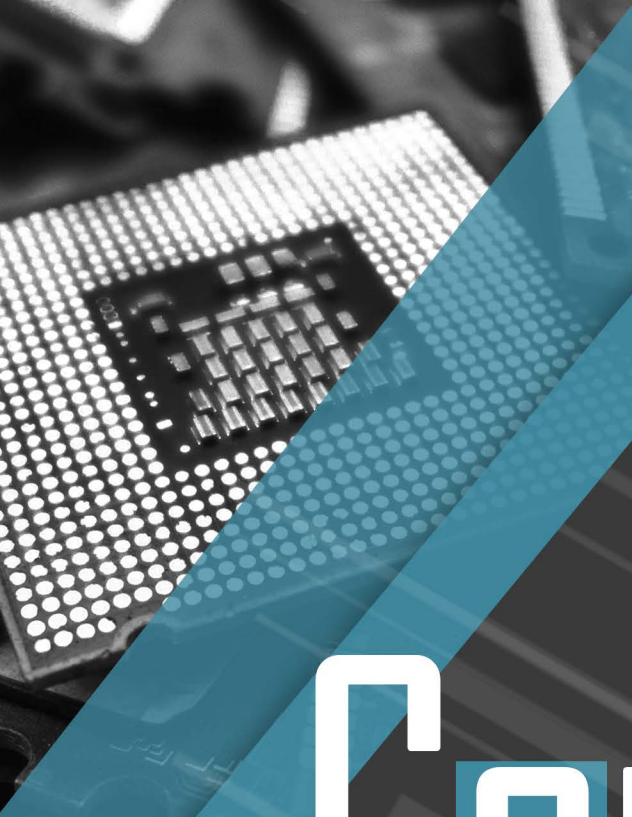


Computer Organization

** فاعل خير

Computer Organization & Design...

الشابتر الأول غير داخل بالامتحان... بس بدنا نحكي عن شوية أشياء مهم إننا نعرفها ونفهم الفكرة منها...

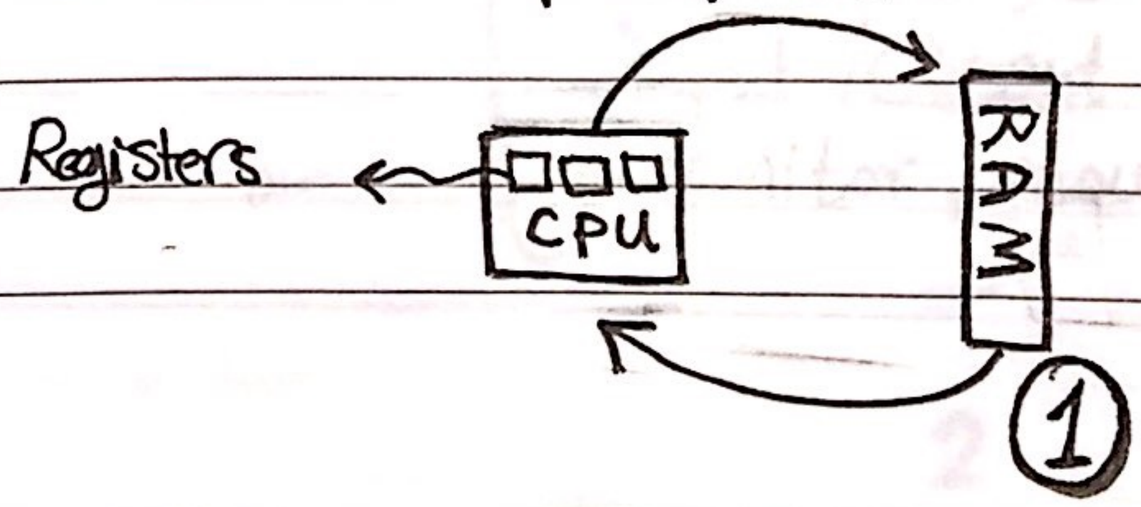
* في البداية لازم نعرف انه (CPU) (المعالج) جزء مهم جدًا في عمليات جهاز الحاسوب وأي عملية تقريبًا بتصير بالحاسوب إنها بتصير داخل الـ CPU.

* المعلومات والأوامر بتكون مخزنة داخل الـ (memory) وبالمادة عنّا منركز على الـ (RAM memory).

* رح ندرس كيف هاي المعلومات والأوامر بتنتقل من الـ RAM وبتوصل على الـ CPU اللي بتحتوي بداخله على ما يسمى بالريجسترات اللي حنشتغل عليها.

* لما بدي أنفذ برنامج أو أحتاج أطلع بيانات من الـ (RAM) الرحلة بتكون بين الـ CPU والـ RAM بحيث انه لما الحاسوب يشغل برنامج يبعث عنوان الـ (RAM) حتى يطلب منها المعلومات أو الأوامر المخزنة بهذا العنوان، والـ RAM بترسل أي من المعلومات أو الأوامر المخزنة بهذا العنوان للـ CPU، يتم معالجتها وتخزينها باستخدام الـ ALU والريجسترات وبمجرد الانتهاء من معالجتها يُرسل عنوان آخر (مؤقت) إلى الـ RAM وبتوصل بيانات جديدة للـ CPU، عادة ما يتم إرسال البيانات المعالجة من الـ CPU إلى الـ RAM لتخزينها.

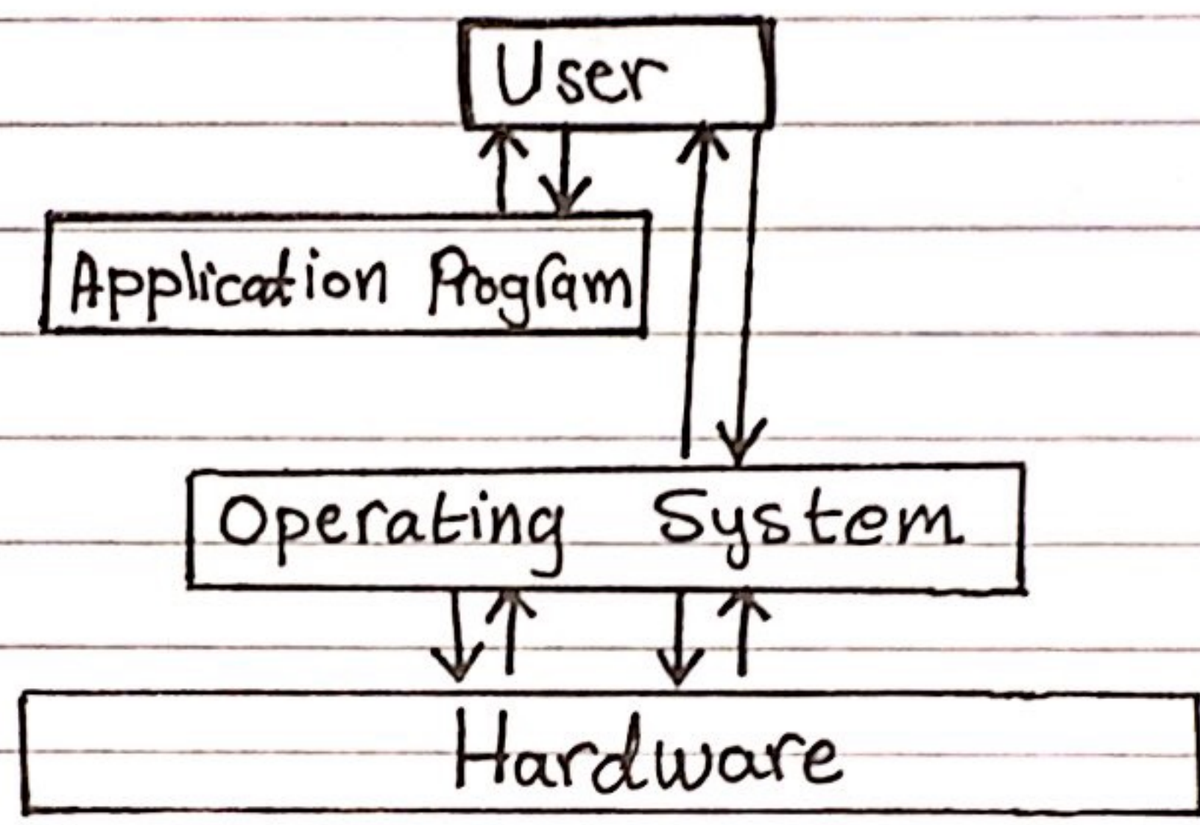
* إذًا الـ RAM عنا بتحتوي على كل البيانات اللي عالجناها بالـ CPU.



* Computer system
  - Hardware
    - physical parts
  - Software
    - programs

* Software
  - System Software (e.g Windows 7)
  - Applications Software (e.g Photo shop)

* كيفية العلاقة بين المكونات المادية والبرجية :-



* Hardware (physical)
  1. *Central Processing Unit (CPU)
     - recieves and executes instruction.
  2. *Main memory
     - holds instructions that are about to be processed.
  3. *Secondary Storage
     - hard drive , DVD... etc
  4. *input /Output Devices
     - monitor , mouse

بالتفصيل instruction شرح الـ
لقدام

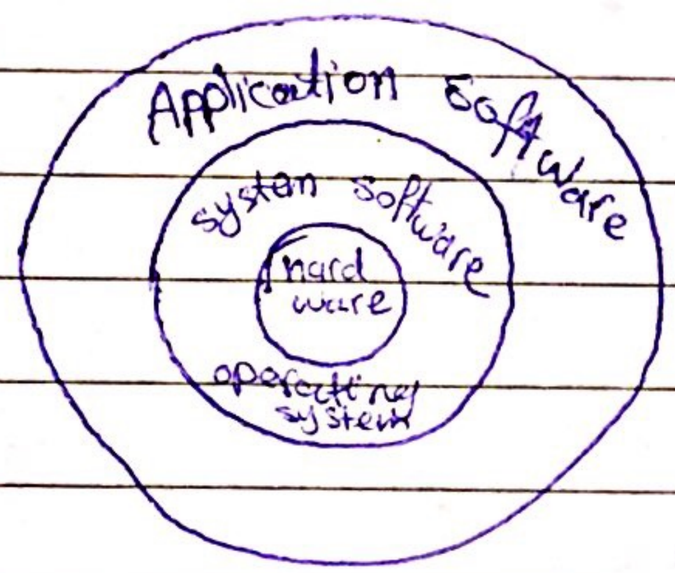
مصطلحات ومفاهيم حيحكي عنها الدكتور بالمحاضرة

* The word of computer language are called instruction and It's Vocabulary is called Instruction set. → Ex: Add, Sub, Jump ....

الأوامر

* How does a computer get instructions?
A computer is given instruction by a human using a keyboard or other input device. the computer uses an operating system or software that is also programmed by a human to translate the user commands into something the computer can understand.

* نظام التشغيل يؤمن التواصل بين البرامج والتطبيقات والمكونات المادية فعندما يطلب المستخدم مثلاً تخزين ملف عبر تطبيق معين يقوم التطبيق عبر نظام التشغيل بطلب تخزين ملف على القرص الصلب فيقوم نظام التشغيل بمخاطبة القرص الصلب وتأمين تخزين الملف في المكان الصحيح.



Levels of program code:
High level language (in C) $\xrightarrow{\text{compiler}}$ Assembly Language (for MIPS) $\xrightarrow{\text{assembler}}$ machine code (Binary digit).

شرحها في الصفحة التالية

مثلاً :-

$C = B + A \xrightarrow{\text{Compiler}}$ add $t_3, t_1, t_2 \xrightarrow{\text{assembler}}$ 0110.....

* Compiler :- A program that translate high-level language into assembly language.
* assembler :- A program which translates assembly language to machine language.

