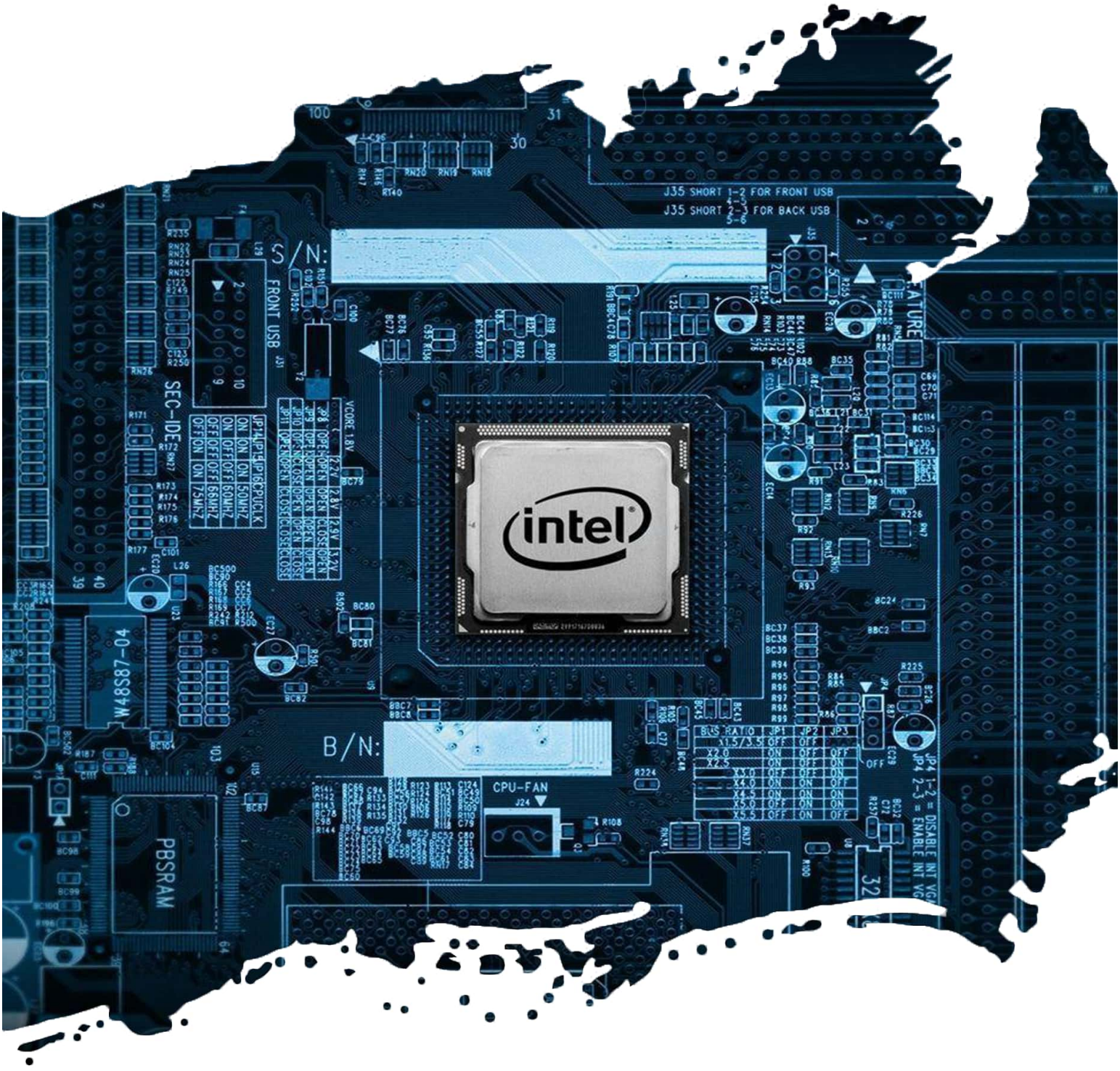
ORG

DR.WALEED DWEIK

KHALID ALNASER

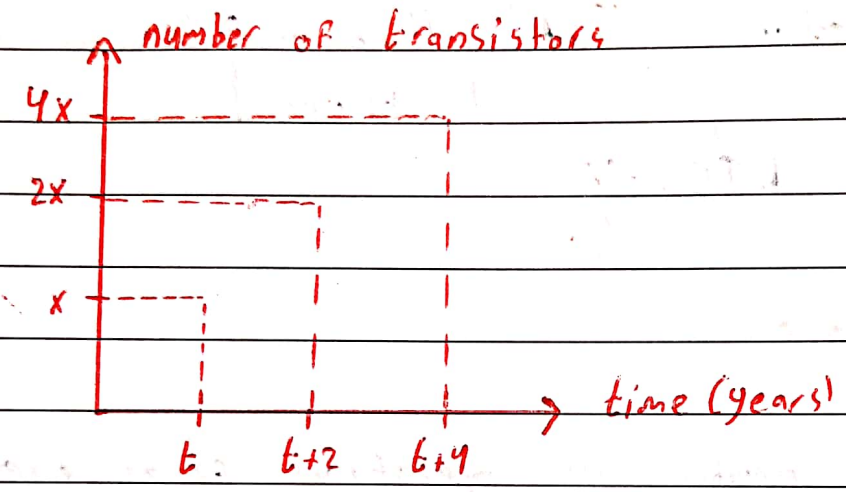


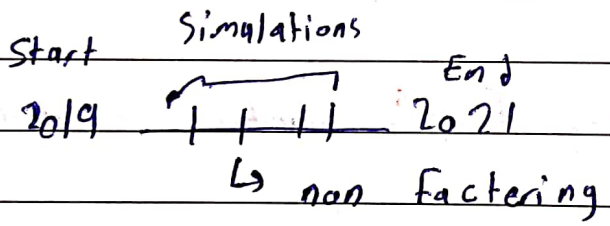
POWER UNIT

Eight great Ideas :

* number of transistors perchip

(A) Design for Moore's law





every 18-24 month.

top500.org Supercomputer

Classes of Coputers:

(A) Super Computer:

T Flop/sec

Tera Floating point operation

(B) Embedded Computers :

Power / Performance / cost
↓ ↑ ↓

* Instruction Set Architecture "ISA" { Hardware / Software } تربط بين

MIPS Intel x86 ARM RISC-V (5)

* Understanding Performance

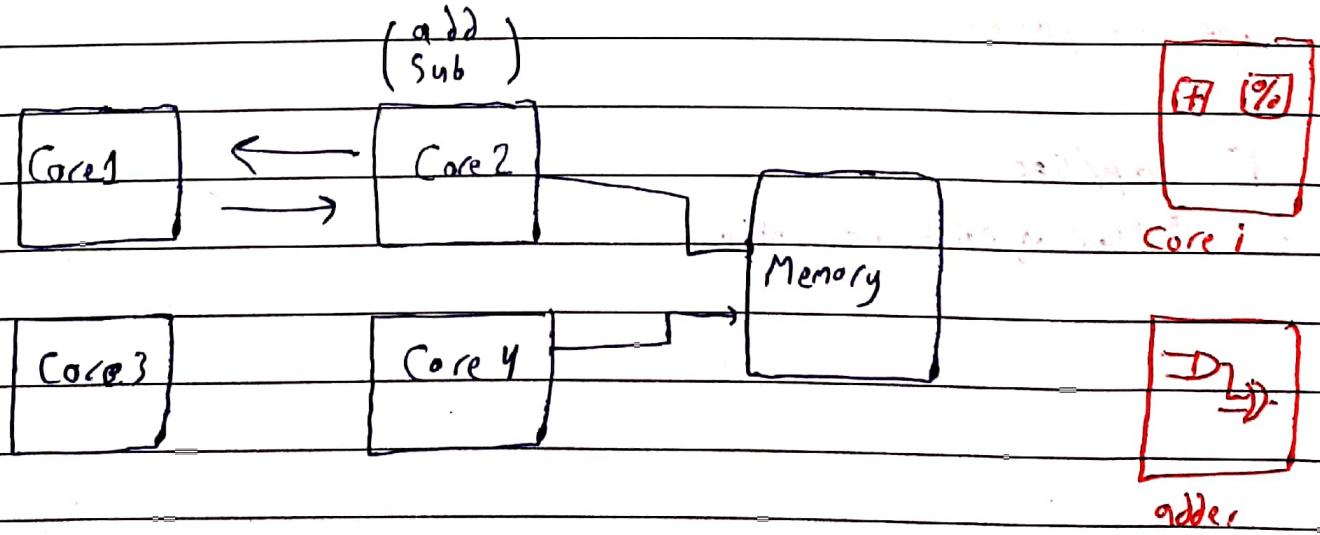
① Algorithm "Sorting" : number of operations to be executed

② Programming language, compiler, ISA : number of machine instructions to be executed

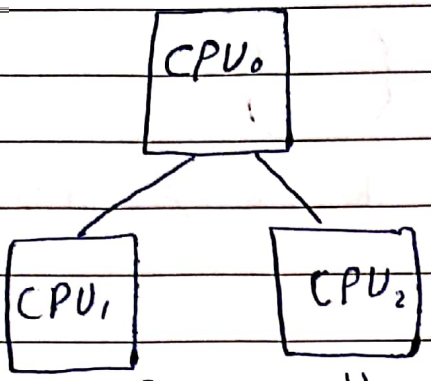
③ Processor and memory

④ I/O

(B) Use a Subtraction to Simplify design :



© Make the common case fast



: 90% addition
10% division

add (1ns)
divide (10 ns)

CPU1: add 0.2 ns, divide 10 ns
CPU2: add 1 ns, divide 5 ns

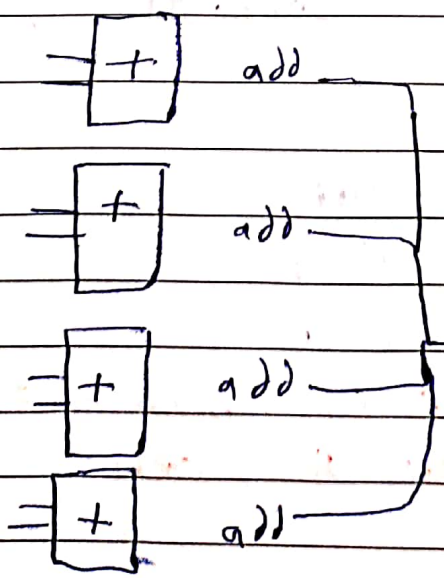
100 operation < 90 add, 10 divide

Time CPU0 = 90 x 1 + 10 x 10 = 190 ns
CPU1 = 90 x 1 + 10 x 5 = 140 ns
CPU2 = 90 x 0.2 + 10 x 10 = 118 ns

* الأفضل نرفع الـ Common Case كتير بطلع الـ time أحسن

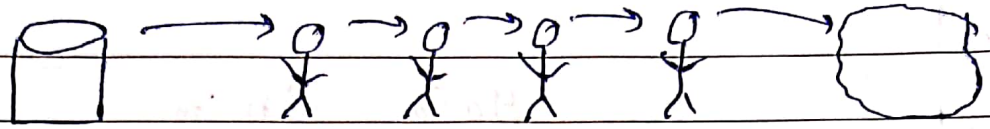
Slide 11

Ⓓ Performance via parallelism "بالتوازي"

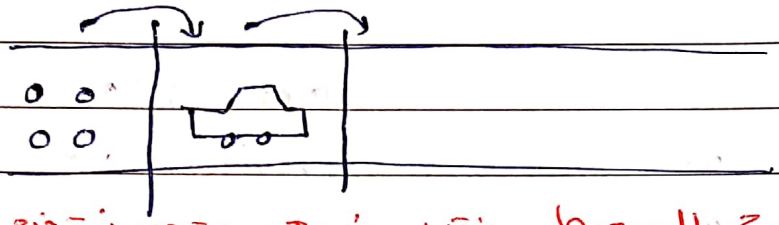


لما نحطهم مع بعض بالتوازي بقل الزمن

(E) Performance via Pipelining



هيك بتكون قسمناها لمراحل و هيك بنسرع مراحل العمل



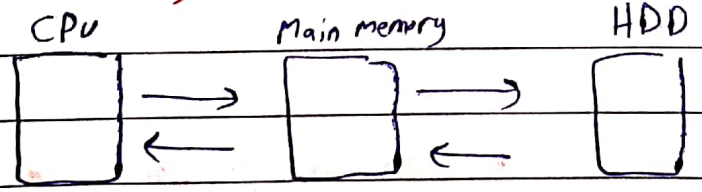
كل ما نخلص مرحلة ونروح للي بعدها بنقدر نرجع نستخدم المرحلة
الي قبل

(F) Performance via prediction

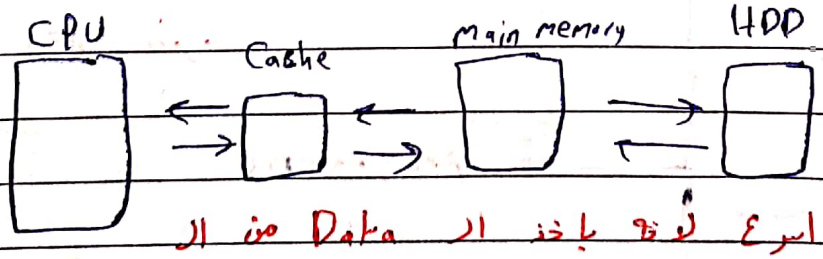
```
if (condition)
{
}
else
{
}
```

* بمر انفذ عن طريق التنبؤ بدل ما استنى الاذن
يعني كحلول بتنبأ انه true
* كل ما زادت نسبة التنبؤ بتستفيد اكتر
* اذا كانت نسبة النجاح > %50 ناجحة
< %50 عالفاضي وبمكن يخسر
%50 عالفاضي

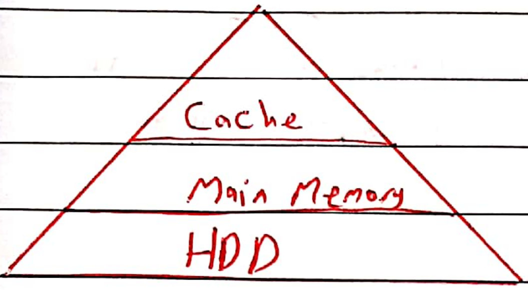
(G) Hierarchy of memories



الداتا بتتخزن باكتر من مكان

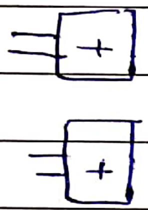


بهاي الحالة صارت اسرع لانه باخذ الـ Data من الـ
Main و الـ Cache



Cost	Speed	Size	access time
↓	↓	↓	↓
decreases	decreases	Increases	Increases

[11] Dependability via redundancy



بنكررها وبنستفيد منها في حال خربت وحدة
مش عشان نحسن ال CPU

§ 1.3

System Software
- → Compiler : translator
- → Operating System (os)

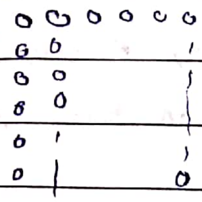
(15)

1. Input
2. output
3. Memory
4. Datapath
5. Control unit

(Input, output) → User Interface: Display / Storage devices / Network adapters

(Datapath, Control unit) → Processor "CPU"

Bit map:



1. Screen Size
2. resolution حجم ال الصورة

ممكن كل Pixel تكون ممثلة بال 8-Bit عشان الالوان
R
G
B

ال Frame buffer ~~وظيفته~~ وظيفته يخزن ال Bit map مؤقتًا

17 ال Capacitive بتسمح للواحد يكبس اكتر من مره مع بعض

18 ال A5 هو ال Processor

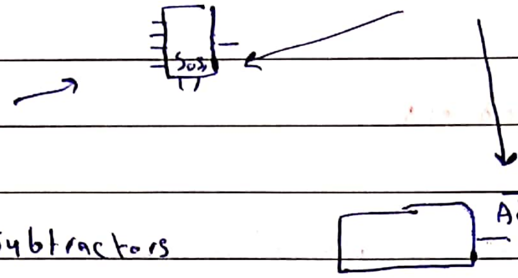
19

Datapath

Control unit

هو الي بيتحكم

MUXes



Decoders

Add/sub

Adder/subtractors

لما تكون ال cache جوا ال Processor بكون اسرع زي A5

20

ال 2 Arm core هدول ال Core تانيين هو Dual core

انا ال Data path ال graphices

22

6-inverters

Cache : Static RAM ع (اسرع)

لما نطفي ال System كل اشي بروح من ال Main

Main memory : Dynamic RAM بسحب Power أكثر

ما on capacitor

Non volatile الداتا ما بتروح منو ولما نطفي ال System ال Data عادي بتظل

23 ال Flash زي ال SSD ال Magnatic زي ال HDD

لازم نعرف انو مين اسرع ما لازم نحفظ الارقام.

