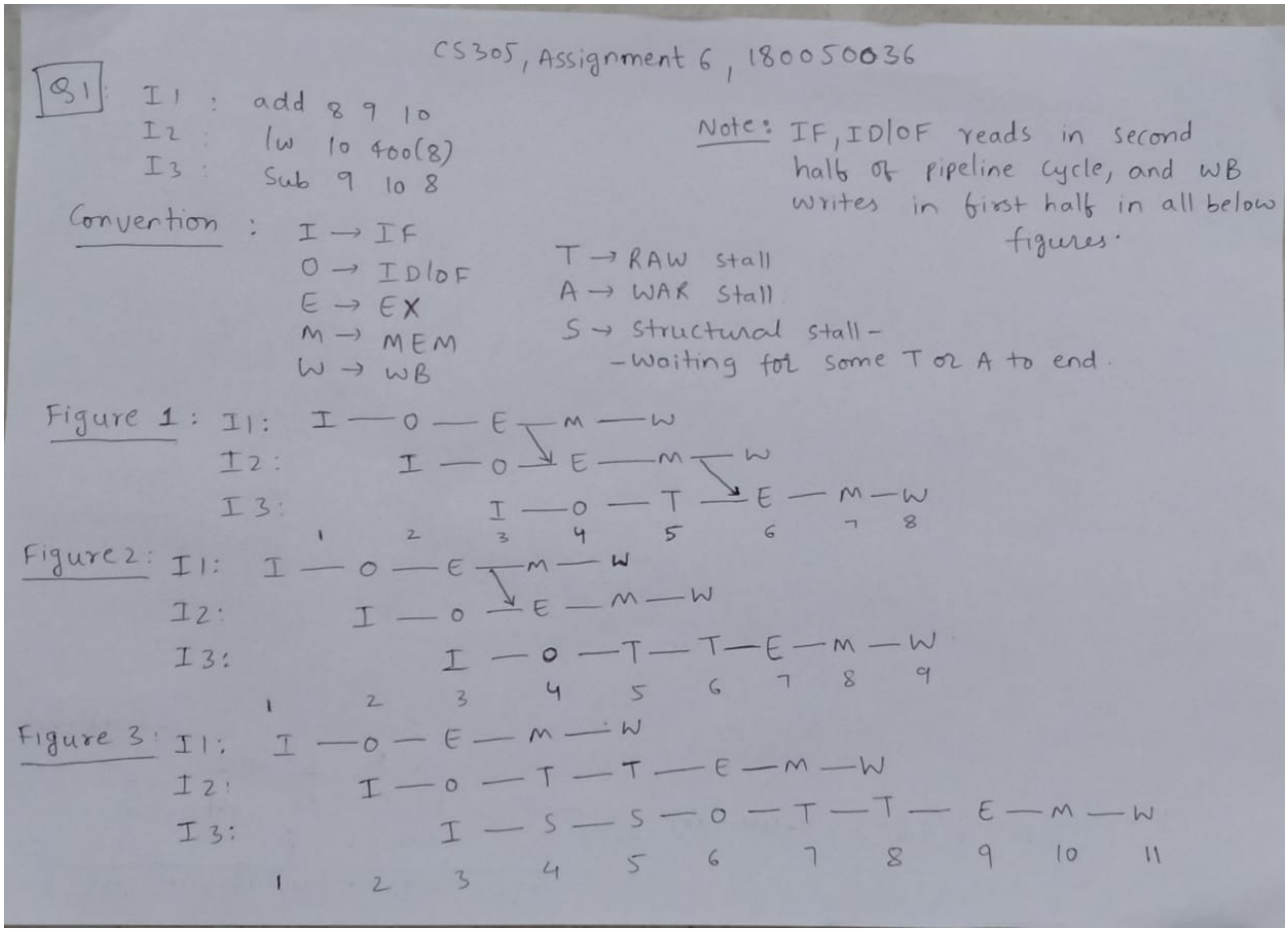
Name: Guttu Sai Abhishek
Roll Number: 180050036

CS 305 – Assignment 6:

Q1:

a) Call the instructions as I1(add one), I2(lw one), I3(sub one) in the order.
True dependences : I1-I2, I2-I3, I1-I3
Anti dependences : I1-I2, I1-I3 (but WAR stalls can't happen/no need in 5 stage MIPS pipeline)(but mentioned here based on the definition)



S above, is equivalent empty box as per winmips convention

b) Please refer figure 1(inside the image) for the answer
c) no forwarding: Please refer figure 3 for the answer
ALU-ALU forwarding: Please refer figure 2 for the answer
full forwarding: Please refer figure 1 for the answer
d) no forwarding: 11*250 pico seconds = 2750 pico seconds
ALU-ALU forwarding: 9*270 pico seconds = 2430 pico seconds
full forwarding: 8*280 pico seconds = 2240 pico seconds
e) average utilisation of pipeline = number of useful cycles/total number of cycles that can happen in the total time used by the program.

P.T.O

no forwarding: (5*3)/(11*5) = 3/11
ALU-ALU forwarding: (5*3)/(9*5) = 3/9
full forwarding: (5*3)/(8*5) = 3/8

f) Instructions can't be reorded as the values in the registers 8,9,10 won't remain same for any reordering. In I1, 8 depends on 9, 10. In I2, 10 depends on 8(whose value is determined by I1). In I3, 9 depends on 10(whose value is determined by I2), 8(whose value is determined by I1). As we can see the dependency of source registers on the previous instructions, reordering can't happen without disturbing the final values in the registers.

