EE2003 Assignment 3

Niranjan A. Kartha, EE21B095

1 Question 1 (Problem 2)

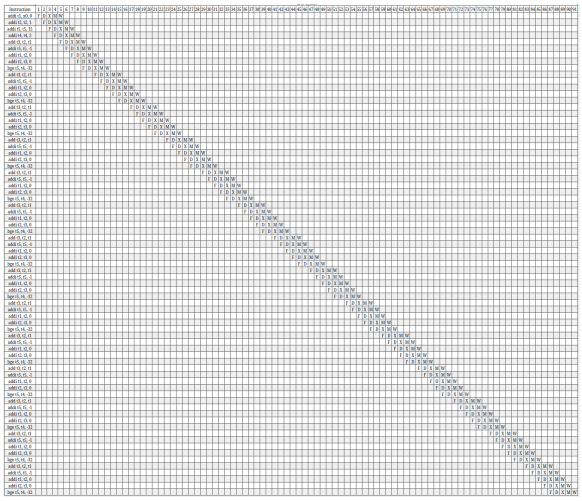
With both architectures, I have attached code written in a way that minimizes and tries to maximize the number of data hazards.
In the minimum-hazard case, I have been able to eliminate all data hazards, and so forwarding does not make a difference in the number of clock cycles taken.
All the below cases output the 15^{th} Fibonacci number as 610.

1.1 Flushing delay slots

1.1.1 With hazard-free code

```
1     mv t1, x0
2     li t2, 1 # answer will be present here
3     li t5, 15 # load arg here
4     li t4, 2
5  FIBLOOP:
6     add t3, t2, t1
7     addi t5, t5, -1
8     mv t1, t2
9     mv t2, t3
10    bge t5, t4, FIBLOOP
```

The above code has no hazards, so the pipeline does not stall, with or without forwarding.



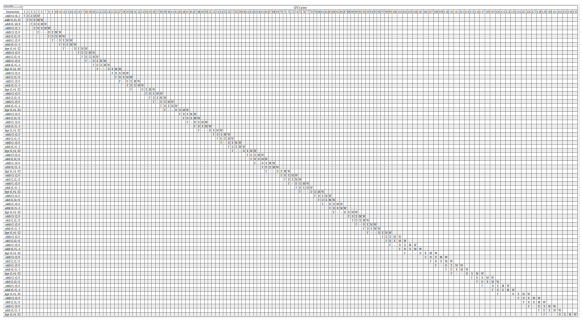
Instruction	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	2
addi t1, x0, 0	F	D	X	M	W															
addi t2, t2, 1		F	D	X	M	W														
addi t5, t5, 15			F	D	X	M	W													
addi t4, t4, 2				F	D	X	M	W												
add t3, t2, t1					F	D	X	M	W											
addi t5, t5, -1						F	D	X	M	W										
addi t1, t2, 0							F	D	X	M	W									
addi t2, t3, 0								F	D	X	M	W								
bge t5, t4, -32									F	D	X	M	W							
add t3, t2, t1											F	D	X	M	W					
addi t5, t5, -1												F	D	X	M	W				
addi t1, t2, 0													F	D	X	M	W			
addi t2, t3, 0														F	D	X	M	W		
bge t5, t4, -32															F	D	X	M	W	

Figure 1: Execution completes in 91 cycles

1.1.2 Code with hazards

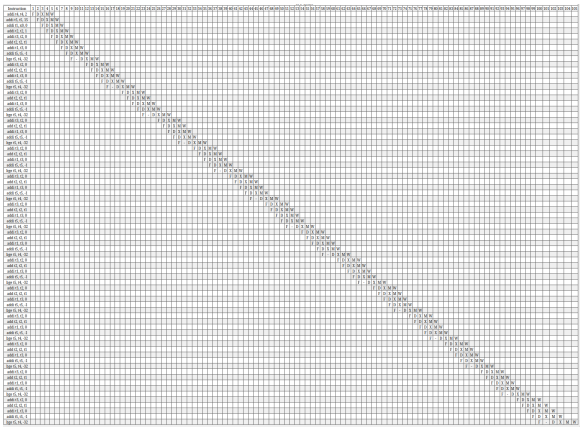
```
1     li t4, 2
2     li t5, 15 # load arg here
3     mv t1, x0
4     li t2, 1 # answer will be present here
5  FIBLOOP:
6     mv t3, t2
7     add t2, t2, t1
8     mv t1, t3
9     addi t5, t5, -1
10    bge t5, t4, FIBLOOP
```

The above code has been written to maximize hazards. Line 6 has a RAW hazard with line 4. Line 8 has a RAW hazard with line 6. Line 10 has a RAW hazard with line 9.



