MicroProcessors

Cycle	IF	ID/OR	Ex	Mem	WB
1	I_1				
2	I_2	I_1			
3	I_3	I_2	I_1		
4	I_4	I_3	I_2	I_1	
5	I_5	I_4	I_3	I_2	I_1
6	I_6	I_5	I_4	I_3	I_2
7	I_7	I_6	I_5	I_4	I_3

latency

every cycle 1 instr' compls

$\therefore CPI \approx 1$

Cycle time = max (IF, ... WB)

↳ same as multi cycle max time.

Single cycle implem^n → Hardware (Regist. interface)

↳ Decoded control signals need to be carried through

$I_1: R_1 = R_2 + R_3$
$I_2: R_4 = R_1 + R_5$
⇒ Dependant
∴ will fail

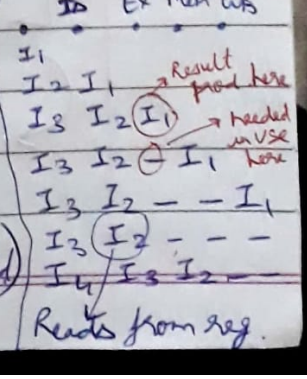
Approx. computing ⇒ where we try to reduce error (media play in comput.)

⇒ Detect dependencies

comparators

↳ $(RS1^{ID} == RD^{Ex}) \,||\, (RS2^{ID} == RD^{Ex})$

∴ When detected, stop ID instr^n until Ex instr^n is completed/writes back

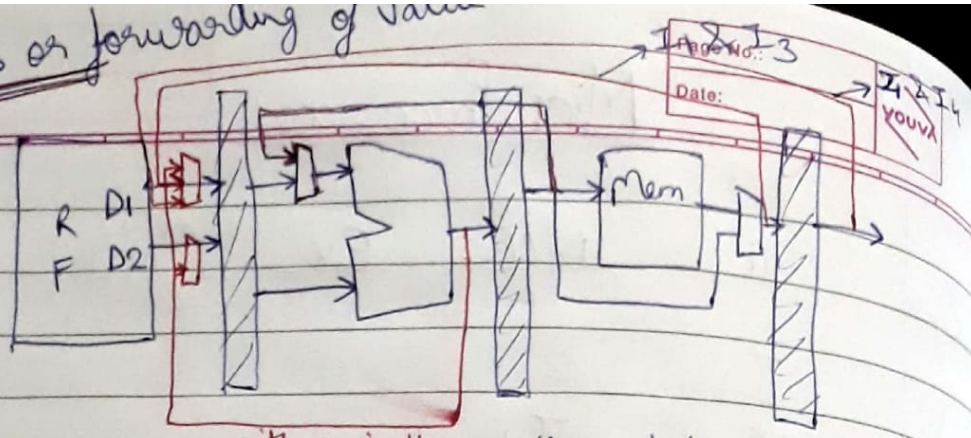


If 10 instr^n have such imm depend.

↳ CPI$_{avg}$ = 1.3

(For every 10 instr's → 10 + 3 cycles needed)

By-pass or forwarding of value



either is fine ⇒ delay ↑ by 1 on longest path

∴ Now, pipeline will work

If I_1 & I_3 dependancy,
↳ $(RS1^{ID} == RD^{mem}) \,||\, (RS2^{ID} == RD^{Mem})$

If I_1 & I_4 dependancy,
↳ $(RS1^{ID} == RD^{WB}) \,||\, (RS2^{ID} == RD^{WB})$

I_1: $R_1 = R_2 + R_3$
I_2: $R_1 = R_1 + R_6$
I_3: $R_7 = R_1 + R_0$

~~a~~ $a = b+c+d+e$

⇓
∴ you prioritize (Use the closer value when both dependancies are there)

Ex I_1 : $R_1 \leftarrow R_2 + 50$
ID I_2 : $R_3 = R_1 + R_4$
↳ Same cycle. ⇒ In this case, we can't do anything as data itself is not produced until before

∴ Here we need to wait for 1 cycle
↳ Load followed by immediate dependancy

↳ load $R_1, (R_2), 50$: I_1
add R_3, R_1, R_4 : I_2

Bubble or Stall

Ex mem WB
4) I4 I3 I2 I1
5) I4 I3 I2 I1
6) I5 I4 I3 I2

Page No.: Date: YOUVA

Effect ⇒ α - ALU Instⁿ
β - Load → 20% → ↑ inst. dependⁿ. → 30%
γ - Store
δ - Branch

∴ CPI = 1 + β λ (l)
= 1 + (.2)(.3)
= 1.06

⇒ How to stall ? ⇒ Disable the registers at IF/ID & ID/Ex interface & other instⁿ store mem.

