Pipeline

Complete Revision

Part 5

- Deepak Poonia

(IISc Bangalore; *GATE AIR 53 & 67*)

Content of this Lecture:

1. Control Dependencies, Hazards
2. GATE PYQs

GATE 2024 Complete Course Link:

https://www.goclasses.in/s/pages/gatecompletecourse

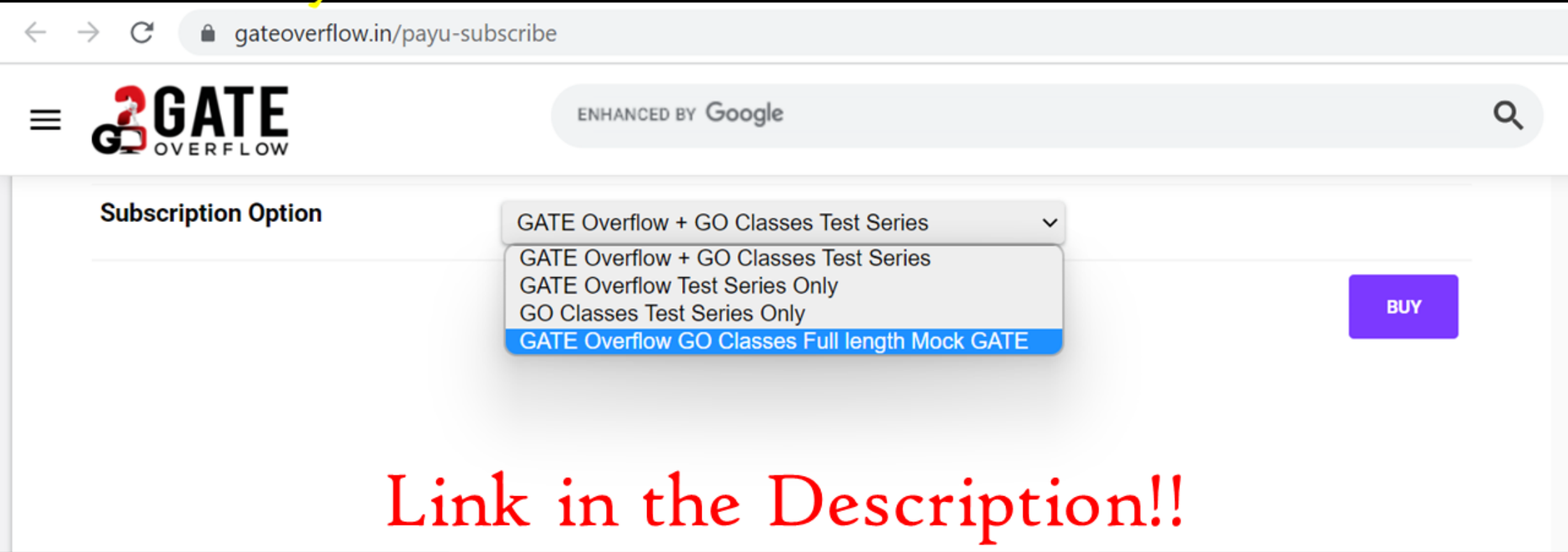
Instructor:

Deepak Poonia

IISc Bangalore

GATE CSE AIR 53; AIR 67; AIR 206; AIR 256

Subscribe for ALL 15 Mock Tests By GateOverflow + GO Classes



Link in the Description!!

GO CLASSES

GATE 2024
COMPLETE COURSE CS-IT

(1 YEAR)

GO CLASSES

GATE 2025

COMPLETE COURSE CS-IT

(2 YEARS)

2023 Discrete Mathematics

★ ★ ★ ★ ★ 5.0 (62 ratings)

Deepak Poonia (MTech IISc Bangal...

Free

2023 C Programming

★ ★ ★ ★ ★ 5.0 (59 ratings)

Sachin Mittal (MTech IISc Bangalor...

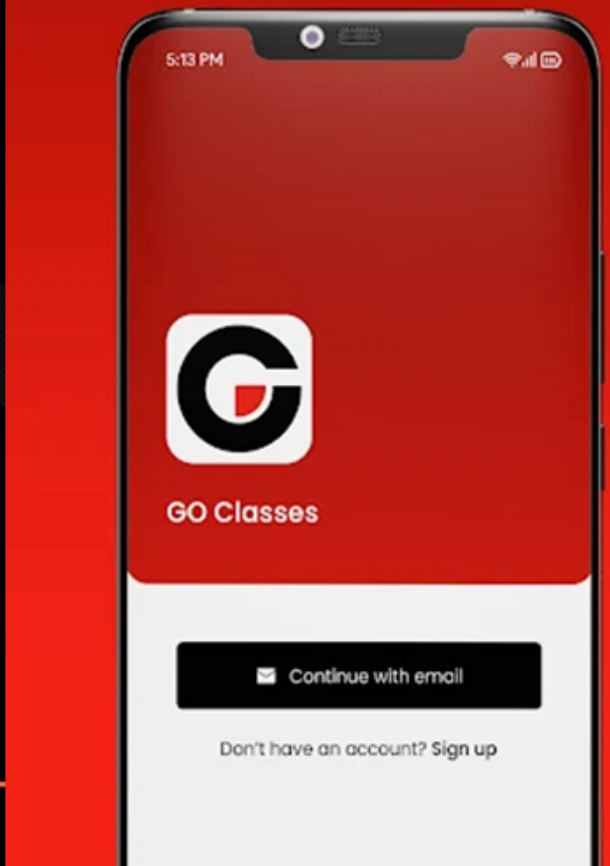
Free

On “GATE-Overflow” Website

Download the GO Classes Android App:

https://play.google.com/store/apps/details?id=com.goclasses.courses

Search "GO Classes" on Play Store.



Pipeline Hazards

- Hazards: situations that makes the pipeline to stall or idle.

1. Structural hazards
   - Caused by resource contention
   - Using same resource by two instructions during the same cycle
2. Data hazards
   - An instruction may compute a result needed by next instruction
   - Hardware can detect dependencies between instructions
3. Control hazards
   - Caused by instructions that change control flow (branches/jumps)
   - Delays in changing the flow of control

- Hazards complicate pipeline control and limit performance

Five Stages of Pipeline

Pipelining is an implementation technique in which multiple instructions are overlapped in execution.

The stages of instruction execution / pipelining are

- IF --- Instruction Fetch
- ID --- Instruction Decode / Register Read
- EX --- Execute in ALU / calculate address
- MEM --- Data memory access
- WB ---- Write back in register

The data flows from the left stage to right stage.

But in WB the result is written back (right to left data flow) in the register

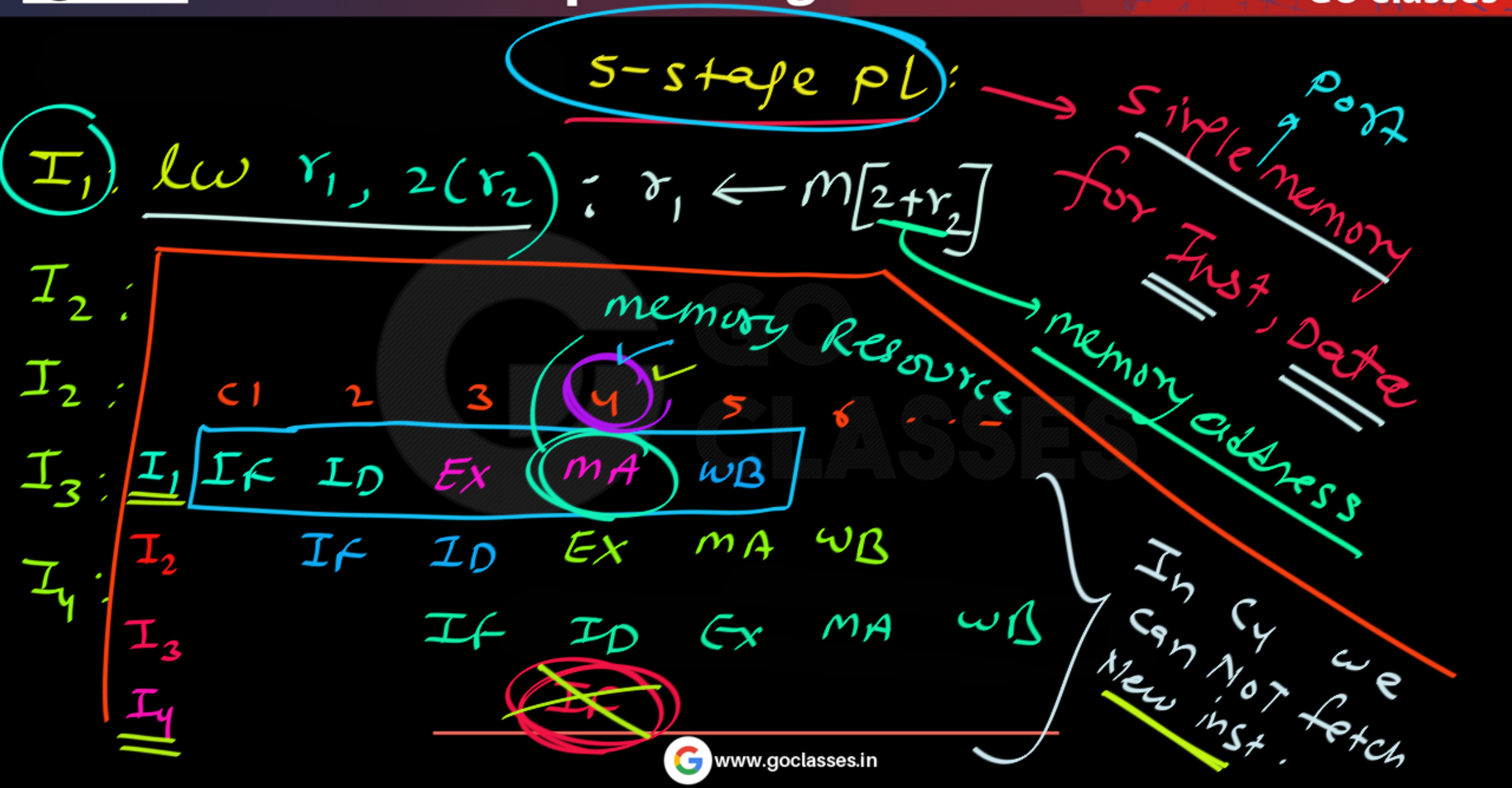
Next Topic:

Structural Hazards

Resource Hazards

RESOURCE HAZARDS A resource hazard occurs when two (or more) instructions that are already in the pipeline need the same resource. The result is that the instructions must be executed in serial rather than parallel for a portion of the pipeline. A resource hazard is sometime referred to as a *structural hazard*.

RESOURCE HAZARDS A resource hazard occurs when two (or more) instructions that are already in the pipeline need the same resource. The result is that the instructions must be executed in serial rather than parallel for a portion of the pipeline. A resource hazard is sometime referred to as a *structural hazard*.

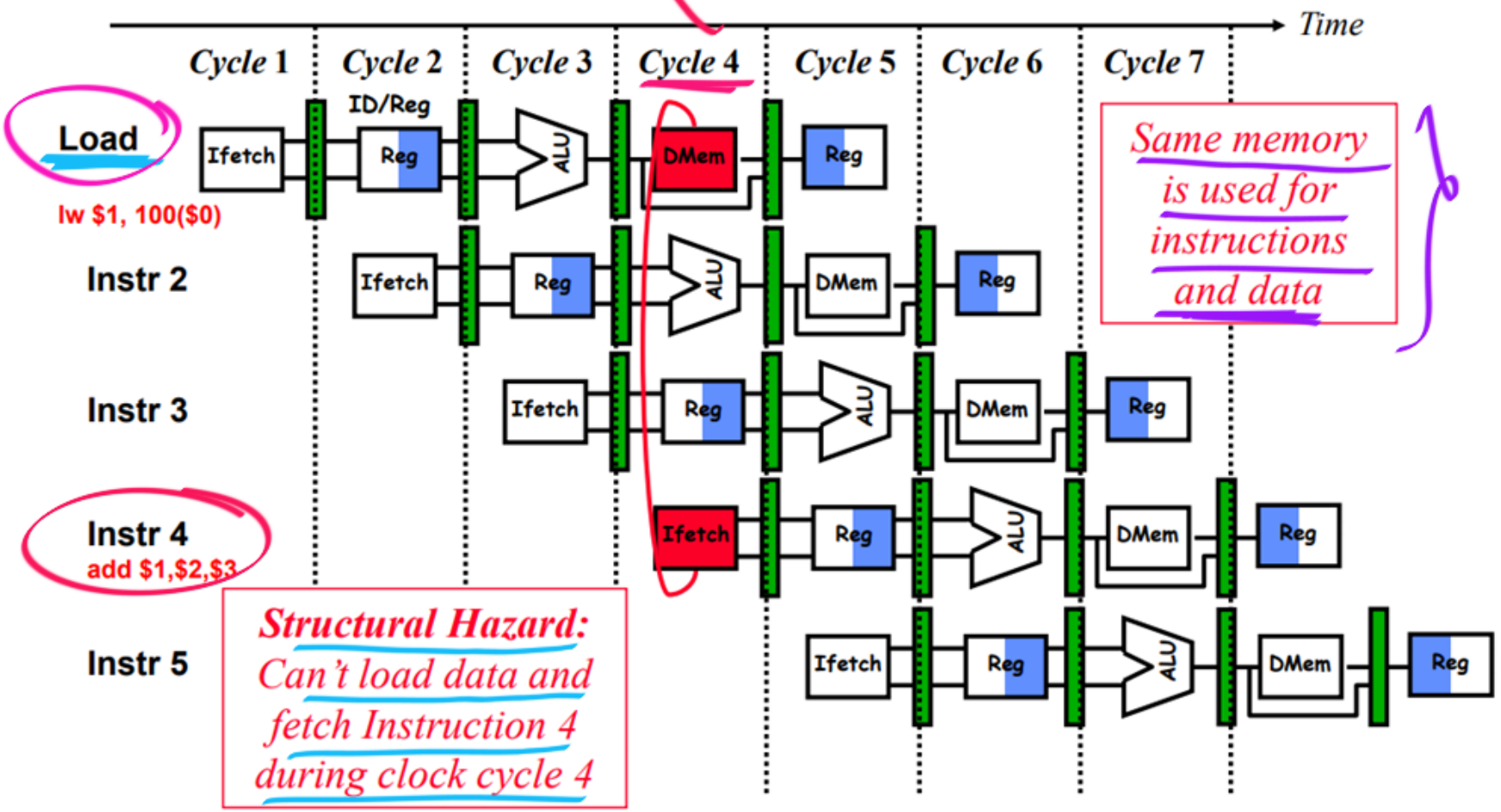

5-stage pL:
simple memory
for Inst, Data
I1: lw r1, 2(r2) ; r1 ← M[2+r2]
memory address
memory Resource
C1 2 3 4 5 6
I1 IF ID EX MA WB
I2 IF ID EX MA WB
I3 IF ID EX MA WB
I4
In Cy we can Not fetch New Inst.
www.goclasses.in

Simple (port) main memory } for Inst & Data

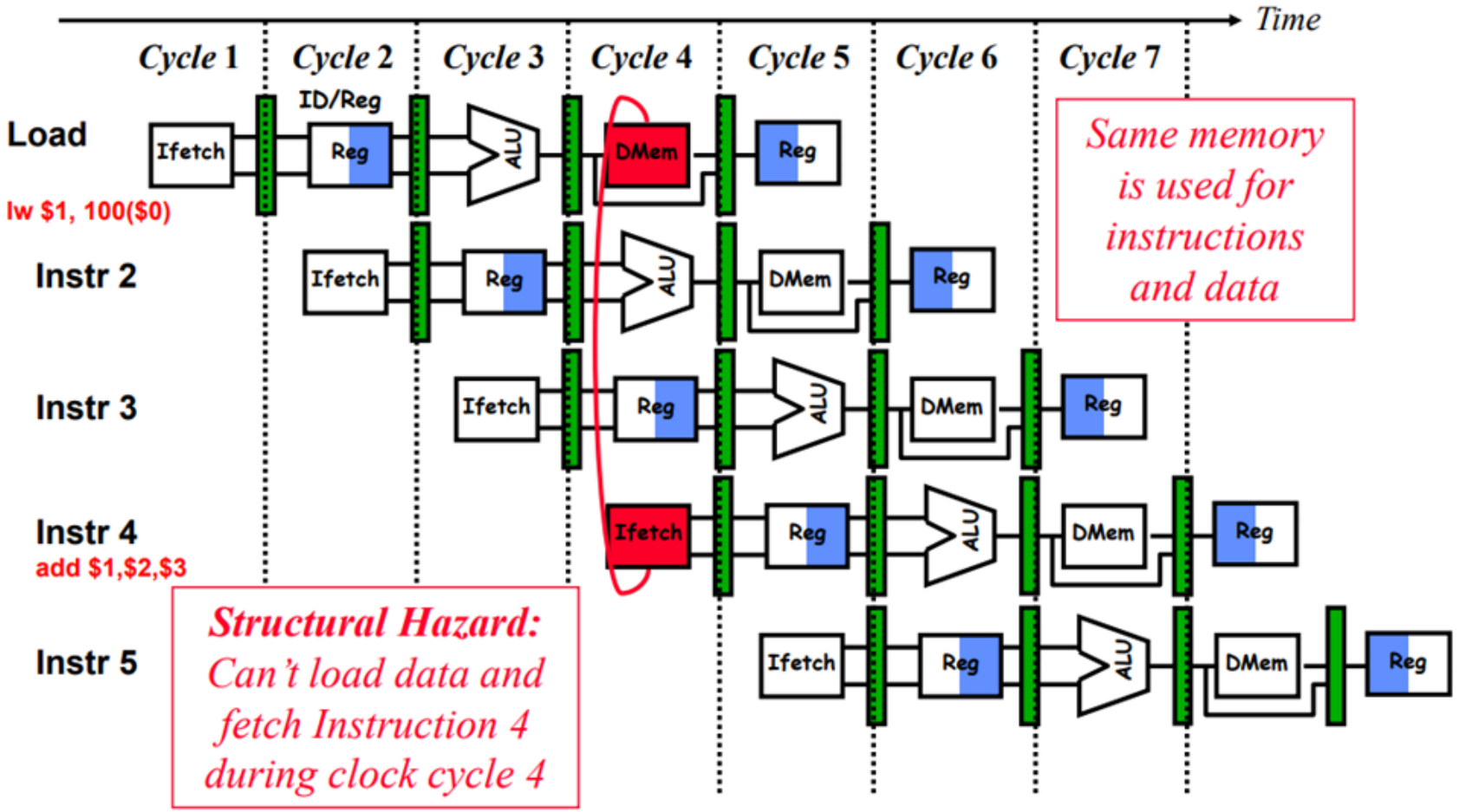
↳ (IF) : Inst. fetch

↳ MA : Data mem. read } Access Same Resource (memory)

1. Structural Hazard - Conflict due to Memory Access



1. Structural Hazard - Conflict due to Memory Access



Solution :