24

(25) Defining Performance

* حسب عدد الركاب
* كم بتقطع بال Full
* Speed
* عدد الركاب مع الـ Speed

(26)

Response (execution) / Elapsed / wall clock time

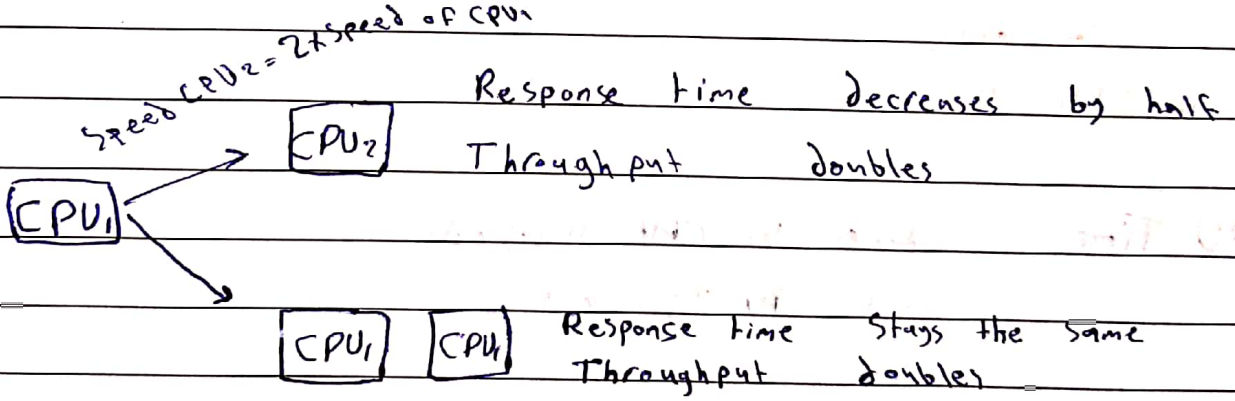
بمثل قديش الـ Computer بده time عشان ينفذ task زي لما تعمل google ع Search

Throughput

قديش عدد الوظائف اللي عملتها خلال زمن معين

Data center لو عندي

task 10 ns (1) through put 60 task/60ns ← اتنين , 20



Response time decreases by half
Throughput doubles

Response time Stays the same
Throughput doubles

(27)

Performance = $\frac{1}{\text{Execution time}}$ علاقة عكسية

X is n-times faster than Y

$\frac{\text{Performance } X}{\text{Performance } Y} = n = \frac{ET\,Y}{ET\,X}$

CPU A = 10 sec

CPU B = 15 sec

$\frac{ET_B}{ET_A} = \frac{15}{10} = 1.5$ ← الاجابة

A is 1.5 times faster than B

$\frac{\text{Performance A}}{\text{Performance B}} = \frac{\frac{1}{5}}{\frac{1}{15}} = \frac{15}{5} = 3$

A is

28 idle time يكون البرنامج يستنى

System performance = $\frac{1}{ET}$

↓

هو الزمن لكل اشي مح الـ Memory

والـ I/O والـ CPU

CPU Time يكون زمن الـ CPU بس الـ OS بنفذ

موجود لانه ممكن الـ CPU يحتاج

CPU Time هاذ لما الـ CPU ينفذ لحاله

System CPU Time هون لما الـ CPU يحتاج لـ OS

* CPU Performance = $\frac{1}{\text{User CPU time}}$

CPU Time = 10 ns

User CPU time 7 ns

System CPU time 3 ns

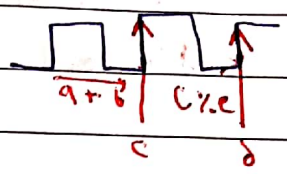
CPU's Sequential System

C = a + b

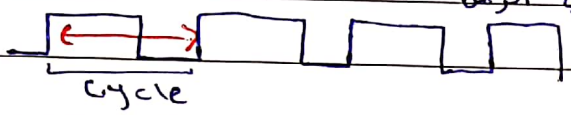
D = C * e

!

System Call X



بنستخدم الـ Cycle عشان نحسب الزمن



Clock Cycle time = Clock Period = 250 10^{-2} ps

= 0.25 ns = $\frac{1}{F} = \frac{1}{4 \times 10^9} = 0.25 \times 10^{-9}$ = 0.25 ns

Clock Frequency = number of Cycles per Second

Clock rate = F = $\frac{1 \text{ Sec}}{250 \times 10^{-12} \text{ Sec}} = \frac{10^{12}}{250} = \frac{1000 \times 10^9}{250} = 4 \times 10^9$ Hz

(30)

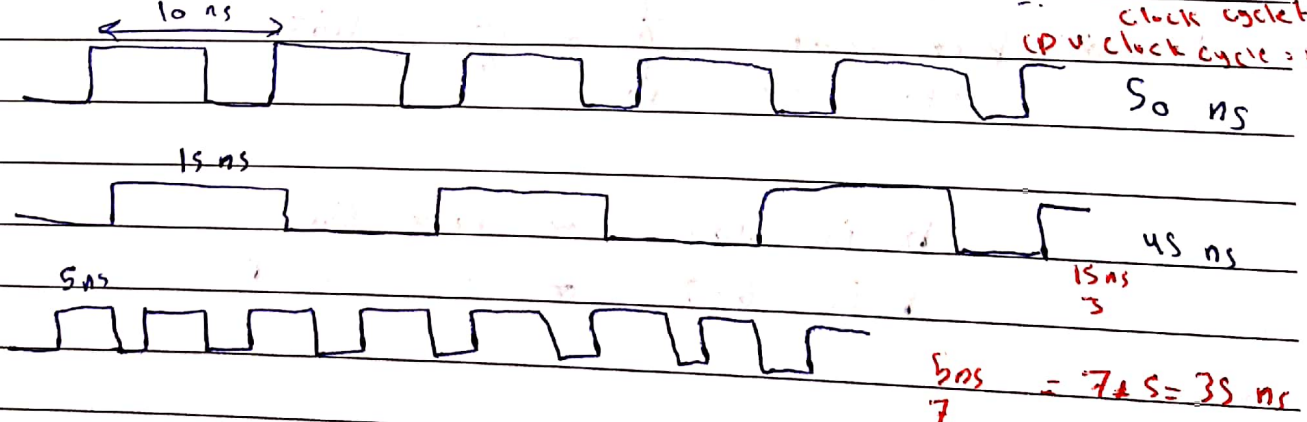
عدد الدورات زمن الدورة = 4 GHz

User CPU time = CPU Clock Cycles X Clock Cycle time

= $\frac{\text{CPU clock cycles}}{\text{Clock rate (F)}}$

User CPU Performance = $\frac{1}{\text{user CPU time}}$

اذا احينا نقلل عدد الـ Cycle لحاله بزيد زمن الـ Cycle الدورة



(31) $F_A = 2$ GHz

CPU time A = 10 Seconds

CPU time B = 6 sec

CPU clock cycles$_B$ = 1.2 × CPU clock cycles A

F_B = ??

$$\text{CPU time} = \frac{\text{CPU clock Cycles}}{\text{Clock Rate}}$$

$$10 = \frac{\text{CPU clock cycles A}}{2 \times 10^9}$$

CPU clock cycles$_A$ = 20×10^9 cycle

$$\text{Cycle time}_A = \frac{1}{F}$$

0.5 ns

$$\text{Cycle time}_B = \frac{1}{4} \text{ ns} = 0.25 \text{ ns}$$

CPU clock cycles B = 1.2 × CPU clock cycles A

$1.2 \times 20 \times 10^9$

$= 24 \times 10^9$ cycle

$$6 = \frac{24 \times 10^9}{\text{Clock Rate B}} =$$

$$\text{Clock Rate B} = \frac{24 \times 10^9}{6} = 4 \times 10^9 \text{ Hz}$$

4 GHz

$$\text{Performance B} = \frac{\frac{1}{6}}{\frac{1}{10}} = \frac{10}{6} = 1.67$$

B is 1.67 times faster than A

B is 67% faster than A

CPU clock cycles = Instruction Count X Clock per Instruction
// // // = IC x CPI

CPU time = Clock cycles x Clock cycle time
CPU time = IC x CPI x clock cycle time

= IC x CPI
Clock Rate

Program
ISA
Compiler

IC: Program, ISA, Compiler

multiply & add * C = a + b + d ;

← MAD: 1 instruction
multiply, add: 2 instructions

* C = a * 3 ;
↓ Compilar

Assembly Compiler 1 : add t,a,a
add c,t,a

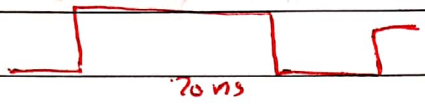
Compiler 2 : multiply c,a,3

* Sorting 3,4,7,0,7,5

HLL 10 lines → IC1 ≠ IC2 ← HLL 30 lines

CPI : Cycle ~~Clock~~ per Instruction

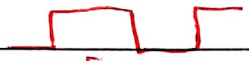
Single Cycle CPU



add 5ns CPI = 1

multiply 20 ns

Multi cycle CPU



add 1 cycle

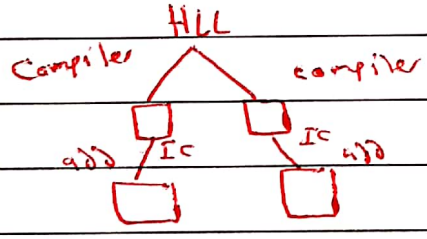
multiply 4 cycles

$CPI = \frac{14}{2} = 2.5$

33

CPU time = CPU clock cycles X Clock cycle time

= IC x CPIavg X clock cycle time



$CPU_{time A} = IC_A \times 2 \times 250 \times 10^{-12}$

$= 500\ IC_A \times 10^{-12}$

$CPU\ time_B = IC_B \times 12 \times 500 \times 10^{-2}$

$= 600\ IC_B \times 10^{-12}$

$$\frac{CPU\ time_B}{CPU\ time_A} = \frac{600 \times IC_B \times 10^{-12}}{500 \times IC_A \times 10^{-12}} = 1.2$$

A is 1.2 times faster than B

A is 20% faster than B

34

Instr	add	Mul	load
CPI	5	10	20
Relative freq	50%	20%	30%
IC	5	2	3

$\frac{Ici}{IC\,total} = \frac{5}{10} = 0.5$

Arithemtic Average CPI $= \frac{5+10+20}{3} = 5\times\frac{1}{3} + 10\times\frac{1}{3} + 20\times\frac{1}{3}$

$= 11.67$

Weighted Average CPI $= 5\times0.5 + 10\times0.2 + 20\times0.3$

$= 2.5 + 2 + 6 = \boxed{10.5}$

$$CPI\ avg = \sum_{i=1}^{n} CPI_i \times \frac{IC_i}{IC\ total}$$

OR $\quad CPI\ avg = \frac{CPU\ clock\ cycles}{IC\ total}$

CPU clock cycle $=$ IC total $\times$ CPI avg

$$= IC\ total \times \sum_{i=1}^{n} CPI_i \times \frac{IC_i}{IC\ total}$$

$$CPU\ clock\ cycles = \sum_{i=1}^{n} CPI_i \times IC_i$$

من الجدول CPU clock cycles = ??

CPU clock cycles $=$ IC total $\times$ CPI avg

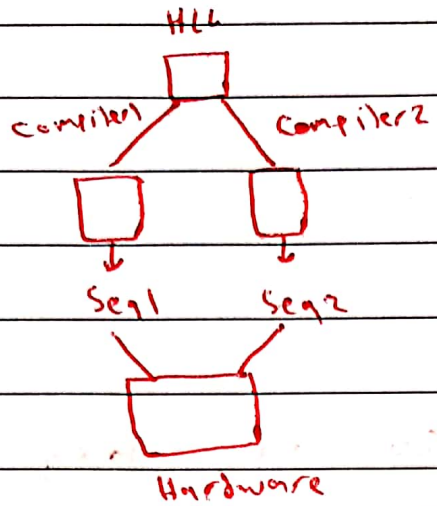
$= 10 \times 10.5$

$= \boxed{105}$

OR CPU clock cycles $= 5\times5 + 10\times2 + 20\times3$

$= 25 + 20 + 6 = \boxed{105}$

35



CPU clock cycle1

CPU time seq1 = $IC_{total} \times CPI\ avg \times$ clock cycle time

$(\sum CPI_i \times IC_i) \times$ clock cycle time

CPU clock cycles seq1 = $1 \times 2 + 2 \times 1 + 3 \times 2 = \boxed{10}$

CPI avg seq1 = $\frac{10}{5} = \boxed{2}$

CPU clock cycles seq2 = $1 \times 4 + 2 \times 1 + 3 \times 1 = \boxed{9}$

CPI avg seq2 = $\frac{9}{6} = \boxed{1.5}$

هاد بكون احسن لانا كانوا نفس ال Hardware

CPU time = CPU Clock cycles × Clock cycle Time

CPU time = IC × CPI avg × clock cycle time

CPU time = $\left(\sum_{i=1}^{n} IC_i \times CPI_i\right)$ × clock cycle time

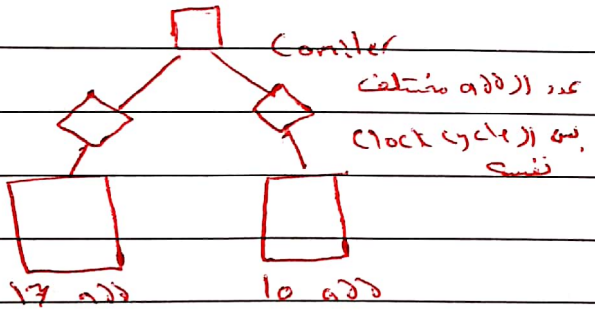
$IC \times CPI_{avg} = \sum_{i=1}^{n} IC_i \times CPI_i \rightarrow CPI_{avg} = \sum_{i=1}^{n} CPI \times \frac{IC_i}{IC}$

