CprE 381: Computer Organization and Assembly-Level Programming

Project Part 2 Report

Team Members: James Minardi

Ella Knott

Project Teams Group #: 306

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

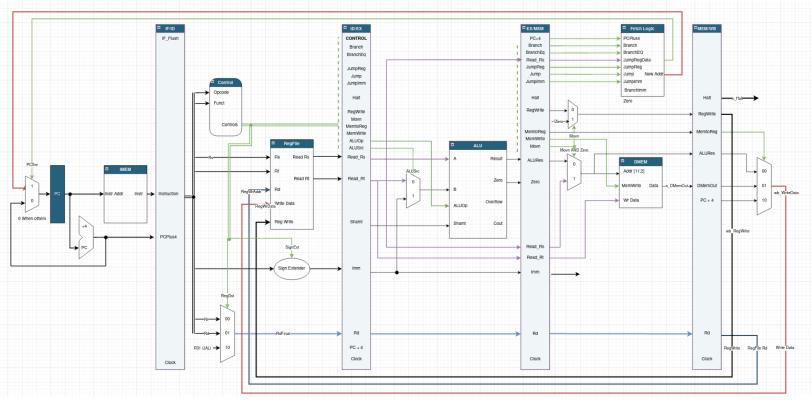
Waveforms and timings for both hardware and software can be found in the proj/directory.

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

RST WE Instruction PCPlus4 Clock

ID/EX	EX/MEM	MEM/WB
Branch	PC+4	Halt
Branch BranchEq JumpReg Jump JumpImm Halt RegWrite Movn MemtoReg MemWrite ALUOp ALUSrc Read_Rs Read_Rt Shamt Imm	PC+4 Branch BranchEq Read_Rs JumpReg Jump JumpImm Halt RegWrite MemtoReg MemWrite Movn ALURes Zero Read_Rs Read_Rt	Halt RegWrite MemtoReg ALURes DMemOut PC + 4 Rd
Rd PC + 4 Clock	Imm Rd Clock	Clock

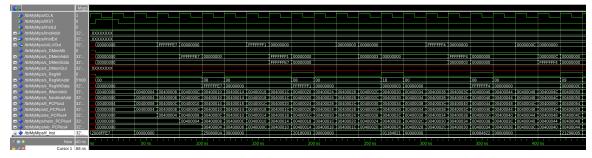
[1.b.ii] high-level schematic drawing of the interconnection between components.



[1.c.i] include an annotated waveform in your writeup and provide a short discussion of result correctness. (The software test that uses all instructions is called software_base.s in mips directory)

This program uses every instruction supported by the software scheduled pipeline processor. It does not conflict and cause hazards as demonstrated by the comparison with the MARS simulation pictured below.

```
bash-4.2$ ./381_tf.sh test --asm-file ./proj/mips/software base.s
Using VDI Python Environment
Testina
WARNING: Software tree location not set or invalid, using $MGC HOME=/usr/local/mentor/calibre
Running: ./proj/mips/software_base.s
Starting Mars Simulation for : ./proj/mips/software base.s ...Success
starting compilation...
Successfully compiled proj/src/MIPS types.vhd
Successfully compiled internal/testpy/tb.vhd
Successfully compiled proj/src/MIPS types.vhd
Successfully compiled proj/src/Multiplexers/mux32t1_N.vhd
Successfully compiled proj/src/Multiplexers/mux2t1.vhd
Successfully compiled proj/src/Multiplexers/mux2t1 N.vhd
Successfully compiled proj/src/Multiplexers/mux32t1.vhd
Successfully compiled proj/src/SubComponents/full_adder.vhd
Successfully compiled proj/src/SubComponents/decoder_N.vhd
Successfully compiled proj/src/SubComponents/invg.vhd
Successfully compiled proj/src/SubComponents/full adder N.vhd
Successfully compiled proj/src/SubComponents/dffg N.vhd
Successfully compiled proj/src/SubComponents/ones_compg.vhd
Successfully compiled proj/src/SubComponents/xorg2.vhd
Successfully compiled proj/src/SubComponents/PC dffg.vhd
Successfully compiled proj/src/SubComponents/barrel_shifter.vhd
Successfully compiled proj/src/SubComponents/andg2.vhd
Successfully compiled proj/src/SubComponents/extender.vhd
Successfully compiled proj/src/SubComponents/dffg.vhd
Successfully compiled proj/src/SubComponents/org2.vhd
Successfully compiled proj/src/SubComponents/add sub.vhd
Successfully compiled proj/src/TopLevel/regfile.vhd
Successfully compiled proj/src/TopLevel/IDEX_reg.vhd
Successfully compiled proj/src/TopLevel/MIPS Processor.vhd
Successfully compiled proj/src/TopLevel/control.vhd
Successfully compiled proj/src/TopLevel/mem.vhd
Successfully compiled proj/src/TopLevel/fetch.vhd
Successfully compiled proj/src/TopLevel/ALU.vhd
Successfully compiled proj/src/TopLevel/EXMEM reg.vhd
Successfully compiled proj/src/TopLevel/PC reg.vhd
Successfully compiled proj/src/TopLevel/MEMWB reg.vhd
Successfully compiled proj/src/TopLevel/IFID_reg.vhd
Starting VHDL Simulation...Success
Victory!! Your processes matches MARS expected output with no mismatches!!
Instructions: 111
                        Cycles: 120
                                        CPI: 1.081
bash-4.2$ modelsim
Reading pref.tcl
```