* البرنامج يكتب بلغة عالية المستوى اللي هي لغة قريبة من الانسان وهو
بقدر يتعامل معها مثل (C++) بس الحاسوب ما بفهم إلا (صفر أو واحد)
فبما انه البرنامج عبارة عن عدة أوامر (instruction) وحتى يقدر الحاسوب
يتعامل معها فبتحول أولاً إلى assembly عن طريق ال assembler
وبعدين إلى لغة الآلة عن طريق ال assembler .

- نرجع للي حكينا:

* داخل ال CPU في عندي رجيسترات ولما تنتقل الأوامر من ال RAM لل CPU بتخزن
جوا رجيسترات بعد ما يتم معالجتها وقبل معالجتها.

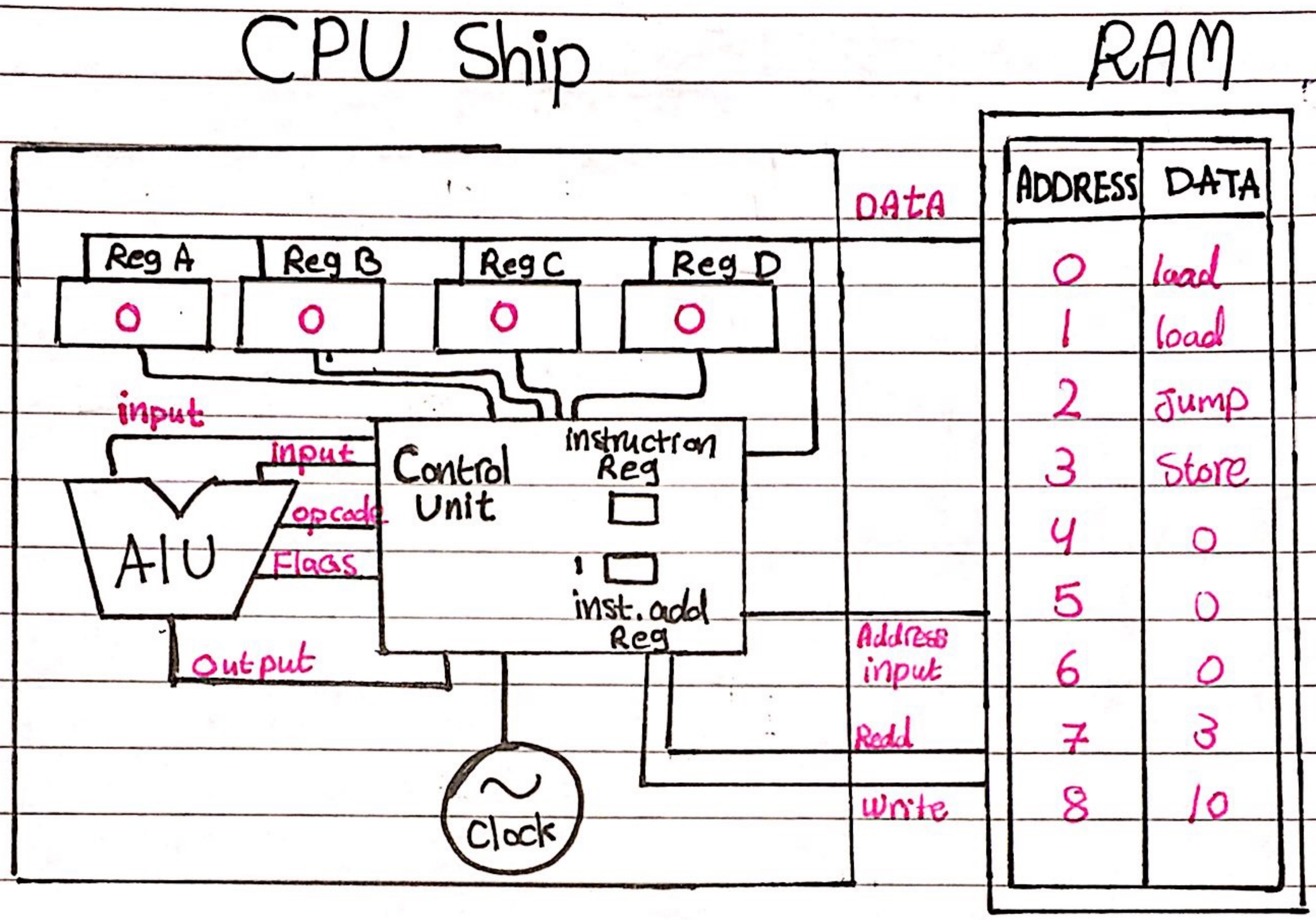
Register : A special , high-speed storage area within the CPU
All data must be represented a register befor it can be
processed. Ex: IF two number are to be multiplied, both numbers
must be in registers, and the result is also placed in a register.

$: رمز الرجيستر / Ex: $t0 contains g+h.

* حيكنا انه الأوامر (instruction) بتتخزن جوا الريجيسترات وهاد الرجيستر
بوسع فقط (32 bit) يعني طوله 32 bit ، اسمه يكون من (5 bit) لأنه
بوخد الارقام من (صفر - 31) ← (00000 ← 11111)

* فعدد الريجيسترات بنظام 32 bit هو 2^5 رجيستر.

* شو يعني نظام كان 32 bits ← يعني أبعد عنوان ممكن يوصله ال CPU
بالميموري هو 2^{32} .



* there are many types of register in Control unit - in process in cpu - that connected with memory.

* في رجيسترات خاصة بالانستركشن ، في رجيسترات بتخزن العنوان ورجيسترات بتخزن المعلومات الخاصة بالانستركشن حتى يعمل العمليات عليها وكمان في رجيسترات بتخزن عمليات الـ ALU حسب العملية وكلهم مشبوكين بأسلاك خاصة بعطوا عنوان وبياخدو من الميموري وتخزنوا بالرجيسترات حسب نوع الداتا وبعملو العمليات والـ ALU بشتغل شغله وبرجعوا تخزنوها بالميموري .

* مش مهم كتير تفاصيل الرسمة الأهم هي الفكرة العامة.

MIPS Register Convention

Name	Register Number	Usage	Preserve on call?
$zero	0	constant 0 (hardware)	n.a.
$at	1	reserved for assembler	n.a.
$v0 - $v1	2-3	returned values	no
$a0 - $a3	4-7	arguments	yes
$t0 - $t7	8-15	temporaries	no
$s0 - $s7	16-23	saved values	yes
$t8 - $t9	24-25	temporaries	no
$gp	28	global pointer	yes
$sp	29	stack pointer	yes
$fp	30	frame pointer	yes
$ra	31	return addr (hardware)	yes

أنواع الريجستر

$t0 - $t7
$s0 - $s7
$t8 - $t9
الأكثر استخداماً

$zero : دائماً قيمته zero ما بتخزن فيه

$t0 - $t9 : temporary.

$s0 - $s7 : final result.

* Memory Operation :-

- instruction are represented as 32-bit.

حكينا انه الريجستر بوسع (32 bit) عشان هيك ال (instruction) بكون عبارة عن 32 bit حتى يوسع جوا الريجستر.

عندنا أنواع مختلفة من ال (instructions) مثل :- add , or , sw , lw .....
حنتعرف عليهم وحدة وحدة بس بالأول بدنا نشرح موضوع مهم جداً. لخص ال memory .

عرفنا انه البيانات والأوامر بتكون موجودة على ال memory وانا لازم أحملها على الريجستر حتى أقدر أعمل عليها (نرجع لتعريف الريجستر)

* عرفنا كمان انه ال instruction الوحدة عبارة عن 32 bit .

* Mips word = 4 byte = 32 bit.

• تخيل ال memory هيك:

0	32 bit
4	32 bit
8	32 bit
12	32 bit

* لما بدي اقرأ الانستركشن الاولى بروح على عنوان صفر مثلاً بكون فيه 32 bit .

* الانستركشن الثانية بكون عنوانها (4) لأن كل انستركشن بدة 4 bit

• شكل آخر لل memory :

0	8 bit
1	8 bit
2	8 bit
3	8 bit
4	8 bit
5	8 bit
6	8 bit
7	8 bit

الانستركشن الاولى = 32 bit = (word)

الانستركشن الثانية

* نلاحظ انه الانستركشن الأولى بدأت من الصفر والثانية من العنوان (4) والثالثة حتبدأ من العنوان 8 ، وهذا ماسمي بالـ (word alignment) يعني أنا بديت من الصفر، وأي عنوان لأي انستركشن بكون بيدأ من صفرين من اليمين فبكون من مضاعفات الاربعة . 0: 0000 / 4: 0100 / 8: 1000 / 12: 1100

* حكينا بدنا نجيب البيانات من ال memory لل register وهاي العملية اسمها (LW) وإذا بعد ما أجريت العمليات اللي بدي اياها على البيانات ورجعت بدي أودعها من ال (register) لل (memory) بسمي هاي العملية (SW).

- بعملية الـ (LW) ← (load word) نحتاج أعرف العنوان اللي بدي أحمل منه.
((From memory to register))
- بعملية الـ (SW) ← (store word) نحتاج أعرف العنوان اللي بدي أخزن فيه.
((From register to memory))

* طريقة حساب العنوان: (real address)

* real address = base address + offset

base address : هو عنوان اللي أعرف (مكان اللي بدي اياه) بالنسبة اله وغالبًا بتكون قيمته موجودة.

offset : الازاحة بالنسبة لـ Base address، وبكون سالب أو موجب.

* offset = index * word size (byte)

index : رقم الـ array

word size : تساوي (4) غالبًا ما نتعامل مع الـ word.

A[8] ⇒ offset = 8 * 4 = 32 , Base address = 100

reat address = 32 + 100

* الصيغ لكتابة instruction تبع الـ (LW) والـ (SW) :-

register ← memory حمل من

- LW Reg1 , offset (Reg2) = (Reg1 = mem [offset + Reg2])

بنجمعوا مع بعض بطريقة رح نعرفها بعدين ويشكلوا الـ (real address)

(Reg2) : يُمثل الـ Base address.

يعني الـ Reg1 صار يحتوي على الـ data الموجودة في الموقع (Reg2 + offset)

خزن من Register الى memory

- sw Reg1 , $\underbrace{\text{offset (Reg2)}}_{\text{real address}}$ = (mem[offset + Reg2] = Reg1)

Reg2: Base address.

يعني الموقع (Reg2+offset) صار يحتوي على الـ data الموجودة بالرجستر (Reg1).

- رح نحكي هلأ كيف بتصير طريقة التخزين.

*Byte Addresses:-

1- Big Endian.
2- Little Endian.

most 31 — least 0

0x A1 B2 C3 D4

		least
a	0X A1	
a+1	0X B2	
a+2	0X C3	
a+3	0X D4	
a+4	0X E5	
a+5	0X F6	
a+6	0X B1	
a+7	0X C3	most

(8 bit) / يعني هيكسا ← a

*كيف حنعبي الرجستر؟

- كل مرة بشيل 4 بلوكات من الـ mem وبحطهم بالرجستر (البلوك الواحد بيكون فيه 8 bit أو 2 byte)

- بركز على الـ (least) تبع الـ (memory) يا إما بحطه:

① بالـ least الخاص بالرجستر ويسمى عندئذ (Little Endian).

② بالـ most الخاص بالرجستر ويسمى عندئذ (Big Endian) وهو النظام اللي بنتعامل معه.