$CPI_{avg} = \frac{\text{CPU clock cycles}}{IC}$

$\frac{IC_i}{IC}$ ← relative frequency

* Same ISA + Same compiler → Same IC_i & Same IC_{total}

* Same hardware (CPU) → - Same Clock cycle time (Clock Rate)
- Same CPI$_i$



36

$\frac{Perf_1}{Perf_2} = 2 = \frac{CPU\ time_2}{CPU\ time_1} = \frac{CPU\ clock\ cycles_2 \times clock\ cycle\ time_2}{CPU\ clock\ cycles_1 \times clock\ cycle\ time_1}$

$= \frac{\sum_{i=1}^{n} IC_i \times CPI_i}{\sum_{i=1}^{n} IC_i \times CPI_i}$

$2 = \frac{\sum_{i=1}^{n} IC_i \times CPI_i}{\sum_{i=1}^{n} IC_i \times CPI_i} = \frac{2\times1 + ?\times2 + 2\times3}{1\times1 + 2\times2 + 4\times3}$

CPU time = IC x CPI avg x clock cycle time

$$= \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{clock cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Cycles}} = \text{Seconds/Program}$$

$$\text{CPI avg} = \frac{\text{CPU clock cycles}}{\text{IC total}}$$

Chapter 2

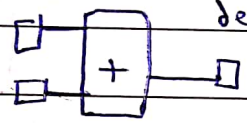
$IC_{Risc} > IC_{cisc}$

RISC	CISC
Fixed length	Variable length
Simple Functionality	complex functionality
fast execution	Slow execution

(5)

3 operands

Sources



لما بنريد نرجع ٢ ارقام لنة بخزنها
ونعمل تخزين لهيك operands ٣

add a,b,c → a = b+c

a = b+c+d+e ;

↓ Compiler

add a,b,c → a = (b+c) ;

add a,a,d → a = (a+d) ;

add a,a,e → a = a+e ;

بصير نفس ال operands
destination, Source يكون

RISC-V → regularity

3 operands هيك بكون

destination, source, source

6 add t0, g, h

add t1, i, j

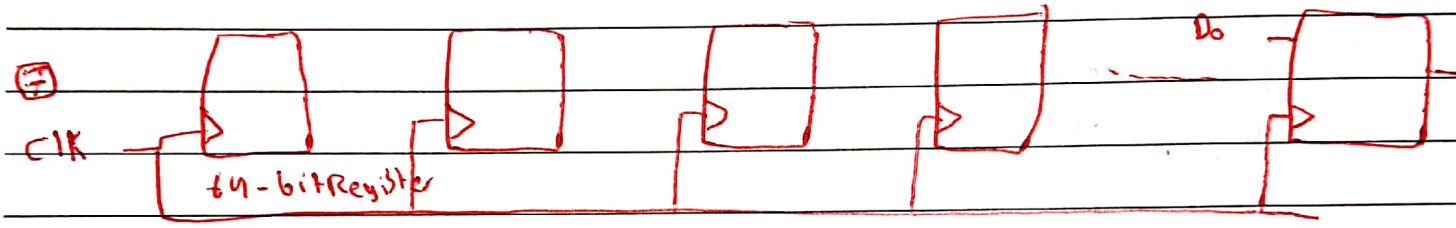
Sub f, t0, t1 و f = t0 - t1 الترتيب مهم

ممكن هيك

add f, g, h

Sub f, f, i

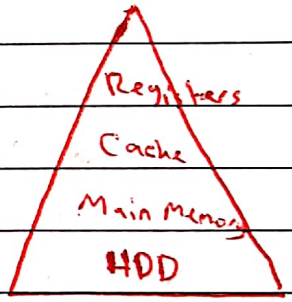
Sub f, f, j



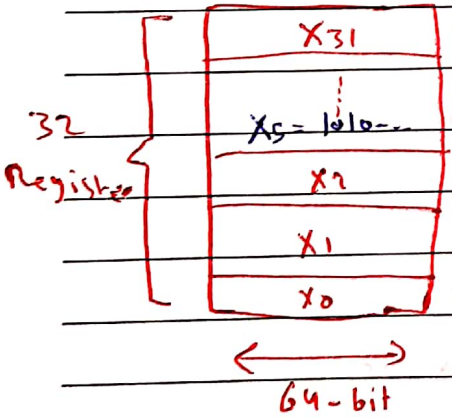
add c, a, b add X5, X7, X5

destination "write" Source "read"

32-bit



Register file



8-bit ≡ byte

32-bit ≡ word

64-bit ≡ double word

استخدمنا 5 عشان يكون 32 وهيك ما بأخذ كتير بخصوص الـ instruction bit منيح هيك عشان نقدر نميز

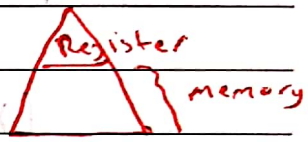
مثلا لو كل وحدة اخذت 5 Bit X5, X7, X5 هي هيك 15 Bit بدل 17 لـ 32

يعني 2^7 = 128 instruction

(9)

```
add  X5 , X20 , X21        OR      add  X19 , X20 , X21
add  X6 , X22 , X23                sub  X19 , X19 , X22
sub  X19 , X5 , X6                 sub  X19 , X19 , X23
```

load memory → Register (Reading from memory & writing to the RF)

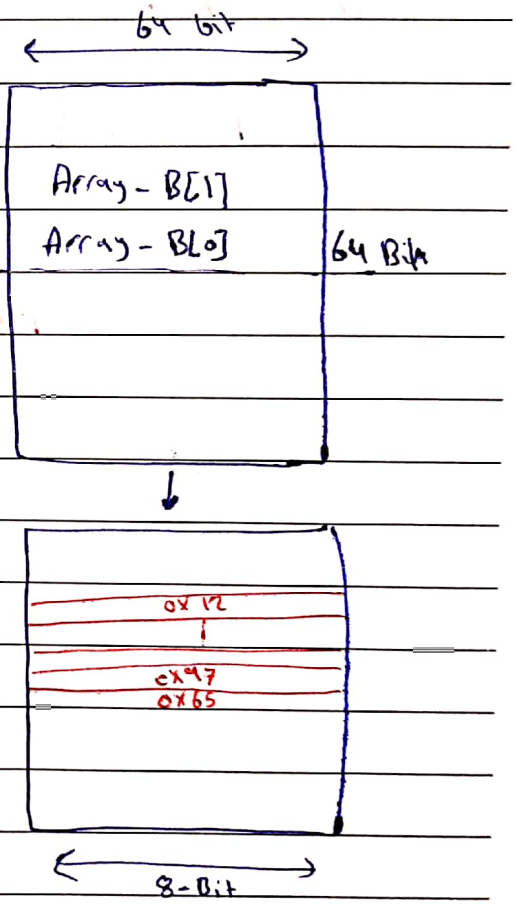


Store ≡ (Reading from RF & writing to the memory) Register → memory

(11)

long long int Array_B[10];

64 Bit



لانها 8 Bit هتتخزن كل تنين مع بعض

اول bit بالشاشة

Actual memory Address : base address of the Array + (index * size)

16 + index * 8

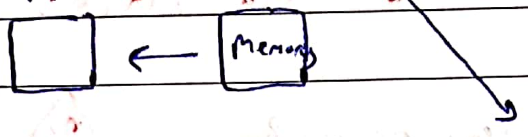
Array B[1] → = 16 + 1 * 8 = 24

little endian بتبدأ من الاخر

load (destination)

[12] Ld X28, 32, (X18)

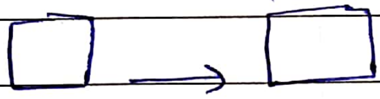
(index * size) offset base register (base address مخزن فيه اي)



(X28) ← Memory [32 + (X18)]

store: الريجستر اللي واصله (source)

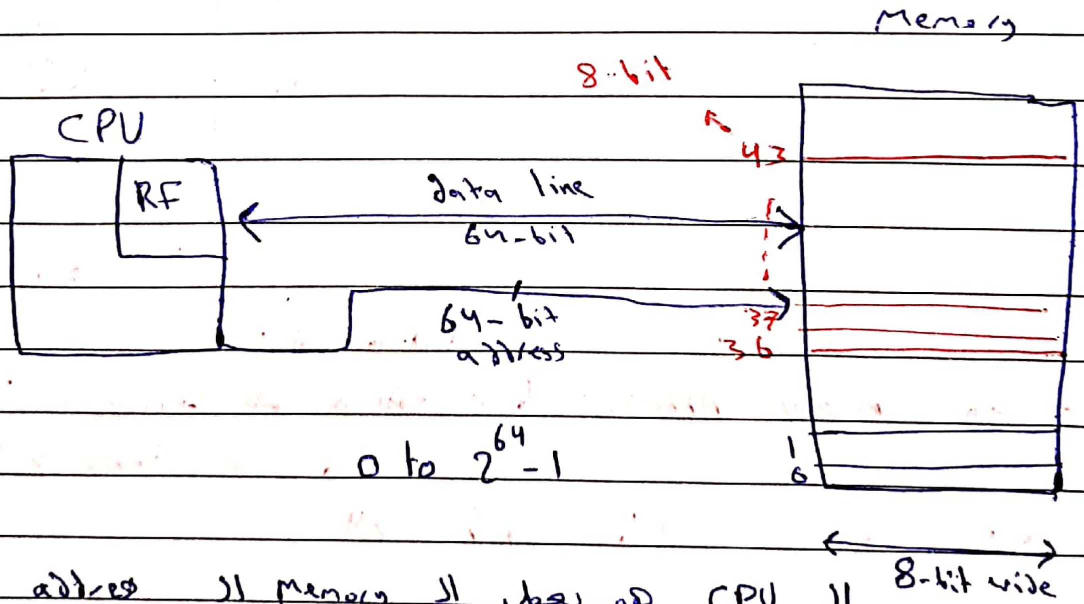
Sd X28, 32 (X18)



Memory [32 + (X18)] ← (X28)

** [11] Revi..

Memory is byte addressable



الـ CPU هو بيعطي الـ Memory الـ address

مرحلة بنخزن

[X28] بـ RF بال ثم offset base register

Ld X28, 32 (X18)

32 + (X18) → 4 = [36]

X28 ← Mem[offset + (base register)]

X28 ← Mem[offset + (X18)]

Store (s)

SD X28, 32(X18)

Mem[offset + (X18)] ← (X28)

* Write an instruction that reads the memory starting from address 8 & write the data into register X17?

LD X17, 8(X0)

[0] = X0 وايها بتكون قيمة

[8] = 0 + 8 في عنان هيك وطيناه

(13)

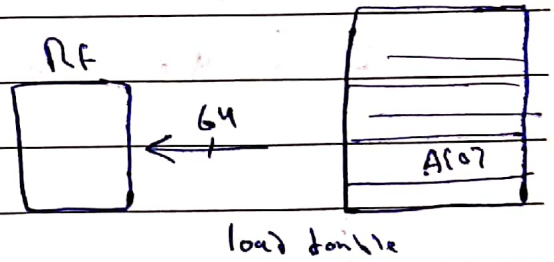
g = h + A[8]

↓ ↓

X20 X21

LD X5, 64(X22)

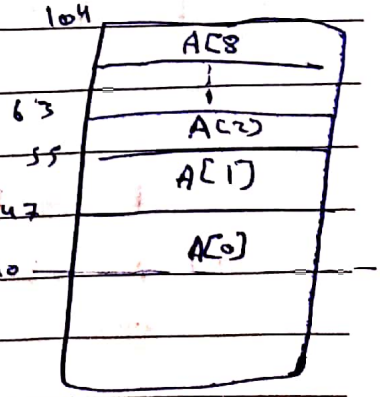
add X20, X21, X5



offset = index x (Size in bytes)

index x 8

Memory adress = (base address register) + (index * Size)
(in byte) offset



* لما تكون الـ Array عالميني
بنكون load عالشمال store .

14

A[12] = h + A[8]

```
Ld  X6, 64(X22)
```
X6 ← بيتسدر نخلبا

```
add  X7, X6, X21
Sd  X7, 96 (X22)
```
96 ← 12 × 8
دايما

g = A[8] - h ;

```
Sub X22, X6, X21
```
↓ load A[8] .

لو خطينا كما سطر الوسن نحمل
X7 بدالها X6
بنروح قيمة الـ load
وبيعمل load كمان مره

16 Immediate (ثابت) Operands

B = A + 4 ; (B → X21, A → X20) ⇒ addi X21, X20, 4

$2^{11} - 1 = 2047$

$-2^{11} = -2048$

اكثر من هيك بطريقة ثانية

D = B - 3 ; (D → X8, B → X7)   addi X8, X7, -3

$X_7 + (-3)$

لو كان

D = 3 - B

ما بزبط لازم نعمل load
وبزون Addi

$D = 3 - B$;   Sub X10, X0, X9 ← (-B)

(D → X8, B → X9) $ addi X8, X10, 3

هيك احسن من انه نحط load

18

$Value = d \times r^i$

$value = 1 \times 2^2 = 4$

$value = 1 \times 2^1 = 2$

0 1 1 0

... i 2 1 0 i

$1 \; --- \; 1,1 \; 1,1 = 2^{64} - 1$

$x5 = b_{63} \; ..... \; b_2, b_1, b_0$

$= b_{63} \times 2^{63} + \; ... \; + b_1 \times 2^1 + b_0 \times 2^0$

$1111 = 2^4 - 1 = 15$ (3 2 1 0)

20

اذا صار شي overflow هو بحلها لحاله

21

ال Sign bit هي ال Most s

addi , , -4

addi , , +4

ال Sign Bit في كانوا يحطوها عاليمين وفي يحطوها عاليسار

0 1 1 1 → $2^{n-1} - 1$ أكبر

1 0 0 0 → $-(2^{n-1} - 1)$ أصغر

① Sign-magnitude   0 100 (+, 4)   1 110 (-, 6)

0 001 (+1)

Sign-mag 1 001 (-1)

1's comp. 1 110 (-1)

1 111 → $2^n - 1$

② 1's complement   بنعكس كلشي

③ 2's complement

لو دمجنا الموجب مع السالب ب 1's

$x + (-x) = 2^n - 1$

المشكلة بال Magn و 1's انه الصفر الها تمثيلين

Scanned by CamScanner

(23) 2's complement

b_{n-1} ---- , b_1, b_0

Value = $b_{n-1} \times -2^{n-1}$, --- $b_1 \times 2^1 + b_0 \times 2^0$

Pos. اكبر 0 1111---1 $= 2^{n-1} - 1$

0 0000---0

neg. اكبر 1 0 00 --- 0 $= -2^{n-1}$

$(1111)_2 = (-1)_{10}$

$1 \times 2^0 + 1 \times 2^1 + 1 \times 2^2 + -1 \times 2^3$

$1 + 2 + 4 - 8 = -1$

دائما بطرح 1- كل الرقم من عدد الـ 1

+16 00010000

-24 11101000

11111000

-8

بكون الـ 0 تمثل واحد الـ Zero

(26) 2's complement

① 1's Complement ② الطريقة العادية

② add 1

$X + \bar{X} = 111---1 = 2^n - 1$

$= -1$

$X + \bar{X} = -1 \rightarrow$ $-X = \bar{X} + 1$

↓ 1's comple.

negation اذا بدنا نحول من + ← − أو − ← +

الرقم مع سالبه

$X + (-X) = 2^n$

$X + (\bar{X} + 1) = (X + \bar{X}) + 1$

$111---1 + 1$

$\downarrow 0---0000 = 2^n$

-2: (1 ... 1111 0)64

+2: 0 ... 0000 10

(27) Sign Extension

Unsigned اذا

Zero بضيف

Sign اذا كان بضيف

Sign bit الـ حسب

Note:

* add, sub X?, X?, X?

load/store X?, offset (X?)

addi X5, X10, 5 → bit 12 يكون

64 bit ↘ 12 bit

addi عشان يقدر يعمل Sign فلازم نعمل extension !

