Fixed -False @ = True • = False

1.1) underline = missed / wrong prediction

@@.@ .@@. @@.@ @.@@ .@.@ @.@@ .... $\frac{57}{112}$ accuracy

..@@ .@.@ @... ...@ @.@@ .@@. @@..

....@ @.@@ .@@@ .@@. ....

...@ @.@@ .@@@ .@@. @@.@ @.@@ .@@.

1.2) static false first, option a (false first, then wle happens next)

@@.@ .@@. @@.@ @.@@ .@.@ @.@@ .... $\frac{50}{112}$ accuracy

..@@ .@.@ @... ...@ @.@@ .@@. @@..

....@ @.@@ .@@@ .@@. ....

...@ @.@@ .@@@ .@@. @@.@ @.@@ .@@.

1.3) Bimodal - False first, then w/e hasn't gone wrong twice in a row — assume we have been wrong once @ start

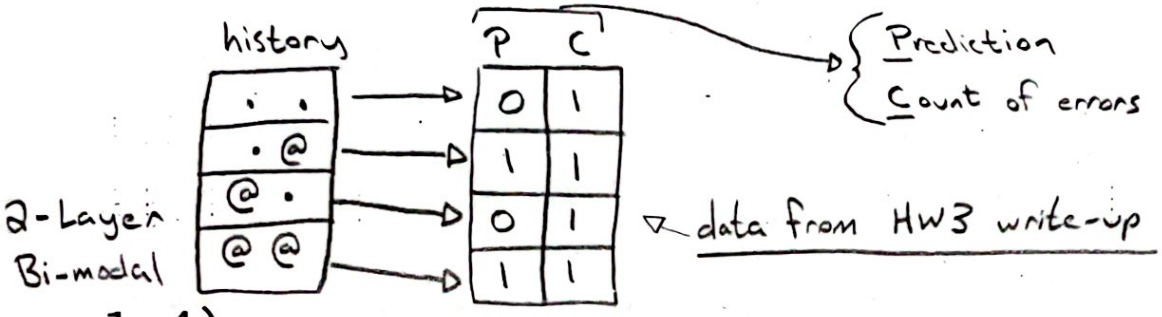
T
@@.@ .@@. @@.@ @.@@ .@.@ @.@@ ...F 71/112

accuracy

..@@ T.@.@ @..F ...@ T@.@@ .@@. @@..

F
....@ T@.@@ .@@@ .@@. .F...

T
...@ @.@@ .@@@ .@@. @@.@ @.@@ .@@.



1.4)

96/112

accuracy

.. @@.@ .@@. @@.@ @.@@ .@.@ @.@@ ....

assumed

..@@ .@.@ @... ...@ @.@@ .@@. @@..

....@ @.@@ .@@@ .@@. ....

...@ @.@@ .@@@ .@@. @@.@ @.@@ .@@.

Feb 27th

Teeny AT → 16-Bit Dynamic Encoding

DATA Instructions

lod rB, [rA - 17]

str [rC+3], rZ ← "register zero"

set rD, rA + 4

psh rA

pop rB

ALU Instructions

add rB, rA - 3

sub, mpy, div, mod, neg

and, or, xor, shf, rot

Control Instructions

cmp

jmp

DJZ - Decrement + Jump if zero

DLY - Delay for a certain # of clock cycles

CAL - call

Registers

PC - Program Counter

SP - Stack Pointer

rZ - Zero register

[rA - rE]

[r3 - r7]

Instruction Level Parallelism (ILP) → faster execution needs more power

"Pipelining"

