Computer organizationENCS2380

تلخيص محاضرات ابو السعود

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

* Computer Architecture: design بهتم بال

* Computer organization: بهتم بالتنفيذ والبناء

technology: organization ← implementation

programmer: Architecture المبرمج

Instruction → العمليات الي بعملها الـ cpu

* The computer Revolution:

Moore's Law → الترانزستورات كل سنة تقريباً بتصير

server ← مش للناس العاديين

هاد بلزم يكون interaction أسرع, أسرع cpu وأقوى

أكبر memory

server ← harddisk بقدر أخزن عليه أكثر من

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.
- Connects to the internet
- Hundreds of dollars.
- smart phone, tablets, electronic glasses.

. I/O , memory , prossesor

* cloud computing.

- warehouse scale computers (WSC)
- Software as Service " Saas "
- portion of software run on pMD & a portion run in the cloud.
- Amazon and Google.

"

* High level Language → compiler → Assemply.
Assemply → Assembler → machine Language.

Machine Language

parallel processing

preformance in parallel

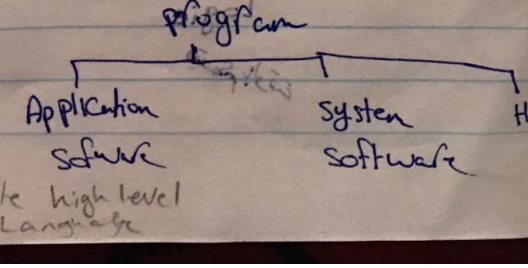
* understanding performance

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

1. Design for Moore's law.
2. use abstraction to simplify design.
3. Make the common case fast
4. performance via parallelism
5. // // pipelining.
6. // // prediction. jump
7. Hierarchy of memory.
8. Dependability via redundancy.



System Software ↑ Hardware
↓ Human Architecture

Compiler → translates HLL → mechin Language.

operation system: service code.

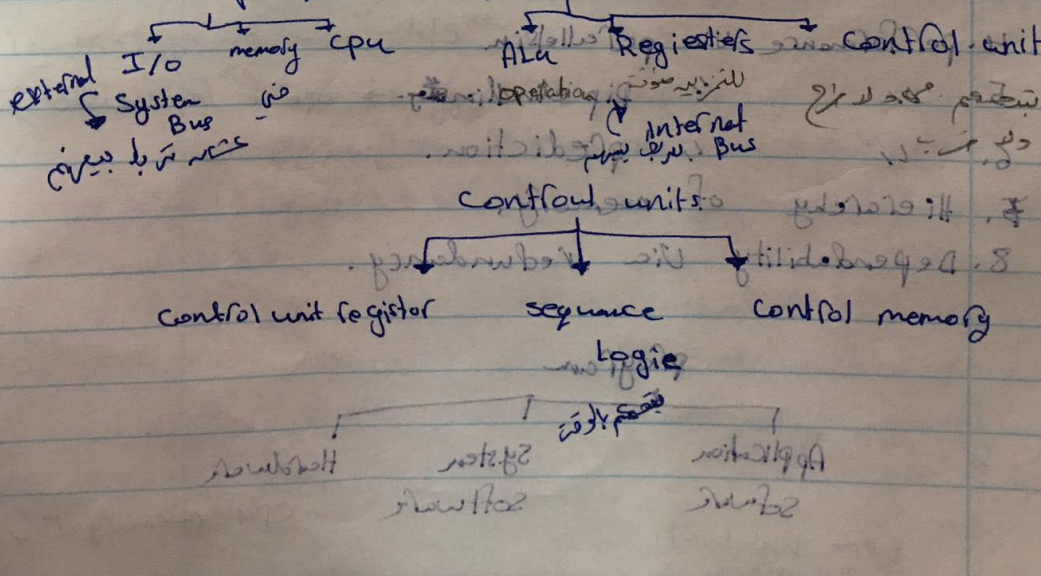
- handling I/O
- Managing memory and storage.
- Scheduling & tasks sharing resources.