1. Put Additional H/W eg: Separate memory for ins & data

OR

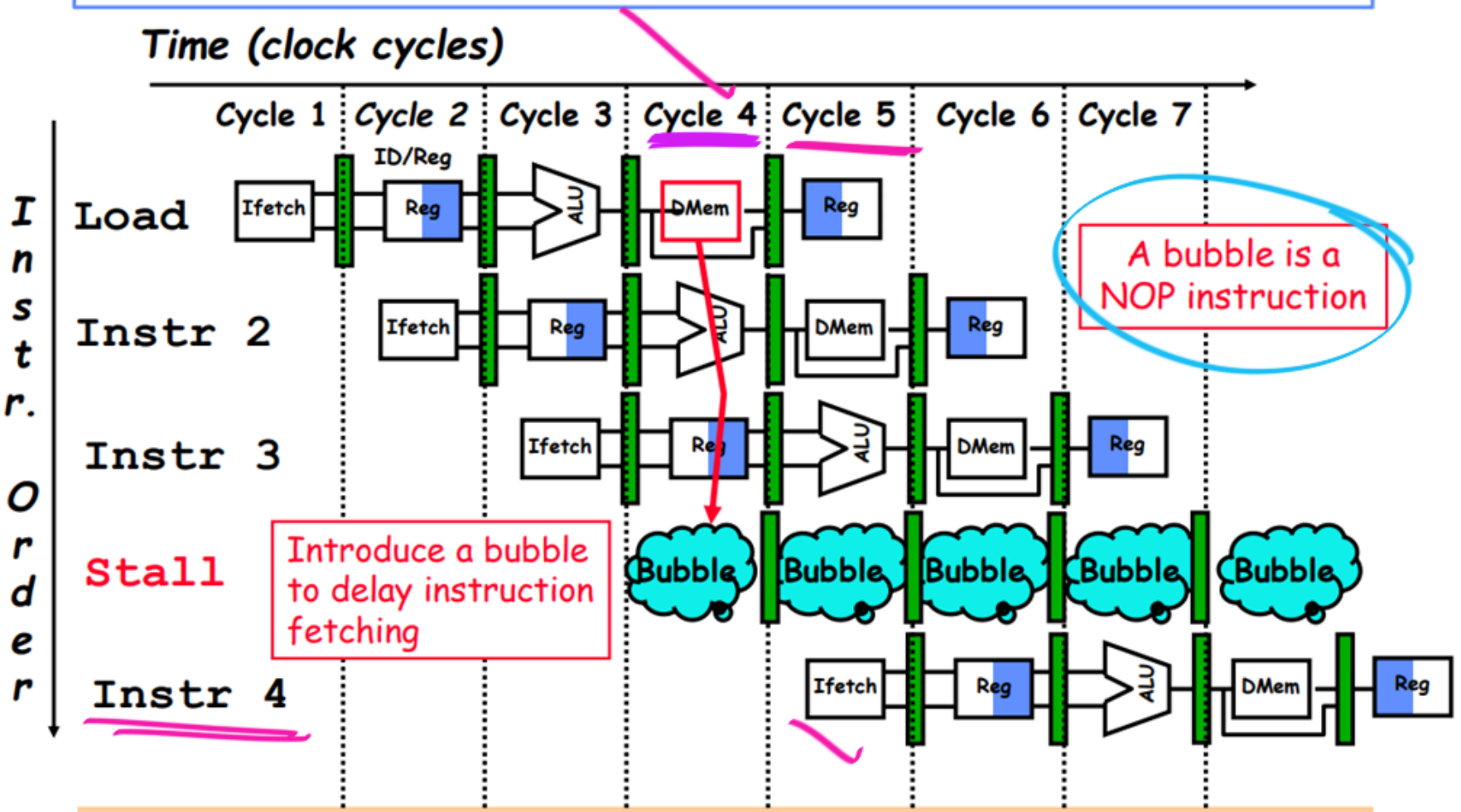
2. Stall / Bubble / Nop

Resolving structural hazards

- Problem
  - Attempt to use the same hardware resource (Memory) by two different instructions during the same cycle
- Solution 1: Wait
  - Must detect the hazard
  - Must have mechanism to delay (stall) instruction access to resource (Introduce bubble / NOP)
  - Serious: hazard cannot be ignored
- Solution 2: Redesign the pipeline
  - Add more hardware to eliminate the structural hazard
  - In our example: use two memories with two memory ports
    - Instruction Memory
    - Data Memory

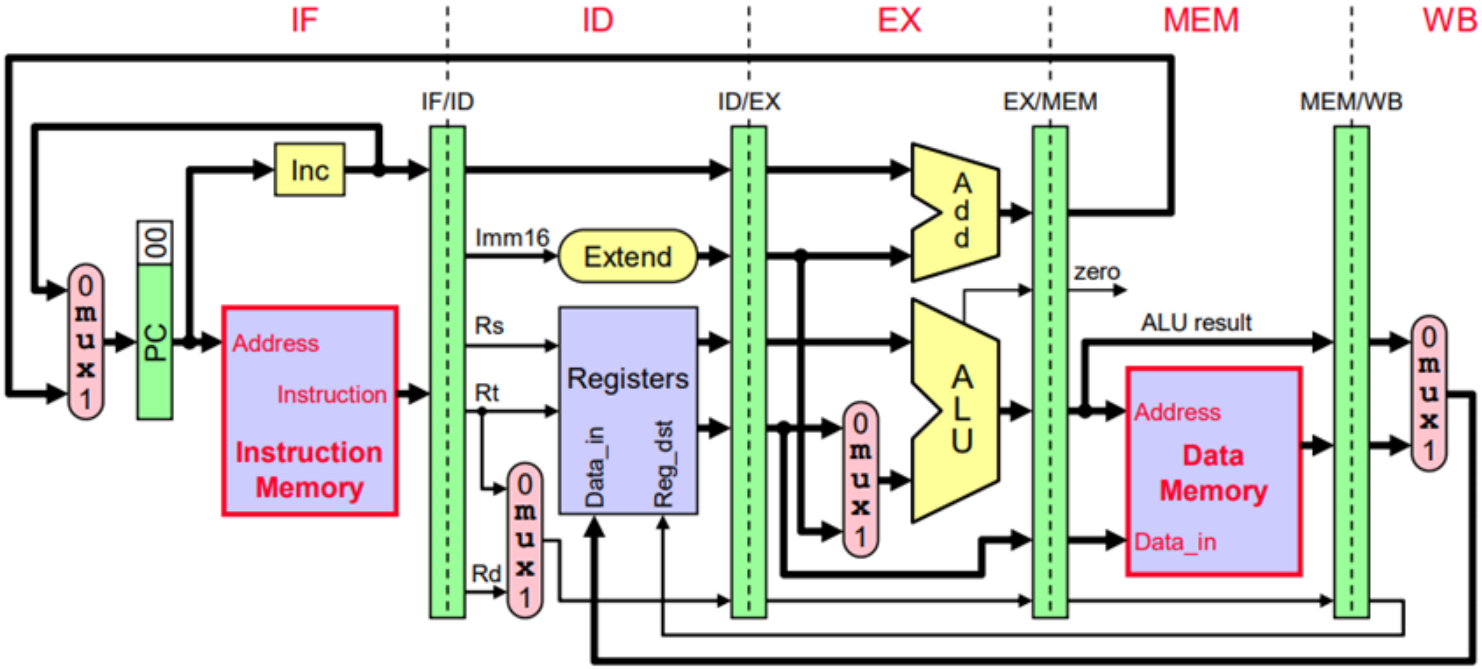
} Can be implemented as caches

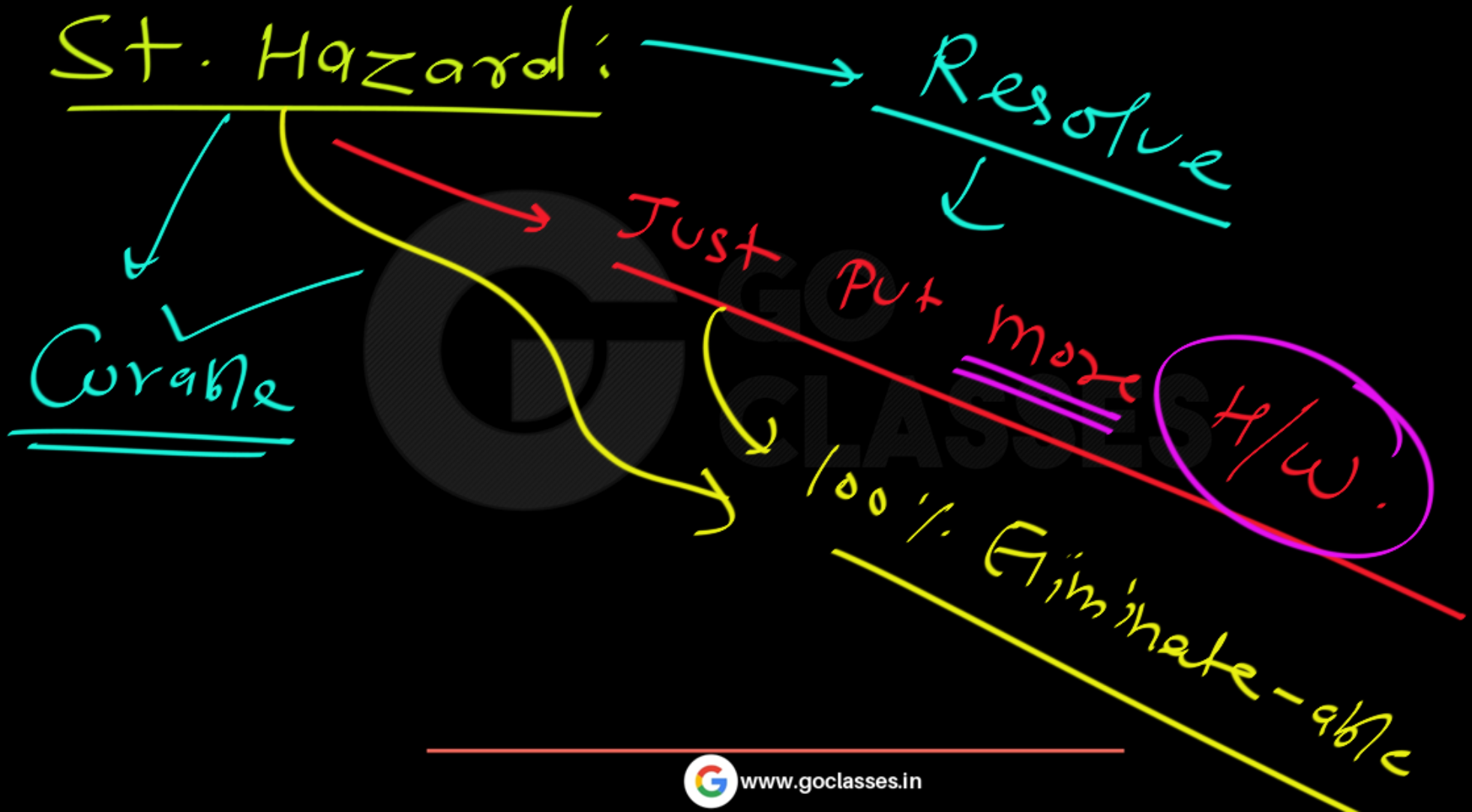
Solution 1 : Detect Structural Hazard and Delay



Solution 2: Add More Hardware (Use Instruction and data memory)

- Eliminate structural hazard at design time
- Use two separate memories with two memory ports
  - Instruction and data memories can be implemented as caches



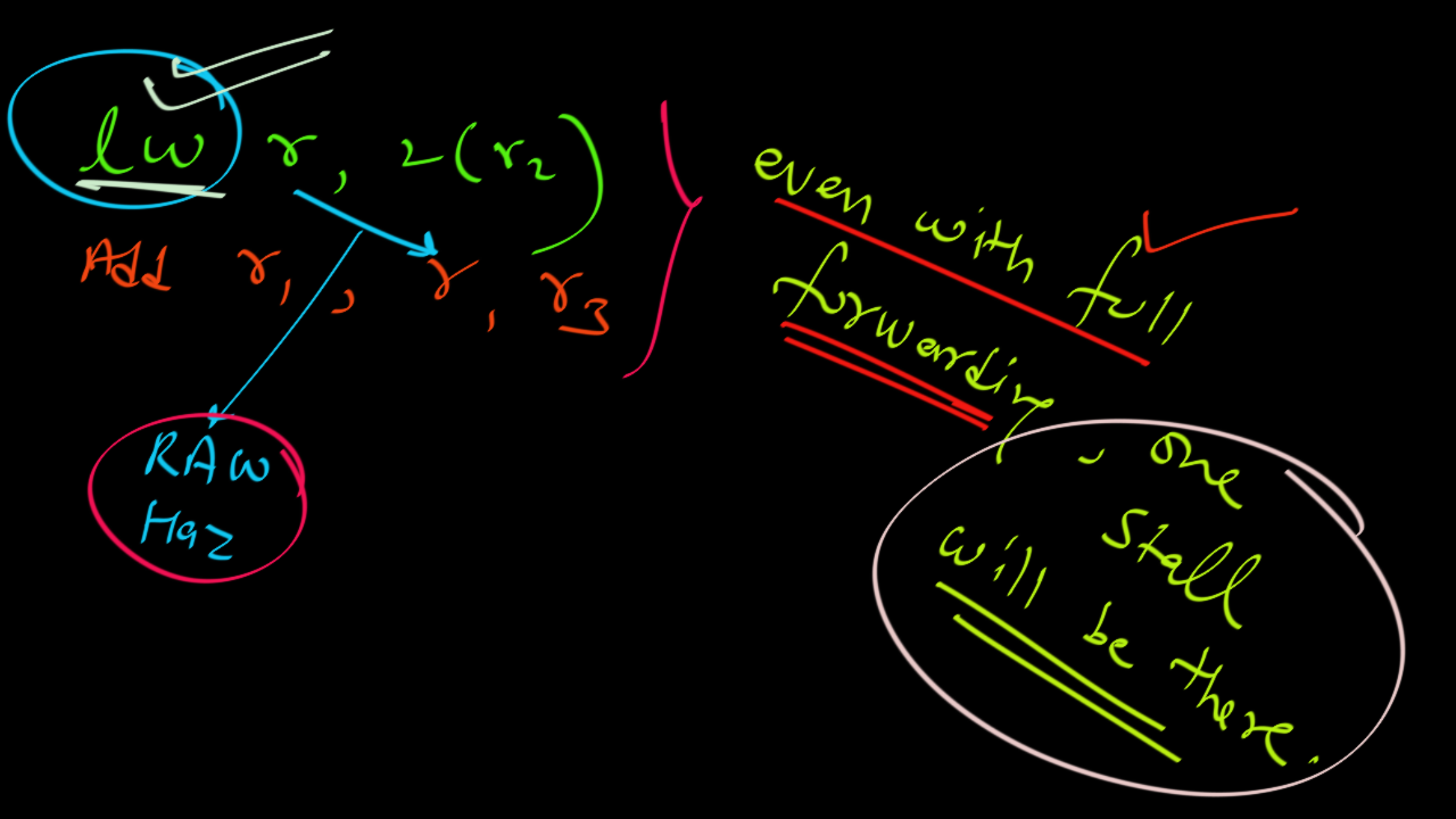

St. Hazard:
Resolve
Curable
Just Put more H/W
100% Eliminate-able

Stall Due to Data Hazard : → 100% Avoidable ? No

WAW Haz
WAR Haz
} 100% Eliminate-able by Reg. Renaming (SSA tech.)

RAW Haz → can minimize using forwarding

NoT 100% Eliminate-able


lw r1, 2(r2)
Add r1, r3, r3
RAW Haz
even with full forwarding, one stall will be there.


So far :
RAW Hazard
NOT 100% Eliminate-able
Real Villain in pipeline so far

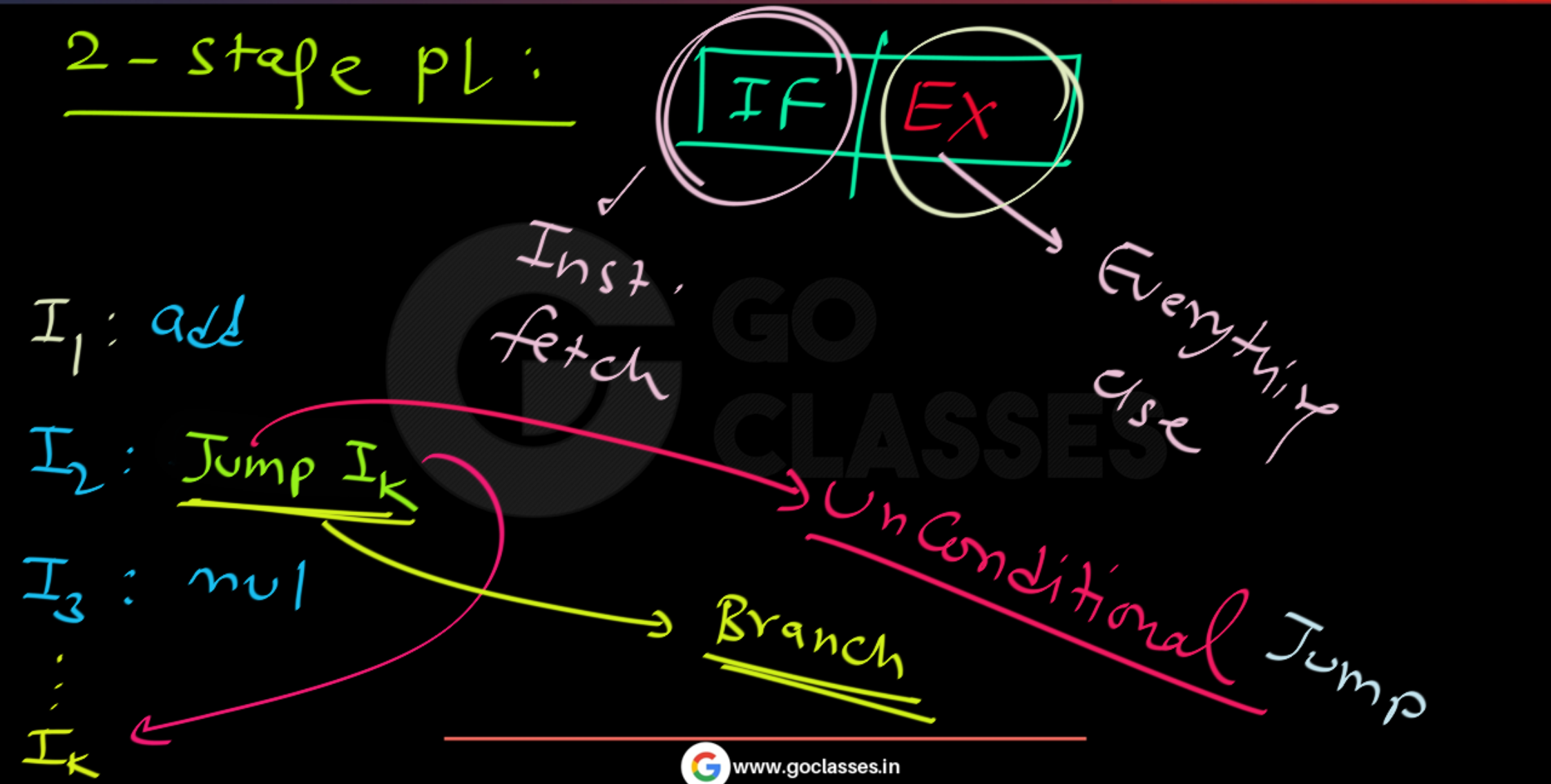
Now;

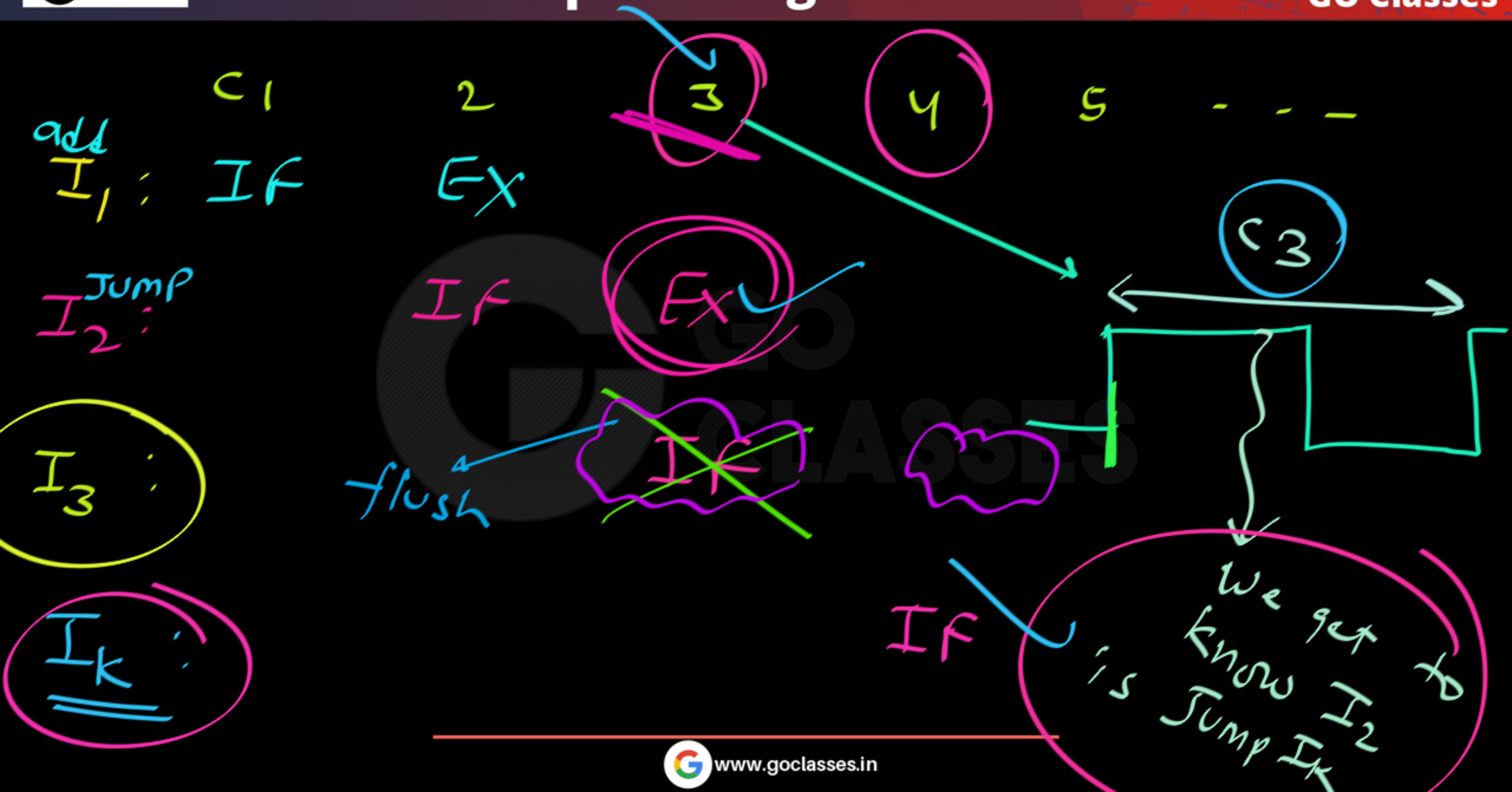
Next Real Villain in Pipeline;