(28) ص §2.6 Representing Instructions

* 32-bit كلتي بكون
* Formats حسب موزعين
* Regularity → منظم

(30) R-format : register-format

	00111	00101		00110	
Funct7	rs2	rs1	Function-3	rd	opcode
7 bit	5bit	5bit	3bit	5Bit	7 Bit

رقم ال destination

بساعدك على تحديد نوع ال instruction

$2^7 = 128$ instruction

بين

opcode + Funct3 + Funct7

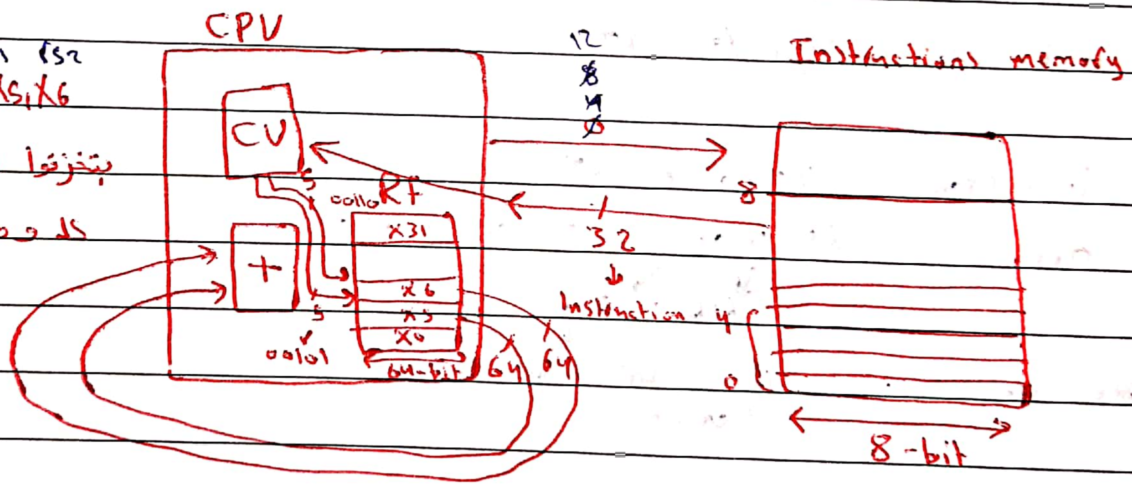
2^{17} instruction

add X6, X5, X7
rd rs1 rs2

rd rs1 rs2
32-bit add X7, X5, X6

بتخزنوا بال Memory ب [4] لوكة

كل وحدة 8bit



Funct7	00110	00101	Funct3	00111	opcode 7-bit

31 تحويل من Assembly ل Machine وبنقدر نجمع كل 4-bit مع بعض ونحول ل octa.

لو اعطانا ← مطلوب نحول ل Assembly

$(015A04B3)_{16}$

0000 0001 0101 1010 0000 0100 1011 0011

x21 x20 3 x9 add

add x9, x20, x21

32 I-Format

or offset immediate 12 bit	rs1 5-bit	funct3 3-bit	rd 5-bit	opcode 7-bits

الي بنستخدم فيها

addi, Ld → sign number ممكن موجب او سالب

addi rd, rs1, imm → 12-bit (rs1 → 64-bit)

Ld rd, offset(rs1)
(offset → 12-b, rs1 → 64-bit)

2's complement

0 111 ... 1 = $+(2^{11}-1)$

اكبر موجب = +2047

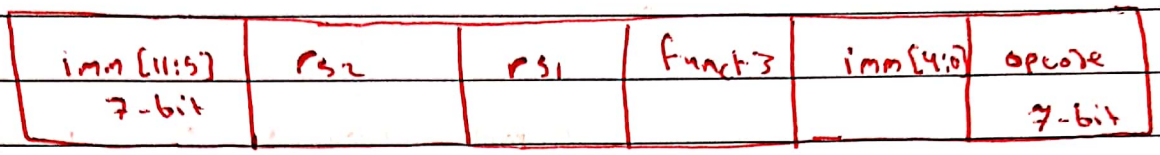
1 0000 ... 0 = -2^{11}

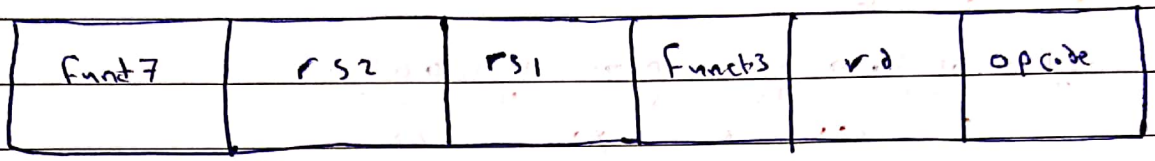
اصغر سالب = -2048

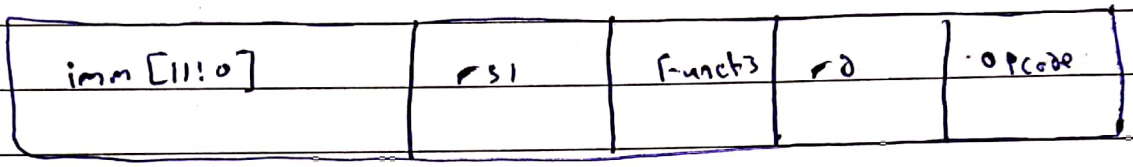
2047 byte ↑
2048 byte ↓

± 2048 (ف) 256 double words لفوق او لتحت

(33) S- Format







S-Format → sd

Regularity الـ 3 متشابهين عشان يكون

sd rs2, offset (rs1)

memory [offset + (rs1)] ← (rs2)

(34) add
sub
] Funct7 بتبين اذا كان add او sub

(35) A [30] = h + A [30] + 1

8 * 30

```
ld   x9, 240 (x10)
add  x9, x21, x9
addi x9, x9, 1
sd   x9, 240 (x10)
```

* [38] Shift Immediate Operations

Shift left logical "slli"
Shift right logical "srli"

slli rd, rs, imm
srli rd, rs, imm

Ex) * slli x6, x5, 10

64-bit
x5 = | 0 - - - 000010 | ← 0000000000

لـ رح يتخزن د x6 في x5 لما تتغير قيمته

x6 = | 0010000000000 |

$2^{11} = 2048$

0 → 0010
0001
0010 ← 0
0100

* اكبر اشي بقدر اعمله الـ Shift 63 لانه اكثر من هيك بصير الرقم صفر واحنا عنا طرق تخليه صفر زي add x, x0, x0 نطرحه من حاله مثلاً.

* الـ Shift left اذا بدنا ما يأثر بالاشارة لازم يكون الرقم صغير لانه ممكن تتغير الاشارة

slli | 10000000 | ← 0 هيك تتغير

slli | 11111 - - - 0 | ← 0 اما هيك ما تأثر

بالـ Shift right اذا كان Positive ما بتأثر اشارته اما اذا كان negative كماله بتحول

srli 0 → | 0 |

srli 0 → | 1 |

اما اذا كان الرقم Unsigned عادي، استخدم Shift right زي ما بدي

(39) and — Immediate → 0x FFF (1111 1111 1111)

OR — Immediate → 0x 000

(41) XOR I-format

1 ⊕ 1 = 0 ; 1 ⊕ 0 = 1

0 ⊕ 0 = 0 ; 0 ⊕ 1 = 1

X10 = 10

1111 1111 ~~~~

1's comp. → 01

* Store 2's complement register X7 in X5

Sub X5, X0, X7 ← OR — Xori X5, X7, 0xFFF ; addi X5, X5, 1

(42) Conditional Operations

branch equal (beq) — beq rs1, rs2, label — rs1 = rs2 — L1 Exit loop

branch not equal (bne)

(43)

f, g, h, i, j
↓ ↓ ↓ ↓ ↓
x19 x20 x21 x22 x23

```
    beq  x22, x23, IF      → F = g+h
    Sub  x19, x20, x21
    beq  x0, x0, Exit
IF: add  x19, x20, x21
Exit:
```

(or)

```
    bne  x22, x23, Else    → F = g-h
    add  x19, x20, x21
    beq  x0, x0, Exit
Else: Sub  x19, x20, x21
Exit:
```

(44)

```
while(Save[i] == k)
    i++;
```

i = x22

k = x24

base (array) x25

2³

8*i + x25
x10 + x25

```
Loop: SLLi  x10, x22, 3
      add   x10, x10, x25
      Ld    x9, 0(x10)
      bne   x9, x24, Exit
      addi  x22, x22, +1
      beq   x0, x0, Loop
Exit:
```

```
    beq  x9, x24, Cont
    beq  x0, x0, Exit
Cont: addi x22, x22, +1
    beq  x0, x0, loop
Exit:
```

(45) blt: branch if less than

bge: " " greater than or equal

Ex

blt x23, x22, L1	or
beq x0, x0, Exit	
L1: addi x22, x22, 1	
Exit: ...	

a > b → , a <= b	
bge x23, x22, Exit	
a>b ← addi x22, x22, +1	
Exit: ...	

(46)

bltu , bgeu

Unsigned

(47)

size = 10

0 ... 9

هون الـcompiler بده يتأكد انه الـ i بين الـ range اللي هو 0-9

A[i]

0 <= i < size

bgeu x20, x11, outofbounds	بدل هيك بنقدر نستخدم
	مثلاً اختبر اول شرط i >= 0
	x10: i
	x11 = 0 → دايما
	شيك بتقدر تشطبها

bge x20, x0, L1
beq x0, x0, handleOutOfBounds
L1: blt x10, x11, L2
beq x0, x0, handleOutofbounds
L2: ...

48 i = x5

base address of Save in x20

```
For(i=; i<10; i++)
  Save[i] = Save[i]*2
```

memory address = base + offset

base + index size

base + index x 8

2^3

```
add  x5, x0, x0
addi x6, x0, #10
loop: bge  x5, x6, Exit
      slli x7, x5, 3
      add  x7, x20, x7
      ld   x9, 0(x7)
      slli x9, x9, 1
      sd   x9, 0(x7)
      addi x5, x5, 1
      beq  x0, x0, loop
Exit: ____
```

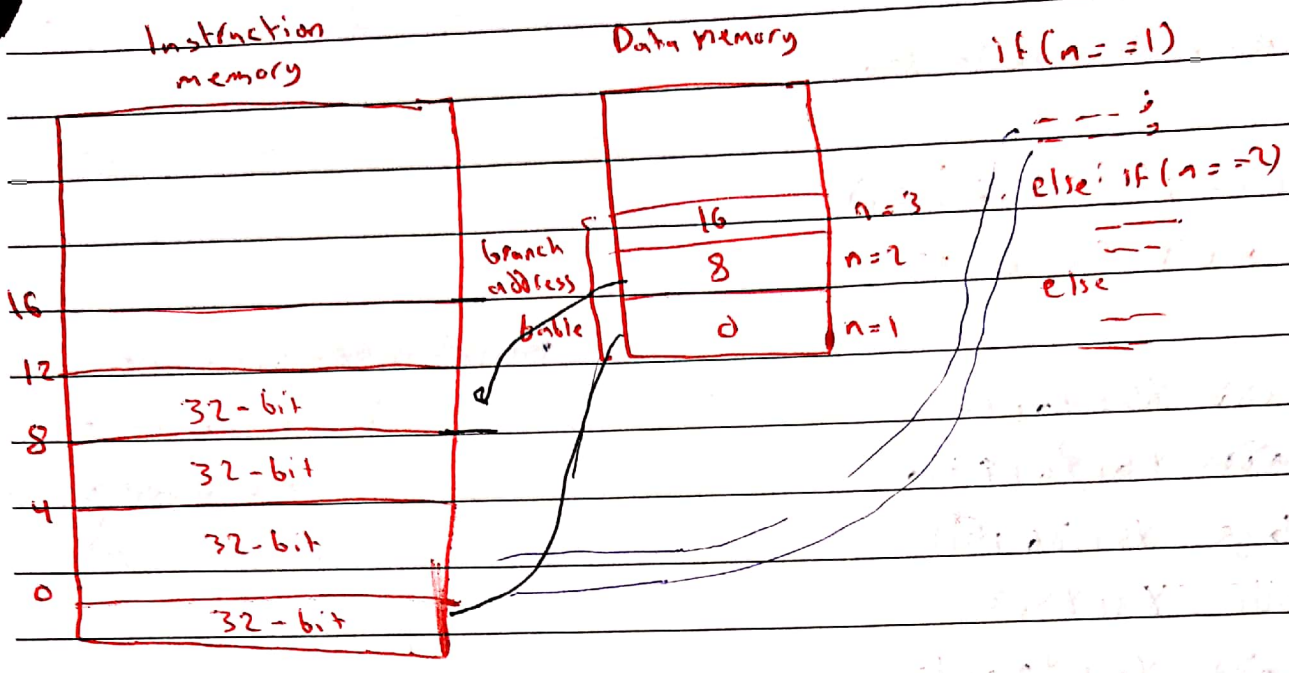
Switch(n) → x20

```
Switch(n) → if(n==-1)
Case 1: __
  __
  break;
Case 2: __
  __
  break;
  :
default: __
  __
  break;
```

```
else if(n==2)
  __
  __
else
  __
  __;
```

```
addi x5, x0, 1
bne  x20, x5, L2
__
__
(breakout) beq x0, x0, Exit
L2: addi x5, x0, 2
bne  x20, x5, Else
__
__
beq  x0, x0, Exit
Else: __
  __
```

300 instruction = 3*100 نزيد 2، Switch(100) لـ



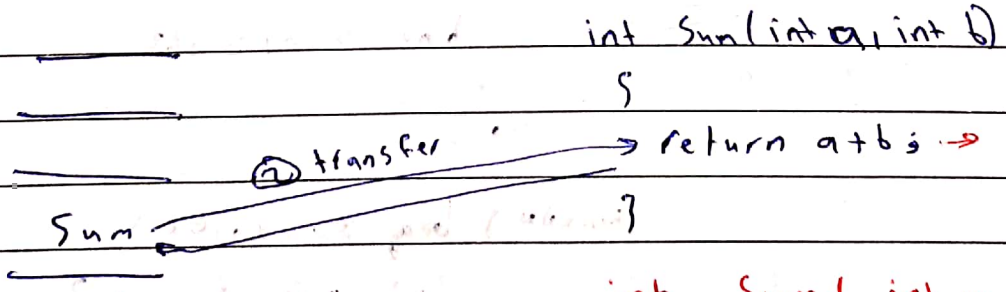
(49)

beq rs1, rs2, label

No branch targets ← address of the target instructions

بس اول اشي بصير

§2.8 (50)



int Sum(int a, int b)
{
int total; → ③ acquire storage
total = a+b;
} return total; → ⑥ return

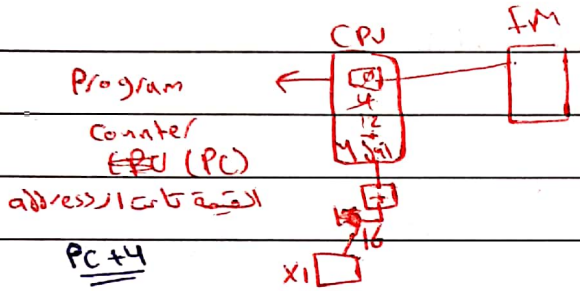
(51) jump and link (jal)

jal rd, label
↓
X1

address
0 add
4 Sub
8 Ld
12 jal X1, Sum
16 — — —

عشان تعمل return
بتخزن الـ address + 4

X1 = 16



Jump and link register (jalr)

jal rd, offset (rs)

① (rd) ← Pc+4
of "jalr"

② (Pc) ← offset + (rs1)

0 add	Sum — — —
4 Sub	— — —
8 Ld	— — —
12 jal X1, Sum	jalr X0, 0(X1)
16 — — —	

هاي بخزنيها
فيها 16 و الـ offset = 0
16
بترجع لـ 16
offset

Unconditional branch الشرط دايما متحقق

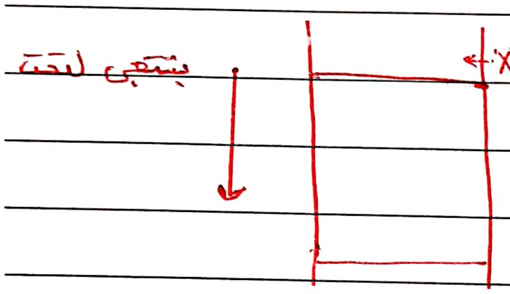
بعادلوا بهيك

beq X0, X0, Exit ≡ jal X0, Exit

52

Stack : last in First out
"LIFO"

Push → Store
Pop → load.