	XXXXXXXXX XXXXXXXXXX XXXXXXXXXXX	00011 FFFFFD0 00000014 00000000	10000000	3 200000000	901 00000000 77FFFF	\$ € 8 10000000 0FFFFFD 1
↑ milhyklipsis DiMemiW 0 ↑ milhyklipsis DiMemiAd 32" 1 ↑ milhyklipsis DiMemiAd 32" 1 ↑ milhyklipsis DiMemiData 32" 1 ↑ milhyklipsis DiMemiAda 50" 1 ↑ milhyklipsis RegiWh 2dat 32" 1 ↑ milhyklipsis RegiWh 2dat 32" 1 ↑ milhyklipsis RegiWh 32" 32" 1 ↑ milhyklipsis NemiAdat 32" 1 ↑ milhyklipsis NemiAdat 32" 1	0000000C 00000000 FFFFFFF4 000000000 XXXXXXXXXX 00 09 00 00000000 0000000C 000000000 0040004C 00400050 00400054 00- 0040040C 00400050 00400055 0040055	00000011 FFFFFFD 0000000 0000000C FFFFFFF 0000000 0000000C FFFFFFF 0000000 00000011 FFFFFFF 0000000 00000011 FFFFFFF 00000000 00000001 0000000 0000000 0000000 0000000 0000000 0000000 0000000 0000000	4 00000000	00000003 00000000 00000011 00000000 00000011 00000000 00000011 000000000 00000001 0000000000	0000001 0000000 0000011 0000000 000 000	7FFFFE8 00000000
□ ↑ m.hh/yhlgotf, P.CPlusd □ ↑ m.hh/yhlgotf, P.CPlusd □ ↑ m.hh/yhlgotfe, P.Clusd □ ↑ m.hh/yhlgot	0040004C 100400050 100400054 100- 0040004R 10040004C 100400050 1004	100058 0040005C 00400060 0040006 100054 00400058 0040005C 0040006 100050 00400054 00400058 0040005 10004C 00400050 00400054 0040005	4 00400068 0040006C 0040007 0 00400064 00400068 0040006 C 00400060 00400064 0040008 8 0040005C 00400060 0040008 01495823 00000000	4 (00400078) [00400076 (00400080 (004000) 00400074) [00400078 (00400076 (00400076 (00400) 6 (00400070) [00400074 (00400078 (00400) 4 (00400060 [00400076 (00400077 (00400) 4 (00400060 [00400076 (00400070 (00400) 00400080 [00400066 (00400070 (00400) 6 (00400080 [00400066 (00400070 (00400) 6 (00400080 [00400066 (00400070 (00400) 6 (00400080 [0040066 (00400070 (00400) 6 (00400080 [0040066 (00400070 (00400) 6 (00400080 [0040066 (00400070 (00400) 6 (00400080 [0040066 (00400) 6 (0040080 [0040080 (00400) 6 (0040080 [00400) 6 (0040080 [00400) 6 (0040080 [00400) 6 (0040080 [00400) 6 (004000) 6 (00400080 [00400) 6 (00400) 6 (00400)	080 [00400084 [00400088 [0040008C [004000	×
X Find:	XXXXXXXXX					
	00000000		00 0000000 00 00 FF 00000000 38 004000 004000C0 0040000	4 0000000 00000 4 0000000 00000 00 00	0000 15550000 00000000 5014 100000000 155550000 00000000 0014 10000000 00000000 00 0C 00 00 00 00000014 0000000 05555000 000000 0004 0000000 0040000 0040000 004000 0040 0000000 0040000 0040000 004000	\$5555555 00000000 15555555 00000000 155555000 0000000 10000000 100000000 100000000
★ hb/MyMips/ex_PCPlus4 32'	004000A4 004000A8 004000 00 004000A0 004000A4 004000A8 00 0040009C 004000A0 004000A4 00	400080 00400084 00400088 0040001 4000AC 00400080 00400084 0040001 4000A8 004000AC 00400080 0040001 4000A4 004000A8 004000AC 0040001 4000A0 004000A4 004000AB 004000	36 004000C0 004000C4 0040000 38 004000 004000C0 0040000 34 00400088 004000BC 0040000	28 004000 004000D0 004000D4 00400 24 004000C8 004000 004000D0 00400 00 004000C4 004000C8 004000 00400	0008 004000 004000E0 004000E4 004000 0004 004000B8 004000 004000E0 004000 0000 004000D4 004000D8 004000 004000 000 004000D0 004000D4 004000B8 004000 000.8 004000 004000D0 004000D4 004000	E8 004000 004000F0 004000F1 004000F8 E4 004000E8 004000EC 004000F0 004000F4 E0 004000E4 004000E8 004000EC 004000F0 004000E0 004000E4 004000E8 004000E6 008 004000 004000E0 004000E4 004000E8
■- 4 Atb/MyMips/oALUOut 32'	XXXXXXXX XXXXXXXX Q0000000 \$5555555	0000000	AA. 00000000	0000000 0000000	00000075 00000000	
Mb/MyMipsis_RegWr 0	00000000 XXXXXXXX 01 00 55555555 00000000 1004000F0 1004000F4 1004000F8 1004000F0 1004000F4 004000F8 1004000F0 1004000F8 004000F8	004000FC 00400100 00400104 00400 00400100 00400104 00400108 00400	0108 0040010C 00400110 00400 010C 00400110 00400114 00400	0114 00400118 0040011C 00400120 004 0114 00400118 0040011C 00400120 004 0118 0040011C 00400120 00400124 004	00124 00400128 0040012C 00400130 0040 00128 0040012C 00400130 00400134 0040	0134 00400138 0040013C 00400140 00400144 00134 00400138 0040013C 00400140 00400144 00138 0040013C 00400140 00400144 00400148
