Computer organizationENCS2380

تلخيص محاضرات ابو السعود

Ch 1+ Ch 2+Ch 3 +Ch 11+Ch 12

* Computer Architecture: design مهتم بال
* Computer organization: مهتم بال تنفيذ والبناء

technology : organization يهتم بال implementation
programerer : Achitecture مهتم بال

Instruction → العمليات الي بعملها cpu

* The computer Revolution:

Moore's Law → الي اكتشف انه كل سنة تقريباً بصير نقلة بالكمبيوتر

server → مش للناس العاديين
interaction هاي بلزم يكون cpu أسرع ، نت أسرع ، وأقوى
ميموري أكبر
server → harddisk بقدر أخزن عليه أكثر

computer: →
1. general purpose variety of software
2. subject to cost / performance trade off

Server:
- Network based
- High capacity, preformance, reliability.
- Range from small servers to building sized.

* personal Mopile Device: (pMD)

- Battery operated. أهم اشي
- Connects to the internet
- Hundreds of dollars. السعر
- smart phone, tablets, electronic glasses.

كلها فيها prossesor, memory, I/o. operation

* cloud computing.

- ware house scale computers (WSC)
- software as service "Saas"
- portion of software run on PMD & a portion run in the cloud.
- Amazon and Google. شركات عندها سيرفر كبير

بجولك الـ access

حلت مشكلة انو تضيع بياناتك "مش محطوطة عند ضرب تلفونك"

بعدة احنا بنحفظها

* High level Language → compiler → Assemply.

Assemply → Assemblier → machine Language.

Machine Language هاي الي ما بفهم 0,1 كل اشي بالكومبيوتر

صوت → bit

صورة → بكسل [illegible]

parallel processing بشغلو cpu أكثر من
preformance بزيد in parallel

* understanding ~~pref~~ performance

1. Algorithm
"number of operation excuted"
2. programming Language, compiler, architecture.
"number of machine Instruction excuted per operation"

*3. processor & memory system
"how fast instruction are excuted"
4. I/o system "including os"
"how fast I/o operation are excuted"

* Eight great Ideas نفتح عدد أكبر من ترانزستورات الـ [illegible] محدودة
1. Design for Moore's law. التطور على الترانزستور
2. use abstraction to simplify design. ما بناخد التفاصيل بتعامل مع black box انه
3. Make the common case fast نعمل الجزء cpu
4. performance Via parallelsim [illegible]
5. " " pipelinding → operation بعمل [illegible] مع بعض
6. " " prediction. jump , Cash [illegible]
7. Hierarchy of memory.
8. Dependability Via redundancy.

$$\begin{tikzcd}
 & \text{program} \arrow[dl, dash] \arrow[d, dash] \arrow[dr, dash] & \\
\text{Application sofware} & \text{system software} & \text{Hardware}
\end{tikzcd}$$

Application sofware: write high level Language
system software
Hardware: cpu, memory I/o

System Software

Hardware

$\Updownarrow$ Human Architecture.

compiler $\rightarrow$ translates HLL $\rightarrow$ mechin Language.

operation system: service code.
- handling I/O
- Managing memory and storage.
- Schedulding of tasks, sharing resources.

* التطور أكثر في organization مع الهيكل ~~or~~ Comparable

programe Code $\rightarrow$
HLL $\rightarrow$ Assemply $\rightarrow$ Hard ware representation.
$\downarrow$
binary digits
Encoded instruction

assembler $\xrightarrow{\text{أسرع}}$ compiler

* Structure.

$$\begin{tikzcd} & \text{computer} \arrow[dl] \arrow[d] \arrow[dr] & \\ \text{I/O} & \text{memory} & \text{cpu} \end{tikzcd}$$

external $\rightarrow$ System Bus
عشان نربط بينهم
في

$$\begin{tikzcd} & \text{cpu} \arrow[dl] \arrow[d] \arrow[dr] & \\ \text{ALU} & \text{Registers} & \text{control units} \end{tikzcd}$$

operation
للتخزين مؤقت
Internet Bus
للربط بينهم
بتحكم بكل اشي

$$\begin{tikzcd} & \text{control units} \arrow[dl] \arrow[d] \arrow[dr] & \\ \text{control unit registor} & \text{sequance Logic} & \text{control memory} \end{tikzcd}$$

بتحكم بالوقت

moore's Law : تنبأ أنو بيقدر نضاعف عدد ترانزستور كل سنة وشهر ١٨ شهر ليغطي الحاجة المحددة

* Multicore Computer structure :-

- General processing unit
- Core. أكثر من cpu مع بعض
- processor. → أكثر من cpu

* cache memory "Mega
  code تخزين داتا و code data & code
  smaller & faster:
  cache السعر بتضاعف إذا زادت الـ cache لتنو غالي
  performance ~~ب~~ Cache بترتفع
  بس إذا زادت وصلت لحد معين ما بتزيد

* a safe place for data [illegible]

1. Volatile main memory.
   loses instruction and data when power off
2. Non Volatile secondary memory.
- magnetic disk بتحفظ الـ Data حتى لو power off
- flash memory.
- optical disk " CD Rom , DVD "

* Networks :

- Communication, resourcesharing, nonlocal
- Local area network (LAN) : Ethernet .
- wide " " (WAN) : the Internet .
- wire less network ; wifi , Blutooth

* History of computers.

first: Vacuum Tubes. G1 "machin Languag" ← G1

حجم كبير ، استخدم Vacuum ، بيوكل power كثير ، بطيء جداً"

stored program concept → أول كومبيوتر كان البرنامج ما يتخزن ، أنا بجيب البرنامج جوا الكومبيوتر بالمعمورة أو مكان وأنا بحكي للكومبيوتر ينفذها .

machine code.

G2 → smaller , cheaper , transistor , made from silicon less heat than vacuum tube control

more complex arithmetic & logic units & control units

use HLL

G3 → Integrated circuits. IC

أسرع ، أرخص ، أقل حجم

كان فيها عدد ترانزستور محدود فيها

G4 → IC → بصير فيها ترانزستور أعداد هائلة .

Gordon Moore: co-founder of Intel

observed number of transistors = that could be put in a single chip was doubling every year.

بصير Comp. آرخص.