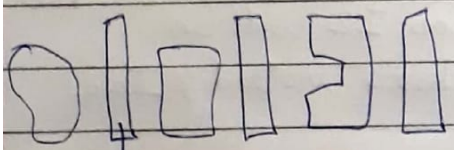
A = B + C + D
T1 = B + C
A = T1 + D

load r_1, B
load r_2 C
Add r_3, r_1, r_2
Store r_3, t_1

to remove the 2 inst. dep.ⁿ
load r_4, D
add r_5, r_4, r_3
Store r_5, A

Change in control flow (Branch/Jump)

I1 : Jump → Taken branch
↳ can be detected earliest at ID stage



IF ID Ex mem WB
Nullify I_1 I_2 I_1 Modify PC soon

@ I_1 → To nullify → not let it change process/system state

I_1 : $r_1 = r_2 + r_3$
I_2 : call ( )
NOP ⇒ If you don't have any other indep. instr.

I_1 :
I_2 : call ( )
$r_1 = r_2 + r_3$
↳ when call & r_1 are indep.ⁿ

Delayed branch execution → specify in ISA
→ ~~hoogle has to be left.~~
→ 1 instⁿ after the branch is executed.

↓
I_2 is already loaded & is in IF. ∴ that will be executed before ~~call instⁿ~~ function.

Cond^n. calls

) Handling of branches → data dependant

I1 : BEQ R1, R2, Imm16

∴ Every misprediction → 2 cycle penalty

IF ID EX Mem WB

I_1

I_2 I_1

~~I_3~~ ~~I_2~~ I_1

I_{20} — — I_1

I_{21} I_{20} — — I_1

Assume branch not taken & go on

CPI = 1 + 0.15 × 0.7 × 2 → 2 cycles lost

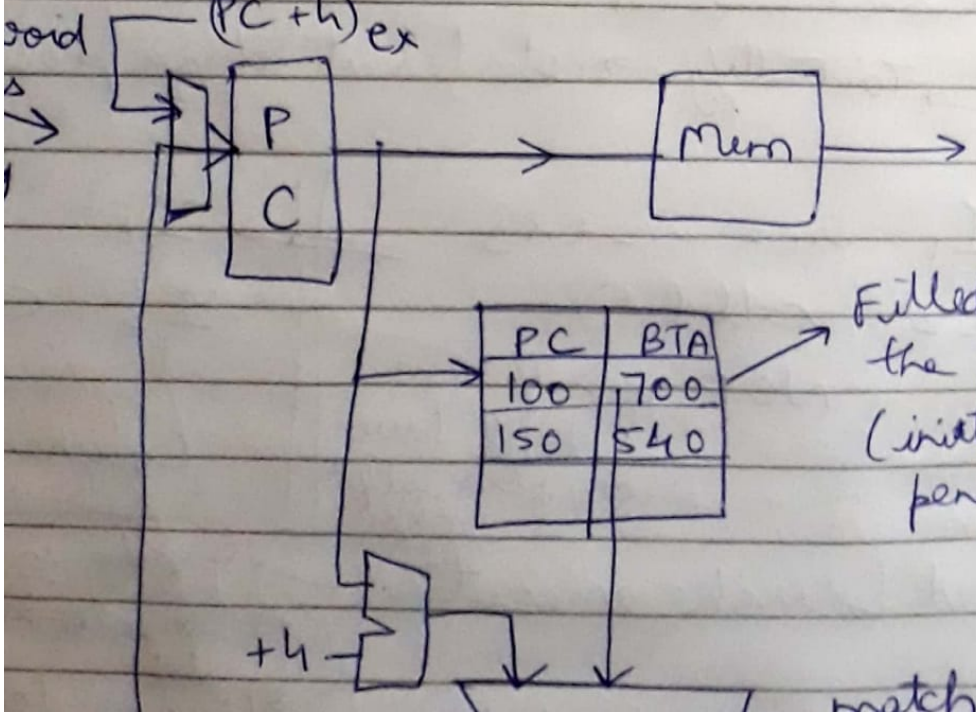
0.15 : % branch instns. 0.7 : % branch taken

But majority instn's require to branch

⇒ Use lookup table & take I_{20} by default

But then Branch instn is detected only at ID stage. ⇒ One cycle will always be lost as I2 will be loaded in IF ~~which has~~ instead of I_{20}.

PC	BTA
100	700
150	540

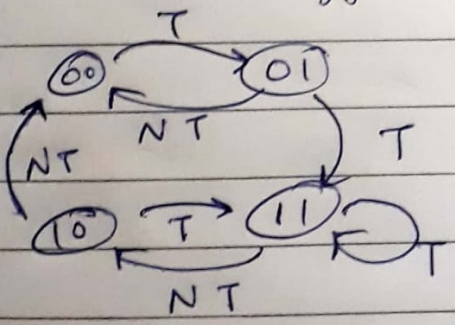


But if branch not taken, table not filled.
But in ex. stage, PC+4 ~~is available~~ & we detect it is branch. → has to be carried till exec^n stage

Pred	Actual	
0	1	entry
1	1	
.	.	
.	.	
.	.	
1	0	exit

∴ only at entry & exit pe it will go wrong.
80%.

looking at more past history, we can further ↑ efficiency to almost 95%.



look at MSB to decide if Branch by default or not.