□	1004000EC 004000F0 1004000F4 1 1004000E8 004000 1004000F0	004000F8 004000FC 00400100 00400 004000F4 004000F8 004000FC 00400	0104 00400108 0040010C 00400 0100 00400104 00400108 00400 00FC 00400100 00400104 00400 01CE7826 00000	0110 00400114 00400118 0040011C 004 010C 00400110 00400114 00400118 004 0108 0040010C 00400110 00400114 004	00120 00400124 00400128 0040012C 0040 0011C 00400120 00400124 00400128 0040 00118 0040011C 00400120 00400124 0040	0134 (00400128 (00400136 (00400140 (00400144) 0130 (00400144 (0400138 (00400136 (0040014) 013C (00400144 (0400138 (00400136 (0040013C) 013C (00400130 (00400134 (00400134 (0040013C) 013E (0040012C (00400130 (00400134 (0040013C) 0000
Miggs	XXXXXXXX XXXXXXXX XXXXXXXX XXXXXXXX XXXX	000000 11000000 000003 10000000	10000000	1600000 0000000 1000000 00000	0000 00000010 00000000 0000000	10 3000000 00000000 1 30000000 1 30000000 1
□ √ mohlyflepsis DMemData 32" √ mohlyflepsis DMemData 32" √ mohlyflepsis PergiVi 0 □ √ mohlyflepsis Pergi	XXXXXXXXX OF 00 000 00000002 000000000 004000140 00400144 00400148 00 00400144 00400144 00400148 00 00400144 00400144 00400146 00 00400145 00400144 00400148 00	0014C 00400150 00400154 0040015 00150 00400154 00400158 0040015 0014C 00400150 00400154 0040015	8 0040015C 00400160 0040016 C 00400160 00400164 0040016 8 0040015C 00400160 0040016	000000003 0	0174 00400178 0040017C 00400180 004001 0178 0040017C 00400180 00400184 004001 0174 00400178 0040017C 00400180 004001	00000003 00000000
Now 40 ns Cursor 1 73 ns	00400138 0040013C 00400140 100 00400138 0040013C 0040013C 00 0040013E 0040013C 00 0040013E 0040013C 00 1650 ns	00040 00000018 0000018 000001 00044 0000018 0000016 000001 000401 0000014 000018 004001 0000014 00000000 17000s 1750 n	IAF280000 I0000000	C 00400160 00400164 00400168 00400 8 0040015C 00400160 00400169 00400 9 18780000 2000008 20040010 00000 1850000 2000008 20040010 00000	190 (0400) (0400) (1400	80 (06400)82 (06400)83 (06400)84 (06400)85 (06400)85 (16
Magal Maga	XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX		00000010 00000000 00000010 00000000 00000000		, 0000000	10000008 10000000 10000008 10000000 10000008 10000000 10000008 10000000 10000008 10000000
	00 000000000 10000000 004001. 00400165 (00400170 10044 004001. 00400165 (00400174 1004 004001. 00400170 100400174 1004 004001. 00400165 (00400170 1004 004001. 00400165 (00400170 1004	08 09 0A 0000 0000003 00000008 00000010174 00400178 00400176 00400180 00400170 00400180 00400180 10178 00400175 00400180 00400184 10178 00400175 00400180 00400180 10178 00400174 00400174 00400175 00400176 00400	00 08 00000000 0000001 00000184 00400188 00400186 00400184 00400180 00400180 00400188 00400180 00400190 00400184 00400180 00400190 00400184 00400188 00400180	00 1,0000000 1,00000190 (00400194 (00400198 1004001 1,00400190 (00400194 (00400198 1004001 1,00400194 (00400198 (00400196 (004001 1,00400190 (00400194 (00400198 (004001 1,00400180 (00400194 (00400194 (004001 1,00400180 (00400194 (004001)44 (00400188 (004001)4	IBC 004001C0 004001C4 004001C8 0040018 IBC 004001C0 004001C4 004001C8 0040018 ICO 004001C4 004001C8 004001 004001.4 IGC 004001C0 004001C4 004001C8 004001.	× 00000000 000000000000000000000000000
De Abhlyklipsimem_PCPlus4 De Abhlyklipsimem_PCPlus4 De Abhlyklipsimem_PCPlus4 De Abhlyklipsimem_PCPlus4 De Belling De Bel	004001 00400160 00400164 0044 8F2800 20090008 200A0010 0009 1850 ns	0168 0040016C 00400170 00400174 00000 21280008 00000000 1900 ns 1950 ns	116A0006	00400184 00400188 0040018C 004001 2 00000000 081000 2050 ns	190 00400194 00400198 0040019C 0040010 187 00000000 1569000 2100 ns 2150 ns	20 004001C4 004001C8 004001 004001.40 33 00000000 0C100078 22200 ns 2220 ns