- CISCs: complex instruction set computer. "Complex Hardware" "simple software" "Intel"
- RISC: reduced instruction set computer. " simple Hardware" "complex software" "Arm"

* micro controller → [illegible] cpu جهاز

Ram → مؤقتة [أسرع]
Rom → دائمة [بطيئة]

* Response time: technology بعتمد على
  How long it takes to do task: كم بدها وقت
* throughput: بدل ما نعتمد على "operation"
  . Total work done per unit time. كل الاشي [illegible] بوحدة زمن

* Replacing the processor with a faster version ?

* Adding more processors ?

CPU أكبر [illegible] تستخدم

* Relative performanse

Define performance = $\frac{1}{\text{ExcutionTime}}$.

"X is n time faster than y"

performance(x) / performance(y)

= Excution Time(y) / Excution time(x) = n

* Example

time taken to run a "program"

- 10 s on A, 15s on B
- Excution time B / Ecution time A =

$15 / 10 = 1.5$

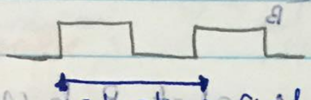
so A is 1.5 time faster than B.

* Measuring Excution Time.

- Elapsed Time:
  * Total response time, inclouding all aspects
  processing I/O, OS overhead, idle time.
  * Determines system performance. مبادلها أكبر؟ البرنامج حقه بيشتغل بسرعة
- Cpu time. cpu هو بس " العمليات "
  * time spent processing a given job
  Discounts I/O time, other job's shares
  * Comprises user Cpu time & system Cpu time

* different programs are affected differently by cpu & system performance.

أهم factor موجود Clock



clock period. → cycle

1 GH
2 GH] clock rate. "Hz"

* clock period : duration of a clock cycle "s"
* clock rate : cycles per second "Hz"
frequancy

$$\text{Cpu time} = \text{cpu clock cycles} * \text{clock cycle Time}$$
$$= \frac{\text{cpu clock cycles}}{\text{Clock rate}}$$

performance improved by.
1. Reducing number of clock cycles.
2. increasing clock Rate. بنزيدها تقل الوقت
3. Hardware designer must often trade off clock rate against cycle count.

عشان يختلف بين برنامج معين بدو 10s

* Example : Computer A : 2 GHz , 10s cpu time.
Designing computer B
. Aim for 6s. cpu time. 6s بدو نعمل البرنامج ليشتغل على
can do faster clock , but causes 1.2 * clock cycle
كدوهم أكبر ب 1.2
How fast must comp B Clock Be ?

$$\text{Clock Rate}_{(B)} = \frac{\text{clock cycle (B)}}{\text{cpu time}_B} = \frac{1.2 * \text{clock cycle(A)}}{6s}$$

$$\text{Clock cycle (A)} = \text{cpu time } \cancel{A} \cdot \text{clock Rate (A)}$$

$$= 10s \times 2\text{ GHz} = 20 \times 10^9$$

$$\frac{1.2 \times 20 \times 10^9}{6s} = \frac{24 \times 10^9}{6} = 4\text{ GHz}$$

لازم "B" يشغل على 4 GHz عشان تخلص البرنامج بـ "6s"

* clock cycle = Instruction count $\times$ cycle per Instruction

"التعليمات اللي بتحكي لـ cpu شو يعمل" بكل التفاصيل"

instruction → $\boxed{}\boxed{}\boxed{\text{safe}}\boxed{\text{safe}}$ وين تخزنها

"opcode" العمليات

clock cycle = عددهم × cycle قديش كل وحدة بدها تفسيرها

(s) cpu time = Instruction count $\times$ cpi $\times$ clock cycle time

$$= \frac{\text{instruction count} \times \text{cpi}}{\text{Clock Rate}}$$

* Instruction count for a program. الها علاقة بـ design

Determined by program, ISA, Compiler.

هاي أكثر ← RISC ← ISA ← Instruction count.

* Average cycles per Instruction.

Determined by cpu hardware. Risc → أسرع

if different instructions have different cpi

Average cpi affects by instruction mix

Clock cycle time "Rate" — technology هاد بيعتمد على
design مش على

* Example :-

computer A: cycle time = 250 ps , CPI = 2.0

computer B: cycle time = 500 ps , CPI = 1.2

Sam ISA → Hardware نفس → instruction يعني نفس عدد

which is faster, and how much?
↓
cpu time.

$$\text{cpu time}_{(A)} = \text{Instruction count} \times \text{cycle time}_A$$
$$= I \times 2.0 \times 250\ \text{ps}$$
$$= I \times 500\ \text{ps}$$ → A is faster

$$\text{cpu time}_{(B)} = I \times 1.2 \times 500 = I \times 600\ \text{ps}$$

$$\frac{\text{cputime}_B}{\text{cpu time}_A} = \frac{I \times 600}{I \times 500} = 1.2$$

A is faster than B by 1.2

cpu الأقل هو الأسرع

* If different instruction classes ~~have~~ take different numbers of cycles → $\text{clock cycle} = \sum_{i=1}^{n} \text{CPI}_i \times \text{instruction count}_i$

* Weighted average CPI

$$CPI = \frac{\text{clock cycle}}{\text{Instruction count}} = \sum_{i=1}^{n} \left[CPI_{(i)} \times \underbrace{\frac{\text{Instruction count}_{(i)}}{\text{instruction count}}}_{\text{Relative frequency}} \right]$$

* CPI Example :

Alternative compiled sequences using instructions in classes A, B, C.

	Class	A	B	C
clock عدد ال	CPI for class	1	2	3
instruction عدد ال	IC in seq. 1	2	1	2
	IC in seq. 2	4	1	1

- Seq 1 : IC = 5 "2+2+1" :

  clock cycle

  = 2×1 + 1×2 + 2×3

  = 10

  Avg. CPI = 10/5 = 2.

