DR. ASHRAF SUYYAGH

DONE BY
SONDOS ABU IDAQ

Five-Stage Pipeline

F: Fetch instruction from the instruction memory

D: Decode instruction and read operands

E: Execute operation or calculate address

M: Memory access

W: Write result to the register

فائدة الـ Piplie :

* السيتج الوحدة صغرت بالتالي كمية الـ logic hw فيها بقل .

* بالتالي اللي بحدد الـ max freq هو أطول stage بالـ path

programm counter : instruction جواته address

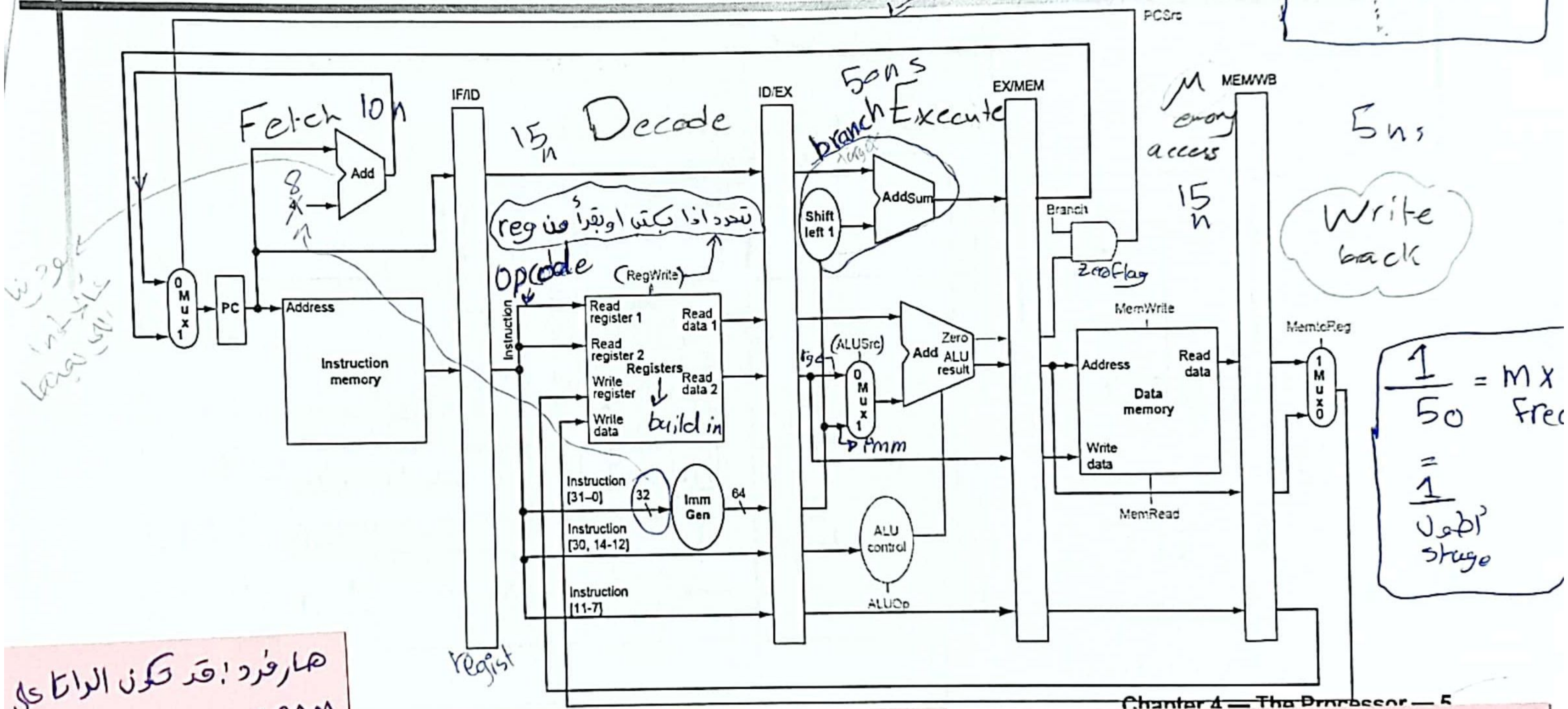
* بتحددهم الـ (Control unit) بناءً على نوع الـ inst

RegWrite
Alu Src
MemWrit
⋮

(Risc-V - 64 bit)

Five-Stage Pipeline

ADD X_1 , X_0 , X_2



$\frac{1}{50}$ = MAX Freq = $\frac{1}{\text{الطول stage}}$

هارفرد: قد تكون الداتا على SRAM و الانستركشنز على فلاش عمودي لا يحدث مشكلة

Von: لازم يكونو تكنولوجي وحدة وغالباً DRAM

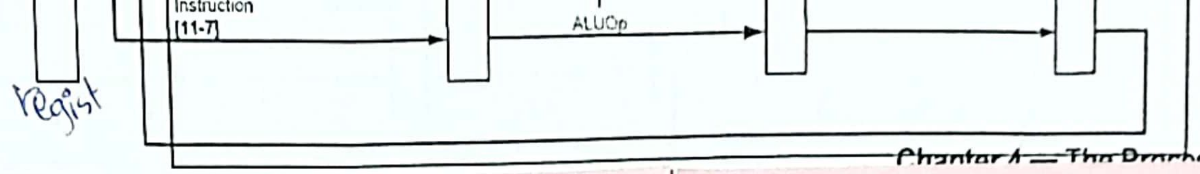
Harv: تستخدم بكثرة في تطبيقات process اللي فيها امبيديد components.

Von: *عمودي بنحط فيها كلشي.
*كل الـ مجتمع الحالية فيها عمودي Von.

سلبيات:
هارفرد: ضعف كمية الاسلاك ← مكلفة
Von: ...

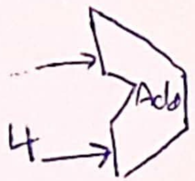
* الـ pipes ← عبارة عن ريجسترز ← FlipFlop
* بنحط الـ pipes ← return info from previos instruction as it goes to the next stage.

inst Mem 2 types:
① Harvard ② von Neuman



طول الانستركشنز : 32 أو 64

Risc-V → 32 bit



64 bit

Harv: process *تستخدم بكثرة في تجاميع اللي فيها اعبيديد components.

Von: *عموري بنحط فيها كلشي.

*كل الاجهزة الحالية فيها عموري Von.

سلبيات:

هارفرد: ضعف كمية الاسلاك ← تكلفة

Von: أبطئ

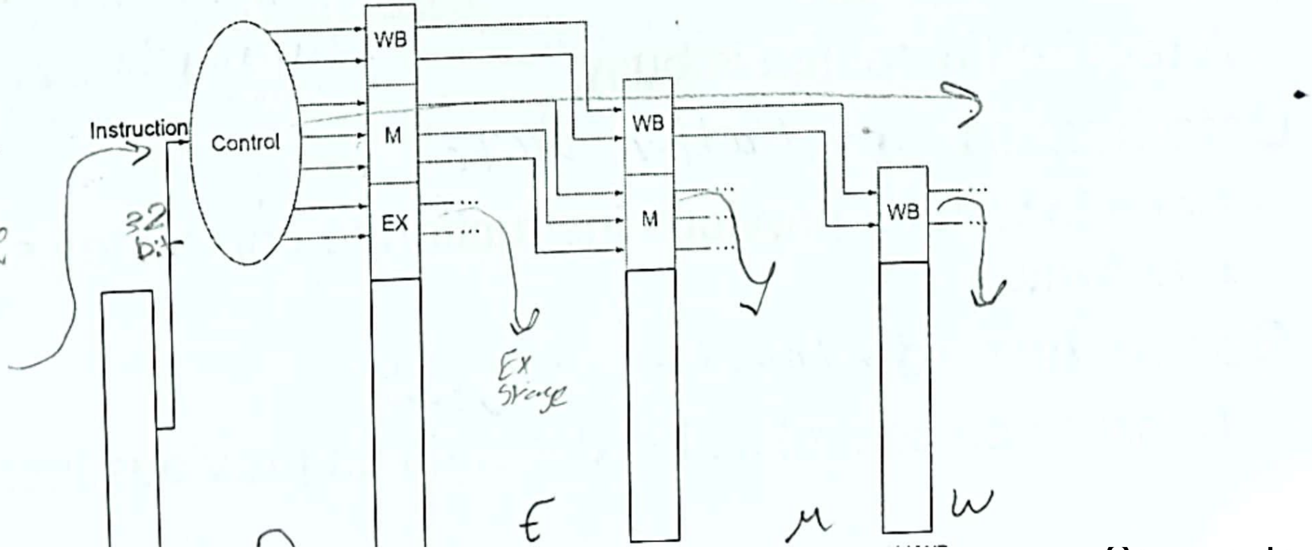
*ال pipes ← عبارة عن ريجسترز ← Flip Flop

*بنحط ال pipes ← return info from previos instruction as it goes to the next stage.

inst mem 2 types:
① Harvard ② von Neuman

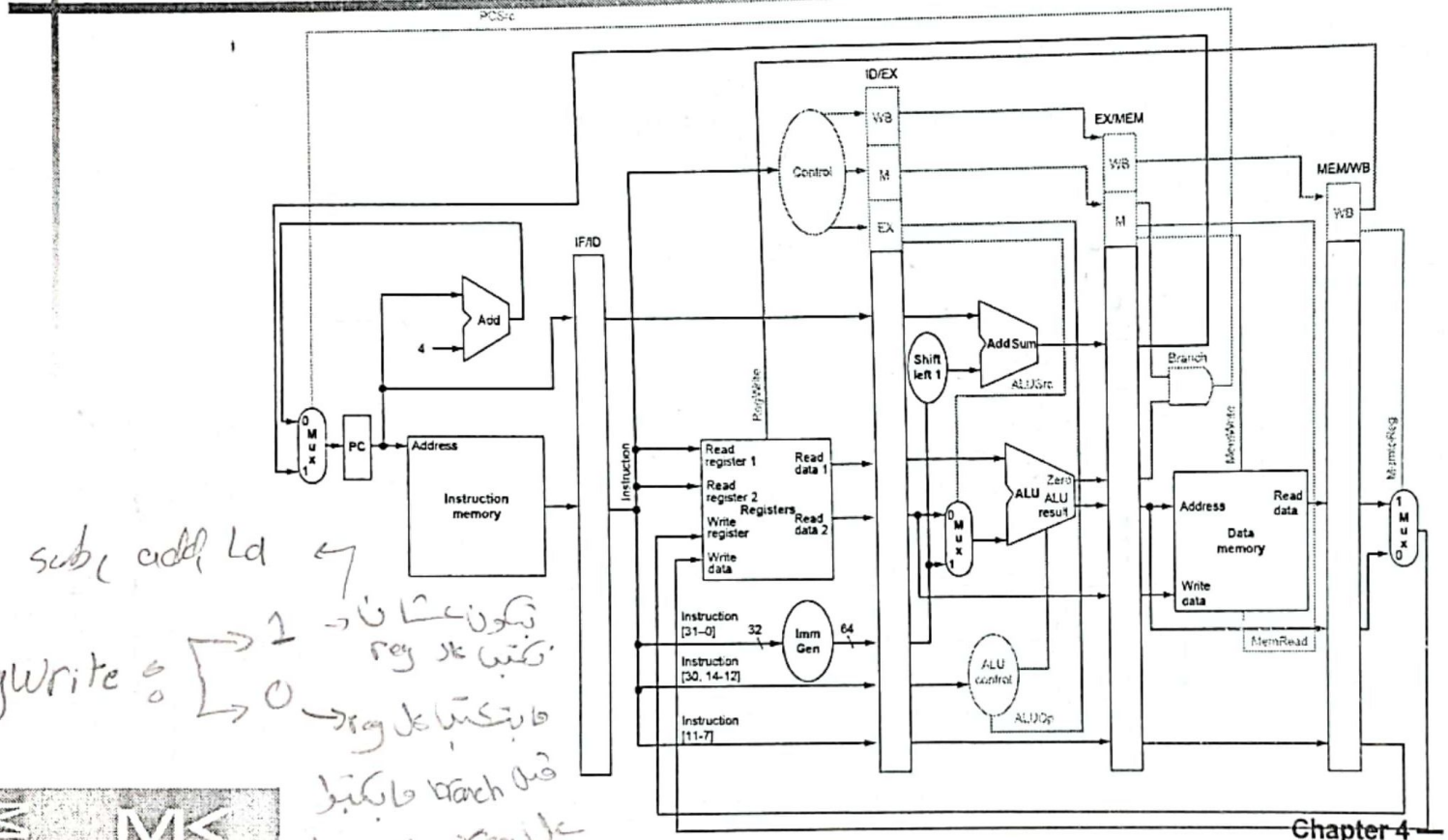
...ed Control

- ontrol signals derived from instruction
  - As in single-cycle implementation



opcode 7 bit نطلع ال control signals

Pipelined Control



sub, add, Ld ←

RegWrite: 1 → بيكون شغال عشان يكتب على reg

0 → reg مابيكتبش على

قبل branch مابيكتبش على reg وال start كمان

Chapter 4 — The Processor — 7

Branch: لما active يكون

branch ← inst يكون نوع

MemtoReg: بيكون 0 لما الداتا اللي جاية من Alu

وبيكون 1 لما الداتا جاية من الميموري قبل load

ALUSrc: 0 ← reg بيجيب من Alu

1 ← imm inst بيجيب من

ALUOp: بت 2 عبارة عن

بيحدد العملية اللي Alu تشتغل عليها

Hazards → صعوبات

- Situations that prevent starting the next instruction in the next cycle

- ✓ Structure hazards
  - A required resource is busy
- ✓ Data hazard
  - Need to wait for previous instruction to complete its data read/write
- ✓ Control hazard → branch → stage متأخر
  - Deciding on control action depends on previous instruction

المهم علاقة بال pipline

* بتصير عما أدخل أكتر من instr بتصير بعدها فاضية
* حالآخر يونت فاضية اشتغل عليها

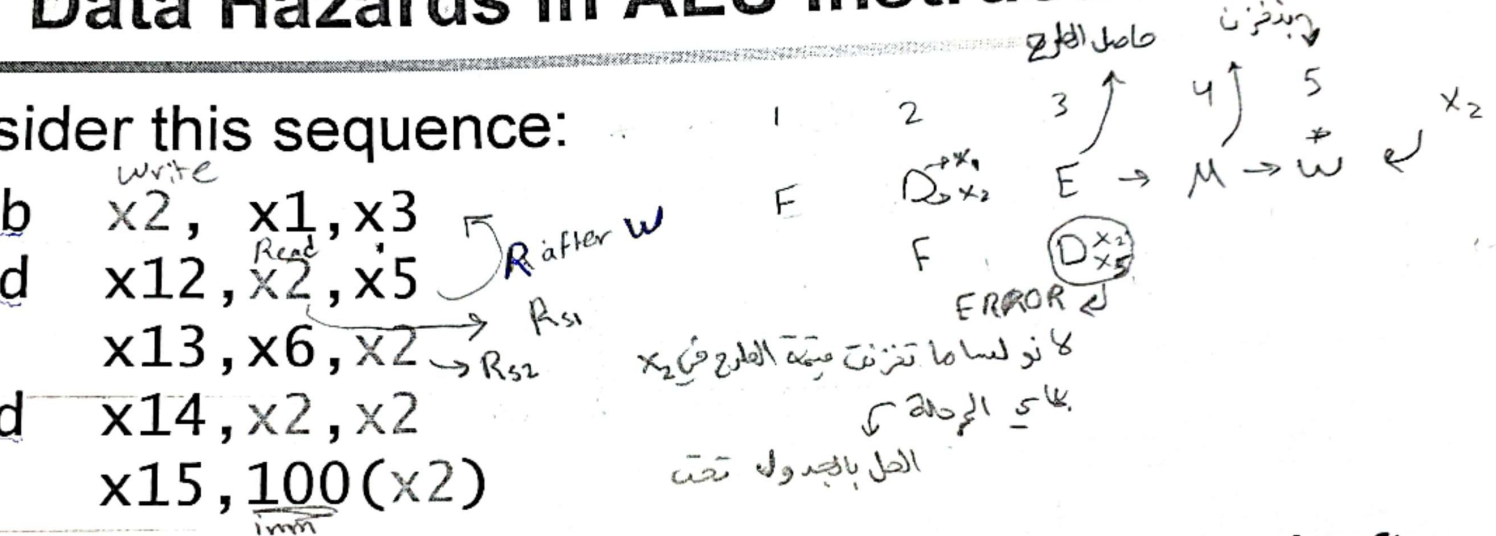
read after write

Data Hazards in ALU Instructions

- Consider this sequence:

```
sub   x2, x1,x3
and   x12,x2,x5
or    x13,x6,x2
add   x14,x2,x2
sd    x15,100(x2)
```

- There are multiple true data dependencies, read-after-write (RAW), on register x2.
- We can resolve hazards with stalls or forwarding.



Assume no forwarding (except through the Register File) and hazards are solved by stalls

write before read

	1	2	3	4	5	6	7	8	9	10	11
sub x2, x1,x3	F	D	E	M	W (x2)						
and x12,x2,x5		F	D	D	D	E	M	W			
or x13,x6,x2			2 stalls because we stopped	F	F	D	E	M	W		
add x14,x2,x2			for 2 cycles	F	F	F	D	E	M	W	
sd x15,100(x2)							F	D	E	M	W