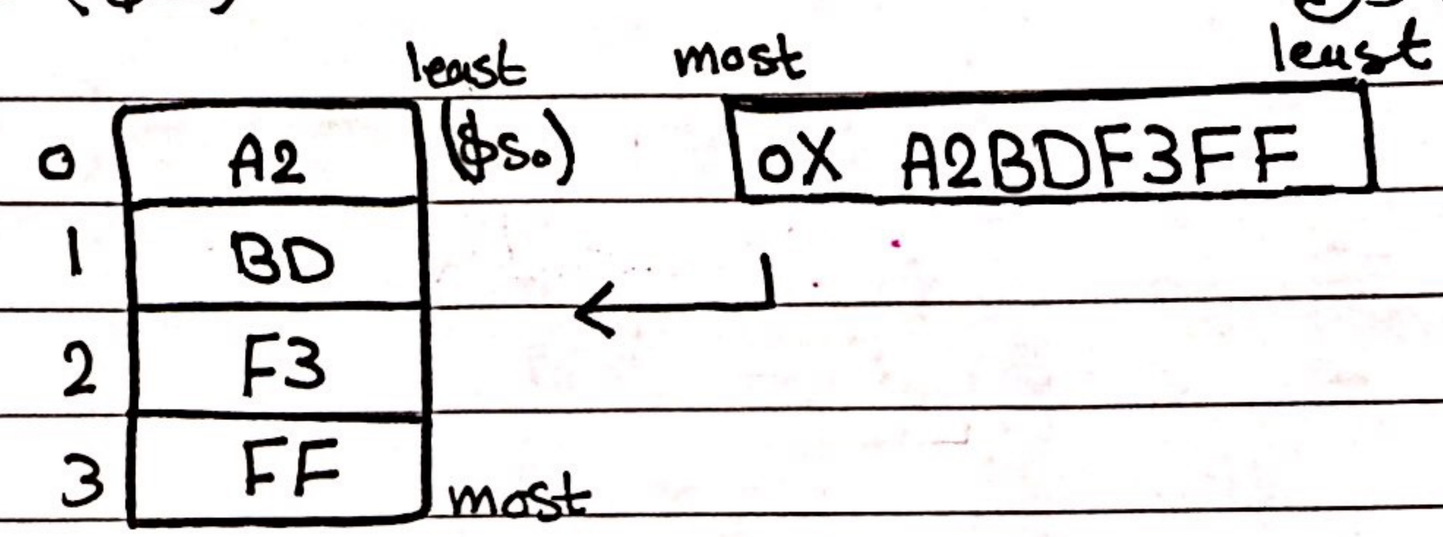
Load
- byte (Lb) 8 bit
- word (Lw) 32 bit
- half (Lh) 16 bit

sw
- byte (Lb) 8 bit
- word (Lw) 32 bit
- half (Lh) 16 bit

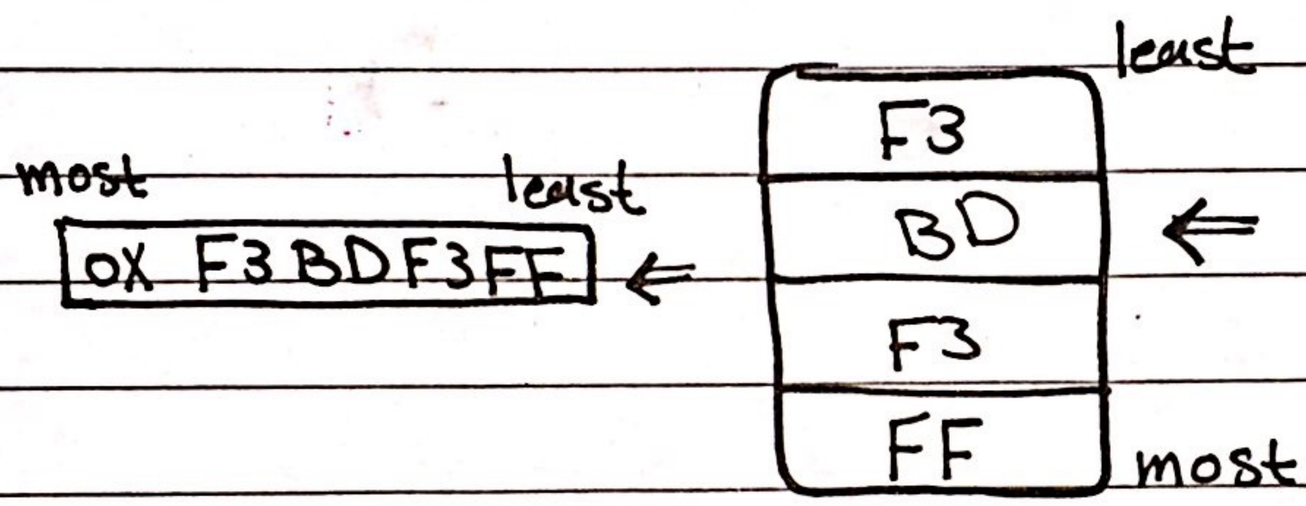
Ex: ① store the following value in memory using big Endian (ox A2 BD F3 FF), ② what is the output of the following code?

```
Lb $t1, 2 ($s0)
Sb $t1, 0 ($s0)
```

* الجزء الأول من السؤال بدي أعبي من الرجستر (ox A2BDF3FF) للميموري.



* الجزء الثاني من السؤال بدي أنفذ البرنامج بدي أعمل (Lb) يعني أعمل (F3) لأنها بالموقع (2+$s0) ومن ثم أخزنها في الموقع (0+$s0) يعني مكان A2.



Instruction Format: أنواع الاستركشن

1- R-format
2- I- format
3- J- format

- كل نوع بيشمل استركشنات محددة بتتوزع جوا الرجيستر بطريقة خاصة لكل استركشن.

1- R-format

بشمل استركشنات كتيرة زي كل عمله وحدة وكيف بتتجمع داخل الرجستر.

6 bit	5bit (rs)	5 bit (rt)	5 bit (rd)	5 bit	6 bit
op code	source 1	source 2	distination	shift amount	Function

تقسيم الرجستر

بشكل عام بقسمهم على الاستركشن بنفس الطريقة:

أرقامهم عبارة عن أسماء رجسترات وإحنا عرفنا انه اسم الرجستر من (5 bit)

- op code : بحدد شو العملية وهو بالـ Rtype = صفر.
- Function : بكون محطوط حسب الاستركشن.
- الـ shift : حسب الاستركشن.

second operand = source2 = rt / first operand = source1 = rs *

* Instructions:

1- add $t0, $t1, $t2 . اجمع محتوى $t1 و $t2 وخزن الناتج بـ $t0.
2- sub $t0, $t1, $t2 . اطرح محتوى $t1 و $t2 وخزن الناتج بـ $t0.

- rs : $t1 ، بروح على جدول الرجسترات وبشوف شو رقم $t1 بالآخر 9 ، بكتبه 01001
- rt : $t2 ، " " " " " " " " " " " " " 10 ، بكتبه 01010
- rd : $t0 ، " " " " " " " " " " " " " 8 ، بكتبه 01000
- shift : صفر 00000 .

* نتعامل مع الـ and و or و nor و slt و nand بنفس تقسيمة الـ add و sub.
* slt $t0, $t1, $t2 (set less than) ← إذا كان $t1 أقل من $t2 خزنه بـ $t0 واحد وغير ذلك بخزن صفر

3_ SLL Reg_1, Reg_2, Shift amount

SLL $\$t_1$, $\$t_2$, 1

• اعمل Shift لليسار للمقدار المخزن بـ($\$t_2$) بمقدار (1) (Shift amount) وخزنها في ($\$t_1$)

Ex: $\$t_2 = 8$ $(0\ldots\ldots001000)$ ⇒ هذا محتوى الريجستر من 32-bit

SLL $\$t_1$, $\$t_2$, 1 بعد ما أعمل (Shift)

$\$t_1 = (0\ldots\ldots0010000)$

* مثل كأني ضربت بـ (2): [كانت $(8 = 0\ldots\ldots1000)$ صارت $(16 = 0\ldots\ldots10000)$]

⇐ تذكير:

$\boxed{1|1|1}$ ← أول Shift لليسار $\boxed{1|1|0}$ ← كمان مرة $\boxed{1|0|0}$ ← كمان مرة $\boxed{0|0|0}$

4_ SRL Reg1, Reg2, Shift

SRL $\$t_1$, $\$t_2$, 1

• اعمل (Shift لليمين) للمقدار الموجود داخل ($\$t_2$) بمقدار (1) وخزنها بـ ($\$t_1$)

Ex: $\$t_2 = 8$ $(0\ldots\ldots001000)$

SRL $\$t_1$, $\$t_2$, 1 بعد ما أعمل الـ Shift

$\$t_1 = (0\ldots\ldots00100)$

* مثل كأني قسمت على 2: كانت 8 وصارت 4.

⇐ تذكير:

$\boxed{1|1|1}$ ← أول Shift لليمين $\boxed{0|1|1}$ ← كمان مرة $\boxed{0|0|1}$ ← كمان مرة $\boxed{0|0|0}$

للـ SRL و SLL

• rs: بالـ SLL و SRL = صفر ، يعني ما إلها first operand

• rt: $\$t_2$ = 10

• rd: $\$t_1$ = 9 • Shift = 1

2- I - Format

بشمل انستركشنات كتيرة ومختلفة عندهم وحدة وحدة.

تقسيم الريجستر:

6 bits	5bit	5 bit	16 bit
op code	rs	rt	constant

• الـ op code هون اله قيمة وكل انستركشن بالـ I-format له (op) خاصة به (محددة).

* Instruction :

1- LW $t0, 1200 ($t1)

2- SW $t0, 1200 ($t1)

* شرحناهم بالتفصيل بموضوع الميموري.

التذكير:-

• $t1 : تمثل الريجستر اللي بحتوي على الـ (Base address).

• $t0 : تمثل الريجستر اللي بحتوي على الـ (real address).

• 1200 : هو الـ offset.

لـ LW والـ SW:

• rs : $t1 = 9 (Base address)

• rt : $t0 = 8.

• constant : وهو الـ offset = 1200

3- addi $t1, $t2, 100

تقرأ add immidiate

* اجمع محتوى $t2 والعدد 100 وخزن الناتج ب $t1.

• rs : $t2 = 10

• rt : $t1 = 9

• constant : وهو العدد الثابت (100) بيكتب بـ 16bit وممكن يكون سالب أو موجب.

ملاحظة: ما في عندي (subi) لأنو الـ constant ممكن يكون موجب أو سالب.

• رح نحكي عن الشرط وهو الـ (conditional) بشمل الـ beq و bnq:

4- beq $t3, $t4, Label
« Branch equal »

• معناها إذا كان $t3 = $t4
إذهب إلى الـ Label وهو عبارة
عن address لـ instruction وإذا
خطأ اذهب لـ (instruction) اللي بعدها.

5- bnq $t3, $t4, Label
« Branch not equal »

• معناها إذا كان $t3 ≠ $t4
إذهب إلى الـ Label وهو عبارة
عن address لـ (instruction) وإذا
خطأ اذهب لـ (instruction) اللي بعدها.

* حَبّيت معناهم وتفهمه أكثر بكتابة الكودات.

Branch

taken

- إذا تحقق جواب الشرط، ويتحقق بالـ beq إذا كانوا $t3=$t4 ويتحقق بالـ bnq إذا كانوا $t3≠$t4
- وهو بروح على مكان بعيد هو (Label) وحكينا انو هو عبارة عن instruction address.

not taken

- إذا ما تحقق جواب الشرط.
- هو بروح على الـ instruction اللي بعده.

• rs : $t3 = 11.
• rt : $t4 = 12.
• constant : Label حكينا انه هو عبارة عن address وانو إذا كان الـ Branch taken حيروح عمكان بعيد فالـ constant حيكون قيمة عدد الاستركشن اللي حيتجاوزهم (لا طريقة حسابها رح نشرحها!!).

