T2: Instruccions i Tipus de Dades

ISA: Especificació que descriu els aspectes del processador visibles 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 prog.
Com es criden les funcions, com es retornen funcions, ...

RISC: Instruccions mida fixa, pocs modes adreçament, Accés mem ↳ Load / Store
Poques instruccions. ARM, MIPS, ... # Reduced Instruction Set Computer

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

La memòria

Memòria ≡ 1B | 2^{32} addreces 0x00000000 [1Byte] ⋮ 0xFFFFFFFF [1Byte]

Fer servir aquest.

▽ **Little-Endian:** Primer el de menys pes. 0x76543210 → [0x10 | 0x32 | 0x54 | 0x76]

Big-Endian: Primer el de més pes. 0x76543210 → [0x76 | 0x54 | 0x32 | 0x10]

Variables

▽ Obligatori inicialitzar les variables.

```
int g1;
Void main() {
    int l1;
    l1 = g1;
}
                C
```

```
    .data
g1:   .Word 0    # Variable global 'g1'
    .text
    .globl main
main:      addr
    la $t0, g1       # Guardar la direcció de g1
    lw $t1, 0($t0)  # l1 a $t1 (Contingut)
     ↑ Word
                                   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 l = 0x8877665544332211
```

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

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

Si vull un vector de "shorts" fiquem '.space n*2' o si és de int '.space n*4'

Recorda que és 2Byte

Recorda que és 4B el .word = int

"**.align n**" Ubica propera dada a @ múltiple de 2^n. ($\log_2(n)$)

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

```
c:  .byte 0
    .align (2)
v:  .space 400
                MIPS
```

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

Operands

5 modes adreçament: - Registre - Memòria - Relatiu PC - Immediat - Pseudodirecte

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

"addui $t1, $t2, 10" # $t1 = $t2 + $\boxed{10}$ ▽ Fa SignExt(imm 16) ▽ Immediat de 16B en Ca 2.

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

▽ "*u" Està relacionat amb l'overflow

#Sh per desplaçament

	Source 1	Source 2	dest	Shift amount	Op. code Ext.
Op.Code	rs	rt	rd	sh	fn
6 b	5 b	5 b	5b	5 b	6 b

"op $rd, $rs, $rt"

	source	dest	
Op. Code	rs	rt	Immediat / Offset
6 b	5b	5b	16 b

"op $rt, $rs"

	Memory Word Address
Op. Code	Sign Target Address
6 b	26 b

"op

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

Fa Sign Ext si és 16 b.

```
li $t1, (0x2211)
# addiu $t1, $zero, $0x2211
                                        MIPS
```

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

Còpia valors entre registre

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

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

```
move $t1, $t2  #Eq. addu...
                           MIPS
```

▽ ▽ MARS treballa amb variables de 32 llavors si fiques "0xFFFF" el MARS llegeix 0x0000 FFFF

Guardar adreça en variable

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

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

```
lb  $t1, 0($t2)   #0x00000011
lb  $t1,2($t2)    #0xFFFFFFCC
lbu $t1, 0($t2)   #0x00000011 }
lbu $t1,2($t2)    #0x000000CC }  No exten signe
                                   MIPS
```

11	?	CC	?
0	1	2	3

Memòria

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

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'
                C
```

```
c1: .char 'R'
                MIPS
```

Vectors

Per accedir a la i-èsima posició del vector. $\overset{\text{Adreça (vec)}}{\&v[0]} + i * \overset{\text{Tamany}}{T}$

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