Instruction	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
addi t4, t4, 2	F	D	X	M	W													
addi t5, t5, 15		F	D	X	M	W												
addi t1, x0, 0			F	D	X	M	W											
addi t2, t2, 1				F	D	X	M	W										
addi t3, t2, 0					F	-	-	D	X	M	W							
add t2, t2, t1								F	D	X	M	W						
addi t1, t3, 0									F	-	D	X	M	W				
addi t5, t5, -1											F	D	X	M	W			
bge t5, t4, -32												F	-	-	D	X	M	W

Figure 2: Without forwarding, takes 135 cycles



Instruction	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
addi t4, t4, 2	F	D	X	M	W																	
addi t5, t5, 15		F	D	X	M	W																
addi t1, x0, 0			F	D	X	M	W															
addi t2, t2, 1				F	D	X	M	W														
addi t3, t2, 0					F	D	X	M	W													
add t2, t2, t1						F	D	X	M	W												
addi t1, t3, 0							F	D	X	M	W											
addi t5, t5, -1								F	D	X	M	W										
bge t5, t4, -32									F	-	D	X	M	W								
addi t3, t2, 0											F	D	X	M	W							
add t2, t2, t1												F	D	X	M	W						
addi t1, t3, 0													F	D	X	M	W					
addi t5, t5, -1														F	D	X	M	W				
bge t5, t4, -32															F	-	D	X	M	W		

Figure 3: With forwarding, takes 105 cycles

Forwarding does not completely fix the RAW hazard right before the branch.

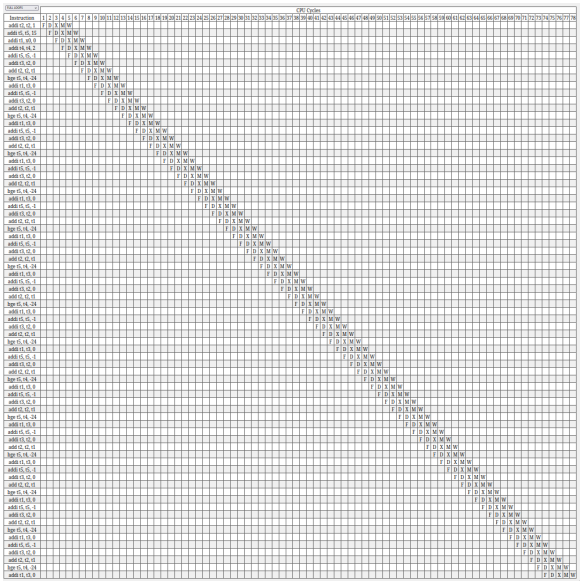
1.2 Executing delay slots

If we execute an instruction in the branch delay slot, we can do more useful work while the branch address is being calculated.

1.2.1 With hazard-free code

```
1      li t2, 1 # answer will be present here
2      li t5, 15 # load arg here
3      mv t1, x0
4      li t4, 2
5  FIBLOOP:
6      addi t5, t5, -1
7      mv t3, t2
8      add t2, t2, t1
9      bge t5, t4, FIBLOOP
10     mv t1, t3
```

The above code has no hazards, so the pipeline does not stall, with or without forwarding. Line 11 also gets executed every time line 10 is encountered.



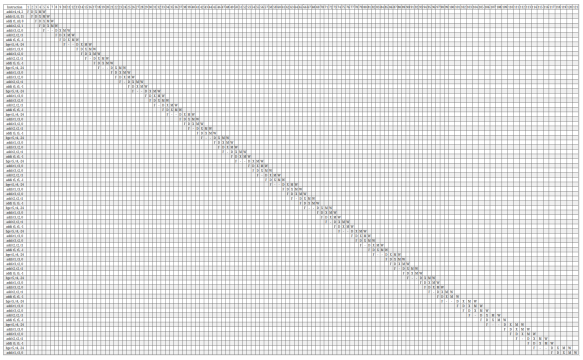
Instruction	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
addi t2, t2, 1	F	D	X	M	W																	
addi t5, t5, 15		F	D	X	M	W																
addi t1, x0, 0			F	D	X	M	W															
addi t4, t4, 2				F	D	X	M	W														
addi t5, t5, -1					F	D	X	M	W													
addi t3, t2, 0						F	D	X	M	W												
add t2, t2, t1							F	D	X	M	W											
bge t5, t4, -24								F	D	X	M	W										
addi t1, t3, 0									F	D	X	M	W									
addi t5, t5, -1										F	D	X	M	W								
addi t3, t2, 0											F	D	X	M	W							
add t2, t2, t1												F	D	X	M	W						
bge t5, t4, -24													F	D	X	M	W					
addi t1, t3, 0														F	D	X	M	W				
addi t5, t5, -1															F	D	X	M	W			
addi t3, t2, 0																F	D	X	M	W		
add t2, t2, t1																	F	D	X	M	W	
bge t5, t4, -24																		F	D	X	M	W