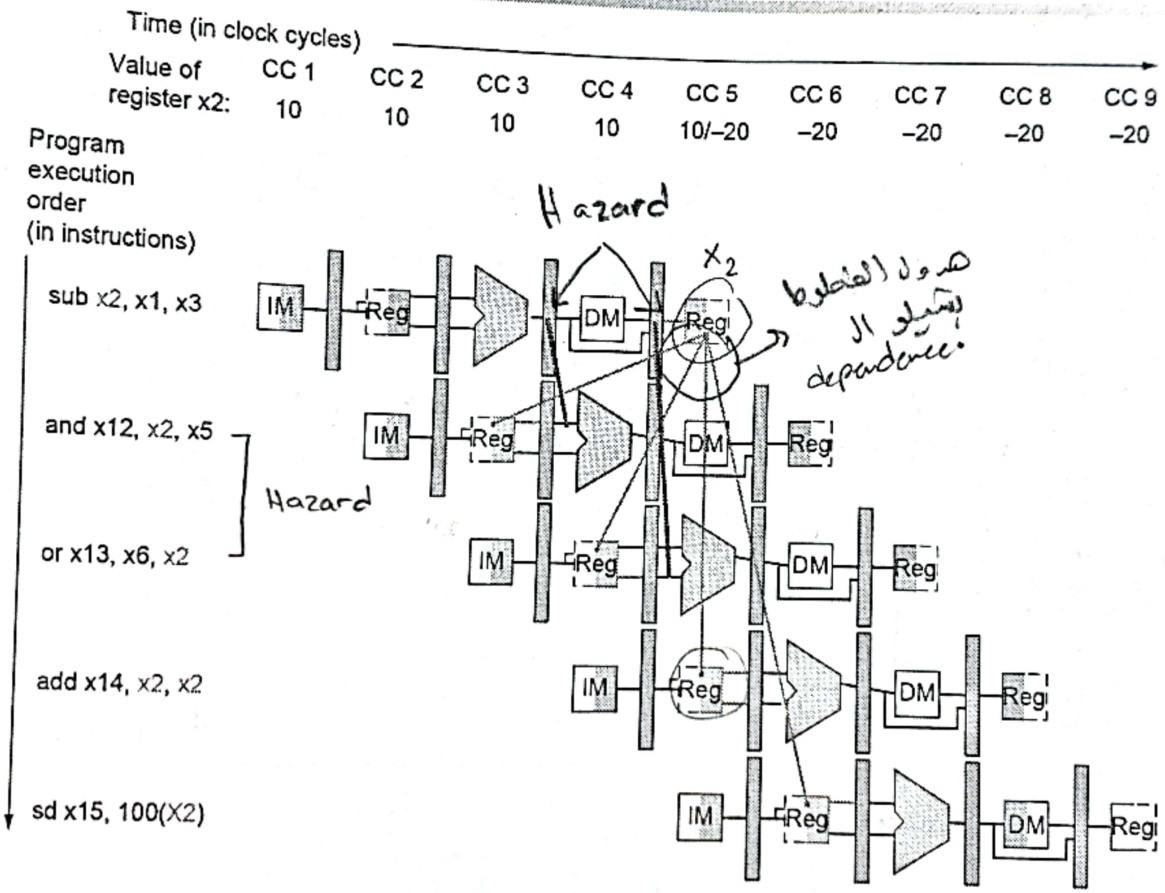
we used 11 cycles in this instruction

# of cycles = 4 + 1C + stalls.

4 + 5 + 2 = 11 cycle

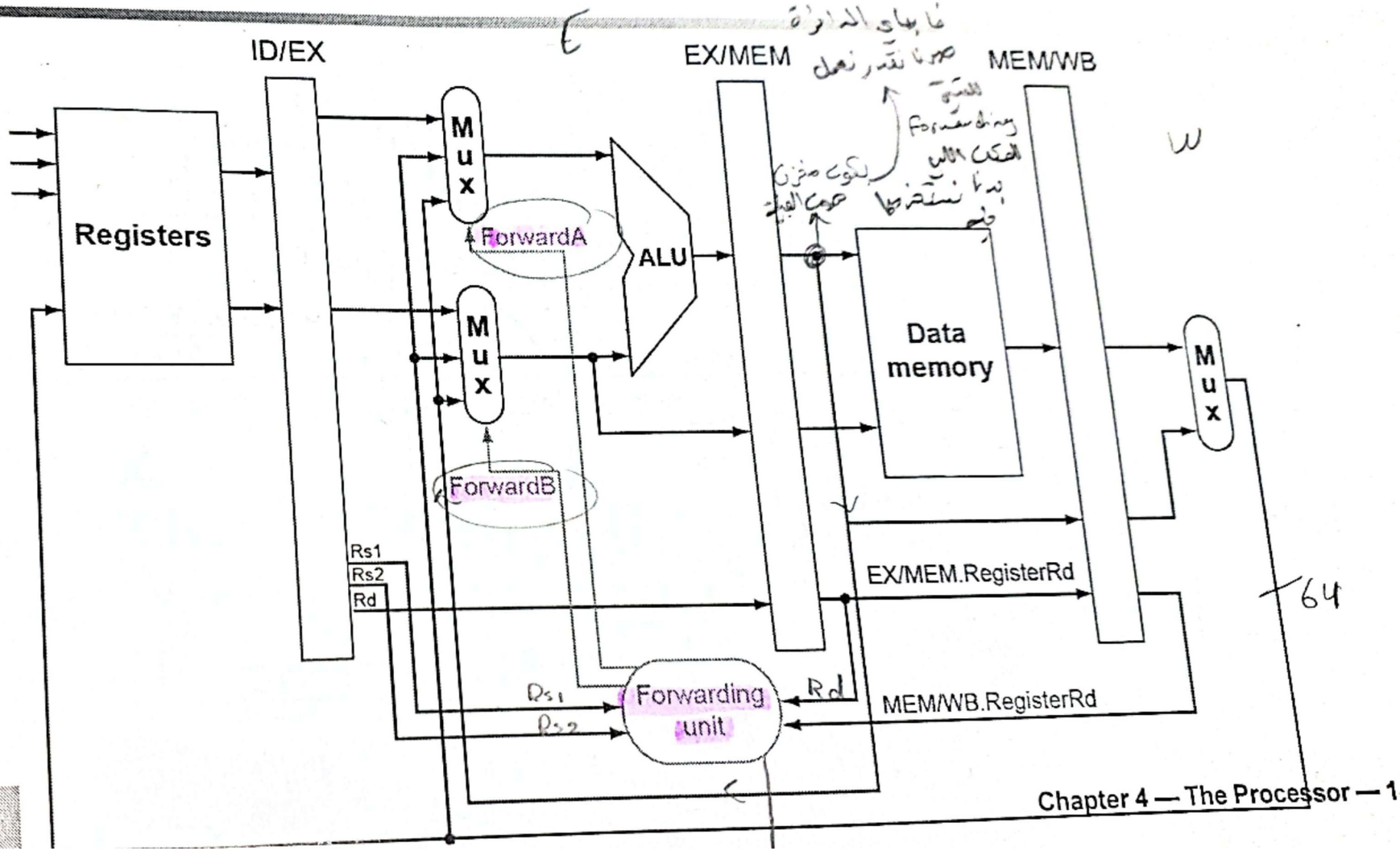
we need 4 cycles so we can fill the pipline and in the 5th cycle we will be finishing the first instruction

Dependencies & Forwarding



* الـ Hazard موجود بين الـ IC الأولى والثالثة وبين الأولى - والثالثة

Forwarding Paths



With Forwarding

* the new x_2 value we get it from the First instruction only by using Forwarding

producer ⇒ x_2 بينتج

		1	2	3	4	5	6	7	8	9	10
1	sub x2, x1,x3	F	D	E	M	W					
2	and x12,x2,x5		F	$D^{x_2}_{x_5}$	E	M	W				
3	or x13,x6,x2			F	D	E	M	W			
4	add x14,x2,x2				F	$D^{x_2}_{x_4}$	E	M	W		
5	sd x15,100(x2)					F	D	E	M	W	

Consumers يستهلكوا الداتا اللي x_2 طالعين

القديمة x_2

القديمة x_2

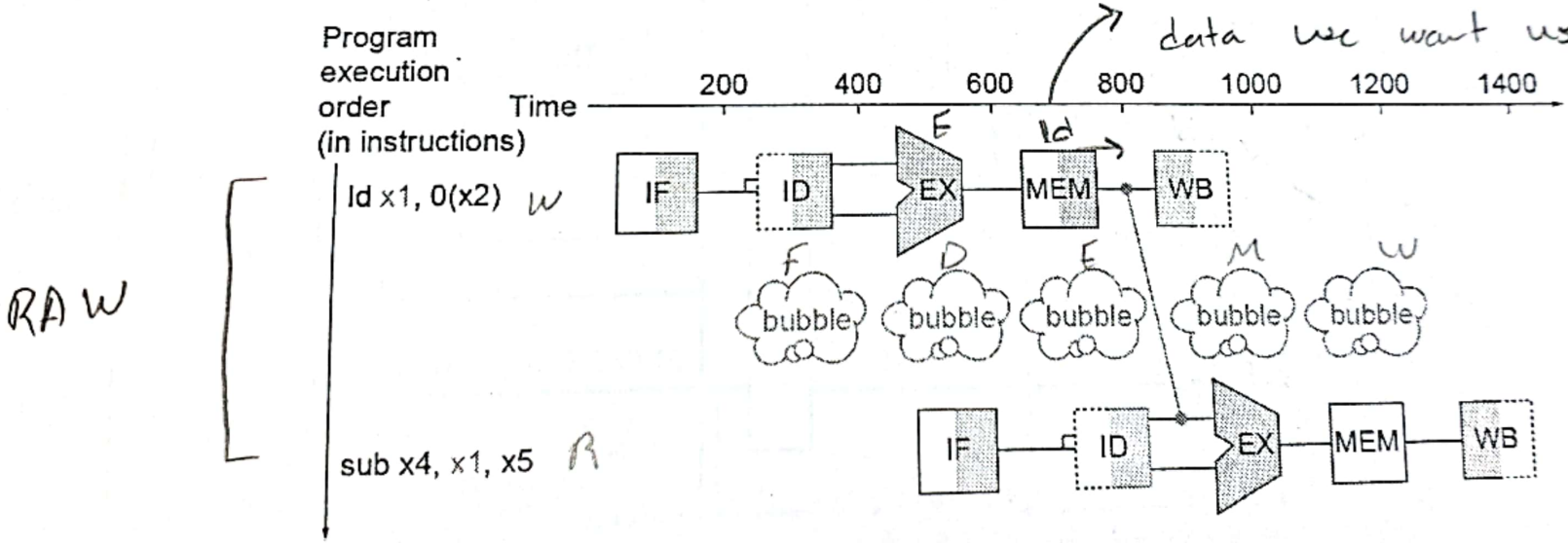
# of cycles = 4 + IC + Stalls

= 4 + 5 + 0 = 9 cycles

Because we used Forwarding in this instructions!

Load-Use Data Hazard

- Can't always avoid stalls by forwarding
  - If value not computed when needed
  - Can't forward backward in time!



we can't use Forwarding because the data we want use, didn't get out from the memory in the memory stage in this IC

so we must Stall

Load-Use Data Hazard

D E M

↑ bubble = no operation

data available هون بتميز ال

	1	2	3	4	5	6	7	8	9	10
ld x1, 0(x2)	F	D	E	M	W					
sub x4,x1,x5		F	D	D	E	M	W			

(x1)

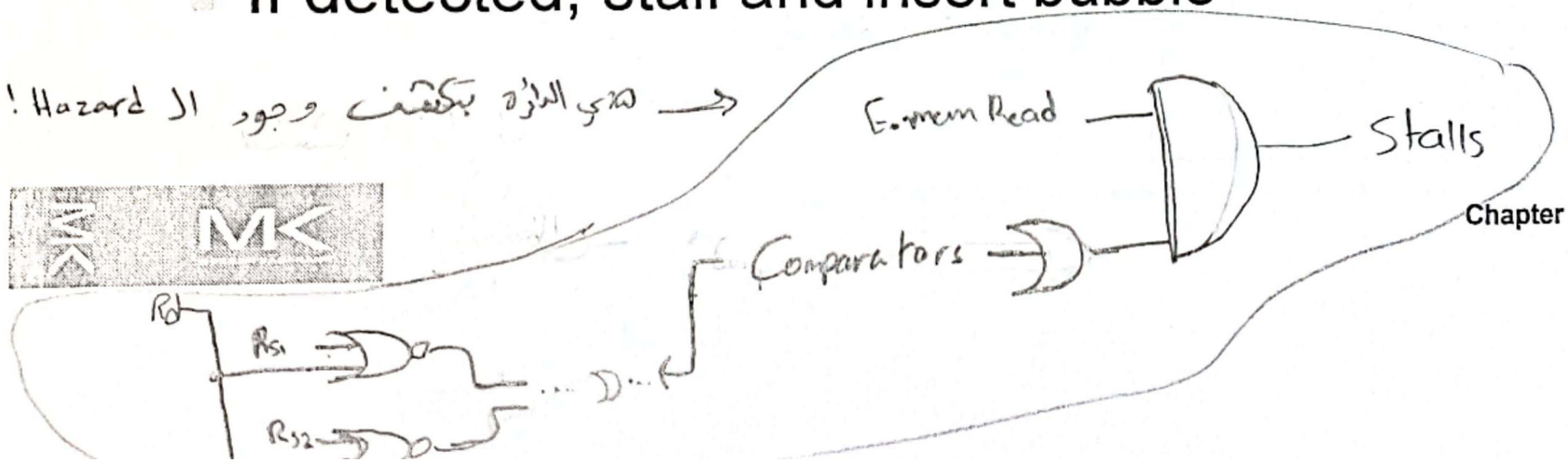
# of cycles = 4 + 2 + (1) = 7 cycles

stalls

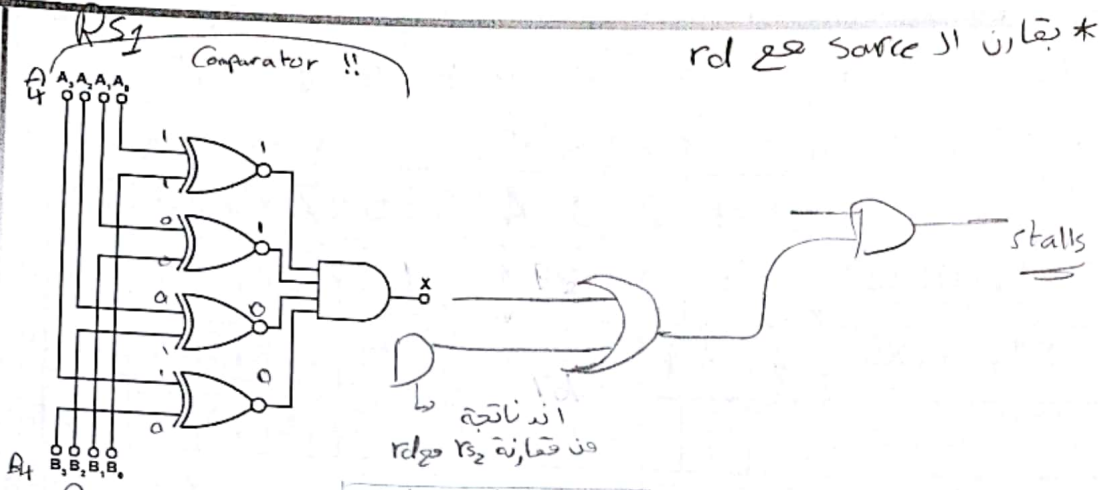
* اذا استخدمت القيمة بعد الـ E أو الـ M بكون عندي عليها forwarding

Load-Use Hazard Detection

- Check when using instruction is decoded in ID stage
- ALU operand register numbers in ID stage are given by
  - IF/ID.RegisterRs1, IF/ID.RegisterRs2
- Load-use hazard when
  - ID/EX.MemRead and
    ((ID/EX.RegisterRd = IF/ID.RegisterRs1) or
    (ID/EX.RegisterRd = IF/ID.RegisterRs2))
- If detected, stall and insert bubble



Stall Circuit



* بقارن ال source مع rd

انه ناتجة
من مقارنة rs2 مع rd

A	B	C
0	0	1
0	1	0
1	0	0
1	1	1

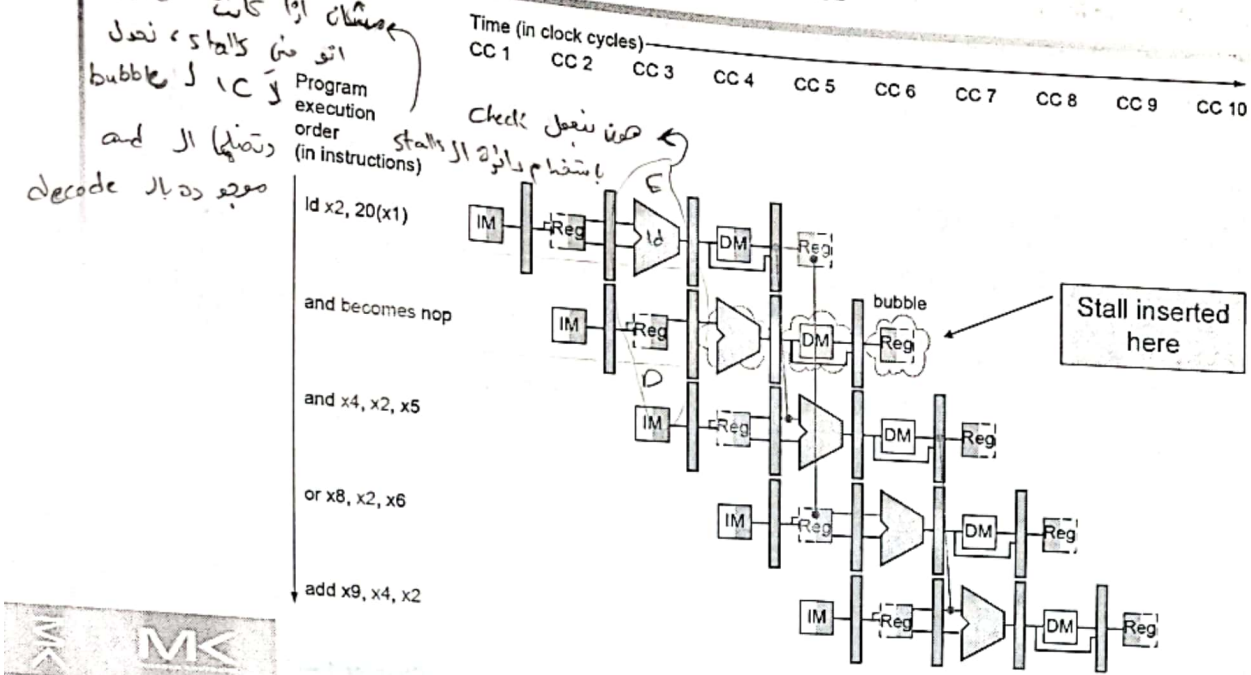
عدد الريجسترز = 32
لذلك انا بحاجة 5 bits

How to Stall the Pipeline

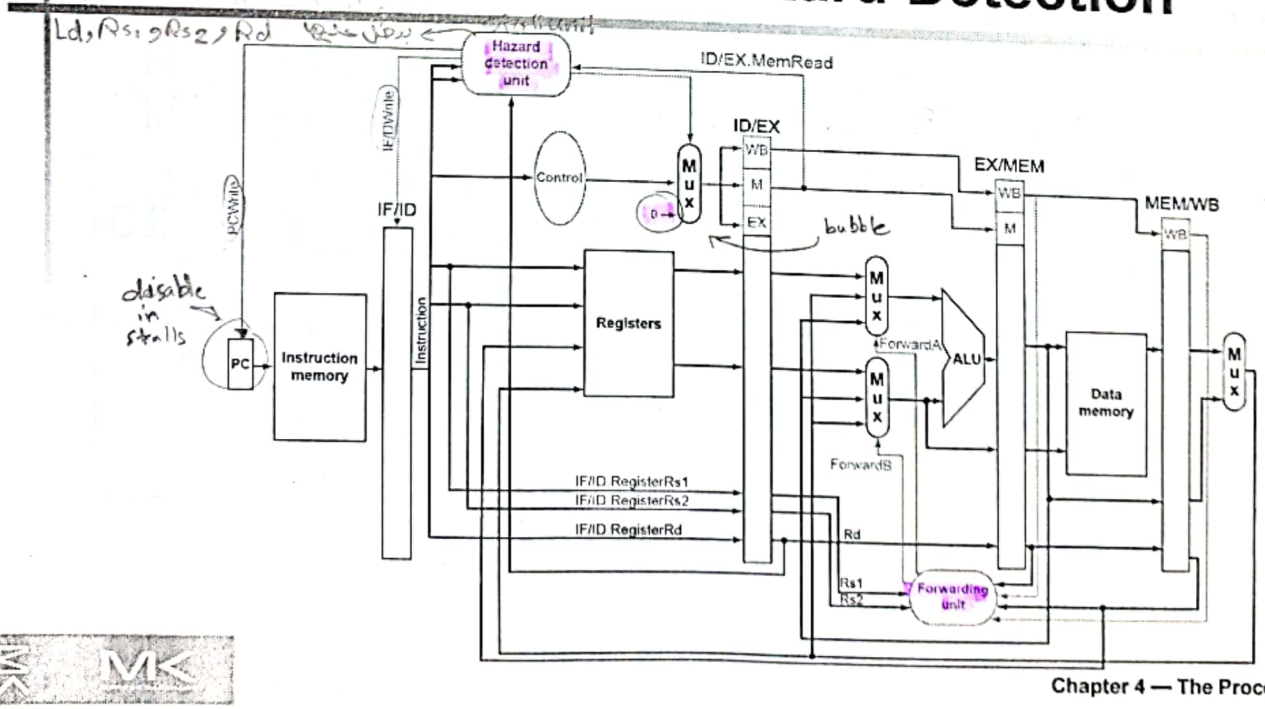
- Force control values in ID/EX register to
  - EX, MEM and WB do nop (no-operation)
- Prevent update of PC and IF/ID register
  - Using instruction is decoded again
  - Following instruction is fetched again
  - 1-cycle stall allows MEM to read data for ld
    - Can subsequently forward to EX stage

* بقدر أوقف عمل ال
سيكوانشال سيركت فقط
اللي تحتوي على FlipFlop
* عن طريق انه البلوك الاول
نصفرها
* او أوقفها عن طريق
EN وأعلها Enable (EN)
disable

