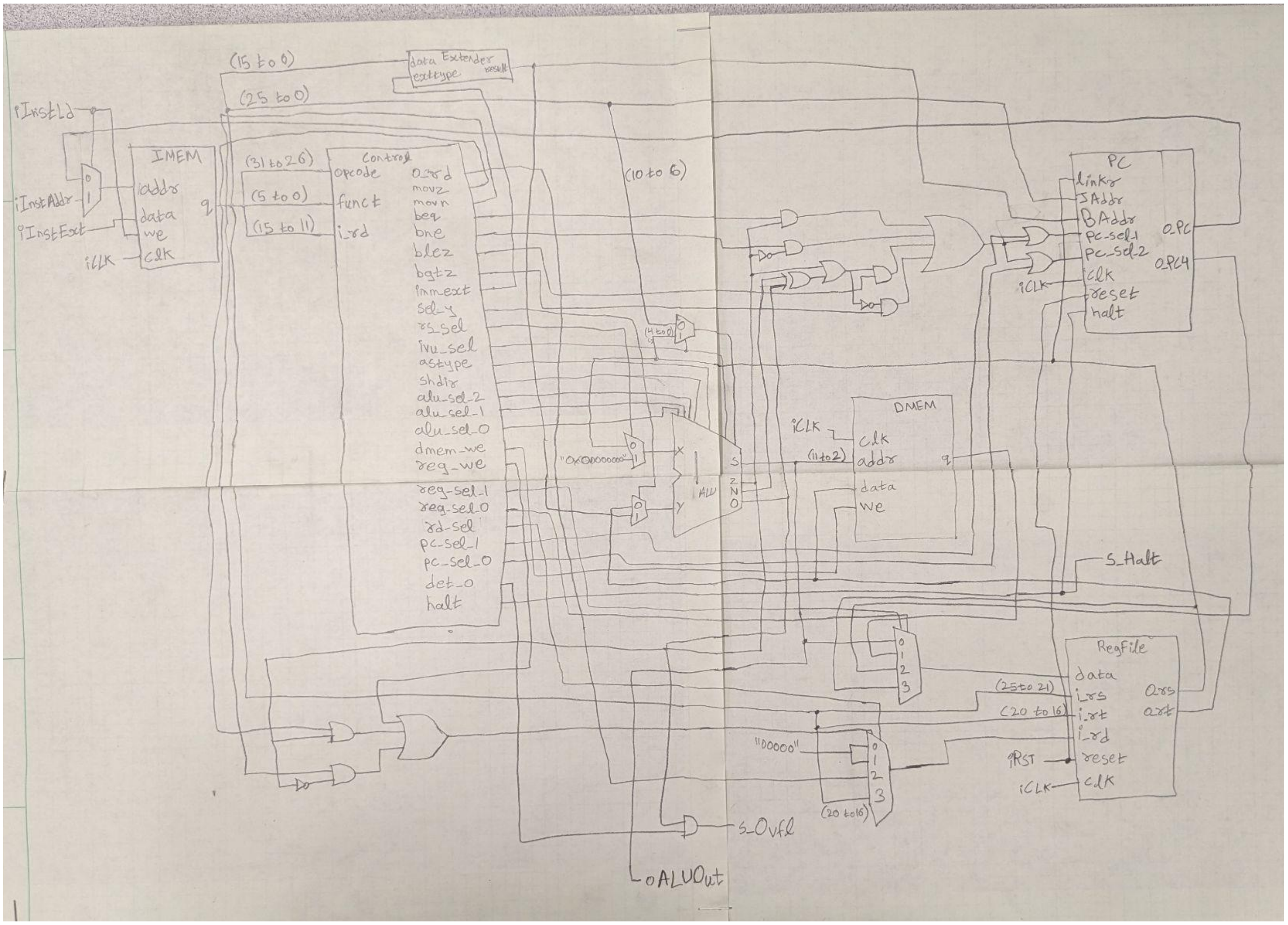
CprE 381: Computer Organization and Assembly-Level Programming

Project Part 1 Report

Team Members: Varun Jain

Project Teams Group #: Proj1_4_2

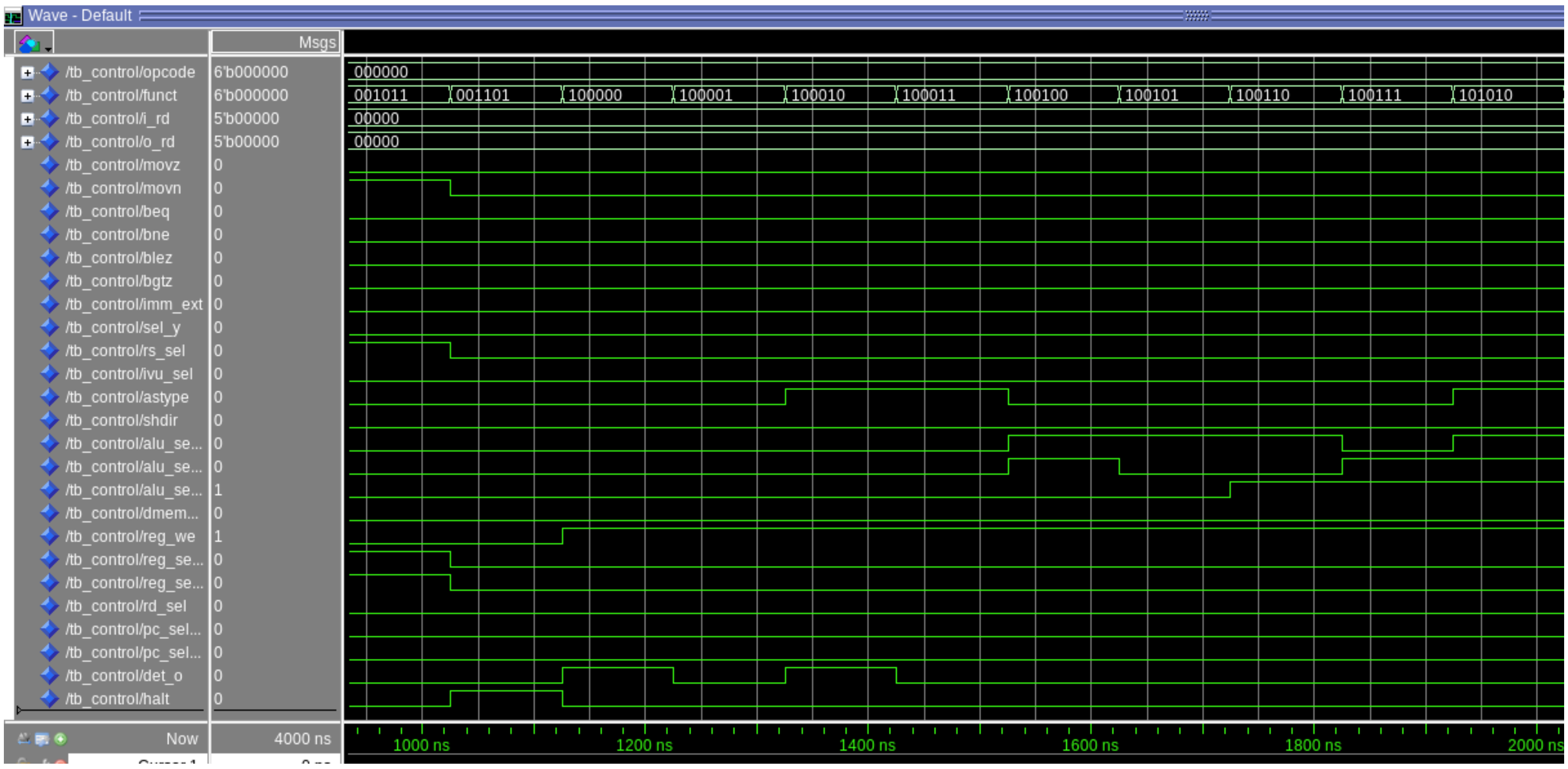
[Part 1 (d)] Include your final MIPS processor schematic in your lab report.



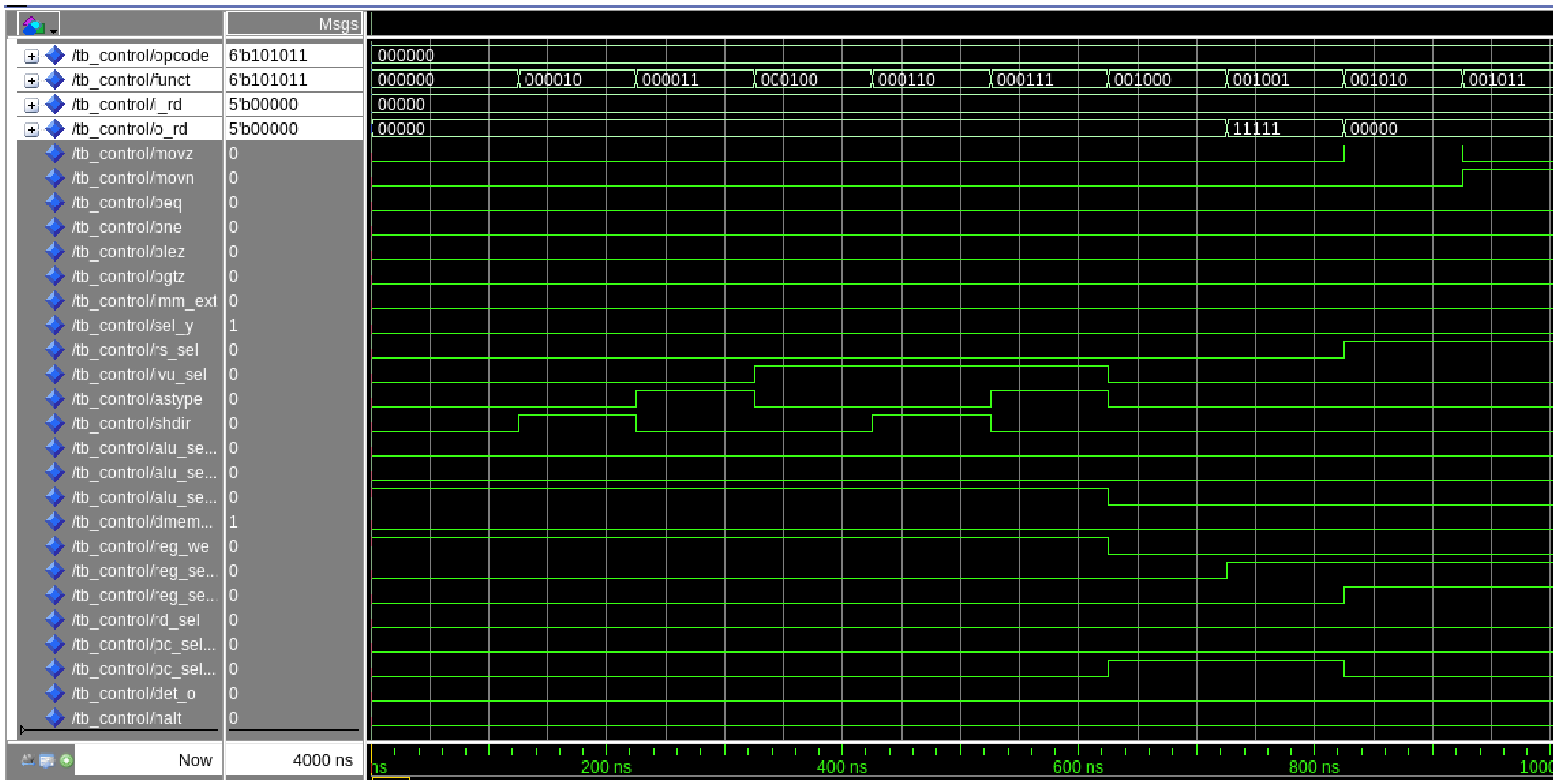
[Part 2 (a.i)] Create a spreadsheet detailing the list of *M* instructions to be supported in your project alongside their binary opcodes and funct fields, if applicable. Create a separate column for each binary bit. Inside this spreadsheet, create a new column for the *N* control signals needed by your datapath implementation. The end result should be an *N***M* table where each row corresponds to the output of the control logic module for a given instruction.