* طريقة حساب الـ constant :

ملاحظات مهمة :

PC : Program counter

• هو عبارة عن رجيستر موجود بالـ (CPU) بخزن الموقع (location) الـ (instruction) اللي بدها تنفذ و بعد ما تخلص بخزن موقع التعليمة اللي بعدها (PC + 4).
لأن كل instr. ↓ 4 bit

• يعني لو بدي اتعامل مع الـ (memory) حتى أطلع منها instruction لازم أعمل عن طريق الـ (PC) ← (pointing) على الـ memory location وسمى الـ address.

بده 4bit (مشروحة قبل).

Branch

taken

$pC = (pC + 4) + 4X$

• هذا القانون من خلاله بقدر أطلع قيمة X وهي تساوي الـ constant.

• بعوض مكان الـ PC قبل المناوأة الموقع اللي بدي أوصله والـ PC بعد المناوأة الموقع اللي أنا فيه وبحلها كمعادلة بمجهول واحد وبطلع قيمة X.

not taken

$PC = (PC) + 4$

يعني الـ PC هو اللي الجديد حيكون فيه هو اللي القديم (PC) يعني الموقع اللي أنا فيه + 4، ليش؟!
عشان بنتقل للـ instruction التالية.

• PC اللي قبل المناوأة : الموقع اللي بدي أوصله.

• PC بعد المناوأة : الموقع اللي أنا فيه.

• مثال :-

- البرنامج الواحد عبارة عن عدد محدد من الـ instructions بتنفذ بالترتيب .

Address	instruction	
1000	beq $s1, $s2, L1	
1004	Add R1, R2, R3	PC+4
1008	nand R3, R1, R2	PC+8
1012	sub R2, R1, R3	PC+12
1016	L1: Add R3, R1, R2	PC+16
1020	Exit	

• لو طلب الدكتور مني اني أعبي beq جوّا الـ Text حتكون المشكلة عندي بالـ L1.

• كلمة L1 ممكن تكون أي كلمة بنختارها بشرط اني أوحدها.

كيف رح أحسب؟؟

• يبدأ العد من (PC+4) إلى (L1) عند العد نحسب واحدة منهم وليس كلاهما (إذا حسبت PC+4 ما بجب الـ (L1))

• إذا نزلنا موجب إذا طلعنا للأعلى سالب .

• وحكينا انه (L1) هون هو الـ constant وبكون من 16 بت .

← بعد العد حنلاقي انه بساوي = 3 ((يعني رح يتجاوز ٣ انستركشن))

0000 0000 0000 0011

ممكن أحسبها عالقانون السابق : PC = (PC+4) + 4X

$$1016 = (1000+4) + 4X \Rightarrow \frac{12}{4} = \frac{4X}{4}$$

$$X = 3$$

مثال ٢:

لوكانت هيك :

1000	L2	
1004		
1008		$L_2 = -6$
1012		
1016		
1020	beq R_1, R_2, L2	

• نفس الطريقة بالثت أعد من PC+4 وطلعت لفوق فكانه العد بالسالب وما حسبت L2 لأني حسبت ال PC+4

PC+4

• لو على القانون

$$PC = (PC+4) + 4X$$
$$1000 = (1020+4) + 4X$$
$$-24 = 4X$$
$$X = -6$$

6- slti $t1, $t2, 100 (constant=100)

• إذا كان $t2 أقل من خزن جوا $t1 واحد، غير ذلك خزن صفر.

3- J- Format

6 bit	26 bit
op code	Constant

*J Loop

• op code = هون دائماً يساوي 2.

• constant ← وهو عبارة عن Address على طريقة الحساب (Loop).

رح نشرح عليها بمثال.

Ex :

	Address
Loop: sll $t1, $t2, 2	80000
add R1, R2, R3	80004
lw R1, offset (R3)	80008
sub R4, R2, R3	80012
add R1, R3, R4	80016
j Loop	80020

- ال Loop هو عنوانه 80000 وممكن انه العنوان عبارة عن 32 bit بس المشكلة شو؟! إنه أنا بدي بس 26 بت لأن تقسيمة الرجستر [6bit | 26 bit] شو اللي بصير؟!

بعمل SLR مرتين يعني متل كأني قسمت على 4 وبحذف آخر 4 بت فبصير 26 بت (ولما أعمل SLR ما بزيد صفرين بالآخر).

* باختصار بوخذ عنوان الـ loop اللي هو 80000 وبقسمه على 4 وبصير يساوي 20000.

$$\left(\text{word address} = \frac{\text{byte address}}{4} \right)$$

20000 ↗ 80000 ↗

- في نوع ثاني لـ Jump بكون (R-format) :-

* J $t0

op code	8	0	0	0	Fun

صفر = op code

يكون معطى → Fun

Jump

- j Loop (J-Format)
  * يعطي عنوان مباشر اللي بدي أعمل الـ Jump له.
- jr (R-format)
  * يكون العنوان مخزن داخل الـ register اللي بدي أعمل Jump له

* الفرق بين الـ (beq و bne) و (Jump) :

- الـ Branch : فيها شرط إذا صح وتحقق تذهب إلى الـ label (conditional).
- الـ Jump : ما فيها شرط ، تعمل بدون شرط (unconditional).

رح نبلش نشوف الأمثلة :

What is the mips machine code for this instruction :

add $s0 , $s2 , $s3 → أرقام الريجسترات معطى

rd rs rt

كيف نحل هاي الأسئلة :

١- نحدد نوع الـ Format.

٢- نقسم الانستركشن لـ rs، rt و rd إن وجد.

تكون معطى

op code	rs	rt	rd	shift	Fun
0	18	19	16	0	32

$s0 = 16
$s2 = 18
$s3 = 19

هاي أرقام الريجسترات

* ممكن يطلب نحوله لـ Hexa ، فنحولها لـ Binary بعدين Hexa .

000000 10010 10011 10000 00000 100000

0X 02538020

هلا رح نحل على السريع ... نحدد rs و rt ونعني الريجستر ...

2- addi $s0 , $s1 , 2 "I-format"

rt rs constant (immidiate)

تكون معطى op code	8	17	16	2

3- LW $s0 , 32 ($s1) "I-format"

rt offset rs

35	17	16	32

4- bne $s1, $s2, 100 ⇝ جاهزهو « I-format »

$s1 → rs, $s2 → rt, 100 → constant

5	17	18	100

5- J 1000 « J- format »

2	1000

2 → op code ثابت

6- J $s1 , Fun = 1000 « R- format »

0	17	0	0	0	1000

7- assume we start a Loop at location 80000

Loop: sll $t1, $s3, 2	80000	bne $t0, $s5, Exit
add $t1, $t1, $s6	80004	↳ كيف أحسبها :-
lw $t0, 0($t1)	80008	على القانون - أو العد .
bne $t0, $s5, Exit	80012	
addi $s3, $s3, 1	80016	PC+4 J Loop ⇝ $\frac{80000}{4}$ =
J Loop	80020	PC+8
Exit	80024	20000

sll	0	0	19	9	2	0
add	0	9	22	9	0	32
lw	35	9	8	0		
bne	5	8	21	2		
addi	8	19	19	1		
J	2	20000				

« ليحيى داغ بالكتاب »

* ممكن السؤال يكون بالعكس يعطي الهيكسا وبده نعرف الانستركشن.

0X 02494820

إذا جمع

1- حوله لباينري

2- بوخد أول 6 بت من الشمال حتى أعرف شو نوع الـ format

0000	00010	00000	1000	0100	1001	0100	0000
32	0	9	9	9	18	0	
		r_d		r_t	r_s		

0 ← "R"

2 ← "J"

أي رقم ← "I"

add $t1, $s2, $t1

هلأ نيجي ع كتابة الكودات:

ملاحظات مفيدة:-

1. حتى أصفر قيمة رجستر في عندي طرق مختلفة مثل:-

- Sub $t1, $t1, $t1
- Add $t1, $zero, $zero ← هذا الرجستر قيمته دائماً صفر
- Addi $t1, $zero, 0

2. حتى أخزن قيمة معينة جوا الرجستر أسهل طريقة:

- Addi $t1, $zero, 5 (القيمة / مثلاً 5)

3. أخلي رجستر = رجستر ثاني

- Add $t1, $t2, $zero.
- Addi $t1, $t2, 0.
- Sub $t1, $t2, $zero.

4. لما بدي أعمل أي عملية ع قيمة معينة لازم أخزنها جوا رجستر يعني أجلبها من الميموري عن طريق LW وتكون قيمة معروفة مثل:

A[7] + E ، هساهون ممكن يكون محددلي انه $s2 = E ، يعني مخزنة بالرجستر $s2

بس A[7] مش محدد هي وين و أخذنا قبل كيف نعمل LW وحسب الـ offset ،

فإذا (offset = 28) والحرفية العامة للـ LW بنعرفها وهو نحدد الـ Base address

(index) ← 7 × 4 → (word)

ولنفترض إنه كان $s1 ، ومثلاً بدي أحملها جوا الرجستر $t2 . . ، كيف حتصير؟

→ LW $t2 , 28 ($1)

← بعدين بعمل عملية الجمع ونختار وين أخزن قيمة الجمع لو كان ب $s3 :

add $s3 , $t2 , $s2

٥- لو كانت الـ index بالـ array مش معروفة مثل (i) هون :

J + B[i] ، بكون محدد لي انه الـ J جوا الرجستر $s1 ، وحتى لو كانت مش جوا رجستر بقدر أحملها عن طريق LW و بكون معطيني الـ Baseaddress والـ offset بحطه هون يساوي صفر.

بس المشكلة هون بالـ (array) أنا بدي القيمة اللي جوا B[i] ، بعطيني $s1 = Baseaddress والـ (i) جوا رجستر $t1 ، فلازم أعمل LW فبدي أحسب الـ offset بالنقطة (4) فحسبته بالضرب العادي (4x7) بس هون لازم أضرب (i x 4) = ($t1x4) لأنه (i) جوا $t1 مثل ما حددنا عن سبيل المثال، وأخدنا انسترکشن بتقوم بعملية الضرب اللي هي (SLL) ونختار وين أخزنه (4i) لأنه حتصير بتمثلي الـ offset :

- SLL $t2 , $t1 , 2 # (4i)

لو بدي أعمل LW على طول هيك حيصير :-

- LW $t3 , $t2 ($s1)

$t3 ↓ اخترنا مكان للتخزين ، $t2 ↓ offset ، $s1 ↓ Base

طبعاً هذا ما بصير لأنه الـ offset لازم يكون رقم (constant) فبعمل حركة صغيرة :-

- لو راجعت موضوع الـ LW حتلاقي انه صرنا نعرف انه الـ offset والـ Base address بنجمعوا مع بعض وبشكلوا الـ (real address) بعدين بعمل LW . هون رح أعمل هاي الحركة أنا فنشو حيصير :

اسمها اختياري

Add $t3 , $t2 , $s1 # $t3 = real address

$t3 ↓ مكان لتخزين real address ، $t2 ↓ offset ، $s1 ↓ Base

أنا اخترت $t3 ممكن أختار أي رجستر ثاني وأخزن فيه العنوان كامل..

← هسا بعمل LW

• LW $t4, 0 ($t3) # $t4 = mem [0 + $t3] = $t3

← عملت هاي الحركة عشان أحط الـ offset رقم (constant) والصفر لما بنجمع مع الـ real address ما حيغير قيمته.