Load-Use Data Hazard



Datapath with Hazard Detection



Rearranging to solve Load-Use Data Hazard

	1	2	3	4	5	6	7	8	9	10
ld x1, 0(x2)	F	D	E	M	W					
sub x4,x1,x5		F	D	D	E	M	W			
add x7,x5,x6			F	F	D	E	M	W		

وقفت بسبب disable PiP

اذا بطلناهم بنحل مشكلة stalls ال

وقفت بسبب disable counter

4 + 3 + 1 = 8 cycles

Rearranging to solve Load-Use Data Hazard

	1	2	3	4	5	6	7	8	9	10
ld x1, 0(x2)	F	D	E	M	W					
add x7,x5,x6		F	D	E	M	W				
sub x4,x1,x5			F	D	E	M	W			

4+3+0 = 7 cycles

Code Scheduling to Avoid Stalls

- Reorder code to avoid use of load result in the next instruction
- C code for a = b + e; c = b + f;

= 7 instructions

locations

b 0	c 32
e 8	
f 16	
a 24	

stalls ← data dependence = (load use) αλλαγή σειράς εντολών

```
ld   x1, 0(x0)
ld   x2, 8(x0)
add  x3, x1, x2
sd   x3, 24(x0)
ld   x4, 16(x0)
add  x5, x1, x4
sd   x5, 32(x0)
```

load dep → stall

load dep → stall

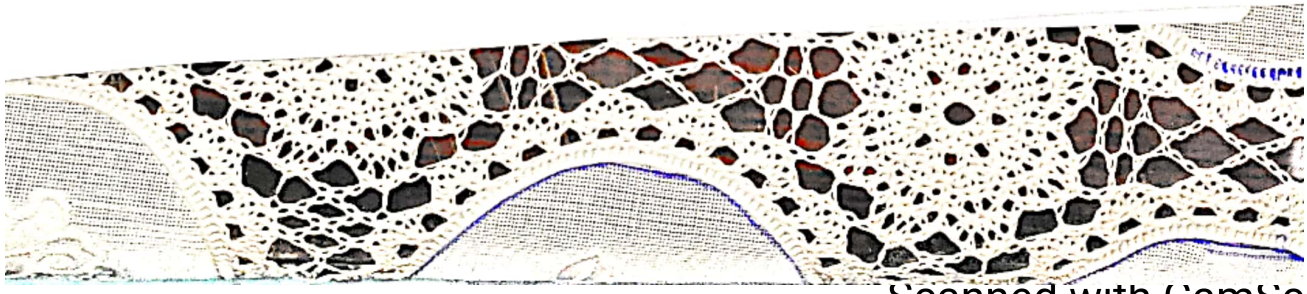
13 cycles

4 + 7 + 2 = 13

```
ld   x1, 0(x0)
ld   x2, 8(x0)
ld   x4, 16(x0)
add  x3, x1, x2
sd   x3, 24(x0)
add  x5, x1, x4
sd   x5, 32(x0)
```

11 cycles

4+7+0=11

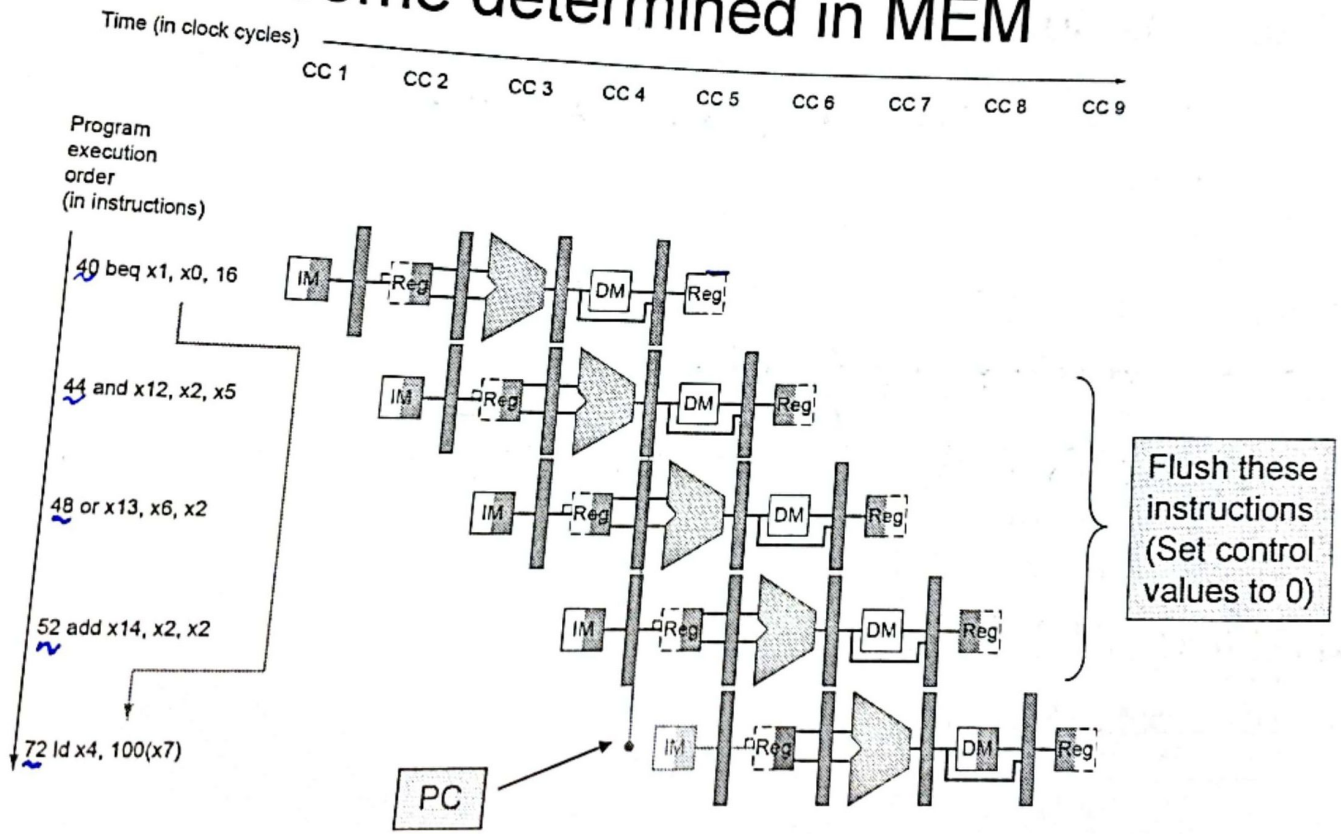


4.8 Control Hazards

- Branch Hazards
- Reducing Branch Delay
- Branch Prediction
- Dynamic Branch Prediction → Predictive and active!
- Calculating Branch Target → address for the target direction

Branch Hazards

- If branch outcome determined in MEM



Solving branches in the Memory stage

40 + 16 * 2 = 72

Assume taken branch

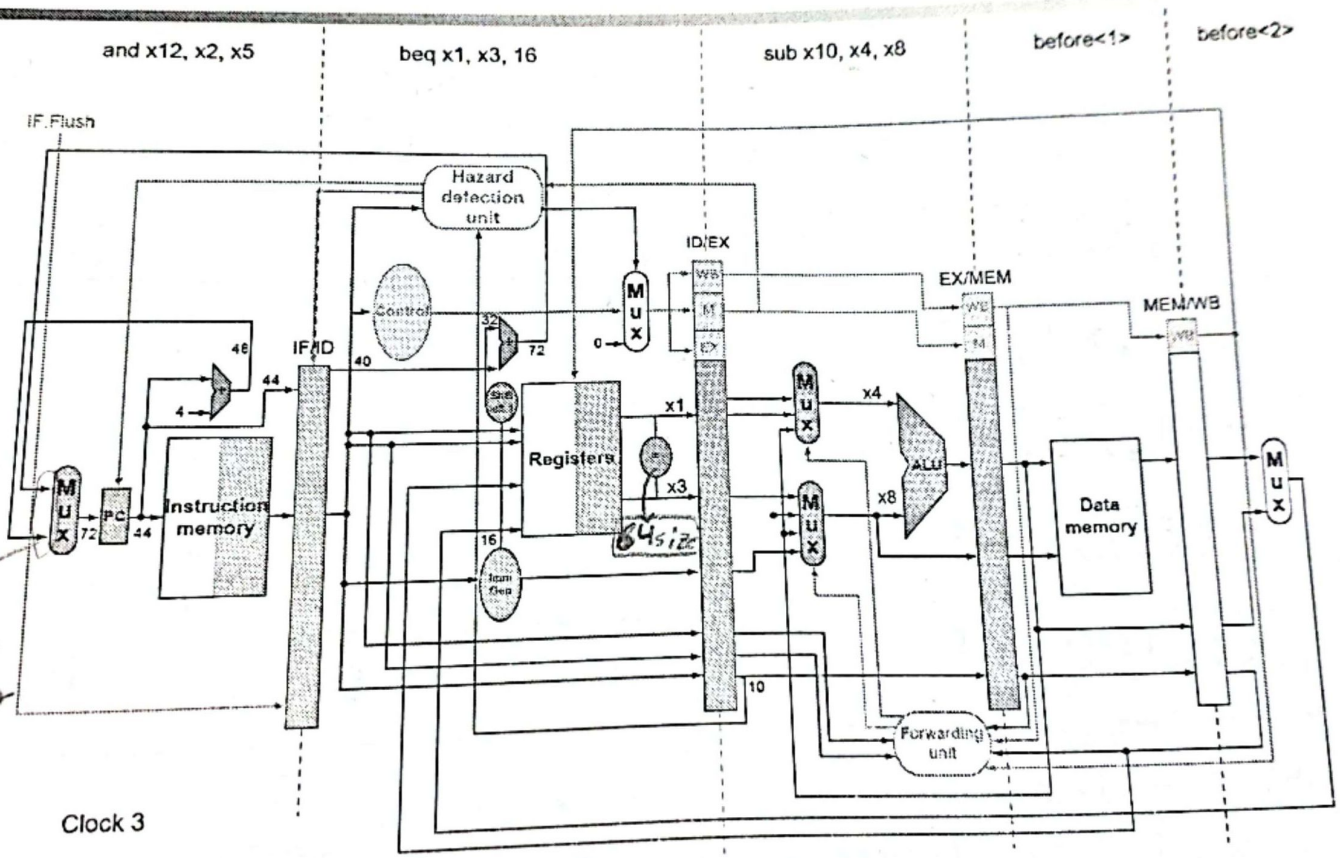
n = bubble (no operation)

adress		1	2	3	4	5	6	7	8	9	10
40	beq x1,x0,16	F	D	E	M	W					
44	and x12,x2,x5		F	D	E	n	n				
48	or x13,x6,x2			F	D	n	n	n			
52	add x14,x2,x2				F	n	n	n	n		
72	ld x4,100(x7)						F	D	E	M	W

4 + 2 + (3) = (9) cycles

Branch Delay Cycles

Example: Branch Taken



*size comparator = 64 bit

* نقل الـ shift عن stay decode إلى Ex

اللي بحدد ايش ياخد Mux هو سجنال الـ comparator

* ماني طالي ادخلهم عالـ Ex واشوف الزيرو flag

Solving branches in the Decode stage

* Assume taken branch

* بدون البرانش MEM stage في Taken or not

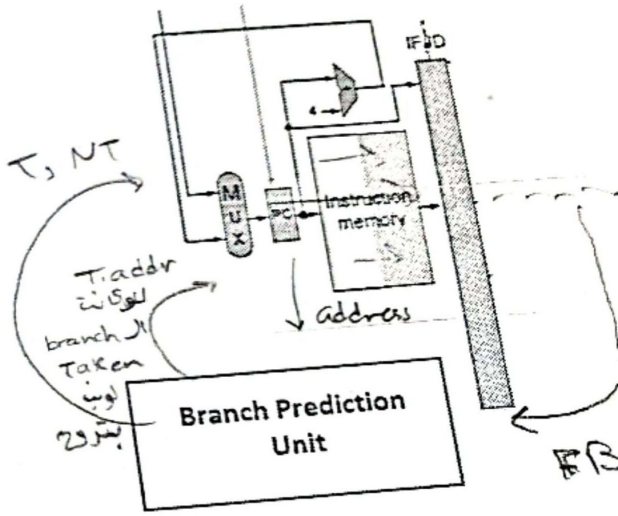
address		1	2	3	4	5	6	7	8	9	10
40	beq x1,x0,16	F	D	E	M	W					
44	and x12,x2,x5		F	n	m	n	n	→ bubbles			
48	or x13,x6,x2										
52	add x14,x2,x2										
72	ld x4,100(x7)			F	D	E	M	W			

(4 + 6 * 2) = 72

4 + 2 + (1) = 7 cycles.

Branch Prediction

- Longer pipelines can't readily determine branch outcome early
  - Stall penalty becomes unacceptable
- Predict outcome of branch
  - Only stall if prediction is wrong
- In RISC-V pipeline
  - Can predict branches not taken
  - Fetch instruction after branch, with no delay



BPU

Predict Not Taken

Solving branches in the Decode stage
Assume branch is not taken.

		1	2	3	4	5	6	7	8	9	10
	beq x1,x0,L	F	D	E	M	W					
	I2		F	D	E	M	W				
						No Delay!					
L	IT										

↓ we predict that it's Not Taken

Predict Not Taken

4+2 +0 = 6

4+2+1 = 7

Solving branches in the Decode stage
Assume branch is taken.

		1	2	3	4	5	6	7	8	9	10
	beq x1,x0,L	F	D	E	M	W					
	I_2		F	n	n	n	n				
L	I_T			F	D	E	M	W			

4+2+1 = (7) cycles

Avg Branch Delay (BD) = 0.5

branch delay كبير طويل زي طلع ال

More-Realistic B

- Static branch prediction → static
  - Based on typical branch behavior
  - Example: loop and if-statement brar
    - Predict backward branches taken
    - Predict forward branches not taken
- Dynamic branch prediction
  - Hardware measures actual branch b
    - e.g., record recent history of each branc
  - Assume future behavior will continue
    - When wrong, stall while re-fetching, and

static branch.p ؟

```
if (x==0)
  cout << x ;
else
  cout << x*2 ;
```

⇒ branch موجود

* مفترض انه not taken

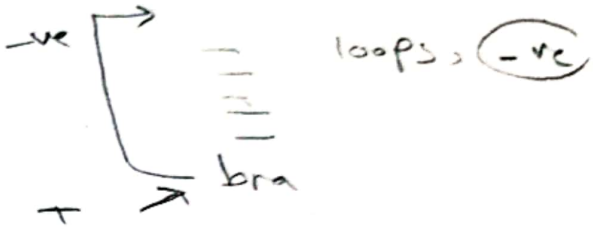
وعا برجع عل else كتير

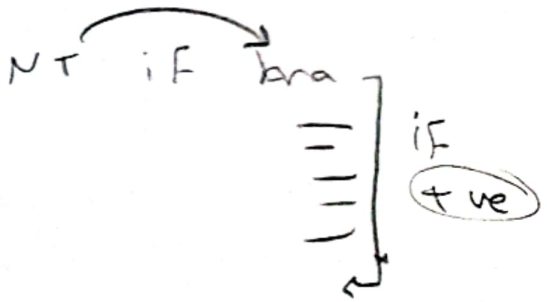
```
for(int=0; i<5; i++){
  ssg
}
```

Loop negative predict ta

More-Realistic Branch Prediction

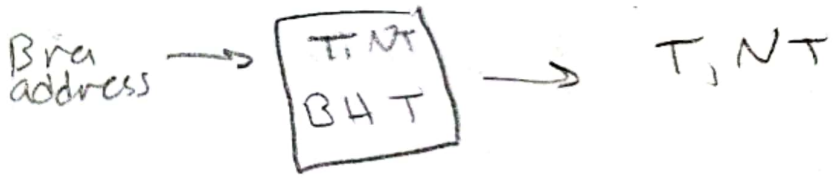
- Static branch prediction
  - Based on typical branch behavior
  - Example: loop and if-statement branches
    - Predict backward branches taken
    - Predict forward branches not taken
- Dynamic branch prediction
  - Hardware measures actual branch behavior
    - e.g., record recent history of each branch
  - Assume future behavior will continue the trend
    - When wrong, stall while re-fetching, and update history





Dynamic Branch Prediction

- In deeper and superscalar pipelines, branch penalty is more significant
- Use dynamic prediction
  - Branch prediction buffer (aka branch history table)
  - Indexed by recent branch instruction addresses
  - Stores outcome (taken/not taken)
  - To execute a branch
    - Check table, expect the same outcome
    - Start fetching from fall-through or target
    - If wrong, flush pipeline and flip prediction



Branch History Table (BHT)

$2^{12} = 4K$ $\quad 2^{10} = K$, $2^{20} = M$, $2^{30} = G$

One-Level Branch Predictor

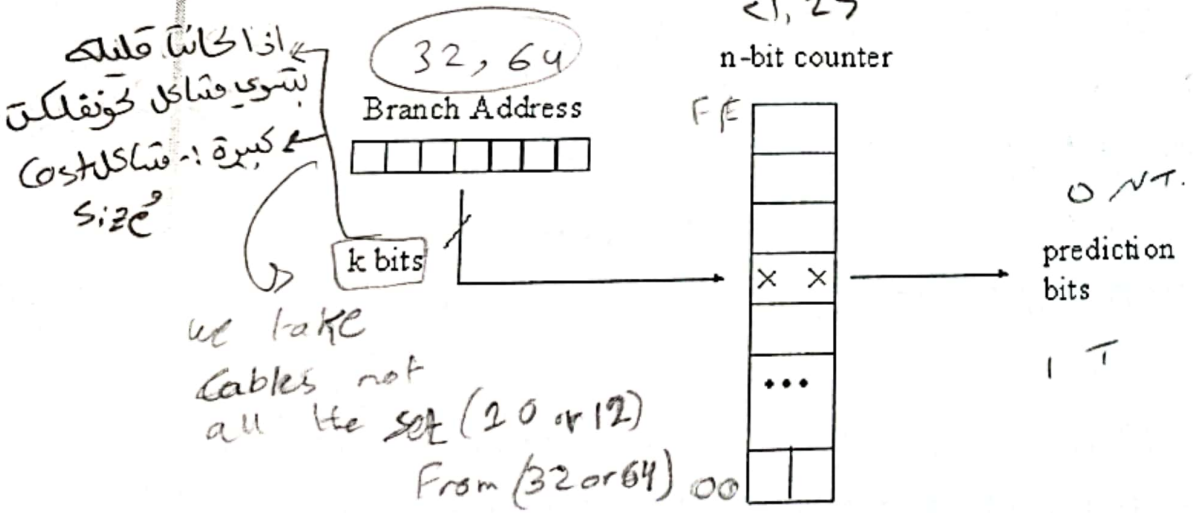
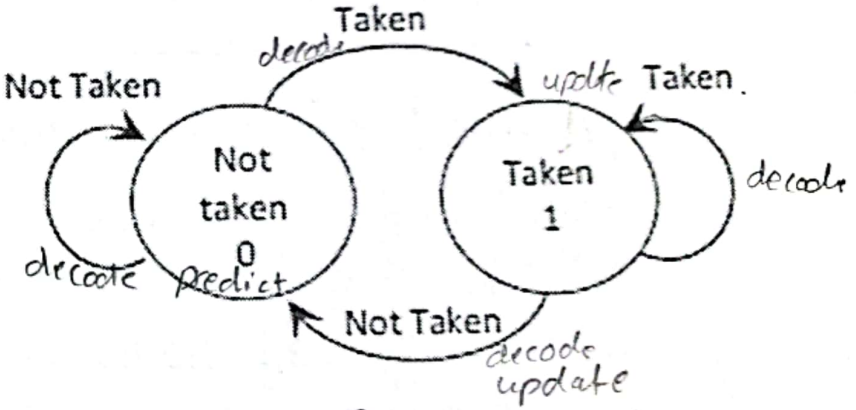


Table size = n × 2^k bits

$= 2 \times 2^{12}$

$= 8$ K bits

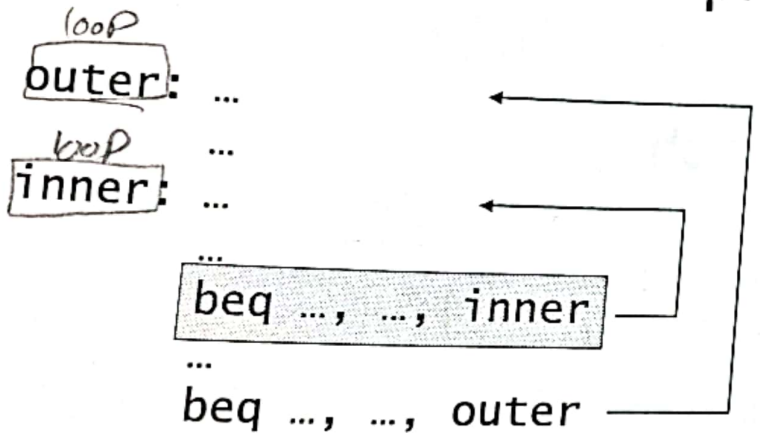
1-bit predictor



85% – 90% accurate

1-Bit Predictor: Shortcoming

- Inner loop branches mispredicted twice!



