المحاضرة 1 عملي معالجات صغرية – ميكاترونيكس

هذه المحاضرة مخصصة لشرح المفاهيم الأساسية التالية:

- أنظمة العد (الثنائي، العشري، الست عشري).
- تمثيل التعليمات والبيانات في الحاسوب.
- تعليمات MIPS الأساسية.
- مفهوم الطفحان (Overflow).

.....

أنظمة العد:

أنظمة العد المستخدمة في الحاسوب هي:

- النظام العشري (Decimal): يستخدم الأرقام من 0 إلى 9، ويعتبر نظام العد الطبيعي للبشر.
- النظام الثنائي (Binary): يستخدم رمزين فقط: 0 و 1، وهو النظام الذي يفهمه الحاسوب داخلياً.
- النظام الست عشري (Hexadecimal): يستخدم الرموز التالية:

  0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F،

  حيث تمثّل الحروف القيم من 10 إلى 15 على التوالي.

كل نظام يُعرف بـ الأساس (Base):

- الأساس للنظام العشري = 10.
- الأساس للنظام الثنائي = 2.
- الأساس للنظام الست عشري = 16.

التحويل بين أنظمة العد:

1. التحويل من العشري إلى الثنائي:

مثال: لتحويل العدد $(250)_{10}$ إلى ثنائي:

250 ÷ 2 = 125 والباقي 0

125 ÷ 2 = 62 والباقي 1

62 ÷ 2 = 31 والباقي 0

31 ÷ 2 = 15 والباقي 1

15 ÷ 2 = 7 والباقي 1

7 ÷ 2 = 3 والباقي 1

3 ÷ 2 = 1 والباقي 1

1 ÷ 2 = 0 والباقي 1

نقرأ الباقي من الأسفل إلى الأعلى:

$(11111010)_2 \equiv (250)_{10}$

250	2	0
125	2	1
62	2	0
31	2	1
15	2	1
7	2	1
3	2	1
1	2	1
0	2	0

2. التحويل من العشري إلى الست عشري:

يتم التحويل باستخدام الطريقة نفسها ولكن بالتقسيم على العدد 16:

250 ÷ 16 = 15 والباقي 10

العدد 15 يُكتب F، والعدد 10 يُكتب A.

لذلك: $(FA)_{16} \equiv (250)_{10}$

كما يمكن الحل بتحويل العدد إلى ثنائي ثم منه إلى ست عشري:

250	16	10
15	16	15
0		

$(250)_{10} \equiv (1111\ 1010)_2$

$(15\ \ 10)_{16} \equiv (F\ A)_{16}$

3. التحويل من الست عشري إلى الثنائي:

يتم بتحويل كل رقم إلى أربع خانات ثنائية:

$(DE5)_{16} = (1101\ 1110\ 0101)_2$

حيث: D = 1101 E = 1110 0101 = 5

4. التحويل من الثنائي إلى العشري:

نرقم الخانات من اليمين لليسار، من الصفر فصاعداً، ويكون رقم الخانة هو الأس الذي سيرفع له الرقم 2، ثم نجمع كل الارقام التي يقابلها الرقم 1، مثال:

$= {}_2(11111010)$

$= 2^0\times0 + 2^1\times1 + 2^2\times0 + 2^3\times1 + 2^4\times1 + 2^5\times1 + 2^6\times1 + 2^7\times1$

$= 0 + 2 + 0 + 8 + 16 + 32 + 64 + 128 = 250$

إذاً: ${}_{10}(250) \equiv {}_2(11111010)$

5. التحويل من الست عشري إلى العشري:

$(FA)_{16} = (15\times16^1) + (10\times16^0) = 240 + 10 = 250$

أمثلة إضافية:

- $(2A)_{16} = (101010)_2 = {}_{10}(42)$
- $(12C)_{16} = (100101100)_2 = {}_{10}(300)$
- $(6F)_{16} = (1101111)_2 = {}_{10}(111)$
- ${}_2(10001) = {}_{16}(11) = {}_{10}(17)$