- Seq 2 : IC = 6 "4+1+1"

  clock cycles

  = 4×1 + 1×2 + 1×3 = 9

  Avg = 9/6 = 1.5

هاي أسرع
لانه أقل Avg

* performance. Summary.

$$\text{CPU Time} = \underset{\downarrow\ \text{design}}{\frac{\text{Instruction}}{\text{program}}} \times \underset{\downarrow\ \text{cpu}}{\frac{\text{clock cycle}}{\text{Instruction}}} \times \underset{\downarrow\ \text{technology}}{\frac{\text{Seconds}}{\text{clock cycle}}}$$

Performance depends on :

- Algorithm : affects Ic , possibly CPI
- programming language : affects Ic , CPI
- compiler : affects Ic , CPI
- Instruction set architecture : affects Ic , CPI , Tc

* power Trends. cpu ← آكثر اشي بيستهلك طاقة

* in CMOS IC technology.

$$\text{power} = \underset{\downarrow\ \times 30}{\text{Capacitive load}} \times \underset{\searrow\ 5V \to 1V}{\text{Voltage}^2} \times \underset{\searrow\ \times 1000}{\text{frequency}}$$

* Reducing power

- suppose a new cpu has
- 85 % of Capacitive load of old cpu
- 15% Voltage and 15% frequency reduction.

$$\frac{P_{new}}{P_{old}} = \frac{C_{old} \times 0.85 \, (V_{old} \times 0.85)^2 \times f_{old} \times 0.85}{C_{old} \times V_{old} \times f_{old}}$$

$$= 0.85^4$$

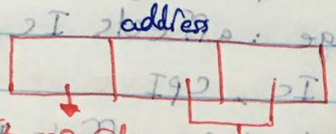
$$= 0.52$$

* The power wall

- we can't reduce voltage further.
- we can't remove more heat.

* Ch 12.

Characteristics and function



يعبر operation — operand
"data"

عدد ← op code. يعبر على

لو كان 16 oper
بدي 4 bits

Instruction.

36 instruction $\xrightarrow{\text{بدها}}$ 6 bits

* عدد n ← bits

oper = 2^n
↓
instruction

[cpu. ~~memory~~ memory 10 bit → 2^{10} → elements memory.]
Address

memory.

128 MB

Cell = 32 bits

قديش لازم address ÷ instruction.

$128 = 2^{27}$ $\qquad \frac{2^{27}}{2^2} = 2^{25}$ cells → memory
عددهم

32 bits = 4

address field بلزم 25 → bit

لو كان ؛ add = 18

2^{18} → cells.

memory كلها $2^{18} \times 2 = 2^{19}$ Bytes.

A+B → العكس

AB+ → post fix أسهل وحدة

+AB → prefix

* Types of operands: بيميز

حسب تعريف ال 10 اذا كان

address - Characters - Number - Logical Data

هل هي 10 ؟ 2 , add , Constant , Char

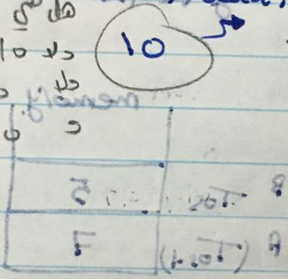
ASCI Code

address 10 ولا

ولا ascI 10

باينري 2 = 10

لازم نحدد



* Single - Instruction - Multiple - Data

"SIMD" Data types

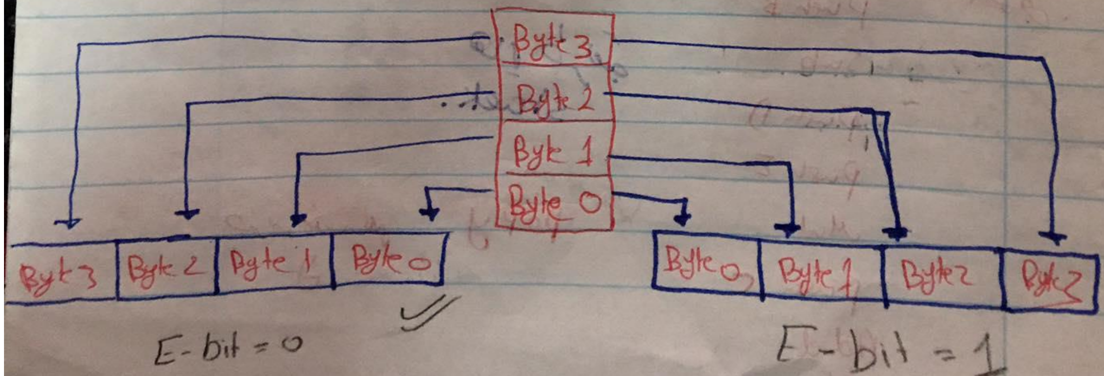
بس 1 instruction

* Arm

8 byte

16 halfword

32 word



Byte
↑
STRB R0 ,[100] خزنلي B ↑ Byte أول من R0 على
STR R0 , [100] memory address 100
خزنلي كل ال Register على
Add 100

move → 1 R → R , Constant → R
load → memory → R

move R0,1 خط واحد جوا
moiv R1 ,0 حط 0
Sub R0, R1, R0 R1 - R0
BZ xyz jump if Zflag = 1 "brach" على xyz

0 - 1 = -1
jump ما بجي Zflag = 0

skip [⋮]

بروح xyz str R0,[100]

*Data Transfer :-
32 bit = 4 Byte
1. Source من
2. destination لوين
3. Data كمية

LDR R0 , [100]
بنقل شوكي الـ [100] إلي جوا ما بنحفظها بتعل

strb r0, [100]
byte.

LDRH R0 , [100] ⇒ 16 bit "$\frac{1}{2}$High"
LDRL R0, [100] ⇒ 16 bit "$\frac{1}{2}$ Low"

$$\begin{tikzcd}
 & \text{I/o} \\
\text{data filed } \underline{2} \text{ in instruction} \arrow[ur] \arrow[r] \arrow[dr] \arrow[ddr] & \text{memory} \\
 & \text{register} \\
 & \text{constant}
\end{tikzcd}$$

4 → register $\xrightarrow{\text{بلزم}}$ 2 bit

32 register $\xrightarrow{\text{بلزم}}$ 5 bits

risc → عدد الـ register اكبر → عدد bits

عدد register = 2^n

24 register $\xrightarrow{\text{بلزم}}$ 5 bits

register << memory من حيث عدد bits

constant لا يكون

5 bits → 0 → 31 un signed.

0 → $2^n - 1$

5 bits → -2^4 → $2^4 - 1$ signed.

-2^{n-1} → $2^{n-1} - 1$

عدد bits هو ← يحدد range الأرقام.

+12 = 01100

−12 = 10100 2's complement.

