CprE 381: Computer Organization and Assembly-Level Programming

Project Part 2 Report

Team Members: _____Fadahunsi Adeife_____

_____Jesutofunmi Obimakinde_____

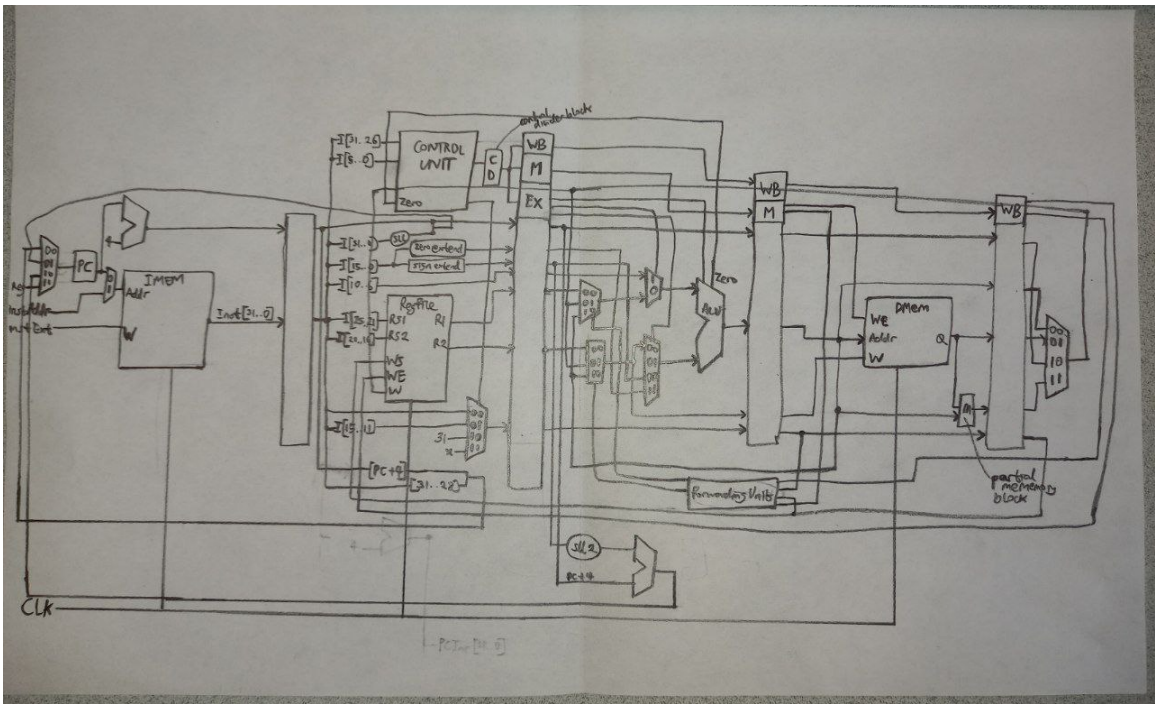
Project Teams Group #:_____2_01_____

Refer to the highlighted language in the project 1 instruction for the context of the following questions.

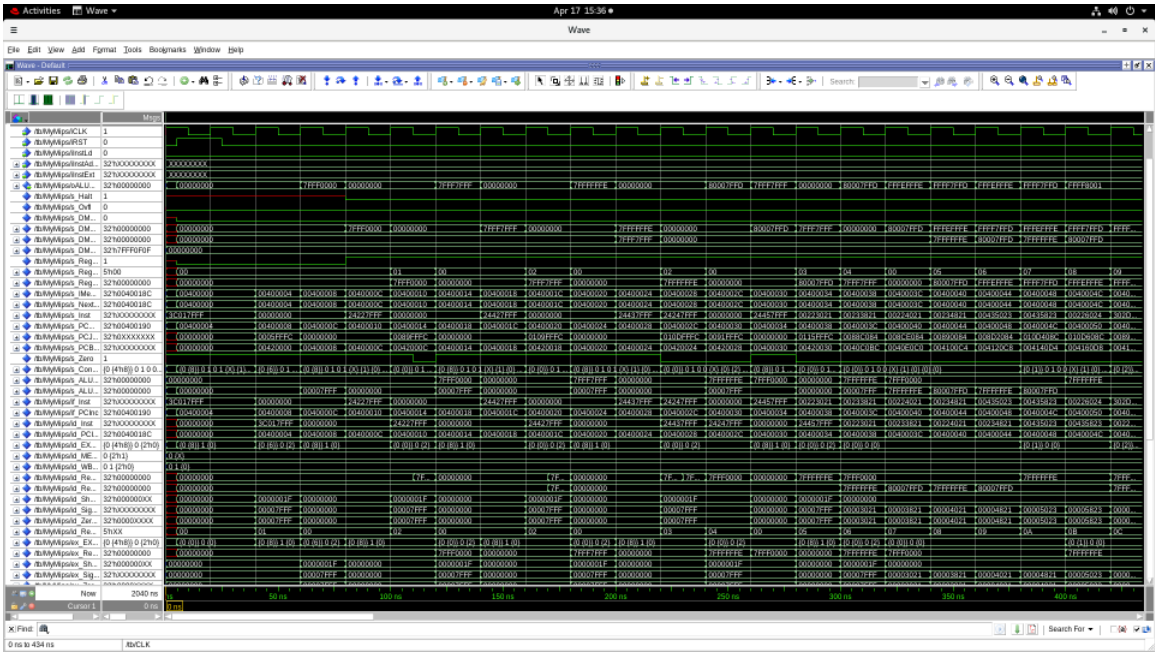
[1.a] Come up with a global list of the datapath values and control signals that are required during each pipeline stage.