Comparable ~~or~~ organization ~~ISA~~ *

Programe Code →
HLL → Assemply → Hardware representation.
binary digits

assembler → compiler Encoded instruction

* structure.



moore's Law : تنبأ أنو بتقدر تضاعف عدد ترانزستورات كل سنة وشوي (١٨ شهر) ببعض المادة الحديدة

* Multicore Computer structure :-

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

* cache memory "Mega"
  - code تخزين داتا و data & code
  - smaller & faster
  - لتوضيح cache بتضاعف إذا زادت الـ performance بتزيد
  - cache ← يعني إذا زادت هوي كافي نفس الشي

* a safe place for data

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.
- wireless network ;: wifi, Blutooth

* History of computers.

first: Vacuum Tubes. G1 "machin Languge" G1

حجم كبير ، استخدم Vacuum ، بيوكل power كتير . بطيء "جدا"

أول كومبيوتر كان البرنامج ما stored program concept →

بيتخزن ، أنا بجيب أخزن البرنامج جوا الكومبيوتر بالمعمورية أرجعله وأنا بحكي للكومبيوتر ينفذها .

machine code

G2 → smaller, cheaper, transistor, made from silicon

less heat than vacuum tube

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 → بقى فيها ترانزستور أعداد هائلة .

Gordon Moore: co-founde of Intel *

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

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

* micro controller

Ram

Rom

* Response time: technology

How long it takes to do task.

* throughput:

Total work done per unit time.

* Replacing the processor with a faster version. ?

* Adding more processors?

* Relative performance

Define performance = $\frac{1}{\text{Excution Time}}$

"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, 15 s on B
- Excution time B / Excution time A

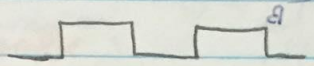
15 / 10 = 1.5

so A is 1.5 time faster than B

* Measuring Excution Time

- Elapsed Time:
  - * Total response time, including all aspects
    processing I/O, OS overhead, idle time.
  - * determines system performance.
- Cpu time.
  - * 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.



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

cpu time = cpu clock cycles * clock cycle Time

= cpu clock cycles / clock rate.

performance improved by.

1. Reducing number of clock cycles.
2. increasing clock Rate.
3. Hardware designer must often trade off clock rate against cycle count.

* Example : Computer A : 2 GHz , 10 s cpu time.
Designing computer B
. Aim for 6s cpu time.
, can do faster clock , but causes 1.2 * clock cycle
How fast must comp B clock be ?

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

$$\text{Clock cycle (A)} = \text{CPU time} * \text{Clock Rate (A)}$$
$$= 10s \times 2\,GHz = 20 \times 10^9$$

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

لازم (B) يشتغل على 4 GHz عشان يخلص البرنامج بـ "6s"

* Clock cycle = Instruction count x cycle per Instruction

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

Instruction →

العمليات "opcode"

Clock cycle = عددهم x cycle قديش كل وحدة بدها تنفيذها

$$\text{(s) CPU time} = \text{Instruction count} \times \text{CPI} \times \text{Clock cycle time}$$
$$= \frac{\text{Instruction count} \times \text{CPI}}{\text{Clock Rate}}$$

* Instruction count for a program. الها علاقة بـ design

Determined by program, ISA, Compiler.

RISC → هاي أسوء ، CISC → أكثر → Instruction count.

* Average cycles per Instruction.

Determined by CPU hardware. RISC → أحسن

if different instructions have different CPI

Average CPI affects by instruction mix

Clock cycle time "Rate" ... technology ... design ...

* Example :

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

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

same ISA ... Hardware ... instruction ...

which is faster , and how much ?

↓

cpu time.

$$\text{cpu time}_{(A)} = \text{Instruction count} \times \text{cycle time}_A$$

$$= I \times 2.0 \times 250 \text{ ps}$$

$$= I \times 500 \text{ ps} \rightarrow A \text{ is faster}$$

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

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

A is faster than B by 1.2

cpu ...

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

* Weighted average CPI

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

Relative frequency

* CPI Example:

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

Class	A	B	C
CPI for class	1	2	3
IC in seq. 1	2	1	2
IC in seq. 2	4	1	1

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

  clock cycle

  $= 2 \times 1 + 1 \times 2 + 2 \times 3$

  $= 10$

  Avg. CPI $= 10/5 = 2$.