ملاحظة:

في فرق بين (i)، a[i] / a[i]: يعني القيمة اللي جوا a[i] مثلاً لو كانت (i) = 1 فهذا لا يعني انه a[1] = 1، ممكن a[i] = 3 أو a[1] أي قيمة أخرى / بس لو حكيت (i = 1) يعني ما في احتمال ثاني لـ (i) إلا إنها تكون واحد عشان هيك بكتابة الكودات بتعامل معهم لما أجلبهم من الميموري بطريقة مختلفة.

٦. لحل الأسئلة بعتمد على الرجسترات اللي هو حددلي إياها، وبفرض رجسترات ثانية لما احتاجها لحل السؤال.

Ex 1: A[12] = h + A[8]

h = $s1

base address = $s0

لما السؤال يكون بالعادة مكتوب بالـ C++ وبطلب تحويله لـ (mips).

← السؤال بيحكي حمل A[8] واجمع إلها (h) وخزنهم بالموقع A[12] عشان هيك الـ SW.

Sol: LW $t0, 32 ($s0) # $t0 = mem [32 + $s0]
Add $t0, $t0, $s1 # $t0 = h + A[8]
SW $t0, 48 ($s0) # mem [48 + $s0] = h + A[8]

٧. إذا كان السؤال فيه شرط مثل (IF) أو غيرها بستخدم بالحل bnq أو beq حسب السؤال، وبعكس الشرط اللي عندي ليش؟
رح نعرف بعدين، بس لما أعكس الشرط لازم أتأكد إنه البرنامج بعطي نفس النتيجة. «رح تتوضح بمثال»

Ex1: IF (i == j) f = g+h f = $s0 g = $s1
else f = g-h h = $s2 i = $s3
j = $s4

* بنلاحظ انه جميع المتغيرات بالسؤال موجودة جوا رجسترات يعني ما بحتاج أعمل lw.

* حكينا انه لازم نعكس الشرط هون الشرط (i هل تساوي j) فبنخليها لا تساوي يعني (bnq).

```
* bnq $s3, $s4, else     # go to else if i≠j.
  Add $s0, $s1, $s2      # $s0 contains g+h.
  j Exit                 # next instruction in Exit.
else: Sub $s0, $s1, $s2  # $s0 contains g-h.
  Exit                   # End program.
```

← لازم جنب كل انستركشن أحط (comment) أبيّن فيها أنا شو عملت.

← هون أول اشي رح يسأل إذا كانت (i ≠ j) رح يروح على مكان بعيد اللي هو else لأنه الـ Branch taken، فحينفذ الـ sub، وبعدين Exit.

وإذا كان (i = j) رح يروح على الـ instruction التالية لأنه Branch not taken فحينفذ الانستركشن Add، بعديه (j Exit) يعني حيروح على Exit.

← بنلاحظ انه لو تتبعت البرنامج المكتوب بالـ C++ لازم يكون اللي كتبته بعطي نفس النتيجة.

← لو كان السؤال IF (i != j) بصير الشرط beq $s3, $s4, else.

Ex2: while (save[i] == k) و i += 1

السؤال: بحكي طول ما ال k = save[i] أزيد عال [i] واحد.

i = $s0

k = $s1

Base address $s3

⇐ حبينا نعرف كيف نعمل LW لعنصر بال (array):

```
Loop: Sll $t0, $s0, 2        # 4i = offset
      Add $t0, $t0, $s3      # real address = 4i + $s3 = $t0
      Lw  $t1, 0($t0)        # $t1 = mem[0 + $t0]

      bnq $t1, $s1, Exit     # IF $t1 ≠ $s1 go to Exit
      Addi $s0, $s0, 1       # i = i+1
      J Loop                 # next instruction at Loop
Exit: Exit                   # End program
```

* حمّل العنصر الأول بال array وقارنه مع k إذا كان ما بساوي ال k حيطلع برا البرنامج لأنه حيكون Branch taken خروج ملكان بجيه، وإذا كان بساوي ال k حيكون ال Branch not taken خروج عال الانستركشن التالية اللي هي Addi ← حيجمع ال (i) واحد (يعني العنصر اللي بعده) بعدين حيعمل [J] عال ال Loop ويرجع تحمل العنصر اللي بعده (يعني بعد الأول) وهكذا... إلى أن يصبح ال k ≠ العنصر (save[i]).

Ex3: X^5 $X = s_{s0}$

مثال للفهم:

```
add $t0, $s0, $s0   # $t0 contains $s0+$s0      2+2=4
add $t0, $t0, $t0   # $t0 "      $t0 + $t0     4+4=8
add $t0, $t0, $t0   # " "        "      "       8+8=16
add $t0, $t0, $t0   # " "        "      "       16+16=32
```

↳ وهو المطلوب

Ex4: int A[100], B[100];
for (i=1; i<100; i++)
A[i] = A[i] + B[i];

i = $t1
base address of A = $s2
base address of B = $s3

```
addi $t0, $zero, 100      # $t0 = 100
addi $t1, $zero, 1        # i = 1
Loop: sll $t2, $t1, 2     # offset = 4i
  Add $t2, $t2, $s2       # real address of A = 4i + $t2
  Lw $t3, 0($t2)          # $t3 = mem [0+$t2]

  sll $t4, $t1, 2         # offset = 4i
  Add $t4, $t4, $s3       # real address of B = 4i + $s3
  Lw $t5, 0($t4)          # $t5 = mem [0+ $t4]

  add $t6, $t5, $t3       # $t6 = A[i] + B[i]
  sw $t3, 0($t2)          # A[i] = A[i] + B[i]

  addi $t1, $t1, 1        # i = i+1
  beq $t1, $t0, Exit      # IF i = 100 go to Exit.

  j Loop                  # next instruction at loop
Exit;                     # End Program
```

(المقصود انه اللي رقمه A[i])

بهاد السؤال بده يجمع أول عنصر من ال array اللي اسمها A مع أول عنصر من ال array اللي اسمها B وبعدين يخزن الناتج في (A array) ويكرر العملية هاي يعني يجمع الثاني مع الثاني والثالث مع الثالث و..... لحد ...100 عنصر وفي كل مرة نخزن الناتج في ال (A array) وبما انه كان يزيد على (i) واحد في كل مرة وما يكون (i) أقل من (100) برجع لل (Loop) وإذا ...100 أو أكثر

بنتهي البرنامج ، ونلاحظ هون انه عملت (sll - add - lw) مرتين مع إنه المتغير (i) نفسه ، لأنه عندي 2 arrays ، لكل array بعمل (sll - add - lw).

← وبالنسبة لأول جملتين addi الأولى بحدد فيها أكبر عنصر بدي أوصله وحطيته جوا الريجستر عشان أقدر أقارنه ، و addi الثانية فهي حددتلي بداية العداد اللي هي واحد .

← هلا رح نفترض وجود تغيرات بالسؤال السابق :

- لو كان نفس السؤال بس طالب : $A[i] = B[i] + A[i-1]$

أول جملتين وآخر أربع جمل بضلوا نفس ما هي اللي حيتغير :

حل 27

```
Loop: sll $t2, $t1, 2          # offset = 4i
      Add $t2, $t2, $s3        # real address of B = 4i + $s3
      LW  $t3, 0 ($s3)         # $t3 = mem [0 + $s3]

      sll $t4, $t1, 2          # offset = 4i
      Add $t4, $t4, $s2        # real address of A = 4i + $s2
      LW  $t5, -4 ($t4)        # $t5 = mem [-4 + $t4]

      Add $t6, $t5, $t3        # $t6 = B[i] + A[i-1]

      sw  $t6, 0 ($t4)         # mem [0 + $t4] = $t6
```

- بنلاحظ إنه الـ comment اللي جنب الانستركشن بتختلف شوي عن الحل بالسؤال اللي قبل والسبب إنه هاي الـ comment حطها أنا عشان أفسر فيها اللي بصير فبختار أكتب اللي بدي إياه بس المهم يكون صحيح .

• بهاد السؤال بده يجمع أول عنصر من الـ (array) اللي اسمها B مع العنصر اللي قبل الأول يعني (A[i-1]) وبكرر هاي العملية لحد ١٠٠ عنصر....

• هوي لما خزن الناتج عمل : SW $t6, 0($t4)

بس التحميل : LW $t5, -4($t4)

بالـ (LW) حطيت الـ offset = -4 لأنه أبا بدي العنصر السابق A(i-1)

ولما رجعت بدي أخزن خزنت بالـ offset = صفر لأنه أنا بدي A[i]

	memory $s4
-4 ← عمل من هون	
0 ← خزن هون	
4	
8	

• فكرة كتابة الكودات إن شاء الله إنها تكون وضحت .

• هلا رح نحل كم مثال للتدريب :

Ex5: for (i=1, i<=9, i++)

sum = sum + a[i]

i = $t1 , sum = $t0 , Base address = $s1

<u>sol:</u>

```
Addi $t3, $zero, 10      # $t3 = 10
Addi $t0, $zero, 0       # sum = 0
Addi $t1, $zero, 1       # i = 1
Loop: sll $t1, $t1, 2    # 4i
      Add $t2, $t2, $s1  # 4i + $s1
      Lw  $t4, 0($t2)    # $t4 = mem[0 + $t2] = a[i]
      Add $t0, $t0, $t4  # sum = sum + a[i]
      Addi $t1, $t1, 1   # i = i + 1
      bnq $t1, $t3, Exit # IF i = 10, go to exit
      J Loop             # next instruction at loop
      Exit               # End program.
```

Ex6:

```
i=1;
j=5;
while (i != j)
{ ++i;
  --j; }
```

$i = \$t_0$

$J = \$t_1$

Sol:

```
        Addi  $t0, $zero, 1      # i=1
        Addi  $t1, $zero, 5      # J=5
Loop:   beq   $t0, $t1, Exit     # IF (i != J) go to Exit
        Addi  $t0, $t0, 1        # i=i+1
        Addi  $t1, $t1, -1       # J=J-1
        J     Loop
        Exit
```

Ex6: $F = A+B+C+D$

- Base address = $\$s_3$

- مش عاملهم LW ولا حاطهم بتسجيلات
لازم أنا أعمل (LW)

```
LW  $t0, 0 ($s3)   # $t0 = mem[0 + $s3] = A
LW  $t1, 4 ($s3)   # $t1 = mem[4 + $s3] = B
LW  $t2, 8 ($s3)   # $t2 = mem[8 + $s3] = C
LW  $t3, 12 ($s3)  # $t3 = mem[12 + $s3] = D
Add $t4, $t0, $t1  # $t4 = A + B
Add $t5, $t4, $t2  # $t5 = A+B+C
Add $t6, $t5, $t3  # $t6 = A+B+C+D
SW  $t6, 16 ($s3)  # mem[16+$s3] = $t6 = F
```

Ex7:

temp = V[k]
V[k] = V[k+1]
V[k+1] = temp

- Base address = $s2
- k = $s3

```
sll  $t0, $s3, 2      # offset = 4k
Add  $t1, $t0, $s2    # real address = 4k + $s2
LW   $t2, 0($t1)      # $t2 = mem[0 + $t1], V[k]

LW   $t3, 4($t1)      # $t3 = mem[4 + $t1], V[k+1]
SW   $t2, 4($t1)      # mem[4 + $t1] = $t2, V[k] = V[k+1]
SW   $t3, 0($t1)      # mem[0 + $t1] = $t3, V[k+1] = temp
```

هي نفسها V[k]

- بلاحظ إني عملت (sll ← Add ← LW) مرة وحدة لأنها مصفوفة وحدة
- بلاحظ إني عملت LW أول مرة وحطيت الـ offset = 0 والمرة الثانية حطيته = 4، لأنا (V[k+1]) يعني العنصر اللي بعده، عكس السؤال السابق اللي كان الـ offset فيه = -4.