[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.

The waveform below is annotated with comments on what the program is doing based on the instruction in the ID stage. Given an array of 1,3,2,5,4,0,7,6 it behaves as expected and swaps the elements needed while leaving alone the pairs that are in order. For example the first iteration of (1,3) is left alone and proceeds to swap the next pair of elements which is (3,2).

Because of where our fetch logic is contained (in the mem stage), when we use nops, it is always in pairs of 3 because solving data hazards requires 3 and control flow hazards requires 3 as well. However, there are a few instances in this program where 3 nops were not required.

In this example, we have two load word instructions back to back without any nops in between them. This is because while they access the same register to use for the memory address, they don't actually modify that value. Where they put the new data are in separate registers meaning they don't create a hazard. Code shown below:

```
lw $s1, 0($a0)  # loads first element in $s1
lw $s2, 4($a0)  # loads second element in $s2
```

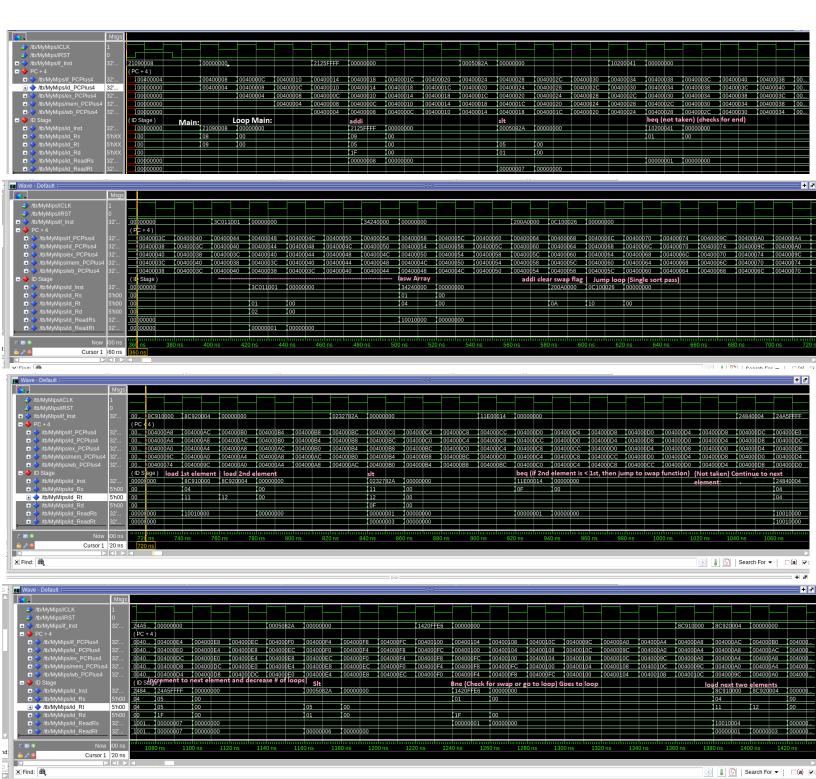
Because 3 instructions are always required after every control flow instruction, there are no locations in this program where the max number of nops were not needed for a control flow hazard.