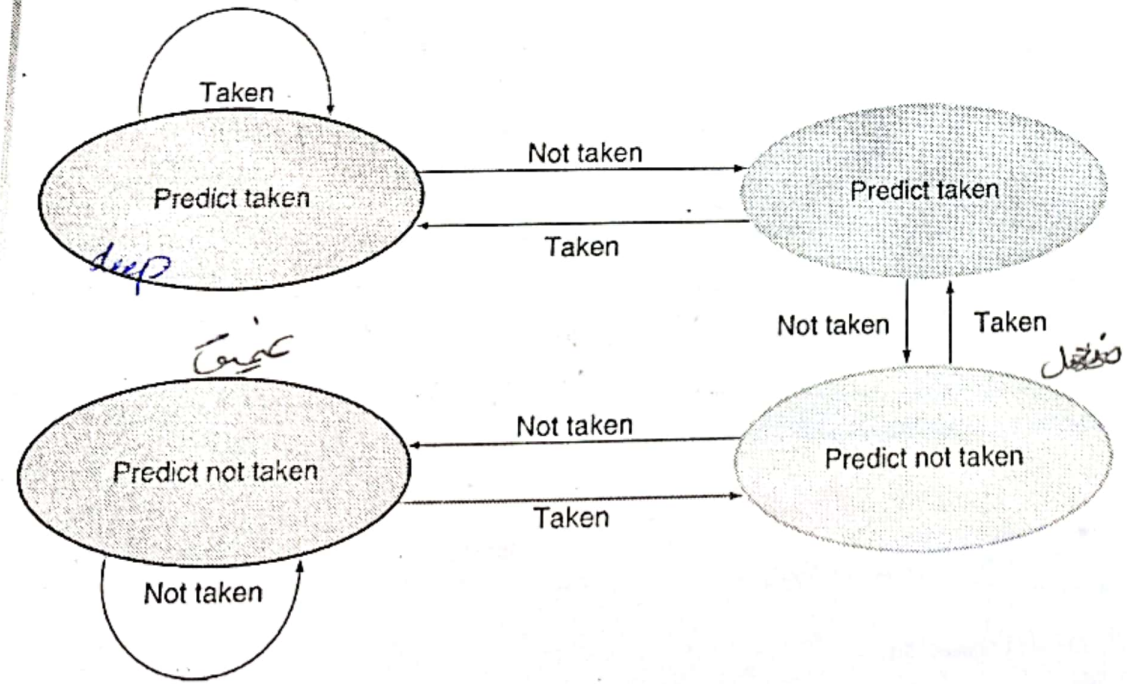
Iteration	997	998	999	0	1
Prediction	T	T	T	NT	T
Result	C	C	I	I	C

T: Taken
NT: Not Taken
C: Correct prediction
I: Incorrect prediction

- *Mispredict as taken on last iteration of inner loop*
- *Then mispredict as not taken on first iteration of inner loop next time around*

2-Bit Predictor

- Only change prediction on two successive mispredictions



Iteration	997	998	999	0	1
Prediction	T	T	T	T	T
Result	C	C	I	C	C

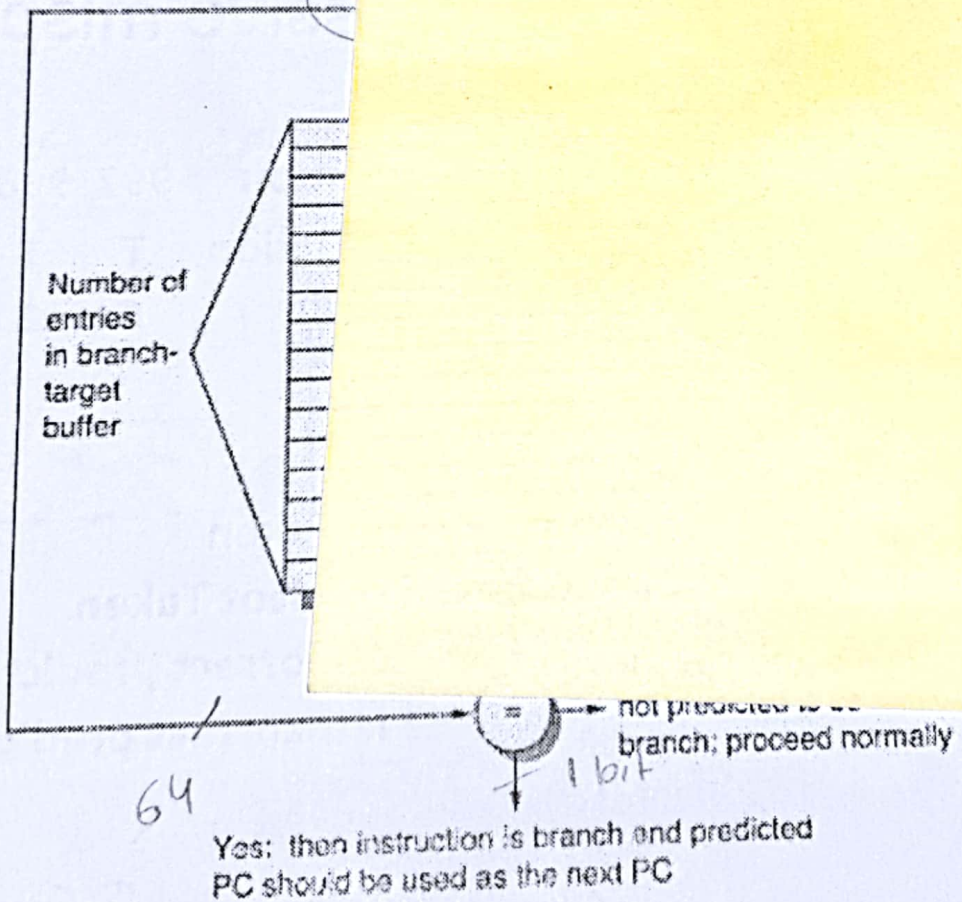
T: Taken
NT: Not Taken
C: Correct prediction
I: Incorrect prediction

90% - 95% accuracy.

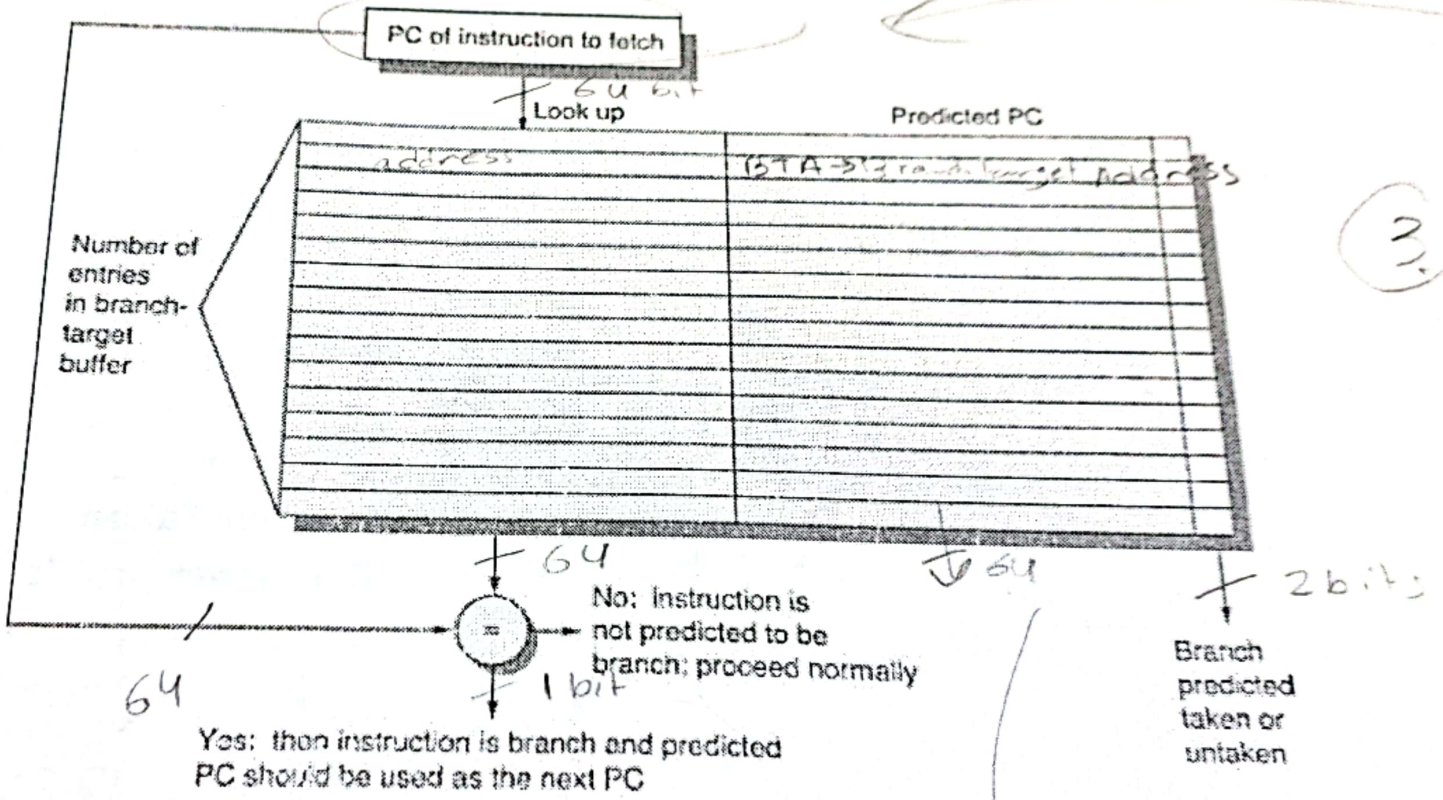
Branch Targe

K=10
R64VI Arch
Instruction 64 bits

total size = 1024 X 64 X 64 X 2



Branch Target Buffer (BTB)



BTA و BTB : اذا لفيت بال

اللي عم بدور عليو انا

Exceptions and Interrupts

- "Unexpected" events requiring change in flow of control
  - Different ISAs use the terms differently

✓ Exception ⇒ سببه الهاردوير تبعلة أو مشاكل داخل

Syscall ⇒ عشان بدي انادي كود فانداو operand
عن شكل functions

Interrupt: I/O → مربوطين بسيركت محددة وظيفتها
باللحظة تبلغ ال بروسيس انه وصلني input من مصدر خارجي

exception:
fEtch stag: مشكلة في
ممكن واحد من ال bits adress
يحيل flip فيروح د adress مختلف

decode stag: undefined opcode
7bits → 128 instruction بتحمل
فممكن تقلب bits ويوديني على opcode انا مش مشترك أو معرفو

Ex: inst1 32bit unsigned
inst2 32 bit unsigned
⇒ ناتج جمعهم أكبر من مدى ال unsigned
= overflow

Handling Exceptions

- Save PC of offending (or interrupted) instruction
  - In RISC-V: Supervisor Exception Program Counter (SEPC)
- Save indication of the problem
  - In RISC-V: Supervisor Exception Cause Register (SCAUSE)
  - 64 bits, but most bits unused
    - Exception code field: 2 for undefined opcode, 12 for hardware malfunction, .
- Jump to handler (ISA)
  - Assume at $0000\ 0000\ 1C09\ 0000_{hex}$

An Alternate Mechanism

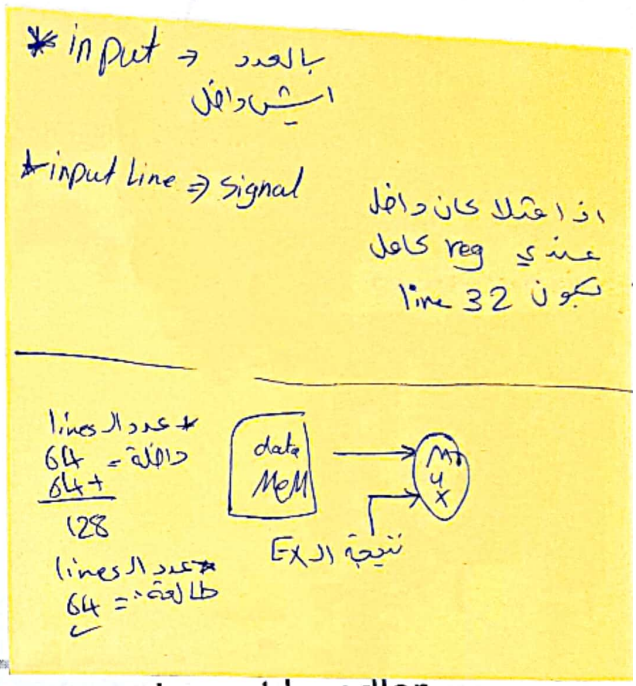
- Vectored Interrupts ⇒
  - *Handler address determined by the cause*
- *Exception vector address to be added to a vector table base register:*
  - *Undefined opcode* $00\ 0100\ 0000_{two}$
  - *Hardware malfunction:* $01\ 1000\ 0000_{two}$
  - *...:* ...
- *Instructions either*
  - *Deal with the interrupt, or*
  - *Jump to real handler*

An Alternate Mechanism

I1		
I2	Undefined opcode	$00\ 0100\ 0000_{two}$
I3	Hardware malfunction:	$01\ 1000\ 0000_{two}$
I4	...:	...
I5		

SEPC, SCAUSE

MK



Handler Actions

- Read cause, and transfer to relevant handler
- Determine action required

- If restartable
  - Take corrective action
  - use SEPC to return to program
- Otherwise
  - Terminate program
  - Report error using SEPC, SCAUSE, ...

MK

Exceptions in a Pipeline

- Another form of control hazard
- Consider malfunction on add in EX stage
  - `add x1, x2, x1`
  - Prevent x1 from being clobbered
  - Complete previous instructions
  - Flush add and subsequent instructions
  - Set SEPC and SCAUSE register values
  - Transfer control to handler
- Similar to mispredicted branch
  - Use much of the same hardware

Exceptions in a Pipeline

ExcOverFlow

		1	2	3	4	5	6	7	8	9	10	11	12	13
✓	I1	F	D	E	M	W								
✗	add x1,x2,x1		F	D	E	n	n							
✗	I3			F	D	n	n	n	"flush"					
	I4				F	n	n	n	n	"flush"				
	I5													
IHS	✓					F	D	E	M	W				
	✓						F	D	E	M	W			
	✓							f	D	E				
	✓								F	D			W	