− 512 → 511 → 10 bits بلزم

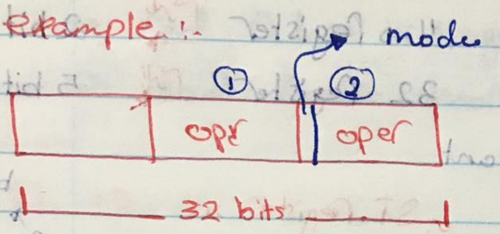
→ -2^9 ⇒ 9 = n−1

n = 10.

Instruction → | op Code | operande Refrence | operand Refrenc |
|---|---|---|
| 4 bits | 6 bits | 6 bits |

16 bits

Example :-



عشان اعرف memory ولا constant

① → only register

② → memory or ~~register~~ Constant.

1 bit يعني بدي

* support 48 operation

* 24 register

op Code => $\# \text{ of operation} = 2^n$

$48 = 2^6$

$n = 6 \text{ bits}$

① → register => $\# \text{ of register} = 2^n$

$24 = 2^5$

$n = 5 \text{ bits}$

* mode = 1 bit

② $= 32 - (6+5+1)$

$= 20 \text{ bits}$

memory حجم

$\# \text{ of Cells} = 2^{20} = 1MB$

memory حجم $= 2^{20} \times = 1MB$

$\frac{2^{27}}{2^{1}} = 2^{\boxed{26}}$ ↷ # of bits in filed.

128 MB
Cell = 16 bits

* Capacity of memory : 128 MB = 128×2^{20}
$= 2^{7} \times 2^{20}$
$= 2^{27}$ Byte

* Cell = 16 bits = 2 Byte
1 Byte = 8 bits

$\frac{2^{27} \text{ Byte}}{2 \text{ Byte}} = 2^{26}$ cell

# of address Bus = 26 bits

$2^{\#\text{ of address bus}}$ = number of cell in memory = length of MAR

* Instruction Representation.
1. opcodes are representation by abbreviations. called mnemonics

* Instruction types :
1. Data processing. "% / - + *" العمليات
2. Data storage. تخزين
3. Control → تحكم "if statement"
4. Data movement. نقل من مكان لمكان

① Three - address instruction

SUB y, A, B يجمع A+B وحطها
ADD y, A, B في distenation y
DIV y, A, B y ← A+B
MPY y, A, B النتيجة

② Two address Instruction

SUB A, B اجمع B,A وحط الجواب في A
MOV A, B A ← B → move B in A.
ADD A, B

③ one address instruction
ADD C

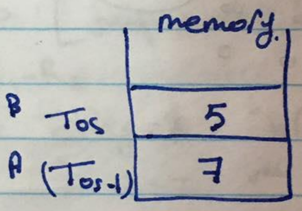
Load D	انقل من memory لـ register ac	+ Add
stor y		− Sub
SuB B →	هون عنا اشي implecit مخزن هوا بنقص من B وبخزن فيها "ac"	÷ div
DiV y.		⟷ Multp
inc C	1 يزيد increment.	

Stor y → y ~~R~~ Ac

نقل من Ac على y

* $y = \dfrac{A-B}{C+(D \leftrightarrow E)}$

1. SuB ~~y~~, A, B
2. ~~MPY~~ T, D, E
3. ADD K, C, T
4. DiV ~~y~~, y, K



ADD , MVP ✓
SuB , div
(Tos-1) op Tos
B − A

A B +   push A , push B , add.
+ A B

للعكس الي فوق *

push A

Push B

Sub ← بطرح أول تنين

Push D

Push E

Mul

push C

Add

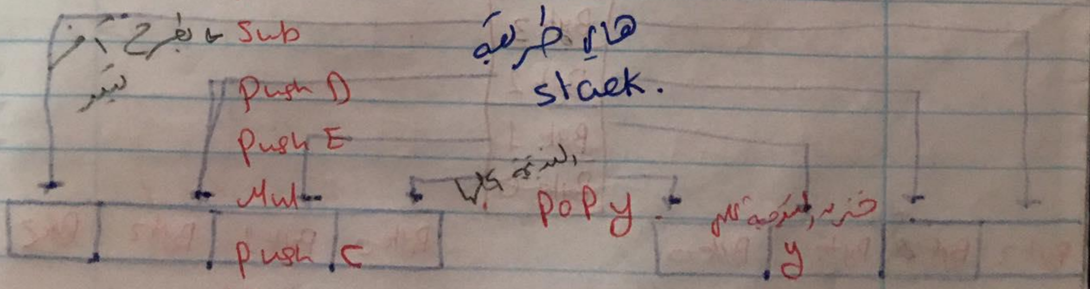
→ y = AB− CDE * + /

هاي طريقة stack.

النتيجة

PoP y

خزن النتيجة على y



A+B → العكس

AB+ → post fix أسهل وعدد

+AB → prefix

* Types of operands: حسب ترتيبهم او ١٥ اوال

address - Characters - Number - Logical Data

؟ ١٠ هل هي 2 , add, Constant. , Char → ASCI Code

address 10 دلد 10 → ??

ascI 10 دلد memory

2 باينري = 10

* Single - Instruction - Multiple - Data

"SIMD" Data types

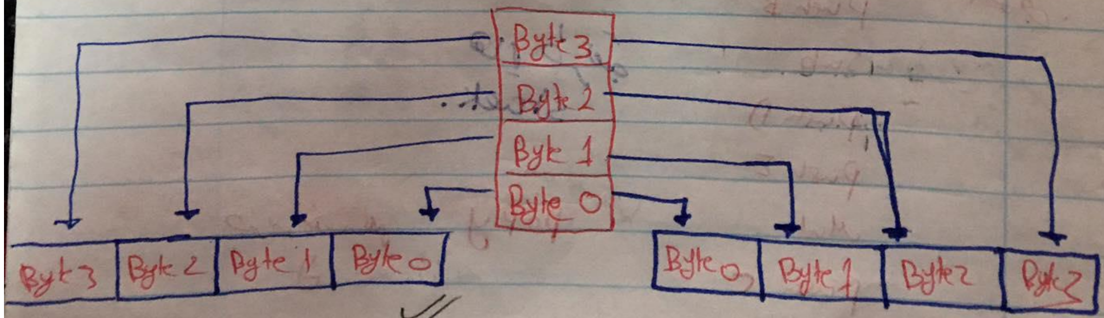
بس 1 instruction

* Arm

8 byte

16 halfword

32 word



E-bit = 0 E-bit = 1

Byte
↑
STRB R0, [100] — Byte أول ↑ خزنلي B من R0 على memory address 100
STR R0, [100]
↓
خزنلي كل ال Register على
Add 100

move → 1 R → R , Constant → R
load → memory → R

move R0, 1 حط واحد جوا
mov R1, 0 حط 0
Sub R0, R1, R0 $R_1 - R_0$
BZ xyz jump if Zflag = 1 "branch" على xyz

[0 − 1 = −1
jump ما بعمل Zflag = 0]

skip [⋮ ⋮]

→ xyz str R0, [100]

*Data Transfer:-

32 bit = 4 Byte
↑
LDR R0, [100]

1. Source من
2. destination لوين
3. Data كمية

بنقل الي جوا ال [100] ما بنحطها بتعل

strb r0, [100]
↓
byte.