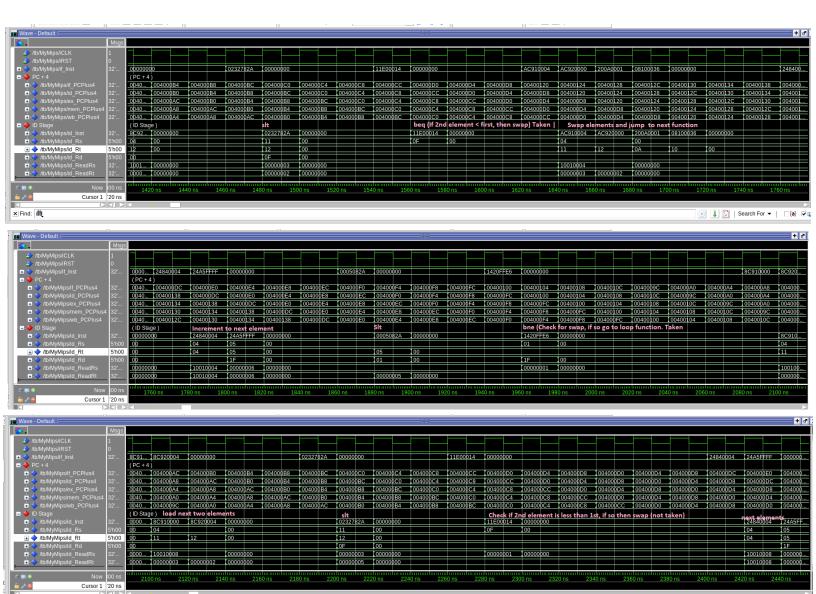
Here are a few more examples of areas where nops were not needed. Firstly, the store words, while accessing \$a0 for the memory address, because of the immediate value they are not storing into the same location thus avoiding a data hazard. Similarly, the addi instruction can sit freely without any nops inbetween the jump instruction and store words because it does not access any modifiable values. And below the jump instruction, three nops are required because the new PC address is calculated in the mem stage.

```
swap:
    sw $s1, 4($a0)  # puts value of [i+1] in s1
    sw $s2, 0($a0)  # puts value of [i] in s2
    addi $t2, $0, 1  # notes a swap

    j next  # jump to next
    nop
    nop
    nop
    nop
end:
```

```
bash-4.2$ ./381_tf.sh test --asm-file ./proj/mips/software_bubblesort.s
Using VDI Python Environment
Testing
WARNING: Software tree location not set or invalid, using $MGC HOME=/usr/local/mentor/calibre
Running: ./proj/mips/software bubblesort.s
Starting Mars Simulation for : ./proj/mips/software bubblesort.s ...Success
starting compilation...
Successfully compiled proj/src/MIPS types.vhd
Successfully compiled internal/testpy/tb.vhd
Successfully compiled proj/src/MIPS_types.vhd
Successfully compiled proj/src/Multiplexers/mux32t1 N.vhd
Successfully compiled proj/src/Multiplexers/mux2t1.vhd
Successfully compiled proj/src/Multiplexers/mux2t1 N.vhd
Successfully compiled proj/src/Multiplexers/mux32t1.vhd
Successfully compiled proj/src/SubComponents/full_adder.vhd
Successfully compiled proj/src/SubComponents/decoder N.vhd
Successfully compiled proj/src/SubComponents/invg.vhd
Successfully compiled proj/src/SubComponents/full_adder_N.vhd
Successfully compiled proj/src/SubComponents/dffg_N.vhd
Successfully compiled proj/src/SubComponents/ones_compg.vhd
Successfully compiled proj/src/SubComponents/xorg2.vhd
Successfully compiled proj/src/SubComponents/PC dffg.vhd
Successfully compiled proj/src/SubComponents/barrel shifter.vhd
Successfully compiled proj/src/SubComponents/andg2.vhd
Successfully compiled proj/src/SubComponents/extender.vhd
Successfully compiled proj/src/SubComponents/dffg.vhd
Successfully compiled proj/src/SubComponents/org2.vhd
Successfully compiled proj/src/SubComponents/add sub.vhd
Successfully compiled proj/src/TopLevel/regfile.vhd
Successfully compiled proj/src/TopLevel/IDEX reg.vhd
Successfully compiled proj/src/TopLevel/MIPS Processor.vhd
Successfully compiled proj/src/TopLevel/control.vhd
Successfully compiled proj/src/TopLevel/mem.vhd
Successfully compiled proj/src/TopLevel/fetch.vhd
Successfully compiled proj/src/TopLevel/ALU.vhd
Successfully compiled proj/src/TopLevel/EXMEM_reg.vhd
Successfully compiled proj/src/TopLevel/PC reg.vhd
Successfully compiled proj/src/TopLevel/MEMWB reg.vhd
Successfully compiled proj/src/TopLevel/IFID_reg.vhd
Starting VHDL Simulation...Success
Victory!! Your processes matches MARS expected output with no mismatches!!
Instructions: 936
                       Cycles: 1213
                                       CPI: 1.296
bash-4.2$
```





[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).

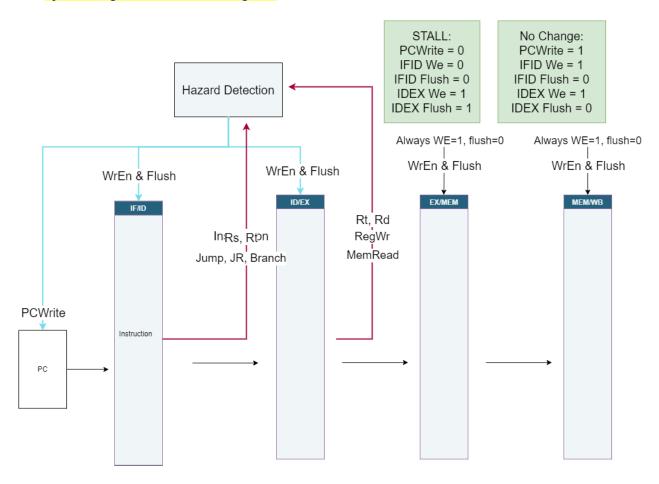
FMax: 55.82mhz

Critical path:

Execute stage:

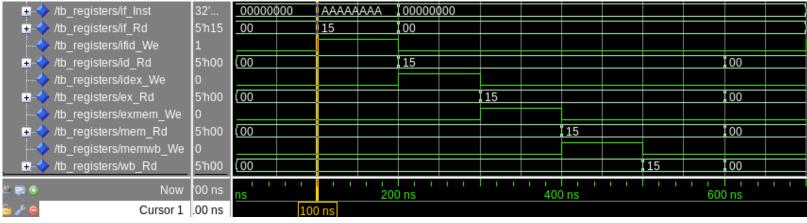
- 1. IDEX register ALUSrc output
- 2. ALU Src mux (Selects ReatRt vs Immediate)
- 3. ALU B input -> add_sub component
- 4. nbit full adder (32 full adders)
- 5. ALU Operation mux
- 6. ALU Zero mux
- 7. ALU Output Zero
- 8. EXMEM register Zero input

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

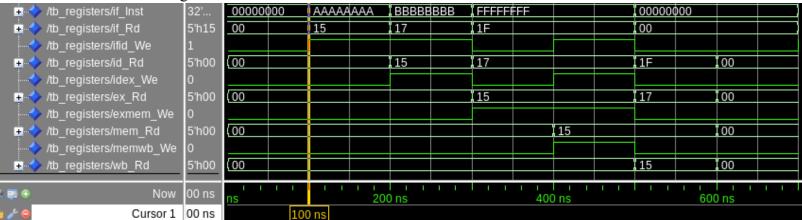


[2.a.iii] Create a testbench that instantiates all four of the registers in a single design. Show that values that are stored in the initial IF/ID register are available as expected four cycles later, and that new values can be inserted into the pipeline every single cycle. Most importantly, this testbench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

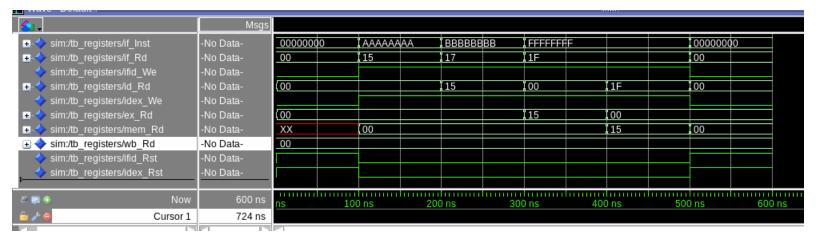
Shows normal instruction propagation through the pipeline:



Demonstrates stalling of pipeline while instruction "BBBBBBBBB" (Rd=17) is in the decode stage.



Demonstrates flushing of the IFID stage for instruction "BBBBBBB". As you can see, it is replaced by a NOP in the following clock cycle as it moves into the ID stage.



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

<mark>).</mark>	Droduco: Cha	nge registers or memory		
		cess registers or memory		
INSTR	PRODUCE	SIGNALS	CONSUME	SIGNALs
add	TRUE	RsData, RtData, ALUResult, RegWrData	TRUE	Rs, Rt
addi	TRUE	RsData, RtData, ALUResult, RegWrData	TRUE	Rs, Imm
addiu	TRUE	RsData, RtData, ALUResult, RegWrData	TRUE	Rs, Imm
addu	TRUE	RsData, RtData, ALUResult, RegWrData	TRUE	Rs, Rt
and	TRUE	RsData, RtData, ALUResult, RegWrData	TRUE	Rs, Rt
andi	TRUE	RsData, RtData, ALUResult, RegWrData	TRUE	Rs, Imm
lui	TRUE	RsData, ALUResult, RegWrData	TRUE	Rs, Imm
lw	TRUE	ALUResult, RegWrData	TRUE	Rs, Imm
nor	TRUE	RsData, RtData, ALUResult, RegWrData	TRUE	Rs, Rt
xor	TRUE	RsData, RtData, ALUResult, RegWrData	TRUE	Rs, Rt
xori	TRUE	RsData, RtData, ALUResult, RegWrData	TRUE	Rs, Imm
or	TRUE	RsData, RtData, ALUResult, RegWrData	TRUE	Rs, Rt
ori	TRUE	RsData, RtData, ALUResult, RegWrData	TRUE	Rs, Imm
slt	TRUE	RsData, RtData, ALUResult, RegWrData	TRUE	Rs, Rt
slti	TRUE	RsData, RtData, ALUResult, RegWrData	TRUE	Rs, Imm
sll	TRUE	RsData, RtData, ALUResult, RegWrData	TRUE	Rs, Rt, Shamt
srl	TRUE	RsData, RtData, ALUResult, RegWrData	TRUE	Rs, Rt, Shamt
sra	TRUE	RsData, RtData, ALUResult, RegWrData	TRUE	Rs, Rt, Shamt
SW	TRUE	RsData, RtData, ALUResult	TRUE	Rt
sub	TRUE	RsData, RtData, ALUResult, RegWrData	TRUE	Rs, Rt
subu	TRUE	RsData, RtData, ALUResult, RegWrData	TRUE	Rs, Rt
beq	FALSE		TRUE	Rs, Rt
bne	FALSE		TRUE	Rs, Rt
j	FALSE		FALSE	
jal	TRUE	RsData, RtData, ALUResult, RegWrData	FALSE	
jr	FALSE		FALSE	
movn	TRUE	RsData, RtData, ALUResult, RegWrData	TRUE	Rs, Rt
repl.qb	TRUE	RsData, RtData, ALUResult, RegWrData	TRUE	lmm
halt	FALSE		FALSE	
Total: 28				