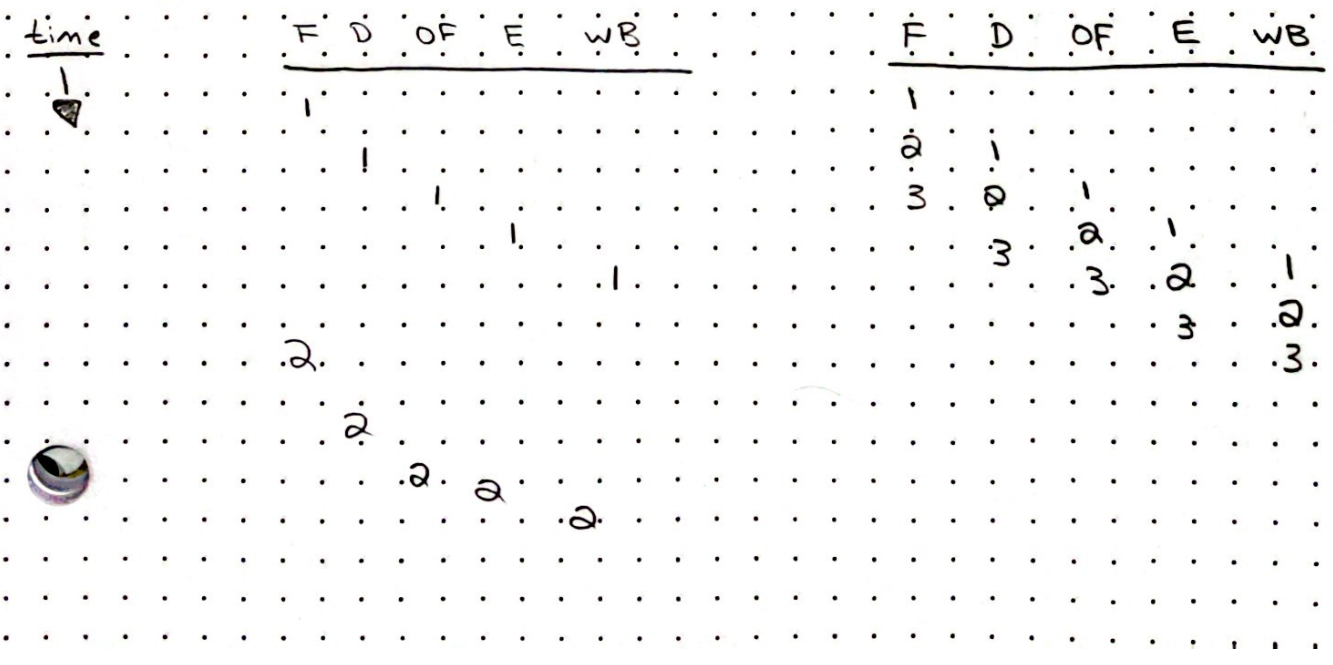
Fetch
Decode
(OF) Operand Fetch
Execute
(WB) Write Back

look-up tatches
flip-flop cycles

100 lines of code

500 cycles

104 cycles (using pipelining)



Hazards (can make pipelining fail)

Structural

```
set rA, 0xFE
add rA, 5
lod rB, [rA]
div rB, 4
```

two instructions need bus @ same time

Data → when one instruction relies on another instruction to finish before it can execute.

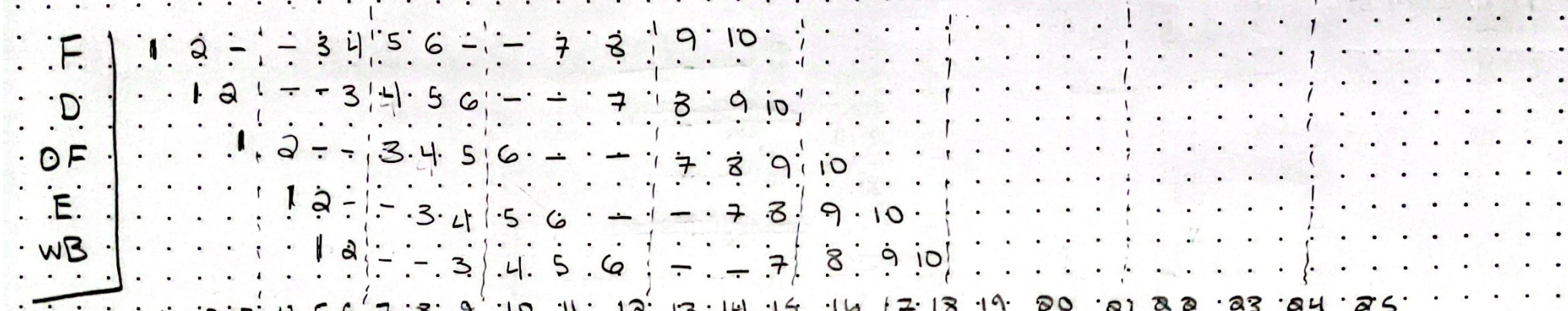
Control → whenever we want to change what happens next, we can cause control hazards (jmp instructions are an example)

Feb 29th

$$dist = (x_1 - x_2)^2 + (y_1 - y_2)^2$$ Euclidean sq.

```
① Lod rA, [x1]   ← loads on (bus) OF, Ex does nothing
② Lod rB, [x2]
③ sub rA, rB
④ mpy rA, rA
⑤ Lod rB, [y1]
⑥ Lod rC, [y2]
⑦ sub rB, rC
⑧ mpy rB, rB
⑨ add rA, rB
⑩ str [dist], rA
```

Pipeline Graph (structural)