Control signal spreadsheet attached in the zip file.

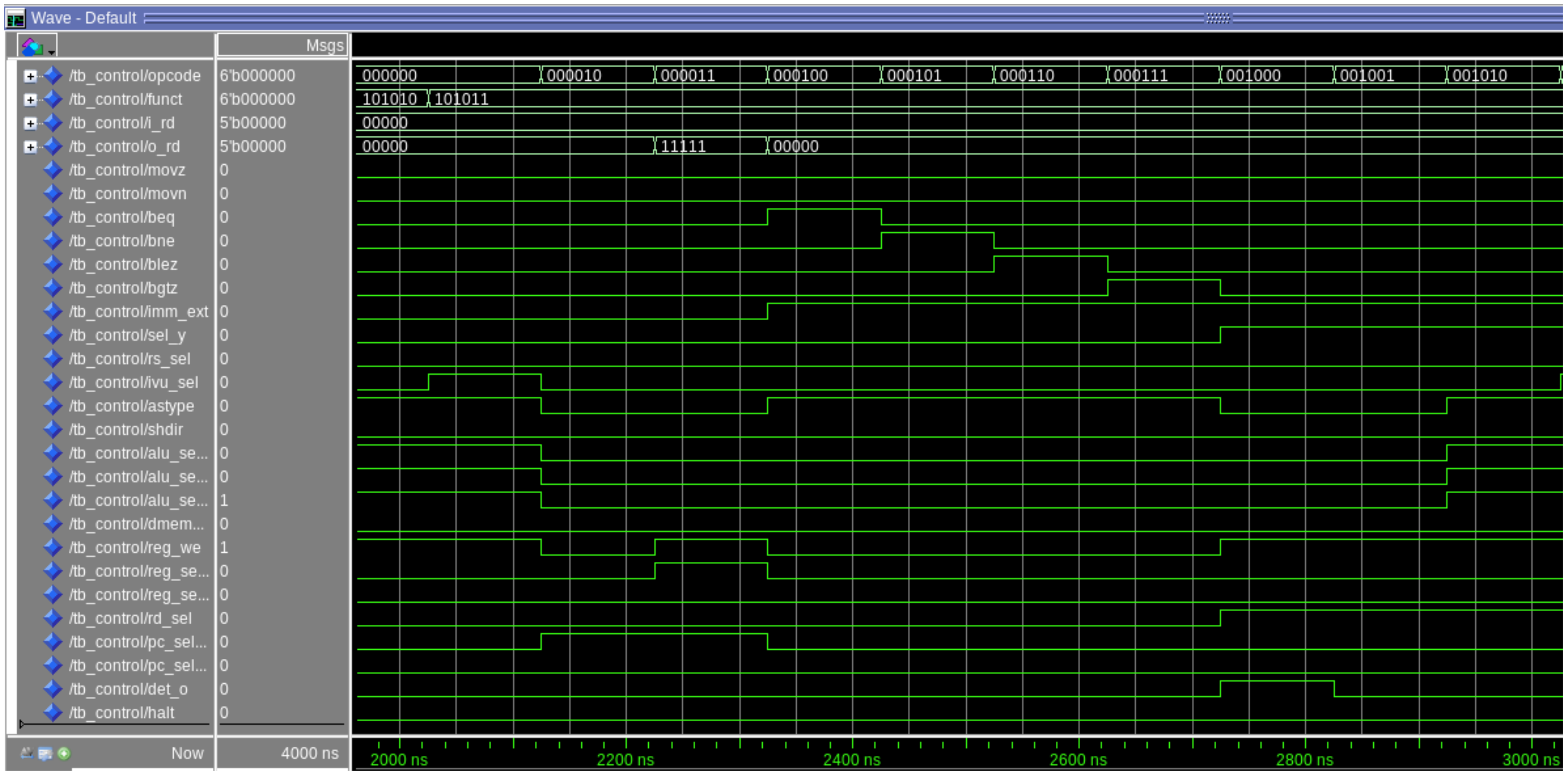
[Part 2 (a.ii)] Implement the control logic module using whatever method and coding style you prefer. Create a testbench to test this module individually, and show that your output matches the expected control signals from problem 1(a).



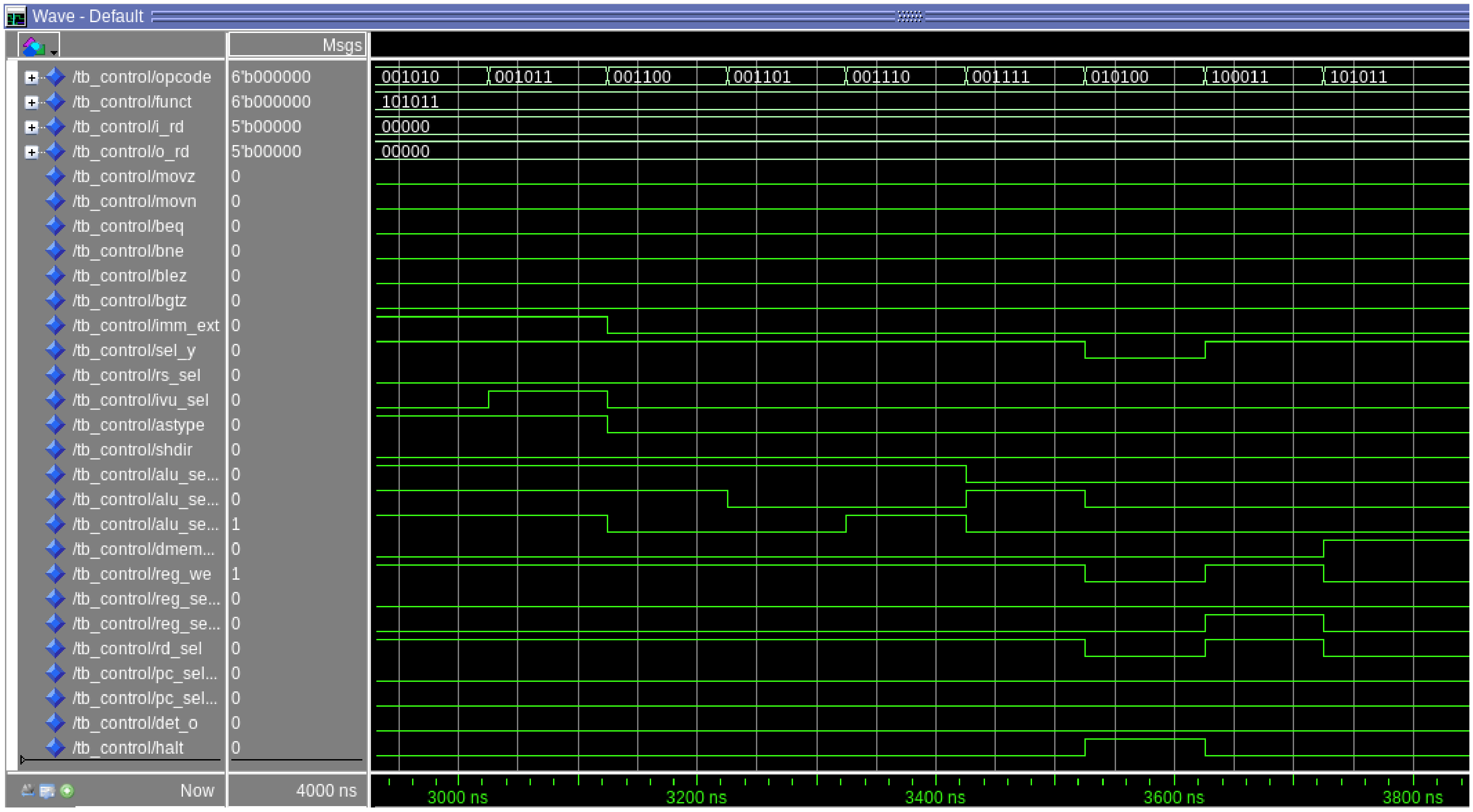
```
/tb_control/opcode    6'b000000
/tb_control/funct     6'b000000
/tb_control/i_rd      5'b00000
/tb_control/o_rd      5'b00000
/tb_control/movz      0
/tb_control/movn      0
/tb_control/beq       0
/tb_control/bne       0
/tb_control/blez      0
/tb_control/bgtz      0
/tb_control/imm_ext   0
/tb_control/sel_y     0
/tb_control/rs_sel    0
/tb_control/ivu_sel   0
/tb_control/astype    0
/tb_control/shdir     0
/tb_control/alu_se... 0
/tb_control/alu_se... 0
/tb_control/alu_se... 1
/tb_control/dmem...   0
/tb_control/reg_we    1
/tb_control/reg_se... 0
/tb_control/reg_se... 0
/tb_control/rd_sel    0
/tb_control/pc_sel... 0
/tb_control/pc_sel... 0
/tb_control/det_o     0
/tb_control/halt      0
```



```
/tb_control/opcode    6'b101011
/tb_control/funct     6'b101011
/tb_control/i_rd      5'b00000
/tb_control/o_rd      5'b00000
/tb_control/movz      0
/tb_control/movn      0
/tb_control/beq       0
/tb_control/bne       0
/tb_control/blez      0
/tb_control/bgtz      0
/tb_control/imm_ext   0
/tb_control/sel_y     1
/tb_control/rs_sel    0
/tb_control/ivu_sel   0
/tb_control/astype    0
/tb_control/shdir     0
/tb_control/alu_se... 0
/tb_control/alu_se... 0
/tb_control/alu_se... 0
/tb_control/dmem...   1
/tb_control/reg_we    0
/tb_control/reg_se... 0
/tb_control/reg_se... 0
/tb_control/rd_sel    0
/tb_control/pc_sel... 0
/tb_control/pc_sel... 0
/tb_control/det_o     0
/tb_control/halt      0
```



```
/tb_control/opcode     6'b000000
/tb_control/funct      6'b000000
/tb_control/i_rd       5'b00000
/tb_control/o_rd       5'b00000
/tb_control/movz       0
/tb_control/movn       0
/tb_control/beq        0
/tb_control/bne        0
/tb_control/blez       0
/tb_control/bgtz       0
/tb_control/imm_ext    0
/tb_control/sel_y      0
/tb_control/rs_sel     0
/tb_control/ivu_sel    0
/tb_control/astype     0
/tb_control/shdir      0
/tb_control/alu_se...  0
/tb_control/alu_se...  0
/tb_control/alu_se...  1
/tb_control/dmem...    0
/tb_control/reg_we     1
/tb_control/reg_se...  0
/tb_control/reg_se...  0
/tb_control/rd_sel     0
/tb_control/pc_sel...  0
/tb_control/pc_sel...  0
/tb_control/det_o      0
/tb_control/halt       0
```