LDRH R0, [100] ⇒ 16 bit "½ High"
LDRL R0, [100] ⇒ 16 bit "½ Low"

القراءة من memory و mov مابترجع الداتا بتعمل Copy
بترجع stack

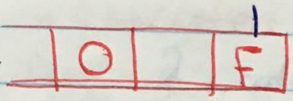
* un signed ← المهم Carry

* signed ← المهم overflow

```
 1001      9
 1110 +   14 +
------    ---
10111     23
```

Carry ← 1

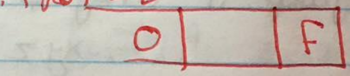
Carry flag = 1
يعني في مشكلة
over flow



بنحطها جوا
Carry flag.

* اذا صار Carry يعني في over flow ←
un signed.

* signed.



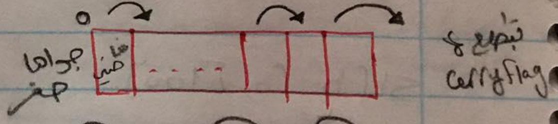
```
 1001      -7  0111  2's c
 1110 +    -2  0010 + 2's c
------
 0111
```

هاي صفر والجواب لازم 9- يعني 1
يعني over flow = 1

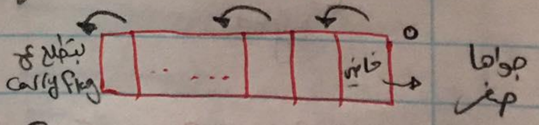
الجواب لازم
-9

* اذا جمعت رقمين (-) والجواب طلع (+) يعني في over flow = 1
* // // // (+) // // (-) // // //
* غيرهم فش over flow

* shift Right ↷



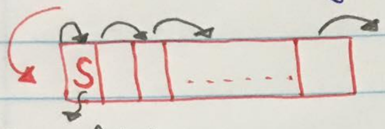
* shift left ↶



Logical تحدد 3
آخر bit طلع ← Carry

* logical → un signed
* Arithmatic → signed] liner shift.

قسمة



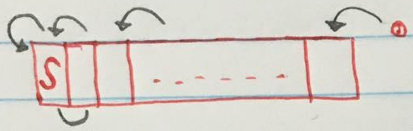
Arithmatic Right shift.

هون بضيف نفس قيمة S

Signed bit

آهم bit ممنوع أتحرك

ضرب



Arithmatic left shift

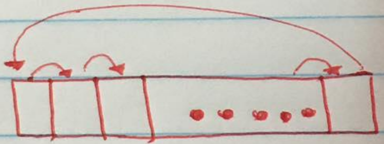
بضيف 0 عادي ما بأثر على S

هدول صفرات يصير overflow

بالضرب الرقم بكبر

القسمة بقل الرقم ومستحيل يصير overflow وهون تقريب

$\frac{5}{2} = 2.5 \simeq 2$ أكبر عدد صحيح أقل من الرقم



Rotate.

12 → 0001 0010 BCD]
0000 1100] → Conversion.

* Transfer of Control :-

jump → Conditional.
jump → un Conditional. true دائماً

* Examples of shift and Rotate operation-

10100110 $\boxed{\text{Logical right shift 3 bit}}$ 00010100

3 bit بنلف عليهم

carry = 1 → register أخر bit طلع من

10100110 $\boxed{\text{Arithmatic left shift 3bit}}$ 1011 0000

10100110 → 10110000

هاي بتشغل خانة ممنوع أغيرها

بنحذفو

اليسار بنعبيهم

1010011 0 $\boxed{\text{Arithamatic right shift 3bit}}$ 11110100

10100$\boxed{110}$ حذف

هاي لازم تظل 1 عشان هيك بنضيف 1 مش 0

11110100

* Conversion :

ex: convert from decimal to binary.

بنستخدم BCD

12	8 bit	0000 1100	Hex
12	8 bit	0001 0010	BCd

BCD → 4 bits = one digit كل

* input && out put

بطريقة كثير أكثر من memory "مابدنا cpu يتعامل معها direct"

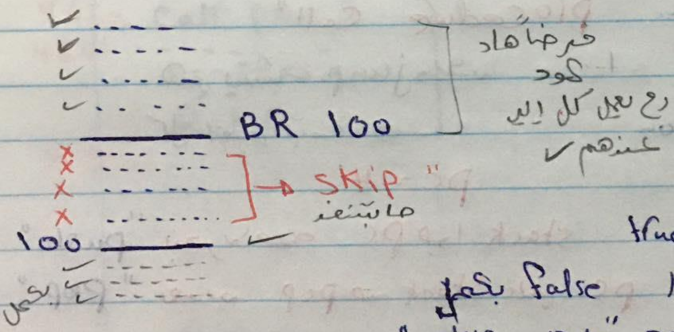
* Transfer of Control : الأصل أنه يتنفذ الكود sequential

يعني سطر سطر حسب الترتيب ويجي مرة instruction ممكن تغير الاشي

ex: jump , Branch "نفس الاشي "العكس

un conditional → true دائماً

Conditinal → لازم نفحص



* un Conditional هيك بتشتغل

* Conditional

لازم أفحص الشرط إذا true بعمل زي un conditional إذا false بعمل "skip ما بعمل" sequential

* Conditions → مبنية على flag

cf, zf, of, sf

هل في carry إذا آه cf = 1

كل النتيجة 0 zf = 1

over flow

اذا النتيجة (-) SF = 1

اذا (+) sf = 0

BR Z 200 إذا Fz=1 آعمل Branch على Address 200

إذا Fz=0 بعمل Code بالترتيب "Seq"

j N Z 300 إذا Fz=0 اعمل jump

* بالعودة إلى قبل PC = 6 "بتأشر على next instruction"

بس لأنه في BR 100 رح يأشر على قيمة PC

"PC = PC + 100"

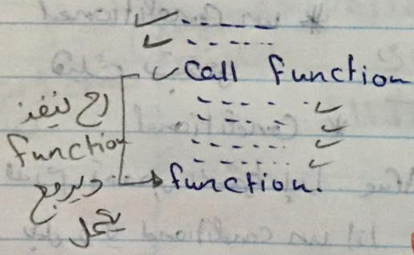
PC = PC + displacement.

