

## T2: Instruccions i Tipus de Dada

ISA: Especificació que descriu els aspectes del processador visible al programador.  
Instruccions, registres, model de memòria, ...

Mateix ISA pot ser implementat diferent. Exemples: MIPS, x86, RISC-V, ...

ABI: Especificació que descriu Interfície de Baix Nivell entre mòduls del programari.  
Com es criden les funcions, com es retornen funcions, ...

RISC: Instruccions mides fixes, pocs modes d'adreçament, Accés memòria  $\hookrightarrow$  Load / Store.

Poques instruccions: ARM, MIPS, ... # Reduced Instruction Set Computer

1 Kb =  $2^{10}$  b    1 Mb =  $2^{20}$  b    1 Gb =  $2^{30}$  b

### La memòria

Memòria  $\equiv$  1B |  $2^{32}$  adreces    0x00000000 | 1Byte  
0xFFFFFFF | 1Byte

Per seguir aquest.

Little-Endian: Primer el de menys pes. 0x76543210  $\rightarrow$

0x10  
0x32  
0x54  
0x76

Big-Endian: Primer el de més pes. 0x76543210  $\rightarrow$

0x76  
0x54  
0x32  
0x10

### Variables

Obligatori inicialitzar les variables.

```
int g1;
void main() {
    int l1;
    l1 = g1;
}
```

```
.data
g1: .word 0 # Variable global 'g1'
.text
.globl main
main:
    la $t0, g1 # Guardar la direcció de g1
    lw $t1, 0($t0) # l1 a $t1 (Contingut)
    # MIPS
```

char [byte] 1 Byte    int [word] 4 Bytes  
short [half] 2 Bytes    long long [dword] 8 Bytes

```
char c = 0x11;
short s = 0x2211;
int i = 0x44332211;
unsigned int ui;
long long ll = 0x8877665544332211;
```

L'adreça de la variable ha de ser múltiple de la grandària de la variable.  
# half (2B) múltiple de 2. | word (4B) múltiple de 4. | dword (8B) múltiple de 8.

"space n" Reserva 'n' Bytes a 0.

Si vull un vector de "shorts" fiquen "space n \* 2" b's. És de int "space n \* 4".

Recorda que és 2 Bytes

Recorda que és 4B d'word

"align m" Ubica propers dades a @múltiple de  $2^m$  ( $\log_2(m)$ )

```
char c;
int v[100];
```

```
c: .byte 0
    .align 2
v: .space 400
```

MIPS

Recorda que  
 $\log_2(4) = 2$   
 i 4 pg. int = 4B

## Operands

5 modes adreçament: - Register - Memory - Relative PC  
 - Immediate - Pseudoregister

"addu \$t1, \$t2, \$t3" # \$t1 = \$t2 + \$t3

Fa Sign Ext (imm 16)

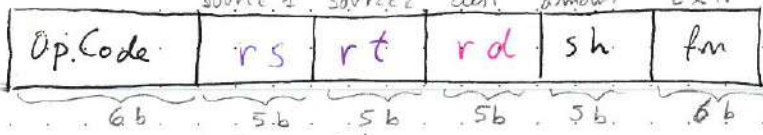
"addui \$t1, \$t2, 10" # \$t1 = \$t2 + 10

Immediat de 16B en Ca 2

"subu \$t1, \$t2, \$t3" # \$t1 = \$t2 - \$t3

# sh per desplaçament

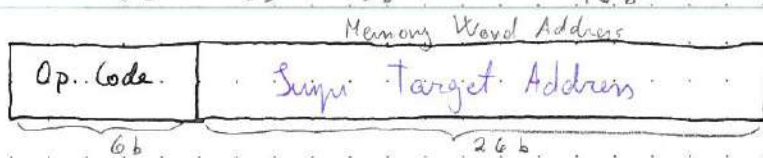
Està relacionat amb l'overflow



"op \$rd, \$rs, \$rt"



"op \$rt, \$rs"



"op

Fa Sign Ext  
 si és 16b.

```
li $t1, 0x44332211 # lui $at, 0x4433
# ori $t1, $at, 0x2211
```

MIPS

```
li $t1, 0x2211
# addiu $t1, $zero, 0x2211
```

MIPS

"li" és una macro que permet guardar operand de 32b.