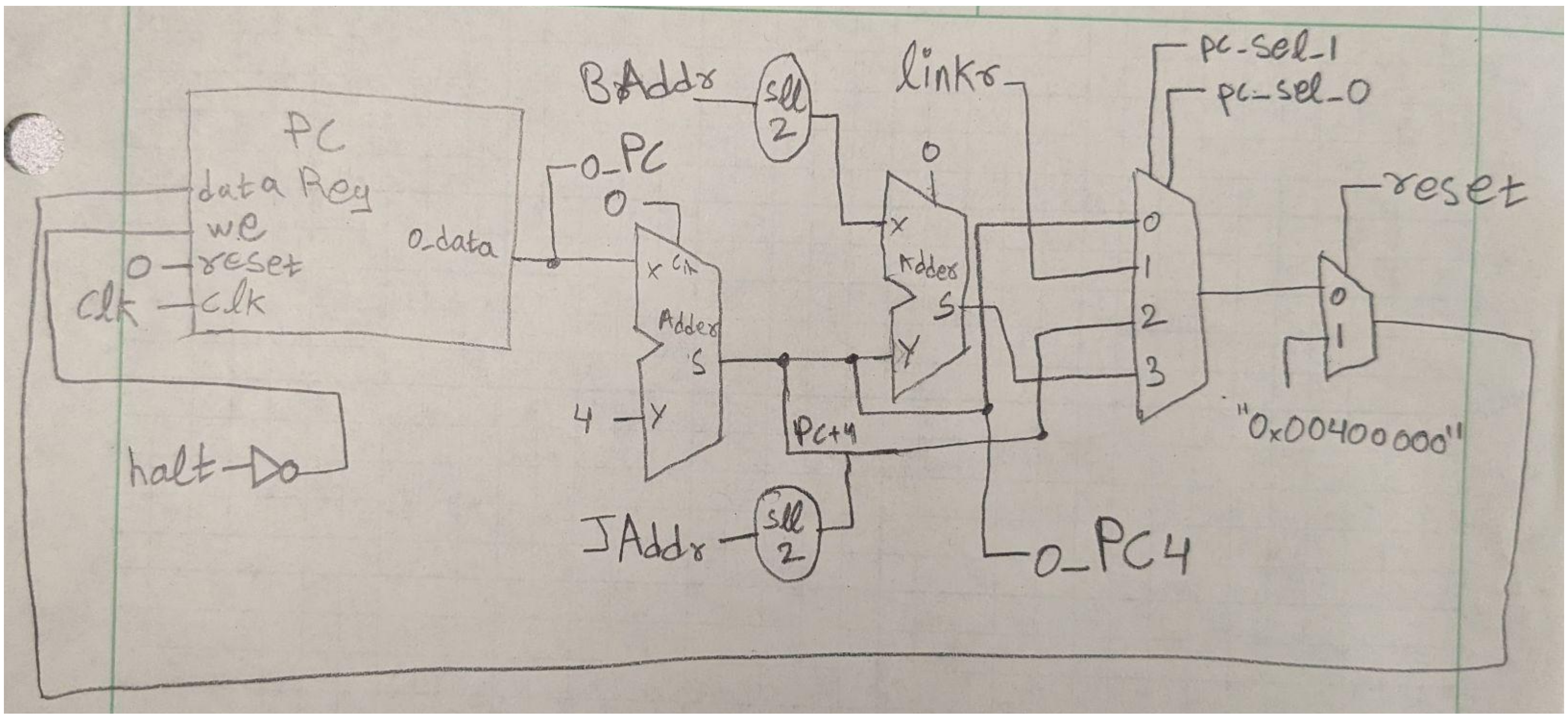


```
/tb_control/opcode     6'b000000
/tb_control/funct      6'b000000
/tb_control/i_rd       5'b00000
/tb_control/o_rd       5'b00000
/tb_control/movz       0
/tb_control/movn       0
/tb_control/beq        0
/tb_control/bne        0
/tb_control/blez       0
/tb_control/bgtz       0
/tb_control/imm_ext    0
/tb_control/sel_y      0
/tb_control/rs_sel     0
/tb_control/ivu_sel    0
/tb_control/astype     0
/tb_control/shdir      0
/tb_control/alu_se...  0
/tb_control/alu_se...  0
/tb_control/alu_se...  1
/tb_control/dmem...    0
/tb_control/reg_we     1
/tb_control/reg_se...  0
/tb_control/reg_se...  0
/tb_control/rd_sel     0
/tb_control/pc_sel...  0
/tb_control/pc_sel...  0
/tb_control/det_o      0
/tb_control/halt       0
```

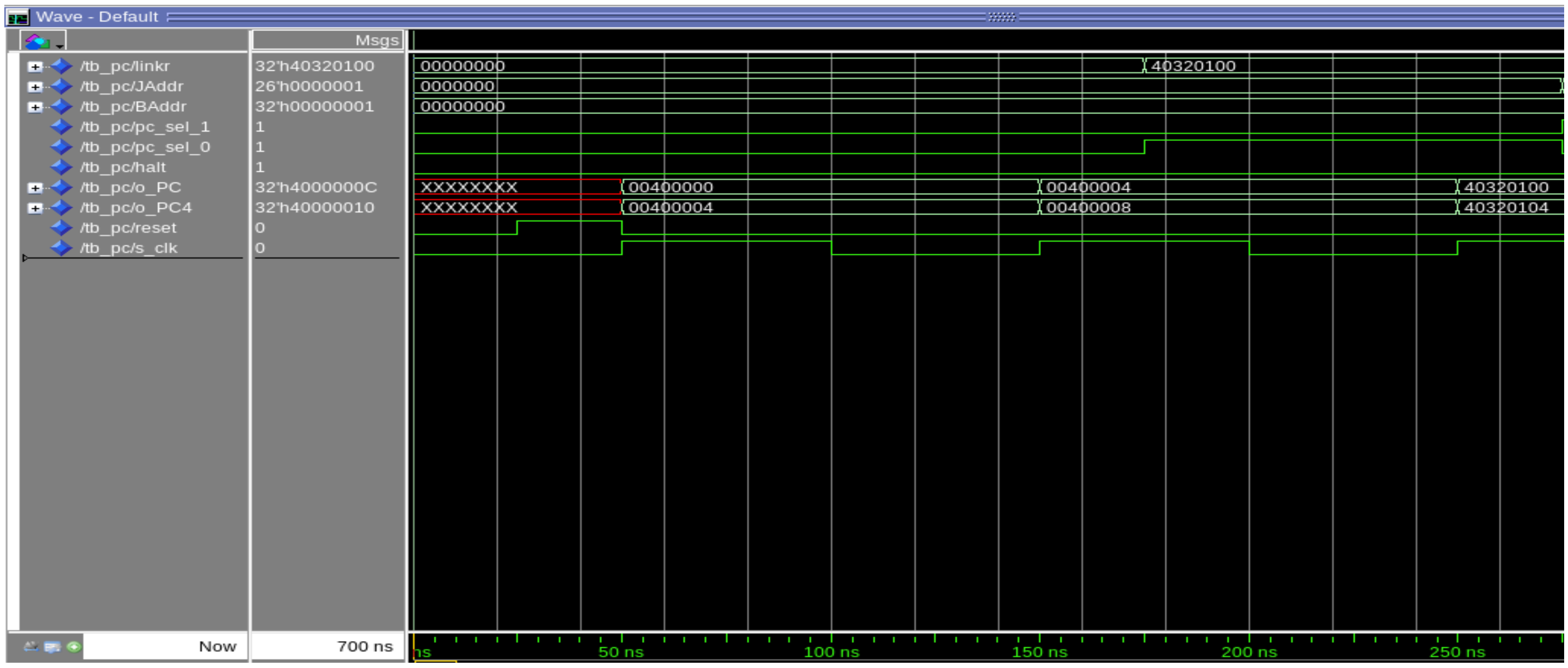
[Part 2 (b.i)] What are the control flow possibilities that your instruction fetch logic must support? Describe these possibilities as a function of the different control flow-related instructions you are required to implement.

Next instruction: PC = PC + 4
`j`: Jump unconditionally to (PC + 4)[31:28] & JumpAddress & "00"
`jal`: Similar to j, but also saves PC + 4 to register 31 ($ra)
`jr`: Similar to j, but jumps to the value in the stated register
`beq`: Branch to PC + 4 + {14'b(immediate[15]) & immediate & "00"} if the stated registers are equal
`bne`: Similar to beq but the stated registers are not equal

[Part 2 (b.ii)] Draw a schematic for the instruction fetch logic and any other datapath modifications needed for control flow instructions. What additional control signals are needed?

