1. (25 pts) Data Hazards

A. (12 pts) Consider the following execution sequence:

1. add $3, $1, $2
2. lw $1, 0($4)
3. and $5, $3, $4
4. and $8, $1, $2
5. or $1, $3, $8
6. sw $1, 4($4)
7. lw $2, 4($4)

F D E M W

F D E M W.

F D E M W.

F D E M W.

F D E M W.

F D E M W.

F D E M W.

Find the all data hazards and fill the following table (you may leave some entries blank or add more rows to accommodate your answer).

Instruction #1	Instruction #2	Register	Hazard Type
1	2	$1	WAR
1	3	$3	RAW
2	4	$1	RAW.
2	5	$1	WAW.
1	5	$1	WAR
4	5	$8	RAW
4	5	$1	WAR.
4	7	$2	WRR.

$6 $1 RAW

B. (4 pts) Suppose the code sequence in A is executed in a 5-stage pipelined in-order execution machine. How many nops do we need to insert into the pipeline without a forwarding unit (your answer should be in the form of A+B+C+...+...=Z, where A, B, C, ... are the nops inserted in the order of the instruction sequence)? Explain.

0+1+0+2+2+0=5

C. (4 pts) How many <u>nops</u> do we need to insert to the pipeline with a forwarding unit (**NO** mem to mem forwarding and the same answer format with B)? Explain.

```
add $3 $1 $2     F D E M W
lw $1 0($4)        F D E M W.
and $5 $3 $4         F D E M W
and $8 $1 $2           F D E M W
or $1 $3 $8              F D E M W.
sw $1 4($4)                F D E M W
lw $2 4($4),                 F D E M W.
```

0+0+0+0+0+0=0

D. (5 pts) Suppose one student says that "Instruction 7 might not load the value stored by instruction 6 since they refer to the same memory location". In an in-order 5-stage pipelined execution machine, is the student right? Why or why not?

No,

At the time of memory stage of inst⑦, the value in 4($4) of memory is already updated.

2. (25 pts) Branch Prediction and Stalls

A. (5 pts) Explain the functionalities of the following hardware components.

BTB--- in the fetch stage, store the branch target address.

BHT--- is in the fetch stage, and contains whether the branch was taken or not recently.

B. (5 pts) Consider a branch that has the following outcome pattern (**T** for taken, **N** for not taken).

N N T T T N N T T T

How many branches are predicted correctly with a static (0-bit) always-taken branch predictor for this branch outcome pattern.

6/10.

C. (5 pts) Using the same sequence from B. How many branches are predicted correctly with a dynamic 1-bit predictor where the initial state is Taken (T) for this branch outcome pattern?

T X N √ N X T √ T √ T X N √ N X T √ T √

N N T T T N N T T T

6/10.

D. (5 pts) How many branches are predicted correctly with a dynamic, saturating counter 2-bit predictor for the branch outcome pattern in B? Suppose the four states are strong not taken (**SN**), weak not taken (**wn**), weak taken (**wt**), and strong taken (**ST**). Assume that the initial prediction state is wt (weak taken).

X n √ N X n X t √ T X t X n X t √ T √

√ N T T T N N T T T

4/10.

E. (5 pts) What are the fundamental differences between pipeline stall and pipeline flush? Provide examples to highlight what instructions will cause stall and what instructions will cause flush, and why.

pipeline stall, stall one instruction, wait for one or more cycles. lw $1 0($4)
add $3 $2 $1. there will be a stall

pipeline flush; will remove all the instructions.
for example in branch instruction, if will predict wrong, we shoul remove all the instructions after that branch instruction.