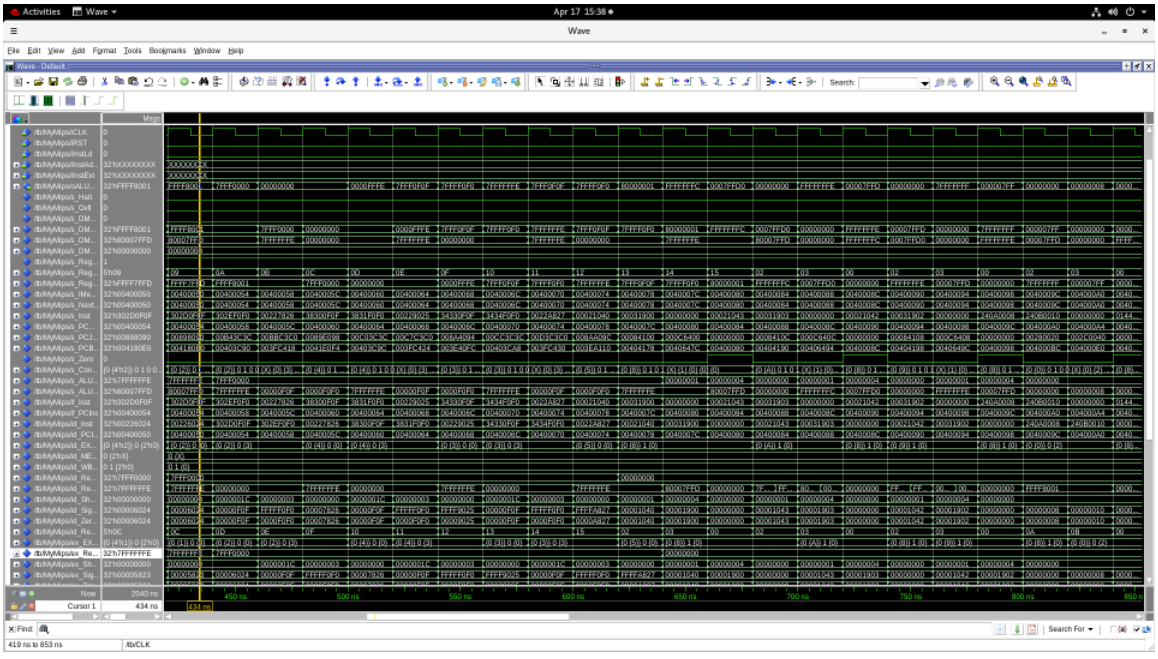
control_unit_pipeline

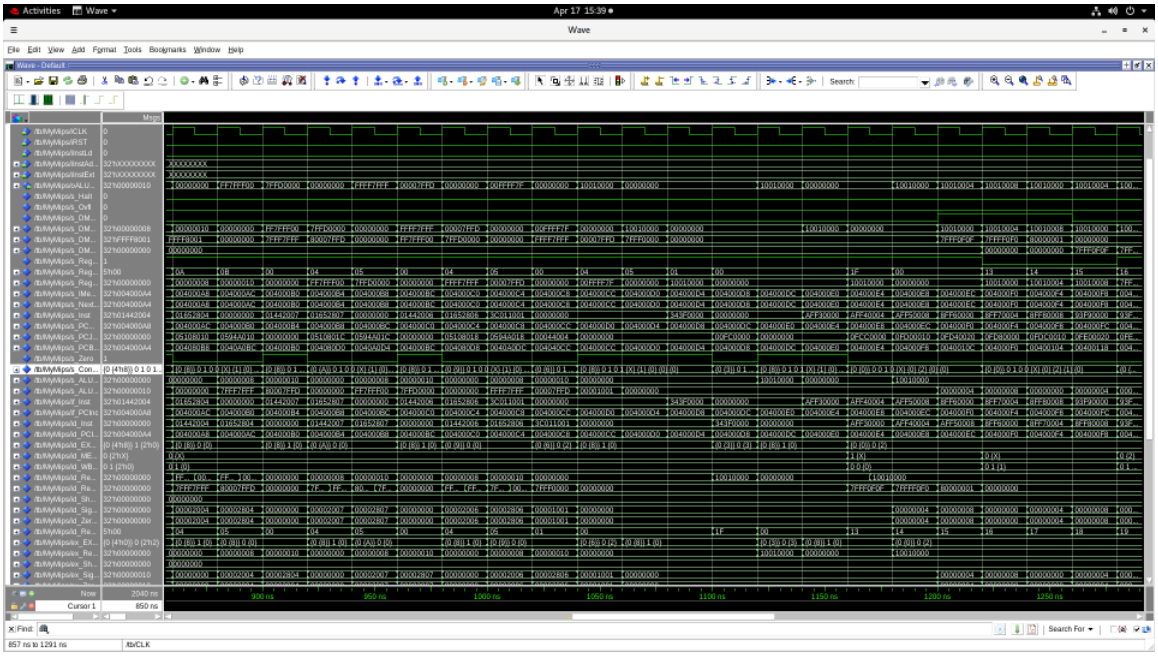
[1.b.ii] high-level schematic drawing of the interconnection between components.