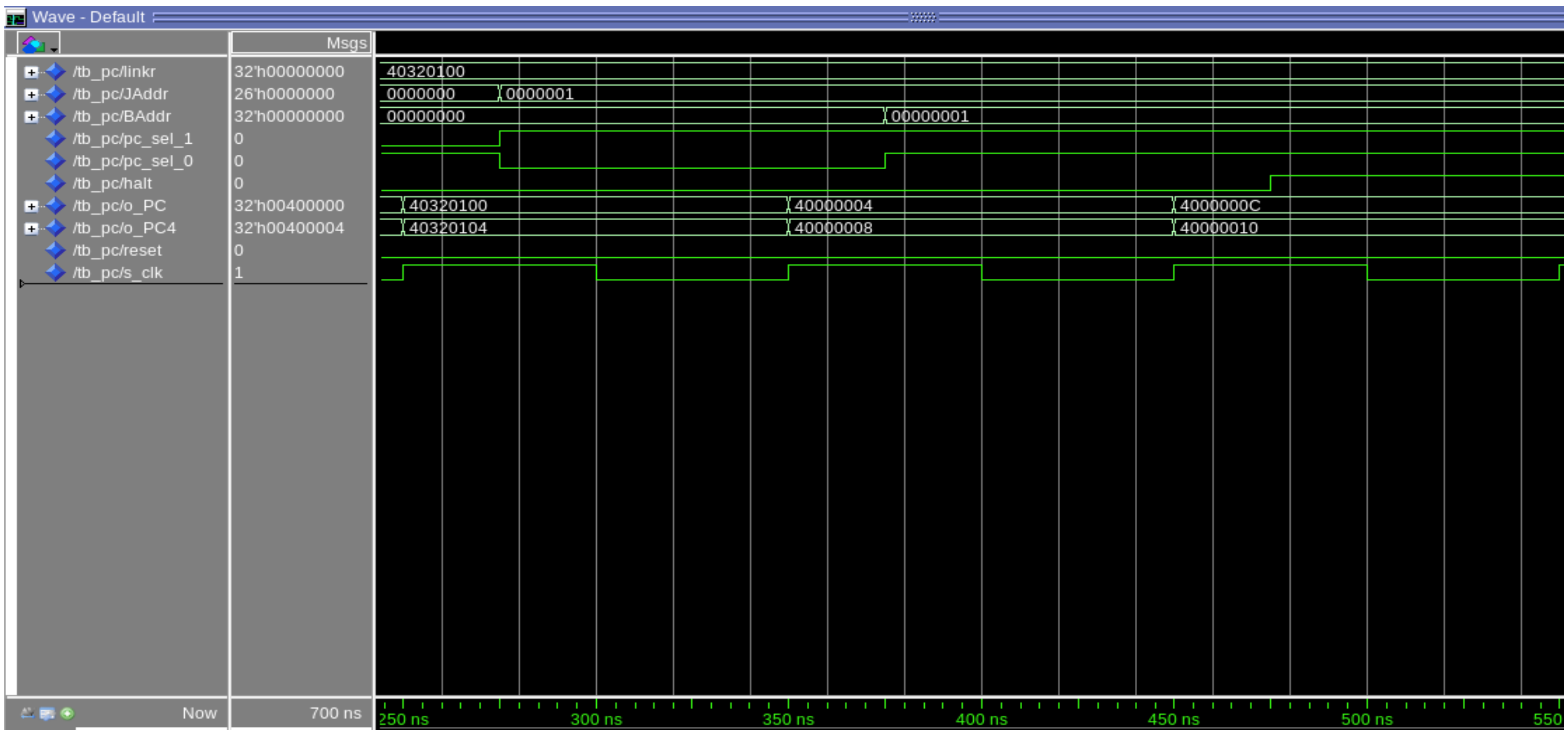


[Part 2 (b.iii)] Implement your new instruction fetch logic using VHDL. Use Modelsim to test your design thoroughly to make sure it is working as expected. Describe how the execution of the control flow possibilities corresponds to the Modelsim waveforms in your writeup.



Wave - Default

	Msgs
/tb_pc/linkr	32'h40320100
/tb_pc/JAddr	26'h0000001
/tb_pc/BAddr	32'h00000001
/tb_pc/pc_sel_1	1
/tb_pc/pc_sel_0	1
/tb_pc/halt	1
/tb_pc/o_PC	32'h4000000C
/tb_pc/o_PC4	32'h40000010
/tb_pc/reset	0
/tb_pc/s_clk	0
Now	700 ns



Wave - Default

	Msgs
/tb_pc/linkr	32'h00000000
/tb_pc/JAddr	26'h0000000
/tb_pc/BAddr	32'h00000000
/tb_pc/pc_sel_1	0
/tb_pc/pc_sel_0	0
/tb_pc/halt	0
/tb_pc/o_PC	32'h00400000
/tb_pc/o_PC4	32'h00400004
/tb_pc/reset	0
/tb_pc/s_clk	1
Now	700 ns

After the reset, we don't set any signals, which increments the PC by 4 after one clock cycle. Then we use jr, setting the linkr (Link register) and setting pc_sel to "01". Then we use the j/jal, setting the Jump Address and setting pc_sel to "10". Finally, we use a conditional branch and skip 1 instruction by setting BAddr and setting pc_sel to "11".

[Part 2 (c.i.1)] Describe the difference between logical (`srl`) and arithmetic (`sra`) shifts. Why does MIPS not have a `sla` instruction?

Logical shift does not account for the most significant bit. It therefore treats the numbers as unsigned. On the other hand Arithmetic shifts account for the most significant bit.

MIPS does not have a sla instruction because it is equivalent to the sll instruction. This adds redundancy to the system.

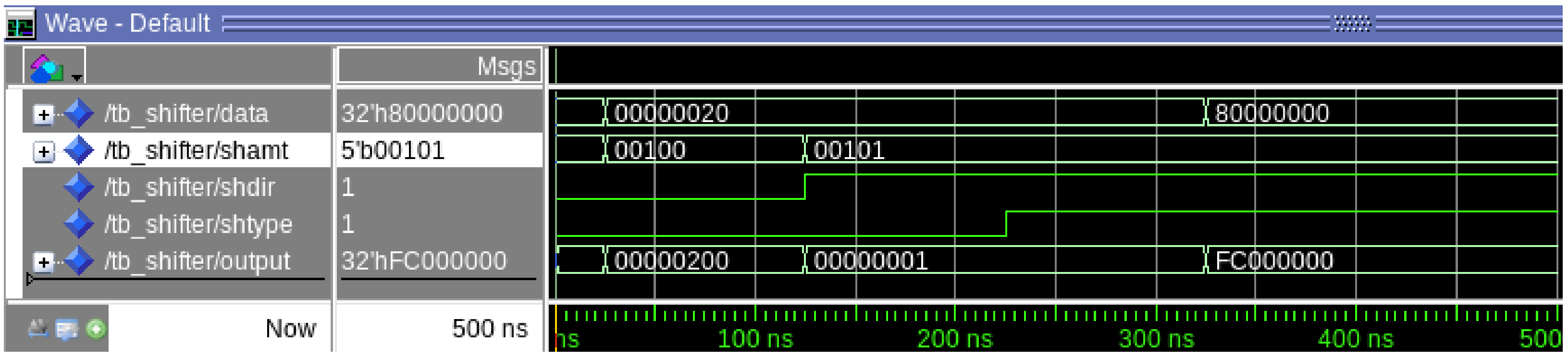
[Part 2 (c.i.2)] In your writeup, briefly describe how your VHDL code implements both the arithmetic and logical shifting operations.

A 64-to-1 multiplexor is used with the select line being the type (logical or arithmetic) followed by the shift amount.

[Part 2 (c.i.3)] In your writeup, explain how the right barrel shifter above can be enhanced to also support left shifting operations.

Another select line specifying the direction of the shift can be added and the multiplexor changed from a 64-to-1 to a 128-to-1.

[Part 2 (c.i.4)] Describe how the execution of the different shifting operations corresponds to the Modelsim waveforms in your writeup.

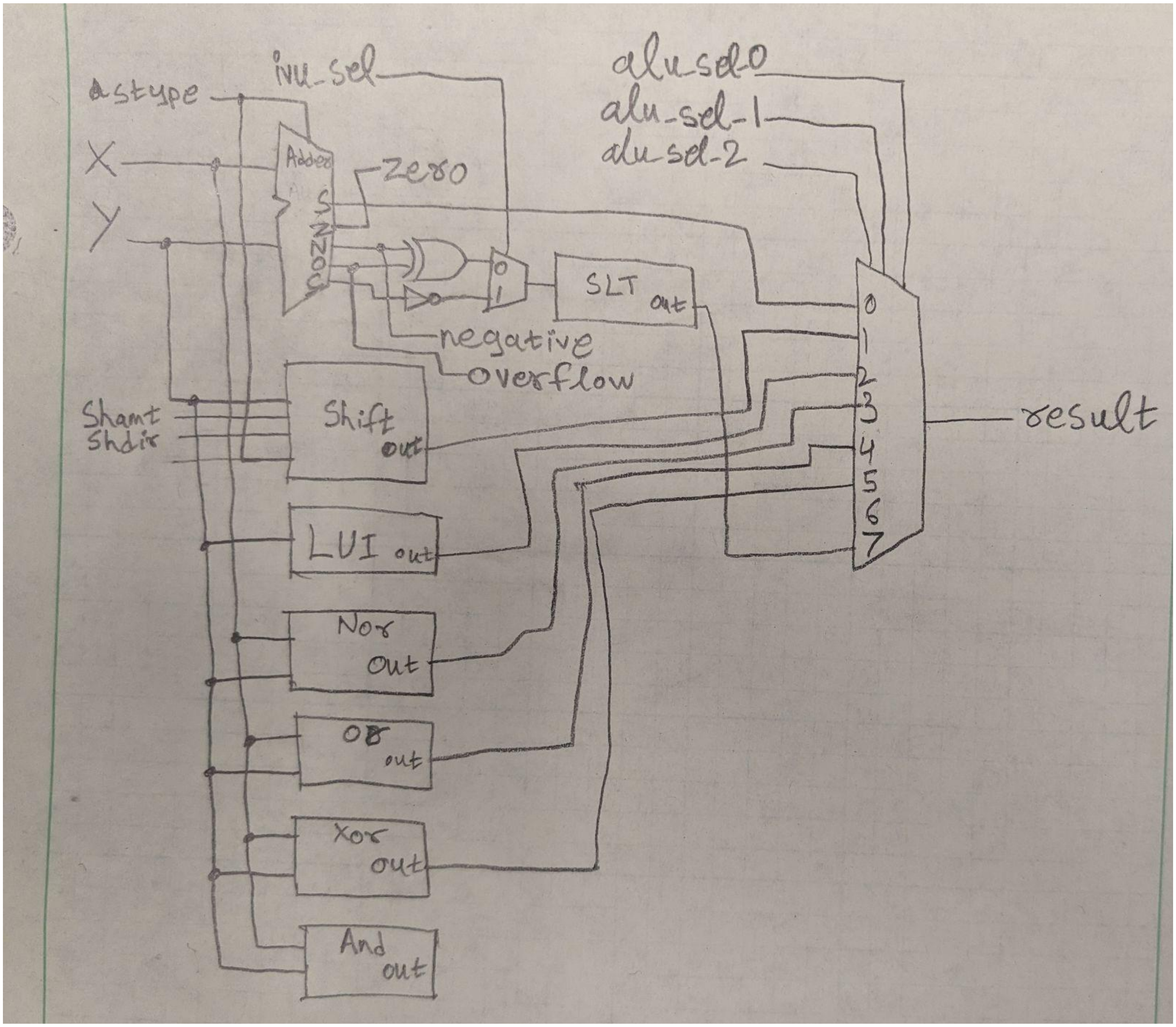


The first operation is "sll 0x20, 4". It then performs "srl 0x20, 5". Then it performs "sra 0x20, 5", and finally "sra 0x80000000, 5".