⇐ بالأسئلة اللي فيها array و for لما عند معين لما أزيد [i] انتبه إذا كان A[i] و B[i] و X و A[j] و B[i] لأنه إذا كانوا مختلفات لازم لما أزيد واحد أعمل Addi لـ j و Addi لـ i.
(راجع Ex4).

ملاحظات مهمة للحفظ : « الدكتور ممكن يحدد أشياء ثانية.. ركزوا »

- Design Principles :

1- simplicity favors regularity.
2- smaller is faster.
3- Good Design demands good compromises.
4- make common cases fast.

- Instruction Addressing modes :

1- PC-Relative addressing (beq , bne).
2- Peseudo-direct addressing (Jump , Jump and link).
3- Register addressing (add , sub).
4- Base addressing (LW , SW).
5- Immidiate addressing (addi).

- Procedure

• عادة مابيجي عليه الاسؤال عدد خطوات عادة يعني :

→ Steps of Procedure: حفظ يحبيم !

1. place parameters in aplace where the procedure can access them. مكان حجز input ($a_0 - $a_3)

2. transfer control to the procedure.
تغيير PC لتنفيذ الانستركشن الثانية ولازم أحفظ من عند المكان اللي طلعت منه
PC + 4 في $ra.

3. Acquire the storage resources needed for the Procedure.

4. Perform the desired task. تنفيذ برنامج معين

5. place the result where the calling program can access it.
بعد تنفيذ الـ task نرجع قيم سوف نخزنها في ($v_0 - $v_3)

6. Return control to the point of origin.
نعود للنقطة التي خرجنا منها من أجل تنفيذ الـ task

• الريجستر المستخدمين في الـ Procedure :

$a_0 - $a_3: save input parameters
$v_0 - $v_1: return value.
$ra : return to the point of origin.

• البرنامج الذي يستدعي يسمى caller
• " " يستدعى يسمى calling

⇐ رح نحكي عنها شوي بطريقة مبسطة ومختصرة يعني "الفكرة العامة".

12	instruction 1
16	instruction 2
20	Jal max
24	instruction 3

رح يبدأ البرنامج عادي بنفذ 12 و 16 ولما يصل إلى Jal max تصبح قيمة الـ PC=60 وهو الموقع الجديد بعد عملية الـ Jump، بس المفروض انه الـ PC يكون PC=PC+4=24 بس عشان الـ Jump الـ PC=60 وعشان أخزن محافظة على العنوان 24 وما يحتاج بعد الـ Jump بخزنه جوا $ra وبعدها (push) وما يوصل لـ 68 رح يعمل (Jump) لـ proc ونفس الاشي حيصير الـ PC=80 وتخزن PC=PC+4=72 جوا $ra وبعدها (push):

max: 60	push
64	instruction 6
68	Jal proc
72	pop
76	Jr $31

قبل ما يطلع من البرنامج رح يعمل pop بخرج العنوان 72 ثم 24.

72
24

proc: 80	push
84	instruction 11
88	pop
92	Jr $31

• طبعاً الـ pop والـ push عبارة عن كودات معينة تكتب بالـ mips instr.

⇐ هاد كل اشي قدرت أفهمه وأجمعه عن الـ Procedure، مش غلط إذا فهمتوا أكثر.

CH4: Binary Arithmetic and ALU Design:

• Number types:

1- Unsigned integers : ما بهتم للإشارة هو

2- Signed integers : بهتم لاشارة الرقم هو، إذا كان آخر بت بالرقم (0) يكون موجب إذا كان (1) يكون سالب

• Signed Extension :

- لما بدي أعمل أي عملية حسابية لرقمين لازم يكونوا الاثنين 32 بت.
- إذا كان واحد أقل بشوف :-

① إذا كان sign إذا آخر بت واحد بكمل 32 بت واحدات.
" " " صفر " " " أصفار.

② إذا كان unsign بكمل 32 بت أصفار.

Ex: addi ⇒ sign , addiu ⇒ unsigned

bits عدد الـ: n — -2^{n-1} to $2^{n-1}-1$: Rang sign

bits عدد الـ: n — 0 to 2^n-1 Rang unsign

• الـ Rang لـ 3 بت (sign) هو ← -4 to 3 أي أكبر رقم ٣ وأقل رقم -4.

لو جمعت ٣ مع ٢ ⇐ المفروض يكون الجواب (5) بس هو الجواب (-٣) وهنا خطأ (overflow)

```
  011
+ 010
-----
  101
```

• إذا صار عندي overflow الـ Mips prosessor بتعمل Exception ← توقف العملية وتعطي مسج انه صار عندي overflow

مثال على كتابة الأعداد :

إذا كان العدد unsigh ⇐ 0 ← 00000

1 ← 00001

15 ← 01111

إذا كان العدد sing ⇐ 0 ← 000000

1 ← 000001

2 ← 000010

آخر بت = 0 إذا موجب

-1 ← 000001 ← 111110

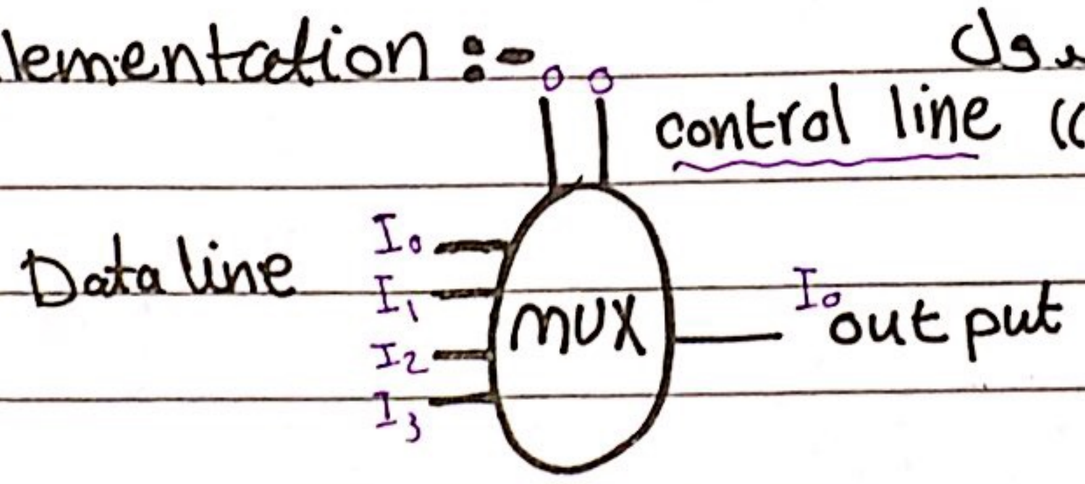
+1

111 1111111

① بكتب العدد موجب ② بوخذ الـ 2nd complement.

• ALU implementation :-

مطلوب حفظ الرسمات + الجدول

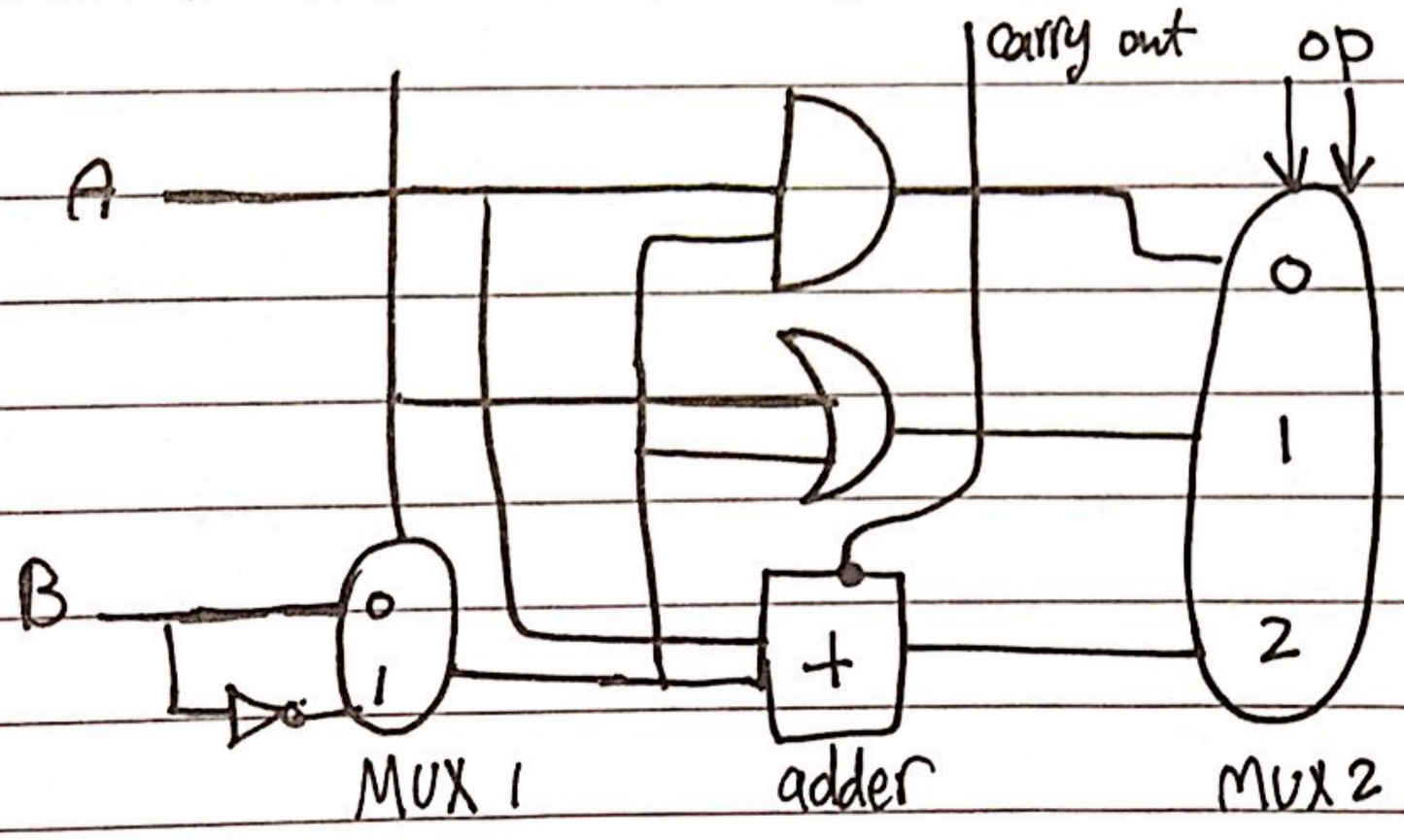


• نختار مدخل معين يطلع بالـ output

وتحدد الـ S.S ((control line)) على أساسها

$n = S.S = 2$

2^n input $= 4$



((One Bit ALU))

تقوم بعدة عمليات حسب الـ op المحدد ((مطلوب حفظهم))

operation	op	number
And	0000	0
OR	0001	1
add	0010	2
sub	0110	6
NoR	1100	12
Slt	0111	7

«لتسهيل الحفظ»

• ممكن نتكلم عن بعض الأوامر مثل

and ⇐ 0000 حيدخل على الـ

op صفريه وعند الـ MUX1 يختار (0)

فحيطلع (B) وحتصير عملية الـ (and)