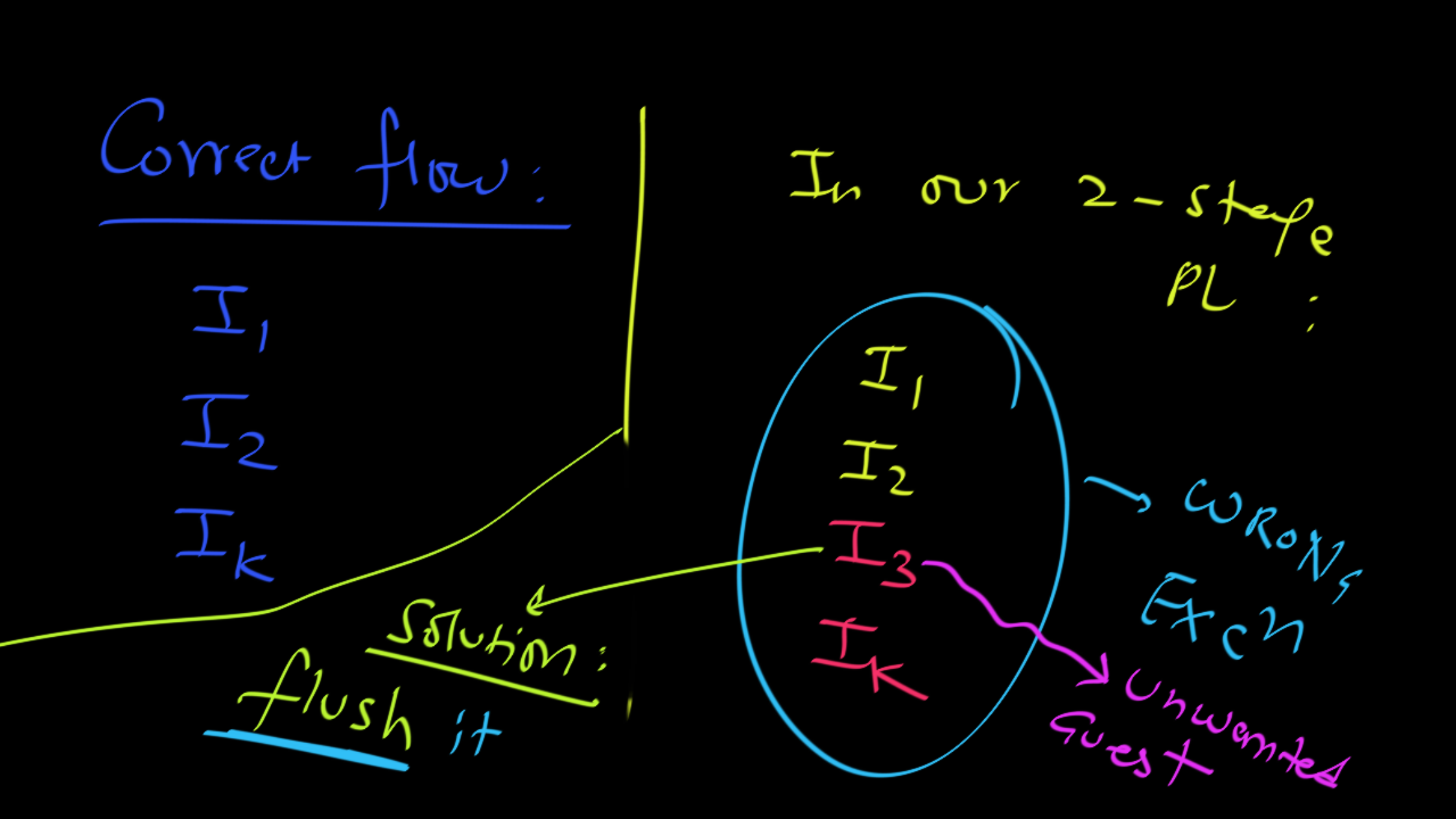
Control Hazards (Not 100% Eliminate-able)

Next Topic:

Control Hazards


2 - stage pl :
IF
Ex
Inst. fetch
Everything else
I1 : add
I2 : Jump Ik
I3 : mul
Ik
Unconditional Jump
Branch


C1
2
3
4
5
- - -
add
I1 : IF
EX
I2 : Jump
IF
EX
C3
I3 :
flush
IF
Ik :
IF
We get to know I2 is Jump Ik


Correct flow:
I_1
I_2
I_k
Solution:
flush it
In our 2-stage PL:
I_1
I_2
I_3
I_k
wrong fetch
unwanted guest

8.3.1 Unconditional Branches

Figure 8.8 shows a sequence of instructions being executed in a two-stage pipeline. Instructions I_1 to I_3 are stored at successive memory addresses, and I_2 is a branch instruction. Let the branch target be instruction I_k. In clock cycle 3, the fetch operation for instruction I_3 is in progress at the same time that the branch instruction is being decoded and the target address computed. In clock cycle 4, the processor must discard I_3, which has been incorrectly fetched, and fetch instruction I_k. In the meantime, the hardware unit responsible for the Execute (E) step must be told to do nothing during that clock period. Thus, the pipeline is stalled for one clock cycle.

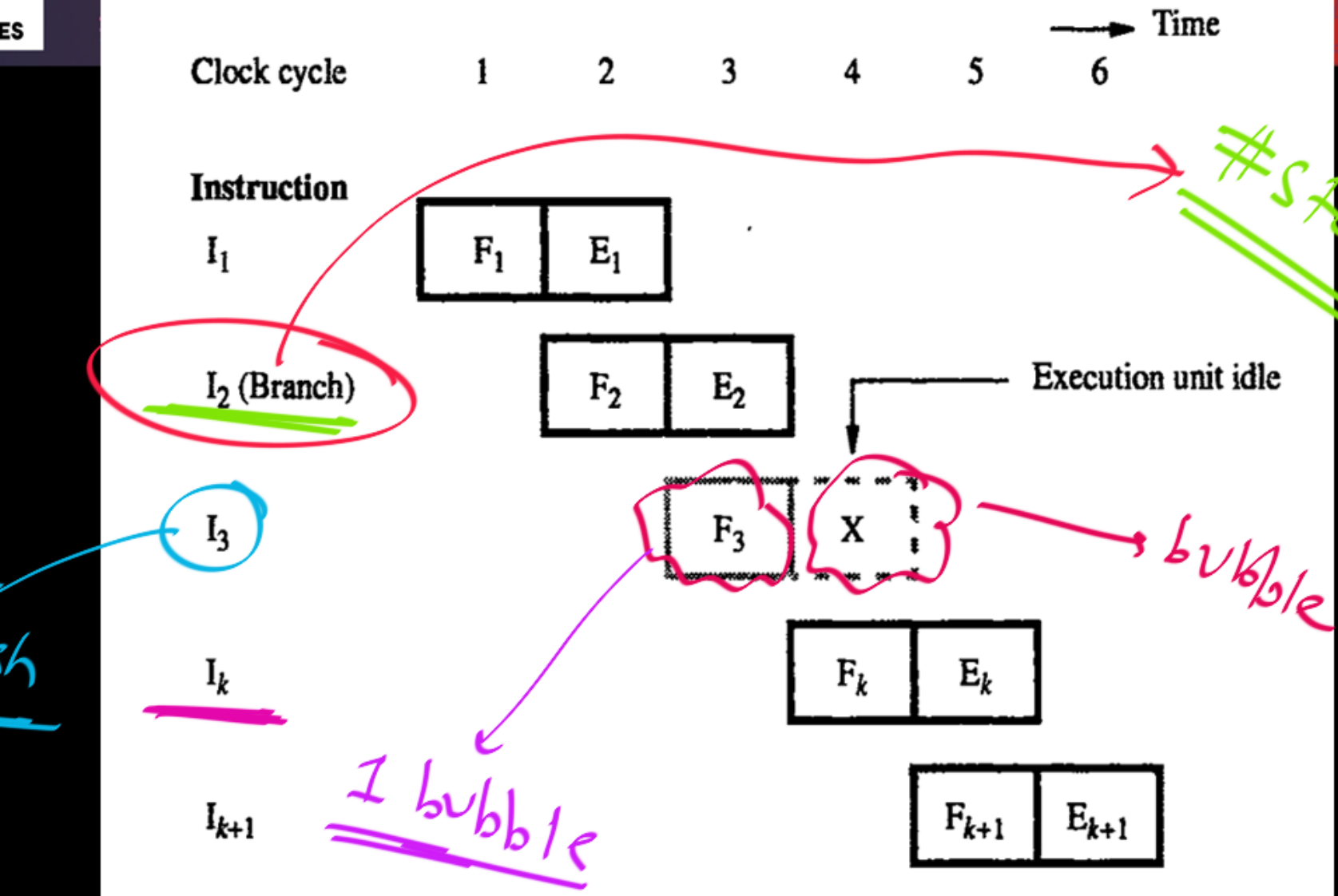


Figure 8.8 An idle cycle caused by a branch instruction.

I_1 :

I_2 : Branch

I_3 :

.

.

.

I_k :

In our 2-stage PL :

we Get 1 Stall (bubble)

due to Branch Instruction

Why ?? because Branch was resolved in 2 stage.

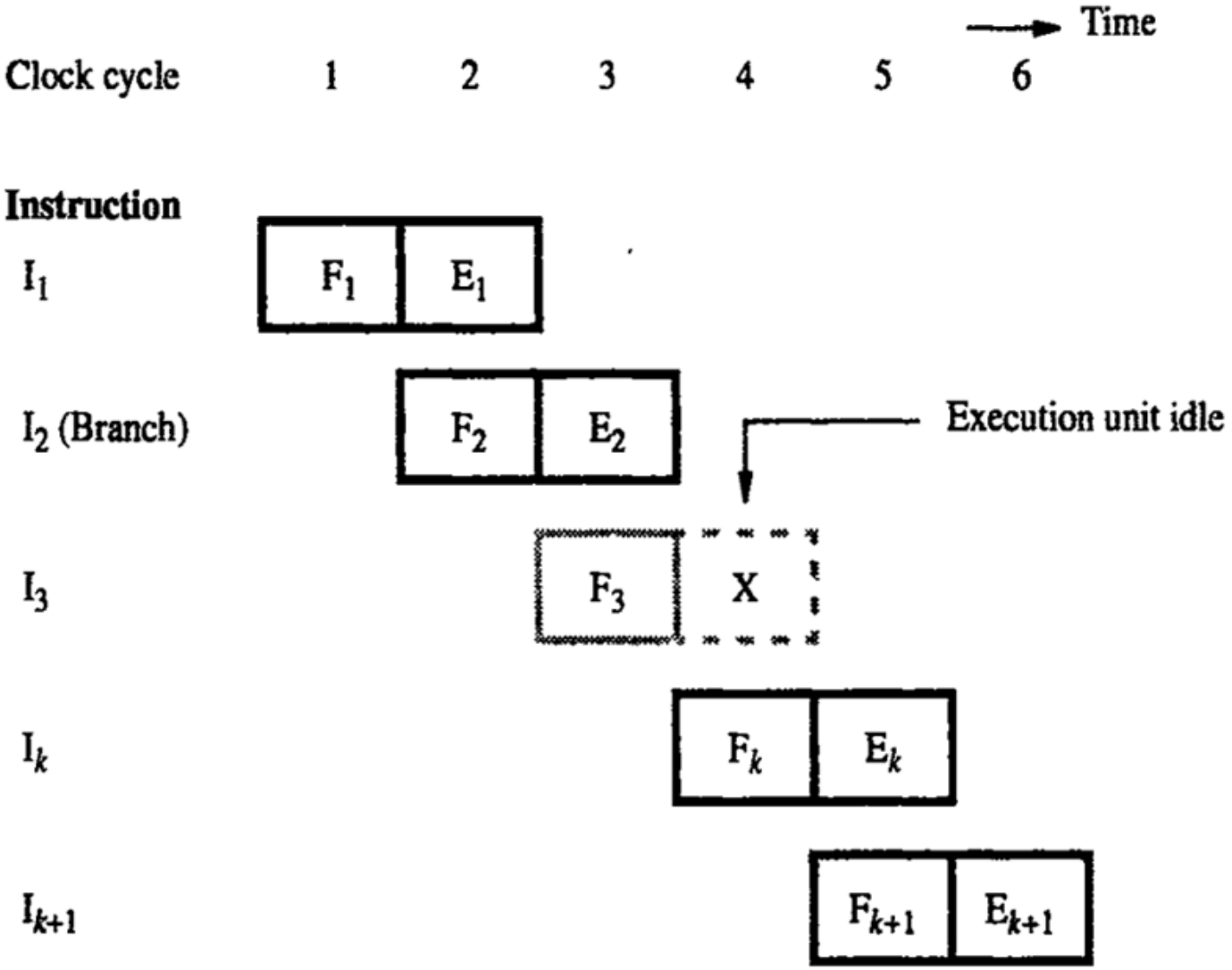
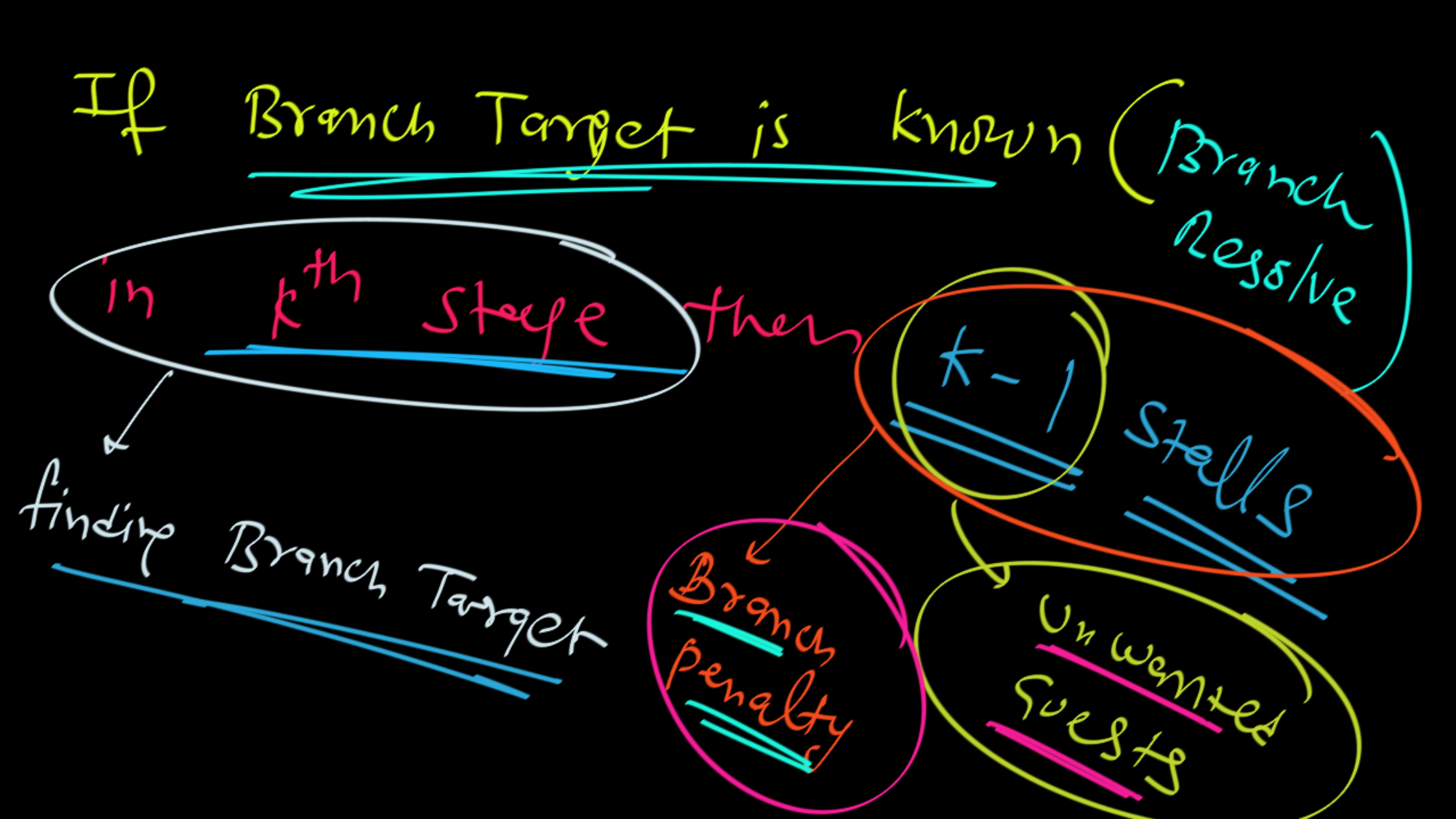


Figure 8.8 An idle cycle caused by a branch instruction.


If Branch Target is known (Branch Resolve
in kth Stage then
finding Branch Target
k-1 Stalls
Branch penalty
Unwanted Fetch

4-Stage PL : If ID WB

Branch Resolve in Ex Stage

Means In Ex stage we find the Target Address of the next Inst that should be executed.

