Assignment 6: Pipelining (Theory Assignment 3)

Anish Deshpande
180100013

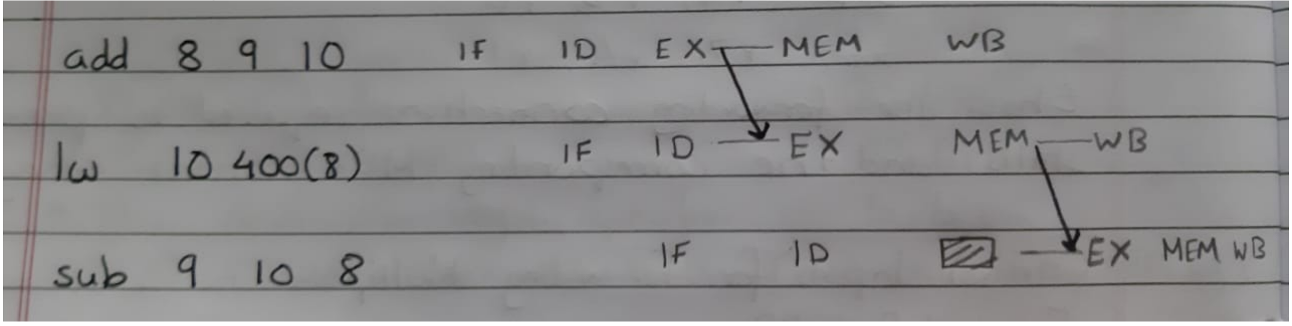
Consider the execution of the following sequence of three instructions on the 5-stage MIPS pipeline.

1)add 8 9 10
2)lw 10 400(8)
3)sub 9 10 8

a) Dependences:

- (Write After Read):
  - Anti-dependence of instruction 2 and 1. (Register 10 might be written to in 2 before it is read from in 1).
  - Anti-dependence of instruction 3 and 1. (Register 9 might be written to in 3 before it is read from in 1).
- (Read After Write):
  - True-dependence of instruction 2 and 1. (Register 8 might be read from in 2 before it is written to in 1).
  - True-dependence of instruction 3 and 2. (Register 10 may be read from in 3 before it is written to in 2)
  - True-dependence of instruction 3 and 1. (Register 8 might be read from in 3 before it is written to in 1).
- no WAW dependence and RAR dependence is not a thing.

b) Forwarding Links:

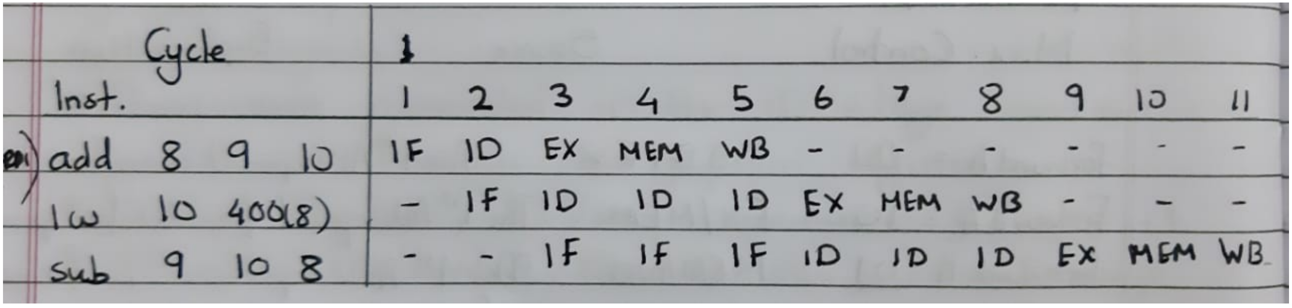


add 8 9 10	IF	ID	EX	MEM	WB			
lw 10 400(8)		IF	ID	EX	MEM	WB		
sub 9 10 8			IF	ID	▨	EX	MEM	WB

Links denoted by the black arrows. The shaded box is a stall, as the sub instruction must wait for the value of register 10 to be supplied before execution.

i) **NO Forwarding**

c)