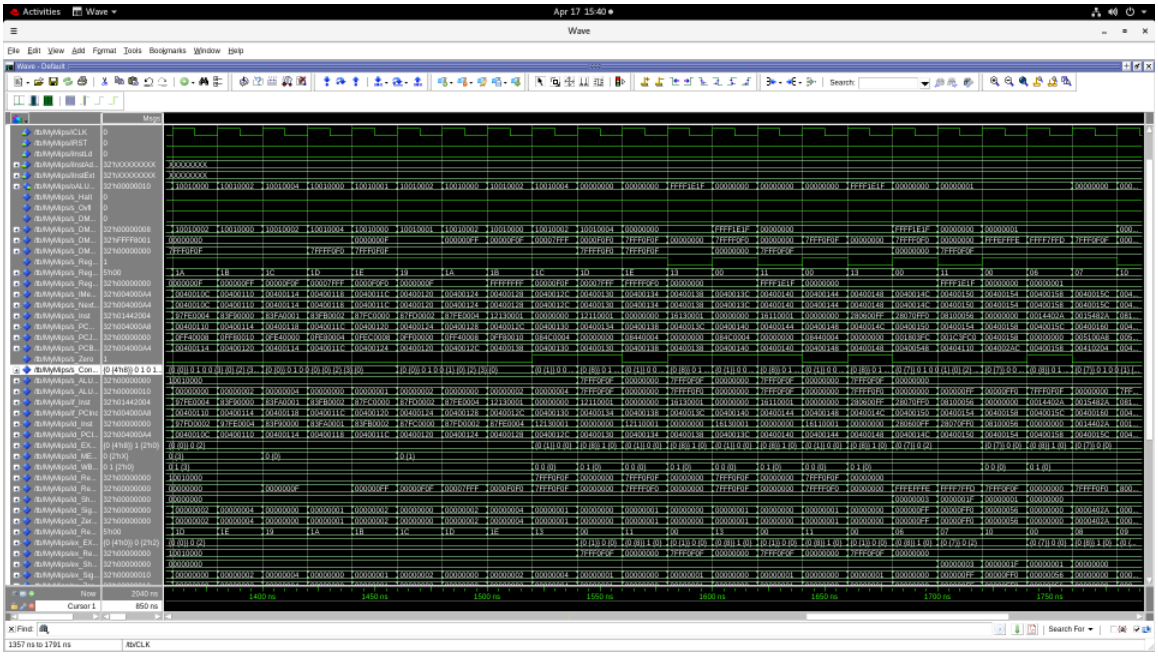


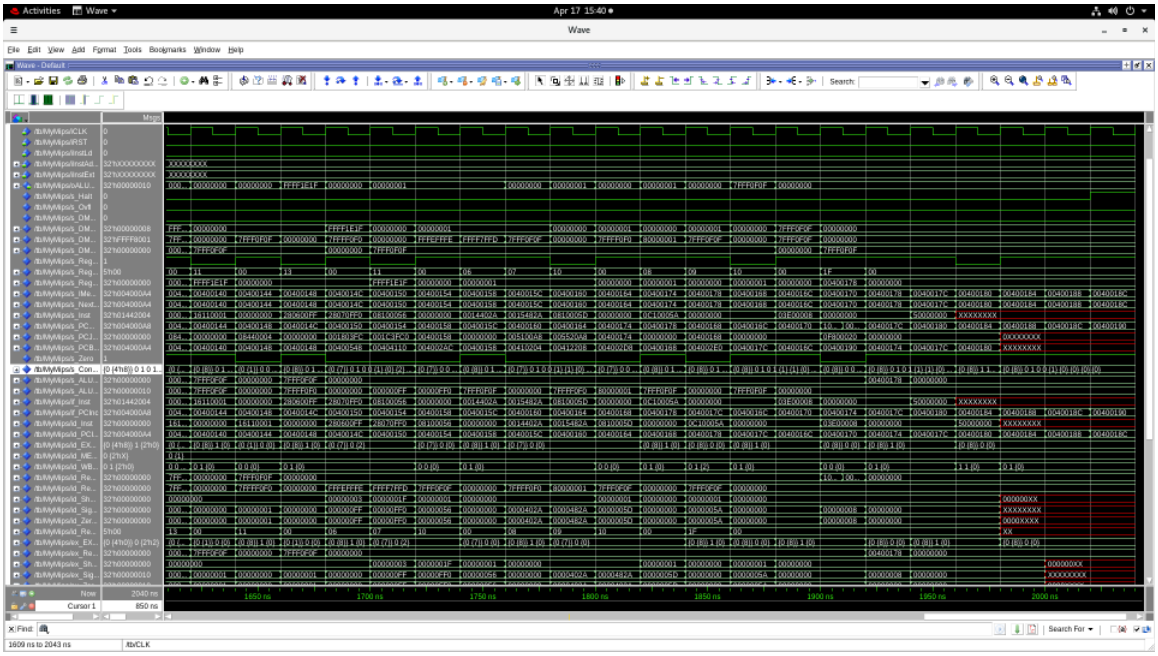
[1.c.i] include an annotated waveform in your writeup and provide a short discussion of result correctness.


Activities
Wave
Apr 17 15:36
Wave
File Edit View Add Format Tools Bookmarks Window Help
Wave - Default
Msgs
/tbMyMips/sCLK
/tbMyMips/sRST
/tbMyMips/sInstLd
/tbMyMips/sInstAd...
/tbMyMips/sInstExt
/tbMyMips/sALU...
/tbMyMips/s_Halt
/tbMyMips/s_Ovfl
/tbMyMips/s_DM...
/tbMyMips/s_Reg...
/tbMyMips/s_Zero
/tbMyMips/s_Con...
/tbMyMips/sld_ME...
/tbMyMips/sld_WB...
/tbMyMips/sld_Re...
/tbMyMips/sld_Sh...
/tbMyMips/sld_Sig...
/tbMyMips/sld_Zer...
/tbMyMips/sld_Re...
/tbMyMips/slex_EX...
/tbMyMips/slex_Re...
/tbMyMips/slex_Sh...
/tbMyMips/slex_Sig...
Now
2040 ns
Cursor 1
0 ns
0 ns to 434 ns
/tb/CLK


Activities
Wave
Apr 17 15:38
Wave
File Edit View Add Format Tools Bookmarks Window Help
Wave - Default
Msgs
/tbMyMips/sCLK
/tbMyMips/sRST
/tbMyMips/sInstLd
/tbMyMips/sInstAd...
/tbMyMips/sInstExt
/tbMyMips/sALU...
/tbMyMips/s_Halt
/tbMyMips/s_Ovfl
/tbMyMips/s_DM...
/tbMyMips/s_Reg...
/tbMyMips/s_Zero
/tbMyMips/s_Con...
/tbMyMips/sld_ME...
/tbMyMips/sld_WB...
/tbMyMips/sld_Re...
/tbMyMips/sld_Sh...
/tbMyMips/sld_Sig...
/tbMyMips/sld_Zer...
/tbMyMips/sld_Re...
/tbMyMips/slex_EX...
/tbMyMips/slex_Re...
/tbMyMips/slex_Sh...
/tbMyMips/slex_Sig...
Now
2040 ns
Cursor 1
434 ns
419 ns to 853 ns
/tb/CLK


Activities
Wave
Apr 17 15:39
Wave
File Edit View Add Format Tools Bookmarks Window Help
Msgs
/tbMyMips/sCLK
/tbMyMips/sRST
/tbMyMips/sInstLd
/tbMyMips/sInstAd...
/tbMyMips/sInstExt
/tbMyMips/sALU...
/tbMyMips/s_Halt
/tbMyMips/s_Ovfl
/tbMyMips/s_DM...
/tbMyMips/s_DM...
/tbMyMips/s_DM...
/tbMyMips/s_DM...
/tbMyMips/s_Reg...
/tbMyMips/s_Reg...
/tbMyMips/s_Reg...
/tbMyMips/s_Reg...
/tbMyMips/s_Me...
/tbMyMips/s_Next...
/tbMyMips/s_Inst
/tbMyMips/s_PC...
/tbMyMips/s_PCJ...
/tbMyMips/s_PCB...
/tbMyMips/s_Zero
/tbMyMips/s_Con...
/tbMyMips/s_ALU...
/tbMyMips/s_ALU...
/tbMyMips/sIf_Inst
/tbMyMips/sIf_PCInc
/tbMyMips/sId_Inst
/tbMyMips/sId_PCJ...
/tbMyMips/sId_EX...
/tbMyMips/sId_ME...
/tbMyMips/sId_WB...
/tbMyMips/sId_Re...
/tbMyMips/sId_Re...
/tbMyMips/sId_Sh...
/tbMyMips/sId_Sig...
/tbMyMips/sId_Zer...
/tbMyMips/sId_Re...
/tbMyMips/sIex_EX...
/tbMyMips/sIex_Re...
/tbMyMips/sIex_Sh...
/tbMyMips/sIex_Sig...
Now
Cursor 1
900 ns
950 ns
1000 ns
1050 ns
1100 ns
1150 ns
1200 ns
1250 ns
Find:
Search For
857 ns to 1291 ns
/tbvCLK