* الإضافات
* قبل كل stage في Mux

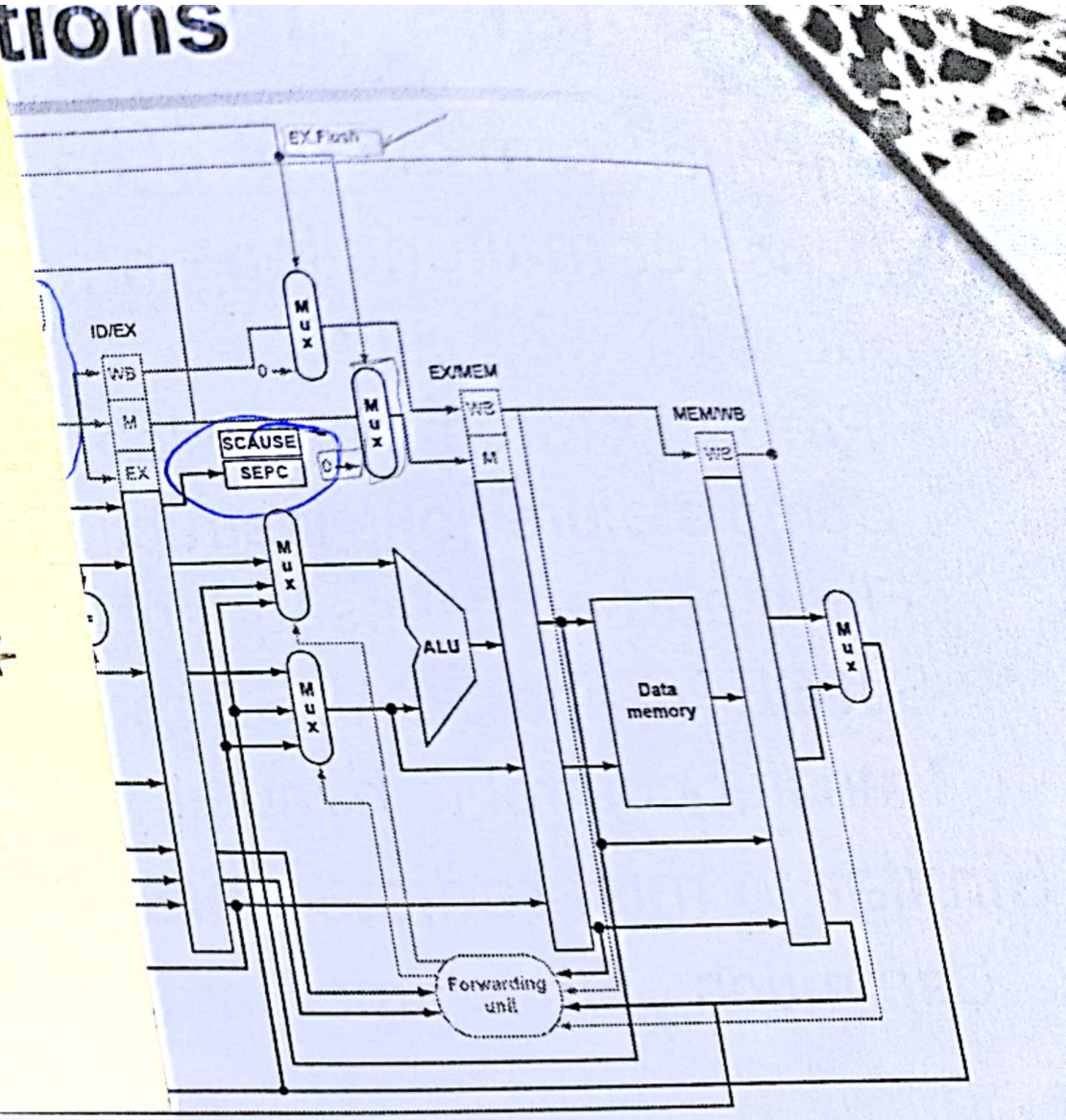
يأخذ القيمة الأصلية أو قيمة الـ Flush

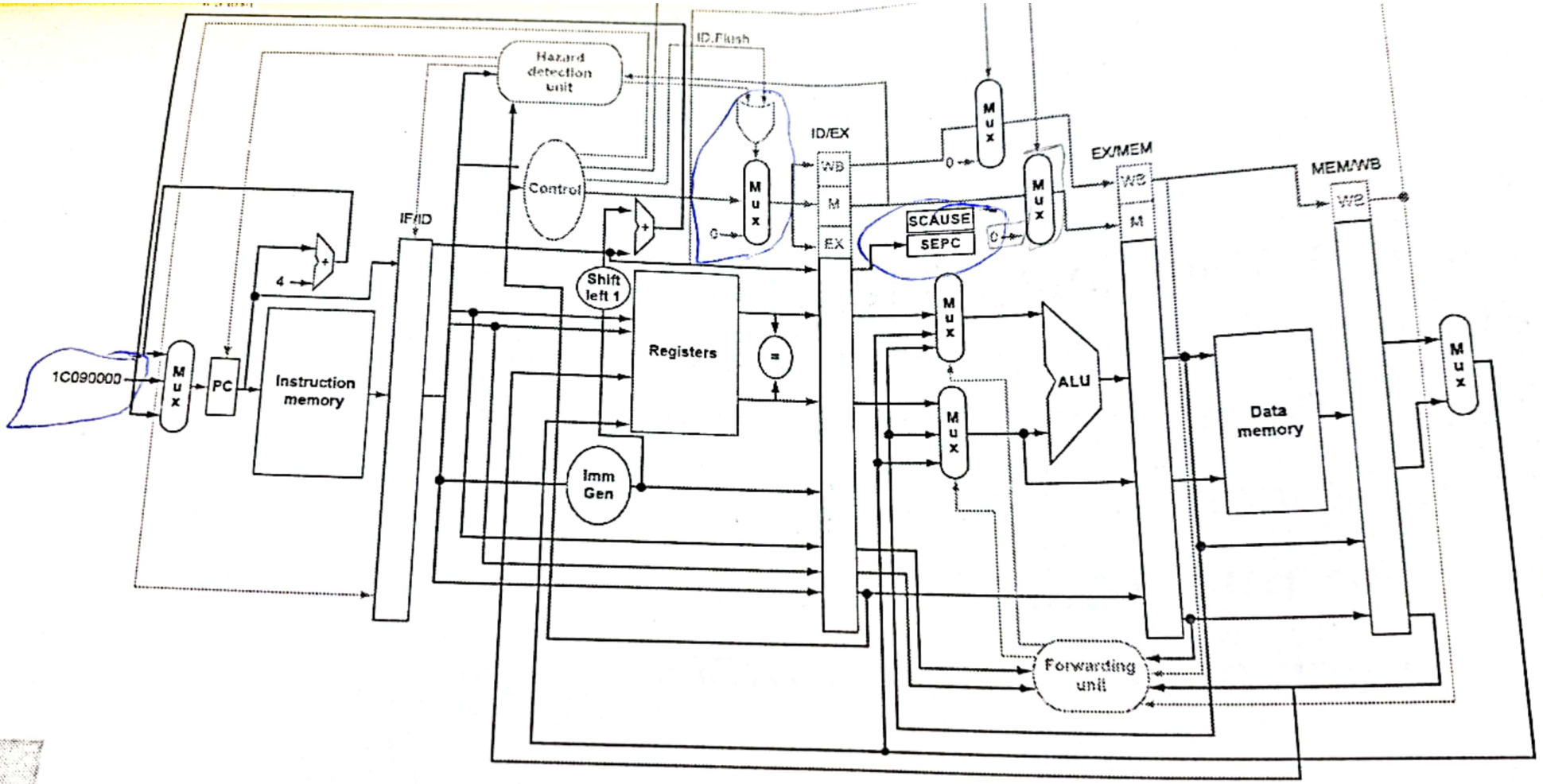
* في مرحلة Ex SCAUSE SEPC

وصول بسلك من الحراس القبل

* pc عند الـ 1C090000

PC+4 → Mux → PC
branch →
1C0... →





Exception Example

- Exception on add in

```
40      sub   x11, x2, x4
44      and   x12, x2, x5
48      or    x13, x2, x6
4c      add   x1,  x2, x1
50      sub   x15, x6, x7
54      ld    x16, 100(x7)
...
```

IHS Handler

```
1c090000    sd   x26, 1000(x10)
1c090004    sd   x27, 1008(x10)
...
```

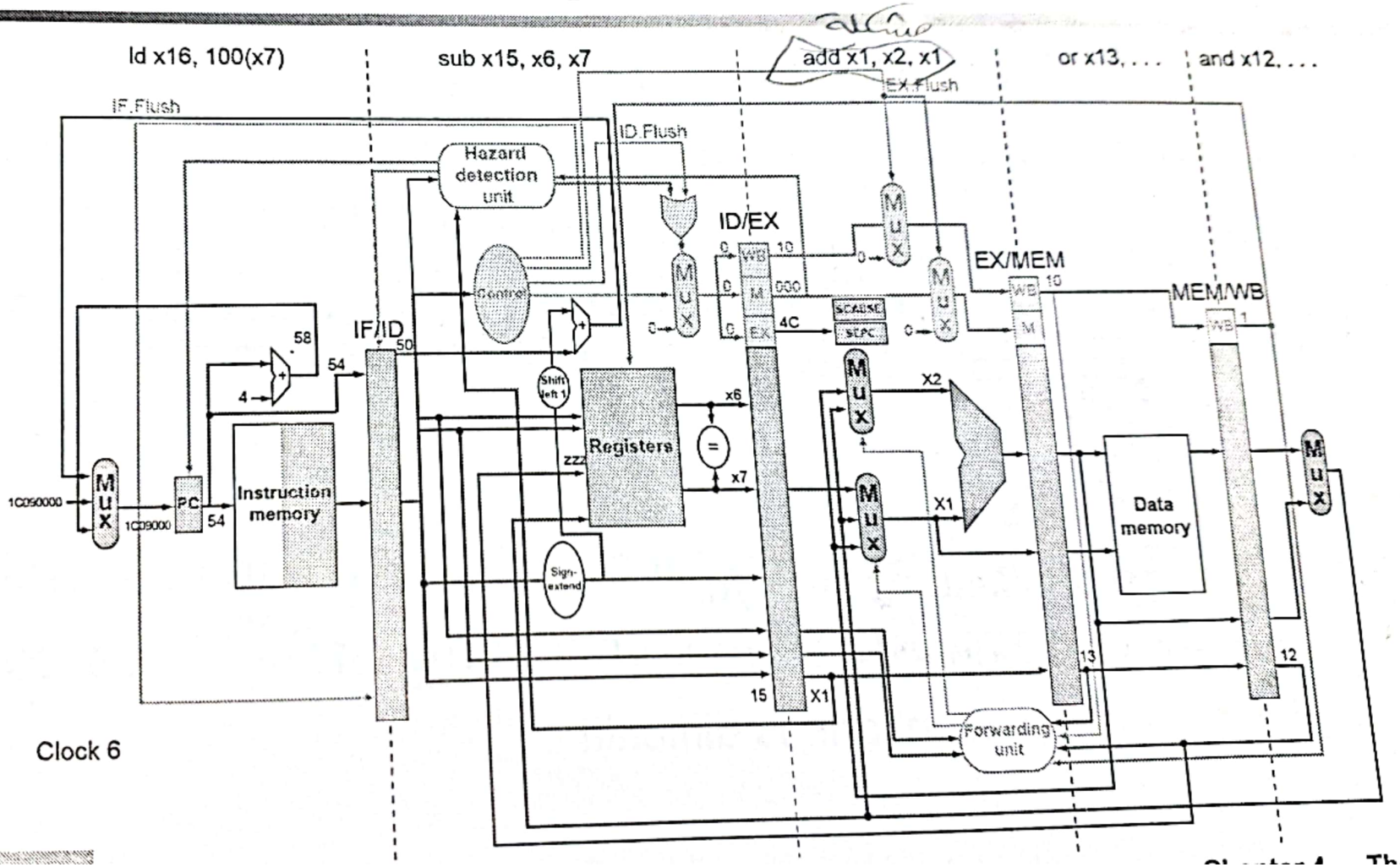
PC+4

]-> context switchin

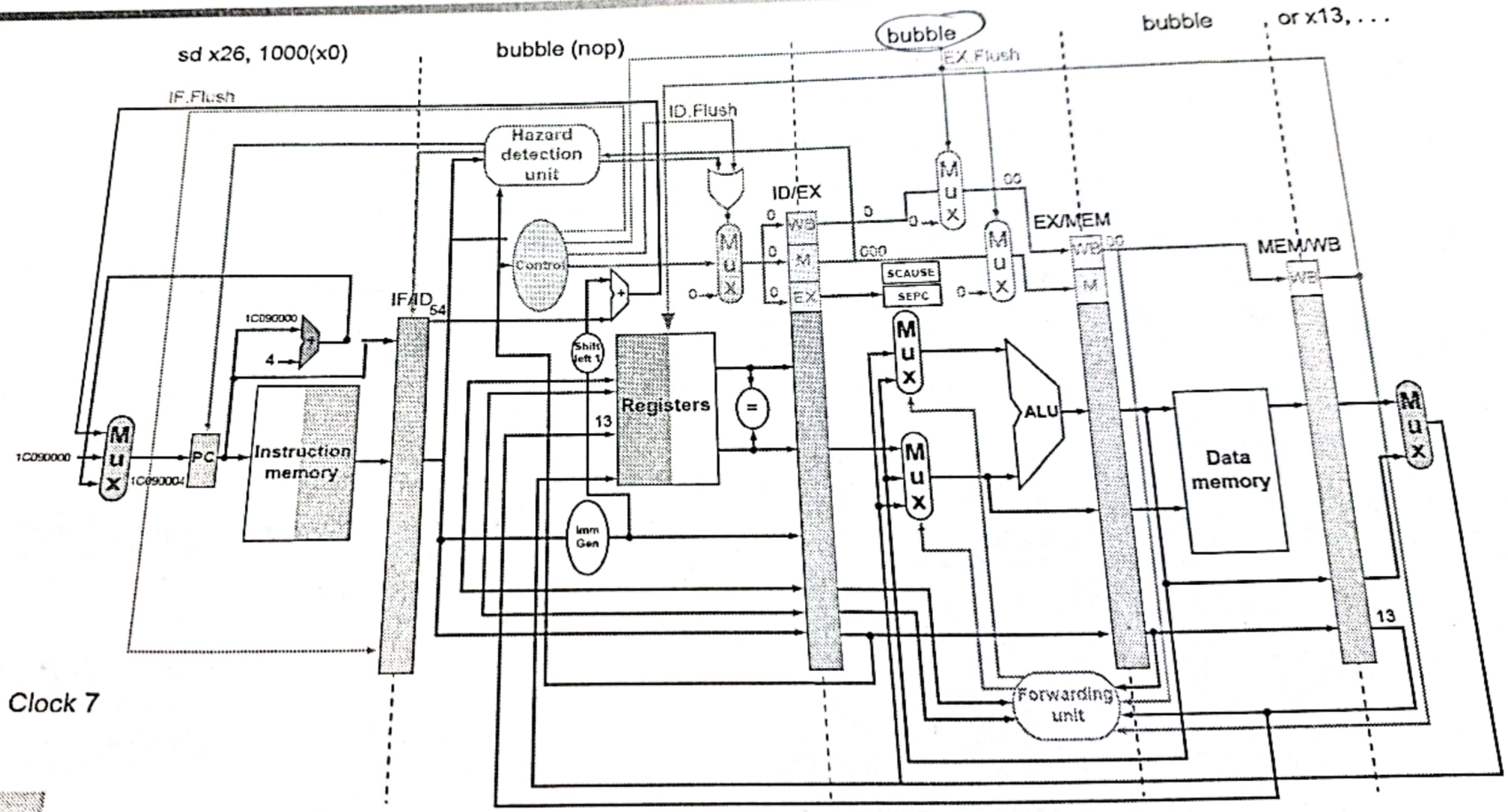
$(x_0 - x_{31})$ → فيها قيم تابعة للبرنامج الأصلي

... القيم الأصلية
... عنوان

Exception Example



Exception Example



Multiple Exceptions

- Pipelining overlaps multiple instructions
  - Could have multiple exceptions at once
- Simple approach: deal with exception from earliest instruction
  - Flush subsequent instructions
  - "Precise" exceptions
- In complex pipelines
  - Multiple instructions issued per cycle
  - Out-of-order completion
  - Maintaining precise exceptions is difficult!

		1	2	3	4	5	6	7	8	9	10	11	12	13
	I1	F	D	E	M	W								
	add x1,x2,x1		F	D	E→M	n								
	I3 (bad)			F	D→E	n	n							
	I4				F	D	n	n	n	flush				
	I5					F	n	n	n	n flush				
IHS							F	D	→					
								F						

* الاولوية للاي اضي بيس بال cpu
* الاولوية لل Exp اعلى من Intrp
* presize handle
* الاولوية لل add

* مشكلة ال cost عالية

(Exception اعلى priority)

		1	2	3	4	5	6	7	8	9	10	11	12	13
	I1	F	D	E	M	W								
	add x1,x2,x1		F	D	E→M	n								
	I3 (bad)			F	D→E	n	n							
	I4				F	D	n	n	n	flush				
	I5					F	n	n	n	n	flush			
IHS							F	D	→					
								F						

* الاولوية للي اضيف بعيس بالـ cpu
* مشكلة الـ cost عالي

* الاولوية لـ add
* presize handle
* Intr الى Exp

Exception أعلى priority أولوية

Imprecise Exceptions

- Just stop pipeline and save state
  - Including exception cause(s)
- Let the handler work out
  - Which instruction(s) had exceptions
  - Which to complete or flush
    - May require "manual" completion
- Simplifies hardware, but more complex handler software
- Not feasible for complex multiple-issue out-of-order pipelines

Contents

Instruction-Level Parallelism (ILP)

- Pipelining: executing multiple instructions in parallel
- To increase ILP
  - Deeper pipeline
    - Less work per stage ⇒ shorter clock cycle
  - Multiple issue
    - Replicate pipeline stages ⇒ multiple pipelines
    - Start multiple instructions per clock cycle → Ideal case
    - CPI < 1, so use Instructions Per Cycle (IPC) > 1
    - E.g., 4GHz 4-way multiple-issue →
      - 16 BIPS, peak CPI = 0.25, peak IPC = 4
    - But dependencies reduce this in practice

$\frac{C}{I} = 1$

cycle : inst2 ideal

$CPI = \frac{1}{2}$

$\text{execution time} = \frac{IS * CPI}{\text{Rate}}$

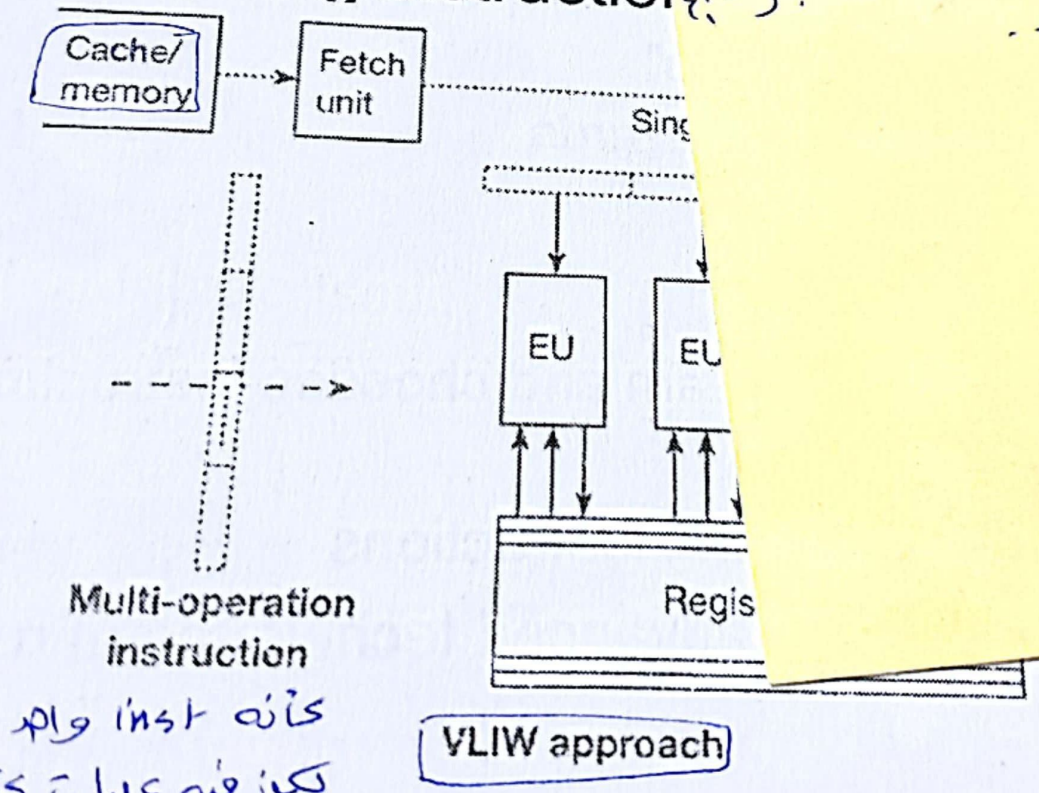
Multiple Issue

① Static multiple issue
- Compiler groups instructions to be issued together
- Packages them into "issue slots"
- Compiler detects and avoids hazards

② Dynamic multiple issue
- CPU examines instruction stream and chooses instructions to issue each cycle
- Compiler can help by reordering instructions
- CPU resolves hazards using advanced techniques at runtime

VILW

VLI
(very long instructio

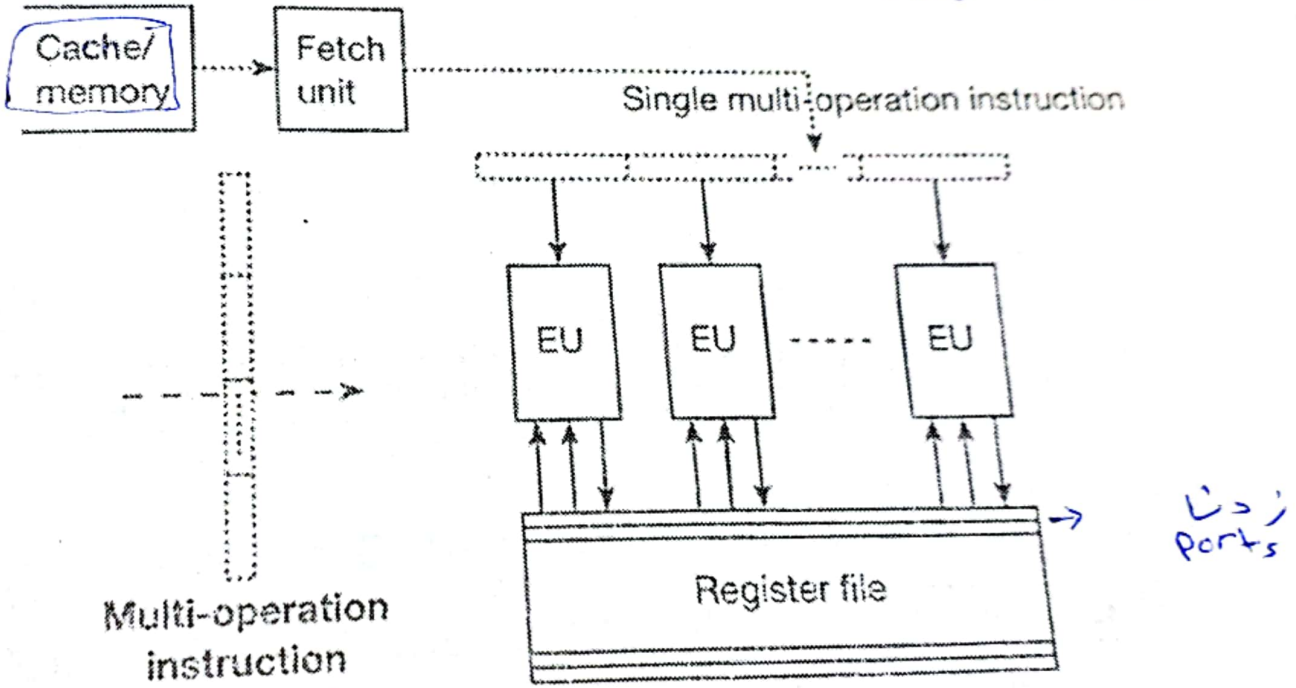


كأنه inst واحد
لكن فيه عمليات كثيرة

Risc
Register (Integer)
$(X_0 - X_{31}) \Rightarrow$

هل يتم عمل replication ؟
الجواب لا ؛ لانه اذا صار ريبليكشن لـ ريجستر فانهم
حينئذٍ X_0 يحل ، X_1 يحل ....