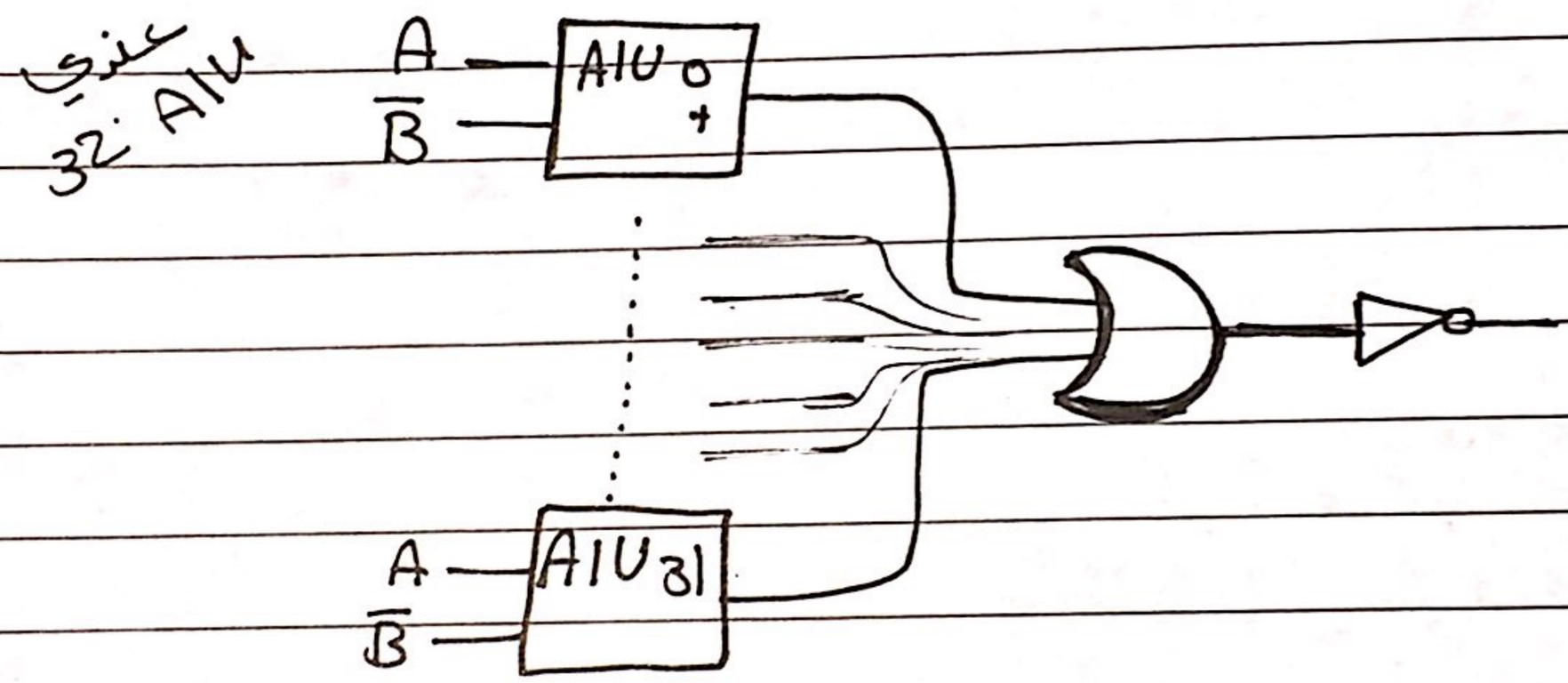
A−B

sub ⇐ من MUX2 نختار (2) لأنها

adder و MUX1 نختار (1) حتى يكون

−B ، وهكذا...

2 Branch instruction



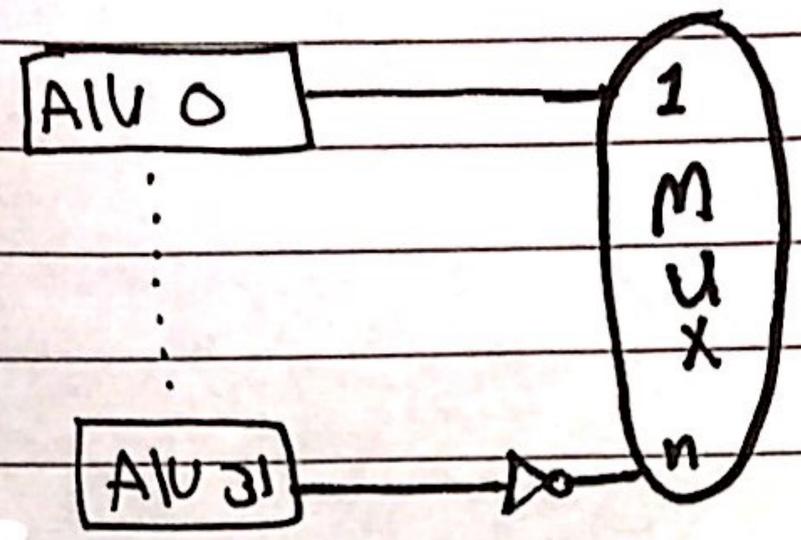
لو بده يقارن بين A و B :

* بودي A و B بطرحهم إذا كانوا متساويين حيطلع 32 بت صفر والـ OR حيكون ناتجها zero وبعد invertor فبصير 1 إذاً (taken)

* إذا غير متساويين حيطلع سلسلة من الأصفار والواحدات فبطلع 1 والـ invertor خليها 0 يعني (not taken)

تذكير: الـ A والـ B عبارة عن سلسلة من واحدات وأصفار بتكون من 32 بت لأنها جوا رجيسترات، بعد عملية الطرح نتج 00000000 حيدخلوا على الـ OR 32 صفر حيكون ناتجه صفر وبعد الـ invertor بصير 1.

3 slt instruction



نقوم بهذه التعليمة بعملية sub
نأخذ (carry out) من آخر ALU
ونخله على (invert) ثم
ندخله على MUX.

slt → A − B
slt = 1 A > B or B = A
slt = 0 A < B

هاد أكثر قسم معكم من المعلومات اللي قدرت أحمّلها بخصوص ال ALU.

- Multiplication and Division عمليات الضرب والقسمة.

Multiplicant ← 6 X 5 → Product الضرب

رح أحط عملية الضرب هاي 6X5 أحلها، ورح أشرح عليها.

Iteration	multiplicant	Product	Remark
0	0110	00000101	initial state
1		0110 0101	L(P) = L(P) + M
		0011 0010	P >> 1
2		0001 1001	P >> 1
3		0111 1001	L(P) = L(P) + M
		0011 1100	P >> 1
4		0001 1110	P >> 1

• بالـ initial state اللي هي البداية بحط الـ product وبكمله أصفار حتى يحمل ضعف الـ Multiplicant يعني الـ P = 8بت ، M = 4بت.

• بشوف إذا كان الـ (P) أوله على اليمين واحد ، بجمعه الـ (M) فقط لليسار تبعه يعني لآخر 4بت فقط (L(P) = L(P) + M) وبعدين بعمل Shift right لكل الرقم.

• إذا كان الـ (P) أوله على اليسار صفر بس بعمل Shift right.

← 5 خطوات. (0 ← 4) • ممكن نحول آخر اشي طلع معنا لعشري حتى نتأكد.

هلا نيجي للقسمة

القسمة 6 ÷ 5 → Divisor ← Reminder

Iteration	Divisor (5)	Reminder (6)	Remark
0	0101	0000 0110	initial state
		0000 1100	R << 1
1		1011 1100	L(R) = L(R) - D
		0000 1100	undo
		0001 1000	R<<1
2		1100 1000	L(R) = L(R) - D
		0001 1000	Undo
		0011 0000	R<<1
3		1110 0000	L(R) = L(R) - D
		0011 0000	
		0110 0000	

Iteration	Divisor	Reminder	Remark
4	0101	0001 0000 0010 0001	L(R) = L(R) − D R<<1 , R = 1
Extra.		0001 0001	R(R)>>1

• بالـ (initial state) اللي هي البداية بحط الـ Reminder وبكمله أصفار حتى يصير ضعف الـ Divisor وبعمل Shift left لكل العدد للـ Reminder والـ Divisor تخليه مثل ماهو.

• بكل مرة بطرح من اليسار تبع الـ (R) مقدار الـ (D) وإذا كان ناتج الطرح سالب بعمل (undo) وبعديه بعمل Shift left لكل العدد، إذا كان ناتج الطرح عدد موجب ما بعمل undo بعمل Shift left لكل الرقم وخلّي أول خانة من العدد (يمين) تساوي واحد.

• بكرر هاي الخطوات خمس مرات (4 ← 0 initial) وفي خطوة (Extra) بعمل Shift right للجهة الشمال من الرقم عشان يطلع الجواب الصح.

0001 0001
الناتج الباقي

• لما الحاسوب بضرب عددين كل واحد ٣٢ بت (mult) الناتج بكون 64 بت وكونه الريجستر نظام ٣٢ بت بوسع بس ٣٢ بت فأول ٣٢ بت بتخزنوا بالـ lo register وآخر ٣٢ بت بتخزنوا بالـ Hi register

• الأمر اللي بخزن أول ٣٢ بت هو (mflo)
• الأمر اللي بخزن آخر ٣٢ بت هو (mfhi)

Ex: (سنوات), Assume MIPS use BigEndian for the byte order convention in this problem. The following Figure shows the memory with the corresponding addresses.

1- What the contenc of registers $t0, $t1, $s1, $s2 after execution the program?

2- Show the content of the same trunk of memory after the execution the program?

Base address حواصم

$s0 = 0 , $s1 = 1
$s2 =

Befor

Address	Content	
ox 7	ox 01	
ox 6	6 x00	
ox 5	ox00	lw $t0, 0($s0)
ox 4	ox00	lw $t1, -9($s2)
ox 3	oxod	mult $t1, $t0
ox 2	o x00	mflo $s1
ox 1	ox 00	sw $s1, 0($s1)
o x 0	ox00	mfhi $s2

after

Address	content
ox 7	ox 01
ox 6	ox00
ox 5	ox00
ox 4	ox00
ox 3	ox00
ox 2	ox00
ox 1	ox00
o x 0	o x 00

Sol: ②

$t0	00 00 00 od
$t1	00 00 00 01
$s1	00 00 00 00
$s2	00 00 00 od

Sol: ①

① رح أعمل LW من الميموري من الموقع (0 + 0) = 0 (يعني رح أعمل
Base offset
٣٢ بت من ال least تبع الميموري لل most تبع الريجستر) يعني
زي ما حكينا بشيل ٤ بلوكات وخزنهم بال $t0

② رح أعمل LW من الميموري من الموقع (13 + -9) = 4
بشيل ٤ بلوكات وخزنهم بال $t4

③ رح أضرب $t0 * $t1 ويكون الناتج 64 بت (00.....00000000d)
أصفار

④ نخزن أول ٣٢ بت بال $s1

⑤ نخزن من $s1 للعنوان بالميموري (0 + 1) = 1 (يعني رح أخزن من
ال most تبع الريجستر لل least تبع الميموري)

⑥ نخزن آخر ٣٢ بت بال $s2

← تنبيه:

← انشاء الله يكون الحل صح! لأنه بالسؤال الدكتور ما كان محدد
ال Base address أنا فرضتهم من عندي! لو تأكدتوا من الحل بكون
أفضل.

لأنه السؤال تكرر مرتين على نفس المبدأ فتأكدوا من الدكتور.

• Floating point

← الأرقام الصحيحة بقدر أكتبها بالكمبيوتر وماني مشكلة
المشكلة عنا بالأرقام العشرية ألها طريقة لكتابتها وح نستخدم نظام :

The IEEE 754 binary representation

→ single precision
→ Double precision

ح نحكي عن شوية ملاحظات بعين كل الأمثلة :

① التحويل من عشري لـ Binary :

العدد 188 و 5

العدد الصحيح سهل . ..... و 101

الكسر (188 و ...) ← $0.376 \times 2 = \underline{\underline{0}}.376$

$0.752 \times 2 = \underline{\underline{0}}.752$

$0.504 \times 2 = \underline{\underline{1}}.504$

$0.008 \times 2 = \underline{\underline{1}}.008$

$0.016 \times 2 = \underline{\underline{0}}.016$

, 00110 (for five digit)

* طبعاً أنا بوقف بس يصير العدد مرفوع بس هاد مجرد مثال عشان نفهم كيف نحول .