```
la   $t0, vec2        # $t0=&vec2[0]
la   $t1, i
lw   $t2,0($t1)       # i
sll  $t2,$t2,2        # i*4        Això només és l'adreça, després falta ▷ contingut
addu $t3,$t2,$t0      # $t3 = &vec2[i]
                                                    MIPS
```

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

- .ascii NO reserva centinella.
- .asciiz SÍ reserva centinella.

Punter

Variable que conté una adreça de memòria. SEMPRE és .word pq sempre fa 32b

Si el punter és global → Haurà d'haver 2 loads.

Si el punter és local → Haurà d'haver 1 Load.

⚠ Quan opero amb punter, hem de multiplicar * tamany. → Si és char $\overset{1B}{*}$p1 ⇒ p1=p1+1
→ Si és $\underset{4B}{\text{int}}$ *p2 ⇒ p2=p2+4

2.1. Tradueix C → 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 → C.

a)
$$\left.\begin{array}{l} f = g + h \\ g = i + j \\ g = g + g \\ f = f + g \end{array}\right\} \Rightarrow f = (g+h) + 2(i+j)$$

b)
$$\left.\begin{array}{l} g = g + h \\ i = i + j \\ f = g + i \\ f = f + 2 \\ f = f + f \end{array}\right\} \Rightarrow f = 2(2 + (g+h) + (i+j))$$

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 = 0xFFFE #0xFE

0x1001001C = 0x5678 #0x78 ← Tq. només 4B i Little Endian

~~No entenc pq. la memòria només guarda~~

Aclaració: Podem dir que [0x1001000C] = 0x FF FF FF FE per simplificar.

En realitat tindrien que [0x1001000C] = 0xFE

Dir "0xFFFE" sí que està malament pq no diu res bé.

[0x1000000D] = 0xFF

[0x1001000E] = 0xFF

[0x100000F] = 0xFF

Recorda que és guarda Little-Endian.

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

```
lw $t1, 0($t0)    $t1 ← 0x0123456789ABCDEF
lw $t3, 0($t2)    $t3 ← 0x0000.ECFD.0E0F.0000.0000
sw $t3, -8($t2)   A ← 0x0000.ECFD.0E0F.0000.0000
lw $t4, 4($t0)    $t4 ← B
sw $t1, -4($t2)   B ← 0x0123456789ABCDEF
sw $t4, 0($t2)    C ← 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 $t1, 0($t1)      # m1
sw $t0, 0($t1)      # *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 incremento 1*(4) → 4Bytes
```

b) a = *p;

```
lb $t3, $t0 pq char *p;
la $t4, a   i char és byte
sb $t3, 0($t4)
```

c) h = &c;

```
la $t2, c # pq h és local i està en $t2
```

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
```

e) *h = *(h+b)

```
la $t3, b  #&b
lw $t4, 0($t3) # b
sll $t5, $t4, 3 # b*8  ## log2(8) = 3
addu $t6, $t2, $t5 # h+b
lw $t7, 0($t6)  } *h = lo (*(h+b))
sw $t7, 0($t2)  }
lw $t7, 4($t6)  } *h = hi (*(h+b))
sw $t7, 4($t6)  }
# Pq long long són 8B i lw només tracta amb 4
```

2.27. Tradueix a 1 línia 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)
```
$\boxed{\texttt{*pdata = *pdata + 4;}}$

b)
```
la $t0, pdata      # & pdata
lw $t1, 0($t0)     # pdata
addiu $t1, $t1, 4  # pdata + 4
sw $t1, 0($t0)
```
$\boxed{q = p + N * T}$

$\boxed{\texttt{pdata = pdata + 1;}}$

Recorda que *pdata és punter i hi ha aritmètica.

c)
```
la $t0, pdata
lw $t0, 0($t0)
lw $t1, 0($t0)
addiu $t0, $t0, 4
sw $t1, 0($t0)
```
$\boxed{\texttt{*(pdata + 1) = *pdata;}}$

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 indx[100];
short meitat[100];
int val[100], vec[100];
main () {
    char c;     $t0
    int i, j;   $t1 $t2
}
```

c) c = indx[i + val[j]];
```
la $t0, val          # & val[0]
sll $t2, $t2, 2
addu $t0, $t0, $t2   # & val[j]
lw $t3, 0($t0)       # val[j]
addu $t1, $t1, $t3   # i + val[j]  ## No fa falta mult.
la $t4, indx         # & indx[0]
addu $t4, $t4, $t1
lb $t4, 0($t4)       # indx[i + val[j]]
move $t0, $t4
```

2.31 "int data; int *pdata;"

a) pdata = &data;

```
la $t0, pdata
la $t1, data
sw $t1, 0($t0)
```

b) *pdata = *pdata + 1;

```
la $t0, pdata
lw $t0, 0($t0)
lw $t1, 0($t0) # *pdata
addiu $t2, $t1, 1
sw $t2, 0($t1)
```

c) pdata = pdata + 1;

```
la $t0, pdata #&pdata
lw $t1, 0($t0) #pdata
addiu $t1, $t1, 4 #pdata+1
sw $t1, 0($t0)
```

d) dade = dade - 1;

```
la $t0, dade
lw $t0, 0($t0)
subiu $t1, $t0, 1
la $t0, dade
sw $t1, 0($t0)
```