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

See 2.b.i

[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.

These four hazards are what make up our hazard detection and forwarding unit, minus control flow hazards and extra checks. For example, we always check for the WE of the

independent instruction and check if the Rd value of the independent instruction = 0. This handles everything from SW, LW, and normal Rtype instruction hazards.

Dependencies
i-type to i-type
i-type to r-type
r-type to r-type
r-type to r-type to r-type
to r-type to r-type
lw to sw

Hazard	Can FWD?	FWD Path:	Require Stall?	
Load-use	FALSE		TRUE	Stall ID once
Regfile bypass	TRUE	WB Rd -> ID Rs/Rt	FALSE	
mem_Rd = ex_Rs/Rt	TRUE	MEM Rd -> EX Rs/Rt	FALSE	
ex_Rd = id_Rs/Rt & JR/Branch	TRUE	Did not do this FWD path	TRUE	Stall ID
mem_Rd = id_Rs/Rt & JR/Branch	TRUE	Did not do this FWD path	TRUE	Stall ID

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

See 2.d for full diagram that includes the registers, hazard unit, and forwarding unit with names for the signals required.

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

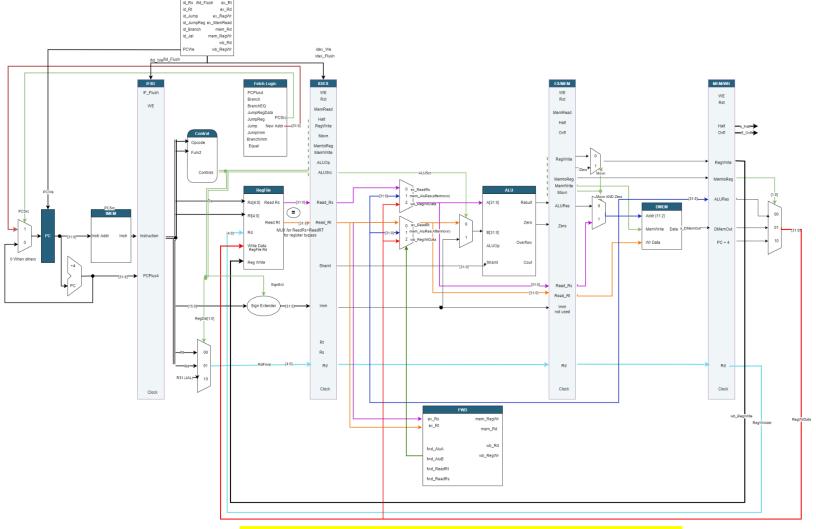
Because we handle control flow data hazards in fwding, no stalling is required for control hazards. All we have to do is flush the fetched instruction and leave the PC write enabled, thus placing a NOP in between the control flow instruction and the following instruction. This flush is required to allow for the fetch logic to fetch the next instruction in the ID stage.

Control Haz	ard Avoidance	
INSTR	Stage of PC Update	Stall/Flush what stages
beq	ID	Flush IFID, continue ID as normal
bne	ID	Flush IFID, continue ID as normal
j	ID	Flush IFID, continue ID as normal
jal	ID	Flush IFID, continue ID as normal
jr	ID	Flush IFID, continue ID as normal
		No Stalling required, handles in FWDing

[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.

See 2.c.i

[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.

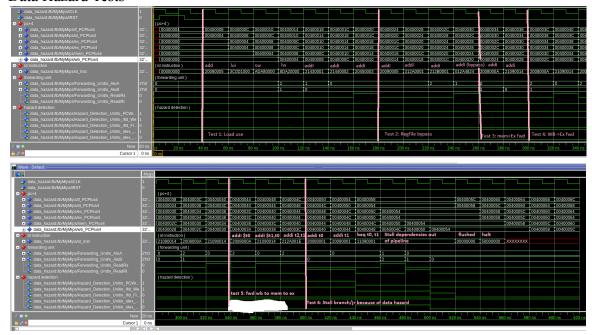


[2.e - i, ii, and iii] In your writeup, show the QuestaSim output for each of the following tests, and provide a discussion of result correctness. It may be helpful to also annotate the waveforms directly.