Activities
Wave
Apr 17 15:40
Wave
File Edit View Add Format Tools Bookmarks Window Help
Msgs
/tbMyMips/sCLK
/tbMyMips/sRST
/tbMyMips/sInstLd
/tbMyMips/sInstAd...
/tbMyMips/sInstExt
/tbMyMips/sALU...
/tbMyMips/s_Halt
/tbMyMips/s_Ovfl
/tbMyMips/s_DM...
/tbMyMips/s_DM...
/tbMyMips/s_DM...
/tbMyMips/s_DM...
/tbMyMips/s_Reg...
/tbMyMips/s_Reg...
/tbMyMips/s_Reg...
/tbMyMips/s_Reg...
/tbMyMips/s_Me...
/tbMyMips/s_Next...
/tbMyMips/s_Inst
/tbMyMips/s_PC...
/tbMyMips/s_PCJ...
/tbMyMips/s_PCB...
/tbMyMips/s_Zero
/tbMyMips/s_Con...
/tbMyMips/s_ALU...
/tbMyMips/s_ALU...
/tbMyMips/sIf_Inst
/tbMyMips/sIf_PCInc
/tbMyMips/sId_Inst
/tbMyMips/sId_PCJ...
/tbMyMips/sId_EX...
/tbMyMips/sId_ME...
/tbMyMips/sId_WB...
/tbMyMips/sId_Re...
/tbMyMips/sId_Re...
/tbMyMips/sId_Sh...
/tbMyMips/sId_Sig...
/tbMyMips/sId_Zer...
/tbMyMips/sId_Re...
/tbMyMips/sIex_EX...
/tbMyMips/sIex_Re...
/tbMyMips/sIex_Sh...
/tbMyMips/sIex_Sig...
Now
Cursor 1
1400 ns
1450 ns
1500 ns
1550 ns
1600 ns
1650 ns
1700 ns
1750 ns
Find:
Search For
1357 ns to 1791 ns
/tbvCLK



The testbench encompasses a range of operations including addition, subtraction, logical operations (AND, OR, XOR, NOR), comparison (SLT), and shifting operations (logical and arithmetic). Each operation is meticulously tested to ensure the ALU's functionality aligns with expectations.

During the addition operation test case, with inputs s_iD0 and s_iD1 set to 0x00000001 and 0x00000002 respectively, and alu_select set to 0000, the expected output (s_oQ) of 0x00000003 (1 + 2 = 3) is calculated. Similarly, in the subtraction operation test case, where inputs s_iD0 and s_iD1 are 0x00000005 and 0x00000002 respectively, and alu_select is 0001, the expected output (s_oQ) of 0x00000003 (5 - 2 = 3) is determined.

Logical operations such as AND, OR, XOR, and NOR are tested with appropriate inputs and alu_select values. For instance, in the AND operation test case, inputs s_iD0 and s_iD1 are set to 0x00000F0F and 0x00000FF0 respectively, resulting in an output (s_oQ) of 0x00000F00.

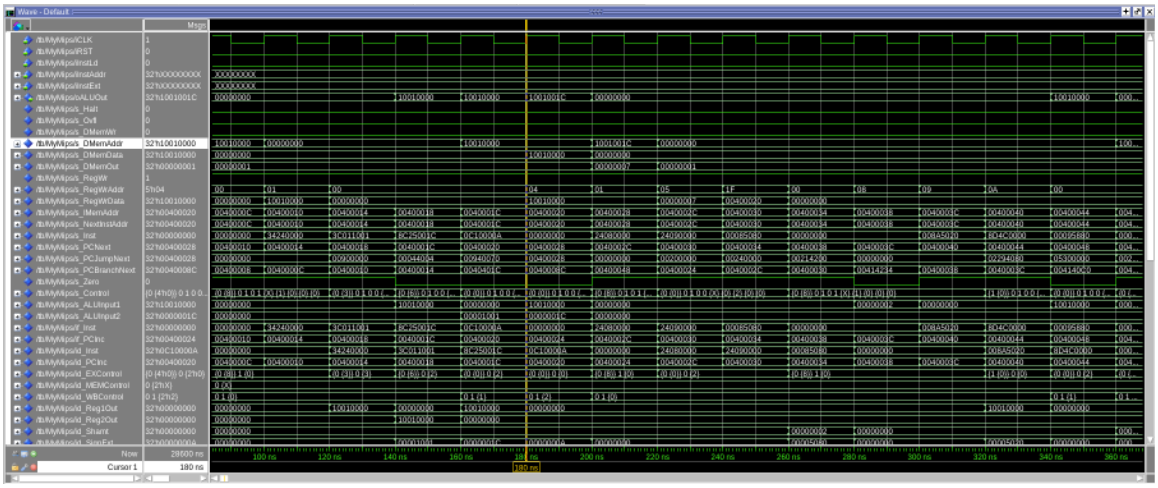
Furthermore, shifting operations, both logical and arithmetic, are examined. In the shift right logical operation test case, input s_iD0 is set to 0x00000008, which is then shifted right by 2 bits, yielding an output (s_oQ) of 0x00000002.

Additionally, the SLT operation is verified, ensuring correct comparison functionality. For instance, in the SLT operation test case, where inputs s_iD0 and s_iD1 are 0x00000001 and 0x00000002 respectively, the ALU correctly returns 0x00000001 as the output (s_oQ), indicating that 1 is less than 2.

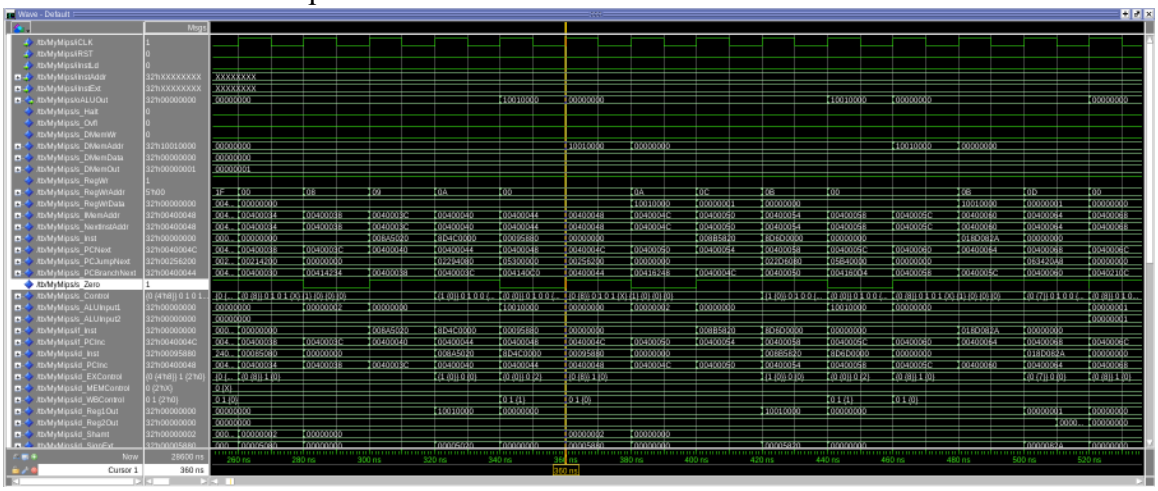
[1.c.ii] Include an annotated waveform in your writeup of two iterations or recursions of these programs executing correctly and provide a short discussion of result correctness. In your

waveform and annotation, provide 3 different examples (at least one data-flow and one control-flow) of where you did not have to use the maximum number of NOPs.