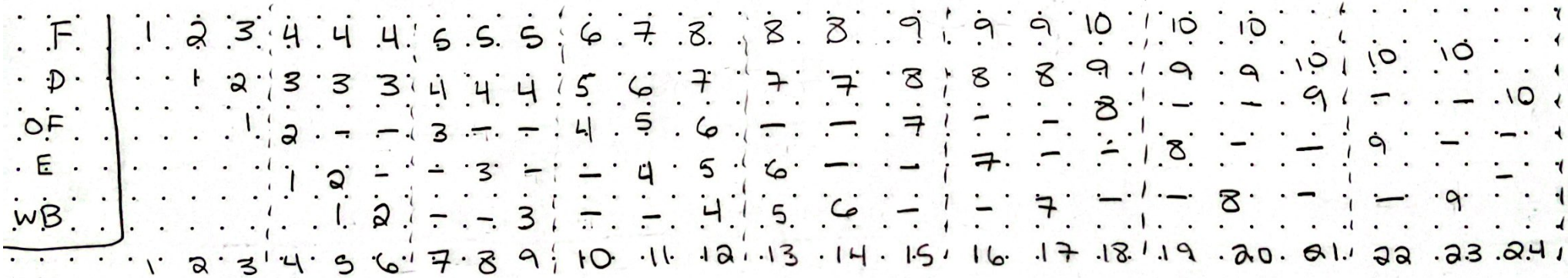


No Pipeline

(Data) 50

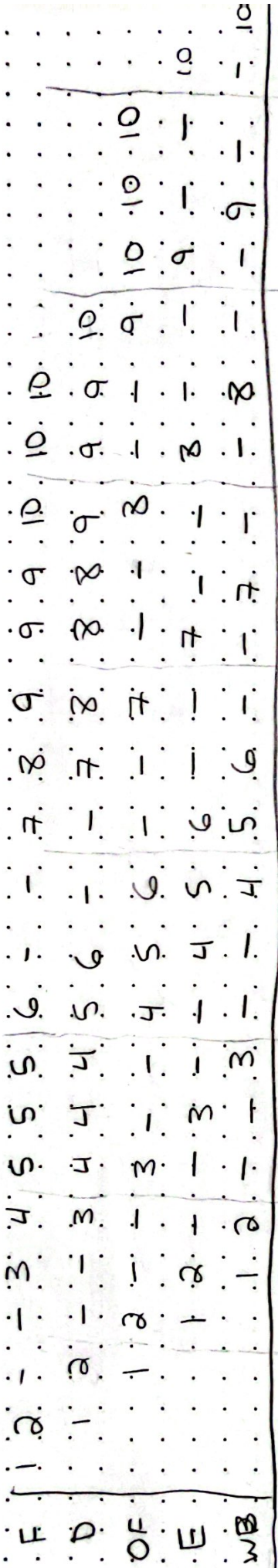
Pipeline

18



inst rX, rY
inst rZ, rX
dependency

10 finishes as expected



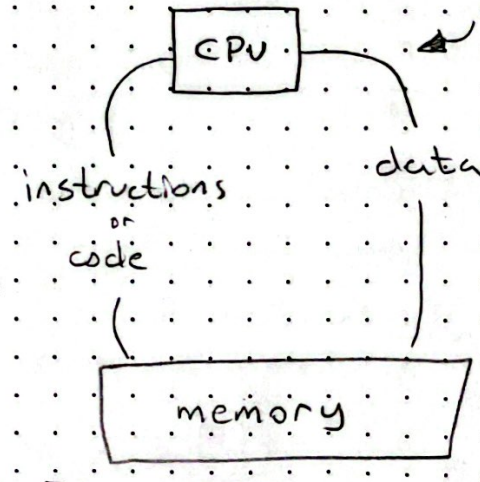
structural + data

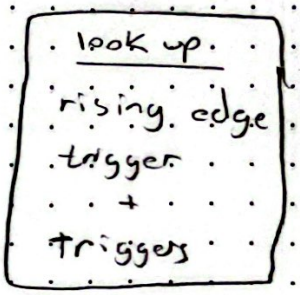
structural solutions

wait one full cycle (stall)