[Part 2 (c.ii.1)] In your writeup, briefly describe your design approach, including any resources you used to choose or implement the design. Include at least one design decision you had to make.

For the 32-bit adder, I designed a Carry Lookahead Adder (CLA). This was implemented using 4 8-bit CLAs. To calculate the overflow, the last carry bit generated by each 8-bit CLA was output to the 32-bit CLA.

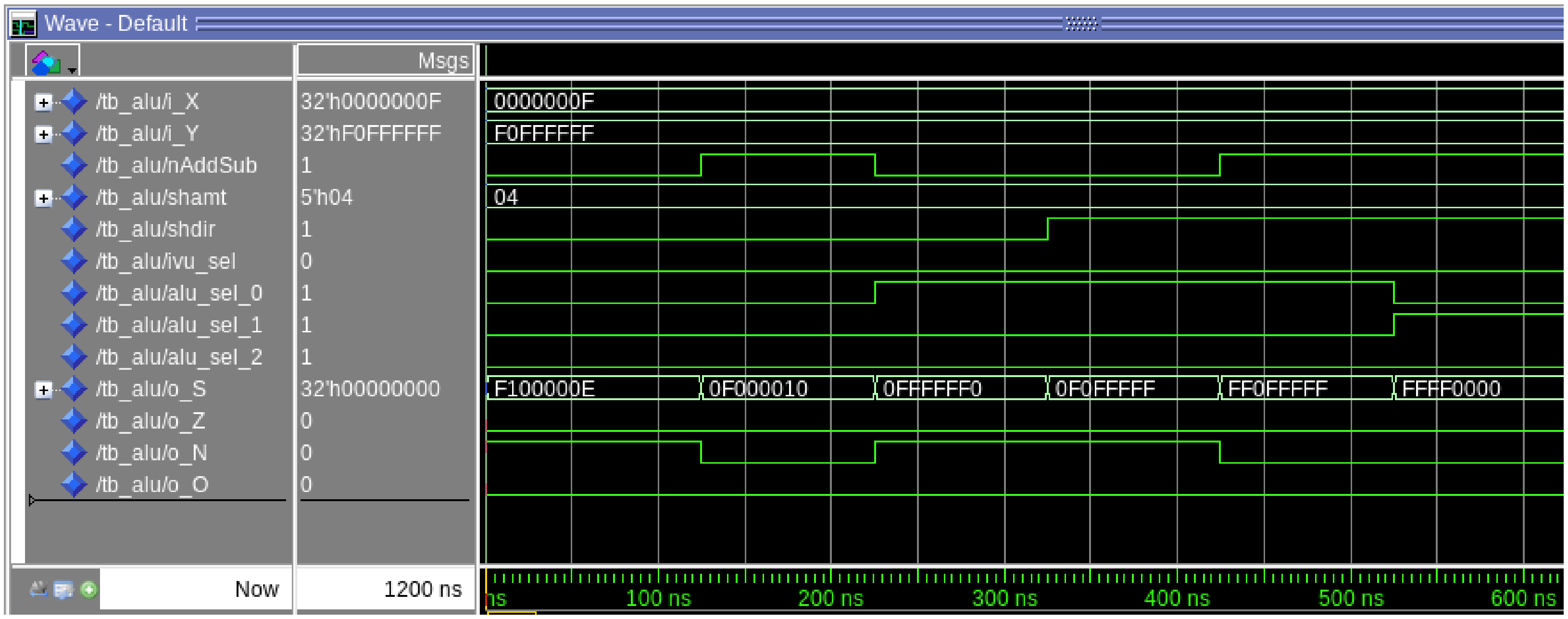
[Part 2 (c.iii)] Draw a simplified, high-level schematic for the 32-bit ALU. Consider the following questions: how is Overflow calculated? How is Zero calculated? How is `slt` implemented?

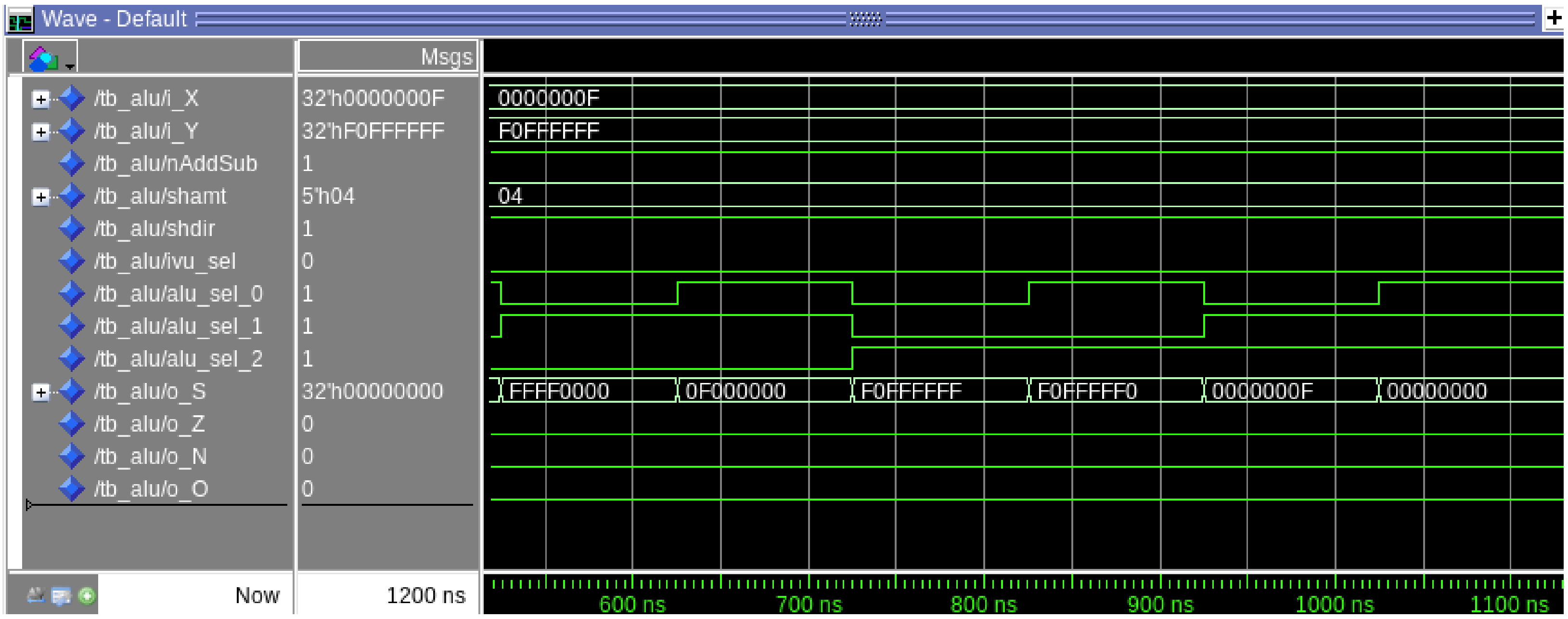


[Part 2 (c.v)] Describe how the execution of the different operations corresponds to the Modelsim waveforms in your writeup.

Add: Adds two numbers and generates the Zero, Negative and Overflow flags as well. The output sum goes to the ALU mux at position 0.
Sub: Similar to Add but subtracts Y from X. Also, the astype (AddSub) control is set to 1.
sll: Shifts Y to the left logically by the amount in shamt. Output goes to position 1.
srl: Similar to sll, but shifts right logically. Control signal shdir is 1.
sra: Similar to sll, but shifts right arithmetically. Control signal astype is set to 1.
lui: Takes the Y input and shifts it left by 16. Output goes to position 2.
nor: Bitwise NOR between X and Y. Output goes to position 3.
or: Bitwise OR between X and Y. Output goes to position 4.
xor: Bitwise XOR between X and Y. Output goes to position 5.
and: Bitwise AND between X and Y. Output goes to position 6.
slt: Outputs 1 if $X < Y$ else 0. Output goes to position 7. Control signal astype is 1.

[Part 2 (c.viii)] justify why your test plan is comprehensive. Include waveforms that demonstrate your test programs functioning.