4-Stage PL : IF ID EX WB

Branch penalty : 2

If Branch Resolve in Ex Stage

Target Address

I_1 : Branch Inst. Jump I_{10}

	C1	C2	C3	C4	C5	. .
I_1	IF	ID	EX	WB		
I_2	IF					
I_3		IF				

Unwanted Guest

Flush

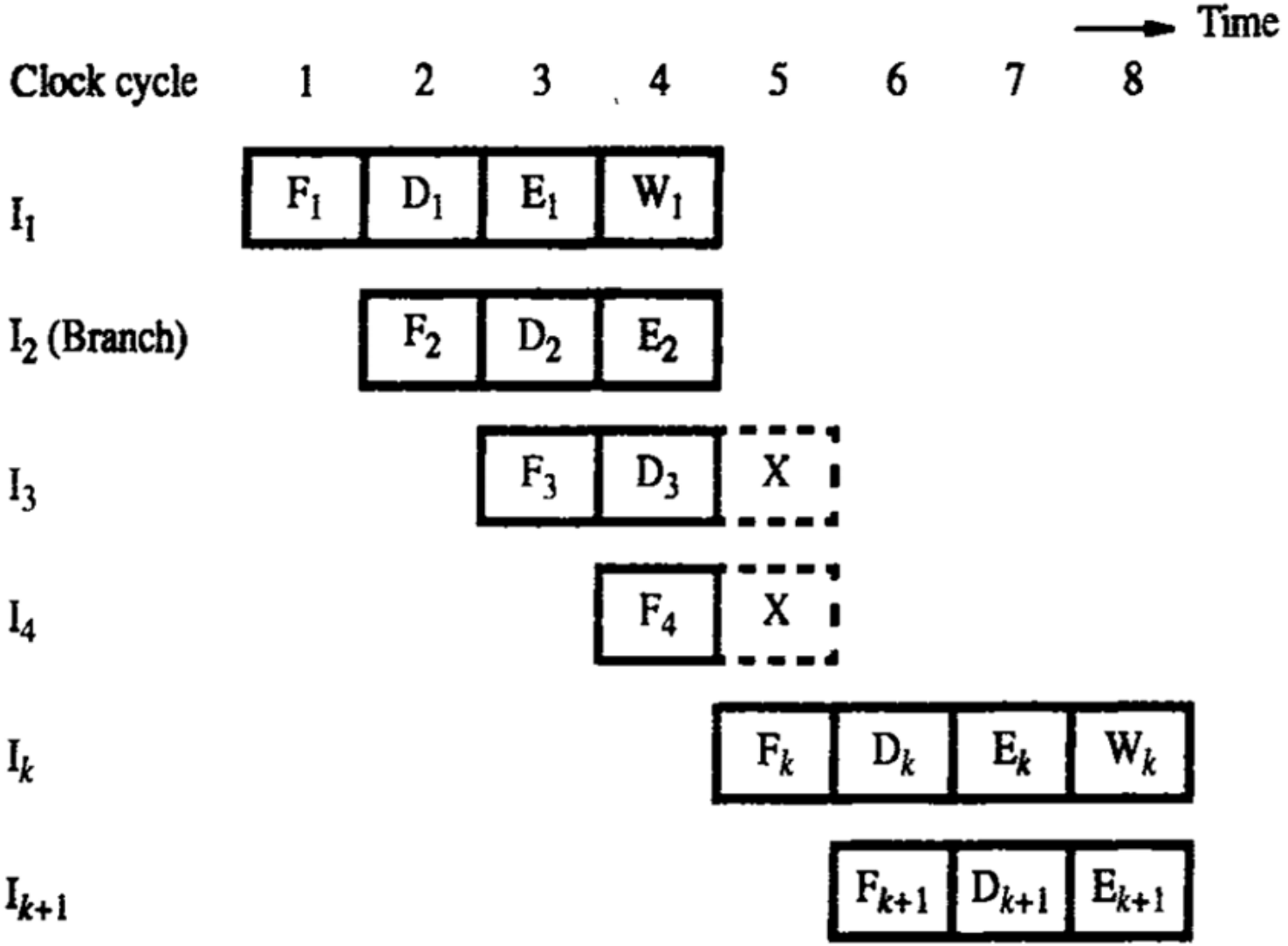
2 stalls in PL

4-Stage PL : IF ID EX WB

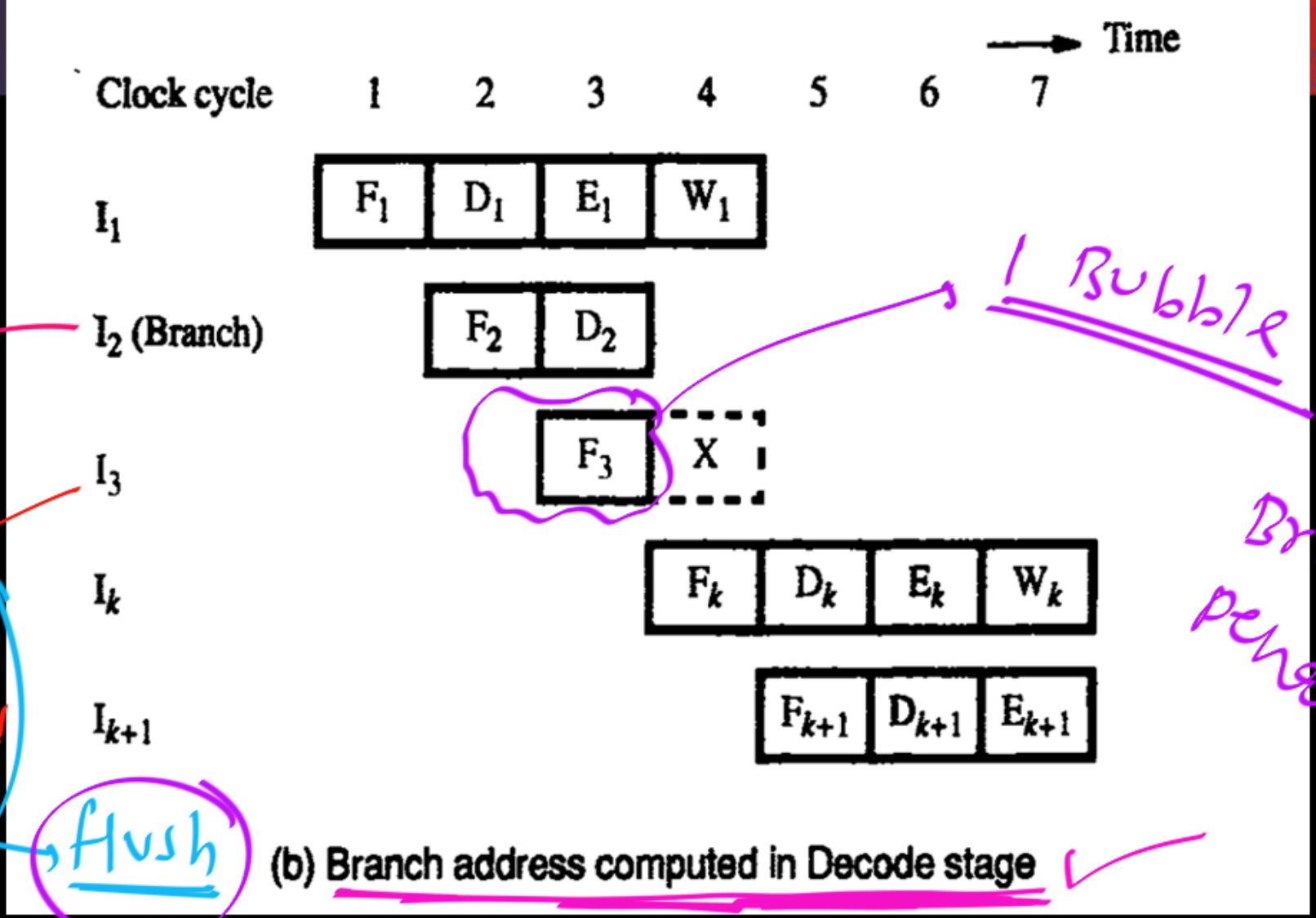
If Branch Resolve in EX Stage

then Branch penalty = 2 stalls

The time lost as a result of a branch instruction is often referred to as the *branch penalty*. In Figure 8.8, the branch penalty is one clock cycle. For a longer pipeline, the branch penalty may be higher. For example, Figure 8.9*a* shows the effect of a branch instruction on a four-stage pipeline. We have assumed that the branch address is computed in step E_2. Instructions I_3 and I_4 must be discarded, and the target instruction, I_k, is fetched in clock cycle 5. Thus, the branch penalty is two clock cycles.



(a) Branch address computed in Execute stage



(b) Branch address computed in Decode stage

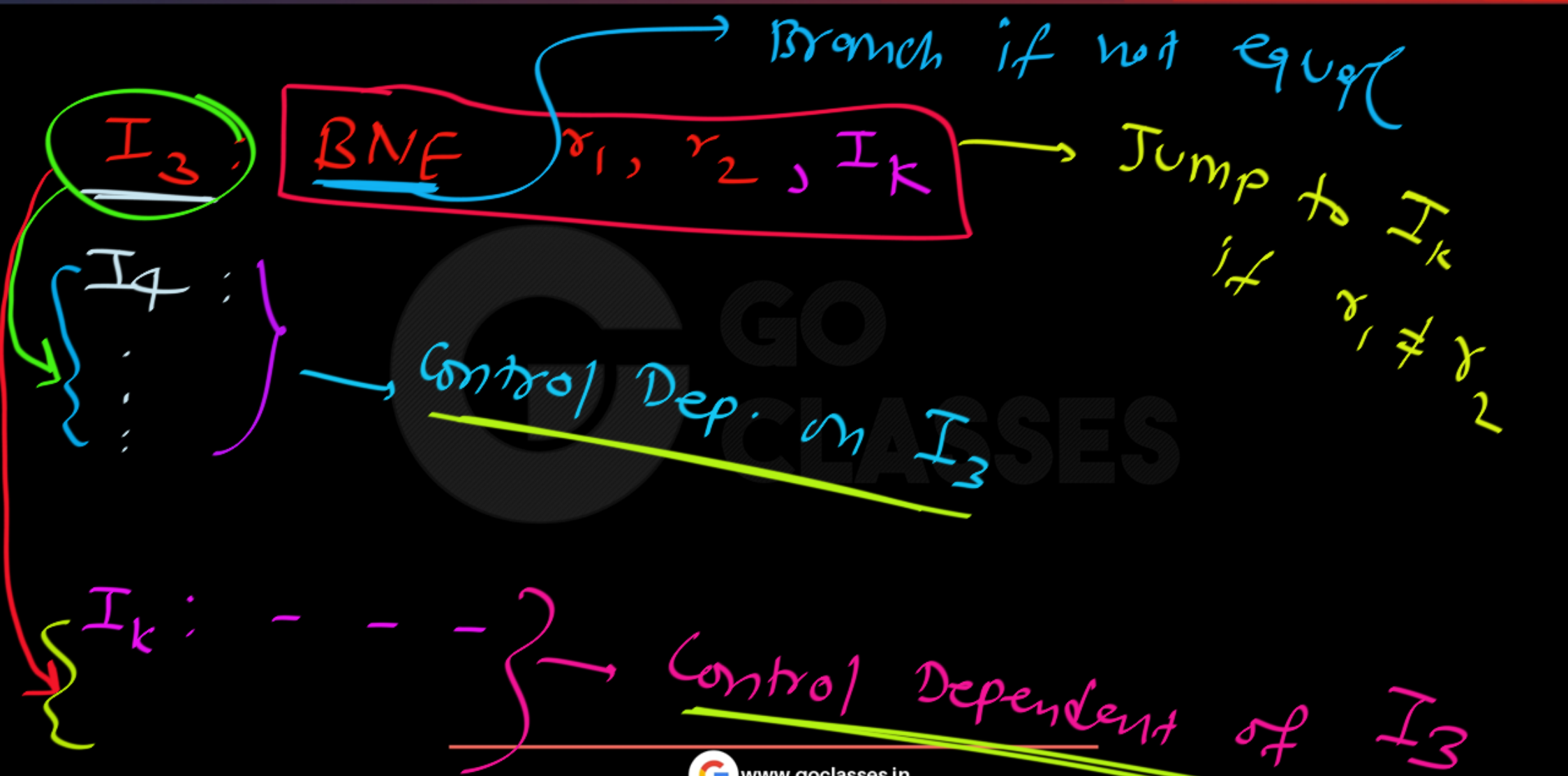
Jump I_k

1 Bubble

Branch penalty = 1

one unwanted Guest

flush


Branch if not equal
I_3 : BNE r_1, r_2, I_k
Jump to I_k if $r_1 \neq r_2$
I_4 :
Control Dep. on I_3
I_k : - - -
Control Dependent of I_3

BEQ means: Jump if Equal

BNE means: Jump if Not Equal

CONTROL DEPENDENCIES


```
        ADD   R1, R1, R2
        BEQ   R1, R3, Label
        ADD   R2, R3, R4
        SUB   R5, R6, R8
        :
Label:
        MUL   R5, R6, R8
        :
        :
```

Control Dependence

Next Topic:

Solutions

Solution 1:

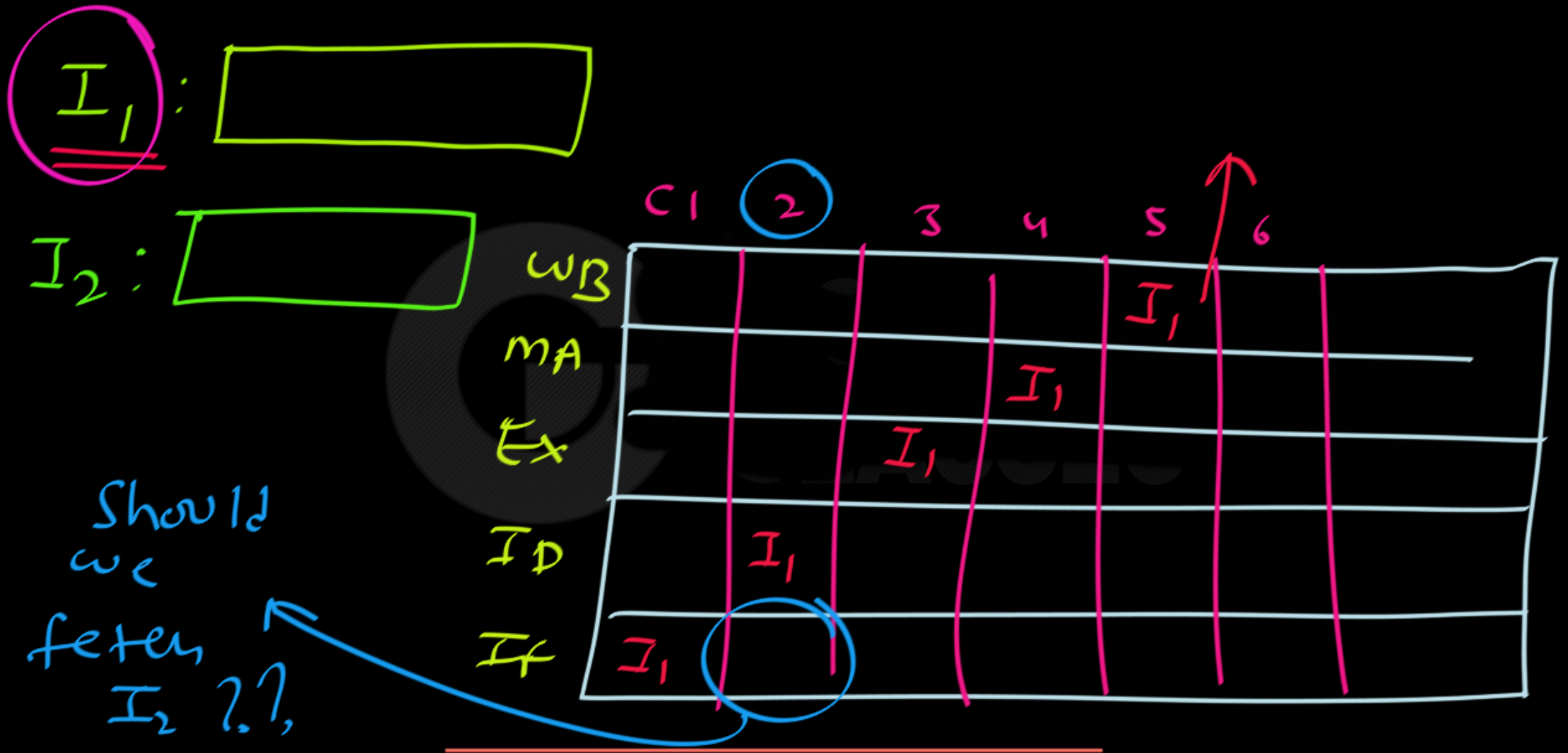
Never fetch wrong Inst.

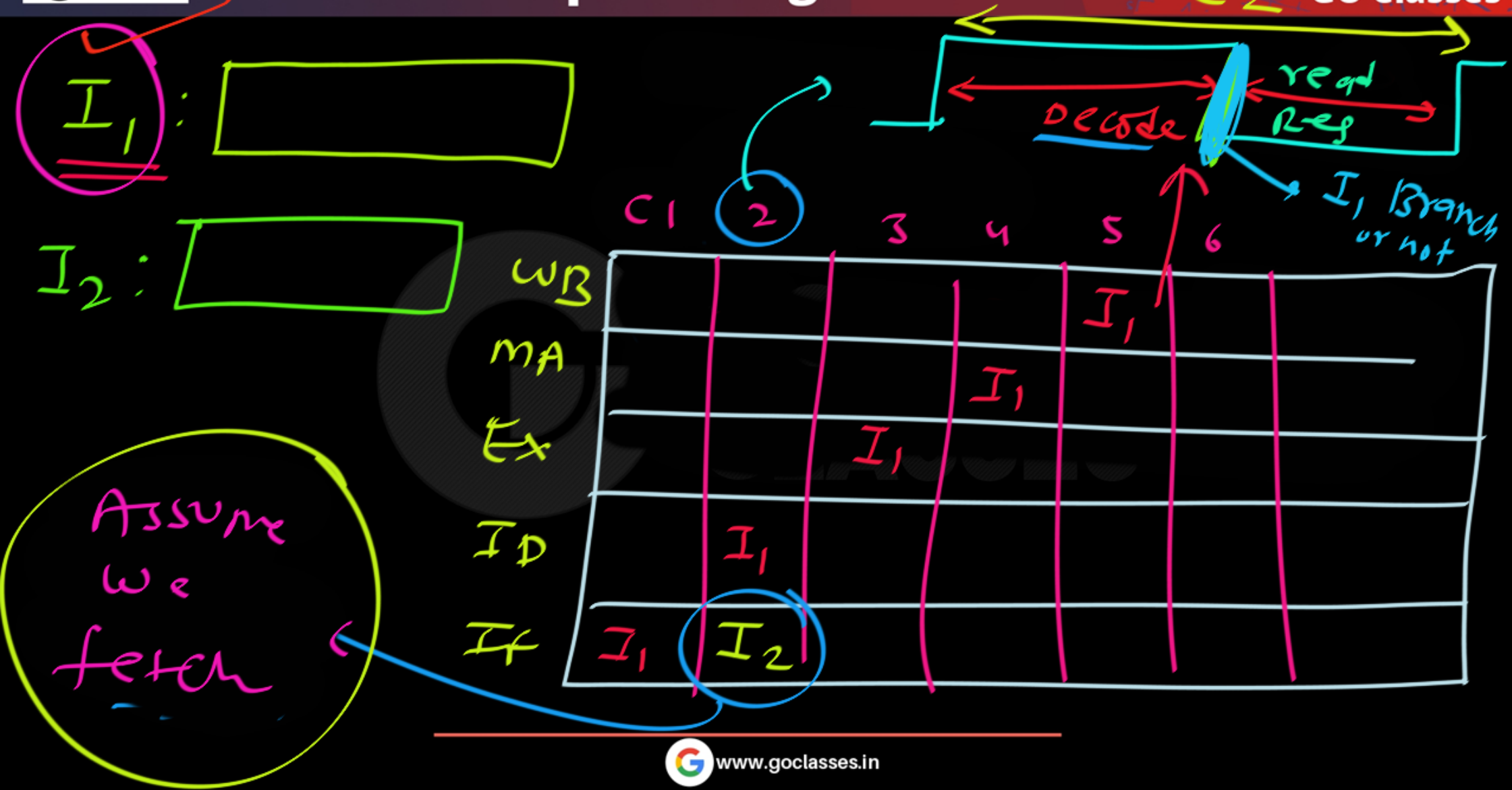
Solution 1:

Stop fetching until the branch target is resolved

Sol 7

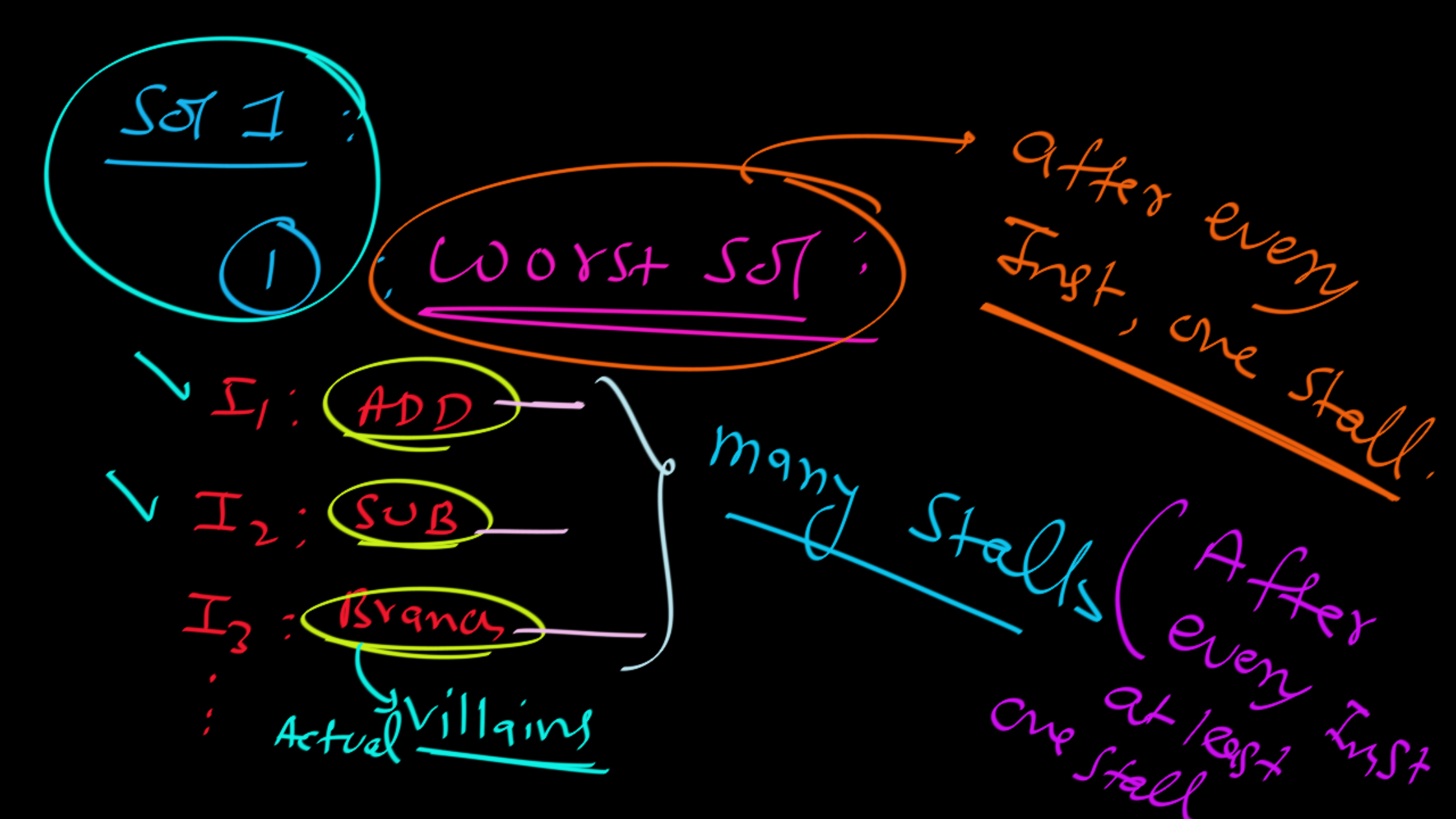
Stop fetching until we know the Target address of the Branch


I1 :
I2 :
C1
2
3
4
5
6
WB
MA
Ex
ID
If
I1
Should we fetch I2 ??