3. (10 pts) VLIW

Consider following MIPS instruction sequence:

```
lw $t0,0($s1)
add $t0,$t0,$s2
sw $t0,0($s1)
addi $s1,$s1,4
lw $t1,0($s1)
add $t1,$t1,$s2
sw $t1,0($s1)
addi $s1,$s1, 4
```

You are required to fill the bundles in the table below for a VLIW machine to have the minimum number of execution cycles. The **LEFT** part of the bundle (second column in the table) can be only an ALU or branch instruction, whereas the **RIGHT** part (third column in the table) can be only a load or store instruction. Note that, if there is a load-use hazard, there will be at least one cycle delay between the load instruction and the use instruction. You are allowed to reorder independent instructions and change the offset of addressing. Explain each instruction-to-slot

mapping decision in sufficient you make detail (i.e., why you have decided so, couldn't instruction be scheduled in an earlier slot (cycle)?, etc).

Cycle	ALU/Branch	Load/Store
1	add	lw $t0 0($s1)
2	addi $s1 $s1 4	
3	add. $t0 $t0 $s2	lw $t1 0($s1)
4	addi $s1 $s1 4	
5	add $t1 $t1 $s2	sw $t0 -8($s1).
6		sw $t1 -4($s1)
7		
8		
9		

4. (40 pts) Value Prediction

In class, we discussed the concept of data dependences and showed that data dependences can prevent both compiler and hardware from changing the order of execution order of instructions and from executing instructions in parallel. While anti- and output-dependences can be eliminated via renaming, flow-dependences are much more challenging (as they indicate actual data transfer, as opposed to simple name reuse).

Value prediction is a method to handle data dependences. In value prediction, hardware predicts the value to be returned by a load instruction, before the instruction accesses the cache/memory. The execution continues with the predicted value. In the background, the cache/memory is accessed and, if the value read (actual/correct value) is different the one predicted, the pipeline is flushed and the execution re-starts from the instruction that comes after the load, using the actual value.

One method of value prediction for an instruction is "last-time prediction." The idea is to predict the value to be produced by the load instruction as the value produced by the same instruction the last time the instruction was executed. If the instruction was never executed before, the predictor predicts the value to be 1. Value prediction "accuracy" of an instruction refers to the fraction of times the value of an instruction is correctly predicted out of all times the instruction is executed.

Assume the following piece of code, which has four load instructions in each loop iteration, loads to arrays x, y, z, t:

```
// initialize integer variables c, d, e, f to zeros

// initialize integer arrays x, y, z, t

    for (i=0; i<1000; i++)

      { c += x[i]; d += y[i]; e += z[i]; f += t[i]; }
```

Assume the following state of arrays before the loop starts executing: x consists of all 0's; y consists of alternating 3's and 6's in consecutive elements; z consists of random values between 0 and 2(32) – 1; and t consists of 0, 1, 2, 3, 4, ..., 999.

A. (10 pts) What is the value prediction accuracy of the aforementioned predictor for the four load instructions in the program? load of x[i], load of y[i], load of z[i], and load of t[i].

X 999/1000

Y 0/1000

Z $= 1/2^{32}$

t 0/1000.

B. (10 pts) Can you design a predictor that can achieve higher accuracy for the predictions of x[i], y[i], z[i] and t[i]? If so, explain your design for each load operation separately.

X[i]: everything is same as the exist predictor, but first time predi

Y[i]: each time predict the value that issued two times before.

Z[i]: random predict,

t[i]: each time predict the value last time loaded and plus one.

C. (10 pts) Discuss (and illustrate on the MIPS pipeline) the changes/additions necessary to implement last value prediction.

in the memory stage, we need a table store all the last time loaded value for each load instruction.

D. (5 pts) Explain in general (not for the example code above) why value prediction works. Suppose that you have a value predictor with 100% accuracy. What would this mean from an architecture design viewpoint?

Because there is a high possibility we load a value from the same address due to data locality.

100% accuracy means we don't need to access memory.

D. (5 pts) What are the similarities and differences between value prediction and branch prediction? Explain.

use prediction strategy to eliminate the latency.

branch predictor in fetch stage, avoid of waiting for next istructio

value predictor in memory stage, avoid of memory access latency.