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

  clock cycles

  $= 4 \times 1 + 1 \times 2 + 1 \times 3 = 9$

  Avg $= 9/6 = 1.5$

Avg CPI

* performance. Summary.

$$\text{CPU Time} = \frac{\text{Instruction}}{\text{Program}} \times \frac{\text{clock cycle}}{\text{Instruction}} \times \frac{\text{seconds}}{\text{clock cycle}}$$

↓ design ↓ CPU ↓ technology

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

* in CMOS IC technology.

$$\text{power} = \text{capacitive load} \times \text{Voltage}^2 \times \text{frequency}$$

* Reducing power

- suppose a new CPU has
  - 85% of capacitive load of old CPU
  - 15% voltage and 15% frequency reduction.

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

$$= 0.85^4$$

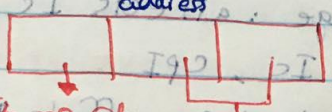
$$= 0.52$$

* The power Wall

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

* Ch 12.

Characteristics and functions



op code.

Instruction.

36 instruction → 6 bits

* n bits عدد

oper = 2^n

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

memory.

128 MB

Cell = 32 bits

$128 = 2^{27}$

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

address field → 25 bit

add = 18 bit

2^{18} → Cells

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

A+B → الشكل العادي

AB+ → post fix

+AB → prefix

* Types of operands:

address – Characters – Number – Logical Data

2 , add, constant, char

10 ??

ASCI Code

* Single - Instruction - Multiple - Data

"SIMD" Data types

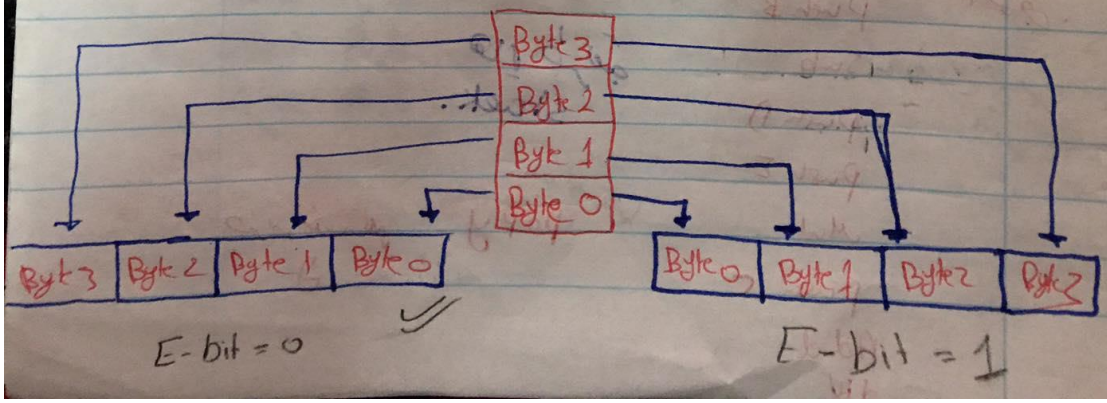
1 instruction ...

* Arm

8 byte

16 halfword

32 word



Byte   Byte أول

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

move → 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   "brach"
[0 - 1 = -1 ; Zflag = 0 ما بجمب jump]
skip [...]   على xyz

بحط → xyz str R0,[100]

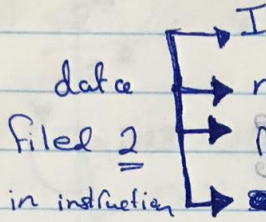
*Data Transfer :-

32 bit = 4 Byte

1. Source من   LDR R0, [100]
2. destination لوين   بنقل شو بدي اللي جوا
3. Data كمية   [100] ما بنحطها بتقل

strb r0, [100]
→ byte.

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



4 → register يلزم 2 bit

32 register يلزم 5 bits

risc → عدد الـ register

عدد bits

register = 2^n

24 register يلزم 5 bits

register << memory. عدد bits

Constant يعني لـ

5 bits → 0 → 31 un signed.

$0 \rightarrow 2^n - 1$ ← عدد الأرقام. عدد bits هو range

5 bits → $-2^4 \rightarrow 2^4 - 1$ signed.