Bubble Sort:



In the above screenshot, we can see that a nop instruction is inserted at 180ns as the s_Inst becomes 0x0000_0000 because a nop operation is required after the jal instruction to allow time for register 31 to be written, which is why RegWrAddr becomes “1F” 4 clock cycles later to avoid a possible control hazard. Two additional nops are also inserted at 260ns where the s_Inst becomes 0x0000_0000 again to treat the data dependency between two instructions where the former writes to $t2 while the next instruction attempts to read the same register. Hence, we can see that 0A becomes the write address, RegWrAddr in 320ns which is 4 clock cycles later from when the instruction was read in 240ns, this helps to write to the register file so the next instruction at 300ns(which reads $t2) can have the newest value of $t2 upon reaching the decode stage. From the above approaches, the program can execute properly without using the maximum number of nops



The same occurs in the above screenshot where a data hazard is avoided using a couple of nops inserted at 360ns in which s_Inst becomes 0x0000_0000 to allow the previous instruction to write properly to its destination register 4 clock cycles later where RegWrAddr is “0B” and it allows the following instruction to read that same register

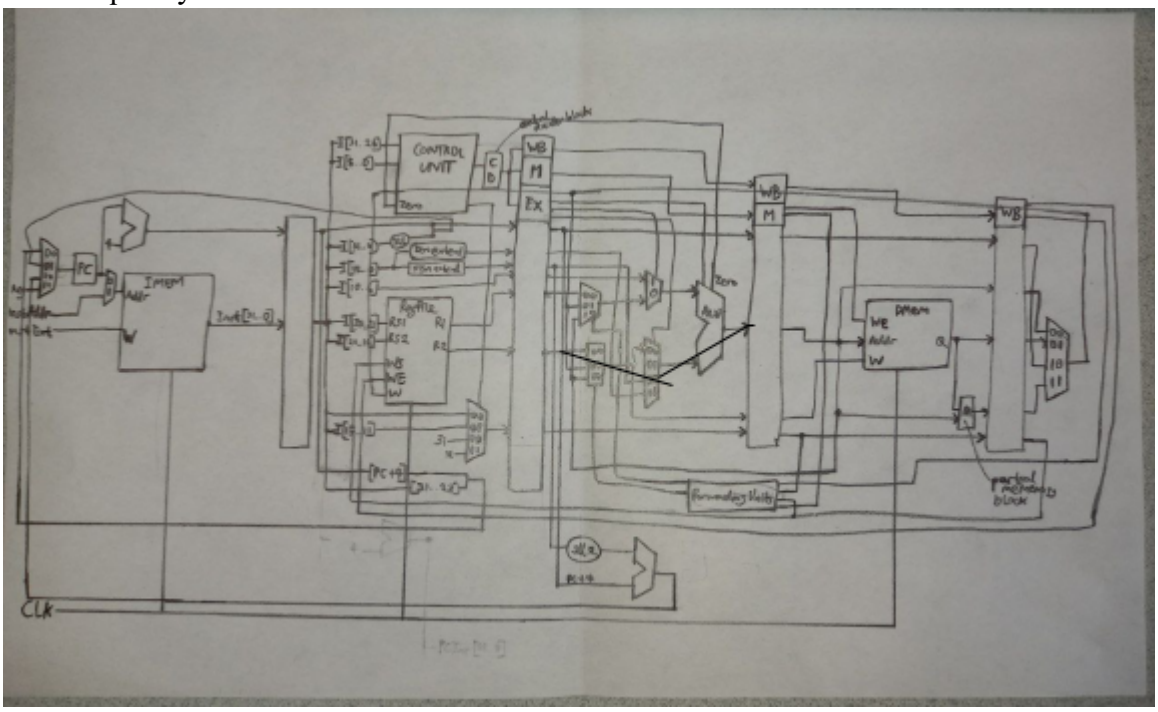
without any hazards occuring. In this approach as well, the program can executes properly without using the maximum number of nops.

```
bash-4.4$ ./381_tf.sh test proj/mips/Proj1_bubblesort.s
Using LAB Python Environment
Testing
All VHDL src files compiled successfully
Testing file: proj/mips/Proj1_bubblesort.s
Mars simulation: pass
Modelsim simulation: pass
Test Result: pass
Mars Instructions: 1247
Processor Cycles: 1429
CPI:  1.15
Results in: output/Proj1_bubblesort.s
-----------------------------------------------
bash-4.4$
```

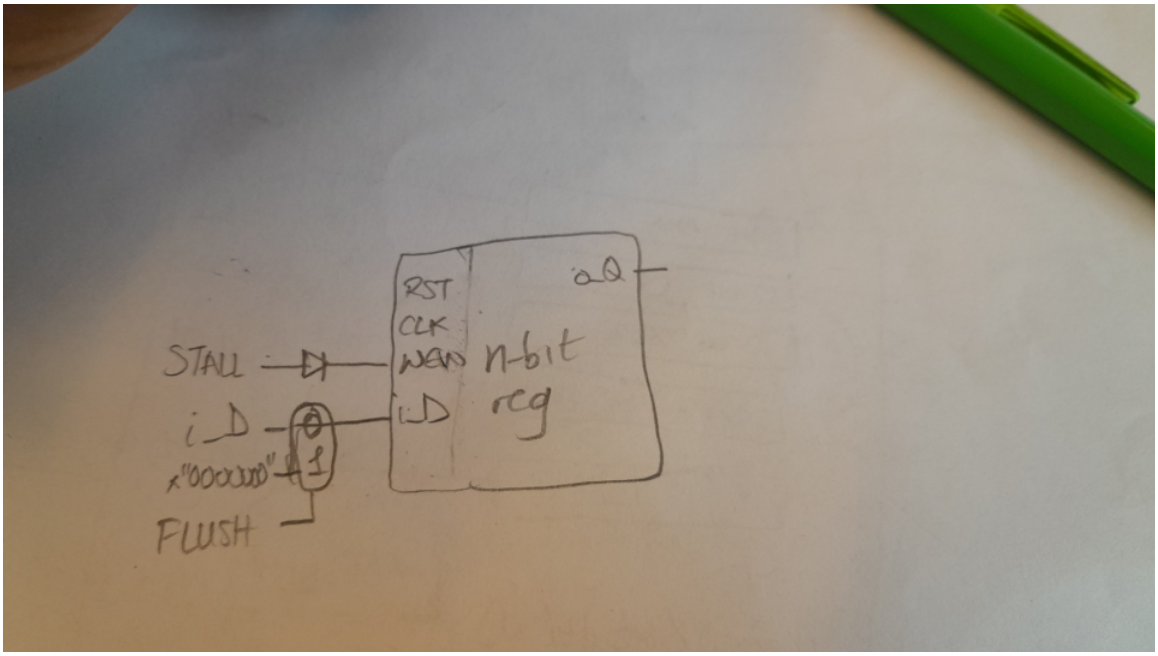
Finally, the above screenshot shows our processor passing the test case for this bubble sort algorithm. Therefore, it reiterates that our program executes and performs correctly.

[1.d] report the maximum frequency your software-scheduled pipelined processor can run at and determine what your critical path is (specify each module/entity/component that this path goes through).

