Ch3:

① Multiplication:

← بنعرف انا في overflow او لا من ال most sig تاع ال product (ازا ساوى ≠ 0)

ازا traditional بكون فيها كل الخطوات (n).

ازا optimized (n) ← ① ال ALU صارت 64Bit (وقت للاقل)

② بدل ما نخزن ال multiplier ب register صار يخزن بال least sig تاعت ال product

③ ال multiplicand بدل ما shift لقدا + وقت كمان ال 64

④ ال most sig بال product همه الي بجمعهم مع ال multiplicand ويكونوا الجواب: حتى اشي يصيروا عالشمال.

* في حال كانوا signed:

① يا بتعاملهم كانهم unsigned والجواب الاخير بحطله الاشارة لو مختلفين

② الطريقة الثانية:

① sign-Extend the partial products

② subtract the last partial if multiplier is -ve

③ ignore carry

* instructions: mul (lower), mulh, mulhu (upper), mulhsu

② Division:

or Q / end / rem

* بال Trad: (n+1) ال divisor بنحط عالطرف الشمال (في البداية)
+ ال dividend في ال remainder (في البداية يمينه)
+ الطول 2 يكون كل ال rem ناقص ال divisor

* بال opti: (n) ال Q صار جوا ال rem

rem ← 5	4 → Q

* instructions: div, rem, divu, remu

③ Floating Points:

single → Float
double → double

① From Binary to decimal:

$$X = (-1)^s * (1 + Fraction) * 2^{ex - bias}$$

bias → 127 (single), 1023 (double)

② From decimal to binary:

S	ex + bias (8 / 11)	Fraction (23 / 52)

* For Addition: ① Allign (بكبر الصغير) ② Add ③ Normalize + check For ↑↓Flow ④ round + renormalize

For multiplication: ① جمع الاسس ② ضرب ③ norm + check ④ round ⑤ الاشارة

* instructions: Flw, Fld, Fsw, Fsd → base → X
Fadd.s, Fsub.s, Fmul.s, Fdiv.s, Fsqrt.s/.d
Feq.s, Flt.s, Fle.s/d → rd (X), rs1 (F), rs2 (F)

Ch4:

ALU control

مفهمه الكل

	ALUop input	operations output	Funct
R-type	10	0000	AND
R-type	10	0001	OR
R-type	10	0010	add
R-type	10	0110	sub
sub	01	0111	slt

ALUop add → 00

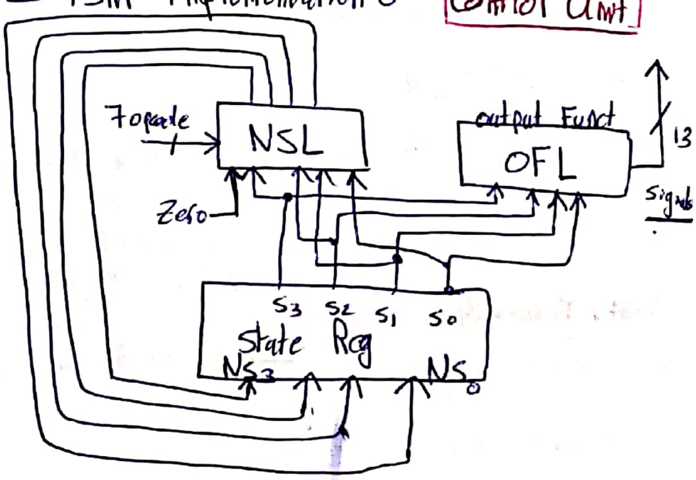
Control Unit:

For single + pipe (→ duplicate في)