$-2^{n-1} \rightarrow 2^{n-1} - 1$

+12 = 01100

-12 = 10100 2's complement.

- 512 → 511 → 10 bits يلزم

-2^{n-1} => 9 = n-1

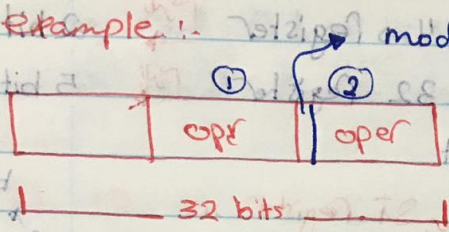
n = 10.

Instruction → | opCode | operande Refrence | operand Refrena |

4 bits 6 bits 6 bits

16 bits

Example :-



① → only register

② → memory or ~~register~~ Constant.

1 bit يعني بدي

* support 48 operation

* 24 register

Op Code => # of operation = 2^n

$48 = 2^6$

$n = 6$ bits

① → register ⇒ # of register = 2^n

$24 = 2^5$

$n = 5$ bits

* mode = 1 bit

② = 32 - (6+5+1)

= 20 bits

memory

of cells = 2^{20} = 1 MB

memory = $2^{20} \times 1$ = 1 MB

$\frac{2^{27}}{2^{*}} = 2^{26}$ ← # 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
ADD  y, A, B
DIV  y, A, B
MPY  y, A, B
```

RAM

y: distination

y ← A+B

② Two address instruction

```
SUB  A, B
MOV  A, B
ADD  A, B
```

A, B: A الجواب

A ← B → move B in A.

③ one address instruction