أول مكان بال Stack بكون X2 دايما

اذا احتجنا نقرا اشي من ال Stack من Stack pointer

Push
Sd , S2 , offset (X2) — SP

Pop ← Ld (SP)

53 leaf

اسمه leaf لانه ما بنادي واحد بعده
بنادي ↓ procedure

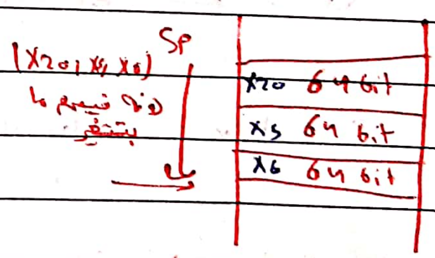
لانه نزل (3) ثم خزنهم ف 24 = 3 * 8

Push X20 , X5 , X6

```
addi sp, sp, -24
sd X20, 16 (sp)
sd X5, 8 (sp)
sd X6, 0 (sp)
```

operation

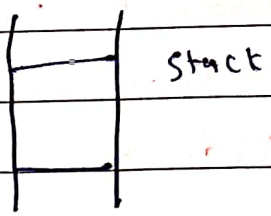
```
add X5, X10, X11
add X6, X12, X13
sub X20, X5, X6
```



الصفحة التانية ↓

```
addi x10, x20, 0
ld   x20, 16(sp)
ld   x5, 8(sp)
ld   x6, 0(sp)
addi sp, sp, 24
jalr x0, 0(x1)
```

Pop



هيك برجع الـ Pointer لمكانه الاصلي

temporaries ← load و store خاص دائي نعمل

save بس الـ

بصير هيك

```
addi sp, sp, -64
sd   x20, 0(sp)

add  x5, x10, x11
add  x6, x12, x13
sub  x20, x5, x6

addi x10, x20, 0
ld   x20, 0(sp)
addi sp, sp, 8
jalr x0, 0(x1)
```

Main:

x10 x11 x12

Z = leaf example (7, 5, 3, 6);

Z = 12 - 9

Z = 3

```
0  addi x10, x0, 7
4  addi x11, x0, 5
8  addi x12, x0, 3
12 addi x13, x0, 6
16 jal  x1, leaf example
```

$n = 5$

$fact(5) = 5 \times 4 \times 3 \times 2 \times 1 = 120$

$Fact(1) = 1$

$Fact(0) = 0.$

$n=5 \longrightarrow 5 \times Fact(4)$ ← $24 \times 5 = 120$
(call)

$4 \times Fact(3)$ ← $4 \times 6 = 24$
(call)

$3 \times Fact(2)$ ← $3 \times 2 = 6$

$2 \times Fact(1)$ ← $2 \times 1 = 2$

$1 \times Fact(0) \longrightarrow 1 \times 1 = 1$

58

```
Fact: 20  addi  Sp, Sp, -16
      24  Sd    X1, 0(Sp)
      28  Sd    X10, 8(Sp)
      32  addi  X5, X0, 1
      36  bge   X10, X5, Else
      40  addi  Sp, Sp, 16
      44  addi  X10, X0, 1
      48  jalr  X0, 0(X1)
52 Else:  addi  X10, X10, -1
      56  jal   X1, Fact
      60  addi  X6, X10, 0
      64  Ld    X1, 0(Sp)
      68  Ld    X10, 8(Sp)
      72  addi  Sp, Sp, 16
      76  mul   X10, X6, X10
      80  jalr  X0, 0(X1)
```

(24–52) بقدر انقل هدول لل Else ما بتغير

X_{10} X_5

$n < 1$

(44) → عشان الارجاع د X_{10}

لماذا Sd ل X_1 و X_{10} ؟؟ لأن بال Else رح يتغير قيمة التنين.

(60) لما ارجع ارجع هون رح يكون القيمة مخزنه ب X_{10} [$X_{10} = Fact(n-1)$

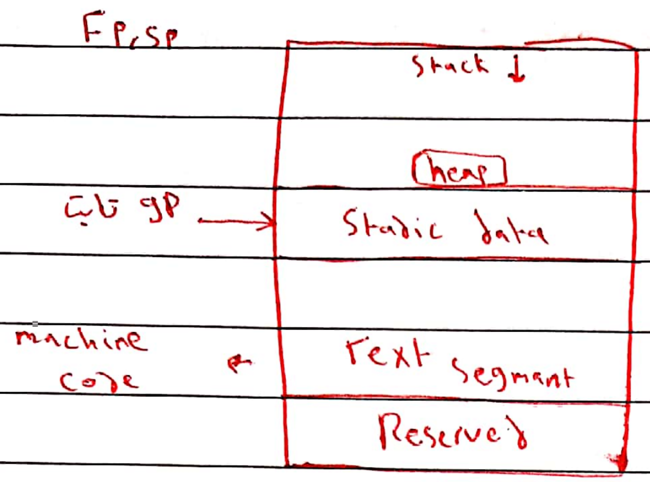
$n-1 \geq 0$

هاي بالبداية عمل

+ اذا كان caller دايما X_1 بتخزنها

Virtual Address Space

(1)



Ld x7, _ (gp)

لما احط ال gp بالنص → -2^{11} to $2^{11}-1$

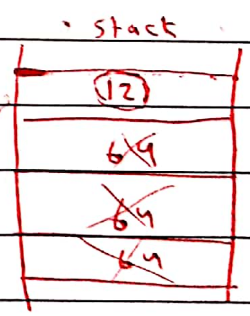
عشان (اكبر) ارقام (+) و (-)

(2) Non leaf ← لازم اخزن ال x1

```
Sum: 40  addi sp, sp, -8
     44  sd x1, 0(sp)
     48  bge x0, x10, Else
     52  add x11, x11, x10
     56  addi x10, x10, -1
     60  jal x1, Sum
     64  beq x0, x0, Exit
Else:    add x12, x11, x10
Exit:    ld x1, 0(sp)
         addi sp, sp, 8
         jalr x0, 0(x1)
```

(52–56) الترتيب مهم لانه بجمع بعدين بطرح

x10 = 3
x11 = 0
x1 = 12




```
Main: 0  addi x10, x0, 3
      4  addi x11, x0, 0
      8  jal x1, Sum
     12  add x20, x12, 0
```

(or) iteration

```
long long int Sum(long long int n, long long int acc)
  while(n>0)
  {
    acc = acc + n;
    n = n-1;
  }
  return acc;
```

```
Sum:  bge  X0, X10, Exit
      add  X11, X11, X10
      addi X10, X10, -1
      beq  X0, X0, Sum  ≡  jal X0, Sum
Exit: add  X12, X10, X0
      jalr X0, 0(X1)
```

(65) (1) lb = load byte (8-bit)

(2) lh = load half word (16-bit)

(4) lw = " word (32-bit)

(8) ld = " doubleword (64-bit)

lb → byte at 32 — rd, offset(rs1)

lh → bytes at 32, 33 — 32(X0) = 32

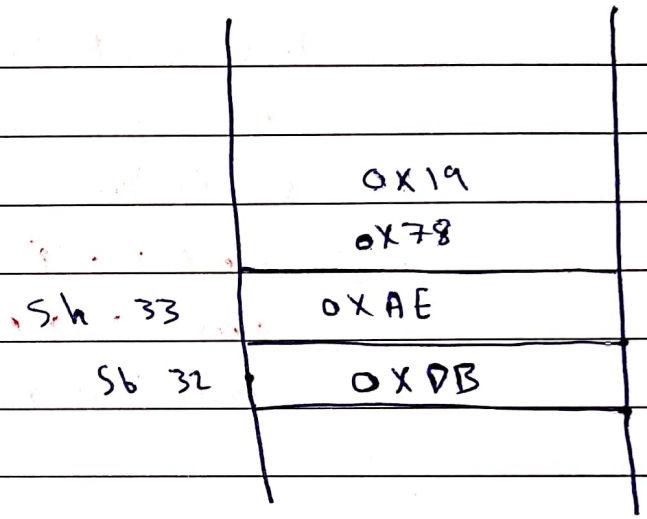
lw → " " 32, 33, 34, 35

ld → " " 32, ---, 39

sb
sh
sw
sd

[s2, offset (rsi)

0x FFBC23 45 19 78 AE DB



16

1000, 000, 000 → 64-bit

16 x 8 = 128-bit

11 x 8 = 88-bit

40 bit

67

```
Strcpy:  addi sp, sp, -8
         sd   x19, 0(sp)
         addi x19, x0, 0
loop:    add  x5, x19, x11
         lbu  x5, 0(x5)
         add  x6, x19, x10
         sb   x5, 0(x6)
         beq  x5, x0, Exit
         addi x19, x19, 1
         jal  x0, loop
Exit:    ld   x19, 0(sp)
         addi sp, sp, 8
         jalr x0, 0(x1)
```

هدفها الي عملها copy للـ Y[i] و Paste جوا الـ x[i]

Y[i] → 1 * i + base address (1 → دايمًا char)

لما نتعامل مع char بنستخدم Lbu

69 § 2.10

imm	rs1	funct3	rd	opcode
12				

-2^{11} to $(2^{11}-1)$

x20 → $X = 0X73CD14AF$ (32 bit)

ما بزبط لانه بوسع 12 وإحنا بدنا 32 → addi x20, x0, ___ (12 bit)

load upper immediate (lui)

imm	rd	opcode
	5-bit	7-bit

sign-ext	imm	0 000---00
31	12	11 ... 0

→ rd

X = 0x73CD14AF

X = 0x73CD18AF

0x73CD2 ← هون زاد الـ 1 مئات بطلع صح.

0x0000 0000	0x73CD1	0x000	X20
0x0000 0000	0x00000	0x4AF = 0100 0x8AF	
0x FFFF FFFF	0x FFFFF		

73CD186

lui X20, 0x73CD1

addi X20, 0x4AF

X20: 0x0000 0000 73CD14AF

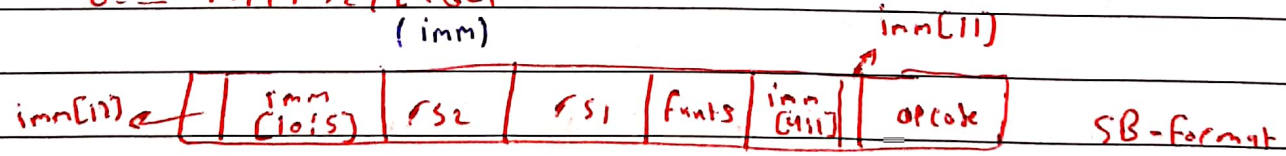
لما الـ Compi. يستوف انه 0x73CD18AF (1) بزيد واحد عشان يطلع صح

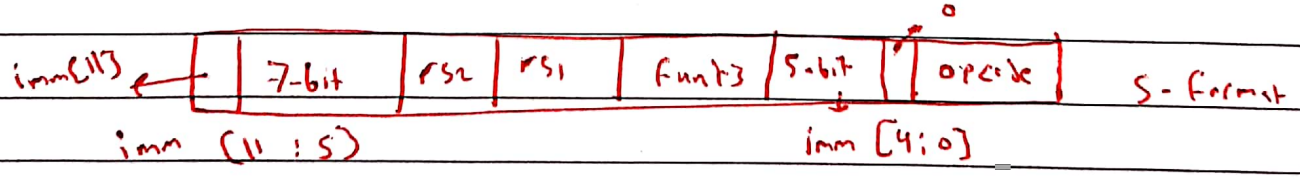
* lui X10, 0x ABCDE	X10 = 0x FFFF FFFF ABCDE 0x000
* addi X10, X10, 0x825	0x FFFF FFFF FFFFF 0x825 (ينتقص واحد)

X10 = 0x FFFF FFFF ABCDD825

(7)

b__ rs1, rs2, label (imm)





target address = PC of branch + immediate x2

بضرب بـ (2) عشان الـ loop

```
0   loop:
4
8
12
16   beq x0,x0,-8
```

→ 16 + -8 * 2 = 0

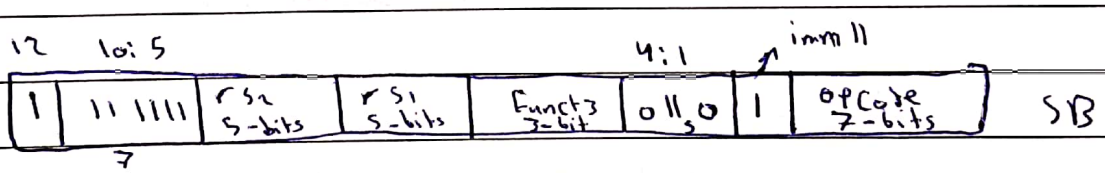
```
0   loop
4
8
12
16
20   beq x0,x0,loop
```

Target address = PC of branch + immediate x2

0 = 20 + immediate x2

imm = -10

$(-10)_{10} = (1111\ 1111\ 01\ 10)_2$



loop هيك برجع بعمل 20 + -10 x 2 = 0

لما يكون الـ target قبل الـ instr. بضيف سالب اذا بدي اعد عدد معين عنا 5 وبنضربه بـ 2 وبنحط سالب

addi, Ld

imm[11:0]	rs1	Funct3	rd	opcode

⇒

Sd

imm[11:5]	rs2	rs1	Funct3	imm[4:0]	opcode

branch

imm[12] imm[11]

imm[12|10:5]	rs2	rs1	Funct3	imm[4:1|11]	opcode

most هاي حطها كون عشان تكون الـ

Sign ex. عشان الـ

imm(11)

imm[20] ←

imm[10:5]	imm[4:1]	imm[19:12]	1bit	19:12	5bit rd	7-bits opcode

jal X0, loop

Target = PC + immx2

branch ⇒ -2^{11} to $(2^{11}-1)$ halfwords

-2048 to 2047 halfwords → -4096 to 4094

± 4K bytes

± 1K instr. (word)

jal → -2^{19} to $(2^{19}-1)$ halfwords

-512K to 512K

اكبر

(74)