Cycle Inst.	1	2	3	4	5	6	7	8	9	10	11
add 8 9 10	IF	ID	EX	MEM	WB	-	-	-	-	-	-
lw 10 400(8)	-	IF	ID	ID	ID	EX	MEM	WB	-	-	-
sub 9 10 8	-	-	IF	IF	IF	ID	ID	ID	EX	MEM	WB

d) Pipeline clock cycle time = 250 ps

Total no. Of cycles = 11

Total execution time = clock cycle time * no. of cycles

= 250 * 11 ps

= 2750 picoseconds

e) Average Pipeline Utilisation:

APU= (number of instructions * number of unique stages/instruction)

(number of cycles * num of possible stages that can be executed/cycle)

(Equivalent to filled boxes/total boxes in the cycles vs. Stage table)

APU = 3*5/(11*5) = 3/11

f) No, the instructions cannot be re-ordered to reduce the execution time.

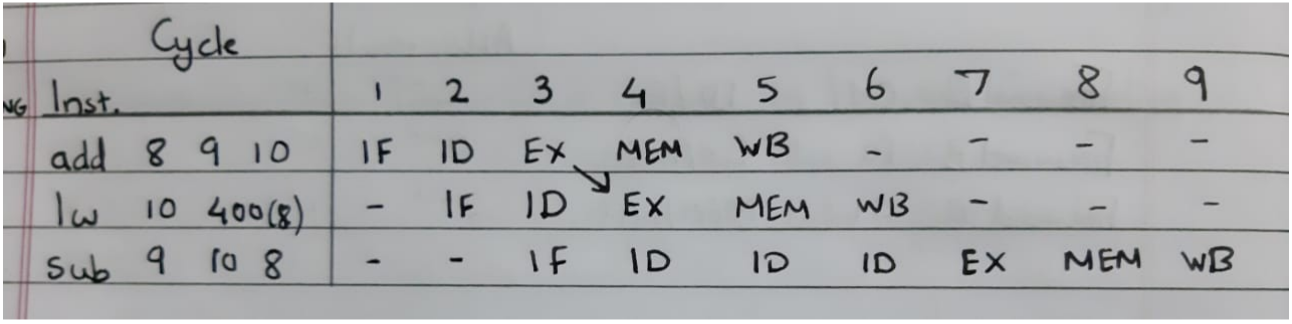
The load word instruction requires the value in register 8 as an operand.

However, register 8 is being written to in the AND instruction. Hence instruction 2 cannot be placed before instruction 1.

Instruction 3 cannot be placed before instruction 2 either. That’s because instruction 3 requires register 10. But register 10 is being updated in the load word instruction.

ii) **ALU-ALU Forwarding**

c)



Cycle											
Inst.			1	2	3	4	5	6	7	8	9
add	8	9 10	IF	ID	EX	MEM	WB	-	-	-	-
lw	10	400(8)	-	IF	ID	EX	MEM	WB	-	-	-
sub	9	10 8	-	-	IF	ID	ID	ID	EX	MEM	WB

d) Pipeline clock cycle time = 270 ps

Total no. Of cycles = 9

Total execution time = clock cycle time * no. of cycles

= 270 * 9 ps

= 2430 picoseconds

e) Average Pipeline Utilisation:

$$\text{APU} = \frac{\text{(number of instructions * number of unique stages/instruction)}}{\text{(number of cycles * num of possible stages that can be executed/cycle)}}$$

(Equivalent to filled boxes/total boxes in the cycles vs. Stage table)

APU = 3*5/(9*5) = 3/9 = 1/3

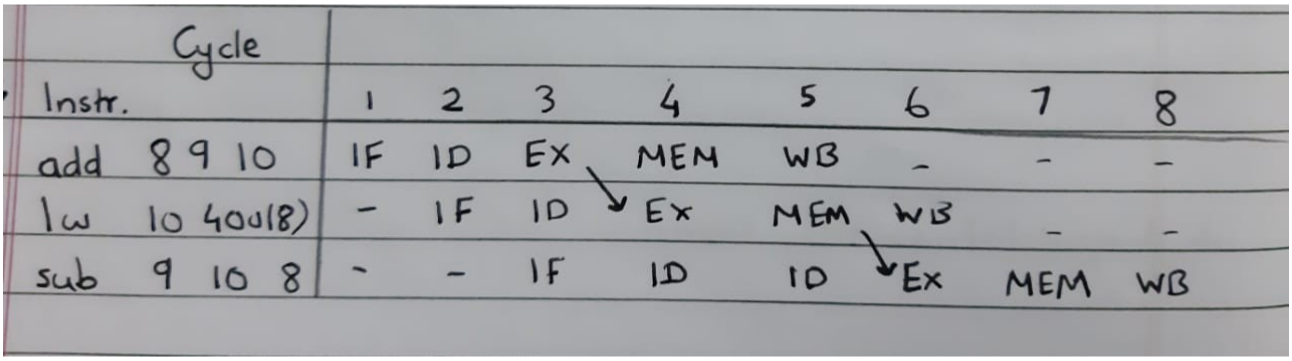
f)No, the instructions cannot be re-ordered to reduce the execution time.

