Q1. Assume I02I architecture with full bypassing and with in-order fetch, decode; out-of-order issues, execute, writeback, and in-order commit. The pipeline has 4 functional units: ALU (1 cycle X0), Loads and stores (2 cycle, S0 and S1) and multiply (4 cycles, Y0, Y1, Y2, Y3). **(10 Marks)**

Instruction No.	
0	add R1,R2, R3
1	sub R4, R5, R6
2	mul R6, R0, R1
3	ld R0, R8
4	mul R7, R6, R9
5	add R7, R1, R2
6	st R7, R8

For the instruction above,

a) Draw the pipeline diagram. **(9 Marks)**

b) Draw state of the scoreboard when instruction 2 is in issue stage of the pipeline. **(1 Mark)**

Ans:

		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
0	add R1, R2, R3	F	D	I	X0	W	C												
1	sub R4, R5, R6		F	D	I	X0	W	C											
2	mul R6, R0, R1			F	D	I	Y0	Y1	Y2	Y3	W	C							
3	ld R0, R8				F	D	I	S0	S2	W	r		C						
4	mul R7, R6, R9					F	D	i		I	Y0	Y1	Y2	Y3	W	C			
5	add R7, R1, R2						F	D	i		I	X0	W	r			C		
6	st R7, R8							F	D	i			I	S0	S1	W	r	C	

(0.5 + 0.5 +1 + 2 + 1 +2+2)

b) State of scoreboard when instruction 2 is in issue stage:

R4 is pending in functional unit ‘X0’ and is marked as having ‘0’ more cycles until writeback. The rest of the registers are not pending. **(1 Mark)**

Q2. A processor has the following specifications: Word size is 16-bits; No. of blocks in L1 cache is 8; Cache block size is 1 word; Cache is 2 way set associative and uses LRU replacement policy. Main memory is word addressable. A single entry victim cache contains the recently evicted data from cache. For L1 cache, the following sequence of memory access happens. Note that all addresses are in decimal.

200, 199, 198, 204, 194, 206, 209, 212, 201, 203

List the memory addresses that are available in L1 and victim after all memory access operations are performed. Assume that all cache lines are invalid when the sequence of operations starts. **(10 Marks)**

Solution:

L1 Cache:

~~200~~ 212
204
209
201
~~198~~ 206
194
199
203

Victim Cache:

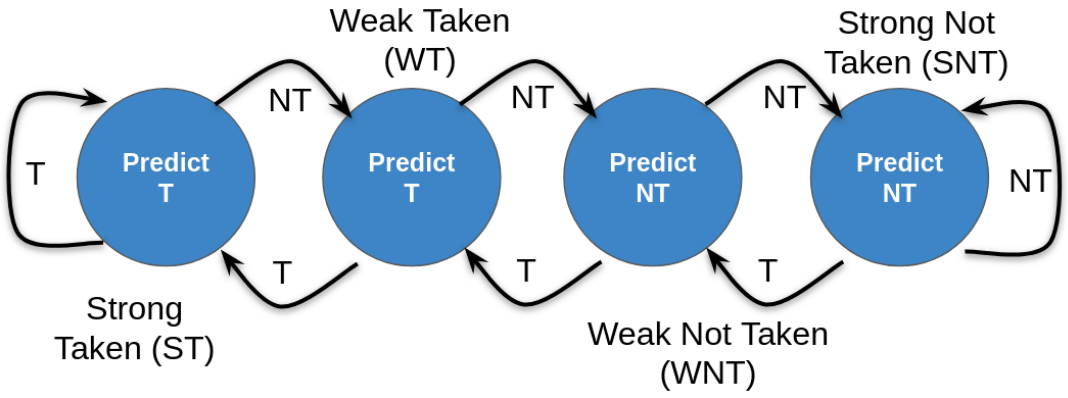
~~198~~ 200

(1*10 = 10 Marks)

Q3. Consider the following two branch-predictors:

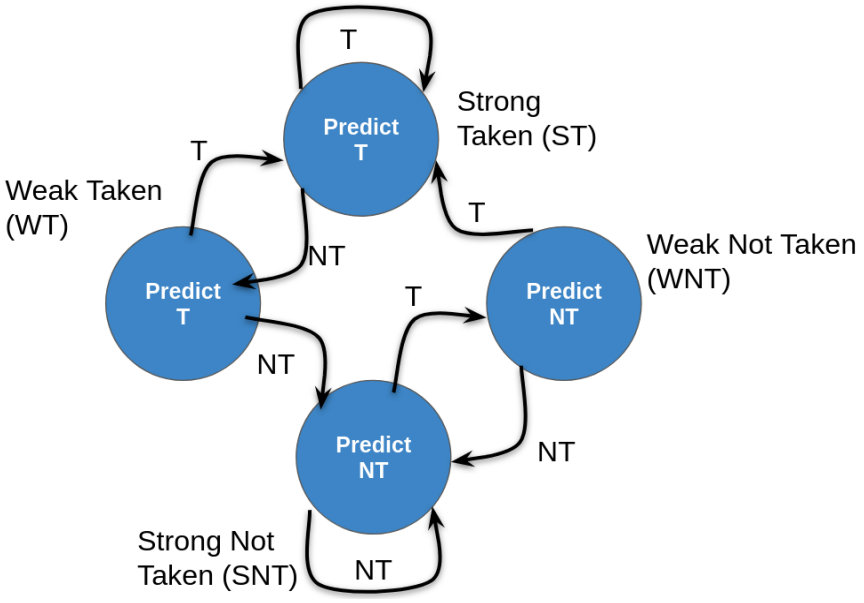
2-Bit Saturating Branch Predictor FSM:

Assume the initial state to be STRONG TAKEN (ST)



2-Bit Non-Saturating Branch Predictor FSM:

Assume the initial state to be STRONG TAKEN (ST)



For the following assembly code, answer the questions that follow:

```
      addi s1, 0, 0    # s1 = sum = 0
      addi s0, 0, 0    # s0 = i = 0
      addi t0, 0, 4    # t0 = 4
for:
      bge s0, t0, done      # i >= 4?
      add s1, s1, s0        # sum = sum + i
      addi s0, s0, 1        # i = i + 1
      j for                 # repeat loop
done:
```

<u>Note</u>: bge s0, t0, loop → branch to "loop" if s0>=t0

(a) Fill in the following table for both types of branch predictors when the above code is executed once. Report the number of mis-predictions for each branch predictor.

[10 Marks]

Iteration Number	Previous State of Branch Predictor (Prediction)	Actual branch taken/not taken	New State of Branch Predictor	Mispredict?
1				

(b) Redo part (a) when the above code is run once again, provided the final states of the branch predictor(s) are retained after the code was run previously. Report the number of mis-predictions for each branch predictor. **[10 Marks]**

Solution:

Two-Bit Branch Predictor (non-Saturating)						Two-Bit Branch Predictor (Saturating)				
Iteration Number	Previous State of Branch Predictor (Prediction)	Actual branch taken/not taken	New State of Branch Predictor	Mispredict?		Iteration Number	Previous State of Branch Predictor (Prediction)	Actual branch taken/not taken	New State of Branch Predictor	Mispredict?
1st RUN:										
1	ST	NT	WT	Y		1	ST	NT	WT	Y
2	WT	NT	SNT	Y		2	WT	NT	WNT	Y
3	SNT	NT	SNT			3	WNT	NT	SNT	
4	SNT	NT	SNT			4	WNT	NT	SNT	
5	SNT	T	WNT	Y		5	SNT	T	WNT	Y
2nd RUN:										
1	WNT	NT	SNT			1	WNT	NT	SNT	
2	SNT	NT	SNT			2	SNT	NT	SNT	
3	SNT	NT	SNT			3	SNT	NT	SNT	
4	SNT	NT	SNT			4	SNT	NT	SNT	
5	SNT	T	WNT	Y		5	SNT	T	WNT	Y

Note: Evaluation will be done on a case to case basis in case jump (j) instruction has been considered for branch prediction FSM. In such cases, the use of FSMs should be logically feasible.