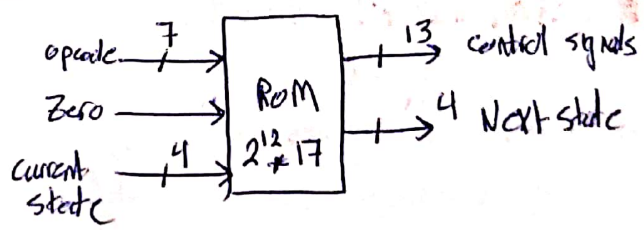
	R-F	ld	sd	beq	addi
ALUsrc	0	1	1	0	1
Memtoreg	0	1	X	X	0
Regwrite	1	1	0	0	1
MemRead	0	1	0	0	0
MemWrite	0	0	1	0	0
Branch	0	0	0	1	0
OP[1:0]	10	00	00	01	00

For multicycle:

1 FSM implementation: Control Unit



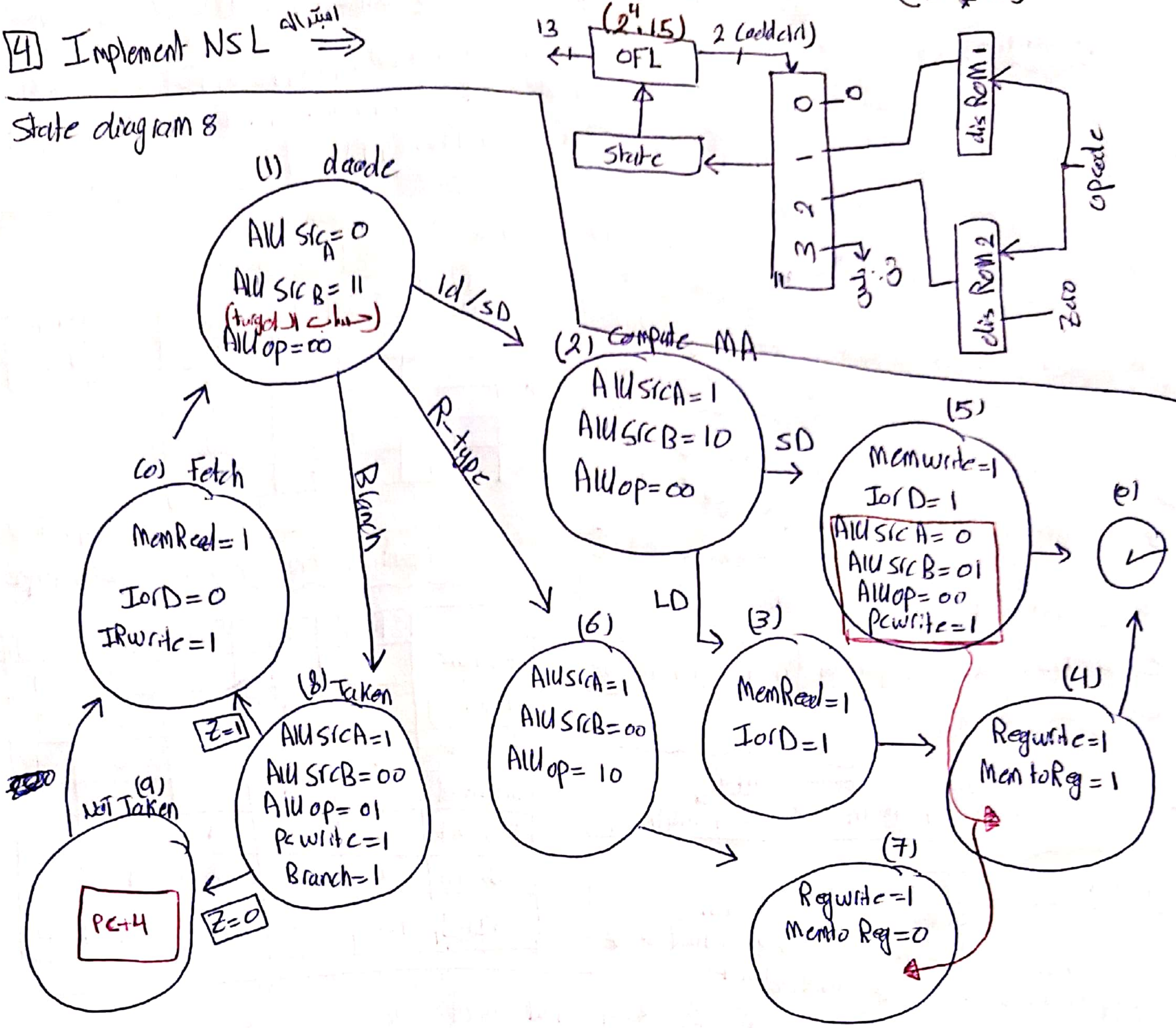
2 ROM implementation:



3 two RoMs : ① NSL RoM ($2^{12} * 4$) ② OFL RoM ($2^4 * 13$)

4 Implement NSL ⟹

State diagram :



* For Pipelining :

IF all stages are balanced + without Hazards

Time between ist (pipe) = $\frac{\text{Time between (not pipe)}}{\text{\# of stages}}$ (200 = 800 / 4)

~~Data Hazards~~

For forwarding : تنسيق ال sd فيها 152

① Mem → EX

② WB → EX → ld → R-type

control Hazard (Branch) :

1 Stall on Branch

2 predict all branches (not taken)

3 Static prediction → backward (taken) / forward (not taken)

4 dynamic "

* For performance :

CPI : For SC = 1 , For MC = القانون

CCT : SC = inst كل انقى بجمع بالبطيء

Memory + RF + AlU + Mem + RF : ld مثلا زي بال

MC = أبطئ unit فيهم (مثلا ال mem)

* For Forwarding conditions :

EX Hazard ≡ EX / Mem

Mem Hazard ≡ Mem / WB

Forward = 00 (عادي)

= 10 (Mem) ← اذا ال ٢ طلعوا ترو بختار الاقرب اللي هو ال mem

= 01 (WB)

Scanned by CamScanner

* For load-Use Data Hazard:

ex:

ld x2, ~, ~
and x4, x2, ~
or ~, x2, ~
add ~, x2, ~

* in c3 → stall = 1
* c4 → stall = 0
* c4 → bubbles في ال ex
* c5 → بتمشي ال M يعني bubble
* There is Forwarding in 7+5

c1	c2	c3	c4	c5	c6	c7	c8	c9
F	D	E	M	W				
	F	D	D	E	M	W		
		F	F	D	E	M	W	
				F	D	E	M	W

ex:

	c1	c2	c3	c4	c5	c6	c7
ld	F	D	E	M	W		
add		F	D	E	M	W	
sub			F	D	E	M	W

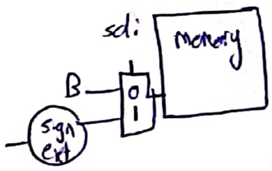
in c4: ID/EX.AluSrc = 0
EX/mem.memtoreg = 1

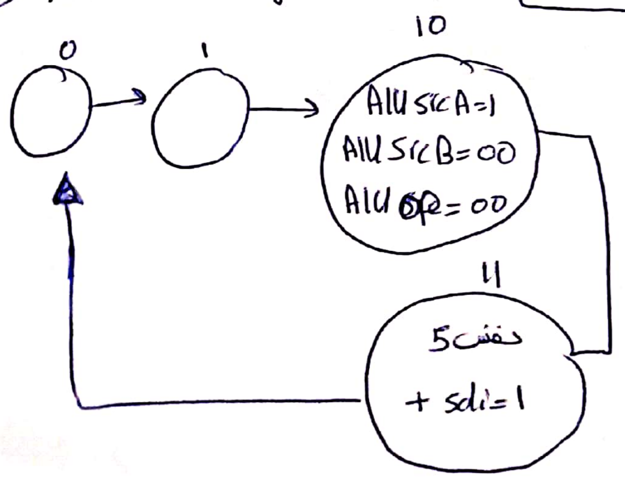
* For ld, R-type → ID/EX.Regwrite = 1
* For sd, beq → = 0