The load word instruction requires the value in register 8 as an operand.

However, register 8 is being written to in the AND instruction. Hence instruction 2 cannot be placed before instruction 1.

Instruction 3 cannot be placed before instruction 2 either. That's because instruction 3 requires register 10. But register 10 is being updated in the load word instruction.

iii) **FULL Forwarding**



Cycle								
Instr.	1	2	3	4	5	6	7	8
add 8 9 10	IF	ID	EX	MEM	WB	–	–	–
lw 10 400(8)	–	IF	ID	Ex	MEM	WB	–	–
sub 9 10 8	–	–	IF	ID	ID	Ex	MEM	WB

c)

d) Pipeline clock cycle time = 280 ps
Total no. Of cycles = 8
Total execution time = clock cycle time * no. of cycles
= 280 * 8 ps
= 2240 picoseconds

e) Average Pipeline Utilisation:

$$\text{APU}= \frac{\text{(number of instructions * number of unique stages/instruction)}}{\text{(number of cycles * num of possible stages that can be executed/cycle)}}$$

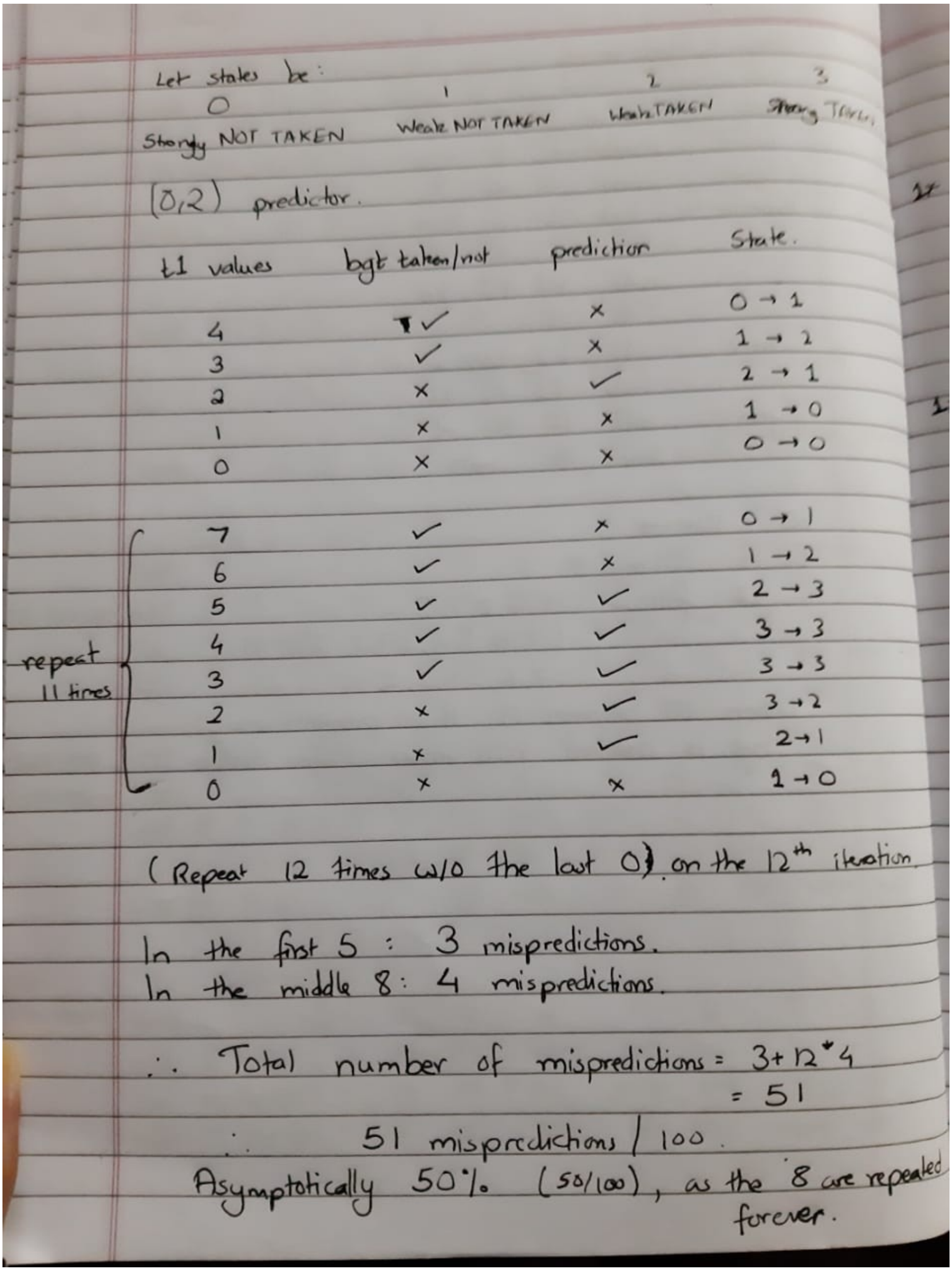
(Equivalent to filled boxes/total boxes in the cycles vs. Stage table)