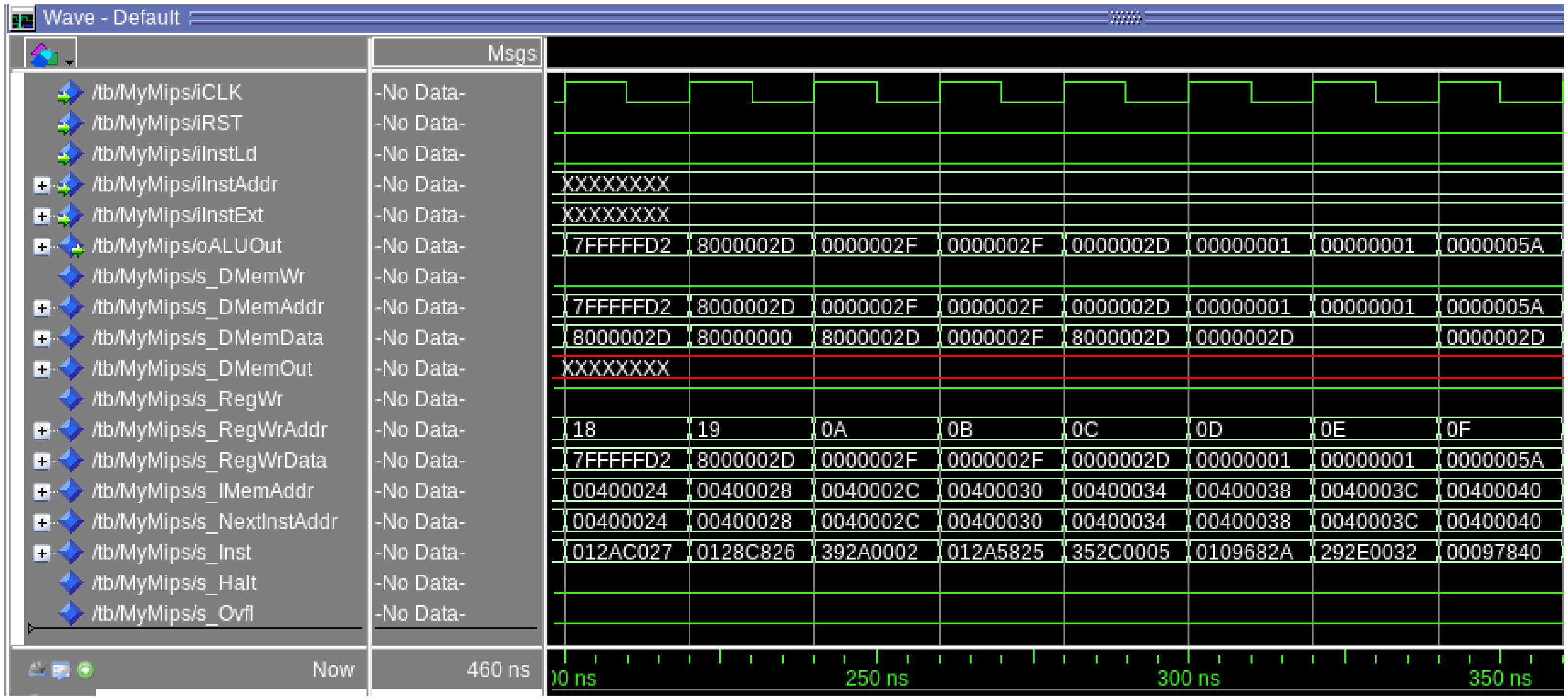


First, we perform ADD, then SUB, SLL, SRL, and SRA, then use LUI, then NOR, followed by OR, XOR, AND, and finally SLT. The tests cover all aspects of the ALU functions. Since most of the other decisions are made outside of the ALU, it does not care about the function being R-, I-, or J-type.

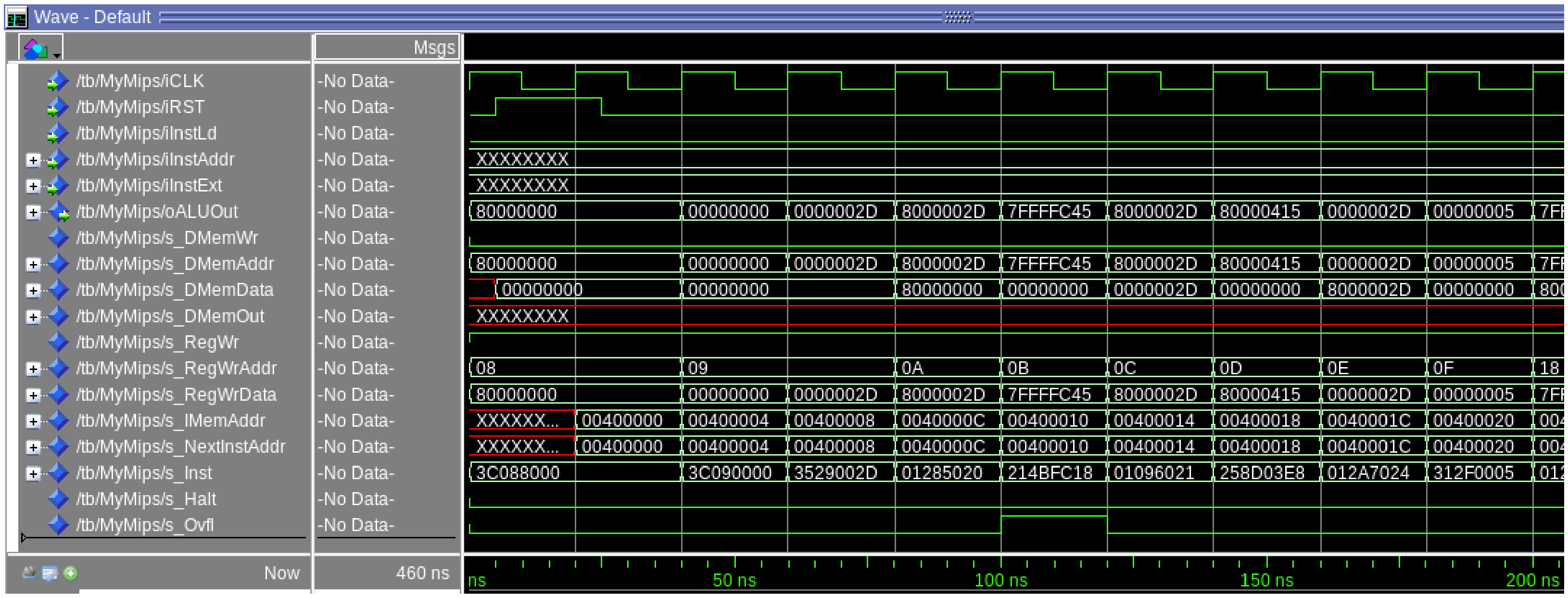
[Part 3] In your writeup, show the Modelsim output for each of the following tests, and provide a discussion of result correctness. It may be helpful to also annotate the waveforms directly.

All test outputs are attached in the zip file.

[Part 3 (a)] Create a test application that makes use of every required arithmetic/logical instruction at least once. The application need not perform any particularly useful task, but it should demonstrate the full functionality of the processor (e.g., sequences of many instructions executed sequentially, 1 per cycle while data written into registers can be effectively retrieved and used by later instructions). Name this file Proj1_base_test.s.

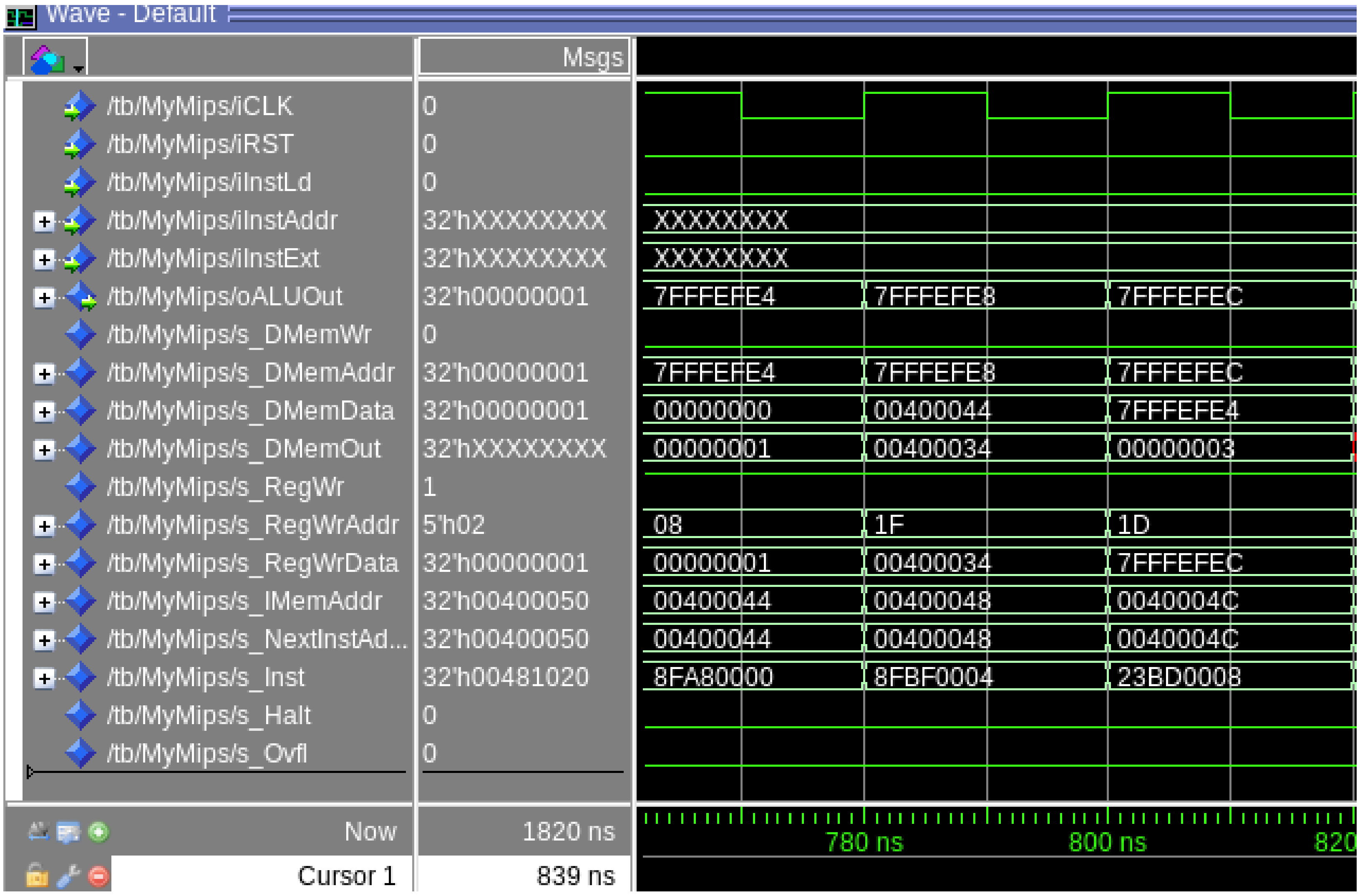


```
Wave - Default
                          Msgs
/tb/MyMips/iCLK           -No Data-
/tb/MyMips/iRST           -No Data-
/tb/MyMips/iInstLd        -No Data-
/tb/MyMips/iInstAddr      -No Data-  XXXXXXXX
/tb/MyMips/iInstExt       -No Data-  XXXXXXXX
/tb/MyMips/oALUOut        -No Data-  7FFFFFD2  8000002D  0000002F  0000002F  0000002D  00000001  00000001  0000005A
/tb/MyMips/s_DMemWr       -No Data-
/tb/MyMips/s_DMemAddr     -No Data-  7FFFFFD2  8000002D  0000002F  0000002F  0000002D  00000001  00000001  0000005A
/tb/MyMips/s_DMemData     -No Data-  8000002D  80000000  8000002D  0000002F  8000002D  0000002D            0000002D
/tb/MyMips/s_DMemOut      -No Data-  XXXXXXXX
/tb/MyMips/s_RegWr        -No Data-
/tb/MyMips/s_RegWrAddr    -No Data-  18        19        0A        0B        0C        0D        0E        0F
/tb/MyMips/s_RegWrData    -No Data-  7FFFFFD2  8000002D  0000002F  0000002F  0000002D  00000001  00000001  0000005A
/tb/MyMips/s_IMemAddr     -No Data-  00400024  00400028  0040002C  00400030  00400034  00400038  0040003C  00400040
/tb/MyMips/s_NextInstAddr -No Data-  00400024  00400028  0040002C  00400030  00400034  00400038  0040003C  00400040
/tb/MyMips/s_Inst         -No Data-  012AC027  0128C826  392A0002  012A5825  352C0005  0109682A  292E0032  00097840
/tb/MyMips/s_Halt         -No Data-
/tb/MyMips/s_Ovfl         -No Data-
Now                        460 ns    )0 ns     250 ns     300 ns     350 ns
```



