Problem 1 - Define Pipeline Hazard. State and define the three types of pipeline hazards.

Pipeline Hazard - A situation in which the next instruction cannot complete execution one clock cycle after completion of the present instruction.

Data Hazard - An instruction cannot be completed because the needed data, to be generated by another instruction in the pipeline, is not available

Structural Hazard - Two instructions cannot execute due to a resource conflict.

Control Hazard - The next instruction to execute cannot be determined

Problem 2 - Assuming N computations, K stages in an ideal simple linear pipeline and tsc as clock period, derive the equation for speedup compared to a non-pipeline and its throughput, where the clock cycle for single cycle is tsc and pipelined is tp.

$$\text{Speedup} = \frac{\text{Single Cycle}}{\text{Pipelined}} = \frac{N*tsc}{(k + N - 1)*tp}$$

Problem 3 - Describe how Data Forwarding is performed in the RISC-V datapath. A full-credit answer will include discussion of what data is forwarded back to the ALU, how those data elements are forwarded back to the ALU, and how the specific data that goes to the ALU is selected.

Data Forwarding takes the addresses of registers at each stage, and uses the Data Forwarding Unit to determine which version of the register value will be used. The Hazard Detection Unit is used to determine if one of the pipelined register hazard conditions exists in the datapath, and inserts a stall if one exists. Multiplexers at both inputs of the ALU are used to select the results

EX/MEM.RegisterRd = ID/EX.RegisterRs1

EX/MEM.RegisterRd = ID/EX.RegisterRs2

MEM/WB.RegisterRd = ID/EX.RegisterRs1

MEM/WB.RegisterRd = ID/EX.RegisterRs2

Problem 4 - Describe the process for implementing exceptions and interrupts. A full-credit answer will include a description of exceptions and interrupts on the RISC-V datapath, where the information regarding the exception is stored, how RISC-V prioritizes exceptions, and how the datapath is "cleared" once an exception is found.

Upon receiving an exception, the processor must save the address of the “offending” instruction in the EPC transfer control to the operating system. The Status Register includes a register that holds a field indicating the reason for the exception. Vectored Interrupts contain the address to which control is transferred is determined by the cause of the exception. Exceptions are prioritized. HW determines which exception is to be serviced first by sorting so that the earlier instr. is serviced 1st. The EPC captures the address of the interrupted instructions, and the Service Register records all possible exceptions in a clock cycle

Problem 5 - Consider the following RISC-V code sequence

```
1     add x7, x18, x20
2     lw  x21, 20(x7)
3     add x5, x21, x22
4     sub x23, x5, x21
5     or  x23, x21, x22
6     sw  x23, 20(x5)
```

a) Assuming no forwarding, identify all pipeline hazards between pairs of instructions

x7 between 1 and 2
x21 between 2 and 3
x21 between 2 and 4
x5 between 3 and 4
x23 between 4 and 6
x23 between 5 and 6

b) Assuming no forwarding, insert stalls as needed to overcome these hazards. How many clock cycles are needed to finish these instructions?

```
      1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8
1     F D E M W
2       S S F D E M W
3             S S F D E M W
4                   S S F D E M W
5                         F D E M W
6                           S S F D E M W
```

18 cycles

c) Assuming we use forwarding, insert stalls as needed to overcome these hazards. How many clock cycles are needed to finish executing these instructions?

```
      1 2 3 4 5 6 7 8 9 0 1
1     F D E M W
2       S F D E M W
3           F D E M W
4             F D E M W
5               F D E M W
6                 F D E M W
```

11 cycles

Problem 6 - 125 points - Consider the following RISC-V code sequence

```
Unset
lw  x18, 40(x10)

add x22, x10, x18

lw  x19, 44(x10)

or  x23, x22, x18

lw  x24, 48(x10)

addi x6, x24, -7
```

a) Assuming we use forwarding, insert stalls as needed to overcome these hazards. How many clock cycles are needed to finish executing these instructions?

```
                        1 2 3 4 5 6 7 8 9 0 1 2
lw   x18, 40(x10)       F D E M W
add  x22, x10, x18        S F D E M W
lw   x19, 44(x10)             F D E M W
or   x23, x22, x18              F D E M W
lw   x24, 48(x10)                 F D E M W
addi x6, x24, -7                    S F D E M W
```

12 cycles

b) Rearrange the instructions to minimize the need for stalls. How many clock cycles are needed to finish executing these instructions?