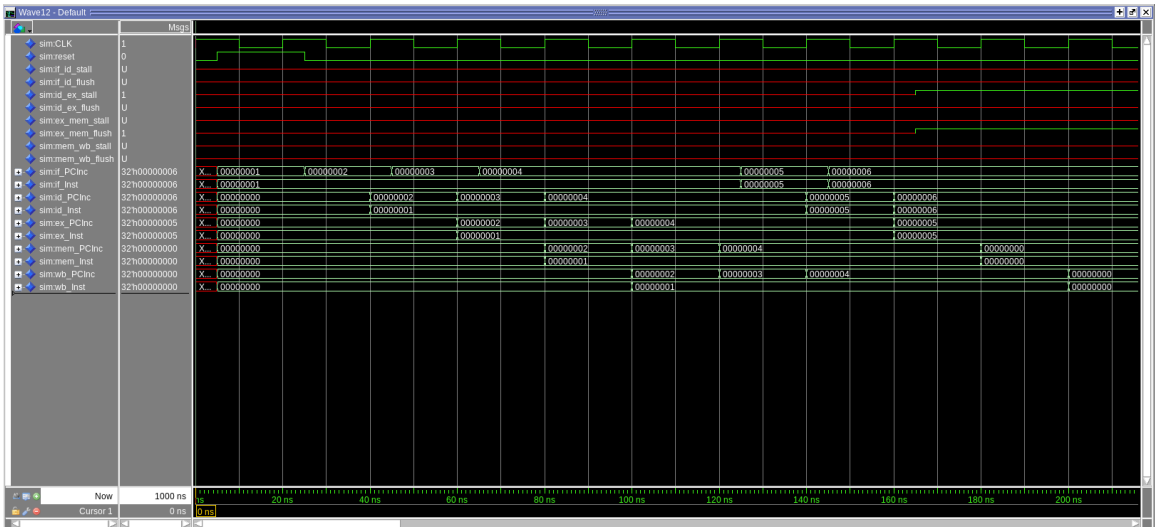
Max frequency: 55.55mhz



[2.a.ii] Draw a simple schematic showing how you could implement stalling and flushing operations given an ideal N-bit register.



[2.a.iii] Create a testbench that instantiates all four of the registers in a single design. Show that values that are stored in the initial IF/ID register are available as expected four cycles later, and that new values can be inserted into the pipeline every single cycle. Most importantly, this testbench should also test that each pipeline register can be individually stalled or flushed.



In file tb_pip_reg.vhd, we start by passing 2, 3, and 4 each after one clock cycle and we watch as these values propagate through the pipeline after as seen in the waveform. THen we go on to pass 5 and 6 into the pipeline one after the other. THen when 5 was in the execute stage and 6 was in the decode stage, I turned on the stall for id_ex pipeline and flushed the ex_mem pipeline. And as expected the 5 zeroed out because of the flush when entering into the mem stage and the 6 was no longer passing out of the id cause the stall for the id_ex pipeline was on

[2.b.i] list which instructions produce values, and what signals (i.e., bus names) in the pipeline these correspond to.

add (oALUOut, RegWrAddr), addi (oALUOut, RegWrAddr), addiu (oALUOut, RegWrAddr), addu (oALUOut, RegWrAddr), and (oALUOut, RegWrAddr), andi (oALUOut, RegWrAddr), lw, nor (oALUOut, RegWrAddr), xor (oALUOut, RegWrAddr), xori (oALUOut, RegWrAddr), or (oALUOut, RegWrAddr), ori (oALUOut, RegWrAddr), slt (oALUOut, RegWrAddr), slti (oALUOut, RegWrAddr), sltiu (oALUOut, RegWrAddr), sltu (oALUOut, RegWrAddr), sll (oALUOut, RegWrAddr), srl (oALUOut, RegWrAddr), sra (oALUOut, RegWrAddr), sllv (oALUOut, RegWrAddr), srlv (oALUOut, RegWrAddr), srav (oALUOut, RegWrAddr), sub (oALUOut, RegWrAddr), subu (oALUOut, RegWrAddr), jal (RegWrAddr)

[2.b.ii] List which of these same instructions consume values, and what signals in the pipeline these correspond to.

add (Reg1Addr, Reg2Addr), addi (Reg1Addr), addiu (Reg1Addr), addu (Reg1Addr, Reg2Addr), and (Reg1Addr, Reg2Addr), andi (Reg1Addr), lw (Reg1Addr), nor (Reg1Addr, Reg2Addr), xor (Reg1Addr, Reg2Addr), xori (Reg1Addr), or (Reg1Addr, Reg2Addr), ori (Reg1Addr), slt (Reg1Addr, Reg2Addr), slti (Reg1Addr), sltiu (Reg1Addr), sltu (Reg1Addr, Reg2Addr), sll (Reg1Addr), srl (Reg1Addr), sra (Reg1Addr), sllv (Reg1Addr, Reg2Addr), srlv (Reg1Addr, Reg2Addr), srav (Reg1Addr, Reg2Addr), sub (Reg1Addr, Reg2Addr), subu (Reg1Addr, Reg2Addr), jal (PC)

[2.b.iii] generalized list of potential data dependencies. From this generalized list, select those dependencies that can be forwarded (write down the corresponding pipeline stages that will be forwarding and receiving the data), and those dependencies that will require hazard stalls.

Forwarding:
Any operation that writes to a register, Rd or Rt, followed by an operation that reads that same register at Rs or Rt 1 or 2 cycles later (forwards from MEM stage to EX stage if 1 cycle ahead, from WB to EX stage if 2 cycles ahead).
Forwarding is also needed if a lw operation is followed by an operation that reads the register that was loaded to 2 cycles later (forwards from MEM stage to EX stage)
Hazard stalls:
One stall cycle is needed if a lw operation is followed by an operation that reads the register that was loaded to 1 cycle later.
e.g lw $1, 4($2)
    sub $4, $1, $5

[2.b.iv] global list of the datapath values and control signals that are required during each pipeline stage