PC = address

* Call function. "procedure Call"

هي بتشبه jump وبعد

كل اشي بتنفذو



PC = 41

"push" بنحط قيمة PC جوا stack

"pop" بعدين pop من stack على PC

* BRE R1, R2, 235

if R_1 equal R_2 do "BR 235"

* Call procedure"

داخل

OP	[Memory]

pc $\curvearrowright$ push on stack "الي بعد Call مباشرة"

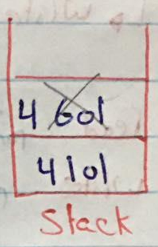
pc = [memory]

ret → بنعمل pop stack

pc = pc الأولية

```
4000 ------ ✓
     ------ ✓
     ------ ✓
4100 Call proc1  → pc = 4101
4101 ------        pc = 4500
     ------
     ------
4500 ------     ┐→ pc = 4501
     ------     │
4600 Call proc2 │→ pc = 4601   "push on stack"
4601 ------     │proc1 pc = 4800
4650 Call proc 2│ pc -
4651 ------     │
     Return     ┘
4800 ------     ┐
     ------     │
     ------     │ proc2
     ------     │
     ------     │
     Return.    ┘  → pop pc  "pc = 4601"
```

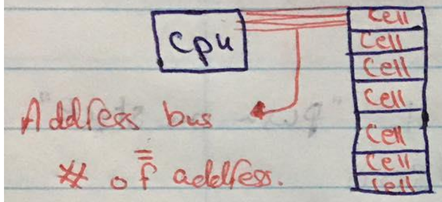
push



X 86 → cpu "نُسخ" مش مهم كتير ☺

* ch 3 :

memory	→ read	تقرأ
	→ write	تكتب
I/o	→ read	ناخذ قيم منهم
	→ write	نطبع عليهم



cells → نفس الlength

Address bus : بكون مكتوب عليها ال address

Control bus : مكتوب عليها شو بعمل cpu r, w
read + write

Data bus : بكون عليها الداتا سواء من
memory أو من cpu

mov [R0], R1 حط قيمة R1 جوا Add جوا
memory.

* MAR : memory address register.
عبارة عن reg وحدة [illegible] مشبوكة مع
address bus

* أي اشي بدي أنقله من reg لـ memory لازم أنقلها بالأول على MAR وهاد بتعتبر مشبوكة مع address bus.

8 line. "~~data~~ address buss" = 2^8 cell = حجم الميموري

$2^{10} = K$

$2^{20} = M$

$2^{30} = G$

$2^{40} = T$

* MBR : memory buffer register. مشبوكة مع Data Bus

Data Bus = cell length.

* I/O AR : I/O address register

هاي مشبوكة مع address bus I/O register

* I/O BR : I/O Buffer register

هاي مشبوكة مع Data bus I/O register

* PC = program Counter فيها add تبع next instruction

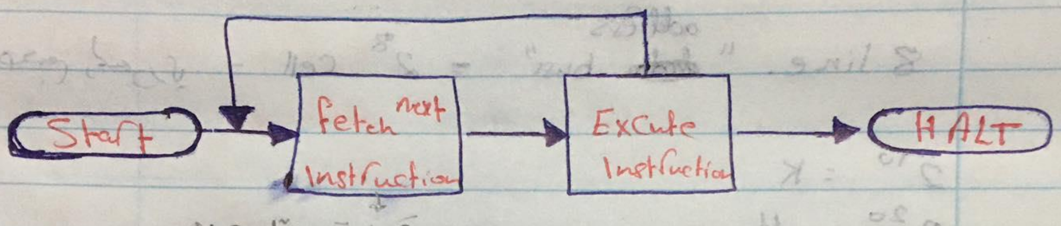
* IR = Instruction Register

$$\begin{tikzcd}
PC \arrow[r, bend left, "\text{قيمتها}"] & MAR
\end{tikzcd}$$
← هنا بأخذ رقم الـ add الي أنا بدي أشوفه memory

$$\begin{tikzcd}
IR & MBR \arrow[l, bend left]
\end{tikzcd}$$
← من memory برجع الداتا من Bus

* pc : auto increment ↶ note.

Fetch cycle   Excute cycle

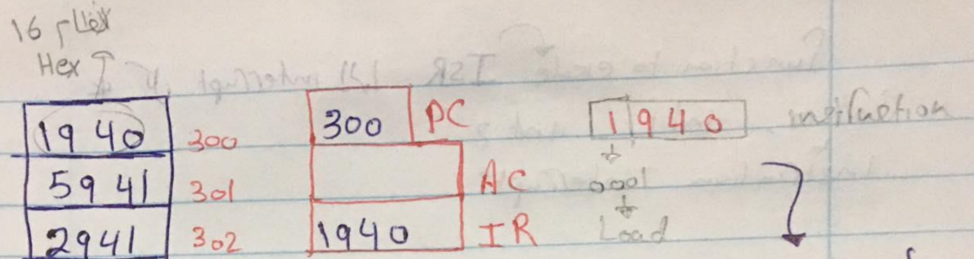


قراءة داتا وهكي

* action Categories "operation 6 أنواع"

- processor-memory : Data transfered from processor to memory or from memory to processor
  - Load "memory to reg" • Store "reg to memory"
- Data processing : operation "+ - * / %" and, or ...
- Control : jump or Branch , Call function
- processor - I/o

* Example.
  - AC → "one register"
  - add [100], [200]   AC خزنه التخزين
  - add [10]
  - اي o.p مش موجود مكانو بكون "AC"