APU = 3*5/(8*5) = 3/8

f)No, the instructions cannot be re-ordered to reduce the execution time.
The load word instruction requires the value in register 8 as an operand.
However, register 8 is being written to in the AND instruction. Hence instruction 2 cannot be placed before instruction 1.
Instruction 3 cannot be placed before instruction 2 either. That’s because instruction 3 requires register 10. But register 10 is being updated in the load word instruction.

Question 2)

(a) The number of mispredictions with a (0,2) branch predictor is 51 out of 100 and is asymptotically 4/8 (50%= 50 out of every 100, 4 out of every 8) .



Let states be:

0	1	2	3
Strongly NOT TAKEN	Weak NOT TAKEN	Weak TAKEN	Strong Taken

(0,2) predictor.

t1 values	bgt taken/not	prediction	State.
4	T ✓	×	0 → 1
3	✓	×	1 → 2
2	×	✓	2 → 1
1	×	×	1 → 0
0	×	×	0 → 0

repeat 11 times

t1 values	bgt taken/not	prediction	State.
7	✓	×	0 → 1
6	✓	×	1 → 2
5	✓	✓	2 → 3
4	✓	✓	3 → 3
3	✓	✓	3 → 3
2	×	✓	3 → 2
1	×	✓	2 → 1
0	×	×	1 → 0