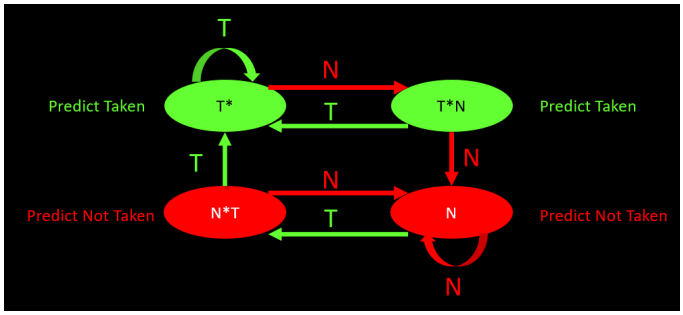
```
                        1 2 3 4 5 6 7 8 9 0
lw   x18, 40(x10)       F D E M W
lw   x19, 44(x10)         F D E M W
lw   x24, 48(x10)           F D E M W
add  x22, x10, x18            F D E M W
or   x23, x22, x18              F D E M W
addi x6, x24, -7                  F D E M W
```

10 cycles

Problem 7 - Draw the 2-bit branch prediction FSM. Then, given the following branch addresses and branches, show the final state of a k=3 correlating prediction model

Address	T/N
10001101	T
10001000	N
10001101	T
10001111	N
10001101	N
10001000	T
10001000	T
10001101	T
10001111	T

Solution:



	LHT Values		LPT Values
000	000 -> 000 -> 100 -> 110	000	00 -> 01 -> 00 -> 00 -> 01 -> 11
001	000	001	00
010	000	010	00
011	000	011	00 -> 01
100	000	100	00 -> 01 ->11
101	000 -> 100 -> 110 -> 011 -> 101	101	00
110	000	110	00 -> 00
111	000 -> 000 -> 100	111	00

Problem 8 - Assume 35% of instructions change the flow of a program:

28% of instructions are branches

- 25% of branches are taken
- Mispredicted branches result in a 3 cycle stall (wait for address to be calculated)

7% of instructions are loads

- 45% of the time, the next instruction uses the loaded value

What is the impact on performance assuming:

- There is an additional 1 cycle stall for load hazard
- Always predict branch not taken
- 5-stage datapath

Solution:

```
(k + N – 1)*tp
(5 + N – 1)*tp
(4 + N)*tp
(4 + (1-0.28-0.07)N + (0.28*0.25)(4N) + 0.28*0.75N + 0.07*.55N + 0.07*0.45*(2N) )*tp
(4 + 0.68*N + 0.28*N + 0.21*N + 0.0385*N + 0.063N )*tp
(4 + 1.2415*N )*tp

Limit as N->infinity is 1.2415
```

9. Consider the following pseudo-C code where a loop compares data elements from two arrays. If the element in the first array is less than the element in the second array, the value of the second array is set to the value of the first array; otherwise it is set to 0. Register names are used as variable names to tie to the RISC-V code in the pipe traces (below) that implements this loop.

```
while (x18 < x20) {      //assume x20 was initialized to 1000 earlier
      x21 = x19[x18];
      x22 = x23[x18 + 4];
      if (x21 < x22)
        x23[x18 + 4] = x21;
      else
        x23[x18 + 4] = 0;
      x18 = x18 + 1;
}
x20 = x20 + x18
```

Complete the pipe trace. You should assume:

1. At the first instance of loop `bge` instruction is **not taken** and **predicted correctly**, and that correct predictions incur no delay (thus, assume the `slli` is fetched in cycle)
2. At the `if` statement, , `x21` is less than `x22` and the second `bge` instruction is **predicted incorrectly**
3. At the second instance of the `while` statement, `x18` is equal to `x19` and the corresponding `bge` instruction is **predicted incorrectly**
4. All `jump` instructions are predicted correctly (again, assume no delay/stall cycle needed)
5. Hazard detection and data forwarding in this RISC-V datapath
6. the outcome of a branch is definitively known after it finishes the EX stage, and there is no additional flush latency - in other words, on a branch misprediction the first instruction on the correct path will be fetched 1 cycle after the branch's Execute stage

Recall that RISC-V registers have **write, then read** capabilities In all parts of this problem, you may assume that a register can be written and read from the register file in the same clock cycle. You only need to fill this pipe trace for 1 iteration (i.e., the instructions below). Finally, note there may be more columns in the pipe trace than needed.

		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
loop:	bge x18, x20, out																								
	slli x10, x18, 2																								
	add x24, x10, x19																								
	lw x21, 0(x24)																								
	add x24, x10, x23																								
	lw x22, 4(x24)																								
	bge x21, x22, else																								
	sw x21, 4(x24)																								
	j end																								
else:	sw x0, 4(x24)																								
end:	addi x18, x18, 1																								
	j loop																								
out:	add x20, x20,x18																								

Solution

① Branch not taken, predicted correctly. No stalls

② Data hazard for x22 following lw. 1 stall

③ x21 < x22, so branch not taken. Predicted incorrectly

④ Jump back to end

⑤ Jump to loop (cycle 14 means bge occurs at cycle 15)

⑥ Branch to add I goest at the instruction bge at loop (so at cycle 18 since bge E occurs at 17