VLIW
(very long instruction word, 1024 bits!)



كأنه inst واحد
لكن فيه عمليات كثيرة

زيادة ports

⇒ Very Long Instruction Word (VLIW)

Static Multiple issue ما له بشبه ال
مثال Itanium

Ex: add x_1, x_2, x_3
sub x_4, x_1, x_5 → dependent ما بنقدر ننفذهم مع بعض
add x_6, x_7, x_8 → independent بقدر أنفذهم مع بعض وأحطهم ب issue packet

→ (add x_1, x_2, x_3 / add x_6, x_7, x_8) → issue Packets

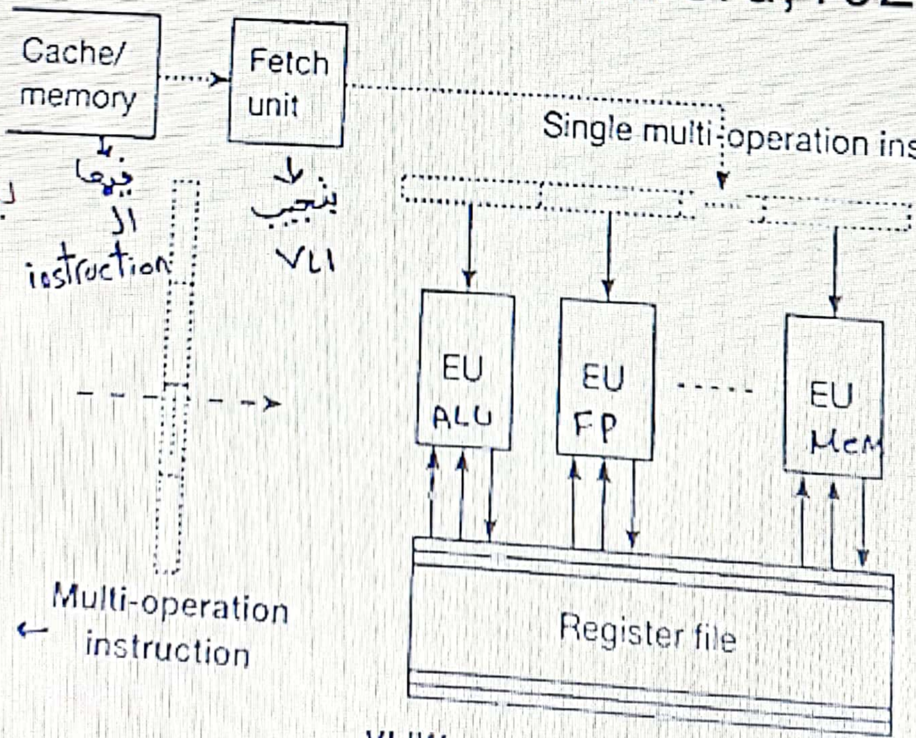
VILW

ممكن نعتبره نوع من ال Static Multiple issue → VLIW ال

instruction الوحدة أكبر بكثير من 32 bit

(very long instruction word, 1024 bits!)

وهاي ال instruction specify multiple concurrent operations يعني بتقول لل Processor اعمل أكثر من one operation بنفس الوقت (concurrent)



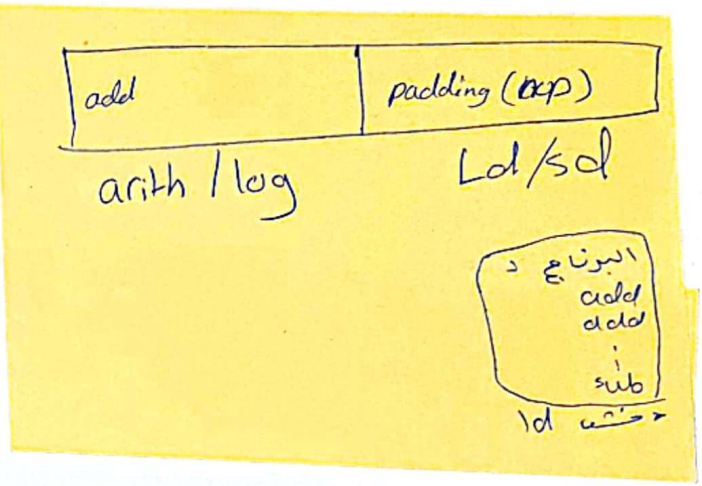
VLIW approach

← يعني instruction فيها Multiple operation بكون 128 or 256 bit

ال Compiler بحدد ايش ال operations الي ممكن ننفذهم مع بعض وحطهم ب VLI وكل operation بتنفذ على Execution unit خاصة

Scheduling Static Multiple Issue

- Compiler must remove some/all hazards
  - Reorder instructions into issue packets
  - No dependencies with a packet
  - Possibly some dependencies between packets
    - Varies between ISAs; compiler must know!
  - Pad with nop if necessary



64 → instruction 32 bits

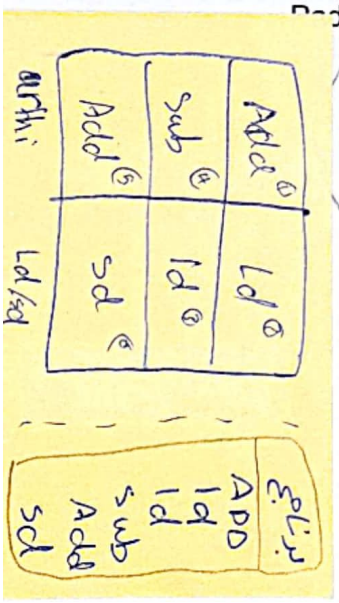
RISC-V with Static Dual Issue

- Two-issue packets
  - One ALU/branch instruction
  - One load/store instruction
  - 64-bit aligned
    - ALU/branch, then load/store
    - Pad an unused instruction with nop

61/8 = 8 bits

PC ← PC+4
PC ← PC+8
2 instructions with 4 bits for each!!

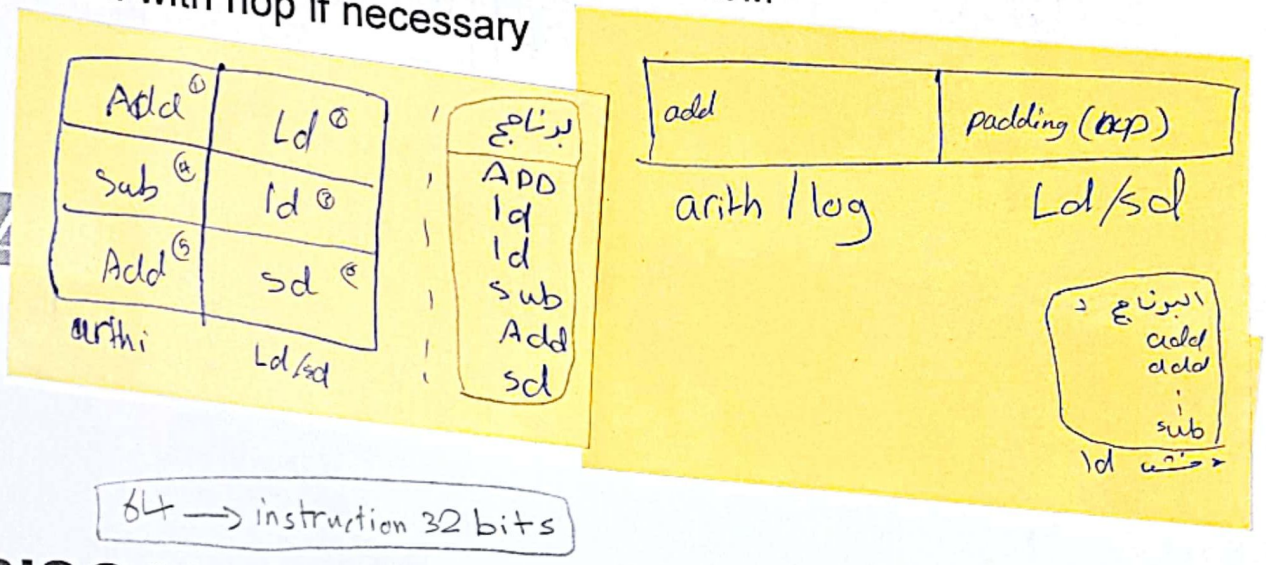
→ (7) cycles



Address	Instruction type	Pipeline Stages						
n	ALU/branch	IF	ID	EX	MEM	WB		
n + 4	Load/store	IF	ID	EX	MEM	WB		
n + 8	ALU/branch		IF	ID	EX	MEM	WB	
n + 12	Load/store		IF	ID	EX	MEM	WB	
n + 16	ALU/branch			IF	ID	EX	MEM	WB
n + 20	Load/store			IF	ID	EX	MEM	WB

Scheduling Static Multiple Issue

- Compiler must remove some/all hazards
  - Reorder instructions into issue packets
  - No dependencies with a packet
  - Possibly some dependencies between packets
    - Varies between ISAs; compiler must know!
  - Pad with nop if necessary



64 → instruction 32 bits

RISC-V with Static Dual Issue

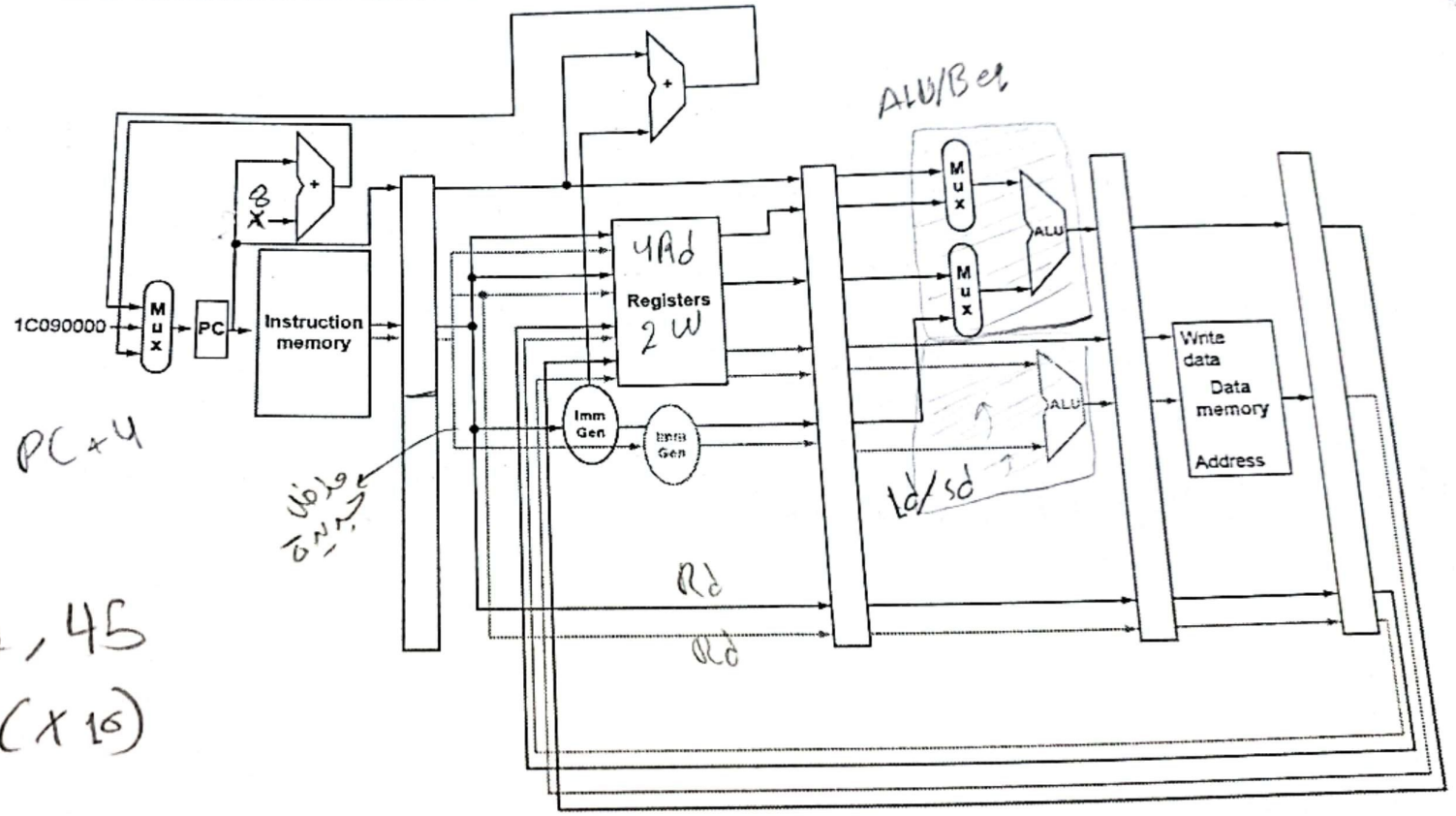
- Two-issue packets
  - One ALU/branch instruction
  - One load/store instruction
  - 64-bit aligned
    - ALU/branch, then load/store
    - Pad an unused instruction with nop

PC ← PC+4
PC ← PC+8
2 instructions with 4 bits for each!!
64/8 = 8 bits

Address	Instruction type	Pipeline Stages						
n	ALU/branch	IF	ID	EX	MEM	WB		
n + 4	Load/store	IF	ID	EX	MEM	WB		
n + 8	ALU/branch		IF	ID	EX	MEM	WB	
n + 12	Load/store		IF	ID	EX	MEM	WB	
n + 16	ALU/branch			IF	ID	EX	MEM	WB
n + 20	Load/store			IF	ID	EX	MEM	WB

→ (7) cycles

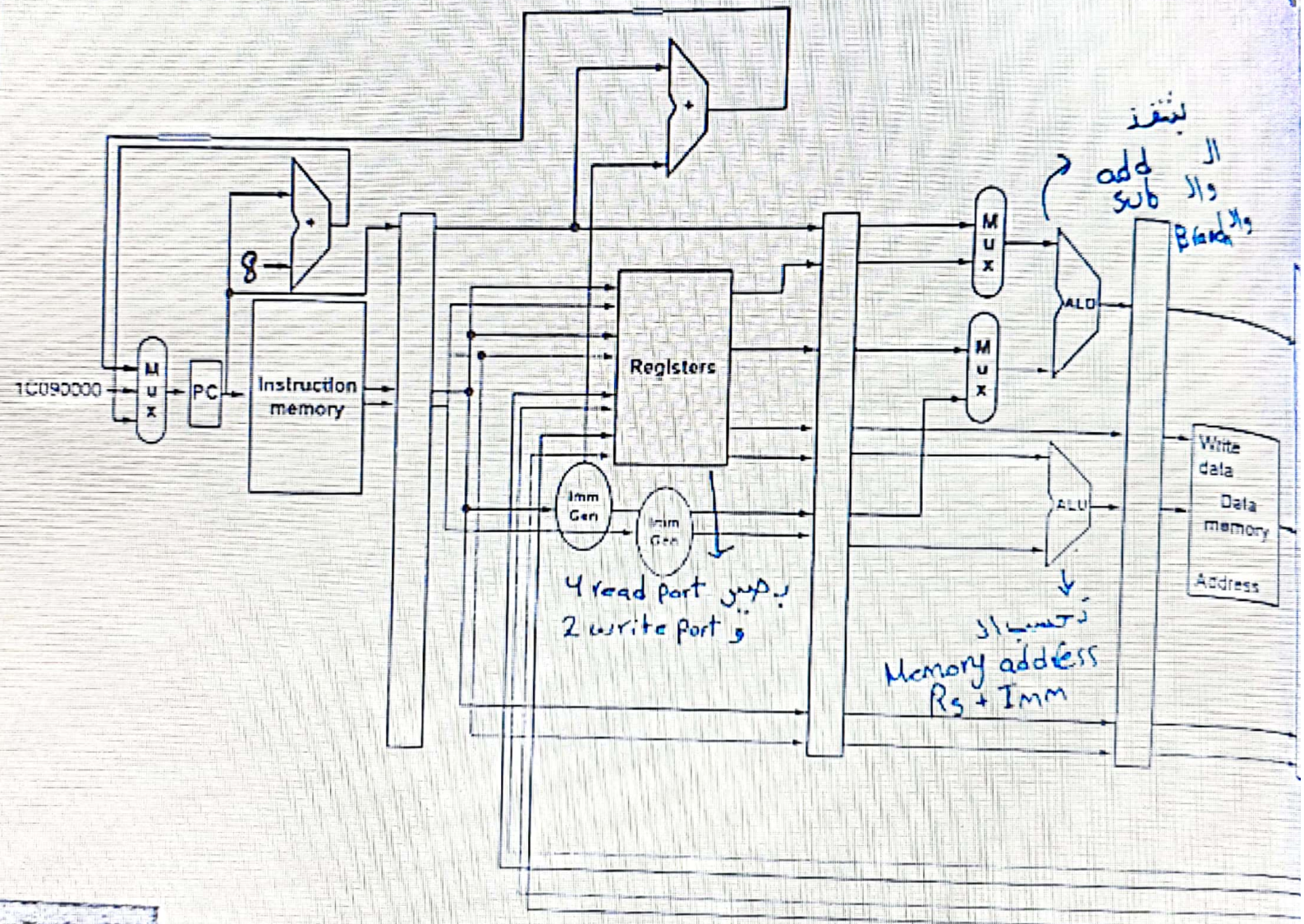
RISC-V with Static Dual Issue



add x1, x2, 45

ld x5, 8(x16)

RISC-V with Static Dual Issue



Hazards in the Dual-Issue RISC-

More instructions executing in parallel

EX data hazard → بين الـ issue و الـ issue اللي بعدجه ممكن يكون في dependent وتحتاج Forwarding

- Forwarding avoided stalls with single-issue
- Now can't use ALU result in load/store in same pack

← مابنقدر نحطهم مع بعض لأنه بيعتمدوا على بعض

```
add  x10, x0, x1
ld   x2, 0(x10)
```

Split into two packets, effectively a stall

Load-use hazard

- Still one cycle use latency, but now two instructions

More aggressive scheduling required

Hazards in the Dual-Issue RISC-V

2 pack dependen

	1	2	3	4	5	6	7	8	9	10
add x10, x0, x1	F	D	E* result	M	W					
ld x2, 0(x10)		F	D	E						

بنحط الـ load بـ packet تانية عشان يكون الفيتشش parallel

*ووقعنا ما عندي forwarding

Hazards in the Dual-Issue RISC-V

	1	2	3	4	5	6	7	8	9	10
ld x1, 0(x2)	F	D	E	M	W					
sub x4,x1,x5			F	D	E	.	.			