Exit

Exit 80024 = 80012 + immx2

Exit = 6

loop 8000 = 8020 + immx2

imm = -10

branch

jal

jalr rd, offset(rs)

Target address = offset + (rs)

64-bit

PC = 0

0X 0000 0000 ABCD 1230

lui x20, 0xABCD1

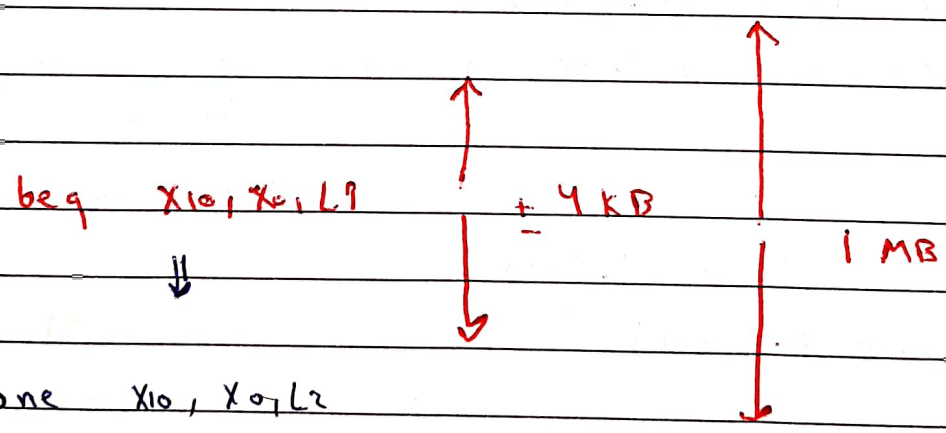
addi

jalr x0, 0(x20)

lui x20, 0xABCD1

jalr x0, 0x230(x20)

ال jalr بتعطينا مجال اكبر



bne x10, x0, L2

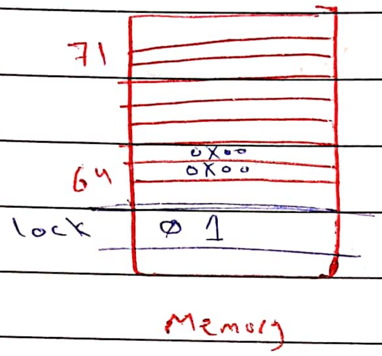
jal x0, L1

مثلا هنا 77 منيح بيكون هنا بالاختبار موجود كلشي الا ال Shift

§2.11 (78)

CPU1 CPU2

CPU1	CPU2
lock();	lock();
① ld x19, 64(x0)	② ld x19, 64(x0)
③ addi x19, x19, 1	④ addi x19, x19, 1
⑤ sd x19, 64(x0)	⑥ sd x19, 64(x0)
unlock();	unlock;



Mutual Exclusive Region
"ME" Region
انه الجزء بس واحد منهم يفوت

لما التنين قرأوا مع بعض طلعت النتيجة [1] غلط لازم [2] لأنه التنين قرأوا البداية صفر لازم كالدور هيك بطلع [2] فلازم ① ② ③ ④ ⑤ ⑥

عشان ال ME Region بنضيف lock() و Unlock هيك كل واحد بستنى التاني

بدخلي ال Value = 1 لما يكون lock ولما يبطل بتصير lock=0 هيك بطل lock

(Atomic)

عملية ال check من ال Value بتبين انها Single بيجي ال CPU بتشوف اذا lock=0 or 1 اذا كانت 0 بجلسا بخليها [1] وهيك صار lock واي واحد بيجيها بتوقفه [1] فبطل busy.

79

lr.d rd, (rs1) ⟶ (rd) ← Memory[(rs1)]

sc.d rd, (rs1), rs2 ⟶ Memory[(rs1)] ← (rs2)

(rd) = 0 Success

(rd) != 0 (Fail)

بكون في Processor تاني غير اللي واحد sc.d مستخدمه

Memory address	Processor
(rs1)	P1
(rs1)	P2

Swap (X20), (X23)

Swap X23, (X20) X

```
0  Loop: Lr.d  X10, (X20)
5        Sc.d  X11, (X20), X23
5        bne   X11, X0, Loop (Fail)
6        addi  X23, X10, 0
```

atomic ⟶

3	
4	
7	
8	

address	Processor
(X20)	P1
X20	P2

لما يجي يعمل ③ رح يلاقي تنتين وهيلاقي ما بزبط فيحذف اليمين نفس المكان بستبدل P1 | X20 هيك لما يجي لـ ④ صارت تزبط لانه بس وحدة P2 | X20

80

Lock ( ); — address of variable lock is in (X20) →

```
      addi  X12, X0, 1
loop: Lr.d  X10, (X20)
      Sc.d  X11, (X20), X12
      bne   X11, X0, loop
      bne   X10, X0, loop
```

→ lock | 1

```
Ld    X19, 64(X0)
addi  X19, X19, 1
Sd    X19, 64(X0)
```

ME Region

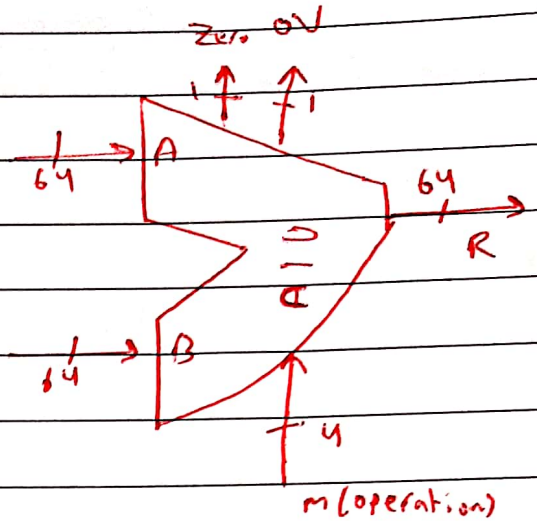
نبز بط هيك لانه هيك بنشيك اول اذا طلع مش ناجحة نلعن

Unlock ( );

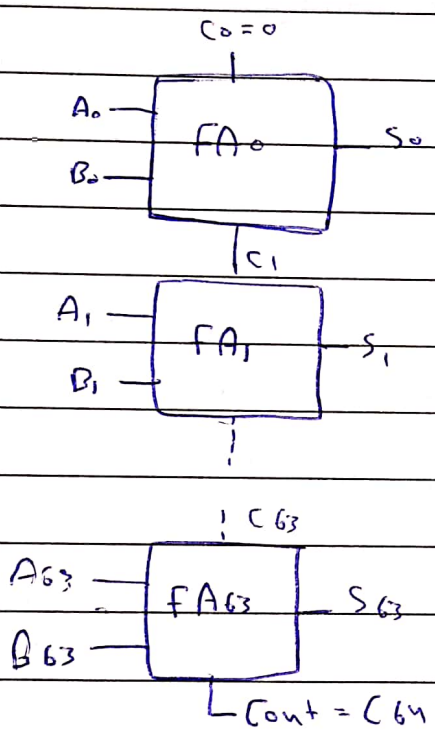
lock ضبطلل (0) هيك صار →

```
Sd  X0, 0(X20)
```

(3)



(7)



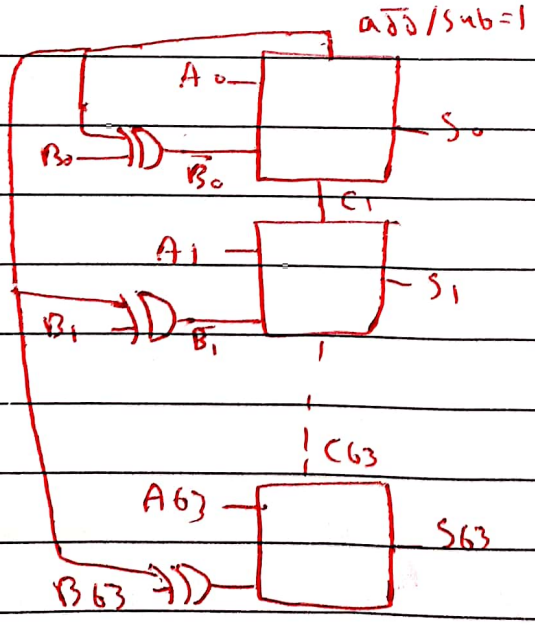
64- bit Ripple Carry adder

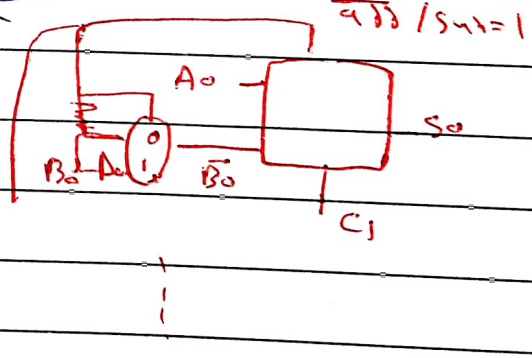
sub is 1 :1

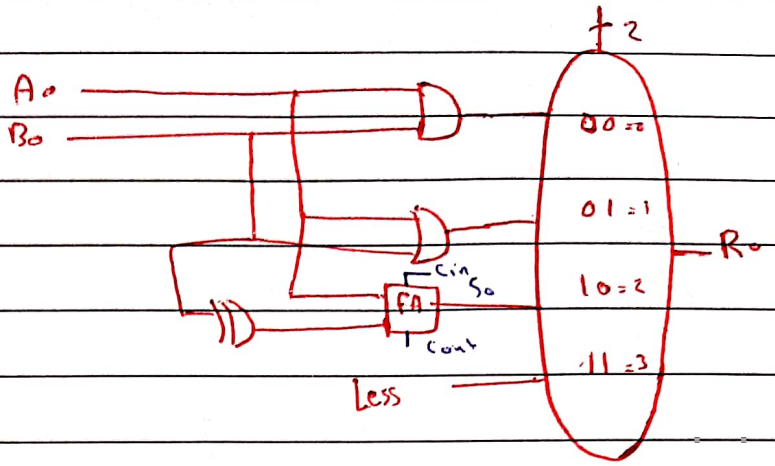
$A - B = A + (-B)$

$= A + (\bar{B} + 1)$

$= A + \bar{B} + 1$

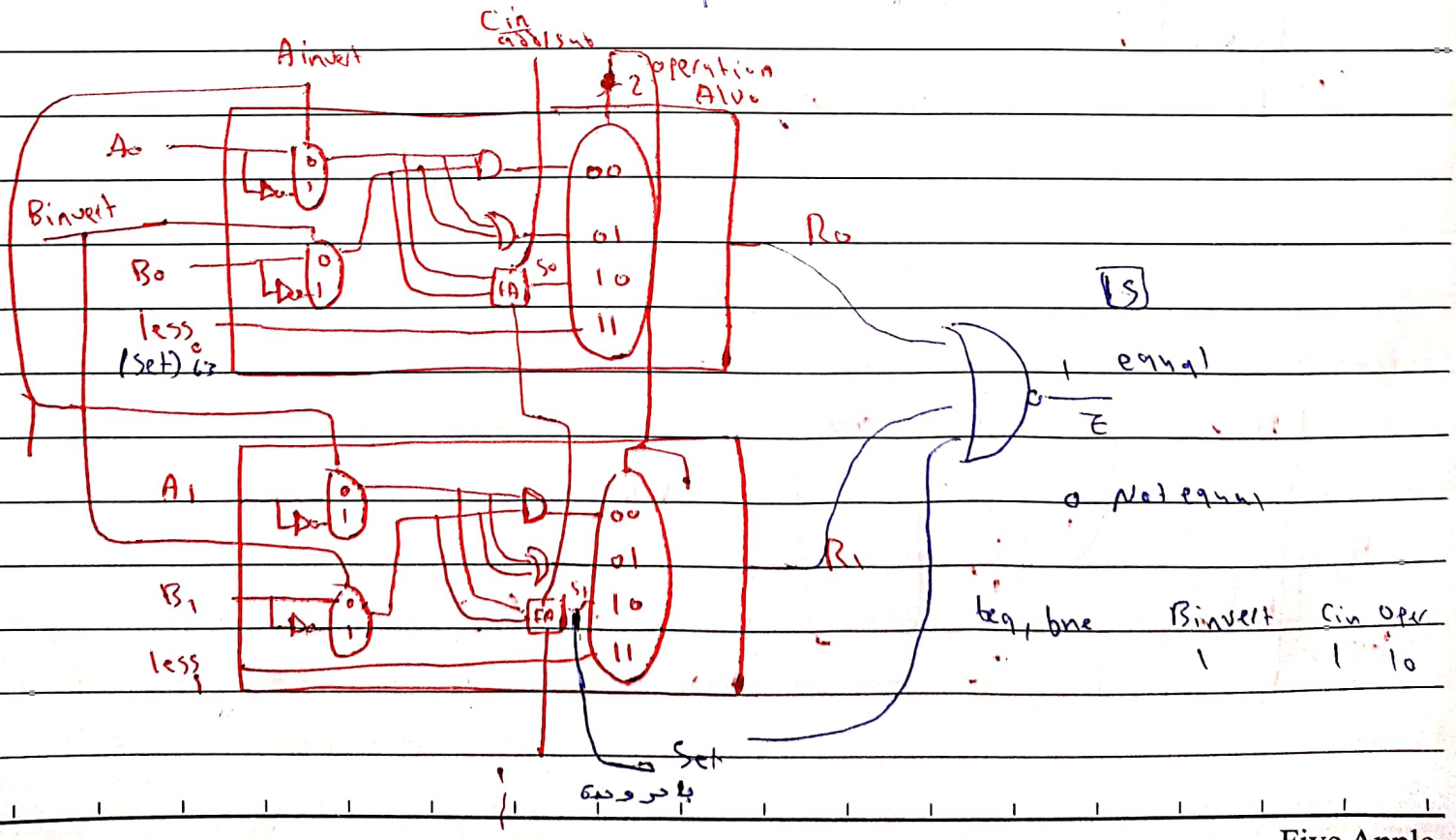






1-bit ALU Slice

Function	Ainvert	Binvert	Cin	operation
and	0	0	X	00
or	0	0	X	01
add	0	0	0	10
Sub	0	1	1	10
Slt				



Five Apple

(10) Set if less than

SLT rd, rs1, rs2

if rs1 < rs2 .

rd = 0x0000000001

else

rd = 0x0000000000

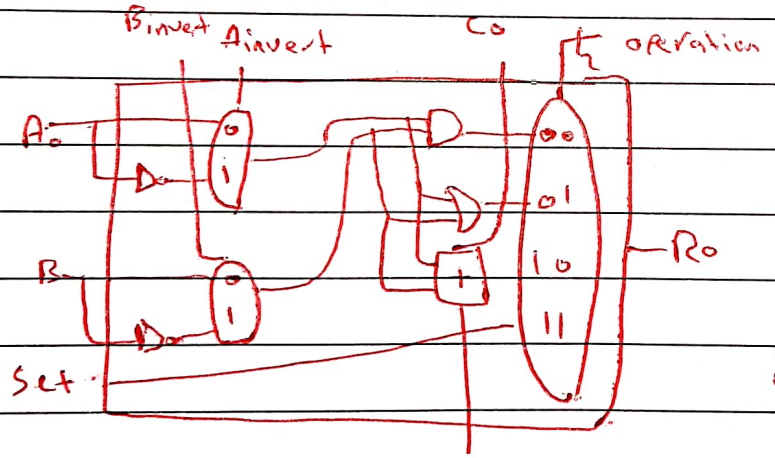
rs1 - rs2 < 0 (S63 = 1)