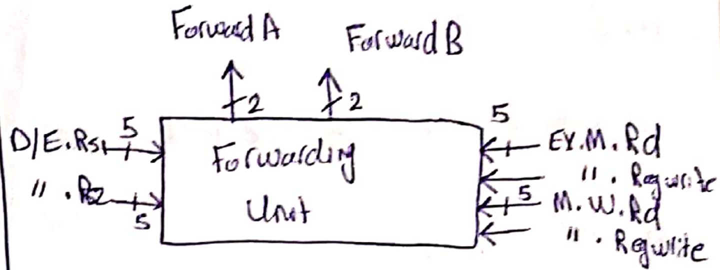
* ex: sdi rs1, rs2, imm (multi-cycle) → sign-ext(imm) → mem[rs1+rs2]

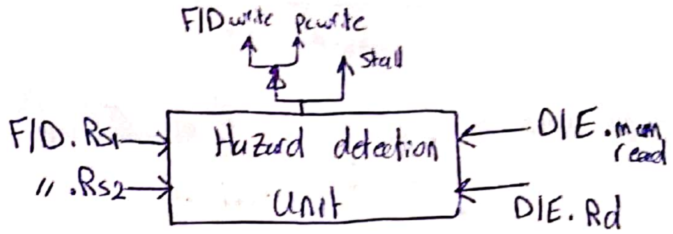
1. For the data path:
2. For FSM inp: (14) زيت عدد ال datapath ملعومات ال Table 2 تتغير كامل
3. single RoM ⇒ (32^{12} * 18)
4. OFL (size = 2^4 * 14)
5. For state diagram:

6. كمان dis RoM, 2 بتغير



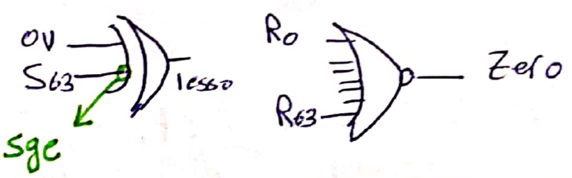






A.5:

slt:

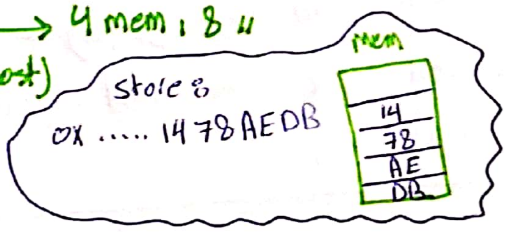


Ch2:

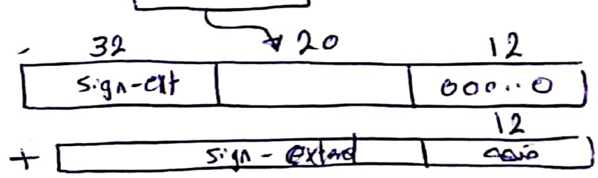
push → store, pop = load (sign-ext)

* lbu → 1 mem, 2 digit
* lhu → 2 mem, 4 "
* lwu → 4 mem, 8 "

(right most)
* sb
* sh
* sw



lui rd, 0x (5 digits)



Scanned by CamScanner

Ch 1 & 8

CPU time = CPU clock cycle * clock cycle time

$$\text{CPU time} = \frac{\text{CPU clock cycle}}{\text{clock rate}}$$

CPU time = IC * CPI * CCTime

↳ clock cycle

Avg &

Total clock cycle = $\sum_{i=1}^{n} (CPI_i * IC_i)$

$$CPI_{avg} = \frac{\text{Total CC}}{IC_{total}} = \sum_{i=1}^{n} \left(CPI_i * \frac{IC_i}{IC_{total}}\right)$$

---

* Same ISA + compiler ⇒ same IC_i and IC_{total}

* Same Hardware (CPU) ⇒ same clock cycle time and clock rate and CPI_i

---