```
ADD C
```

Load D — انقل من memory على register ac
stor y
SuB B → فرعنا اشي implecit حرفيا بنقص من B والنتيجة نخزنها في "ac"
DiV y.
inc C — 1 يزيد increment.

Stor y → y ← Ac
نقل من Ac على y

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

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

	memory
B Tos	5
A (Tos-1)	7

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

+	Add
−	Sub
÷	div
↔	Multp

A B + — push A, push B, add.
+ A B

للعش الي فوق * — push A
Push B
Sub ← بطرح A من B
Push D
Push E
Mul
Push C
Add
pop y ← خزنه المتبقي في y

→ y = AB- CDE * + /

هاي طريقة stack.

A+B → الشكل العادي

AB+ → post fix

+AB → prefix

* Types of operands:

address – Characters – Number – Logical Data

2 , add, constant, char

10 ??

ASCI Code

* Single - Instruction - Multiple - Data

"SIMD" Data types

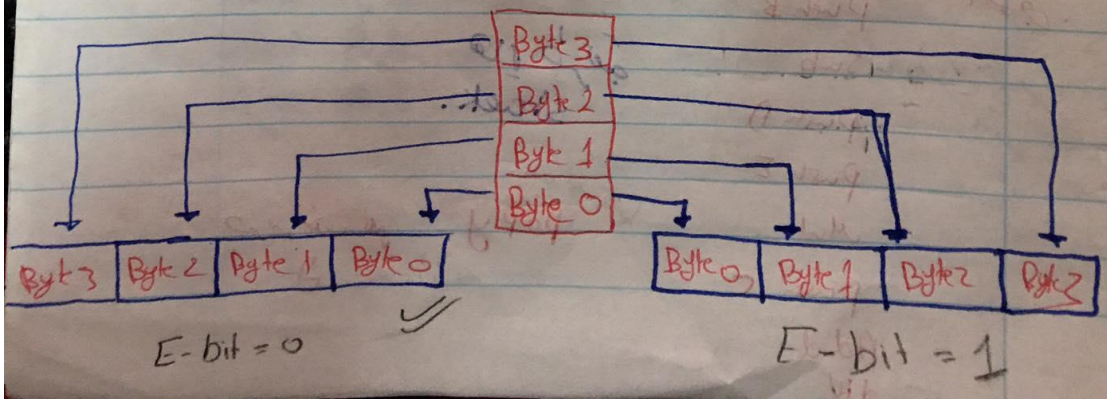
1 instruction ...

* Arm

8 byte

16 halfword

32 word



Byte ↑ STRB R0 ,[100] — خزنلي B من R0 على (↑ أول Byte) 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 — خط واحد جوا
moiv R1,0 — حط 0
Sub R0, R1, R0 — $R_1 - R_0$
BZ xyz — jump if Zflag = 1 "brach" على xyz

[0 - 1 = -1 ; jump ما بجي Zflag = 0]

skip [⋮ ⋮]

بيجي → xyz str R0,[100]

*Data Transfer:-

32 bit = 4 Byte ↓ LDR R0, [100]
نقل شوي للداتا الي جوا [100] ما بنحذفها بنقل

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

strb r0, [100]
↓ byte.

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

المقراءة من memory و mov مابتروح الداتا بس copy
بتروح stack

* unsigned المهم carry

* signed المهم overflow

1001	9
1110 +	14 +
10111	23

1 ← carry . بنحطها جوا carry flag .

Carry flag = 1 يعني في مشكلة overflow — | O | | F (1) |

* اذا في carry يعني في overflow ← unsigned.

* signed. — | O (1 هاد المهم) | | F |

1001	−7 0111 2's c
1110 +	−2 0010 + 2's com
0111 هاي صفر والجواب لازم −9 يعني 1	الجواب لازم −9

يعني overflow = 1

* اذا جمعت رقمين (−) والجواب طلع (+) يعني في overflow = 1
* // // // (+) // // (−) // // //
* غيرهم فش overflow

* shift Right → جواها صفر [0 | .... | | |] → بتروح ع carry flag

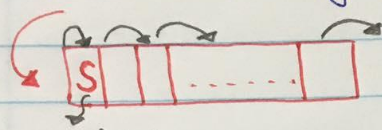
* shift left ← بتطلع ع carry flag [.... | | |] ← 0 جواها صفر

آخر bit طلع ← carry

Logical جدول 3

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

قسمة



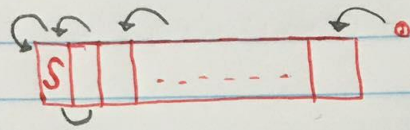
Arithmatic Right shift.

هو بيضيف نفس قيمة S

Signed bit

أهم bit ممنوع التحرك

ضرب



Arithmatic left shift

بيضيف 0 عادي ما بأثر على S