Forwarding in Dual-Issue RISC-V

From W in memory pipeline to E in ALU pipeline

	1	2	3	4	5	6	7	8	9	10
ld x31, 0(x20)	F	D	E	M* → W						
sub x31,x31,x21			F	D	E	M	W			

Forwarding in Dual-Issue RISC-V

From M in ALU pipeline to M in memory pipeline

	1	2	3	4	5	6	7	8	9	10
add x31, x31, x21	F	D	E	M	W					
sd x31, 0(x20)	F	D	E	M	W					

Scheduling Example

loop unrolling

* حطيناهلن IPC = 1.25 عشان الـ nop instructions ← nop فلو حطينا بدل الـ nop بزيد الـ IPC فلذلك بنستخدم حل أحسن والي هو

Schedule this for dual-issue RISC-V

```
Loop: ld   x31,0(x20)    // x31=array element
      add  x31,x31,x21   // add scalar in x21
      sd   x31,0(x20)    // store result
      addi x20,x20,-8    // decrement pointer
      blt  x22,x20,Loop  // branch if x22 < x20
```

2 issue Packets

ممكن نبدأ بـ 1 برضه ← حطيناها أول وحدة لأنها أول عملية

	ALU/branch	Load/store	cycle
Loop:	nop	ld x31,0(x20)	1
	addi x20,x20,-8	nop	2
	add x31,x31,x21	nop	3
	blt x22,x20,Loop	sd x31,8(x20)	4

- IPC = 5/4 = 1.25 (c.f. peak IPC = 2)

لـ instruction في issue Packet

الـ addi قبلها صار فلازم نخليها 8 مو 0 عشان نعكس أثر الـ addi

الإرغابيتجي بعد الـ add بس ممكن أحطها برضه

Scheduling Example

- Schedule this for dual-issue RISC-V

```
Loop: ld   x31,0(x20)     // x31=array element
      add  x31,x31,x21    // add scalar in x21
      sd   x31,0(x20)     // store result
      addi x20,x20,-8     // decrement pointer
      blt  x22,x20,Loop   // branch if x22 < x20
```

F D E M W
F D E M W
F D E M W

	ALU/branch	Load/store	cycle
Loop:	nop	ld x31,0(x20)	1
	addi x20,x20,-8	nop	2
	add x31,x31,x21	nop	3
	blt x22,x20,Loop	sd x31,8(x20)	4

- IPC = 5/4 = 1.25 (c.f. peak IPC = 2)

Loop Unrolling

IPC instructions عدد الـ

Replicate loop body to expose more parallelism

- Reduces loop-control overhead فوائده

Use different registers per replication

- Called "register renaming" → بخلينا نتغلب على أنواع dependence الخافية من الـ
- Avoid loop-carried "anti-dependencies"

  Store followed by a load of the same register

  Aka "name dependence", write-after-read

  Or "output dependence", write-after-write

  - Reuse of a register name

MK

Unrolling Steps

1. Replicate the loop instructions n times

مثال:

```
if (---; i+=1)
    a[i] = a[i]+3;
```

```
if (---; i+=2) {
    a[i] = a[i]+3;
    a[i+1] = a[i+1]+3;
}
```

Chapter 4 — The Processo

	ALU/branch	Load/store	cycle
Loop:	addi x20,x20,-32	ld x28, 0(x20)	1
	nop	ld x29, 24(x20)	2
	add x28,x28,x21	ld x30, 16(x20)	3
	add x29,x29,x21	ld x31, 8(x20)	4
	add x30,x30,x21	sd x28, 32(x20)	5
	add x31,x31,x21	sd x29, 24(x20)	6
	nop	sd x30, 16(x20)	7
	blt x22,x20,Loop	sd x31, 8(x20)	8

2 general 32 Floating ← register بس عادي لإنه عنا كثير

IPC = 14/8 = 1.75

ما وصلنا 2 كشان لسا في nop

بس هاي طريقة أحسن

بإنه قبل كنا نستخدم بس x31, x20, .. لبس هلا صرنا نستخدم x23, x24, x25 بالإضافة للسابق

Closer to 2, but at cost of registers and code size

فلو يعني الـ Performence ما رح يعني لو حجم البرنامج كبر شوي

هو بحسن الـ Performence بس على حساب الـ registers والـ code size

يعني استهلكنا Memory usage ولكن هاي المشاكل مقبولة والـ Mem هاي الأيام رخيصة وكبيرة

كبر حجمه أكثر من 3 أضعاف ← 32 bit

* الـ code الجديد : 8 x 2 x 4 = 64 bytes

issue Packets ← instruction per issue packets

* الـ code اللي قبل : 5 x 4 = 20 byte ← 32 bit

instructions

Loop:

```
ld   X31, 0 (X20)
add  X31, X31, X21
sd   X31, 0(X20)
addi X20, X20, -8
blt  X22, X20, loop
```

① Replicate the loop instructions n times → 4

Loop:

```
ld   X31, 0(X20)
add  X31, X31, X21
sd   X31, 0 (X20)
addi X20, X20, -8
blt  X22, X20, loop
```

```
ld   X31, 0 (X20)
add  X31, X31, X21
sd   X31, 0 (X20)
addi X20, X20, -8
blt  X22, X20, loop
```

```
ld   X31, 0(X20)
add  X31, X31, X21
sd   X31, 0 (X20)
addi X20, X20, -8
blt  X22, X20, loop
```

```
ld   X31, 0 (X20)
add  X31, X31, X21
sd   X31, 0 (X20)
addi X20, X20, -8
blt  X22, X20, loop
```

② Remove unneeded loop overhead

```
Loop:
    ld   X31, 0(X20)
    add  X31, X31, X21
    sd   X31, 0(X20)

    ld   X31, -8(X20)
    add  X31, X31, X21
    sd   X31, -8(X20)

    ld   X31, -16(X20)
    add  X31, X31, X21
    sd   X31, -16(X20)

    ld   X31, -24(X20)
    add  X31, X31, X21
    sd   X31, -24(X20)
    addi X20, X20, -32
    bit  X22, X20, Loop
```

③ Modify instructions

```
Loop:
    ld   X31, 0(X20)
    add  X31, X31, X21
    sd   X31, 0(X20)

    ld   X31, -16(X20)
    add  X31, X31, X21
    sd   X31, -16(X20)
```

③ Modify instructions

```
Loop:
    ld   X31, 0(X20)
    add  X31, X31, X21
    sd   X31, 0(X20)

    ld   X31, -8(X20)
    add  X31, X31, X21
    sd   X31, -8(X20)

    ld   X31, -16(X20)
    add  X31, X31, X21
    sd   X31, -16(X20)

    ld   X31, -24(X20)
    add  X31, X31, X21
    sd   X31, -24(X20)
    addi X20, X20, -32
    blt  X22, X20, loop
```

4. Rename registers

```
Loop:
    ld    x28,0(x20)
    add   x28,x28,x21
    sd    x28,0(x20)

    ld    x29,-8(x20)
    add   x29,x29,x21
    sd    x29,-8(x20)

    ld    x30,-16(x20)
    add   x30,x30,x21
    sd    x30,-16(x20)

    ld    x31,-24(x20)
    add   x31,x31,x21
    sd    x31,-24(x20)
    addi  x20,x20,-32
    blt   x22,x20,Loop
```

Loop Unrolling Example

	ALU/branch	Load/store	cycle
Loop:	addi x20,x20,-32	ld x28, 0(x20)	1
	nop	ld x29, 24(x20)	2
	add x28,x28,x21	ld x30, 16(x20)	3
	add x29,x29,x21	ld x31, 8(x20)	4
	add x30,x30,x21	sd x28, 32(x20) 0+32	5
	add x31,x31,x21	sd x29, 24(x20) -8+32	6
	nop	sd x30, 16(x20) -16+32	7
	blt x22,x20,Loop	sd x31, 8(x20) -24+32	8

بزبط تكون بـ cycle

→ More fast but extra cost because we used more registers

- IPC = 14/8 = 1.75
  - Closer to 2, but at cost of registers and code size

Org. dis - x = -32

x = org. dist 32

x = -24 + 32 = 8 !!

Dynamic Multiple Issue

أحدث وأقوى وبيحرقوا كهربا أكتر

"Superscalar" processors → dynamic Multiple issue الاسم الثاني لل

CPU decides whether to issue 0, 1, 2, ... each cycle

مثلاً ال ALU ما بيبعتلوا 2 instruction مع بعض

- Avoiding structural and data hazards

Avoids the need for compiler scheduling

- Though it may still help

↳ ما بنحتاجه ولكن لو عملنا مثلاً loop unrolling بيحسن أكتر

- Code semantics ensured by the CPU

↓

* حجة ال Code مسؤولية ال CPU مش مسؤولية ال Compiler

MK

Dynamic Pipeline Scheduling

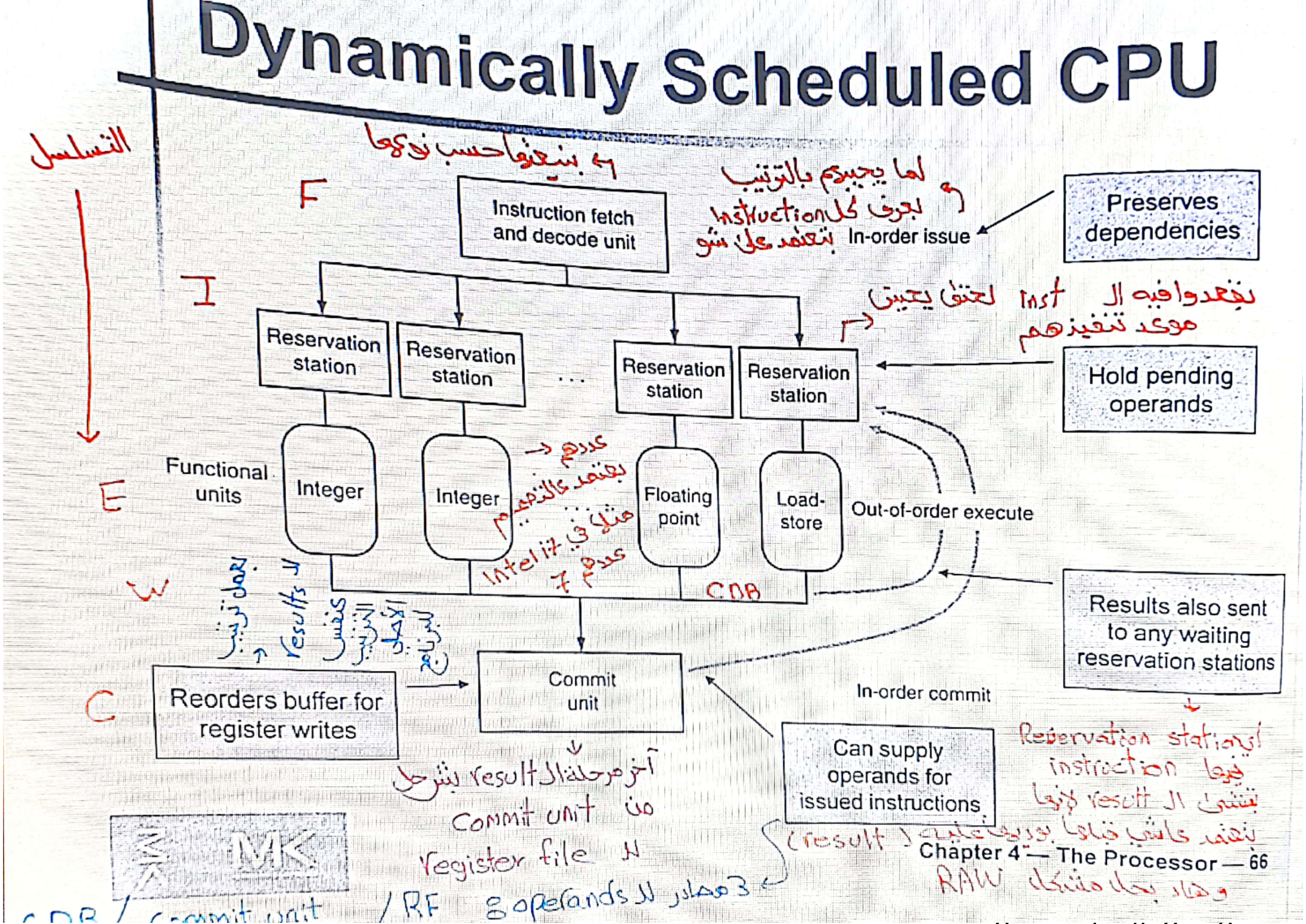
- Allow the CPU to execute instructions out of order to avoid stalls
  - But commit result to registers in order

Example

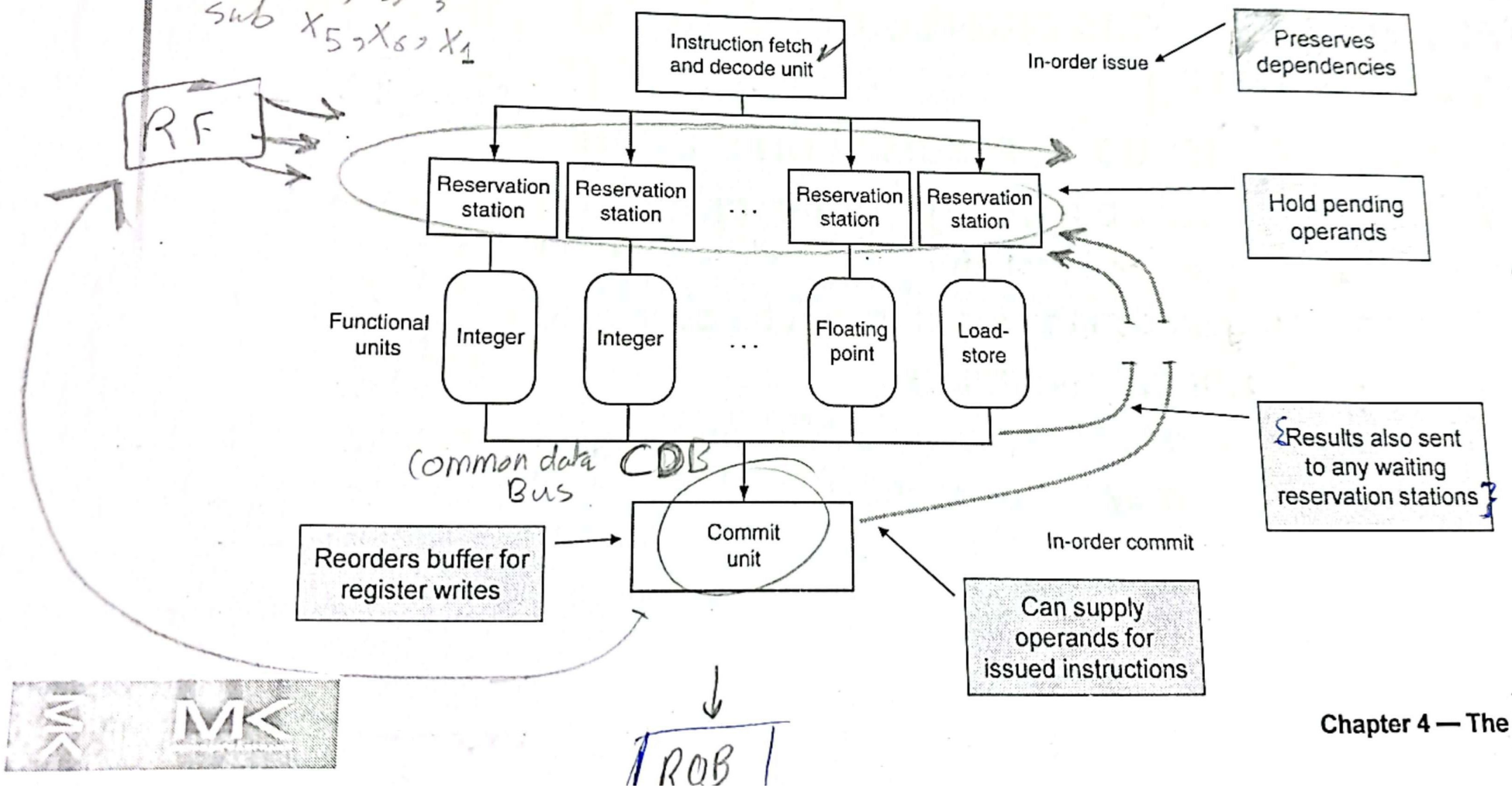
```
ld    x31,20(x21)
add   x1,x31,x2
sub   x23,x23,x3
andi  x5,x23,20
```

- Can start sub while add is waiting for ld

Dynamically Scheduled CPU

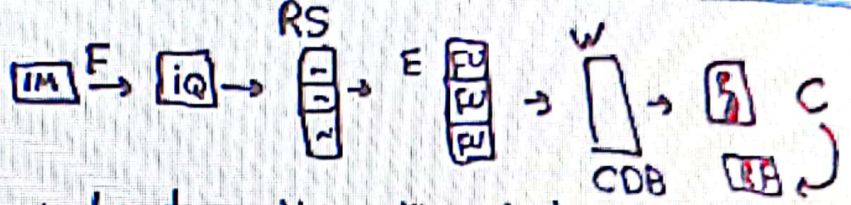


Dynamically Scheduled CPU



execution out of orde

Pipeline Stages



F: Fetch from instr. memory (IM) to instr. queue (IQ).

I: Issue from IQ to reservation stations (RS), reading ready operands from register file (RF).

E: Execute when functional unit (FU) is free and instr. In RS has ready operands. → متغيراتها جاهزين

W: Write result from FU through common data bus (CDB) to reorder buffer (ROB) and RS.

C: Commit results in order from ROB to RF and memory → Store بال بنكتب فوقها

Loads have **FIAMWC**, stores have **FIAC**. **A**: Address calculation = بدال ما نسميها Address Calculation

بال C بنكتب بال Memory

Pipeline Stages

F: Fetch from instr. memory (IM) to instr. queue (IQ).

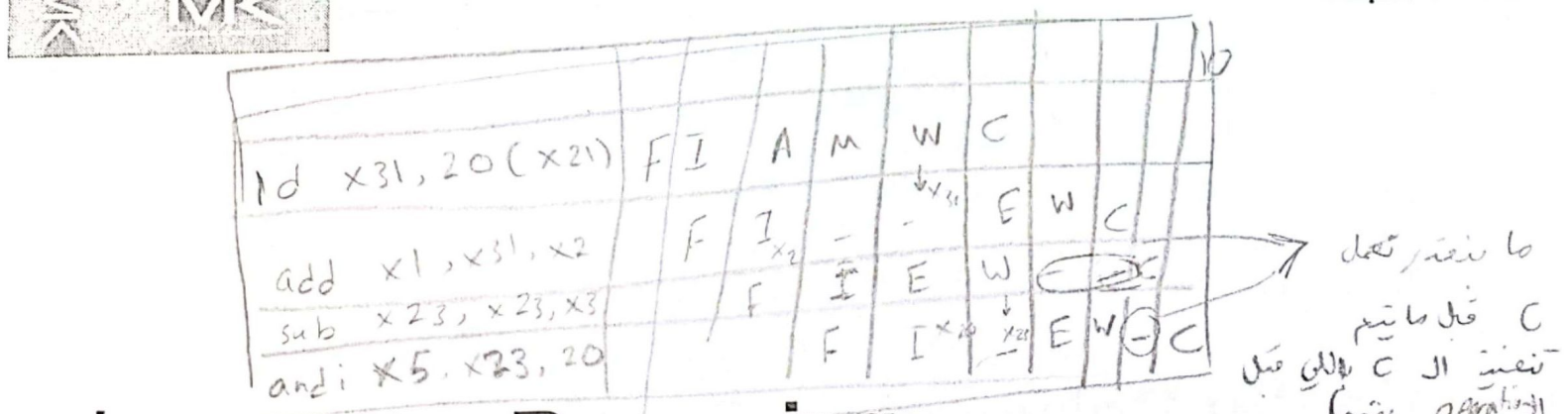
I: Issue from IQ to reservation stations (RS), reading ready operands from register file (RF).

E: Execute when functional unit (FU) is free and instr. in RS has ready operands.

W: Write result from FU through common data bus (CDB) to reorder buffer (ROB) and RS.

C: Commit results in order from ROB to RF and memory.