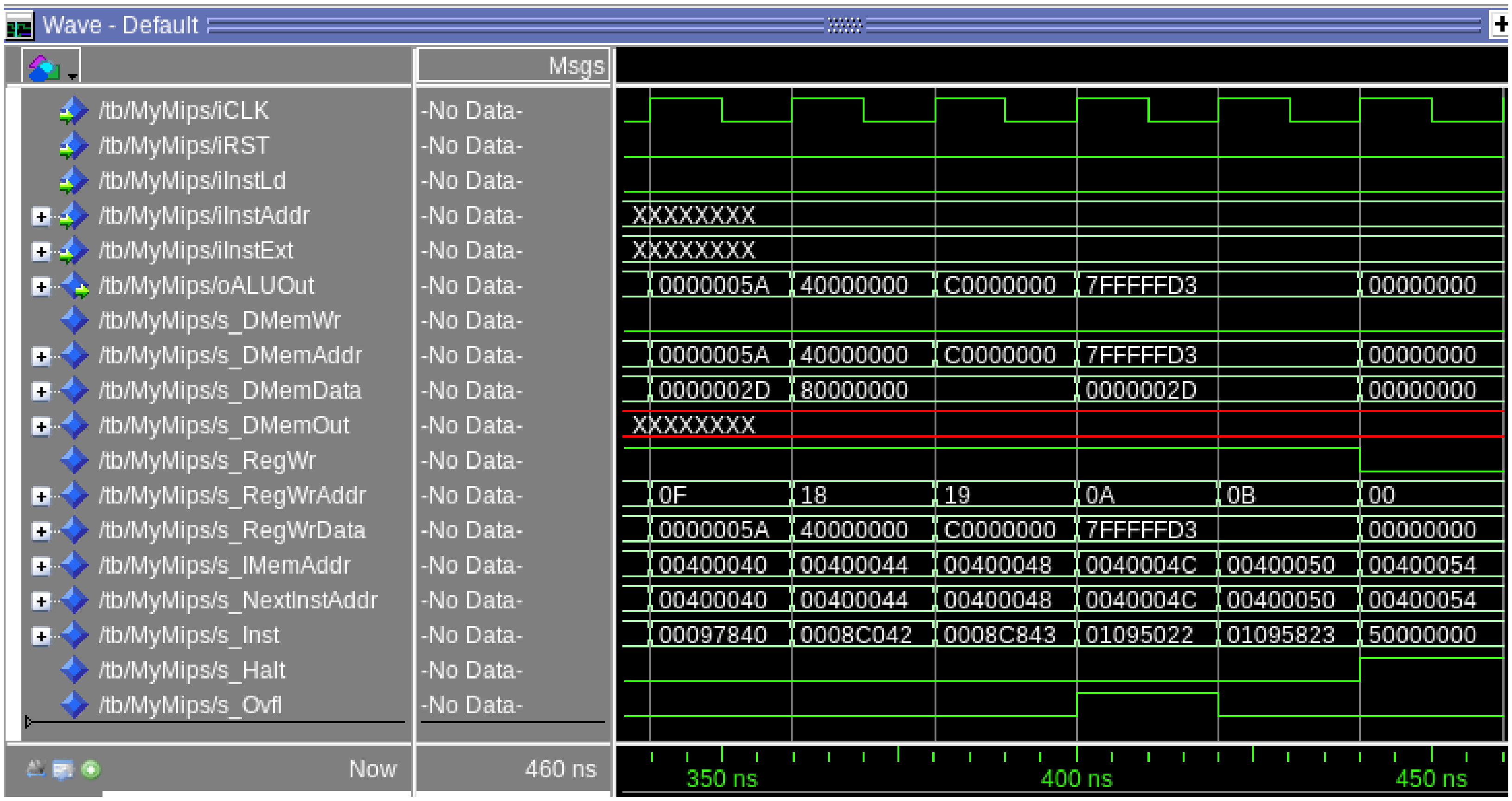
```
Wave - Default
                          Msgs
/tb/MyMips/iCLK           -No Data-
/tb/MyMips/iRST           -No Data-
/tb/MyMips/iInstLd        -No Data-
/tb/MyMips/iInstAddr      -No Data-  XXXXXXXX
/tb/MyMips/iInstExt       -No Data-  XXXXXXXX
/tb/MyMips/oALUOut        -No Data-  80000000  00000000  0000002D  8000002D  7FFFFC45  8000002D  80000415  0000002D  00000005  7FF
/tb/MyMips/s_DMemWr       -No Data-
/tb/MyMips/s_DMemAddr     -No Data-  80000000  00000000  0000002D  8000002D  7FFFFC45  8000002D  80000415  0000002D  00000005  7FF
/tb/MyMips/s_DMemData     -No Data-  00000000  00000000  80000000  00000000  0000002D  00000000  8000002D  00000000  800
/tb/MyMips/s_DMemOut      -No Data-  XXXXXXXX
/tb/MyMips/s_RegWr        -No Data-
/tb/MyMips/s_RegWrAddr    -No Data-  08        09        0A        0B        0C        0D        0E        0F        18
/tb/MyMips/s_RegWrData    -No Data-  80000000  00000000  0000002D  8000002D  7FFFFC45  8000002D  80000415  0000002D  00000005  7FF
/tb/MyMips/s_IMemAddr     -No Data-  XXXXXX...  00400000  00400004  00400008  0040000C  00400010  00400014  00400018  0040001C  00400020  004
/tb/MyMips/s_NextInstAddr -No Data-  XXXXXX...  00400000  00400004  00400008  0040000C  00400010  00400014  00400018  0040001C  00400020  004
/tb/MyMips/s_Inst         -No Data-  3C088000  3C090000  3529002D  01285020  214BFC18  01096021  258D03E8  012A7024  312F0005  012
/tb/MyMips/s_Halt         -No Data-
/tb/MyMips/s_Ovfl         -No Data-
Now                        460 ns    ns     50 ns     100 ns     150 ns     200 ns
```

[Part 3 (b)] Create and test an application which uses each of the required control-flow instructions and has a call depth of at least 5 (i.e., the number of activation records on the stack is at least 4). Name this file Proj1_cf_test.s.



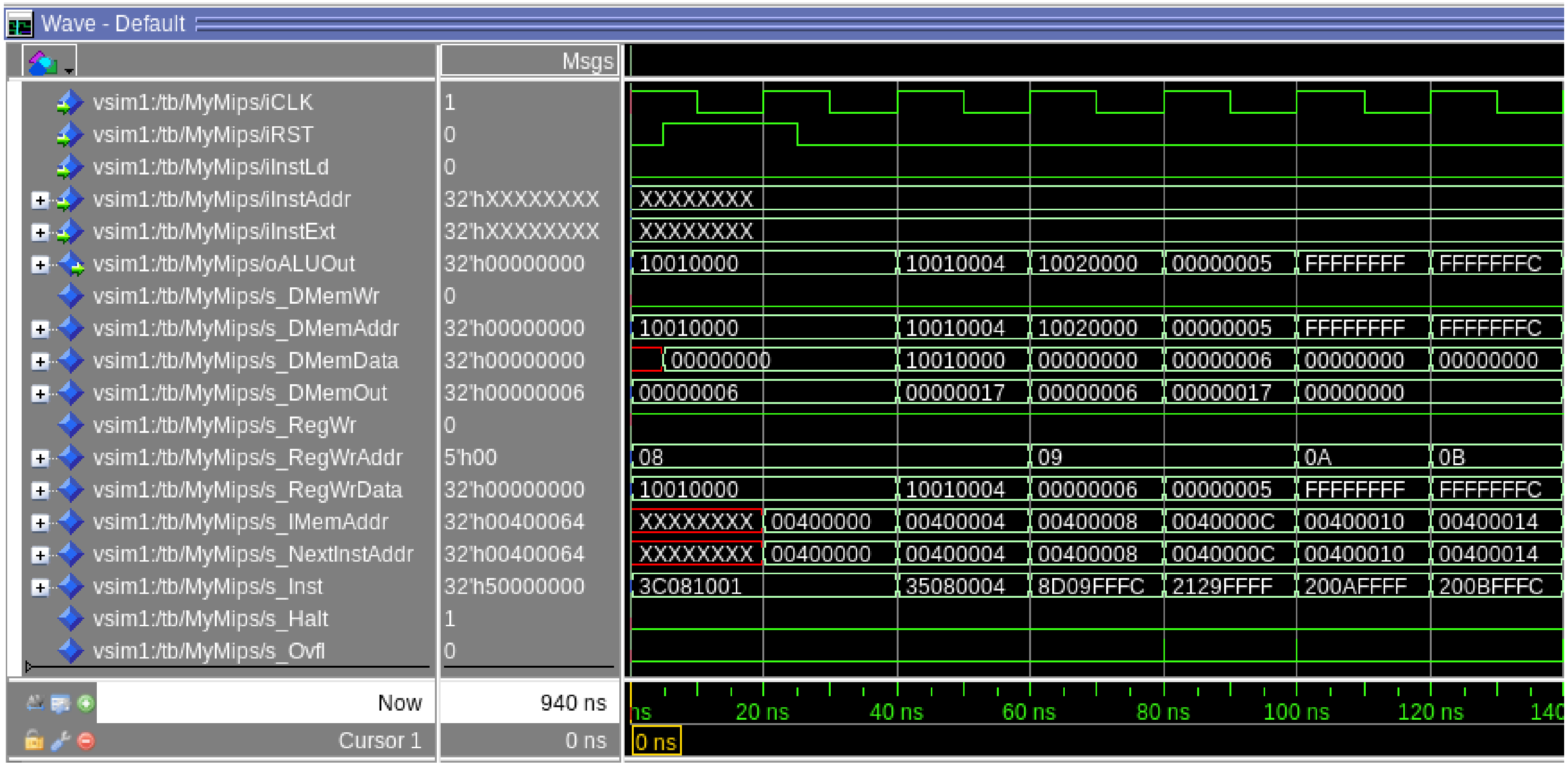
```
Wave - Default
                                Msgs
/tb/MyMips/iCLK                 0
/tb/MyMips/iRST                 0
/tb/MyMips/iInstLd              0
/tb/MyMips/iInstAddr            32'hXXXXXXXX   XXXXXXXX
/tb/MyMips/iInstExt             32'hXXXXXXXX   XXXXXXXX
/tb/MyMips/oALUOut              32'h00000001   7FFFEFE4   7FFFEFE8   7FFFEFEC
/tb/MyMips/s_DMemWr             0
/tb/MyMips/s_DMemAddr           32'h00000001   7FFFEFE4   7FFFEFE8   7FFFEFEC
/tb/MyMips/s_DMemData           32'h00000001   00000000   00400044   7FFFEFE4
/tb/MyMips/s_DMemOut            32'hXXXXXXXX   00000001   00400034   00000003
/tb/MyMips/s_RegWr              1
/tb/MyMips/s_RegWrAddr          5'h02          08         1F         1D
/tb/MyMips/s_RegWrData          32'h00000001   00000001   00400034   7FFFEFEC
/tb/MyMips/s_IMemAddr           32'h00400050   00400044   00400048   0040004C
/tb/MyMips/s_NextInstAd...      32'h00400050   00400044   00400048   0040004C
/tb/MyMips/s_Inst               32'h00481020   8FA80000   8FBF0004   23BD0008
/tb/MyMips/s_Halt               0
/tb/MyMips/s_Ovfl               0

Now         1820 ns        780 ns   800 ns   820
Cursor 1    839 ns
```


```
Wave - Default
                                 Msgs
/tb/MyMips/iCLK                  -No Data-
/tb/MyMips/iRST                  -No Data-
/tb/MyMips/iInstLd               -No Data-
/tb/MyMips/iInstAddr             -No Data-   XXXXXXXX
/tb/MyMips/iInstExt              -No Data-   XXXXXXXX
/tb/MyMips/oALUOut               -No Data-   0000005A  40000000  C0000000  7FFFFFD3            00000000
/tb/MyMips/s_DMemWr              -No Data-
/tb/MyMips/s_DMemAddr            -No Data-   0000005A  40000000  C0000000  7FFFFFD3            00000000
/tb/MyMips/s_DMemData            -No Data-   0000002D  80000000            0000002D            00000000
/tb/MyMips/s_DMemOut             -No Data-   XXXXXXXX
/tb/MyMips/s_RegWr               -No Data-
/tb/MyMips/s_RegWrAddr           -No Data-   0F        18        19        0A        0B        00
/tb/MyMips/s_RegWrData           -No Data-   0000005A  40000000  C0000000  7FFFFFD3            00000000
/tb/MyMips/s_IMemAddr            -No Data-   00400040  00400044  00400048  0040004C  00400050  00400054
/tb/MyMips/s_NextInstAddr        -No Data-   00400040  00400044  00400048  0040004C  00400050  00400054
/tb/MyMips/s_Inst                -No Data-   00097840  0008C042  0008C843  01095022  01095823  50000000
/tb/MyMips/s_Halt                -No Data-
/tb/MyMips/s_Ovfl                -No Data-

Now                              460 ns        350 ns                  400 ns                  450 ns
```