العدد النهائي : 101 و 00110

② التحويل من Binary لـ عشري :

العدد 111 و 01101

- العدد الصحيح سهل : ..... و 7

- الكسر :

$$0\,1101 = (0*2^{-1}) + (1*2^{-2}) + (1*2^{-3}) + (0*2^{-4}) + (1*2^{-5}) = 0.40625$$

2^{-1} 2^{-2} 2^{-3} 2^{-4} 2^{-5}

العدد النهائي : 7,40625

③ في عندي اشي اسمه Normalized scientific وهي إني أكتب العدد بال Binary بس أخلي قبل الفاصلة عدد واحد وقيمته كمان تكون واحد مثل:

$$11.0010 = 1.10010 * 2^{+3}$$

$$0.0001 = 1.0 * 2^{-1}$$

④ بما إني حستخدم نظام IEEE 754 بتكون كتابة العدد كما يلي:

1 bit	8 bit	23 bit	(Single)
sign	Exponent	Fraction	32-bit

1bit	11 bit	20 bit	32 bit	(Double)
sign	Exponent	Fraction	Fraction	64-bit

- رح نستخدم بالحل ال (single)

- sign : بتكون (1) إذا كان العدد سالب و (0) إذا كان العدد موجب.
- Exponent : بعد ما أخلي قبل الفاصلة رقم واحد بشوف شو صارت القوة $2^{\square}$ وبجمع للقوة □ + 127 إذا كانت single أو 1023 إذا كانت double، وبمثل الرقم اللي طلع معي من 8 بت إذا single أو 11 بت اذا كانت double.
- Fraction : هو ما بعد الفاصلة كاملاً والباقي بحطه أصفار.

⑤ رح نستخدم القانون هاد إذا كان الرقم الموجود بال Binary وبدنا نحوله لعشري بنظام ال IEEE754.

$$(-1)^{S} * (1+F) * (2^{E-127})$$

S: sign F: Fraction E: Exponent ← طبعاً بالعشري.

⑦ طريقة الحل :

أ- لو أعطاني الرقم عشري وبده أكتبه بنظام الـ IEEE754 ،
رح أحوله لـ Binary وأعمله normalize وبعدين بكتبه هون :

sign	Exponent	Fraction

مثل : 12.5

12 : 1100 , ...

0.5 : 0.5 * 2 = 1.0

0,0 : 0.0 * 2 = 0.0

12.5 = 1100,1

- normalize : $1,1001 * 2^{+3}$
- sign : 0
- Exponent : 3 + 127 = 130 ⇐ 1000 0010
- Fraction : 1001..... و

تذكير الأحرف بالـ Hexa :
A : 10 , B : 11
C : 12 , D : 13
E : 14 , F : 15

0	1000 0010	1001 0000 0000 00
1 بت	8 بت	23 بت

ب- لو أعطاني الرقم (Hexa) رح يطلب مني أحوله لعشري باستخدام نظام الـ IEEE754 ، فبحوله لـ Binary وبقسم الأرقام حسب عدد البتات ورح أستخدم ← القانون (5) .

مثل : 0XC1D00000

$\underbrace{1}_{sign}\ \underbrace{100\ 0001}_{Exponent}\ \underbrace{1010000\ 0000\ 0000\ 0000\ 0000}_{Fraction}$

$(-1)^S * (1+F) * (2^{E-127}) =$

$(-1)^1 * (1+\frac{5}{8}) * 2^{131-127} = -26$

1 = sign

131 = E

$1*2^{-1} + 0*2^{-2} + 1*2^{-3} = 0.625 = \frac{5}{8}$ = F

أسئلة :

How to present A number in IEEE 754 32-bit floating point notation.

1- $-11011,01101001 \cdot$

$-1,101101101001 * 2^4$

• 1 : sign

• $4+127=131$: E

• 10110110100 : F

131

1	1000 0011	10110110100 0000000000

2- -0.5

• 1 : sign

• $-1+127=126$: E

• 0 : 0000

• -0.5 : $0.5*2=1.0$ → $0.0*2=0.0$

• 90 = F

$-0.5 = 0000,10$

$1,0 * 2^{-1}$

126

1	0111 1110	0000 0000 0000 0000 0000 000

3- -13.6

-13 : 1101

0.6 : $0.6*2=1.2$

$0.2*2=0.4$

$0.4*2=0.8$

$0.8*2=1.6$

$0.6*2=1.2$

الرقم بعيد نفسه

يعني دوري يوقف

هون ويكرر النمط

$\Rightarrow 0.6 = 1001100110011 \ldots$

$-13.6 = -1101,1001100110011\ldots$

$-1,101100110011\ldots * 2^3$

1 : sign

$3+127 = 130$: E

F : 01100110.....

130

1	1000 0010	10110011001100110011001

4- 3,75

3 : 0011

,75 : $0.75 * 2 = 1.5$

$0.5 * 2 = 1.0$

,75 = ,11

$\boxed{3,75 = 11,11 \quad 1,111 * 2^1}$

- sign : 0
- E : $1+127 = 128$
- F : 1110

128

0	1000 0000	1110 0000 0000 0000 0000 0000

* ممكن السؤال ييجي بطريقة أخرى :-
يعطيني إياه بال IEEE 754 و يطلب الرقم :
⇐ مثل :

بعطيني هاد

0	0 111 1110	1000 0000 0000 0000 0000 000

ما بعد الفاصلة

نحلها عن القانون :

$(-1)^s * (1+F) * (2)^{E-127}$

$(-1)^0 * (1+0.5) * 2^{126-127} = 0.75$

$s = 0$

E : 126

F : 100.... و

$1 * 2^{-1} = 0.5$

• وحكينا ممكن يعطيني الرقم بالـ Hexa

مثل : 0X 3F80 0000

0 011 1111 1000 0000 0000 0000 0000 0000

$(-1)^{s} * (1+F) * (2^{(E-127)})$

0 : sign

126 : E

0 : F

$(-1)^{0} * (1+0) * 2^{126-127} = 0.5$

CH3: Assessing and understanding performance:

- في هاد الشابتر رح يكون عندي عدة قوانين تتعلق بأداء الجهاز والسرعة وزمن تنفيذ العمليات بالـ CPU.

القوانين:

- $CR = \frac{1}{CT}$

  CR ↓ clock rate (Hz)

  CT ↓ clock time (sec)

  cycle/sec = Hz

  MHz = 10^6 Hz

  GHz = 10^9 Hz

- Ex = CPI * IC * CT

  = CPU clock cycle * CT

  Ex: Excution time — زمن التنفيذ داخل الـ CPU

  IC: Instruction count — عدد الانستركشن inst/prog

  CPI: clock per instruction — clock cycle/ins حتى انفذ الانستركشن بحتاج

  CT: clock time — sec/cycle

- $Performance = \frac{1}{Ex\ time}$

- $Speed\ up = \frac{Ex\ time\ (\text{بطيء})}{Ex\ time\ (\text{سريع})} = \frac{Performance\ (\text{سريع})}{Performance\ (\text{بطيء})}$

- CPU clock cycle = CPI * IC

  number of clock cycle for program = CPI * IC

- $F(i) = \frac{C(i)}{IC} = \frac{\text{عدد الانستركشن للنوع الواحد}}{\text{عدد الانستركشن الكلي}}$

  ↓ Fraction

- Fraction of total Ex. time

$$\sum_{i=1}^{n} \frac{CPI_{(i)} * F(i)}{CPI}$$

- $CPI = \sum_{1}^{n} \frac{C(i) * CPI(i)}{IC} = \sum_{1}^{n} F(i) * CPI(i)$

$$= \frac{\text{CPU Ex cycle}}{IC}$$

- $MiPS = \frac{CR}{CPI * 10^6}$

Ex 1: Execution time on machine A = 1 secound
" " " " B = 10 "
Find Speed-up ??

Sol:

$$\text{speed up} = \frac{Ex.B}{Ex.A} = \frac{10}{1} = 10$$

A is 10 time Faster than B.

Ex 2: Execution time on machine A = 10 sec and CR = 4 GHz
" " " " B = 6 sec and CR = ??
the number of clock cycle for B = 1.2 * number of clock cycle for A, Find CR for B.

Sol:

Ex. time A = number of clock cycle for A * CT

$10 = \text{number of } \text{"} \text{"} \text{"} \text{"} * \frac{1}{4*10^9}$

- number of clock cycle for A = $4*10^{10}$

- " " " " " B = $1.2 * 4*10^{10}$
  $= 4.8*10^{10}$

- Ex time B = number of clock cycle for B * CT

$6 = 4.8*10^{10} * \frac{1}{CR}$

CR = 8 GHz

Ex 3:

instruction	$F_{(i)}$	$CPI_{(i)}$	$CPI_{(i)} * F_{(i)}$
Alu	50%	1	0.5
load	20%	5	1
store	10%	3	0.3
Branch	20%	2	0.4

Find:

1- CPI.

$$CPI = \sum_{i=1}^{4} CPI_{(i)} * F_{(i)}$$
$$= 0.3 + 0.4 + 1 + 0.5$$
$$= 2.2$$

2- Fraction of total Ex time.

$$\sum_{i=1}^{i=4} \frac{CPI_{(i)} * F_{(i)}}{CPI} = \frac{0.5}{2.2} + \frac{1}{2.2} + \frac{0.3}{2.2} + \frac{0.4}{2.2}$$
$$= 1$$

Ex 4: Consider of the following : → CR = 100 MHz

instruction class	$CPI_{(i)}$
A	1
B	2
C	3

Find : 1- CPU Ex time for each compiler
2- MIPS rate
3- Which is faster.

We have two compiler produce the following excecuted instruction count:

code form	A	B	C	
compiler 1	5	1	1	IC millions ---- 7
compiler 2	10	1	1	---- 12

① $CPU_1 = CPI * IC * CT$

$$CPI = \frac{5*1 + 1*2 + 1*3}{7} = 1.43$$

$$CPU_1 = 1.43 * 7 * 10^6 * \frac{1}{100 * 10^6}$$

$$= 0,1$$

$CPU_2 = CPI * IC * CT$

$$CPI = \frac{(10*1) + (1*2) + (1*3)}{12} = 1.25$$

$$CPU_2 = 1.25 * 12 * 10^6 * \frac{1}{100 * 10^6} = 0,15$$

② $$\text{Mips rate}_① = \frac{CR}{CPI * 10^6} = \frac{100 * 10^6}{1.43 * 10^6} = 70 \text{ mips}$$

$$\text{Mips rate}_② = \frac{CR}{CPI * 10^6} = \frac{100 * 10^6}{1.25 * 10^6} = 80 \text{ mips}$$

③ $$\text{speed up} = \frac{Ex \text{ بطيء}}{Ex \text{ سريع}} = \frac{0,15}{0,1} = 1,5$$

1 is faster than 2.

* نموذج امتحان (Mid)

Q1 : write Instruction Addressing modes. "
ص 32"

Q2: write Steps of Procedure.
ص 33

Q3: مثل ص 21 "فرع 7 "

Q4: write this code in Mips instruction :-
A[i] = B[i] + A [i-1]
لمئة عنصر " ص 28 "

Q5: سؤال ص 41
مع اختلاف في الـ instruction في هذا الامتحان
بس على نفس المبدأ

Q6: * سؤال الـ (floating point) ← ص 48