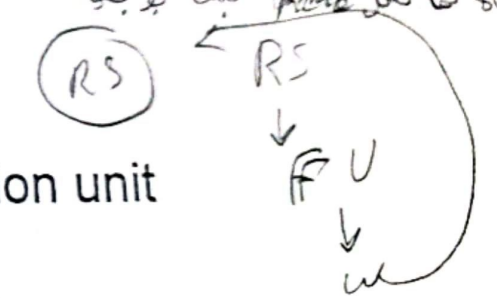
Loads have **FIAMWC**, stores have **FIAC**. **A**: Address calculation



Register Renaming

Register Renaming

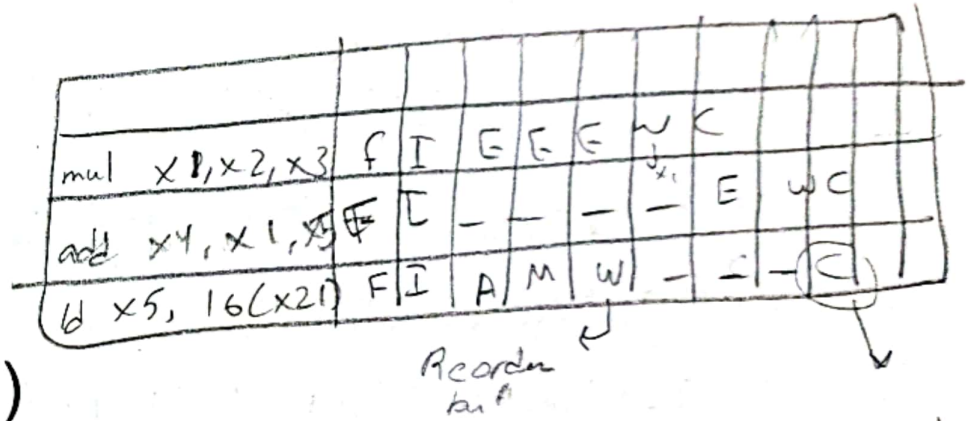
- Reservation stations and reorder buffer effectively provide register renaming
- On instruction issue to reservation station
  - If operand is available in register file or reorder buffer
    - Copied to reservation station
    - No longer required in the register; can be overwritten
  - If operand is not yet available
    - It will be provided to the reservation station by a function unit
    - Register update may not be required



Examples

- Assume superscalar processor of degree 3
- Name dependence (WAR)

```
mul   x1,x2,x3
add   x4,x1,x5
ld    x5,16(x21)
```



- Output dependence (WAW)

```
mul   x1,x2,x3
add   x4,x1,x5
ld    x1,16(x21)
```

Speculation →

dynamic Exec.. مبني على ال Speculation وبزيد عليه ال

- "Guess" what to do with an instruction
  - Start operation as soon as possible
  - Check whether guess was right
    - If so, complete the operation
    - If not, roll-back and do the right thing
- Common to static and dynamic multiple issue
- Examples
  - Speculate on branch outcome
    - Roll back if path taken is different
  - Speculate on load
    - Roll back if location is updated

جزا مفيد لانه التنبؤ حاليا بكون صحيح بنسبة 95% تقريبا

Branch Speculation

Predict branch and continue issuing

- Don't commit until branch outcome determined → ما بنعمل commit الا بس

عشان نتأكد من التنبؤ Branch لا نعمل

Example: Assume a superscalar processor of degree 2 and the branch prediction is not taken.

2 instructions بقدر يجيب Every cycle

```
ld    x1,0(x20)
beq   x1,x2,Skip
I3
I4
```

stored values to load unit

Don't commit load until speculation cleared

Example: Superscalar of degree 3.

```
ld   x1,0(x20)  ]
sd   x2,0(x1)   ] RAW
ld   x3,0(x21)  →
```

independent ما بعتمدوا كبتر عالأخلب

F I A M W C
F I A C
F I ما بتعتمد عاللي فوق. A M W C

في حالات نادرة بكونوا dependent لو مثلاً ال Address اللي حطيناه بـ X1 بالصدفة كان يساوي ال Address اللي موجود بـ X21 وهاي حالة نادرة

لو بدون Speculation حنضطر نتأخر لنعرف لا Speculation ولا لا

Speculation and Exceptions

What if exception occurs on a speculatively executed instruction?

- e.g., speculative load before null-pointer check

Static speculation

- Can add ISA support for deferring exceptions

Why Do Dynamic Scheduling?

Why not just let the compiler schedule code?

Not all stalls are predicable

- e.g., cache misses

Can't always schedule around branches

- Branch outcome is dynamically determined

Different implementations of an ISA have different latencies and hazards

much more complex than single pipeline

Complexity of dynamic scheduling and speculations requires power

more transistors
more circuites

Multiple simpler cores may be better

ما كان بحاجة نحط هي وحدة
تغير خلال ٤٠ سنة تقريبا

مختلف عن اللي درسناه لانفذ ب Simple pipeline زي اللي درسناه

Microprocessor	Year	Clock Rate	Pipeline Stages	Issue Width	Out-of-Order/ Speculation	Cores/ Chip	P[illegible]
Intel 486 (CISC)	1989	25 MHz	5	1	No	1	5
Intel Pentium	1993	66 MHz	5	2	No	1	[illegible]
Intel Pentium Pro	1997	200 MHz	10	3	Yes	1	[illegible]
Intel Pentium 4 Willamette	2001	2000 MHz	22	3	Yes	1	[illegible]
Intel Pentium 4 Prescott	2004	3600 MHz	31	3	Yes	1	[illegible]
Intel Core	2006	2930 MHz	14	4	Yes	2	[illegible]
Intel Core i5 Nehalem	2010	3300 MHz	14	4	Yes	2-4	[illegible]
Intel Core i5 Ivy Bridge	2012	3400 MHz	14	4	Yes	8	[illegible]

لـ لما الـ Pipeline stages بنزيد الأشي اللي بدي اعمله ب stage وحدة بجزئه على اكثر والأزمة ب stage وحدة بتصير أصغر وبالتالي بتصير تشتغل على shorter clock period يعني أداء performance

ولهذا لما تزيد الـ Power بصير كثير بس سيطروا عليه آخر السنوات اقلوا عدد الـ Pipeline stages

Example 1: Assume SuperScalar processor of degree 3 with 2 address calculations units.

Predict branch as not taken, but resolve to taken. The ld has exception in M.

هون بنكتشف إنها Taken

	1	2	3	4	5	6	7	8	9	10
beq X1, X2, L1	F	I	E	w	C					
ld X5, 16(X21)	F	I	A	M	n					

جبناها بناء على الافتراض تاكنا إنها beq ↳

Not taken ←

هون اكتشفنا إنه فيه Exception ↳

معناها لازم نروح لـ EHS ← Exception Handling Subroutine

بس بالتصاميم الحديثة إذا فيه Exception احنا ما بنروح إلا بال Commit stage بس كمان

احنا بننفذ الـ ld على الفاضي لإنها beq ← taken فبسياطة بنعملها Null

وال Exception وكأنه صار

Example 2 : Assume superScalar Processor of degree 3 with 2 address calculation units.

Assume First sd has exception in C.

	1	2	3	4	5	6	7	8	9	10	
ld X1, 0(X20)	F	I	A	M	W	C					عرفنا انه فيه exception فبشحول من C إلك null
sd X1, 0(X21)	F	I	A	-	-	n					
sd X2, 16(X21)	F	I	-	A	-	n					
EHS :							F				

وبروح لـ EHS وبس أخلص والأمور تكون ملائمة برجع بعمل F لـ instruction اللي عملتلوم الـ null وبرجع أنفذهم مرة ثانية

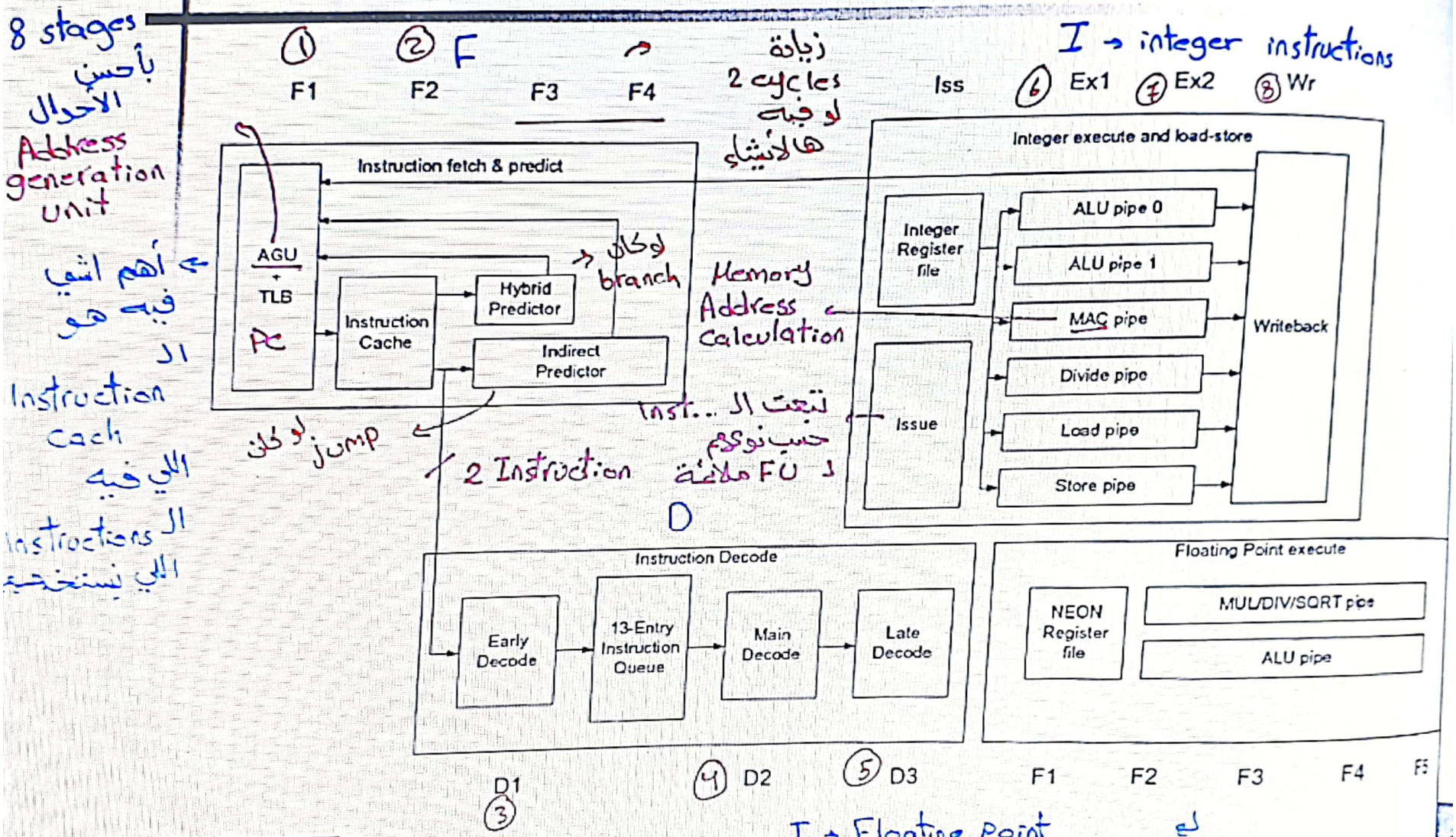
Cortex A53 and Intel i7

Processor	ARM A53	Intel Core i7 920
Market	Personal Mobile Device	Server, cloud
Thermal design power	100 milliWatts (1 core @ 1 GHz)	130 Watts
Clock rate	1.5 GHz	2.66 GHz
Cores/Chip	4 (configurable)	4
Floating point?	Yes	Yes
Multiple issue?	Dynamic	Dynamic
Peak instructions/clock cycle	2	4
Pipeline stages	8	14
Pipeline schedule	Static in-order	Dynamic out-of-order with speculation
Branch prediction	Hybrid	2-level
1st level caches/core	16-64 KiB I, 16-64 KiB D	32 KiB I, 32 KiB D
2nd level caches/core	128-2048 KiB	256 KiB (per core)
3rd level caches (shared)	(platform dependent)	2-8 MB

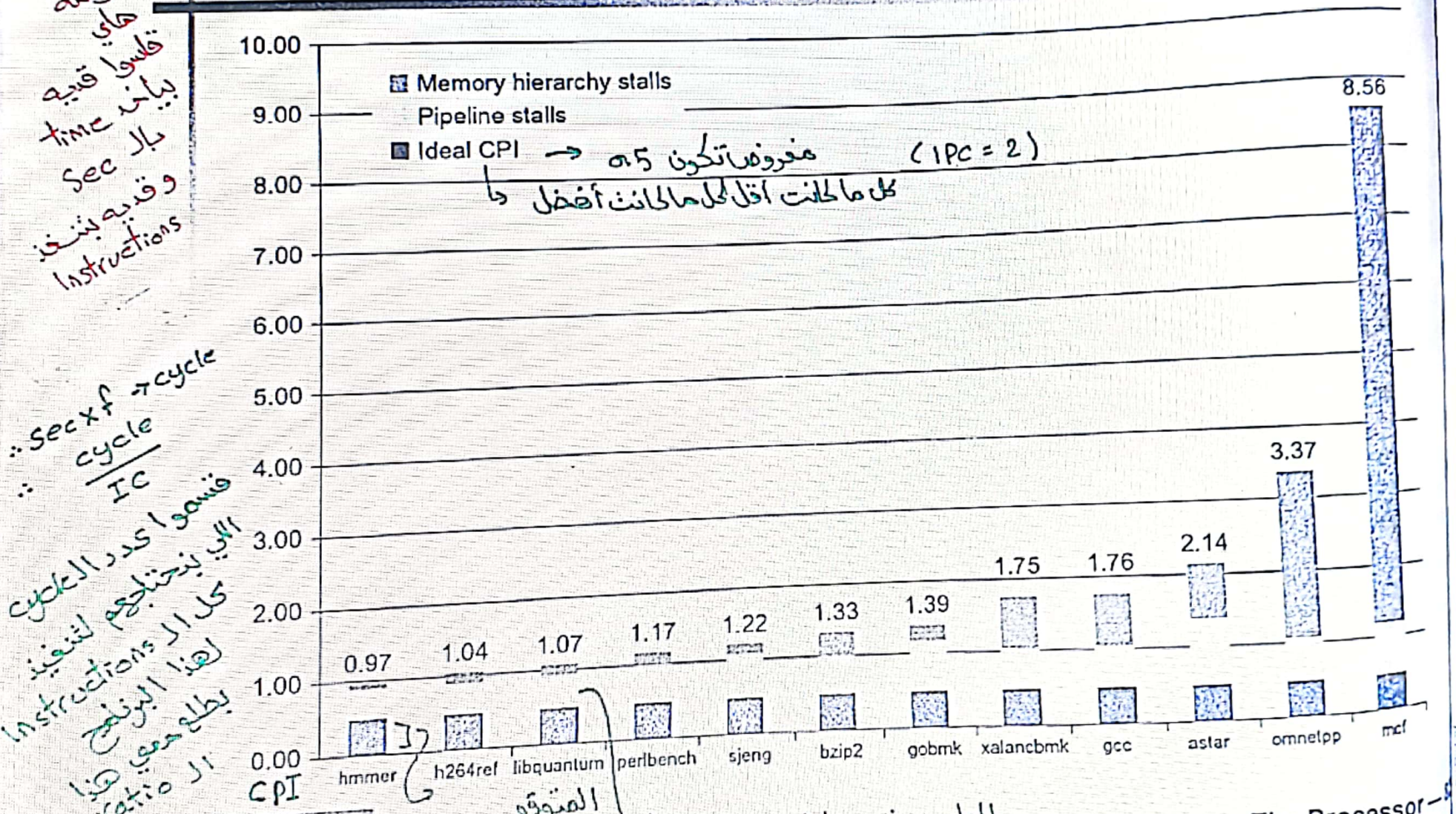
microprocessors هي سلسلة ناجحة جدا من ال

ARM Cortex-A53 Pipeline

في 8 stages بأحسن الأحوال



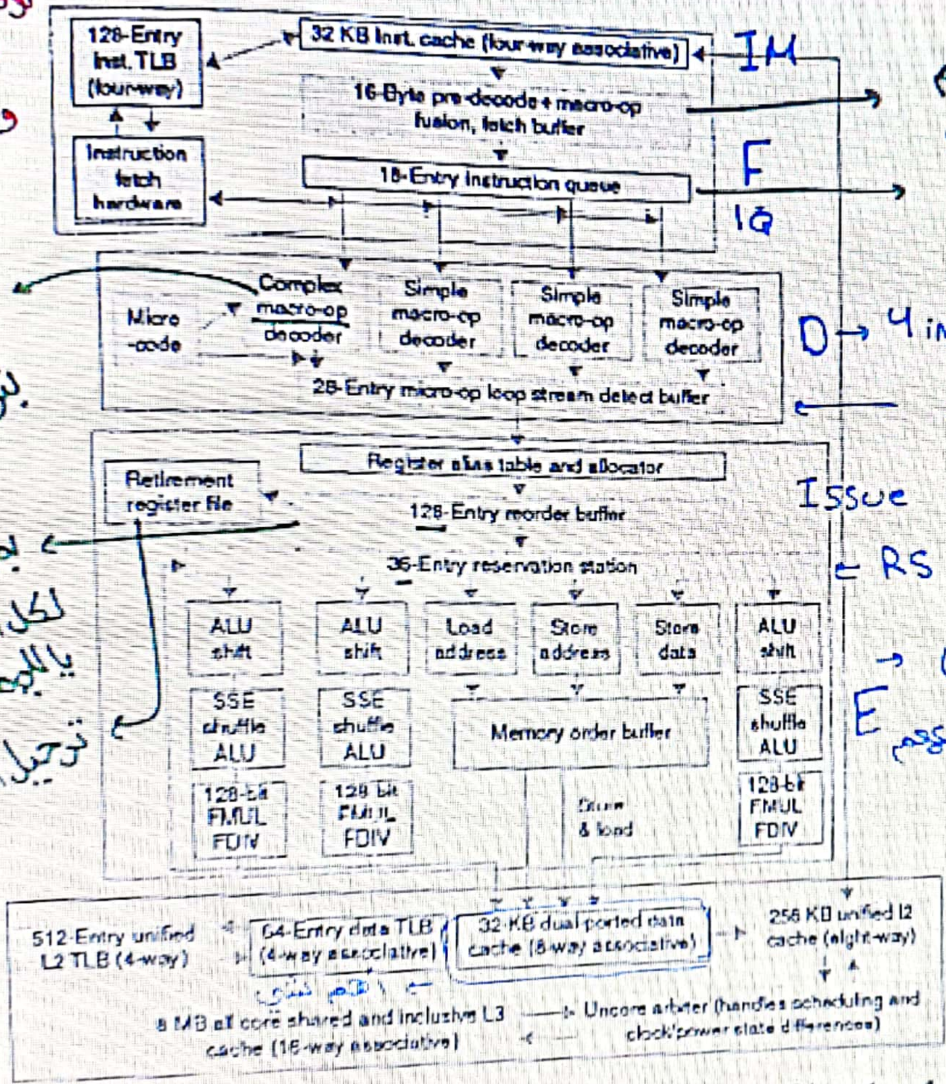
ARM Cortex-A53 Performance



Core i7 Pipeline

→ Processor much more complex
Superscalar Processor with degree 4

كمان تغير من واسن لهلا اللي بختلف من وقت لوقت عدد واحجام ال RS و ROB و IQ وحتى عدد ال FU



IM

F → ما بنعرف كم Instructions فيهم

IQ → بسع 18 مجموعة → 18 * 16 byte

D → 4 instructions ال

decode instructions ال بينحطوا فيه

complex instructions بترجمهم لل simple

Issue

I

RS

بمرحلة ال Issue لكل instruction بنحجز بال ROB يالمحصوص

بصير ال commit ترجيل النتائج اله حسب ترتيبها الأصلي

E → تنفيذ ال Instructions بال (FU) حسب نوعهم Functional unit

→ بنحروا ينعملوا لل Floating Point

M

→ عشان ندخل كم بالوقت الحلقي

المفر ثاني →

Core i7 Performance