[Part 3 (c)] Create and test an application that sorts an array with *N* elements using the BubbleSort algorithm (link). Name this file Proj1_bubblesort.s.



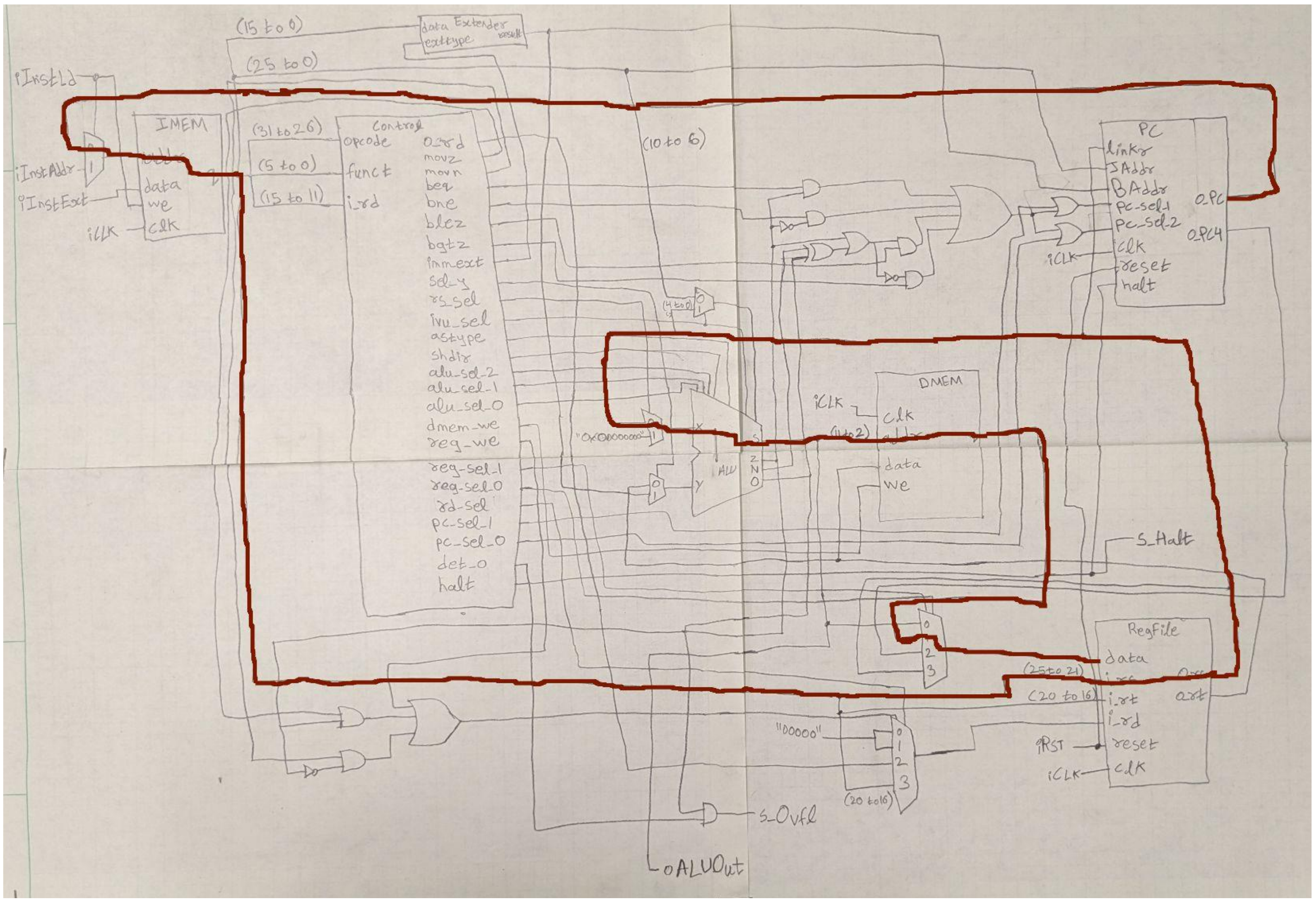
Wave - Default

```
                                Msgs
vsim1:/tb/MyMips/iCLK           1
vsim1:/tb/MyMips/iRST           0
vsim1:/tb/MyMips/iInstLd        0
vsim1:/tb/MyMips/iInstAddr      32'hXXXXXXXX   XXXXXXXX
vsim1:/tb/MyMips/iInstExt       32'hXXXXXXXX   XXXXXXXX
vsim1:/tb/MyMips/oALUOut        32'h00000000   10010000            10010004  10020000  00000005  FFFFFFFF  FFFFFFFC
vsim1:/tb/MyMips/s_DMemWr       0
vsim1:/tb/MyMips/s_DMemAddr     32'h00000000   10010000            10010004  10020000  00000005  FFFFFFFF  FFFFFFFC
vsim1:/tb/MyMips/s_DMemData     32'h00000000     00000000          10010000  00000000  00000006  00000000  00000000
vsim1:/tb/MyMips/s_DMemOut      32'h00000006   00000006            00000017  00000006  00000017  00000000
vsim1:/tb/MyMips/s_RegWr        0
vsim1:/tb/MyMips/s_RegWrAddr    5'h00          08                            09                  0A        0B
vsim1:/tb/MyMips/s_RegWrData    32'h00000000   10010000            10010004  00000006  00000005  FFFFFFFF  FFFFFFFC
vsim1:/tb/MyMips/s_IMemAddr     32'h00400064   XXXXXXXX  00400000  00400004  00400008  0040000C  00400010  00400014
vsim1:/tb/MyMips/s_NextInstAddr 32'h00400064   XXXXXXXX  00400000  00400004  00400008  0040000C  00400010  00400014
vsim1:/tb/MyMips/s_Inst         32'h50000000   3C081001            35080004  8D09FFFC  2129FFFF  200AFFFF  200BFFFC
vsim1:/tb/MyMips/s_Halt         1
vsim1:/tb/MyMips/s_Ovfl         0

Now       940 ns    ns   20 ns   40 ns   60 ns   80 ns   100 ns   120 ns   140
Cursor 1  0 ns      0 ns
```

[Part 4] report the maximum frequency your processor can run at and determine what your critical path is. Draw this critical path on top of your top-level schematics. What components would you focus on to improve the frequency?

Maximum Freq: 25.10 MHz
Component to Focus: Regfile Mux, Shifter and ALU Mux


(15 to 0)
data Extender
exttype
result
(25 to 0)
iInstLd
IMEM
iInstAddr
iInstExt
data
we
clk
iCLK
(31 to 26)
(5 to 0)
(15 to 11)
Control
opcode
funct
i_rd
o_rd
movz
movn
beq
bne
blez
bgtz
immext
sel_y
rs_sel
ivu_sel
astype
shdir
alu_sel_2
alu_sel_1
alu_sel_0
dmem_we
reg_we
reg_sel_1
reg_sel_0
rd_sel
pc_sel_1
pc_sel_0
det_o
halt
(10 to 6)
(4 to 0)
"0x00000000"
x
y
ALU
S
Z
N
O
DMEM
iCLK
clk
(11 to 2)
data
we
PC
linkr
JAddr
BAddr
pc_sel_1
pc_sel_2
clk
reset
halt
o_PC
o_PC4
iCLK
S_Halt
RegFile
data
(25 to 21)
(20 to 16)
i_rs
i_rt
i_rd
reset
clk
o_rs
o_rt
iRST
iCLK
"00000"
(20 to 16)
S_Ovfl
oALUOut