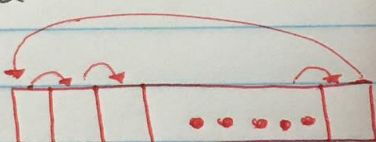
هدول صفرات يصير over flow

بالضرب الرقم بكبر

القسمة بقل الرقم ومستحيل يصير over flow وهون تقريب

$\frac{5}{2} = 2.5 \simeq 2$

أكبر عدد صحيح أقل من الرقم



Rotate.

12 → 0001 0010 BCD] → Conversion.
0000 1100

* Transfer of control :
  - jump → Conditional.
    - → un conditional. true "دائماً"

* Examples of shift and Rotate operation-

10100110 | Logical right shift 3 bit | 00010100

3 bit بنزاحيف

Carry = 1 → آخر bit طلع من register

10100110 | Arithmatic left shift 3 bit | 10110000

10100110 → 10110000

هاي بتضل ثابته ممنوع أغيرها — بنحذفو — اليأ ثلاثة نضيفهم

10100110 | Arithmatic right shift 3 bit | 11110100

10100110 حذف

هاي لازم تضل 1 عشان هيك نضيف 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 آكتر من memory "مابدنا cpu يتعامل معها direct"

* Transfer of control : الأصل أنه ينفذ الكود sequential

يعني سطر سطر حسب الترتيب وبعدين في Instruction ممكن تغير الاشي

ex : jump , Branch "نفس الاشي بالمعنى"

un conditional → true دائماً

Conditional → لازم نفحص

✓ ....
✓ ....
✓ ....
✓ ....
BR 100 — فرضنا هاد كود رح يعمل كل إلي عندهم ✓

✗ ....
✗ ....
✗ ....
✗ .... → skip "ماينفذ"

100 ____ ✓

... بكمل

* un conditional هيك تشتغل

* Conditional

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

* Conditions → flag مبنية على

cf , zf , of , sf

cf → هل في carry إذا آه cf = 1

zf → لما 0 كل النتيجة zf = 1

of → over flow

sf → إذا النتيجة (-) SF = 1

إذا (+) sf = 0

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

إذا Fz=0 بعمل Code بالترتيب "Seq"

jNZ 300 — اذا Fz=0 اعمل jump

* بالعود إلى قبل pc = 6 "بتأشر على next instruction"

بس لأنه فيه BR 100 رح يأشر على قيمة pc

"pc = pc + 100"

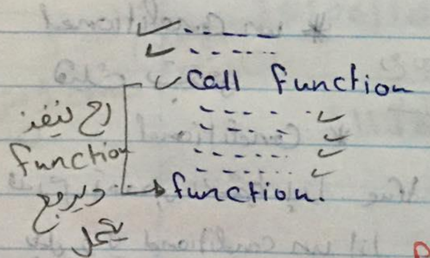
pc = pc + displacement.

pc = address

* Call function. "procedure Call"

هي بتشبه jump وبعد

كل اشي بتنفذو



pc = 4

"push" بتحزن 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 ↷ push on stack "اللي بعد Call مباشرة"
pc = [memory]
ret → بيعمل pop stack
pc = pc الأولية

4000 ------ ✓
------ ✓
------ ✓

push → 4601 (crossed out)
4101
Stack

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 proc2 pc -
4651 -------

Return
4800 -------

------- proc2

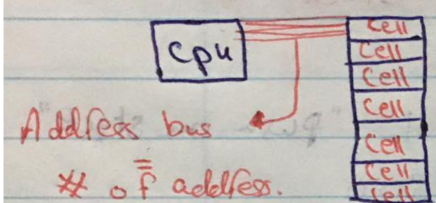
Return. → pop pc "pc = 4601"

X 86 → cpu "نُسَخ" مش مهم كتير ^^

* ch 3 :

memory → read تقرأ
→ write نكتب

I/o → read نأخذ قيم منهم
→ write نطبع عليهم



cells → نفس ال length

Address bus : بكون مكتوب عليها ال address