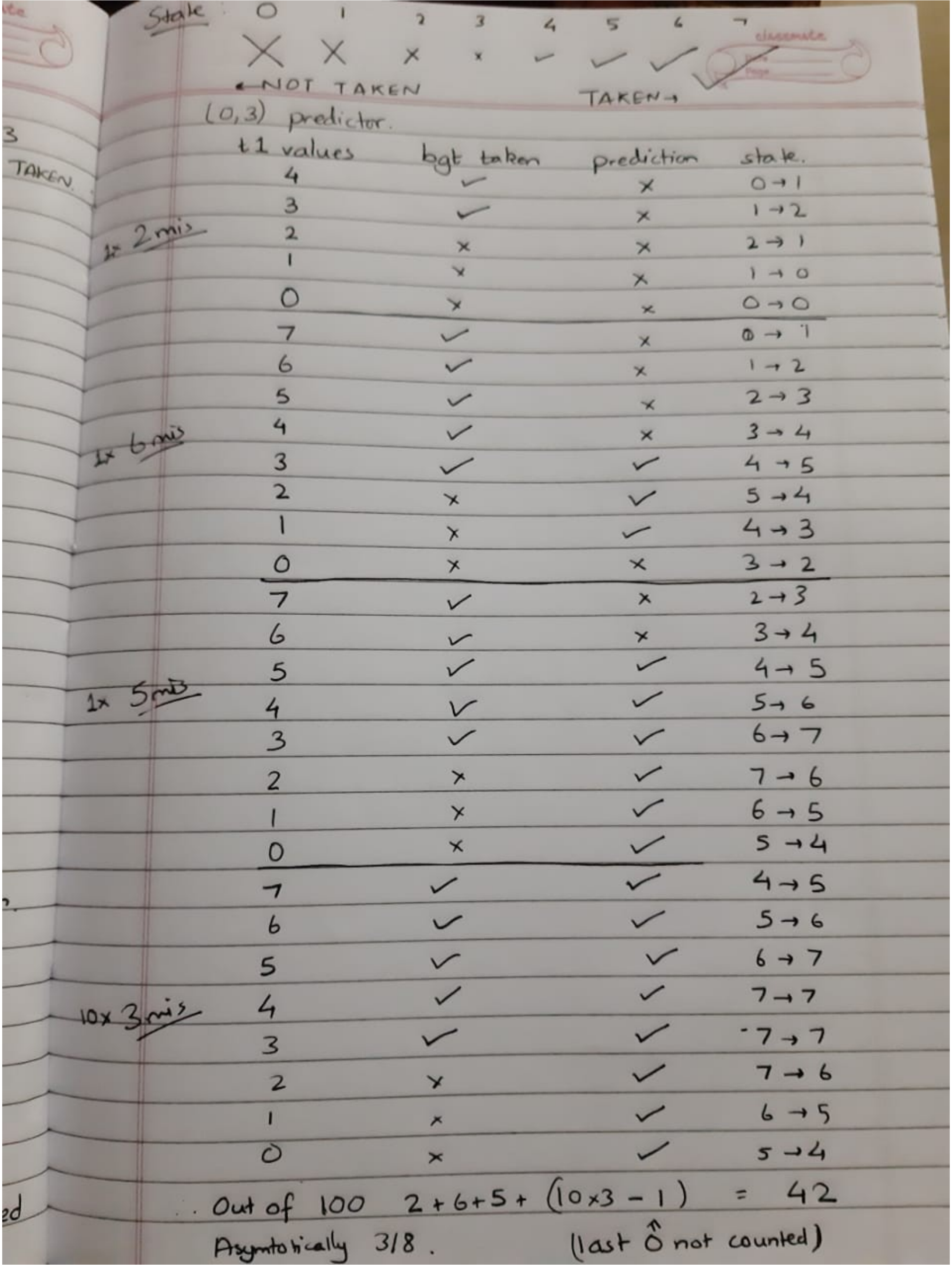
(Repeat 12 times w/o the last 0). on the 12th iteration.

In the first 5 : 3 mispredictions.
In the middle 8: 4 mispredictions.

∴ Total number of mispredictions = 3+ 12 * 4
= 51

∴ 51 mispredictions / 100.
Asymptotically 50%. (50/100), as the 8 are repeated forever.

(b) The number of mispredictions with a (0,3) branch predictor is 42 out of 100 and is asymptotically 3/8.