control_unit_hardware_pipeline

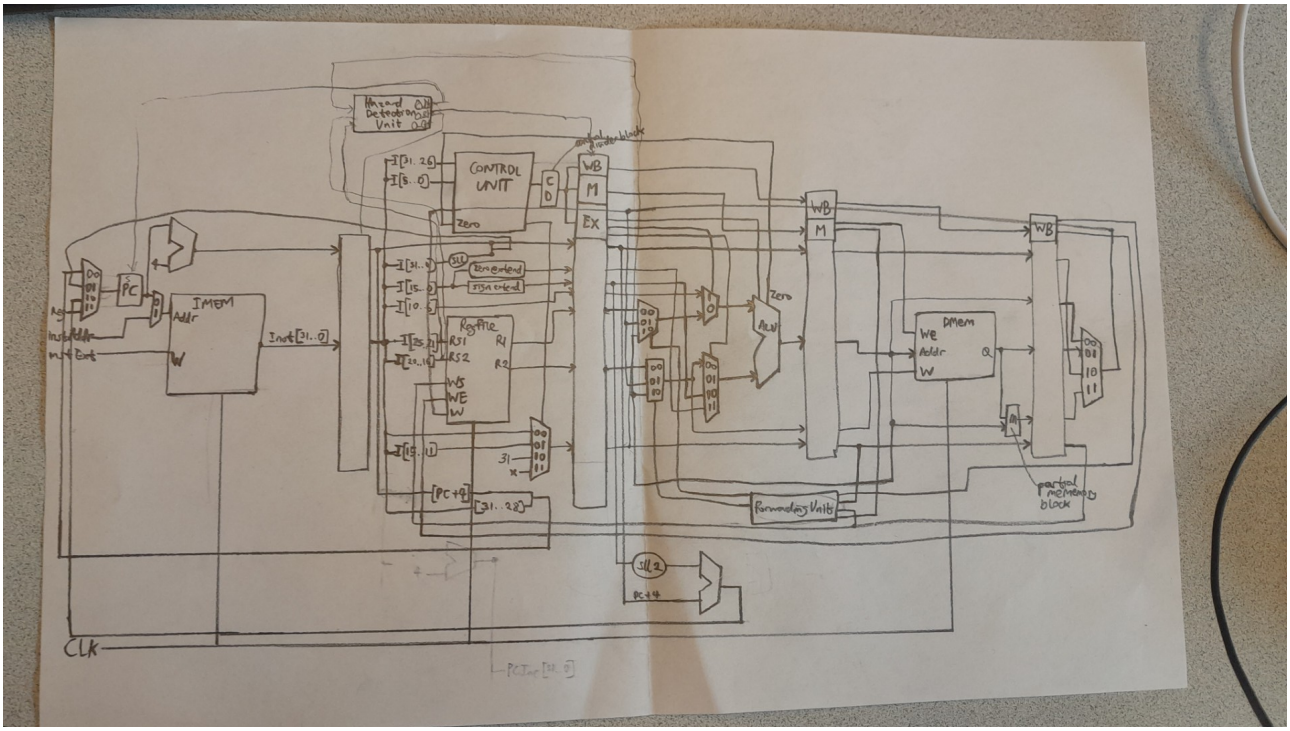
[2.c.i] list all instructions that may result in a non-sequential PC update and in which pipeline stage that update occurs.

- beq, bne, jr - execute stage
- j, jal - decode stage

[2.c.ii] For these instructions, list which stages need to be stalled and which stages need to be squashed/flushed relative to the stage each of these instructions is in.

Instruction	HDU
j, jal	if pcsel == jump, we flush IF/ID.
beq, bne, jr	if pcsel=branch or jr, we flush if/id and id/ex
lw, lb, lbu, lh, lhu	stall PC and IF/ID. Then we flush ID/EX.

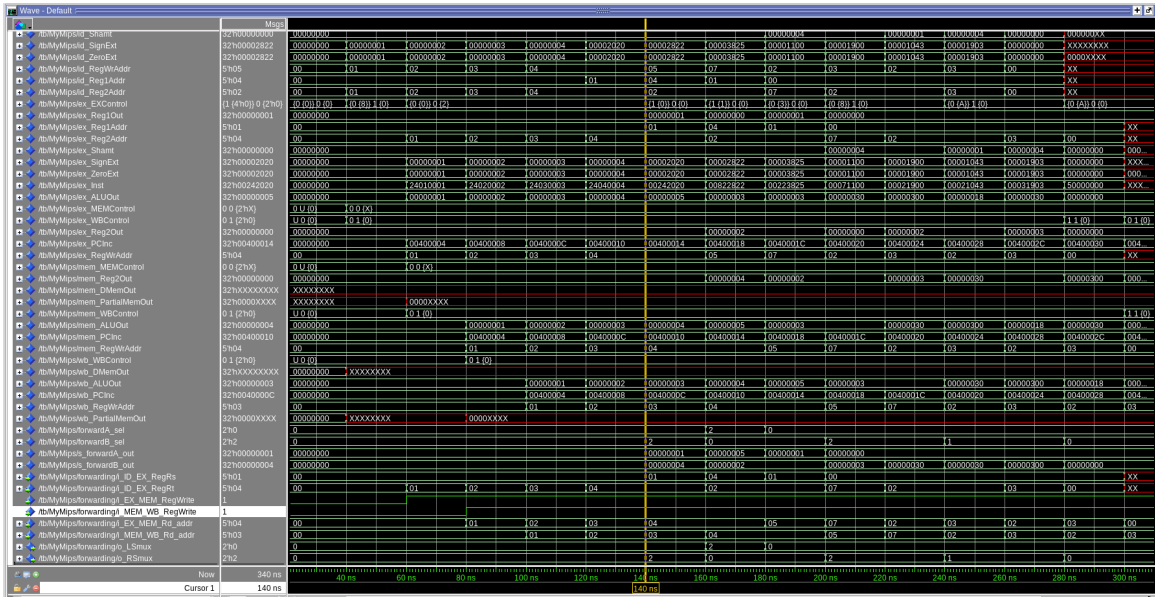
[2.d] implement the hardware-scheduled pipeline using only structural VHDL. As with the previous processors that you have implemented, start with a high-level schematic drawing of the interconnection between components.



[2.e – i, ii, and iii] In your writeup, show the QuestaSim output for each of the following tests, and provide a discussion of result correctness. It may be helpful to also annotate the waveforms directly.

[2.e.i] Create a spreadsheet to track these cases and justify the coverage of your testing approach. Include this spreadsheet in your report as a table.

Instruction	Lines	Hazards
add	lines 4 and 5	ex hazard
sub	lines 4 and 6	ex hazard
sll	lines 7 and 8	ex hazard
sll	lines 8 and 9	ex hazard
sra	lines 8 and 10	mem hazard
sra	lines 9 and 11	mem hazard



From the above screenshot, we can see the processor performs the regular write operations to register 1, 2, 3, 4 at 60ns, 80ns, 100ns, and 120ns respectively. However, the next instruction at 140ns, which was line 5 shows a read from register 4 which is written to in the previous operation at line 4. Thus, it detects this ex hazard and forwards the updated value of register 4 from the ex stage, which is why RSmux(Rt) becomes 2, which is an indication of an ex hazard detection. We also observe this occur again at 160ns, which is the instruction at line 6 that reads register 4 at Rs, which is why LSmux(Rs) indicates another hazard to perform a forwarding of the updated value of register 4. This shows our processor working properly on its forwarding logic and hazard detection.