Còpia valors entre registre

```
addiu $t1, $t2, 0 # $t1 = $t2
```

MIPS

```
addu $t1, $zero, $t2 # $t1 = $t2
```

MIPS

```
move $t1, $t2 # E.g. addu...
```

MIPS

MARS treballa amb variables de 32 llavors s'figen "0xFFFF"

el MARS llegeix 0x0000 FFFF

Guardar adreça en variable

```
"la rdest, address" # lui $at, hi(address)
# ori rdest, lo(address)
```

```
.text
a: .word 0
    .text
    .globl main
main:
    la $t0, a # $t0 = &a = 0x10010000
```

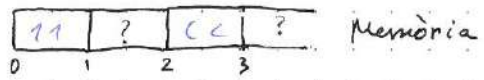
MIPS



```

lb $t1, 0($t2) # 0x00000011
lb $t1, 2($t2) # 0xFFFFFCC
lbu $t1, 0($t2) # 0x00000011
lbu $t1, 2($t2) # 0x000000CC

```



No exten signe

MIPS

## Repàs de Ca2

Representació d'enters en excés  $2^{n-1} - 1$ :

- Funció d'interpretació  $x = x_u - (2^{n-1} - 1)$
- Funció de representació  $x_u = x + (2^{n-1} - 1)$

Rang  $x \in [-(2^{n-1} - 1), 2^{n-1} - 1]$

x	$x_u$	x
000	0	-3
001	1	-2
010	2	-1
011	3	0
100	4	1
101	5	2
110	6	3
111	7	4

## Representació Caràcters

```

char c1 = 'R'
c1: char 'R'

```

## Vectors

Per accedir a la i-èsima posició del vector.

Adreça  $\&vec[0]$  i  $i$   $\cdot$  Tamany

$$\&vec[0] + i * T$$

```

lb $t0, vec2 # $t0 = &vec2[0]
lb $t0, vec2 + 3*4 # $t0 = &vec2[3]

```

```

lb $t0, vec2 # $t0 = &vec2[0]
lb $t1, i
lw $t2, 0($t1) # i
sll $t2, $t2, 2 # i * 4
addu $t3, $t2, $t0 # $t3 = &vec2[i]

```

Això només és l'adreça, després falta comptar

```

char nom[4] = {'P', 'a', 'u', '\0'};
char nom = "Pau";

```

- ascii NO reserve centinella
- ascii z SI reserve centinella

## Punter

Variable que conté una adreça de memòria. SEMPRE és word p4 memòria 32b.

# Si el punter és global  $\rightarrow$  Havia d'haver 2 loads.

# Si el punter és local  $\rightarrow$  Havia d'haver 1 load.

⚠️ Quan operem amb punter, hem de multiplicar  $\cdot$  tamany.

Si és char  $*p1 \Rightarrow p1 = p1 + 1$   
 Si és int  $*p2 \Rightarrow p2 = p2 + 4$   
 4B  
 TC-2-T-2



2.1. Tradueix C  $\rightarrow$  MIPS.  $f = \$t0, g = \$t1, h = \$t2, i = \$t3, j = \$t4$

a)  $f = g + h + i + j$

addu  $\$t0, \$t1, \$t2$

addu  $\$t3, \$t3, \$t4$

addu  $\$t0, \$t0, \$t3$

b)  $g = f + (h + 5) - i$

addui  $\$t2, \$t2, 5$

addu  $\$t0, \$t0, \$t2$

subu  $\$t1, \$t0, \$t3$

2.2. Tradueix de MIPS  $\rightarrow$  C.

a)  $f = g + h$

$g = i + j$

$g = g + g$

$f = f + g$

$\Rightarrow f = (g + h) + 2(i + j)$

b)  $g = g + h$

$i = i + j$

$f = g + i$

$f = f + 2$

$f = f + f$

$\Rightarrow f = 2(2 + (g + h) + (i + j))$

2.3. Indica valor en Hexa dels valors guardats en  $0x1001000C$  i  $0x1001001C$

data  $0x10010000$   
 byte 1, 2, 3, 4  
 word -1, 1, -2, 2, -3, 3  
 word  $0x12345678$

$0x1001000C = 0xFFFFE \# 0xFF$

$0x1001001C = 0x5678 \# 0x78$   
 7. memòria GB. Little Endian

~~ET No existeix pg. la memòria memòria guarda~~

Aclaració: Podem dir que  $[0x1001000C] = 0xFFFFFE$  per simplificar.

En realitat tenim que  $[0x1001000C] = 0xFF$

Dir "0xFFFFE" si que està  $[0x1001000D] = 0xFF$

mentre pg. no diu  $[0x1001000E] = 0xFF$

res bé.  $[0x1001000F] = 0xFF$

2.4. Det. quin reg. o adreça es modifica i el contingut.

lw  $\$t1, 0(\$t0) \quad \$t1 \leftarrow 0x0123456789ABCDEF$

lw  $\$t3, 0(\$t2) \quad \$t3 \leftarrow 0x0000.ECFD.0E0F.0000.0000$

sw  $\$t3, -8(\$t2) \quad A \leftarrow 0x0000.ECFD.0E0F.0000.0000$

lw  $\$t4, 4(\$t0) \quad \$t4 \leftarrow B$

sw  $\$t1, -4(\$t2) \quad B \leftarrow 0x0123456789ABCDEF$

sw  $\$t4, 0(\$t2) \quad C \leftarrow B$



2.26.

```

int *m1, *m2;
main () {
    int *r1, *r2;
    {
        a) r1 = r2;
        move $t1, $t2
        b) *r1 = *r2;
        lw $t3, 0($t2)
        sw $t3, 0($t1)
    }
}

```

```

c) m1 = m2;
la $t0, m2
lw $t0, 0($t0)
la $t3, m1
sw $t0, 0($t3).