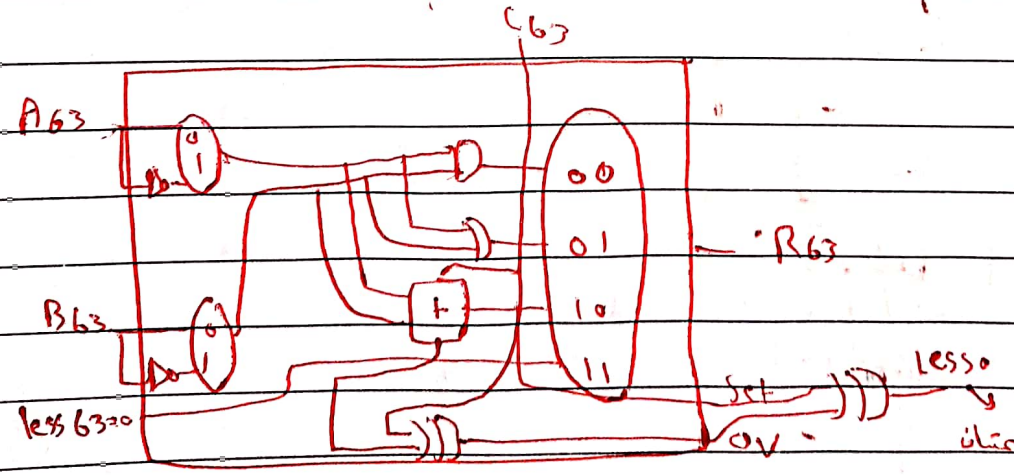
lesso = 1

rs1 - rs2 ≥ 0 (S63 = 0)

less0 = 0

كل ال less متيركين ع Zero ما عدا اول less0 ممكن Zero او 1





اذا صار مع بعض

اذا غير OV

$$\overline{A + B} = \bar{A} \cdot \bar{B}$$

$$\overline{A \cdot B} = \bar{A} + \bar{B}$$

$$A - B = A + \bar{B} + 1$$

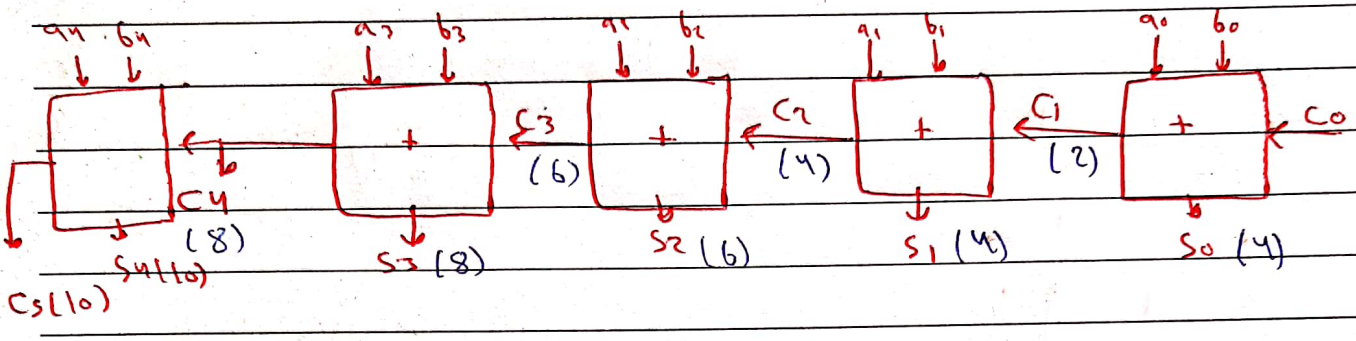
16)

set	ov	less o
0	0	0
0	1	1
1	0	1
1	1	6

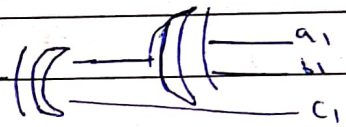
A-6 : Carry look a head adder

D = 1 gate delay DD = 2 gate delay

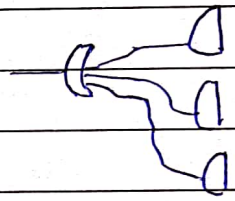
4-bit Ripple carry adder "RCA"



(4 gate delays) Si



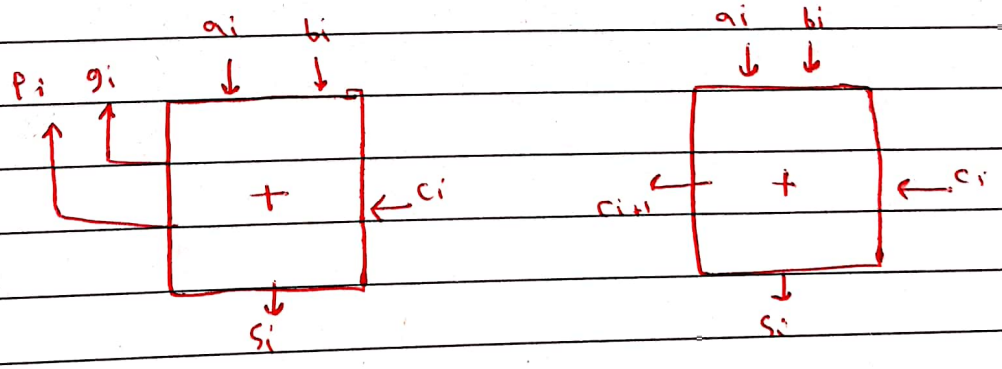
(2 gate delays) Ci+1



Delay of 4-bit RCA = 8 gate delays

" " 5-bit " = 10 gate delays

Delay of n-bit RCA = 2 x n gate delays



gi = Carry generate

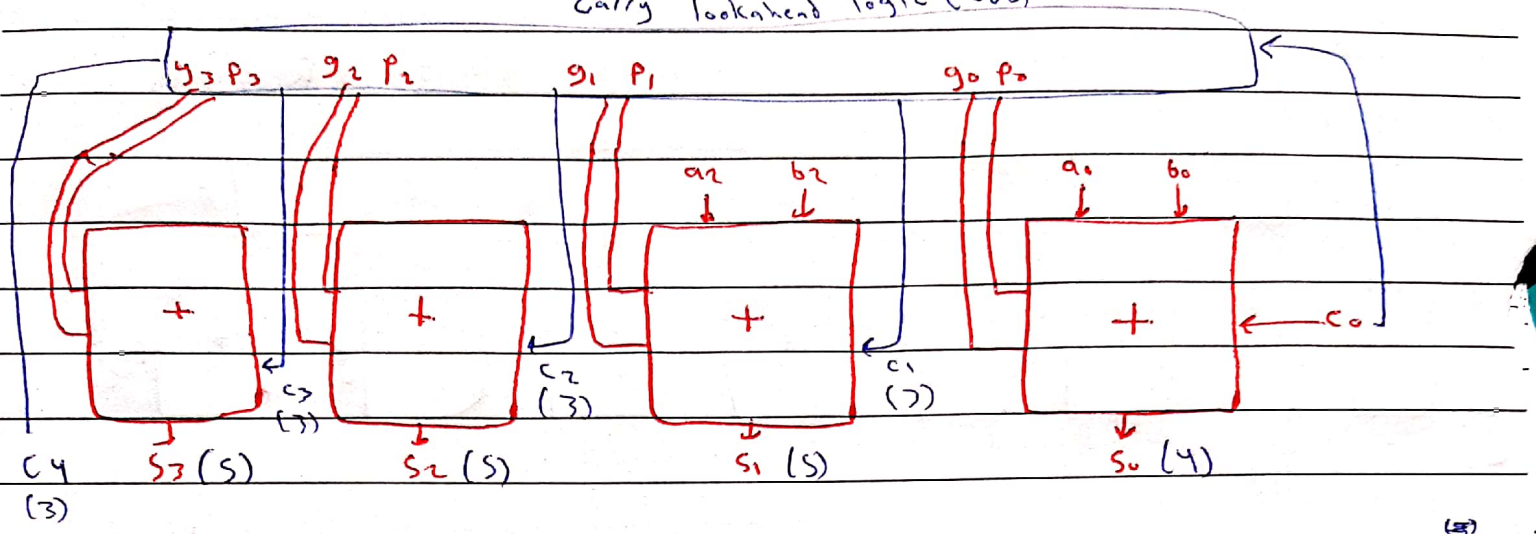
Pi = Carry propagate

gi = ai . bi

Pi = ai + bi

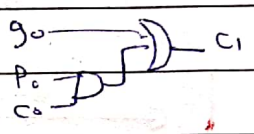
Five Apple

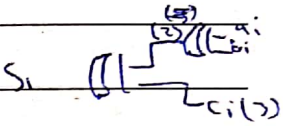
Carry lookahead logic (CLL)



$C_1 = g_0 + P_0 . C_0$

(3 gate delay)

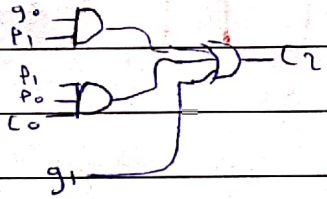




$C_2 = g_1 + P_1 C_1$

$C_2 = g_1 + g_0 P_1 + P_1 P_0 C_0$

(3 gate delays)



$C_3 = g_2 + g_1 P_2 + g_0 P_1 P_2 + P_2 P_1 P_0 C_0$

(3 gate delays)

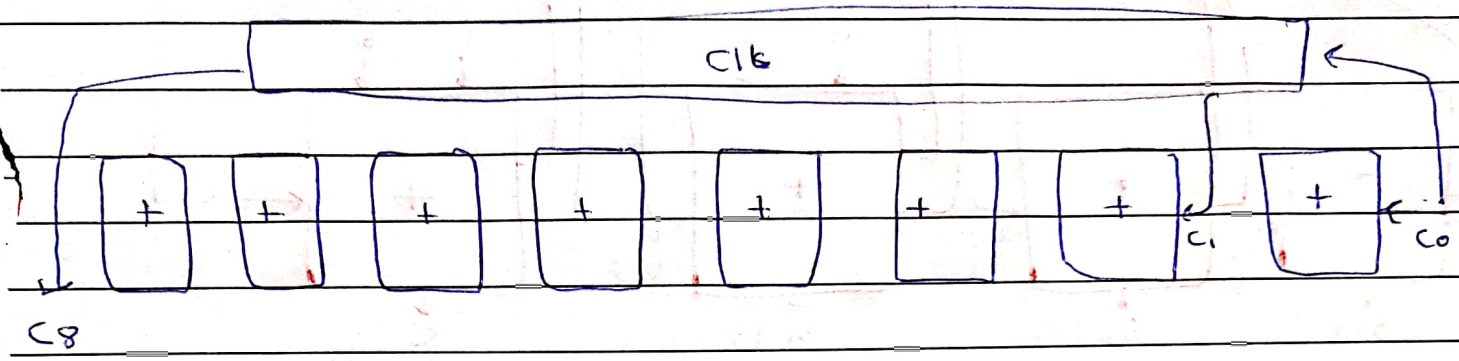
$C_4 = g_3 + g_2 P_3 + g_1 P_2 P_3 + g_0 P_1 P_2 P_3 + P_3 P_2 P_1 P_0 C_0$

(3 gate delays)

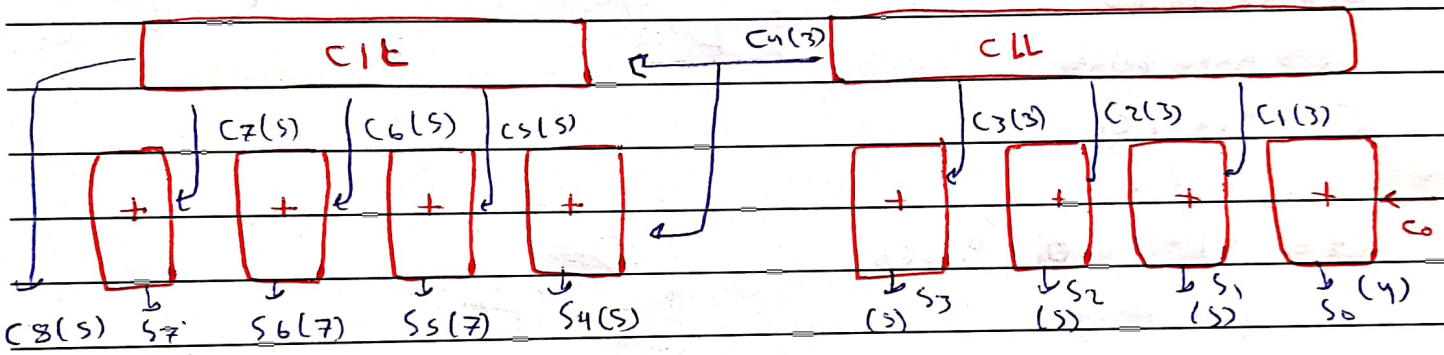
$$\frac{\text{Delay of 4-bit RCA}}{\text{Delay of 4-bit CLA}} = \frac{8}{5} = 1.6$$