Figure 4: Execution completes in 78 cycles

1.2.2 Code with hazards

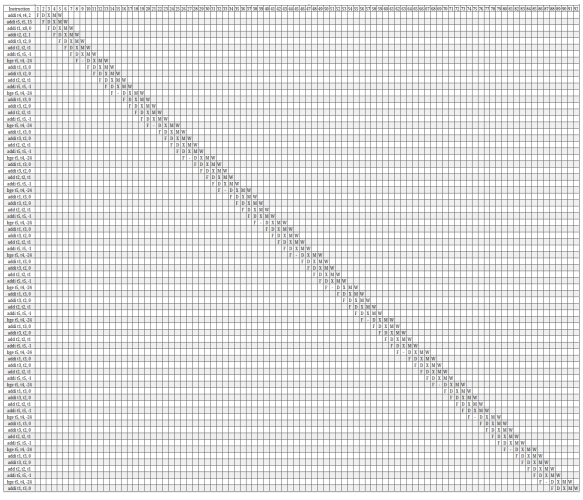
```
1     li t4, 2
2     li t5, 15 # load arg here
3     mv t1, x0
4     li t2, 1 # answer will be present here
5  FIBLOOP:
6     mv t3, t2
7     add t2, t2, t1
8     addi t5, t5, -1
9     bge t5, t4, FIBLOOP
10    mv t1, t3
```

The above code has been written to maximize hazards. Line 6 and 7 have a RAW hazard with line 4. Line 9 has a RAW hazard with line 8. Line 6 has a RAW hazard with line 10.



Instruction	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	2
addi t4, t4, 2	F	D	X	M	W																				
addi t5, t5, 15		F	D	X	M	W																			
addi t1, x0, 0			F	D	X	M	W																		
addi t2, t2, 1				F	D	X	M	W																	
addi t3, t2, 0					F	-	-	D	X	M	W														
add t2, t2, t1								F	D	X	M	W													
addi t5, t5, -1									F	D	X	M	W												
bge t5, t4, -24										F	-	-	D	X	M	W									
addi t1, t3, 0													F	D	X	M	W								
addi t3, t2, 0														F	D	X	M	W							
add t2, t2, t1															F	-	D	X	M	W					
addi t5, t5, -1																	F	D	X	M	W				
bge t5, t4, -24																		F	-	-	D	X	M	W	

Figure 5: Without forwarding, takes 121 cycles



Instruction	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
addi t4, t4, 2	F	D	X	M	W														
addi t5, t5, 15		F	D	X	M	W													
addi t1, x0, 0			F	D	X	M	W												
addi t2, t2, 1				F	D	X	M	W											
addi t3, t2, 0					F	D	X	M	W										
add t2, t2, t1						F	D	X	M	W									
addi t5, t5, -1							F	D	X	M	W								
bge t5, t4, -24								F	-	D	X	M	W						
addi t1, t3, 0									F	D	X	M	W						
addi t3, t2, 0										F	D	X	M	W					
add t2, t2, t1											F	D	X	M	W				
addi t5, t5, -1												F	D	X	M	W			
bge t5, t4, -24													F	-	D	X	M	W	

Figure 6: With forwarding, takes 92 cycles

Forwarding does not completely fix the RAW hazard right before the branch.

1.3 Conclusion

Even with the same instructions being executed, the order in which they are specified is very important for the performance of the program. Instructions need to be written in a way to minimize hazards.
In the above experiments, even with advanced architectural features such as forwarding, code written with care to avoid hazards is always able to outperform code with many hazards.

2 Question 2 (Dataset 2)

For the `matrixmulti.riscv` dataset, these were the results:

Strategy	CPI	Control Hazards	Data Hazards
AT	1.4395	29662	105128
NT	1.4139	40678	110957
BTFNT	1.3893	29258	110841
BPB	1.4129	29482	108238

Observations

Matrix multiplication involves a lot of looping, and so the results are as follows:

- AT performs poorly because it mispredicts almost every time a loop performs an iteration.
- NT performs well, since it fits backward branches in loops well.
- BPB performs well, since it is able to learn how branches behave.
- BTFNT performs best, since this is the natural way that branches tend to behave in normal use-cases. It does not need to waste branch mispredictions in trying to learn how a branch will behave.