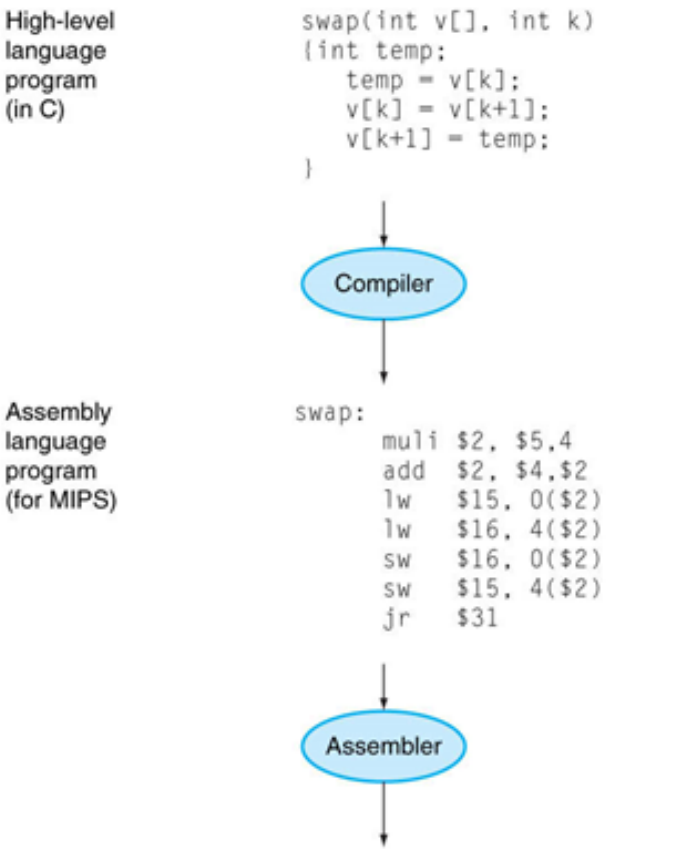
ملاحظة: يمكن استخدام الآلة الحاسبة أو الجداول المخصصة لتسريع هذه التحويلات، ولكن من المهم فهم الطريقة اليدوية لفهم كيفية تمثيل البيانات داخل الحاسوب.

البنية الأساسية لمعالج MIPS:

التعرف على بعض التعليمات الأساسية بلغة الأسميلي:

للمعالجات أنواع عديدة وأنماط مختلفة، ستعامل بدراستنا في بنية الحاسب والمعالجات على فهم المعالجات MIPS وهي المعالجات التي تقوم بتنفيذ ملايين التعليمات في الثانية الواحدة.

نقصد بالتعليمات الأسطر التي تمثل البرنامج المكتوب بلغة معينة وغالباً ما تتم كتابة البرنامج بلغة عالية المستوى مثلاً C++ أو JAVAوكون هذه اللغة لا يفهمها المعالج (فالمعالج يتعامل فقط مع الأصفار والواحدات) لذا فإنه يجب علينا تحويل هذه اللغة. يتم التحويل على مرحلتين باستخدام المترجم Compiler يتم تحويل البرنامج المكتوب بلغة عالية المستوى إلى لغة الأسميلي (وهي اللغة التي ستتعلمها) ومن ثم نستخدم المجمع Assembler لتحويل لغة الأسميلي إلى أصفار وواحدات يتعامل معها المعالج والشكل التالي يوضح ما تم ذكره.



تحتوي البنية على 32 مسجلاً (Registers)، كل منها بعرض 32 بت، موضحة في الجدول التالي:

Register	Alias	Usage	Register	Alias	Usage
$0	$zero	constant 0	$16	$s0	saved temporary
$1	$at	used by assembler	$17	$s1	saved temporary
$2	$v0	function result	$18	$s2	saved temporary
$3	$v1	function result	$19	$s3	saved temporary
$4	$a0	argument 1	$20	$s4	saved temporary
$5	$a1	argument 2	$21	$s5	saved temporary
$6	$a2	argument 3	$22	$s6	saved temporary
$7	$a3	argument 4	$23	$s7	saved temporary
$8	$t0	unsaved temporary	$24	$t8	unsaved temporary
$9	$t1	unsaved temporary	$25	$t9	unsaved temporary
$10	$t2	unsaved temporary	$26	$k0	reserved for OS kernel
$11	$t3	unsaved temporary	$27	$k1	reserved for OS kernel
$12	$t4	unsaved temporary	$28	$gp	pointer to global data
$13	$t5	unsaved temporary	$29	$sp	stack pointer
$14	$t6	unsaved temporary	$30	$fp	frame pointer
$15	$t7	unsaved temporary	$31	$ra	return address

أهم المسجلات في MIPS:

- $zero أو $0: مسجل خاص دائماً قيمته 0، ولا يمكن تعديله.
- $t0 – $t9: مسجلات مؤقتة (Temporary registers)، تُستخدم لتخزين القيم المؤقتة.
- $s0 – $s7: مسجلات مخصصة للدوال (Saved registers)، يُفترض أن تبقى قيمها بعد الانتهاء من الدالة.
- $a0 – $a3: مسجلات تُستخدم لتمرير المعاملات (Arguments) إلى الدوال.

ملاحظة: عند كتابة التعليمات بلغة التجميع، يجب الحفاظ على استخدام السجلات حسب وظيفتها لضمان توافق البرنامج مع الدوال الأخرى.