I1 :
I2 :
C1
2
3
4
5
6
WB
MA
Ex
ID
IF
I1
I2
Decode
read Reg
I1 Branch or not
Assume we fetch

Stop fetching :

(1) After "EVERY" Inst, Stop fetching until in ID stage we set to know if the Inst is Branch or not.

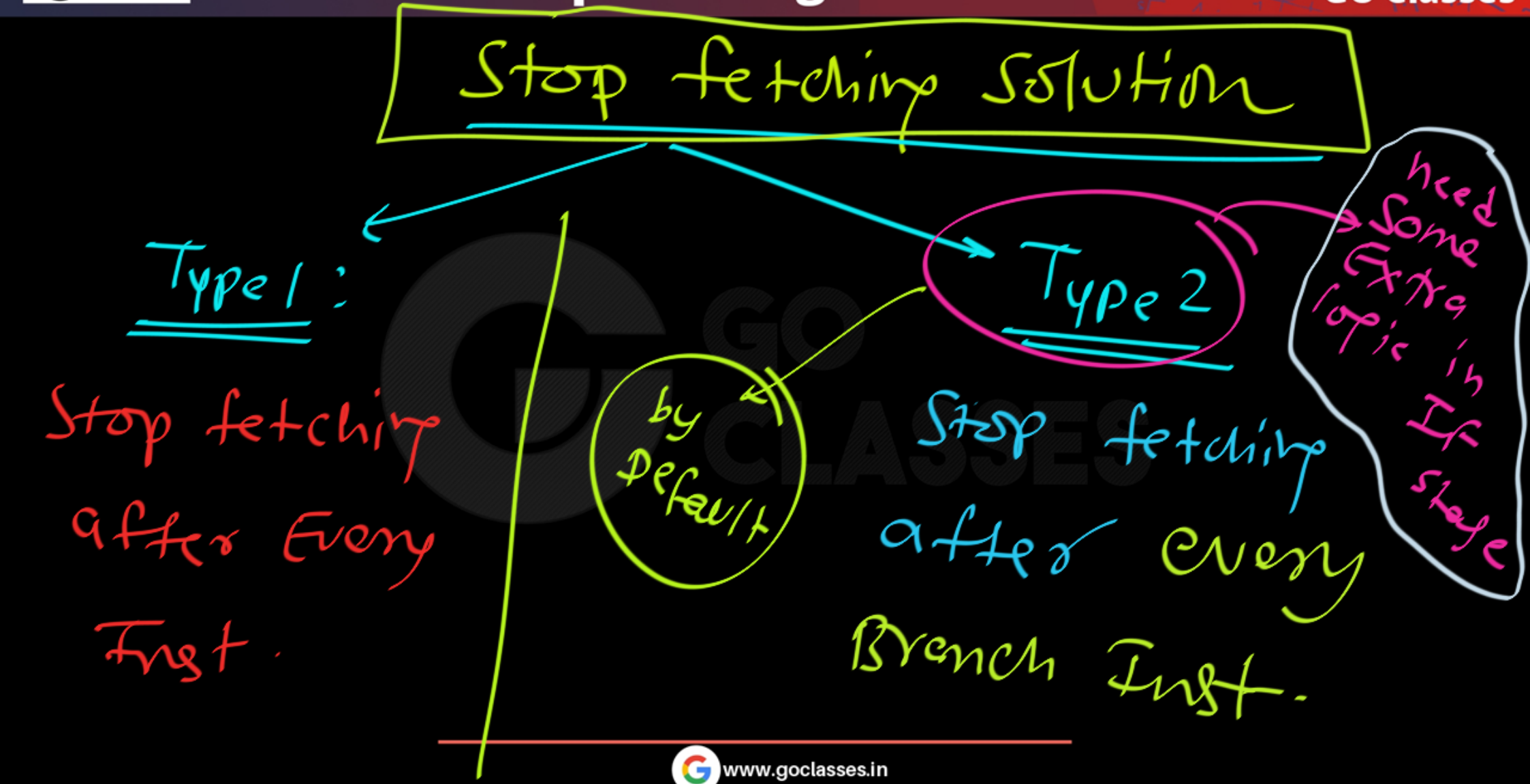

Set 1 :
1
Worst Set :
after every Inst, one stall.
I_1 : ADD
I_2 : SUB
I_3 : Branch
:
Actual Villains
many Stalls
After every Inst at least one stall

Stop fetching.

2) IF Stage has extra HW Just to find out if inst is Branch Inst or Not.

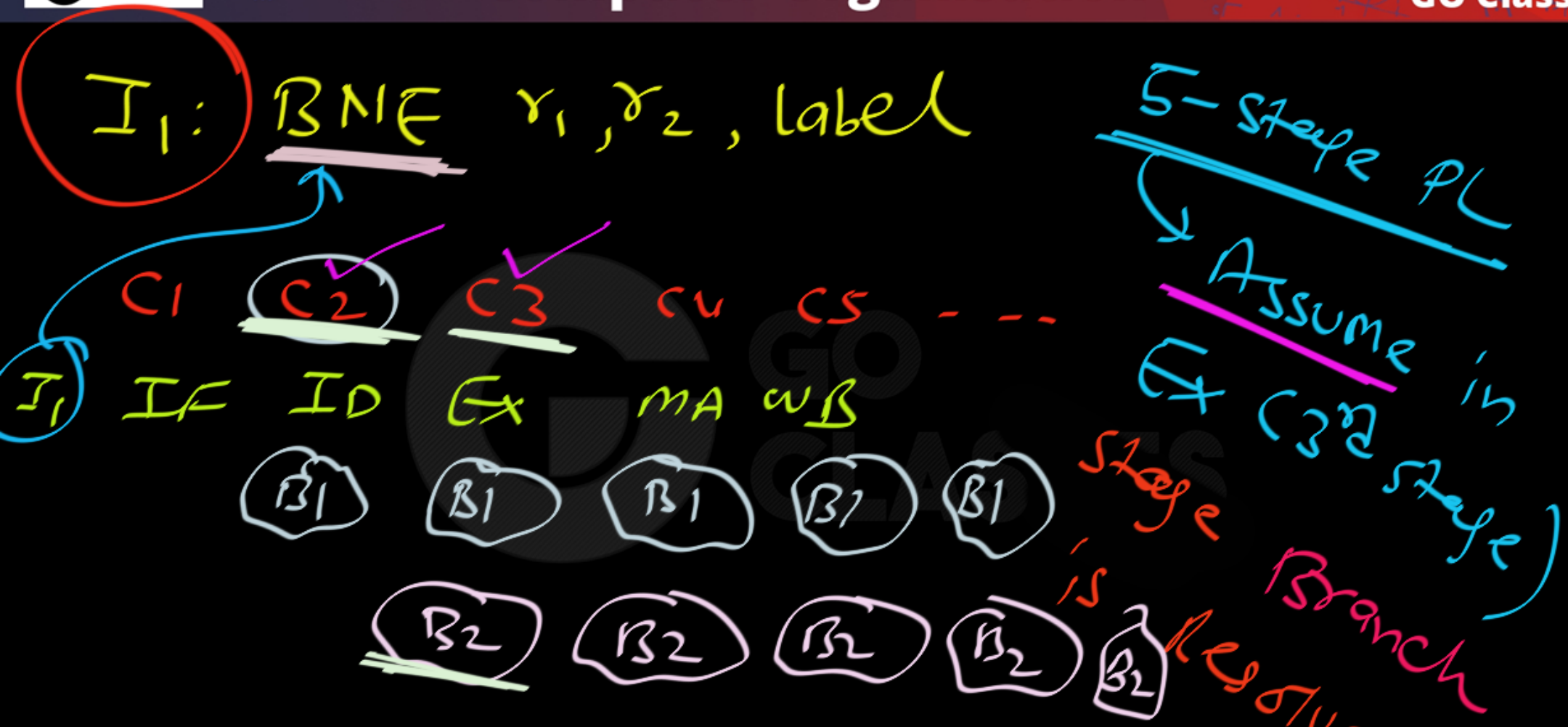
Sol 1

(2) :— "Stop fetching" only after 'every' Branch inst.

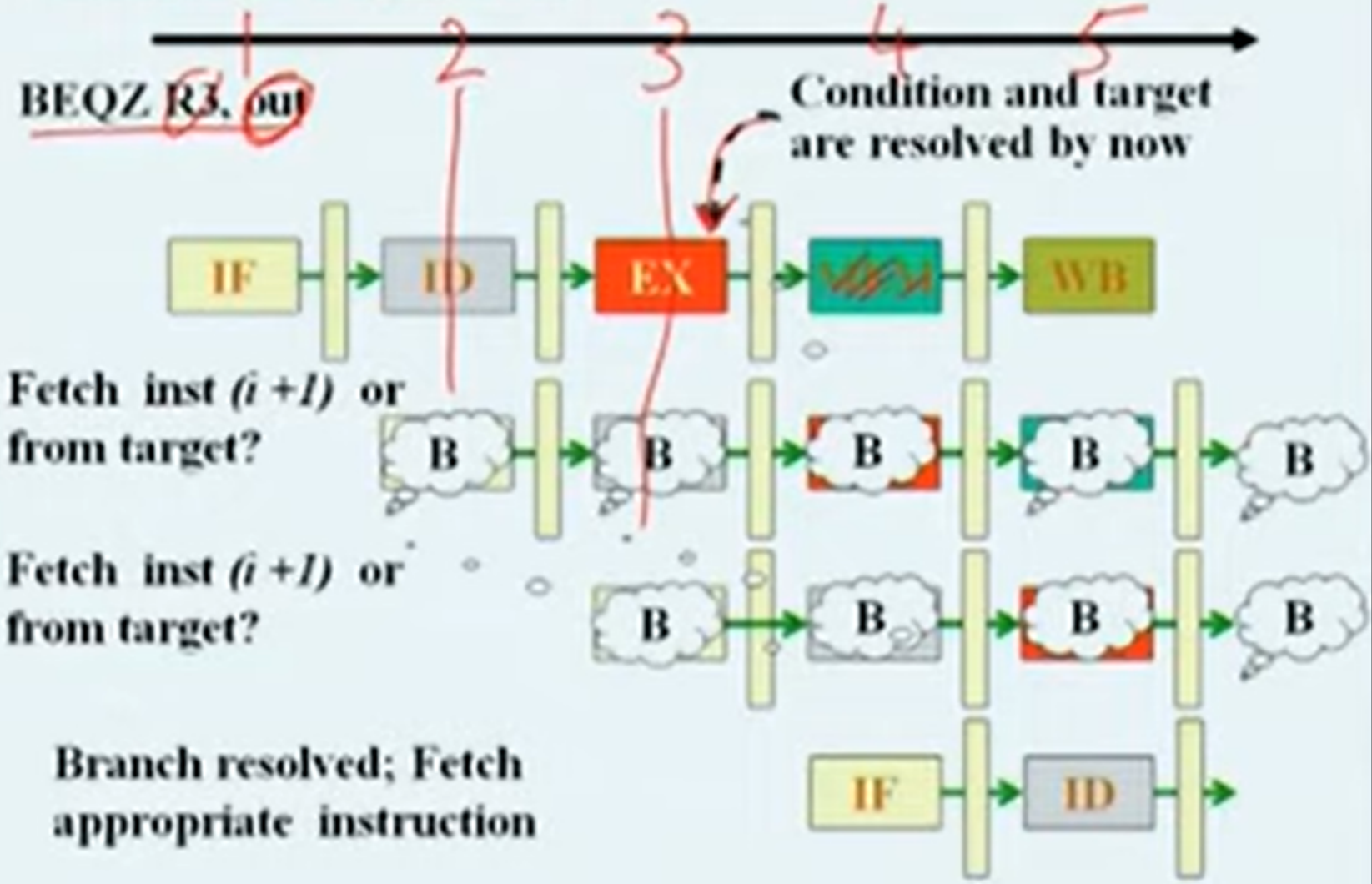

Stop fetching Solution
Type 1 :
Stop fetching after Every Inst.
by Default
Type 2
Stop fetching after every Branch Inst.
need some extra logic in IF stage

Conclusion :

Solution 1 : Stop fetching "After every branch inst" until Target address is known.


I1: BNE R1, R2, label
5-stage PL
Assume in EX (C3) stage
C1 C2 C3 C4 C5 ---
I1 IF ID EX MA WB
B1 B1 B1 B1 B1
B2 B2 B2 B2 B2
stage Branch is resolved.

Control Hazards



Control Hazards

- Observation: Since the branch is resolved only in the EX stage, there must be 2 stall cycles after every conditional branch instruction

Solution 2:

Static Branch Predication

Solution 1: Stop fetching after every branch inst until Target address is known

Solution 2: Why after "Every" branch Stop fetching ??

I_4: BEQ r_1, r_2, Label

I_5:

:

:

Label :

Assume $r_1 \neq r_2$

Correct Exn

I_4

I_5

Solution 2 :

keep fetching ; If any problem, we flush ;

Sol 1

Stop fetching after every branch inst : 2 stalls

Better

Sol 2

Keep fetching ; flush if problem any after some branch inst : 2 stalls

Solution 2 : Static prediction

I_1 : Branch inst

I_2
I_3
} fall through path

.....

I_k
.....
} Taken path

Static prediction

prediction :

Not - Taken

Solution 2 : Static prediction

I_1 : Branch inst

I_2
I_3
} fall through path

I_k :
} Taken path

Prediction :

Prediction

Taken

If Prediction fails :

then flush

k - 1 stalls

Branch resolved in k^{th} stage

Static Branch Predication

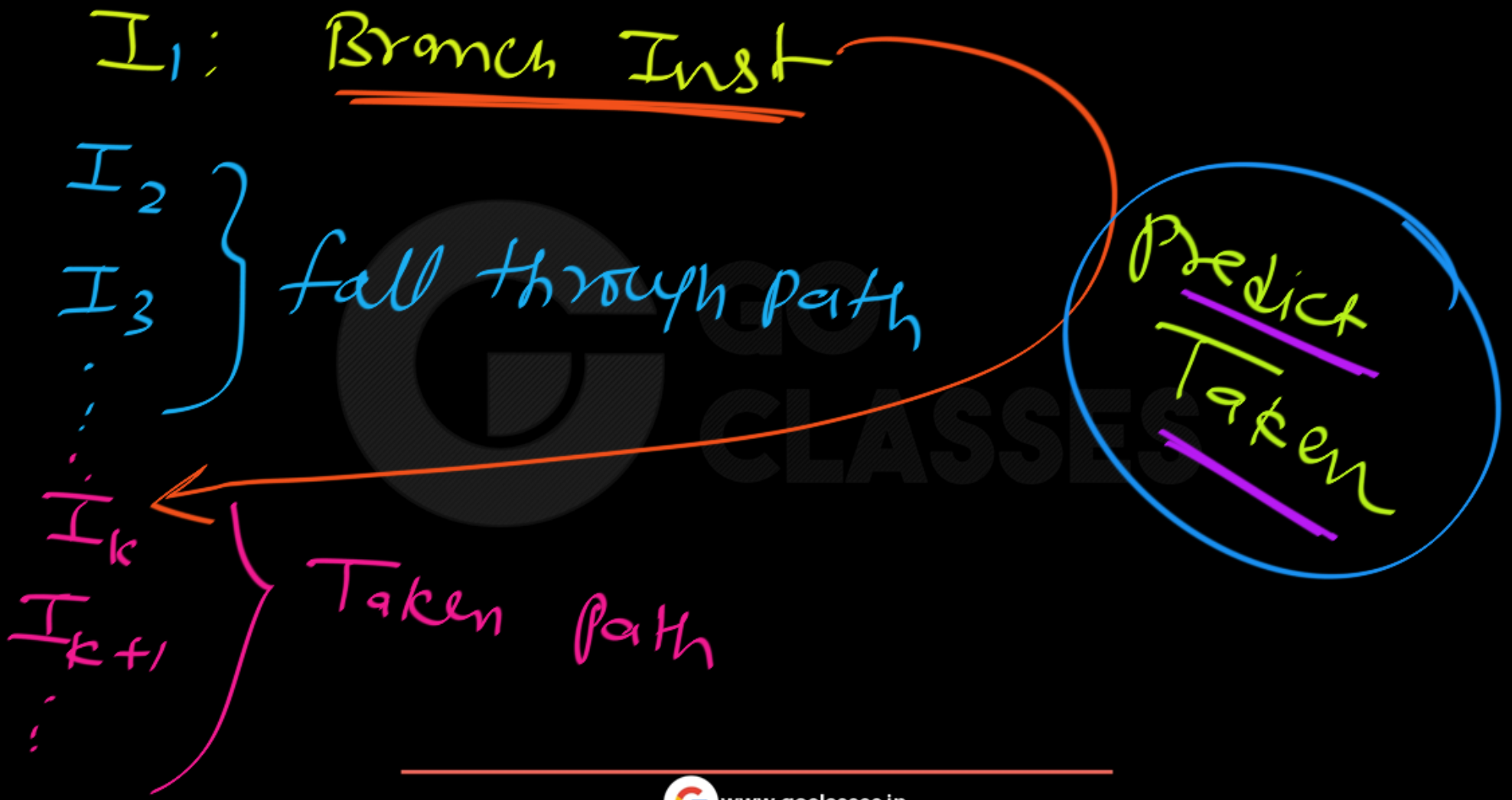
1. Predict Not Taken
2. Predict Taken

Predict Taken Scheme :

by Default Assume

(If stage knows inst is branch or not

→ for every branch inst, we assume it will Always go to Taken Path


I1 : Branch Inst
I2
I3
fall through path
Ik
Ik+1
Taken Path
Predict Taken

Predict Taken Scheme :

I_1 : BEQ r_1, r_2, Label

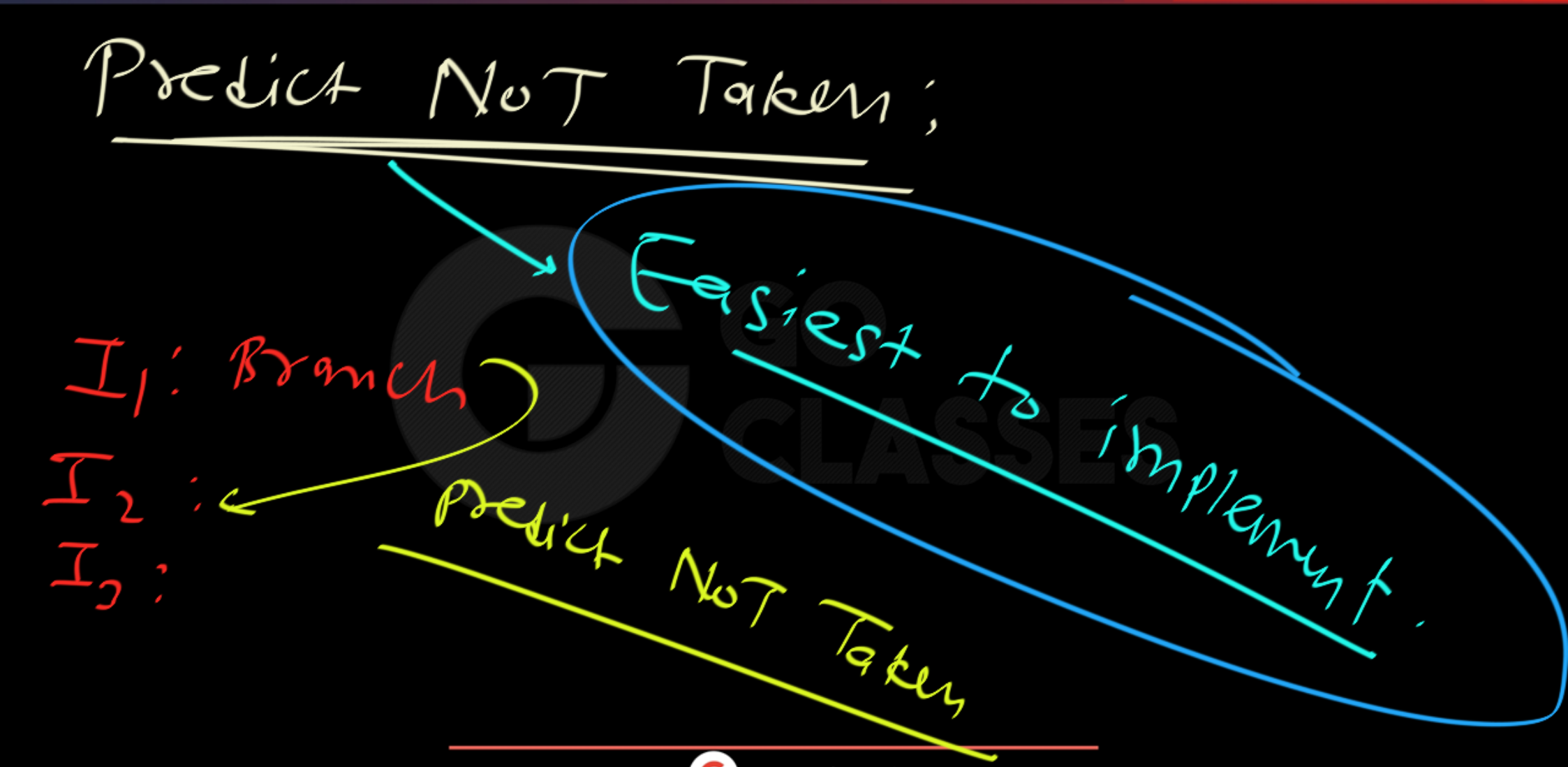
I_2 :

I_3 :

Label : - -

Predict Taken

Just go to Label without checking r_1, r_2


Predict NoT Taken;
Easiest to implement.
I1: Branch
I2 :
I3 :
Predict NoT Taken

Prediction

Playing the odds...