Below are our waveforms for our data and control flow hazard tests. They are each annotated with the test number and the instructions within that test that correspond to our spreadsheet. Our tests are confirmed to work using the toolflow, matching the MIPS output with no errors. On top of that, we can confirm our functionality by looking at the forwarding unit and hazard unit output signals which match our expectations.

Although not pictured below, our processor also contains no mismatches with the MIPs dump for all other test programs including our original bubble sort, base tests, branch test, fibonacci, and grendel.

Data Hazard Tests



Control Flow Hazard Tests



[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.

Our data hazard assembly program properly tests each of our forwarding and hazard unit edge cases in unit and combination. Really, there aren't that many cases. We must forward from WB to EX, MEM to EX, WB to ID, and stall control flow data hazards, all of which are tested in this program. On the right hand size, there is a coverage list containing an explanation of what our test programs actually test.

Data hazard tests		Control hazard tests		Coverage	
Test	Instructions	Test	Instructions	Test	Included:
load use	lw \$1 , 0(\$2)	jump	j jump	FWD wb_WriteData to id_ReadRs/Rt (Regfile bypass)	TRUE
combination	andi \$4, \$1 , 5 (Stall)	unit	addi \$t0, \$0, 5	FWD mem_Rd to ex_Rs/Rt (EX Hazard)	TRUE
	andi \$6, \$1 , 7 (Fwd)		jump: addi \$t1, \$0, 1	FWD wb_Rd to ex_Rs/Rt (Mem Hazard)	TRUE
	ori \$8, \$1 , 9 (ok)			STALL Load-Use	TRUE
				STALL Branch/Jump for dependency in EX Stage	TRUE
Regfile Bypass	andi \$1, \$0, 5	data flow & control hazard	addi \$t5, \$0, 5	STALL Branch/Jump for dependency in MEM Stage	TRUE
combination	andi \$2, \$1, 1 (6) (fwd)	combination	j jump	STALL Branch/Jump for dependency in WB Stage	TRUE
	andi \$3, \$1, 1 (6) (fwd)		addi \$t0, \$0, 3 (flush)	FLUSH IF if control flow is in ID (Control flow hazard)	TRUE
	and \$1, \$1, \$2 (rf bypass)		jump: addi \$t1, \$5, 1		
fwd mem -> ex	addi \$t0, \$0, 10	beq	addi \$t0, \$0, 1		
unit test	addi \$t1, \$t0, 20 (fwd)	combination	addi \$t1, \$0, 1		
			beq \$t0, \$t1, branch (stall for both)		
fwd wb -> ex	addi \$t0, \$0, 10		addi \$t0, \$0, 3 (flush)		
unit test	nop		branch: add \$t2, \$t1, \$t0 (no fwd)		
	addi \$t1, \$t0, 20 (fwd)		no fwd needed b/c branch stalls ^		
		branch + Jump	addi \$t0, \$0, 1		
fwd wb-> ex & mem	addi \$t0, \$0, 10	combination	addi \$t1, \$0, 1		
unit test	addi \$t1, \$t0, 20 (fwd)		j jump		
			addi \$t0, \$0, 3 (flush)		
			jump:		
fwd wb -> mem -> ex	addi \$t0, \$0, 10		beq \$t0, \$t1, branch(stall for \$t1)		
combination	addi \$t1, \$t0, 20 (fwd)		addi \$t0, \$0, 3 (flush)		
	addi \$t2, \$t1, 30 (fwd)	-1	branch: add \$t2, \$t1, \$t0 (ok)		
stall branch/jr b/c of data hazard	addi #+0 #0 1				
data nazard	addi \$t0, \$0, 1				
	addi \$t1, \$0, 1				
	beq \$t0, \$t1, branch (stall) branch: halt				
•	pranch: hait				

[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.

See 2.e.i for spreadsheet. Similar to 2.e.i, our control hazard tests focus on our forwarding and hazard unit edge cases in unit and combination. We test a simple flushing of the next instruction after a control flow instruction, and build upon that to complete multiple control flow instructions each flushing their own instructions and forwarding data hazards on top of that.

[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).

FMax: 48.96mhz Critical Path:

EXECUTE Stage:

- 1. IDEX ALUSrc output
- 2. ALU Src mux (Selects ReadRt vs Immediate)
- 3. ALU B input
- 4. Add_sub component
 - a. nbit full adder
 - i. 32 full adders
- 5. ALU Operation mux (Select add sub)
- 6. ALU Zero mux (Selects zero based on result)
- 7. ALU Output Zero
- 8. EXMEM Register Zero input

It is essentially the same as our software processor, but at the beginning it says this about the Dmem:

 $3.726-0.263-uTco\ mem:DMem\ |\ altsyncram:ram_rtl_0\ |\ altsyncram_eg81:auto_generated\ |\ ram_block1a0\sim porta_we_reg$

6.575 2.849 FR CELL DMem | ram_rtl_0 | auto_generated | ram_block1a0 | portadataout[0]

We are not sure what this means.

Waveforms and timings for both hardware and software can be found in the proj/directory.