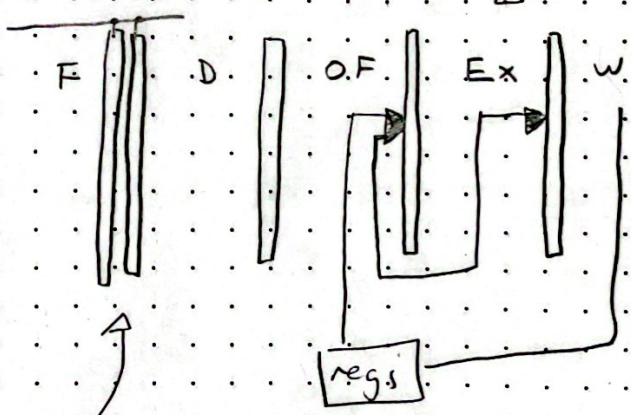
multiple buses

Harvard Architecture





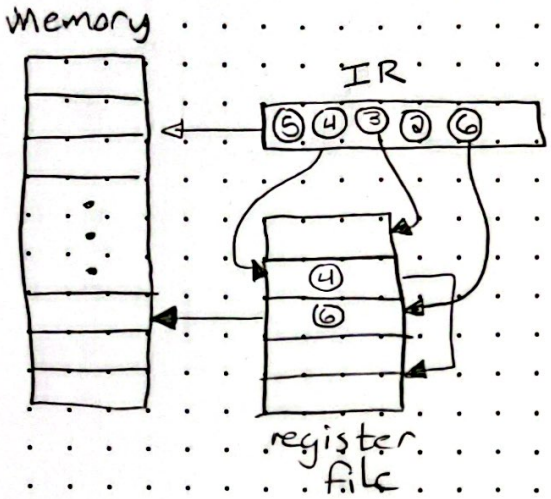
Register Forwarding



latch = temp storage device

Mar 5th

addressing (modes)



① Implicit addressing (don't tell it what to do)
alpha 'x'

② Immediate addressing (some value)

③ Register addressing (accessing directly)

④ Register by register addressing (register points to another register)

⑤ Indirect addressing (immediate value)

⑥ Register indirection addressing.

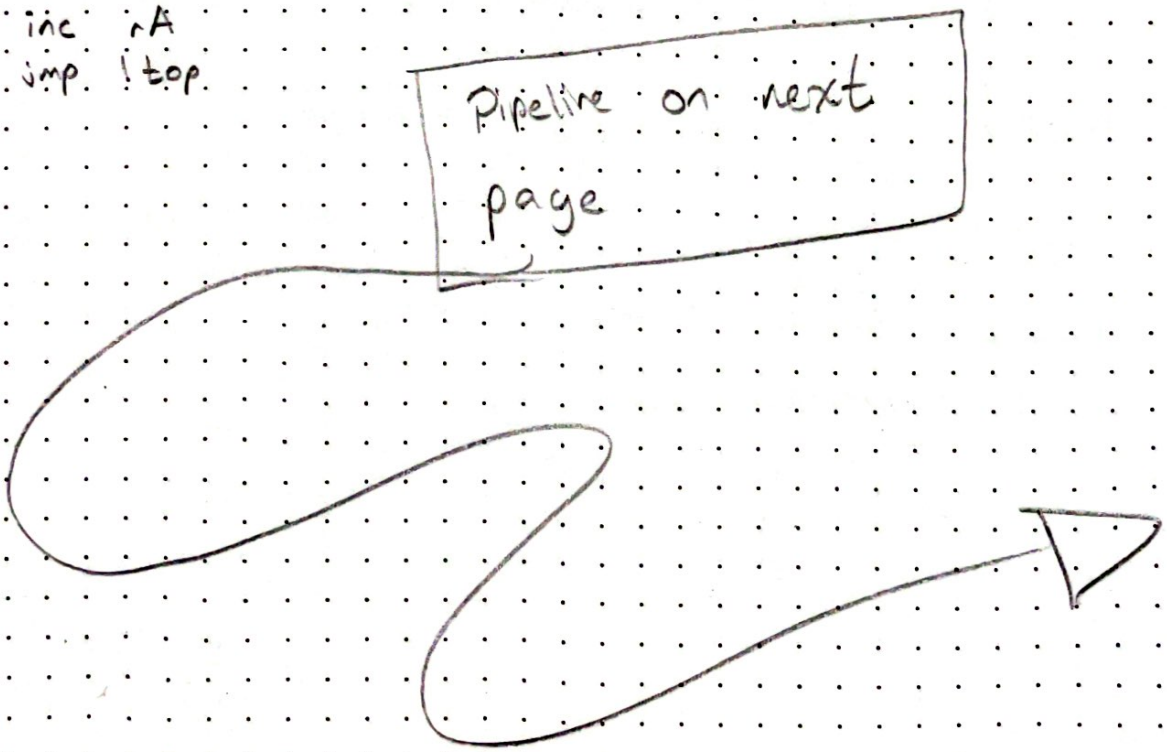
control hazards

```
for (int i = 3; i < 5; i++){
    if (i % 2 == 0)
        i++;
}
```