When a RISC machine encounters a branch instruction, there are only two possible outcomes, either the branch is taken, or the branch is not taken. So, if you do not want to wait until the branch instruction completes, you must guess which instruction you will execute next. This is prediction.

There are multiple methods of branch prediction, predict-taken, predict-not-taken, and dynamic-hardware-prediction will be covered below. Predict-taken and predict-not-taken are just flip sides of the same coin. They both assume that more often than not, a program will do the same behavior. With predict-not-taken, the hardware assumes that the next instruction to be executed will be the instruction following the branch instruction in memory. This (possible) next instruction is placed in the pipe and execution begins. If after the branch is finished executing, it is found that we really wanted to take the branch, then the (possible) next instruction that we had been executing is thrown out by the hardware, and we being executing the correct instruction, the branch target.

Prediction

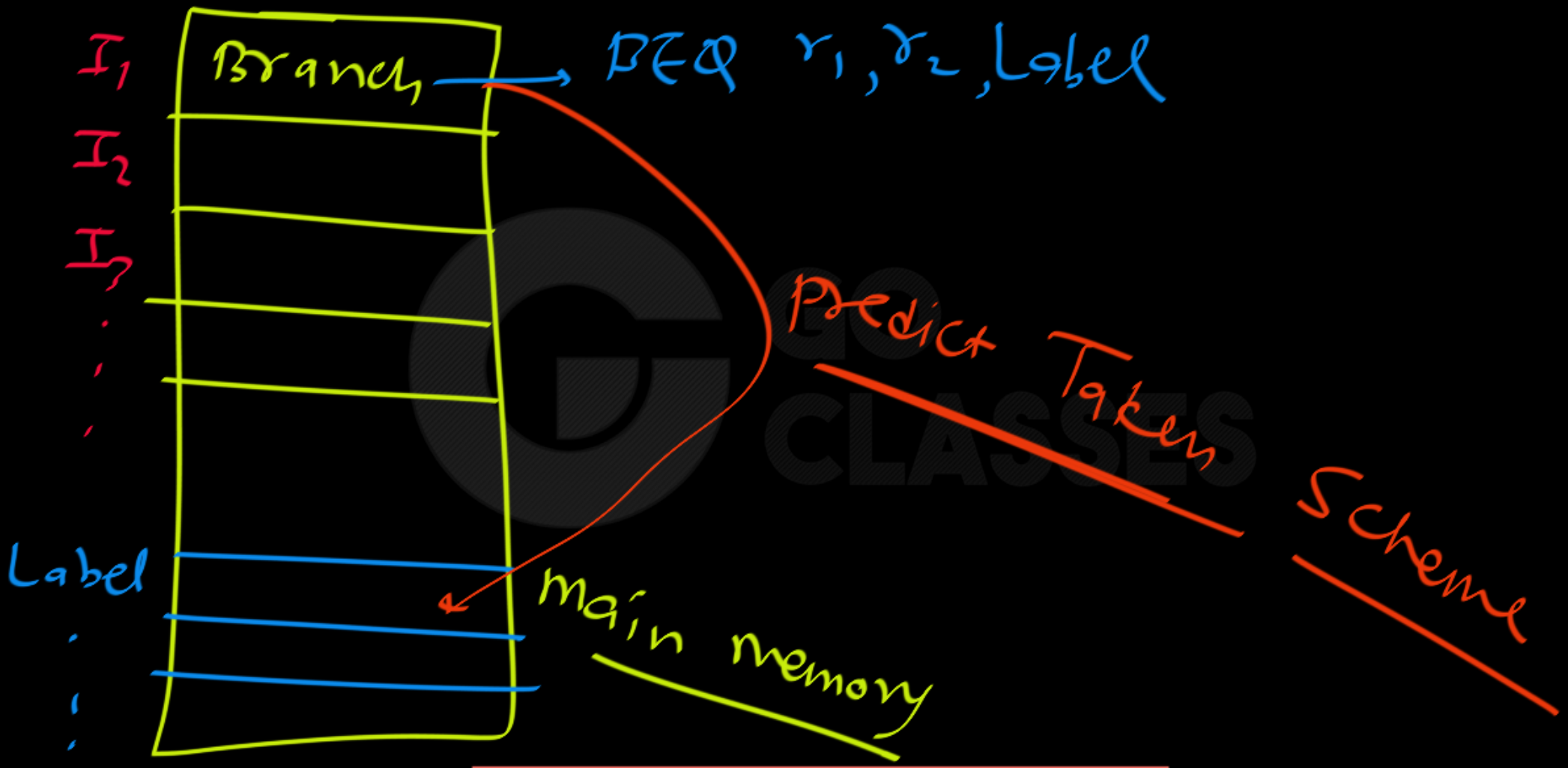
Playing the odds...

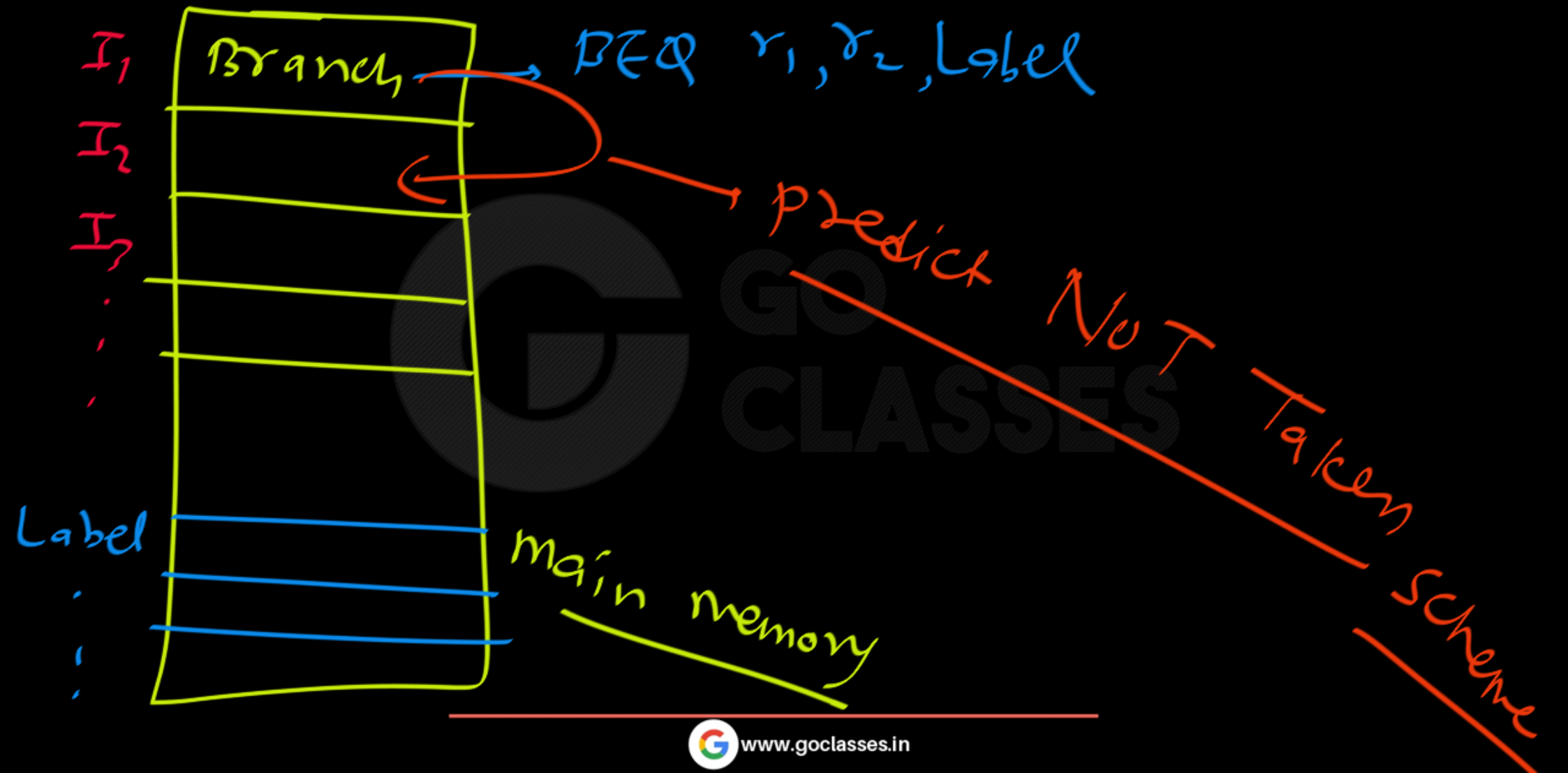
When a RISC machine encounters a branch instruction, there are only two possible outcomes, either the branch is taken, or the branch is not taken. So, if you do not want to wait until the branch instruction completes, you must guess which instruction you will execute next. This is prediction.

There are multiple methods of branch prediction, predict-taken, predict-not-taken, and dynamic-hardware-prediction will be covered below. Predict-taken and predict-not-taken are just flip sides of the same coin. They both assume that more often than not, a program will do the same behavior. With predict-not-taken, the hardware assumes that the next instruction to be executed will be the instruction following the branch instruction in memory. This (possible) next instruction is placed in the pipe and execution begins. If after the branch is finished executing, it is found that we really wanted to take the branch, then the (possible) next instruction that we had been executing is thrown out by the hardware, and we being executing the correct instruction, the branch target.

Next Topic:

Performance Affected By Control Hazards


I_1
I_2
I_3
Branch
BEQ r_1, r_2, Label
Predict Taken Scheme
Label
main memory


I_1
I_2
I_3
Branch
BEQ r_1, r_2, Label
Predict NOT Taken Scheme
Label
main memory

Classical 5-STAGE pipeline CPI ?

If Ex stage, Branch Resolved

3 stage

- 20% OF INSTS ARE BRANCH/JUMP

Solution: Stop fetching after every branch inst.

- SLIGHTLY MORE THAN 50% OF BR/JMP ARE TAKEN

Classical 5-STAGE pipeline CPI ?

- 20% OF INSTS ARE BRANCH/JUMP

In EX stage, Branch resolved → 3 stage

Solution: Stop fetching after every branch inst.

- SLIGHTLY MORE THAN 50% OF BR/JMP ARE TAKEN

Irrelevant

Irrelevant

CPI : Cycles per Inst.

Our Solution : Stop fetching after every Branch Inst until Branch is Resolved.

We Don't Care about Taken or not.

CPI : Cycles per Inst.

Think in terms of Per instruction

20% of all insts are Branch inst

Per inst, How many branch inst?

$\frac{20}{100} = 0.20$

Per inst :

$1 + (0.20)(2)$

2 stalls per branch inst

by every Inst.

Ideal CPI

because in 3 stage branch resolved

Per Ins 7 :

$$\#cycles = 1 + (0.20)(2)$$

$$= 1.4$$

Solution / practice

5-STAGE CPI ?

Branch Resolve : 2nd stage

- 20% OF INSTS ARE BRANCH/JUMP

Solution: Predict NoT_Taken

- SLIGHTLY MORE THAN 50% OF BR/JMP ARE TAKEN

Predict Not Taken; → Prediction

≡ We stop fetching after Taken Branches

≡ for Taken Branches, Stall will be there.

Prediction fails

CPI : = 1.2

$1 + (0.20)(0.50)(2)$

Ideal CPI

Branch insts per inst

Taken branches

2 stalls for every Taken Branch

CONTROL DEPENDENCE QUIZ

- 25% OF ALL INSTRUCTIONS ARE TAKEN BRANCH/JUMP
- 10-STAGE PIPELINE
- CORRECT TARGET FOR BR/JMP COMPUTED IN 6TH STAGE
- EVERYTHING ELSE FLOWS SMOOTHLY

predict NOT-Taken scheme by Default

ACTUAL CPI IS _____

CONTROL DEPENDENCE QUIZ

- 25% OF ALL INSTRUCTIONS ARE TAKEN BRANCH/JUMP
- 10-STAGE PIPELINE
- CORRECT TARGET FOR BR/JMP COMPUTED IN 6TH STAGE
- EVERYTHING ELSE FLOWS SMOOTHLY

#Stalls = 5

solution: Predict Not Taken

ACTUAL CPI IS _____

Sol: predict Not Taken :

$$CPI = 1 + (0.25)(5) = 2.25$$

- 1 → Ideal CPI
- 0.25 → Taken Branches per inst.
- 5 → 5 stall per Taken branch

1.18.3 Pipelining: GATE CSE 2000 | Question: 12 top

https://gateoverflow.in/683

An instruction pipeline has five stages where each stage take 2 nanoseconds and all instruction use all five stages. Branch instructions are not overlapped. i.e., the instruction after the branch is not fetched till the branch instruction is completed. Under ideal conditions,

A. Calculate the average instruction execution time assuming that 20% of all instructions executed are branch instruction. Ignore the fact that some branch instructions may be conditional.
B. If a branch instruction is a conditional branch instruction, the branch need not be taken. If the branch is not taken, the following instructions can be overlapped. When 80% of all branch instructions are conditional branch instructions, and 50% of the conditional branch instructions are such that the branch is taken, calculate the average instruction execution time.

Cycle time : 2 ns

1.18.3 Pipelining: GATE CSE 2000 | Question: 12 top

https://gateoverflow.in/683

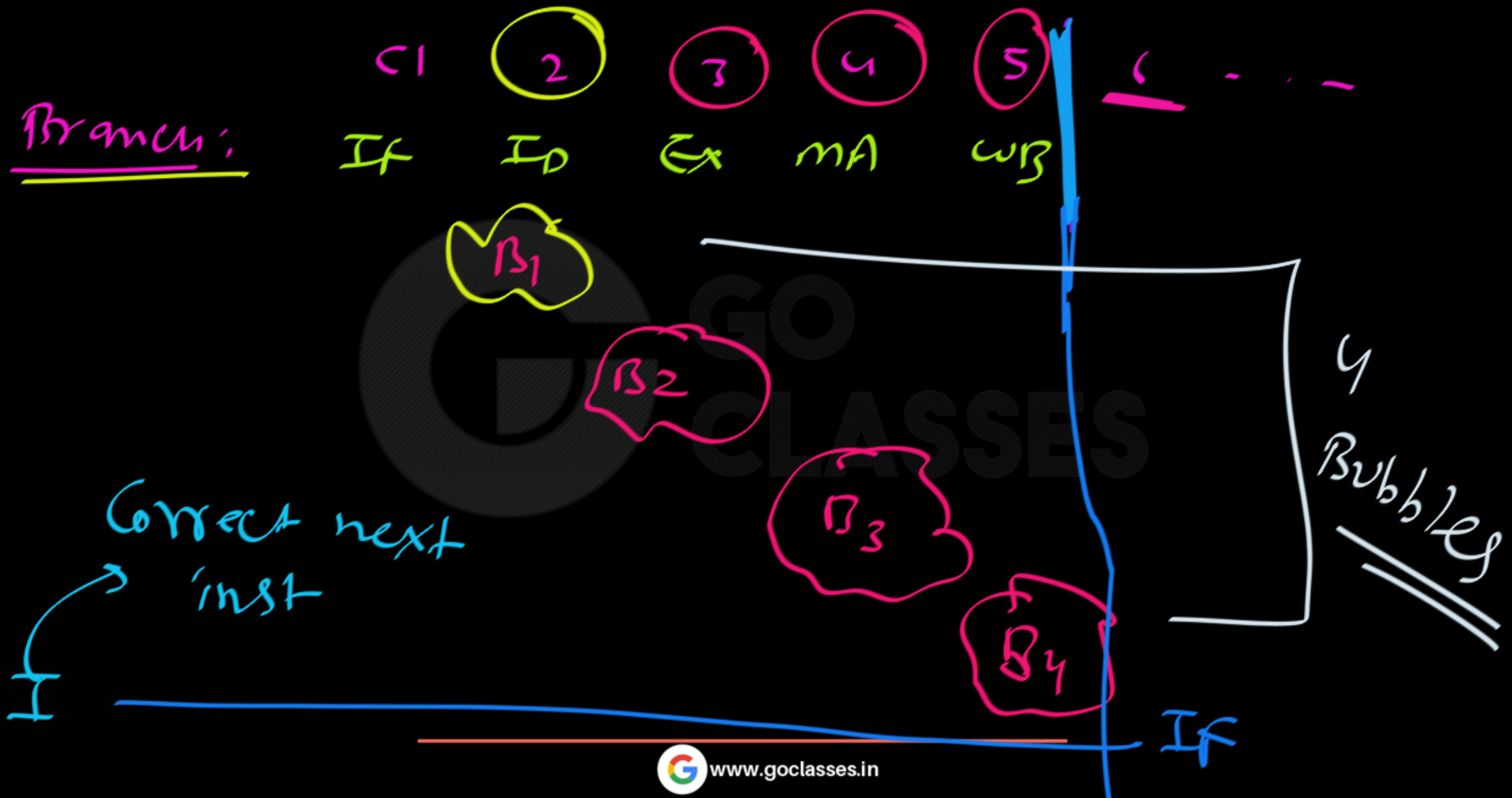
An instruction pipeline has five stages where each stage take 2 nanoseconds and all instruction use all five stages. Branch instructions are not overlapped. i.e., the instruction after the branch is not fetched till the branch instruction is completed. Under ideal conditions,

A. Calculate the average instruction execution time assuming that 20% of all instructions executed are branch instruction. Ignore the fact that some branch instructions may be conditional.