فالتعليمة تقوم باستخدام القيم الموجودة لدى المسجلات، لذا علينا معرفة التعليمات التي يمكن استخدامها أيضاً في لغة الأسمبلي.

أنواع التعليمات الأساسية في MIPS:

Category	Instruction	Example	Meaning	Comments
Arithmetic	add	add $s1,$s2,$s3	$s1 = $s2 + $s3	Three register operands
	subtract	sub $s1,$s2,$s3	$s1 = $s2 – $s3	Three register operands
	add immediate	addi $s1,$s2,20	$s1 = $s2 + 20	Used to add constants
Data transfer	load word	lw $s1,20($s2)	$s1 = Memory[$s2 + 20]	Word from memory to register
	store word	sw $s1,20($s2)	Memory[$s2 + 20] = $s1	Word from register to memory
	load half	lh $s1,20($s2)	$s1 = Memory[$s2 + 20]	Halfword memory to register
	load half unsigned	lhu $s1,20($s2)	$s1 = Memory[$s2 + 20]	Halfword memory to register
	store half	sh $s1,20($s2)	Memory[$s2 + 20] = $s1	Halfword register to memory
	load byte	lb $s1,20($s2)	$s1 = Memory[$s2 + 20]	Byte from memory to register
	load byte unsigned	lbu $s1,20($s2)	$s1 = Memory[$s2 + 20]	Byte from memory to register
	store byte	sb $s1,20($s2)	Memory[$s2 + 20] = $s1	Byte from register to memory
	load linked word	ll $s1,20($s2)	$s1 = Memory[$s2 + 20]	Load word as 1st half of atomic swap
	store condition. word	sc $s1,20($s2)	Memory[$s2+20]=$s1;$s1=0 or 1	Store word as 2nd half of atomic swap
	load upper immed.	lui $s1,20	$s1 = 20 * 2^{16}	Loads constant in upper 16 bits
Logical	and	and $s1,$s2,$s3	$s1 = $s2 & $s3	Three reg. operands; bit-by-bit AND
	or	or $s1,$s2,$s3	$s1 = $s2 | $s3	Three reg. operands; bit-by-bit OR
	nor	nor $s1,$s2,$s3	$s1 = ~ ($s2 | $s3)	Three reg. operands; bit-by-bit NOR
	and immediate	andi $s1,$s2,20	$s1 = $s2 & 20	Bit-by-bit AND reg with constant
	or immediate	ori $s1,$s2,20	$s1 = $s2 | 20	Bit-by-bit OR reg with constant
	shift left logical	sll $s1,$s2,10	$s1 = $s2 << 10	Shift left by constant
	shift right logical	srl $s1,$s2,10	$s1 = $s2 >> 10	Shift right by constant
Conditional branch	branch on equal	beq $s1,$s2,25	if ($s1 == $s2) go to PC + 4 + 100	Equal test; PC-relative branch
	branch on not equal	bne $s1,$s2,25	if ($s1!= $s2) go to PC + 4 + 100	Not equal test; PC-relative
	set on less than	slt $s1,$s2,$s3	if ($s2 < $s3) $s1 = 1; else $s1 = 0	Compare less than; for beq, bne
	set on less than unsigned	sltu $s1,$s2,$s3	if ($s2 < $s3) $s1 = 1; else $s1 = 0	Compare less than unsigned
	set less than immediate	slti $s1,$s2,20	if ($s2 < 20) $s1 = 1; else $s1 = 0	Compare less than constant
	set less than immediate unsigned	sltiu $s1,$s2,20	if ($s2 < 20) $s1 = 1; else $s1 = 0	Compare less than constant unsigned
Unconditional jump	jump	j 2500	go to 10000	Jump to target address
	jump register	jr $ra	go to $ra	For switch, procedure return
	jump and link	jal 2500	$ra = PC + 4; go to 10000	For procedure call

1. التعليمات الحسابية (Arithmetic): مثل الجمع والطرح والجمع مع عدد ثابت.
2. التعليمات المنطقية (Logical): عمليات and, or, nor إضافة للإزاحة المنطقية.
3. تعليمات الحمل والتخزين (Load/Store).
4. تعليمات القفز المشروط وغير المشروط.

كيف يمكن استخدام التعليمات؟؟

كما ذكرنا سابقاً فالتعليمة تقوم بعملية معينة واحدة وهذه العملية يتم تطبيقها على المسجلات أو القيم الثابتة أو مواقع الذاكرة.

يجب أن تعلم أنه لا يمكن وضع قيمة مؤلفة من أكثر من 32 bit في أي مسجل من المسجلات السابقة. أي أن حجم أي مسجل ضمن معالجات MIPS هو 32bit.