4-bit CLA is 1.6 times faster than 4-bit RCA

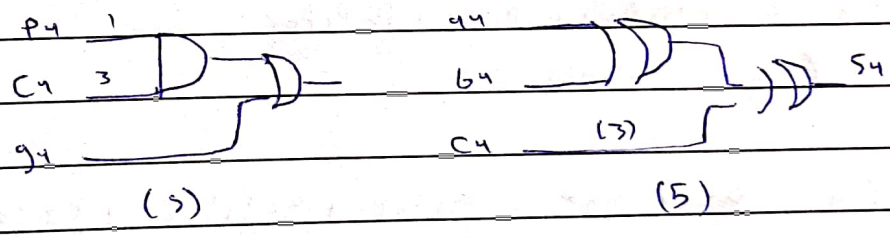
8-bit RCA = 16

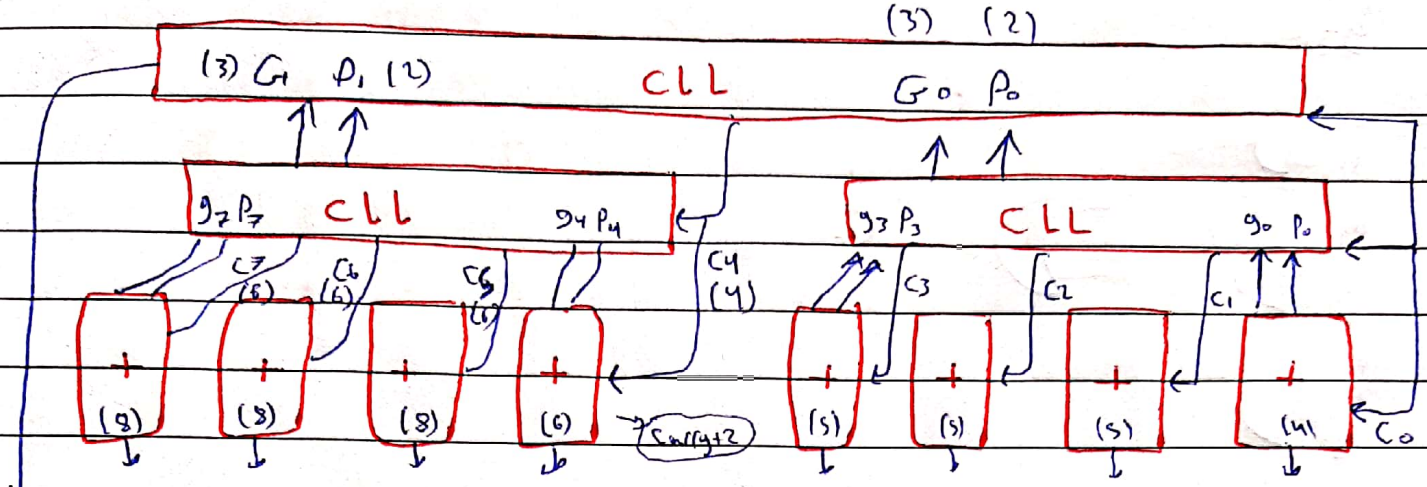


اكبر اشي الـ adder ← 5 inputs



$C_5 = g_4 + p_4 C_4$





C8

8-bit Carry lookahead adder

8-bit CLA

group generate :

$$G_0 = g_3 + p_3 g_2 + p_3 p_2 g_1 + p_3 p_2 p_1 g_0$$

$$P_0 = p_3 p_2 p_1 p_0$$

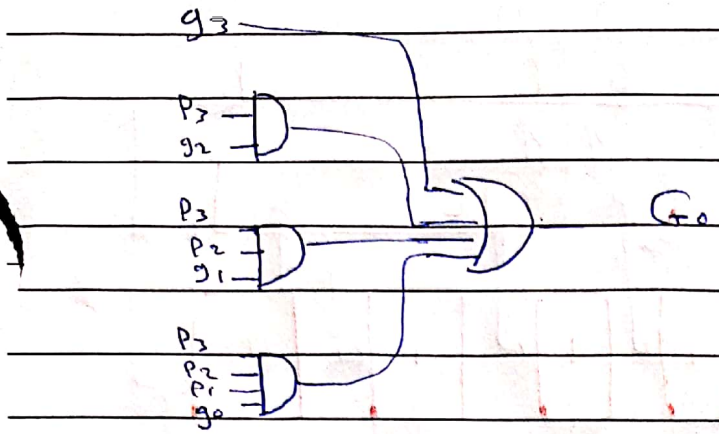
$$C_4 = G_0 + P_0 \cdot C_0$$

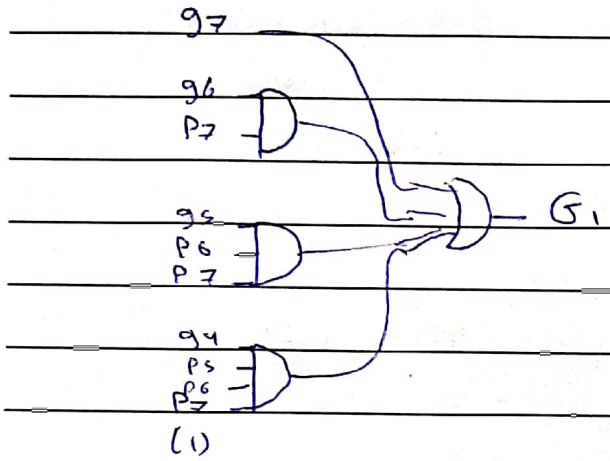
$$G_1 = g_7 + p_7 g_6 + p_7 p_6 g_5 + p_7 p_6 p_5 g_4$$

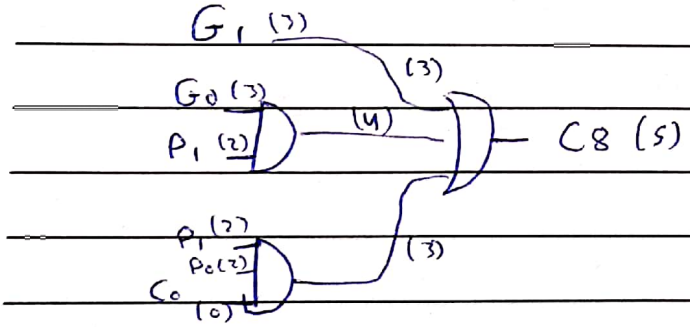
$$P_1 = p_7 p_6 p_5 p_4$$

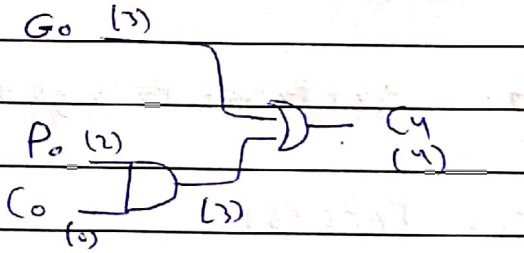
C_4

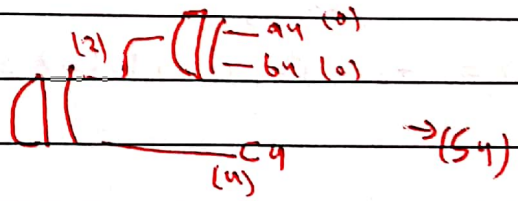
$$C_8 = G_1 + P_1 (G_0 + P_0 C_0) = G_1 + P_1 G_0 + P_1 P_0 C_0$$

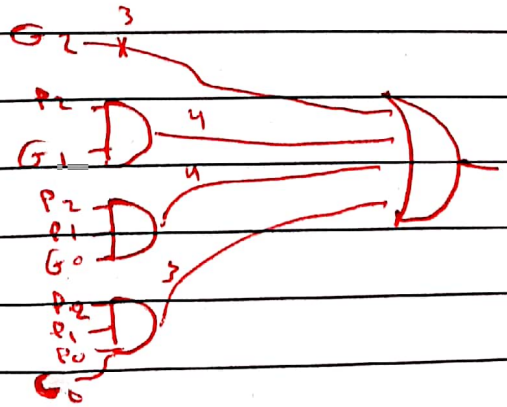












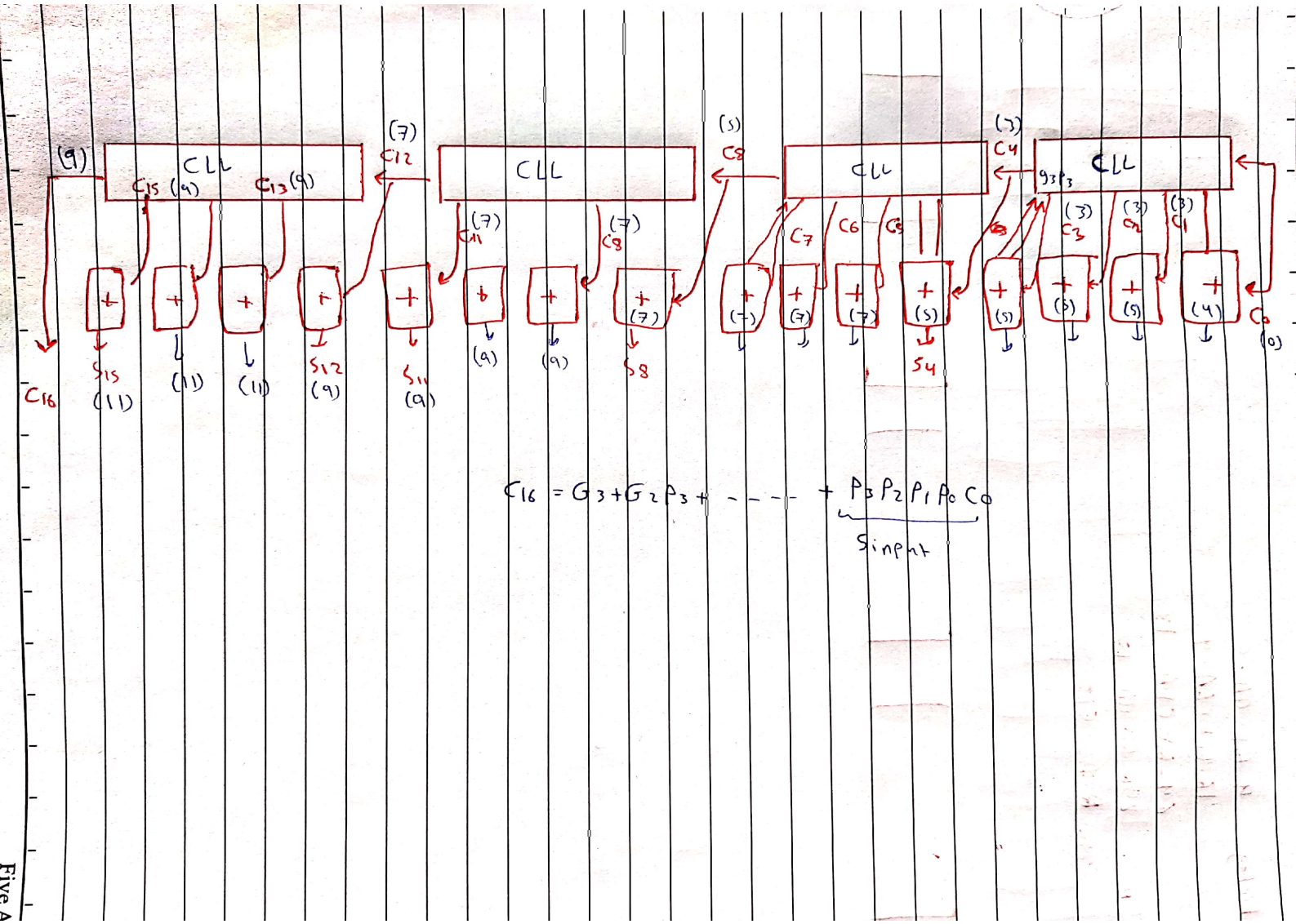
Delay 4-bit CLA = 5

" 16-bit CLA = 9

" 64-bit CLA = 13

" 256 bit CLA = 17

256 RCA = 512



Ch.4

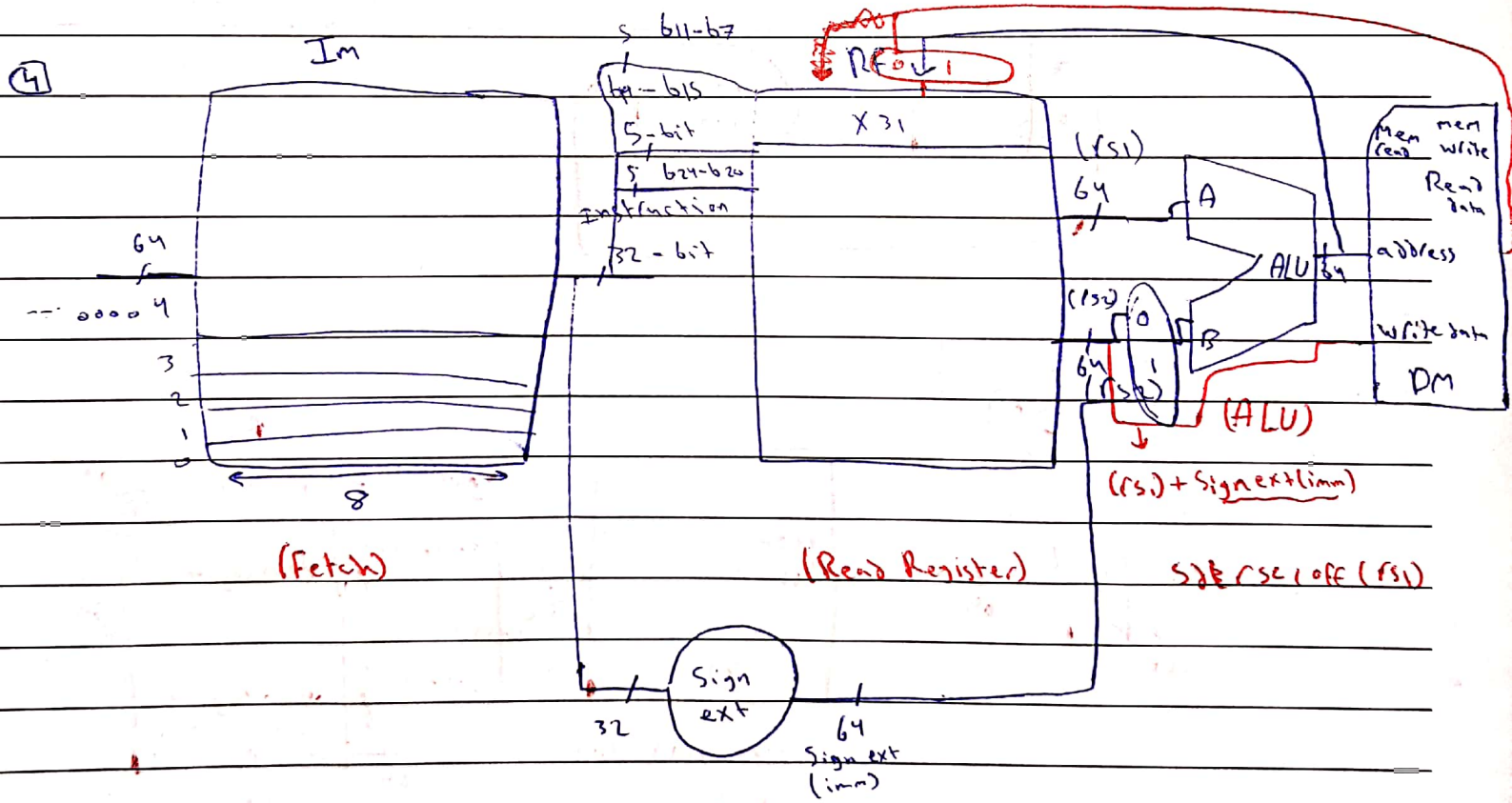
③ Memory address = (rs1) + Sign-Ext(imm)) ISA

(rs1) op (rs2) → R-type instruction (ALU)
op: ±, and, or

(rs1) - (rs2) = 0 } branch ← PC = PC+4
PC = target address
= PC + 2×imm

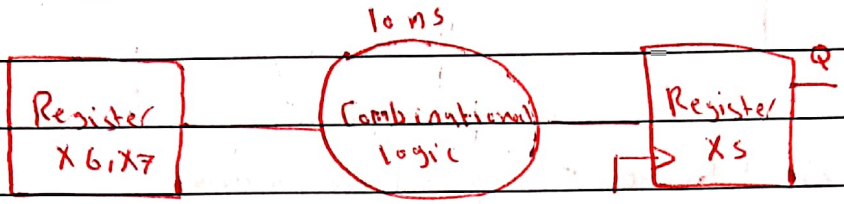
Access data memory — لا تمر بالـ R-type ال

④

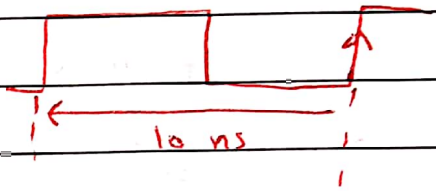


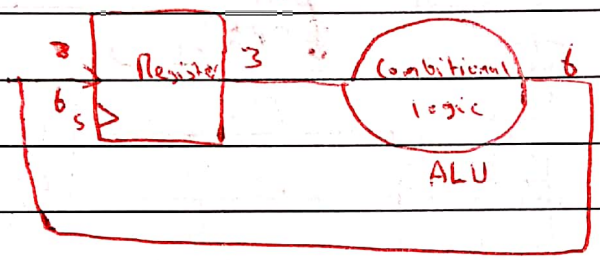
add x5, x6, x7

(x5) ← (x6) + (x7)



ALU





(13)