Avg exec time

Solution 1, Type 2

Stalls = 4


C1
2
3
4
5
Branch:
IF
ID
EX
MA
WB
B1
B2
B3
B4
4 Bubbles
Correct next inst
IF

CPI :

$$1 + (0.20)(4) = 1.8 \text{ cycles per inst}$$

Ideal CPI

branch inst per inst

CPI : 1.8

final Ans :

Avg Exe^{n} time : 1.8 X (2 ns)

Cycle time

= 3.6 ns

#Stalls = 4

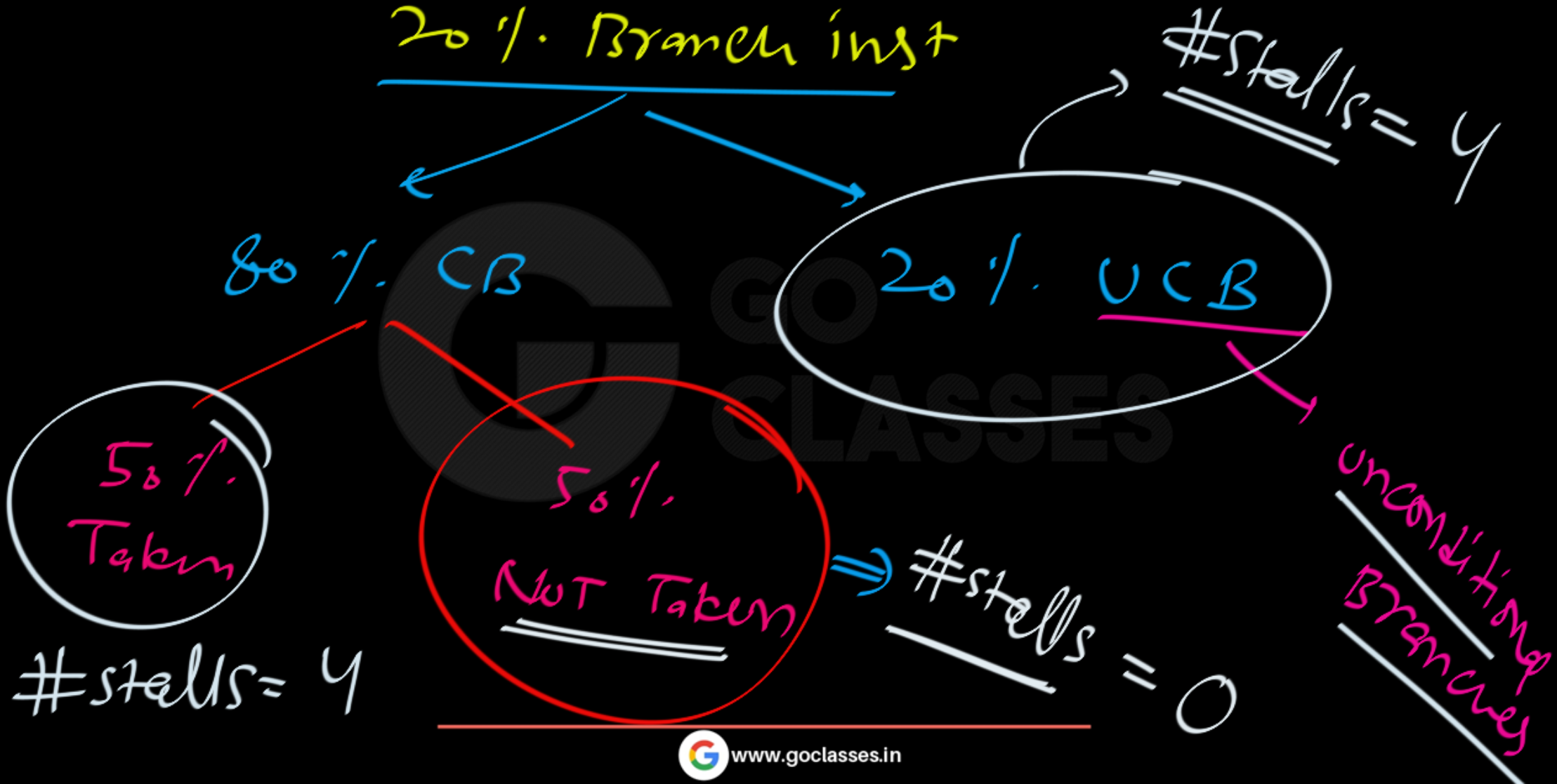
1.18.3 Pipelining: GATE CSE 2000 | Question: 12 top

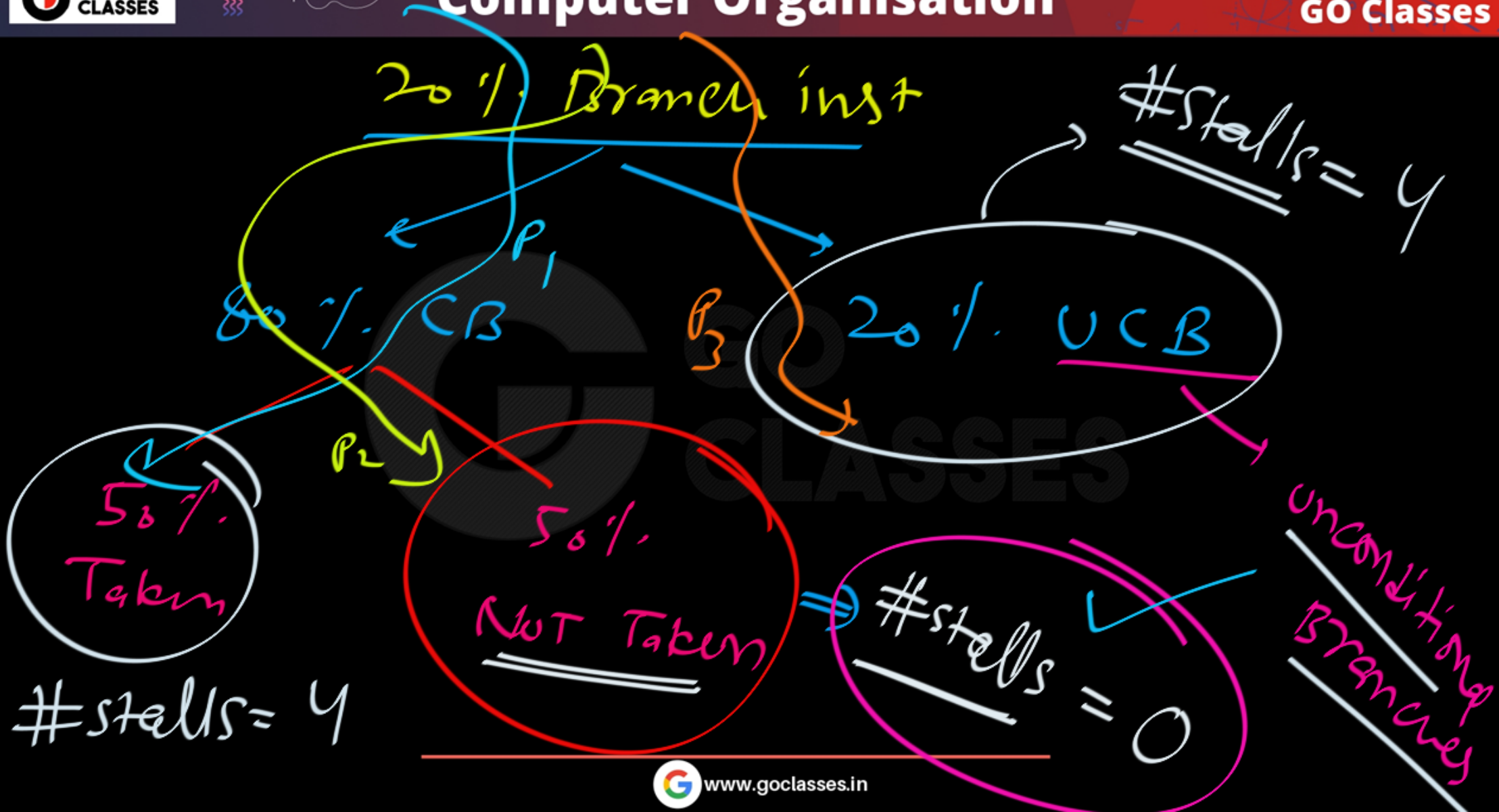
https://gateoverflow.in/683

An instruction pipeline has five stages where each stage take 2 nanoseconds and all instruction use all five stages. Branch instructions are not overlapped. i.e., the instruction after the branch is not fetched till the branch instruction is completed. Under ideal conditions,

A. Calculate the average instruction execution time assuming that 20% of all instructions executed are branch instruction. Ignore the fact that some branch instructions may be conditional.
B. If a branch instruction is a conditional branch instruction, the branch need not be taken. If the branch is not taken, the following instructions can be overlapped. When 80% of all branch instructions are conditional branch instructions, and 50% of the conditional branch instructions are such that the branch is taken, calculate the average instruction execution time.

Solution: Predict Not Taken


20 %. Branch inst
#Stalls = 4
80 %. CB
20 %. UCB
50 %. Taken
50 %. Not Taken
#stalls = 0
#stalls = 4
Unconditional Branches


20% Branch inst
#Stalls = 4
P1
80% CB
P3
20% UCB
P2
50% Taken
50% Not Taken
#stalls = 0
Unconditional Branches
#stalls = 4

CPI :

$$1 + (0.20)(0.80)(0.50)(4) + (0.20)(0.80)(0.50)(0) + (0.20)(0.20)(4)$$

1 → 'Ideal CPI'

$(0.20)(0.80)(0.50)(0)$ → 0

$(0.20)(0.20)(4)$

CPI :

$1 + (0.20)(0.80)(0.50)(y) + (0.20)(0.20)(y)$

$(0.20)(0.80)(0.50)(y) = 0.32$

$(0.20)(0.20)(y) = 0.16$

'Ideal CPI'

$CPI = 1.48$

CPI = 1.48

Avg. Exec time = 1.48 × 2ns

= 2.96 ns

1.18.9 Pipelining: GATE CSE 2006 | Question: 42 top

https://gateoverflow.in/1818

A CPU has a five-stage pipeline and runs at 1 GHz frequency. Instruction fetch happens in the first stage of the pipeline. A conditional branch instruction computes the target address and evaluates the condition in the third stage of the pipeline. The processor stops fetching new instructions following a conditional branch until the branch outcome is known. A program executes 10^9 instructions out of which 20% are conditional branches. If each instruction takes one cycle to complete on average, the total execution time of the program is:

A. 1.0 second
B. 1.2 seconds
C. 1.4 seconds
D. 1.6 seconds

1 ns cycle time

1.18.9 Pipelining: GATE CSE 2006 | Question: 42 top

https://gateoverflow.in/1818

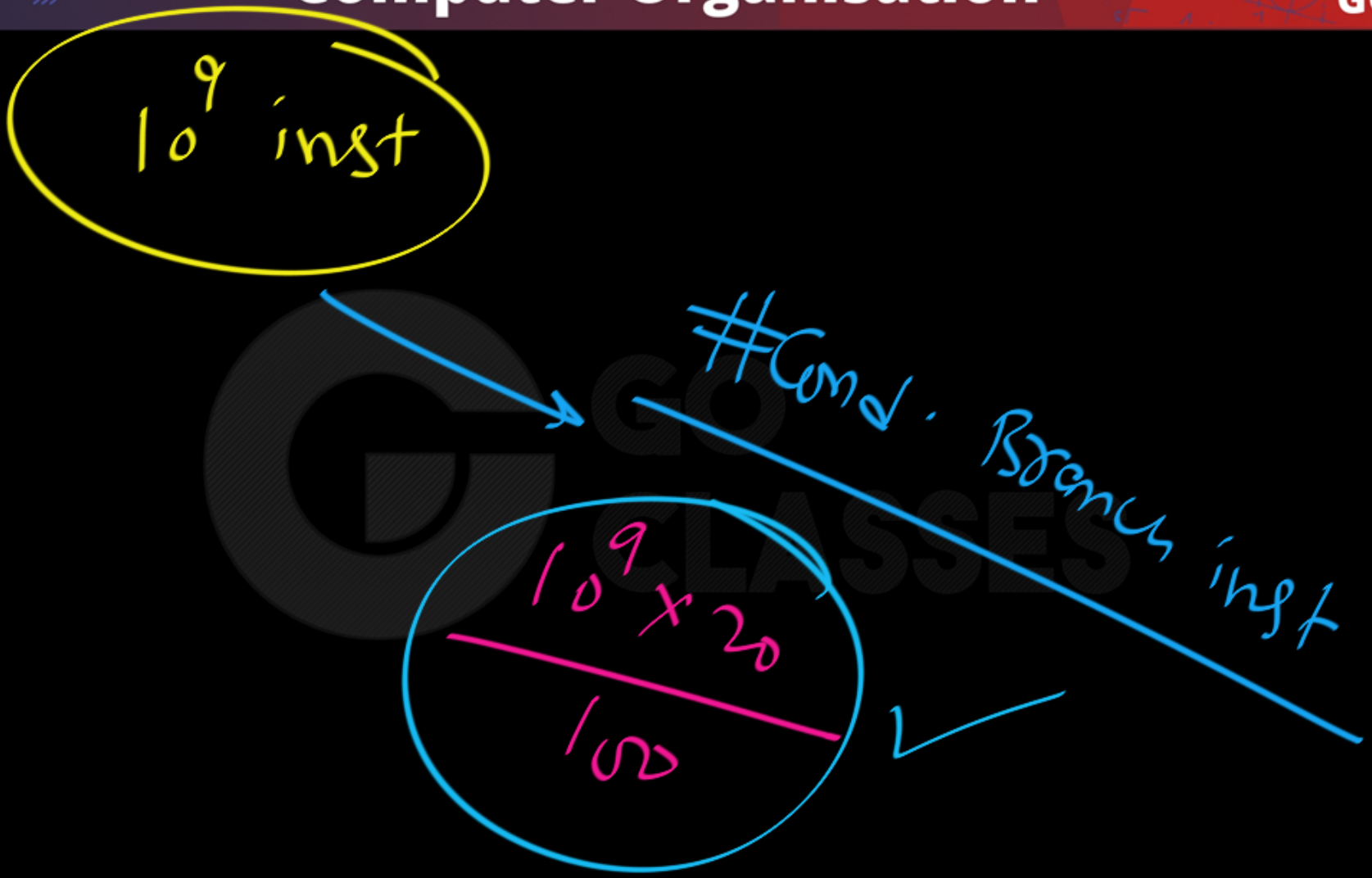
A CPU has a five-stage pipeline and runs at 1 GHz frequency. Instruction fetch happens in the first stage of the pipeline. A conditional branch instruction computes the target address and evaluates the condition in the third stage of the pipeline. The processor stops fetching new instructions following a conditional branch until the branch outcome is known. A program executes 10^9 instructions out of which 20% are conditional branches. If each instruction takes one cycle to complete on average, the total execution time of the program is:

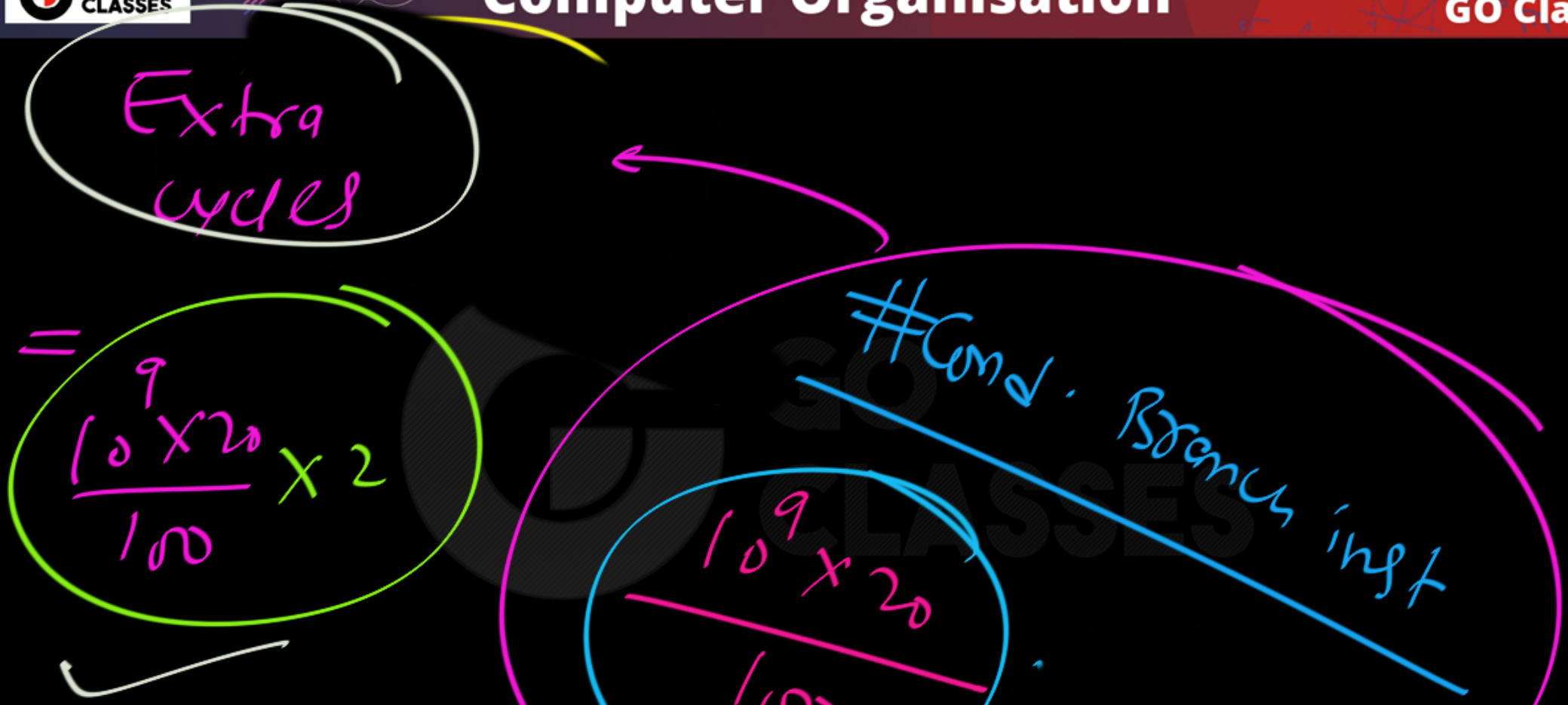
A. 1.0 second
B. 1.2 seconds
C. 1.4 seconds
D. 1.6 seconds

Solution:

Solution 1, Type 2

#stalls = 2


10^9 inst
#Cond. Branch inst
$\frac{10^9 \times 20}{100}$


Extra cycles
= 9/10 x 20/100 x 2
#Cond. Branch inst
9/10 x 20/100

Avg Exec^n time :

$$\frac{9}{10} \times 1\text{ns} + \left(\frac{10^9 \times 20}{100} \right) \times 2 \times 1\text{ns}$$

$\frac{9}{10}$: each inst, on avg, takes one cycle to complete

10^9

Extra time

Avg Exn time :

1 sec + 0.4 sec = 1.4 sec

Solution 3:

Dynamic Branch Predication

Like machine Learning

In Case of Control Hazards:

$$CPI = 1 + (\#mispredictions\ per\ inst.) \times (\#stalls\ per\ misprediction)$$

ideal CPI

BRANCH PREDICTION ACCURACY

$$CPI = 1 + \underbrace{\frac{\text{MISPREDS}}{\text{INST}}}_{\text{PREDICTOR ACCURACY}} \times \underbrace{\frac{\text{PENALTY}}{\text{MISPRED}}}_{\text{PIPELINE}}$$

- 20% OF ALL INSTS ARE BRANCHES

for Branch Predictor

ACCURACY ↓	RESOLVE BR. IN 3RD STAGE	RESOLVE BR. IN 10TH STAGE
50% FOR BR 100% ALL OTHER	?	?
90% FOR BR 100% ALL OTHER	?	?

BRANCH PREDICTION ACCURACY

$$CPI = 1 + \frac{\text{MISPREDS}}{\text{INST}} \times \frac{\text{PENALTY}}{\text{MISPRED}}$$

PREDICTOR ACCURACY — PIPELINE

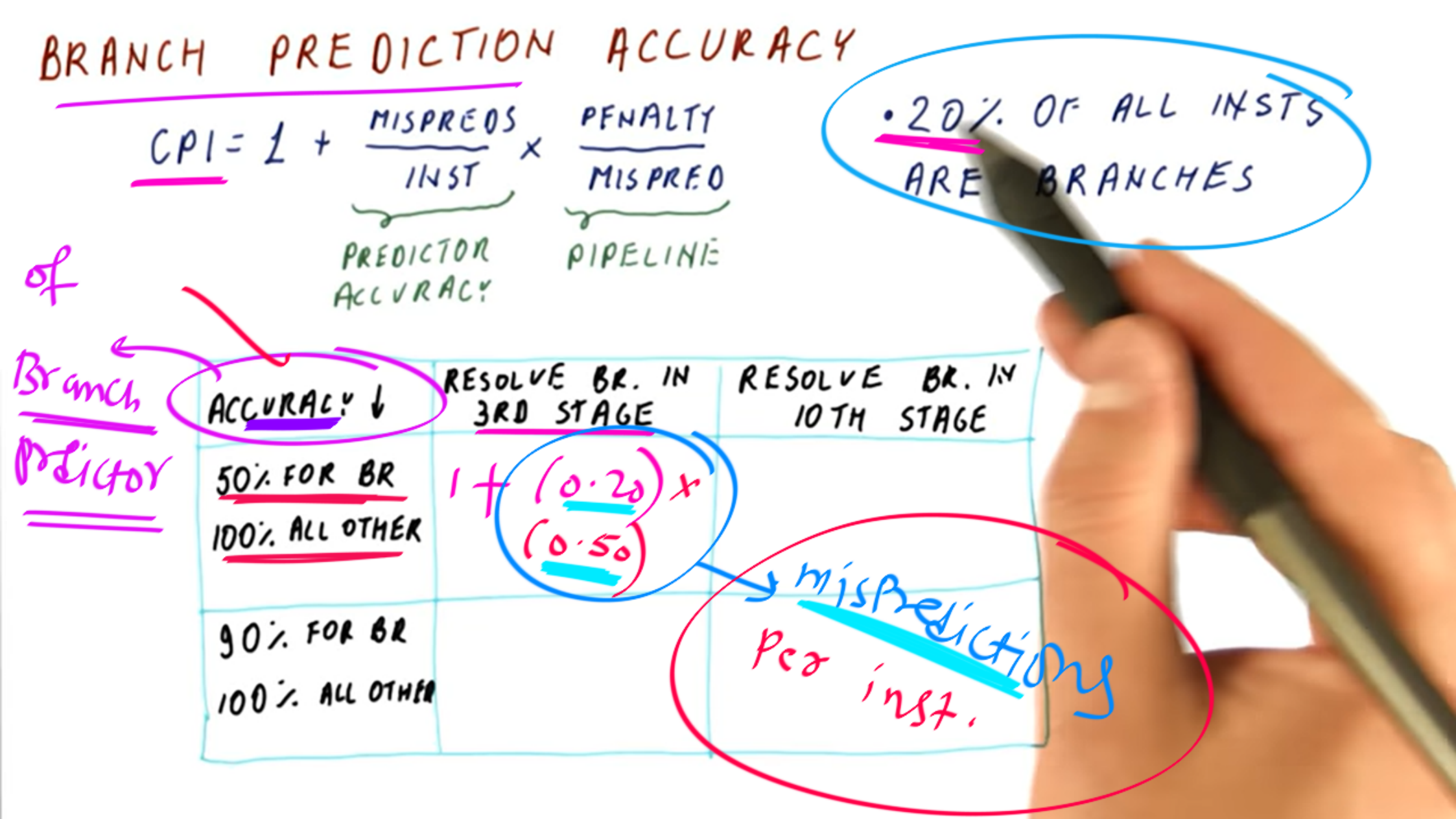
- 20% OF ALL INSTS ARE BRANCHES

of

Branch Predictor

ACCURACY ↓	RESOLVE BR. IN 3RD STAGE	RESOLVE BR. IN 10TH STAGE
50% FOR BR 100% ALL OTHER	$1 + (0.20) \times (0.50)$	
90% FOR BR 100% ALL OTHER		

→ mispredictions per inst.