State	0	1	2	3	4	5	6	7
	×	×	×	×	✓	✓	✓	✓

← NOT TAKEN TAKEN →

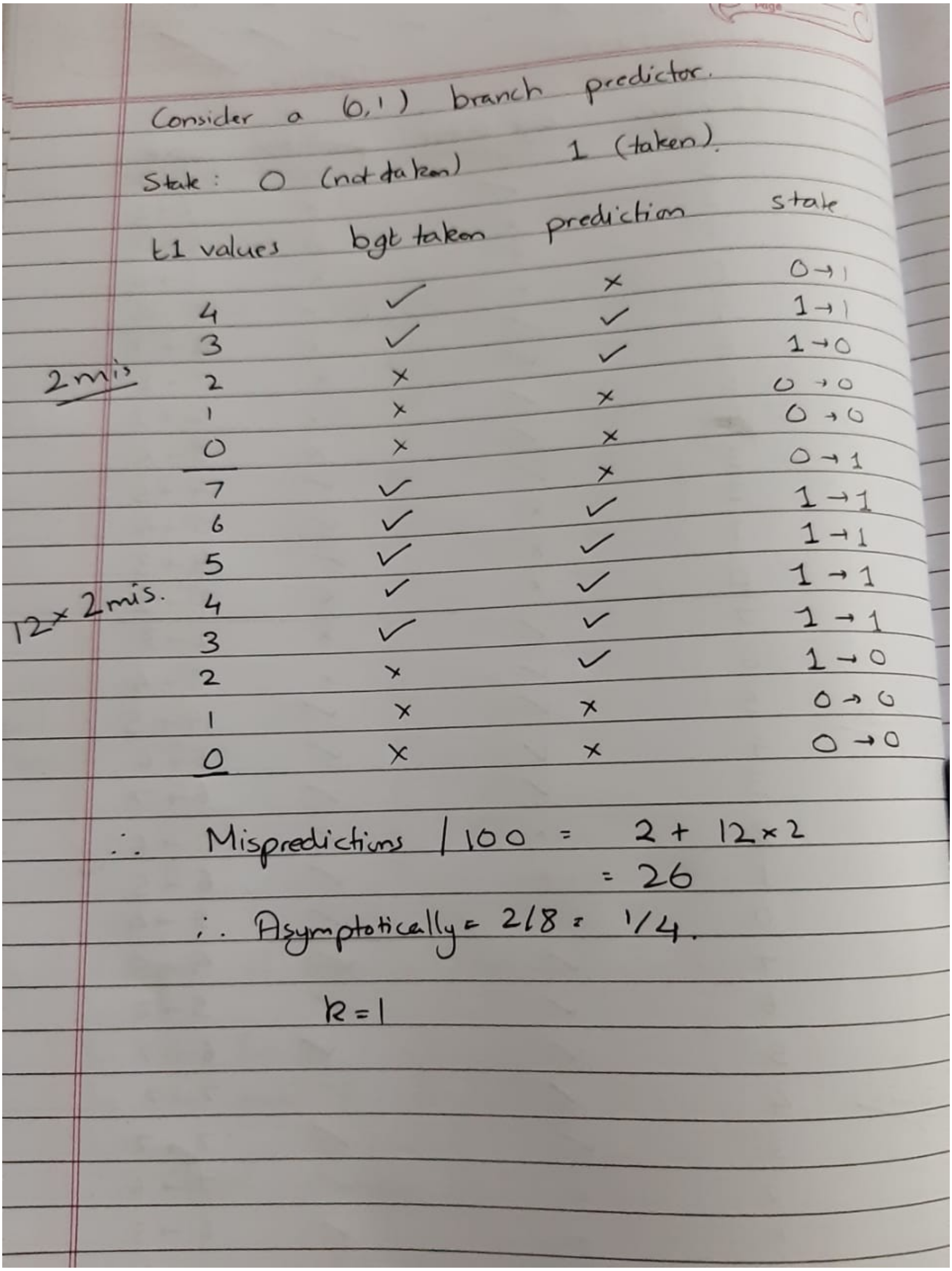
(0,3) predictor.

	t1 values	bgt taken	prediction	state.
1x 2 mis	4	✓	×	0 → 1
	3	✓	×	1 → 2
	2	×	×	2 → 1
	1	×	×	1 → 0
	0	×	×	0 → 0
1x 6 mis	7	✓	×	0 → 1
	6	✓	×	1 → 2
	5	✓	×	2 → 3
	4	✓	×	3 → 4
	3	✓	✓	4 → 5
	2	×	✓	5 → 4
	1	×	✓	4 → 3
	0	×	×	3 → 2
1x 5 mis	7	✓	×	2 → 3
	6	✓	×	3 → 4
	5	✓	✓	4 → 5
	4	✓	✓	5 → 6
	3	✓	✓	6 → 7
	2	×	✓	7 → 6
	1	×	✓	6 → 5
	0	×	✓	5 → 4
10x 3 mis	7	✓	✓	4 → 5
	6	✓	✓	5 → 6
	5	✓	✓	6 → 7
	4	✓	✓	7 → 7
	3	✓	✓	7 → 7
	2	×	✓	7 → 6
	1	×	✓	6 → 5
	0	×	✓	5 → 4

∴ Out of 100 $2+6+5+(10\times 3-1) = 42$

Asymtotically 3/8. (last $\hat{0}$ not counted)

(c) The minimum number of mispredictions with a (0,k) branch predictor is 26 out of 100, is asymptotically 1/4 and occurs for k = 1 (as higher k dimishes performance, asymtotically 3/8 for k>=3)



Consider a (0,1) branch predictor.