Control bus : مكتوب عليها شو بعمل ال 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 مشبوكة مع address bus

* أي اشي بدي انقله من reg ل memory لازم انقلها بالأول على MAR وهذا بتغير 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} = Th

* MBR : memory buffer register. مشبوكة مع Data Bus

Data Bus = Cell length.

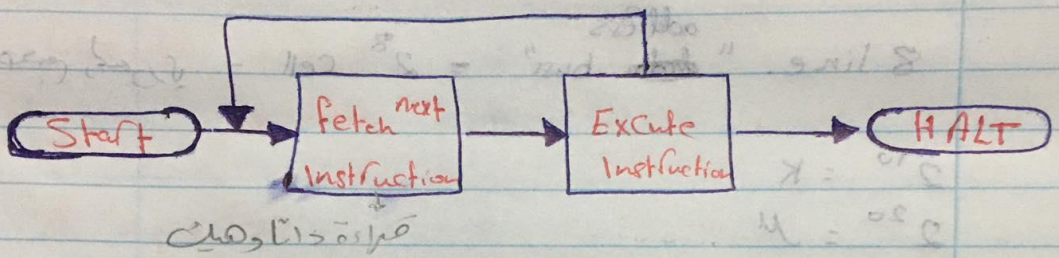
* I/O AR : I/O address register
هاي مشبوكة مع address bus I/O register

* 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

PC → (فيعطيها) MAR ← هنا بأخذ رقم الي أشرت عليه add memory
IR ← MBR ← من memory برجع الداتا من Bus

* pc: auto increment ← note.
fetch cycle Execute cycle

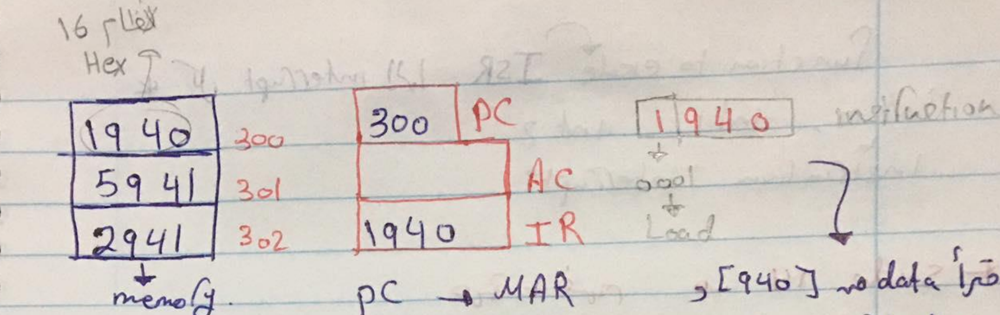


قراءة ذاتية

* action categories "operation تصنيف"

- processor-memory: Data transferred 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

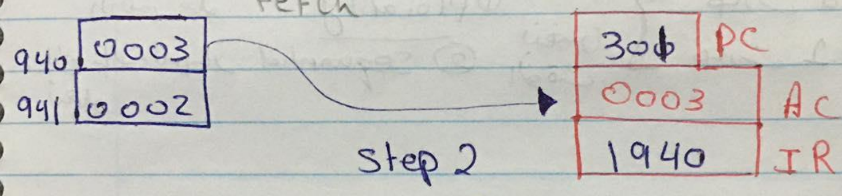
* Example.
  AC → "one register"
  add [100], [200] AC تخزن النتيجة
  add [10]
  "AC" أي op من موجود مكان بكون



PC → MAR
IR ← MBR

اقرأ data من [940] و حط القيمة ب Ac

step 1
fetch



Step 2

* لازم أعمل increment لـ PC

* Slide 12 مهم

* Interrupts: مقاطعة

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

• Interrupt: بعطي صبح أو اشارة انه صار interrupt

Interrupt Bus → 0 → عادي بضل شغال
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 + فهم
* Slide 15 نفهم الرسمة

أولوية "الكتابة فيها خطأ"

* int 0 صارت] بنفس الوقت
  int 1 صارت]

① proiarity. الأهم نعمله

② sequantial اللي بالأول صار نعمله

* كل الـ interrupt ليها ISR function to excute

int 0, int 1, int 2

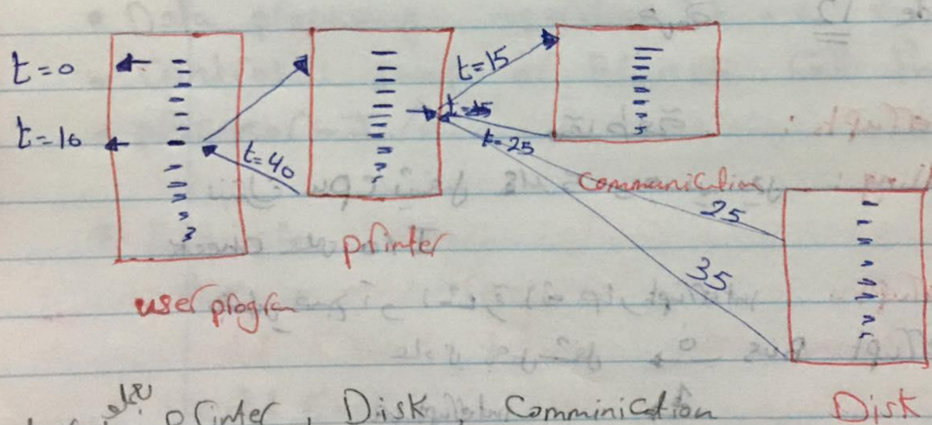
instruction "interrupt"

* Slide 14 مهم + فهم
* Slide 15 نفهم الرسمة

أولوية "الحاجة يعني خطأ"

* int 0 جات] بنفس الوقت — ① prioirity. الأهم نعملو
  int 1 جات — ② sequential اللي بالأول جار نعملو

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



interrupt بعلو printer, Disk, Comminication

كل وحدة فيهم بدها 10 sec عشان تخلص كل instruction → prioirity. : printer = 1

communication = 2

Disk = 3

من الأول: Disk > Comminication > printer

① البرنامج "user program" كان بشتغل لحد 10 sec
عند 10 sec صار "printer interrupt" لحد 15 sec صار
"Communication interrupt" يعني الـ printer اشتغلت بس 5sec بنجيب نقارن
هي الي priority أكبر "Communication" بروح تخلص شغلها بعدها
10 sec وهي أصلا وقفت عند 15 بتصير 15 + 10 = 25
عند t = 25 صار Disk interrupt بعدها 10 ثواني بتخلص وترجع
25+10 = 35 بترجع عند printer / بتكمل بعدها 5sec ثواني
خلص t = 40 بترجع عند user program