BRANCH PREDICTION ACCURACY

$$CPI = 1 + \underbrace{\frac{\text{MISPREDS}}{\text{INST}}}_{\text{PREDICTOR ACCURACY}} \times \underbrace{\frac{\text{PENALTY}}{\text{MISPRED}}}_{\text{PIPELINE}}$$

- 20% OF ALL INSTS ARE BRANCHES

of

Branch Predictor

ACCURACY ↓	RESOLVE BR. IN 3RD STAGE	RESOLVE BR. IN 10TH STAGE
50% FOR BR 100% ALL OTHER	$1 + (0.20) \times (0.50) \times 2$	$1 + (0.20)(0.50) \times 9$
90% FOR BR 100% ALL OTHER	$1 + (0.20)(0\%) \times 2$	$1 + (0.20)(0\%) \times 9$

BRANCH PREDICTION ACCURACY

$$CPI = 1 + \underbrace{\frac{\text{MISPREDS}}{\text{INST}}}_{\text{PREDICTOR ACCURACY}} \times \underbrace{\frac{\text{PENALTY}}{\text{MISPRED}}}_{\text{PIPELINE}}$$

- 20% OF ALL INSTS ARE BRANCHES

Branch Predictor of

ACCURACY ↓	RESOLVE BR. IN 3RD STAGE
50% FOR BR 100% ALL OTHER	$1 + (0.20) \times (0.50) \times 2$
90% FOR BR 100% ALL OTHER	$1 + (0.20)(0\%) \times 2$

1.04

1.2

Which is better pipeline?

Speedup of lower one over upper one : $\frac{1.2}{1.04} = 1.15$

Lower one

BRANCH PREDICTION ACCURACY

$$CPI = 1 + \underbrace{\frac{\text{MISPREDS}}{\text{INST}}}_{\text{PREDICTOR ACCURACY}} \times \underbrace{\frac{\text{PENALTY}}{\text{MISPRED}}}_{\text{PIPELINE}}$$

- 20% OF ALL INSTS ARE BRANCHES

ACCURACY ↓	RESOLVE BR. IN 3RD STAGE	RESOLVE BR. IN 10TH STAGE
50% FOR BR 100% ALL OTHER	$1 + 0.5 \times 0.2 \times 2$ 1.2	$1 + 0.5 \times 0.2 \times 9$ 1.9
90% FOR BR 100% ALL OTHER	$1 + 0.1 \times 0.2 \times 2$ 1.04	$1 + 0.1 \times 0.2 \times 9$ 1.18
	1.15	1.61

BRANCH PREDICTION BENEFIT QUIZ

- 5-STAGE PIPELINE
- BRANCH RESOLVED IN 3RD STAGE
- FETCH NOTHING UNTIL SURE WHAT TO FETCH
- EXECUTE MANY ITERATIONS OF

```
LOOP:
    ADDI  R1, R1, -1
    ADD   R2, R2, R2
    BNEZ  R1, LOOP
```

SPEEDUP OF PERFECT PREDICTOR IS _____

BRANCH PREDICTION BENEFIT QUIZ SOLUTION

- 5-STAGE PIPELINE
- BRANCH RESOLVED IN 3RD STAGE
- FETCH NOTHING UNTIL SURE WHAT TO FETCH
- EXECUTE MANY ITERATIONS OF

```
LOOP:
    ADDI  R1,R1,-1   2   1
    ADD   R2,R2,R2   2   1
    BNEZ  R1,LOOP    3   1
                     ~   ~
                     7   3
```

SPEEDUP OF PERFECT PREDICTOR IS 2.33

Solution 4

Delayed Branch

Solution 4

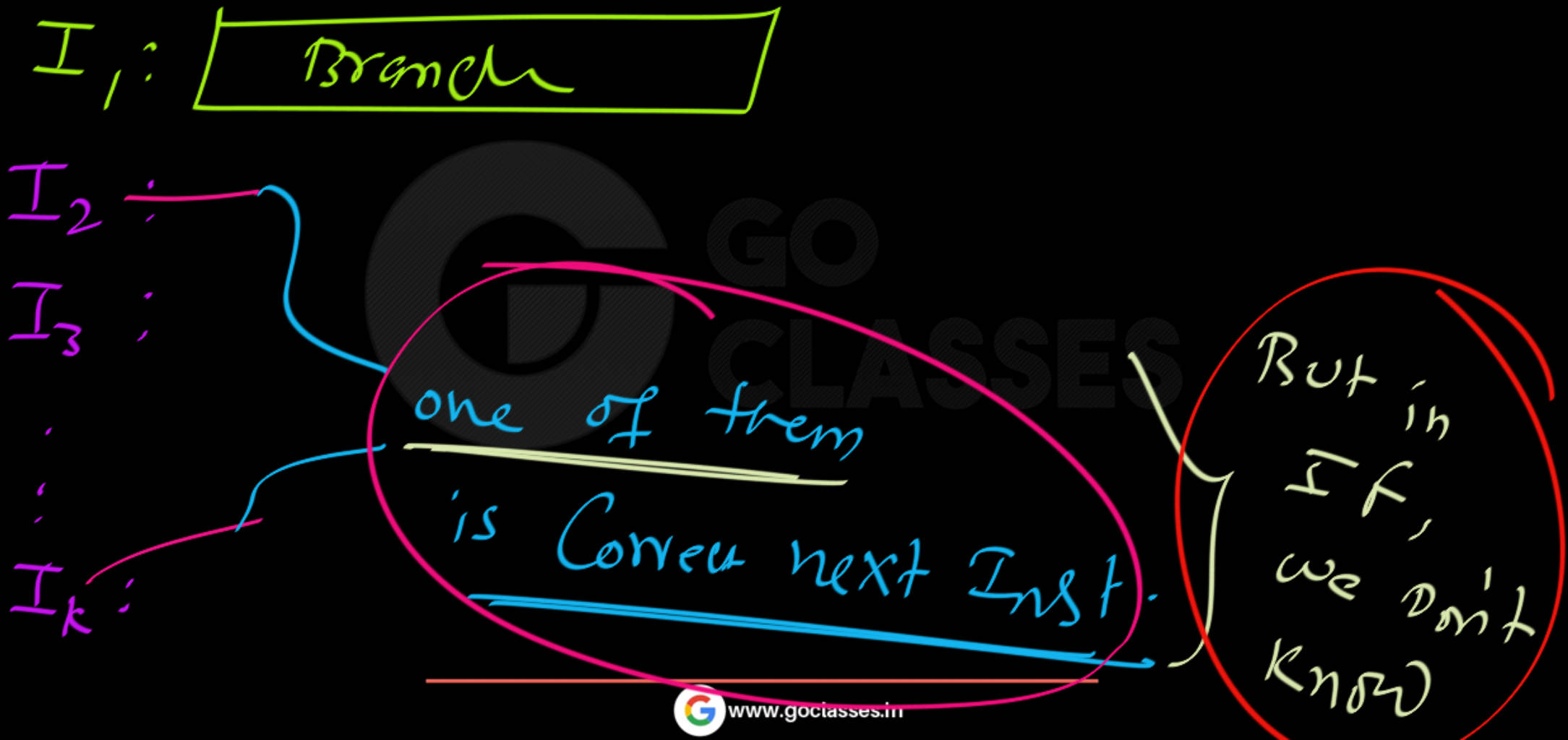
Delayed Branch

- Used in Classical 5-stage MIPS (RISC) Pipeline

Assume in ID Stage, Branch Resolved. → 2^{nd}

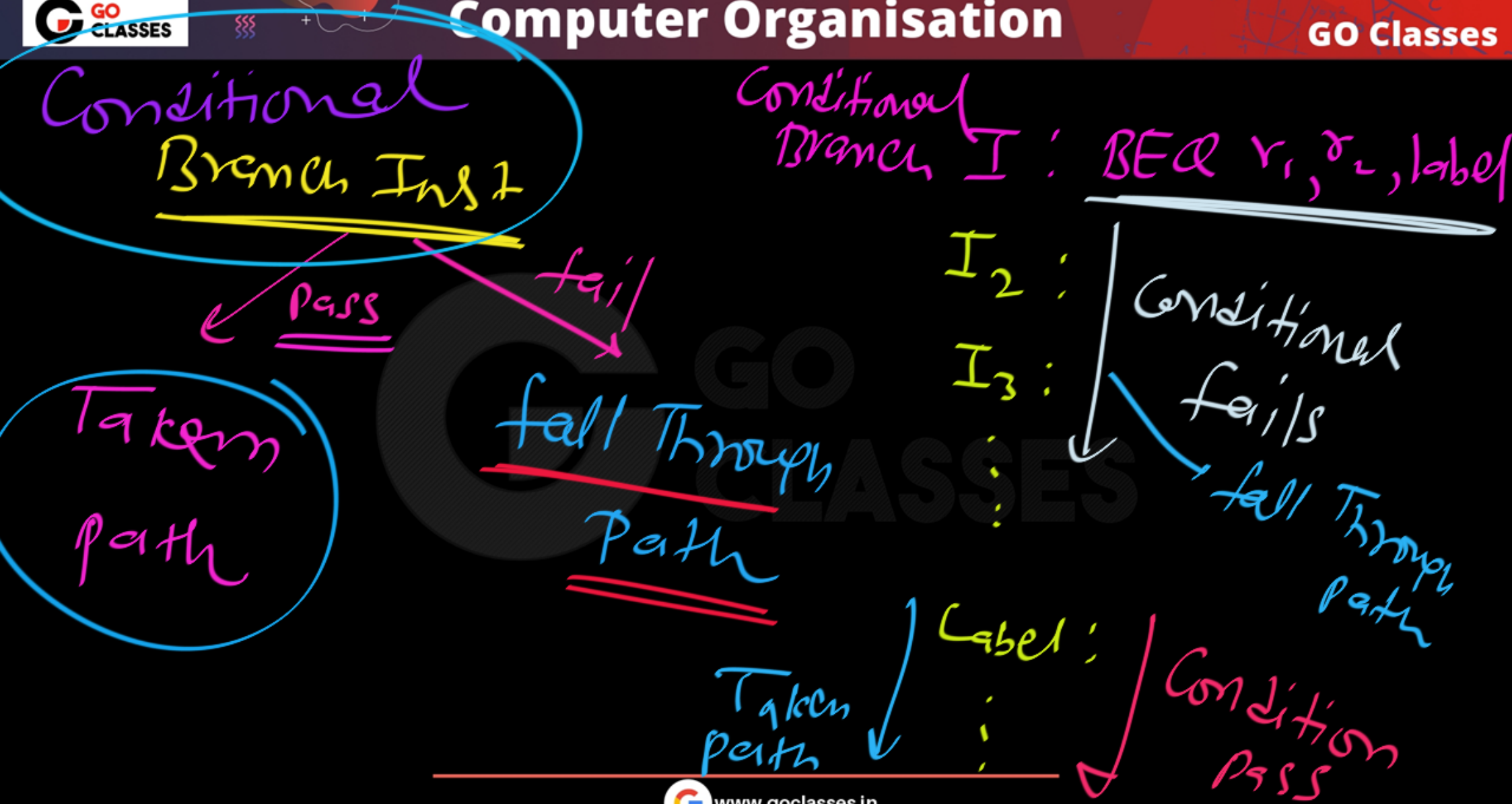
Then # Stalls Due to mispredictions

= 1 ⟹ Delay Slot


I1 : Branch
I2 :
I3 :
:
:
Ik :
one of them
is Correct next Inst.
But in
If,
we don't
know

Delay Slot :

if Branch Resolves in kth stage then k-1 Delay Slots


Conditional Branch Instr
Pass
fail
Taken path
fall Through Path
Conditional Branch I : BEQ r1, r2, label
I2 :
I3 :
Conditional fails
fall Through path
Taken path
Label :
Condition Pass

Delayed Branch Scheme :

#delay slot = 1

I_1 : Branch Inst

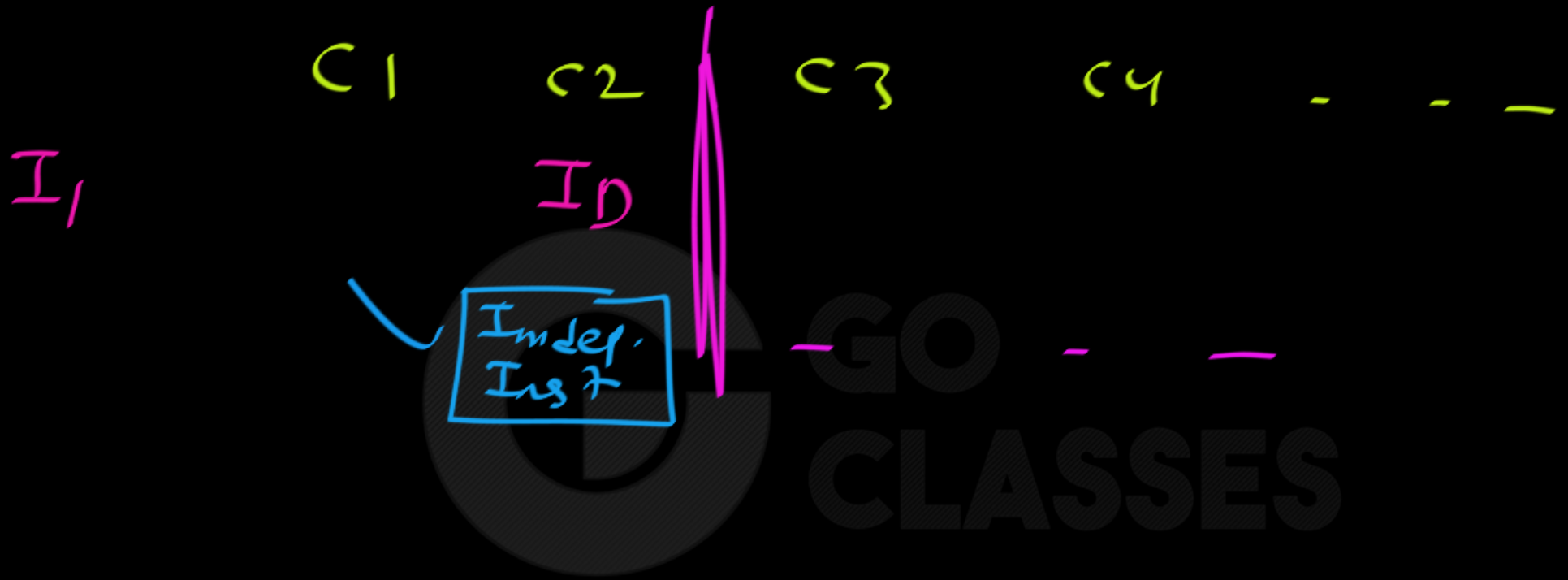
Delay Slot → Independent Inst → by Compiler ✓

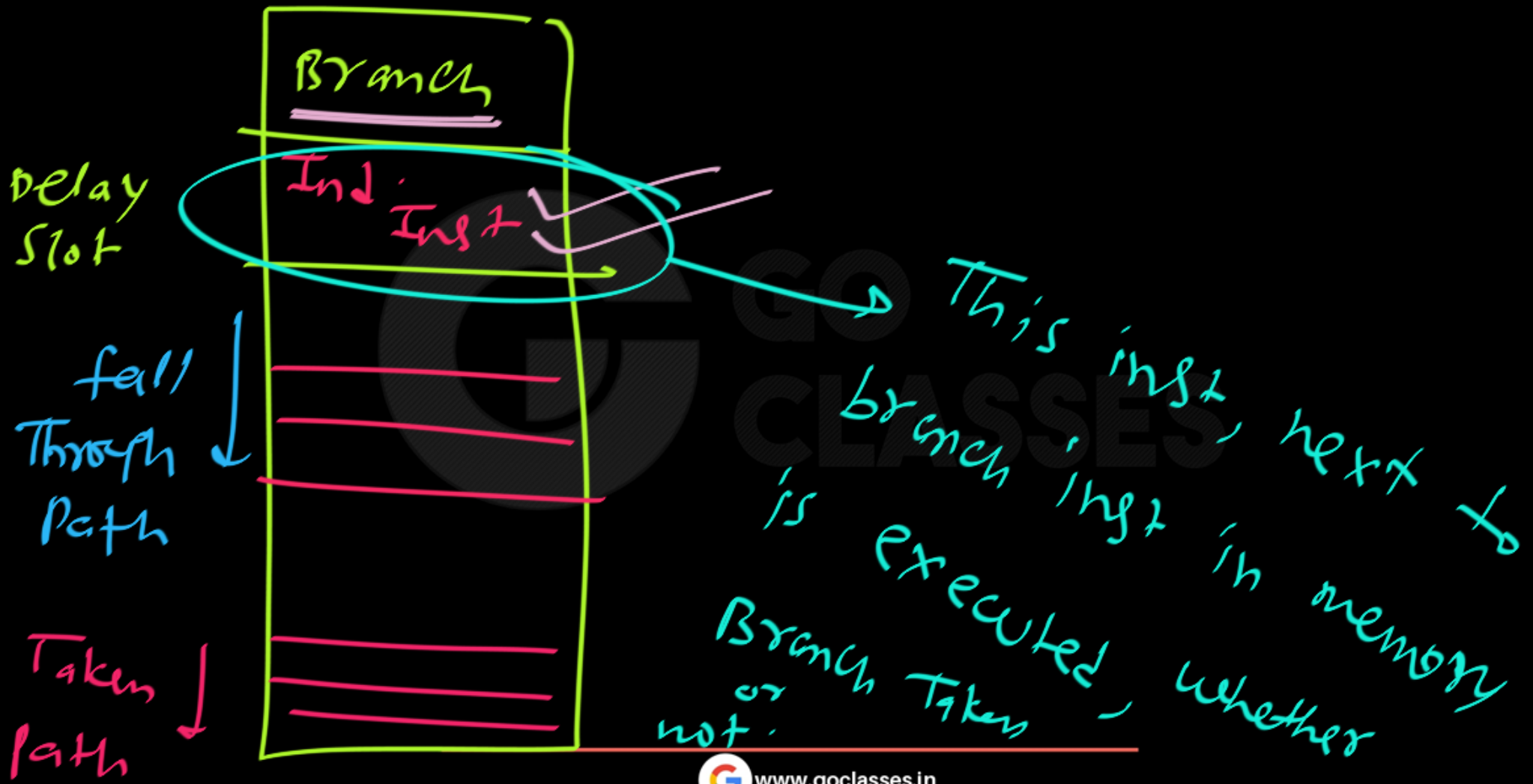
fall Through Path

I_2 :

I_3 :

Label : → Taken Path


C1
C2
C3
C4
I1
ID
Indef. Inst


Branch
Ind. Instr
Delay Slot
fall Through Path
Taken Path
This instr, next to branch instr in memory is executed, whether Branch Taken or not.

Q: If Compiler can NOT find out suitable Instr to put in Delay slot then?

Q: If Compiler Can NOT find out suitable Inst to put in Delay slot then!

then it put NOP inst.

Q: # Delay slots ?

K − 1

→ stage in which branch is resolved.

Q: If #Delay slots increases then for Compiler, is it easy or hard to fill Delay slots with suitable instr?

Q: If #Delay slots increases then for Compiler, is it easy or Hard to fill Delay Slots with Suitable instr ?