
            for i in 0 to (NUM_VALS - 1) loop
              DataInSig <= DataIn_vals(i);
```

```
                wait for TIME_DELAY;
            end loop;
            wait;
    end process stimulus;


end behavior;
```



SIMULATION - Behavioral Simulation - Functional - sim_1 - SignExtensionTestbench

Name	Value
clk_sig	1
DataInSig[3:0]	1100
ExtendedDataSig[15:0	11111111
TIME_DELAY	20000 ps
NUM_VALS	2
DataIn_vals[0:1][3:0]	3,c

```
DataInSig:        0011              1100
ExtendedDataSig:  0000000000000011  1111111111111100
TIME_DELAY:       20000 ps
NUM_VALS:         2
DataIn_vals:      3,c
```