1. لجمع قيمة مسجلين مثلاً يمكن استخدام تعليمية الجمع الحسابي وفق الآتي: add $s1, $s2, $s3;

   وهي تعني جمع قيمتي المسجلين $s2, $s3 ووضع الناتج في المسجل $s1 أي أن $s1=$s2+$s3

2. عملية الطرح تتم بشكل مماثل كما يلي: sub $s1, $s2, $s3;

   وهي تعني طرح قيمتي المسجلين $s2, $s3 ووضع الناتج في المسجل $s1 أي أن $s1=$s2−$s3

3. أما لجمع قيمة معينة ثابتة لمسجل ما فتستخدم addi كما يلي: addi $s2, $s1, 22

   وهي تكافئ العملية الحسابية $s2=$s1+22

مثال:

إذا أردنا القيام بالعملية الرياضية التالية:

$s5=($s2+$s4)−($s1+7)

يتم ذلك من خلال التعليمات التالية:

```
add $t0, $s2, $s4;
addi $t1, $s1, 7;
sub $s5, $t0, $t1;
```

- لاحظ كيف تم استخدام المسجلات المؤقتة $t0, $t1 لحفظ النتائج المؤقتة
- لاحظ كيف تم إجراء العمليات وفقاً لأولوية الأقواس
- لاحظ استخدام التعليمات المناسبة عند الحاجة إليها: add, addi, sub
- لاحظ أننا وضعنا الناتج النهائي في مسجل خاص بحفظ القيم من نوع $s وليس موقت (من نوع $t).

الطفحان Overflow:

الطفحان (Overflow) لا يحدث عند:

- جمع عددين بإشارتين مختلفتين.
- طرح عددين من نفس الإشارة.

حالات حدوث الطفحان في الجمع:

- جمع عددين موجبين والحصول على عدد سالب.
- جمع عددين سالبين والحصول على عدد موجب.

حالات حدوث طفحان في الطرح:

- طرح عدد سالب من عدد موجب والحصول على عدد سالب.
- طرح عدد موجب من عدد سالب والحصول على عدد موجب.

ملاحظات هامة:

- العدد السالب في النظام الثنائي تكون قيمة البت الأكثر أهمية (أول بت من جهة اليسار) قيمته 1، أما العدد الموجب فتكون قيمة البت الأكثر أهمية مساوية 0.
- لاحظ الأمثلة التالية ولاحظ حالات حدوث طفحان في بعضها وفقاً للحالات المذكورة سابقاً.

-39 + 92 = 53:

```
  1   1 1
  1 1 0 1 1 0 0 1
+ 0 1 0 1 1 1 0 0
  0 0 1 1 0 1 0 1
```

Carryout without overflow. Sum is correct.

-19 + -7 = -26:

```
  1 1 1 1     1
  1 1 1 0 1 1 0 1
+ 1 1 1 1 1 0 0 1
  1 1 1 0 0 1 1 0
```

Carryout without overflow. Sum is correct.

44 + 45 = 89:

```
      1   1 1
  0 0 1 0 1 1 0 0
+ 0 0 1 0 1 1 0 1
  0 1 0 1 1 0 0 1
```

No overflow nor carryout.

104 + 45 = 149:

```
    1 1   1
  0 1 1 0 1 0 0 0
+ 0 0 1 0 1 1 0 1
  1 0 0 1 0 1 0 1
```

Overflow, no carryout. Sum is not correct.

-75 + 59 = -16:

```
      1 1 1 1 1 1
  1 0 1 1 0 1 0 1
+ 0 0 1 1 1 0 1 1
  1 1 1 1 0 0 0 0
```

No overflow nor carryout.

-103 + -69 = -172:

```
  1   1 1 1   1 1
  1 0 0 1 1 0 0 1
+ 1 0 1 1 1 0 1 1
  0 1 0 1 0 1 0 0
```

Overflow, with incidental carryout. Sum is not correct.

10 + -3 = 7:

```
  1 1 1 1
  0 0 0 0 1 0 1 0
+ 1 1 1 1 1 1 0 1
  0 0 0 0 0 1 1 1
```

Carryout without overflow. Sum is correct.

127 + 1 = 128:

```
    1 1 1 1 1 1 1
  0 1 1 1 1 1 1 1
+ 0 0 0 0 0 0 0 1
  1 0 0 0 0 0 0 0
```

Overflow, no carryout. Sum is not correct.

-1 + 1 = 0:

```
  1 1 1 1 1 1 1 1
  1 1 1 1 1 1 1 1
+ 0 0 0 0 0 0 0 1
  0 0 0 0 0 0 0 0
```

Carryout without overflow. Sum is correct.