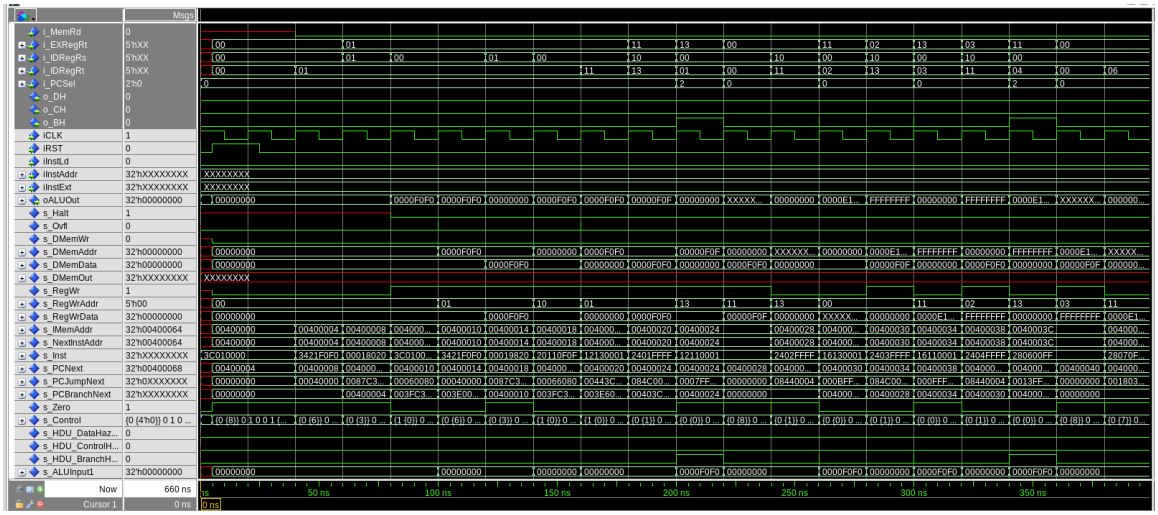
```
-------------------------------------------
bash-4.4$ ./381_tf.sh test proj/mips/hazarddect.s
Using VDI Python Environment
Testing
WARNING: Software tree location not set or invalid, using $MGC_HOME=/usr/local/mentor/calibre
All VHDL src files compiled successfully
Testing file: proj/mips/hazarddect.s
Mars simulation: pass
Modelsim simulation: pass
Test Result: pass
Mars Instructions: 12
Processor Cycles: 16
CPI:  1.33
Results in: output/hazarddect.s
-------------------------------------------
bash-4.4$
```

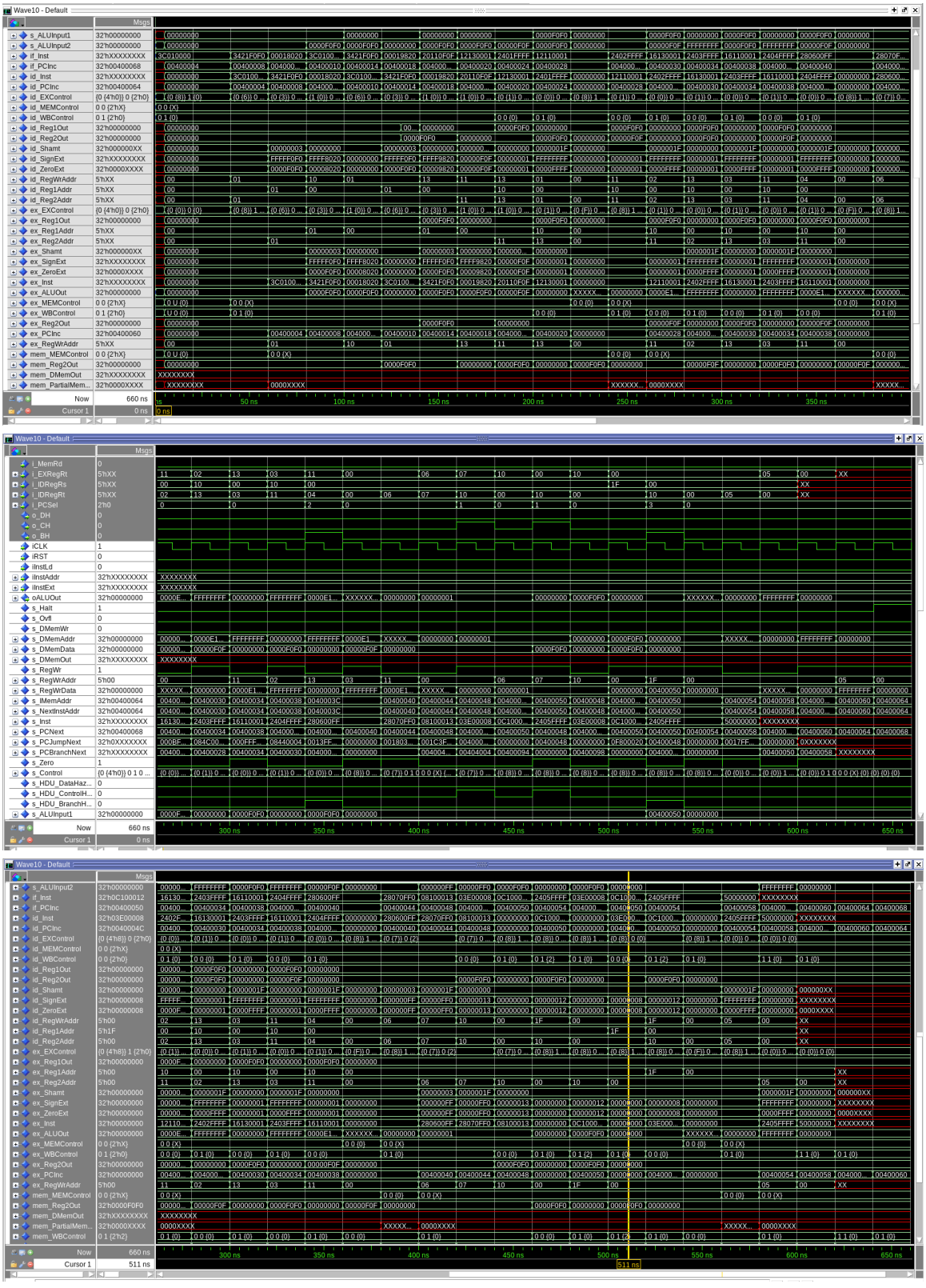
From the above screenshot shows our processor passing the testing for data forwarding and hazard capabilities. Therefore, it reiterates that our program executes and performs correctly.

[2.e.ii] Create a spreadsheet to track these cases and justify the coverage of your testing approach. Include this spreadsheet in your report as a table.

t_control.sh

Instruction	Lines	Hazards
beq	lines 5 and 9	ex hazard
bne	lines 13 and 17	ex hazard
jr	lines 27	ex hazard
jal	lines 30	id hazard
j	lines 24	id hazard





From annotation, we can see that s_HDU_Branch becomes 1 when we have a jr and a bne or beq instruction that fulfills condition. We then notice after s_HDU_Branch

becomes active dat the id_ex and the if_id pipeline registers are flush and that the output from them are all 0's.

From annotation, we can see that s_HDU_Control becomes 1 when we have a jal and j. We then notice after s_HDU_Control becomes active dat the the if_id pipeline registers are flush and that the output from them are all 0's.

[2.f] report the maximum frequency your hardware-scheduled pipelined processor can run at and determine what your critical path is (specify each module/entity/component that this path goes through).

50.26mhz