```

d) \*m1 = \*m2;

```

la $t0, m2    # &m2
lw $t0, 0($t0) # m2
lw $t0, 0($t0) # *m2
la $t1, m1    # &m1
lw $t4, 0($t1) # m1
sw $t0, 0($t4) # *m1 = *m2

```

2.30.

```

char a;
int b;
long long int c;
main () {
    char *p; # $t0
    int *q; # $t1
    long long int *h; # $t2
}

```

a) q = q + 1;

```

addiu $t2, $t2, 4 # Donat que és int necessito 1 * 4 → 4 Bytes

```

b) a = \*p;

```

lb $t3, $t0 p q char *p;
b $t4, a    # char és byte

```

s(b) \$t3, 0(\$t4)

d) b = \*(q + b)

la \$t3, b # &b

lw \$t4, 0(\$t3) # b

sll \$t5, \$t4, 2 # b \* 4

addu \$t6, \$t1, \$t5 # q + b

lw \$t7, 0(\$t6) # \*(q + b)

sw \$t7, 0(\$t3) # b = \*q + b

c) h = &c;

la \$t0, c # p q h és local i està a \$t2

e) \*h = \*(h + b)

la \$t3, b # &b

lw \$t4, 0(\$t3) # b

sll \$t5, \$t4, 3 # b \* 8 ## log<sub>2</sub>(8) = 3

addu \$t6, \$t2, \$t5 # h + b

```

lw $t7, 0($t6) } *h = l0 (* (h + b))
sw $t7, 0($t6) }

```

```

lw $t7, 4($t6) } *h = lw (* (h + b))
sw $t7, 4($t6) }

```

# p q long long són 8B; lw només trauu a mb 4

2.27. Tradueix a 4 línies de C el següent apartat. "int \*pdata;".

a) la \$t0, pdata # & pdata

lw \$t0, 0(\$t0) # pdata

lw \$t1, 0(\$t0) # \*pdata

addiu \$t1, \$t1, 4 # \*pdata + 4

sw \$t1, 0(\$t0)

\*pdata = \*pdata + 4;

b) la \$t0, pdata # & pdata

lw \$t1, 0(\$t0) # pdata

addiu \$t1, \$t1, 4 # pdata + 4

sw \$t1, 0(\$t0)

[pdata = pdata + 1;]

$$Q = P + N \cdot T$$

Recordo que \*pdata és punter i li he autmetica.

c) la \$t0, pdata

lw \$t0, 0(\$t0)

lw \$t1, 0(\$t0)

addiu \$t0, \$t0, 4

sw \$t1, 0(\$t0)

[(pdata + 1) = \*pdata;]

2.22. "char A = 'c'; int B = -1;"  
0x10010000 0x10010004

a) Tradueix a MIPS.

la \$t0, A

li \$t0, 'c'

la \$t1, B

li \$t1, -1

b) Valor de \$t0 post. exec.

text

la \$t0, A

la \$t1, B

lb \$t0, 0(\$t0)

lw \$t1, 0(\$t1)

addu \$t0, \$t0, \$t1

0x00000042

2.32. Tradueix.

char index[100];

short meitat[100];

int val[100], vec[100];

main() {

char c;

int i, j;

{

c) c = index[i + val[j]];

la \$t0, val # & val[0]

slr \$t2, \$t2, 2

addu \$t0, \$t0, \$t2 # & val[j]

lw \$t3, 0(\$t0) # val[j]

addu \$t1, \$t1, \$t3 # i + val[j] # No fe fallte mult.

la \$t4, index # & index[0]

addu \$t4, \$t4, \$t4

lb \$t4, 0(\$t4) # index[i + val[j]]

move \$t0, \$t4

2.31. "int data; int \*pdata;"

a) ~~pdata~~ = data;

l<sub>w</sub> \$t0, pdata

l<sub>w</sub> \$t1, data

sw \$t1, 0(\$t0)

b) \*pdata = \*pdata + 4;

l<sub>w</sub> \$t0, pdata

l<sub>w</sub> \$t0, 0(\$t0)

l<sub>w</sub> \$t1, 0(\$t0) # \*pdata

addiu \$t2, \$t1, 4

sw \$t2, 0(\$t0)

c) pdata = pdata + 1;

l<sub>w</sub> \$t0, pdata # \$pdata

l<sub>w</sub> \$t1, 0(\$t0) # \*pdata

addiu \$t1, \$t1, 4 # \*pdata + 1

sw \$t1, 0(\$t0)

d) data = data - 1;

l<sub>w</sub> \$t0, data

l<sub>w</sub> \$t0, 0(\$t0)

subiu \$t1, \$t0, 1

l<sub>w</sub> \$t0, data

sw \$t1, 0(\$t0)