State : 0 (not taken) 1 (taken).

t1 values	bgt taken	prediction	state
4	✓	×	0→1
3	✓	✓	1→1
2	×	✓	1→0
1	×	×	0→0
0	×	×	0→0
7	✓	×	0→1
6	✓	✓	1→1
5	✓	✓	1→1
4	✓	✓	1→1
3	✓	✓	1→1
2	×	✓	1→0
1	×	×	0→0
0	×	×	0→0

2 mis

12× 2 mis.

∴ Mispredictions / 100 = 2 + 12×2

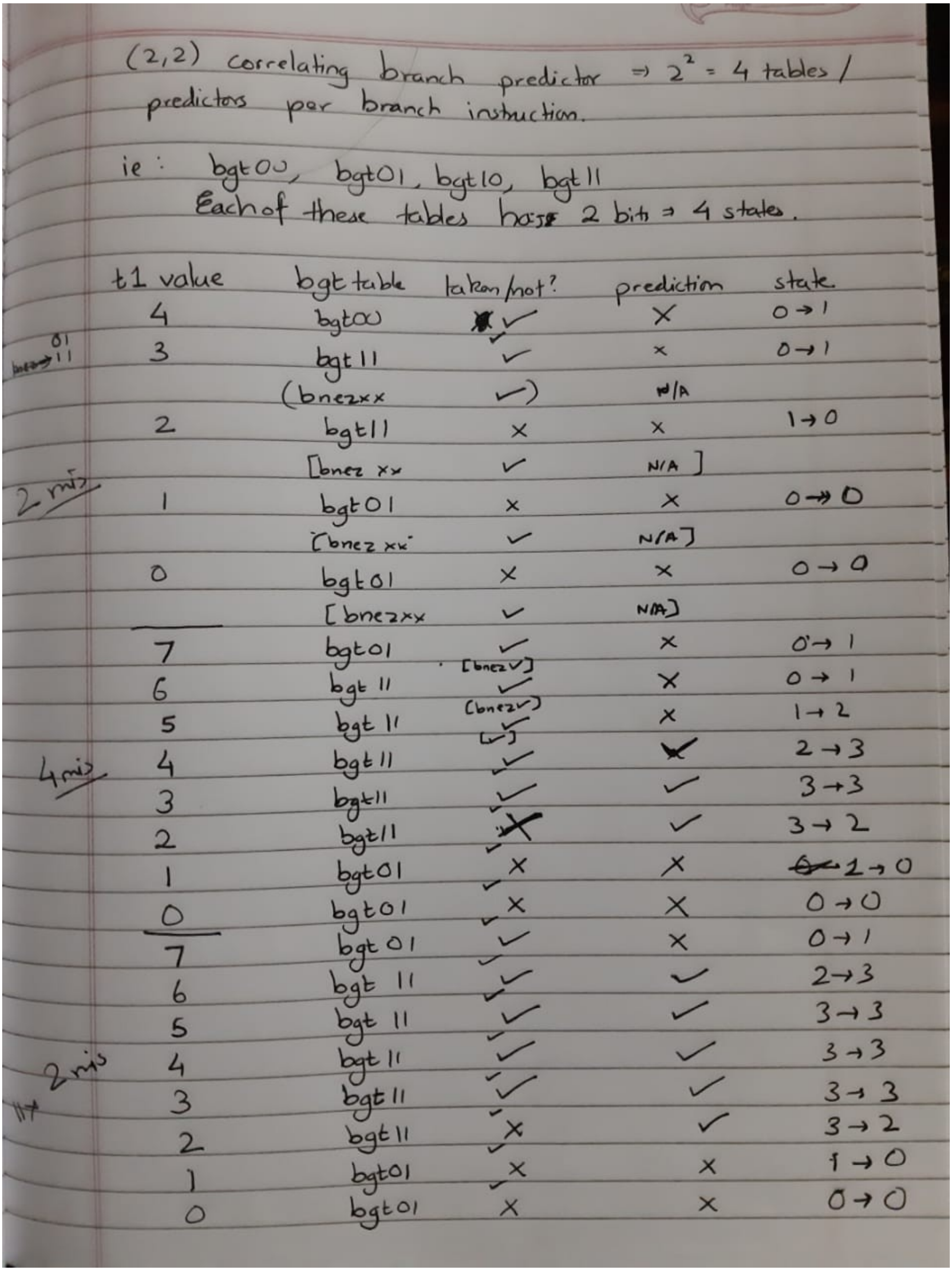
= 26

∴ Asymptotically = 2/8 = 1/4.

k=1

2 + 4 + 2*11 = 28/100 mispredictions , asymptotically 2/8 = 1/4.