اقرأ data من [940] و

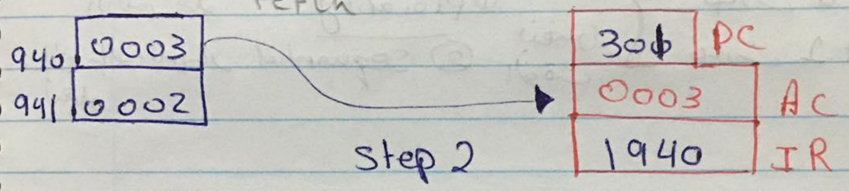
حط القيمه ب Ac

$PC \rightarrow MAR$

$IR \leftarrow MBR$

step 1

Fetch



Step 2

* لازم آعمل increment لل PC

* Slide $\underline{\underline{12}}$ مهم

* Interrupts : مقاطعة

• polling : بنترك cpu يشتغل 5ms وبعدين يعمل check على I/o

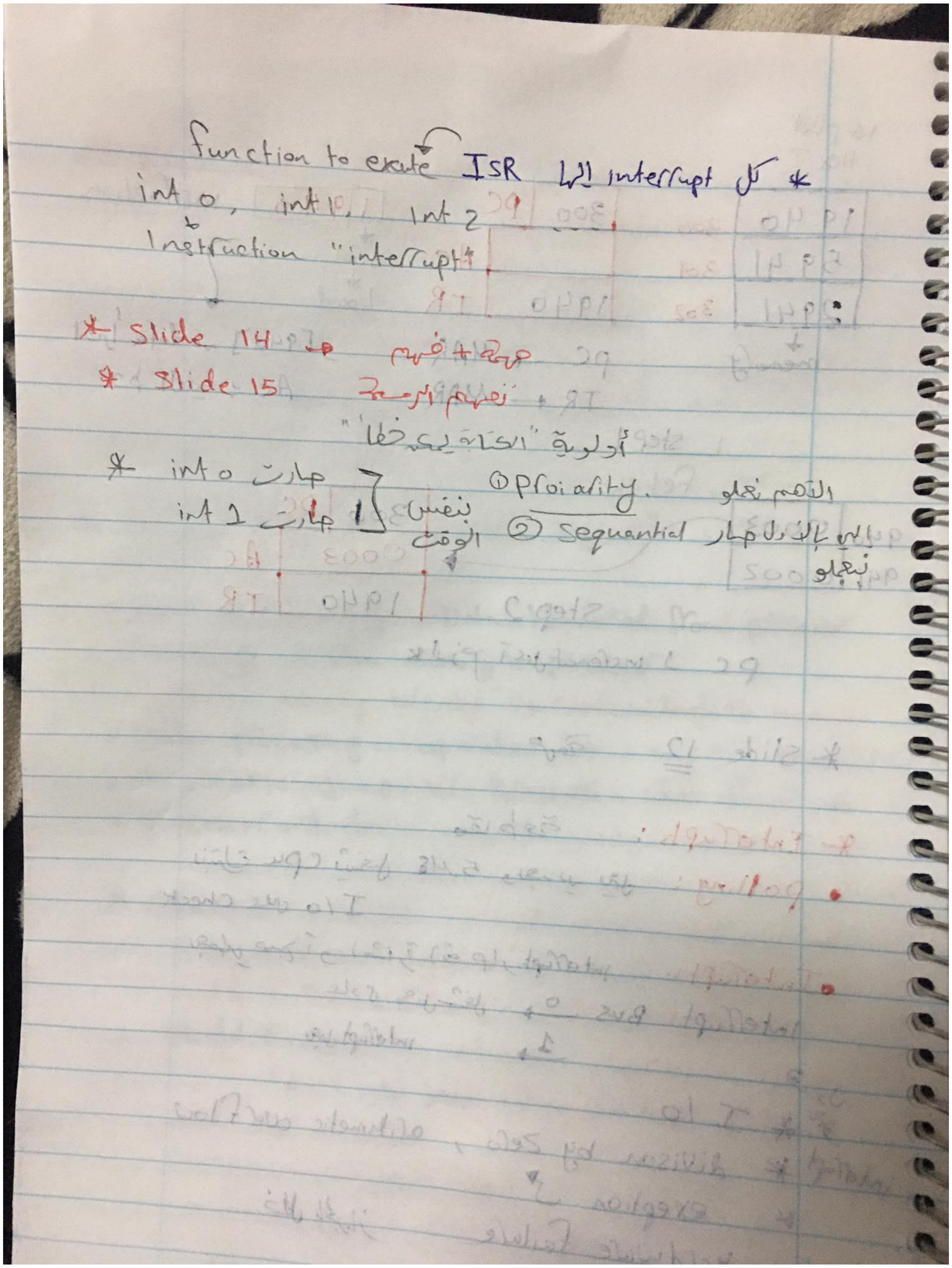
• Interrupt : بعطي صح آو اشارة انه صار interrupt

Interrupt Bus $\xrightarrow{0}$ عادي بعمل شغل

$\xrightarrow{1}$ بعمل interrupt

هدول بعملو interrupt

* I/o
* divison by zero , arithmetic overflow
* exeption ↺
* Hardware failure خلل بالجهاز
* Timer



* كل الـ interrupt إلها ISR → function to exute

int 0, int 1, int 2 ...

Instruction "interrupt"

* Slide 14 مهم مهمة + فهم

* Slide 15 نفهم الرسمة

أولوية "الكتابة فيها خطأ"

* int 0 صارت
int 1 صارت
] بنفس الوقت

① proiarity. الأهم نعملو

② sequential الي بالأول صار نعملو

* كل interrupt إلها ISR
function to excute

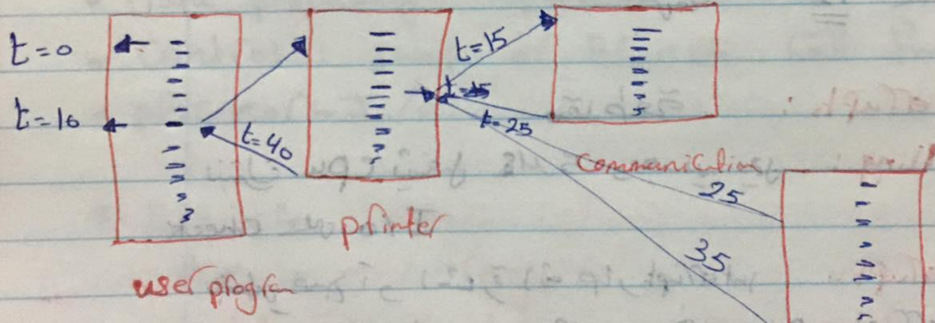
int 0 , int 1 , int 2
Instruction "interrupt"