ASM

```
1     set rA, 3
  !top
2     cmp rA, 5
3     jge !done
4     set rB, rA
5     mod rB, 2      // EQ : 1 if equal
6     jne !continue
7     inc rA
  !continue
8     inc rA
9     jmp !top
  !done
```

Pipeline on next page



control + data hazards

* = flush

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
F	1	2	3	3	4	4	5	6	6	7	7	8	–	8	9	*	2	3	4	4	5	6	6	7	7		
D		1	2	2	3	3	4	5	5	6	6	7	–	–	8	9	–	2	3	3	4	5	5	6	6	7	
OF			1	–	2	–	3	4	–	5	–	6	–	–	–	8	9	–	2	–	3	4	–	5	–	6	7
E				1	–	2	–	3	4	–	5	–	6	–	–	–	8	9	–	2	–	3	4	–	5	–	6	
WB					1	–	2	–	3	4	–	5	–	6	–	–	–	8	9	–	2	–	3	4	–	5	–		

- use register forwarding
- unconditional JMP known at decode, so flush
- conditional JMP known at execute, so flush (OF grabs address, Eq flag + gt flag)

	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
F																
D	8															
OF	–	8														
E	7	–	8													
WB	6	7	–	8												

Mar 7th

study
- caching

Hand-out

```
set rB, 5
mpy rA, rB
jmp !top

// more code
```

helps pipeline

```
set rB, 5
jmp !top
mpy rA, rB
```

(put 'true' cases at top of if statement to make code more efficient)

Branch Prediction

① Fixed False - We assume all conditional branches will not branch (are false)
↳ easiest to predict / cheapest

② Static False First

diff OP codes → a)
- jmp ⇒ true
- je/jne ⇒ false first, after, whatever happens first
  ↳ have to use caching, storing info about how instructions should work

b) false first, after, whatever was first by address.

③ Bimodal - False first, whatever hasn't gone wrong twice in a row next
↳ keep predicting what we are predicting until we are wrong twice in a row

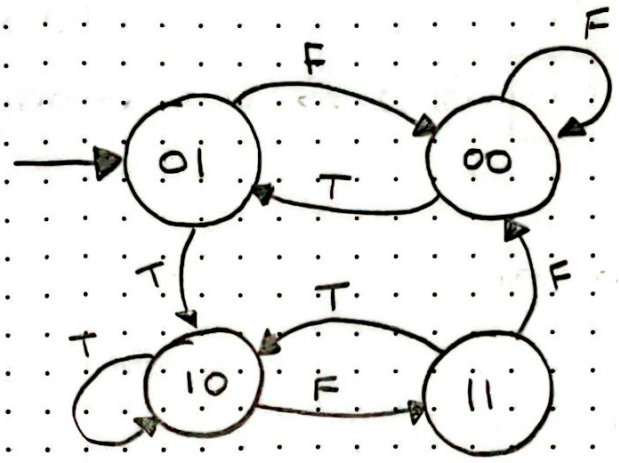
④ Two-Layer Bimodal -

work-sheet

2 bits need to be added to use Bimodal

cnt = how many wrongs
prediction = T/F

P:C



PC & 10

Mar 14th

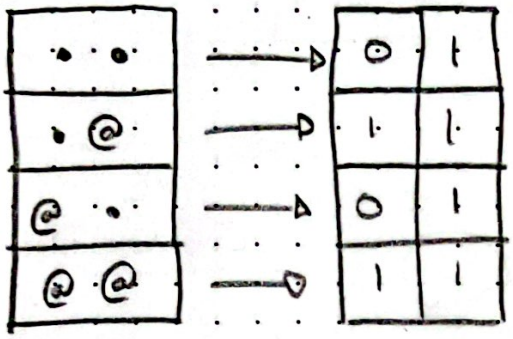
Line 13 of HW 1 has operands reversed

[Intel PIN demonstration]

- Ran Bubble Sort code to show how he used PIN to show branching / patterns
- used g++ to compile, then logged branches of compiler

Two-layer Bi-modal

Assume 0 branches were not taken (specific to HW3, not all 2-layer Bimodal)



① 10 >> 11 >> 00 >> 01 >> 00 *

② 10 >> 11 >> 10 >> 11 >> 10 *

③ 10 >> 11 >> 10 >> 11 >> 10 *

④ 00 >> 01 >> 00 >> 01 >> 00

① * >> 01 >> 00 >> 01

② * >> 11 >> 10

③ * >> 11 >> 10 >> 11 >> 10

④