(d) The number of mispredictions with a (2,2) correlating branch predictor is 28 out of 100 and is asymptotically 1/4 (25%).



(2,2) correlating branch predictor ⇒ $2^2 = 4$ tables / predictors per branch instruction.

ie : bgt00, bgt01, bgt10, bgt11
Each of these tables ~~has~~ 2 bits ⇒ 4 states.

	t1 value	bgt table	taken/not?	prediction	state
	4	bgt00	~~✗~~ ✓	✗	0 → 1
01 bnez→11	3	bgt11	✓	×	0 → 1
		(bnezxx	✓)	N/A	
	2	bgt11	×	×	1 → 0
		[bnez xx	✓	N/A]	
2 mis	1	bgt01	×	✗	0 → 0
		[bnez xx	✓	N/A]	
	0	bgt01	✗	×	0 → 0
		[bnezxx	✓	N/A]	
	7	bgt01	✓ [bnez✓]	×	0 → 1
	6	bgt 11	✓ (bnez✓)	✗	0 → 1
	5	bgt 11	✓ [✓]	×	1 → 2
4 mis	4	bgt 11	✓	✓	2 → 3
	3	bgt11	✓	✓	3 → 3
	2	bgt11	✗	✓	3 → 2
	1	bgt01	✗	✗	~~0~~ 1 → 0
	0	bgt01	✗	✗	0 → 0
	7	bgt 01	✓	×	0 → 1
	6	bgt 11	✓	✓	2 → 3
	5	bgt 11	✓	✓	3 → 3
2 mis	4	bgt 11	✓	✓	3 → 3
117	3	bgt 11	✓	✓	3 → 3
	2	bgt11	✗	✓	3 → 2
	1	bgt01	✗	×	1 → 0
	0	bgt01	✗	×	0 → 0

After a while, I am representing any bnez branch predictor as just a small tick mark (the bnez is always taken) Each of the 4 branch predictors per branch instruction has 4 states: 0,1,2,3 (Strongly not taken to strongly taken) The table shows which branch predictor for bgt is used as t1 changes, what actually happens at bgt (taken/not taken), what each predictor predicts, and the state changes for each predictor (based on its last prediction).

(e) As the number of iterations, n, tends to infinity, the minimum number of mispredictions with an (m,k) correlating branch predictor is achieved for m=10 and k= 1. For this predictor, the number of mispredictions is 7 out of 100 and is asymptotically 0 .

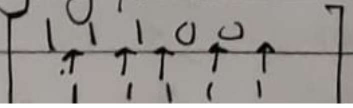
By trial and error, it does not work for m = 8 or m = 6. As there are 5 takens and 3 not takens, it seems that 5 bits for the bgt alone is the minimum required to have a sufficient number of branch predictors, whose states are set appropriately, to predict correctly each time.

2 + 5 + 0*11 = 7 mispredictions out of 100.
0 mispredictions thereafter.

Consider a (10,1) predictor
(Every other bit is 1, as bnez is always taken,
so basically (5,1) w.r.t bgt.

	t1 values	bgt predictor	taken/not?	prediction	state
	4	00000	✓	×	0→1
	3	00001	✓	×	0→1
	2	00010	×	×	0→0
2 mis	1	00110	×	×	0→0
	0	01100	×	×	0→0
	7	11000	✓	×	0→1
	6	10001	✓	×	0→1
	5	00011	✓	×	0→1
5 mis	4	00110	✓	×	0→1
	3	01100	✓	×	0→1
	2	11111	×	×	0→0
	1	11110	×	×	0→0
	0	11100	×	×	0→0
	7	11000	✓	✓	1→1
	6	10001	✓	✓	1→1
	5	00011	✓	✓	1→1
0 mis	4	00111	✓	✓	1→1
	3	01111	✓	✓	1→1
	2	11111	×	×	0→0
	1	11110	×	×	0→0
	0	11100	×	×	0→0

Those 5 bit numbers are actually 10 bits long.
(there is a hidden 1 representing bnez taken
in every gaps and to the right



1 1 1 0 0
1 1 1 1 1