* لو مشينا نفس الـ "Sequential" "whith out priority"

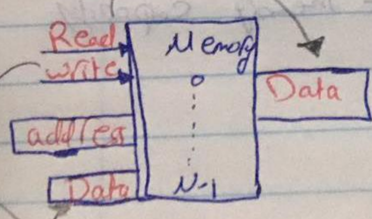
printer → start : 10
→ End : 20

Communication → Start : 20
→ End : 30

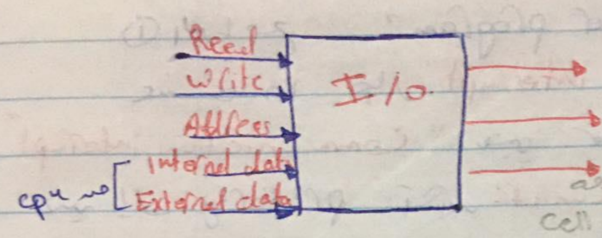
Disk → start : 30
→ End : 40

* I/O function

* لكل I/O في الها address مشان نقرأ الـ data



address → cell address
read → بنطلع بس الـ Data إلى طالعة
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.

width of cells = width of Data.

* MBR = width of data
* MAR = length of address bus

Address bus = 27 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 أقرأ data بمرة وحدة.

* Address Bus.

بأثر على حجم memory الي بتعامل معها cpu

* determine the Maximum possible memory capacity of the system.

* point to point Interconnect
واحد بربط معين كامل.

* Ch 11 : Addressing modes.

ADD 1000

شو بفهم cpu ?? 1000 = Constant ?

1000 = memory address ??

AC = AC+1000 ??

] هاي معناها Addressing mode

• Immediate "Constant" — بتكون موجودة بال instruction نفسها

• Direct "memory Address" — [1000]

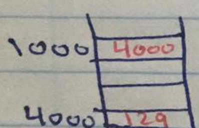
• Indirect "memory "in direct""

→ address 1000 بروح على

operand باخذ القيمة الي جواها

[1000] — بروح على address 1000

القيمة الي جواه هي address "مثلاً"



بروح على 1000 شو جواها؟ → Access — بنعمل مرتين

4000 هي عبارة address — للميموري "هون أنظر"

باخذ القيمة الي جواها — مع direct

129

• Register:

ADD R0,R1,R2 — * كلهم reg

• Register indirect — * جوا reg هو memory address

"مثلاً" R1=1000

LDR R0,[R1]

LDR R0, 4000 ✓

• Displacement.

LDR R_0, [R_1 +2]

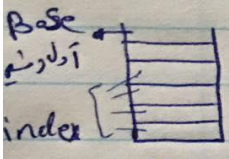
• stack.

push R_0

* كيف نميز حركياً "mode bit" حسب عدد وخيارات.

* Displacement Addressing.

R_1 لحتوي base address



A + (R)

base address

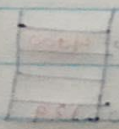
index لحتوي reg.

ADD R_0, [R_1 +1]

ADD R_0, [Array + R]

base ↵ ↳ index in reg.

*note "LDA, STR" alm "mov, Mov" Intel



* Relative Addressing:

Jump, Branch حسب ما هم قبل

PC = PC + [ ] الرقم label

* Instruction length: كل ما كانت أقصر بكون أسرع

Allocation of Bits: - آلية توزيع كل البت في instruction

* # of add. modes
* # of operands
* reg. versus memory. " operation بين reg or memory
* # of reg.
* # of address range.

* التركيز فيها والمتابعة Slides على Ch 11

* Arm → instruction "same length"

* Relative Addressing.
  jump 100 → PC = PC + 100

* displacement → الجمع مع قيمة معينة

* Variable - Length Instruction :
  Risc → Variable length.

* Add r3, r3, #19.
  Add r3, #19. Thumb "اختصار"