* Slide 14 مهمة فهم
* Slide 15 تعميم الرسم
"أولوية "ا [illegible]

* int 0 جات }
int 1 جات } بنفس الوقت

① priority . التهم نعملو
② sequential اللي بالأول جاي نعملو

* slide 16 : مهم للعلم
* 17 - 20 "slide" فهم
* Silde 21 مهم



interrupt نعلو printer , Disk , Comminication
كل وحدة فيهم بدها 10 see عشان كل instruction → proiority : printer = 1
Communication = 2
Disk = 3

Disk > Comminication > printer من الأول

① البرنامج "user program" كان يشتغل لحد 10 sec

عند 10 sec صار "printer interrupt" لحد 15 sec صار

"communication interrupt" يعني printer اشتغلت بس 5sec بنجيب نقارن

مين إلو prioiorty أكبر "communication" بروح نخلص نشغلها بعدها

10 sec ومين أولاً" وقفت عند 15 بنصير 15 + 10 = 25

عند t = 25 صار Disk interrupt بعدها 10 ثواني بخلص وبرجع

25+10 = 35 بيرجع عند printer / بيضل بعدها 5sec ثواني

لحد t = 40 بيرجع عند user program

* لو صنفنا نفس الـ prioirty whith out " Sequantial "

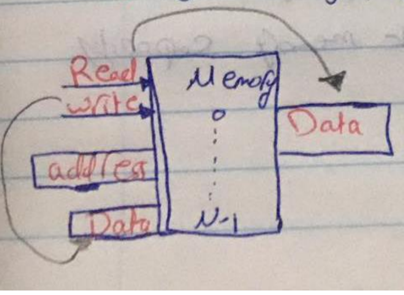
printer → start : 10
→ End : 20

Communication → start : 20
→ End : 30

Disk → start : 30
→ End : 40

* I/o function

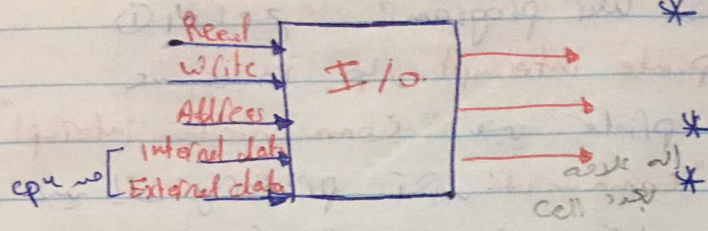
* كل I/O في إلها address منها بنقرأ الـ data



address → cell address

read → بنطلع منها الـ Data إلى طابعة

write → الـ Data إلى داخلة



* Capacity of memory = 256 MB

* Cell = 16 bits

* Address bus = ??

$256 \times 2^{20} = \square$ Byte

1 cell = 2 Byte.

* 128 M cells $= 2^7 \cdot 2^{20}$

$= 2^{27}$ cells.

Address bus = 27 bus

width of cells =

width of Data.

* MBR = width of data

* MAR = length of address bus

* Address bus in memory ≠ Address bus in Cpu

* Slide 25 → مهم التعريفات

* Slide 26:

* Data Bus

⇒ 32 = one access we read 32 bit

⇒ 64 = 〃 〃 〃 〃 64 bit

هاد بأثر على performance وكل ما زاد بيزيد قديش بقدر أقرا data بمرة وحدة.

* Address Bus.

بأثر على حجم memory إلي بتعامل معها cpu

* determine the Maximum possible memory capacity of the system.

* point to point Interconnec

واحد يربط معهم كلهم .

*-Ch 11 : Addressing modes.

ADD 1000

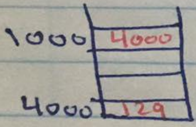
شو بفهم cpu ?? 1000 = Constant ?
1000 = memory address ??
AC = AC+1000 ??

هاي معناها Addressing mode

- Immedate "Constant" بتكون موجودة بال instruction نفسها
- Direct "memory Address" [1000]
- Indirect "memory "in direct""

address 1000 بروح على
باخذ القيمة الي جواها operand.

[1000] بروح على address 1000
القيمة الي جواه هي address "مثلاً"



بروح على 1000 شو جواها؟ → A acces نعمل مرتين
4000 هي كمان address "هو أبطئ" للميموري
باخذ القيمة الي جواها من direct
129

- Register:

ADD R0,R1,R2

* كلهم reg

- Register indirect
"مرضا" R1=1000

LDR R0,[R1]
LDR R0, 4000 ✓

* جوا reg هو memory address

• Displacement.

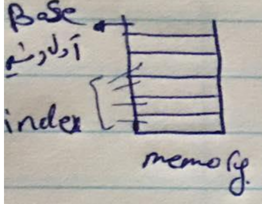
LDR R_0, [R_1+2]

• stack.

push R_0

* كيف بميز ~~[illegible]~~ ميكانيكيا "mode bit" حسب عدد والخيارات.

* Displacement Addressing



A + (R)
base address
index
هو ال
reg.

R_1 هو ال base address

ADD R_0, [R_1+1]

ADD R_0, [Array + R]

base
index in reg.

*note "LDA, STR" arm

"mov, Mov" Intel

* Relative Addressing!

Jump, Branch شرحناهم قبل

PC = PC + $\boxed{}$ الرقم المعطى

* Instruction length: هل ما كانت أقصر بكون أسرع؟

Allocation of Bits:- آنا بحدد كل اللي تحت لما بدي أعمل instruction

* # of add. modes
* # of operands
* reg. versus memory. " operating اللي بيسمح نتعامل مع reg or memory
* # of reg.
* # of address range.

* التركيز بهاد التشابتر على Ch 11 slides

* Arm → instruction "same length"

* Relative Addressing.
  jump 100 → $AC = AC + 100$

* displacement → المسافة من نقطة معينة

* Variable - Length Instruction :
  Risc → Variable length.

* Add r_3, r_3, #19.
  Add r_3, #19. Thumb " اختصار "