Q2:

part	Mispredictions Out of 100	% Mispredictions asymptotically	Other Remarks
a	51	(4/8)*100 = 50%	
b	42	(3/8)*100 = 37.5%	
c	26	(2/8)*100 = 25%	k = 1
d	28	(2/8)*100 = 25%	
e	7	0%	m = 10, k = 1

I have taken initial global history to be a m-bit 0's for all parts(a to e). Where 0 means that branch not taken and 1 mean branch taken. To update the m-bit global history after a branch instruction, I removed the leftmost bit of global history & appended a 1, if branch is taken at that branch instruction, else 0, to the rightmost end.

Lets say the FSM states are 1 to 2^(n) (= N) i.e., from 1 to N. Where 1 is strongly branch Taken, N is strongly branch Not Taken.
All n-bit FSM of the BPB(branch prediction buffer) are initialised to N as said in the question.

The steady states mentioned here from part a to d are the states in which the bgt branch is there in BPB just before the branch instruction is evaluated.
For parts a to c, index(or m_bit+last_k_bits_of_address) in BPB is not mentioned as the index (last_k_bits_of_address_of_bgt_instruction, since m = 0) of BPB is same for all states.
But for part d, since m != 0, indices of BPB for each state(in the steady state pattern) would be different and hence are mentioned{only m-bit global history just before the execution of bgt branch instruction is mentioned since there is a one-to-one mapping(because last_k_bits_of_address is fixed here and there are not too many branch instructions to have same last k bits in their address) from index(or m_bit+last_k_bits_of_address) of BPB to the m-bit global history}.
a) sequence of states in steady states: 3, 4, 3, 2, 1, 1, 1, 2
b) sequence of states in steady states: 3, 4, 3, 2, 1, 1, 1, 2
c) sequence of states in steady states: 2, 2, 2, 1, 1, 1, 1, 1 (k = 1)
d) sequence of states in steady states: 3, 4, 4, 2, 1, 1, 1, 1
and the m-bit history just before branch instruction or an equivalent index in BPB:

01, 01, 01, 11, 11, 11, 11, 11

in above, the n-bit state of 01 index stays same(3) where I meantioned about 11. And n-bit state of 11(2) stays same where I mentioned about 01.
Specifically, a complete steady state cycle looks like this:

m-bit state just before bgt is done (1->01, 3->11):	1 3 3 3 3 3 1 1 1 3 3 3 3 3 1 1 1 3 3 3 3 3 1 1...
t1 :	7 6 5 4 3 2 1 0 7 6 5 4 3 2 1 0 7 6 5 4 3 2 1 0...
t1 > 2:	1 1 1 1 1 0 0 0 1 1 1 1 1 0 0 0 1 1 1 1 1 0 0 0...
bgt branch taken:	1 1 1 1 1 0 0 0 1 1 1 1 1 0 0 0 1 1 1 1 1 0 0 0...
n-bit state of 1 or 01 index in BPB before the bgt happens:	4 3 3 3 3 3 3 4 4 3 3 3 3 3 3 4 4 3 3 3 3 3 3 4...
n-bit state of 3 or 11 index in BPB before the bgt happens:	2 2 1 1 1 1 2 2 2 2 1 1 1 1 2 2 2 2 1 1 1 1 2 2...

n-bit states if index 10 and 00 don't change in the steady state as they don't occur anymore, and are 4, 3 respectively.

e) As said, if we take 1 meaning branch taken and 0 meaning branch not taken in m-bit global history then bnez always contribute 1 to m-bit global history before reaching bgt. So if we see the index of BPB for each of the possible of value of t1 in the branch bgt, those indices have alternate 1's contributed by bnez and the other alternate set have either 1 or 0 contributed by the its previous bgt braches. Now if we can give an unique index to each of possible value of t1, those indices in BPB can train their n-bit FSM to stay in the particular state that predicts correct always.
Possible values of t1: 0, 7, 6, 5, 4, 3, 2, 1, 0, 7, 6, ...
will bgt branch?(t1>2?): 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, ...
Now, we need to associate unique index to each of the possible t1 where index have alternate 1 or 0's whose values are decided if the previous bgt's are branched or not. Now if we take t1 = 3 it's four previous values(the ones for t1 = 7,6,5,4) are brached ie., 1,1,1,1 and same is the case for t1 = 2. Now if we allot only 4 alternate places, for previous bgt's brached or not values, in index, then we can't differentiate between t1 = 3,2. Similar problems arise if we allot only 3 or 2 alternate places for previous bgt's info (brached or not?). But if we allot 5 alternate places for previous bgt's info, such problems won't arrise and all possible values for t1 have unique indices in BPB to train their n-bit values to give correct prediction all the time.
So, 5 bits in index for previous bgt's history and more 5 bits to store the contribution of 1's by bnez ie., m = 10 bit global history will allow us to maintain unique index for each possible t1 value and train their n-bit FSM to predict the correct value. And we need k = 1 since k = 1 mean two states(which is the least no.of states needed) Taken, Not taken and each index, irrespective of what initial state we set the FSM in, can train FSM to be in correct state.
As a side note, m>10 will also do 0% mispredictions asymptotically using the similar reasoning but might more time at the start to reach steady state due to multiple index mapping from a possible value of t1