



Arnau_FIB
www.wuolah.com/student/Arnau_FIB

11186

RESUM EC TEMES 1-5.pdf RESUMEN EC TEMAS 1-5



1º Estructura de Computadores



Grado en Ingeniería Informática



Facultad de Informática de Barcelona (FIB)
UPC - Universidad Politécnica de Catalunya



**Encontré un perro
en tus apuntes.**
WUOLAH

TEMA 1: RENDIMENT I CONSUM

Rendiment condicionat per:

- Temps d'execució
- Productivitat (Throughput)

Temps d'execució A EC, $t_{exe} = t_{CPU}$ \rightarrow Rendiment = $\frac{1}{t_{exe}}$

Speedup (Guany de rendiment) Speedup = $\frac{R_{millorat}}{R_{original}} = \frac{T_{exe\ original}}{T_{exe\ millorat}}$

Factors que influïxen en el texe

$$t_{exe} = n_{cycles} \times t_c = \frac{n_{cycles}}{f_{rellotge}} \rightarrow \text{Promedio de Ciclos por Instrucción}$$

Reduir n cycles \rightarrow $n_{cycles} = n_{ins} \cdot CPI \rightarrow$ Reduir CPI \rightarrow Millorar microarquitectura
 Substituir instr. costoses

$$t_{exe} = n_{ins} \times CPI \times t_c$$

Reducir el t_c \rightarrow Augmentar f_{rellotge} (No sempre millora el rendiment)

Llei Amdahl

El màxim speedup que es pot aconseguir és:

$$S_t = \frac{1}{(1-P) + \frac{P}{S}} \rightarrow S_{max} = \frac{1}{1-P}$$

Potència dinàmica

consum d'energia produt durant els cicles de càrrega / descàrrega dels transistors.

$$P_{din} = \alpha \cdot C \cdot V^2 \cdot f_{rellotge}$$

α : Fracció de transistors que commuten per ordre
 C : Capacitància

Potència estàtica

$$P_{est} = I_{leak} + V$$

I_{leak} : Intensitat en circuit obert

Potència total

$$P_{tot} = P_{din} + P_{est}$$

$$E = P \cdot t$$

Ponle nombre a lo que quieras ser

Master BIM Management



60 Créditos ECTS



Jose María Girela
Bim Manager.



TEMA 2 : ENSAMBLADOR MIPS I TIPUS DE DADES BÀSIQUES

PROCESSA DOR: MIPS

- Tipus RISC
- Adreça de 32 bits
- 32 registres de propòsit general

→ Memòria:

-per	11
	22
	33
+pes	44
	00

Conveni little-endian
word 44, 33, 22, 11

VARIABLES

Declaracions

Global

Local (dins una funció)

Mida de les variables:

	bytes
.byte char	1
.half short	2
.word int	4
.dword long long	8

REGISTRES DEL MIPS

- \$zero → conté el 0
 \$at → Temporal per pseudoinstruccions
 \$v0 - v1 → Resultats de subrutines
 \$a0 - a3 → Paràmetres de subrutines
 \$t0 - t7 → temporals
 \$s0 - s7 → Segurs
 \$t8 - t9 → temporals
 \$k0 - k1 → Reservats pel S.O.
 \$gp → Global pointer
 \$sp → Stack pointer
 \$fp → Frame pointer
 \$ra → Return address

OPERANDS EN MODE MEMÒRIA

- | | |
|------------------|------------------|
| lw rd, off16(rs) | lb rd, off16(rs) |
| sw rt, off16(rs) | sb rt, off16(rs) |

- * Un cop declarades en col·loquen a la primera posició múltiple de n, on n és els bytes que ocupen.

INSTRUCCIONS BÀSIQUES

- addu rd, rs, rt
 addiu rd, rs, immediat
 subu rd, rs, rt

PSEUDOINSTRUCCIONS / MACROS

- move rd, rs
 li rd, immediat
 lui rd, immediat
 la rd, etiq
 la rd, etiq + immediat

SISTEMES DE REPRESENTACIÓ

1) Naturals

$$x_{10} = 724 \quad x_{10} = \sum_{i=0}^{n-1} x_{i_2} \cdot 2^i \rightarrow x_2 = 10\ 1101\ 0100$$

2) Enters (en Ca2)

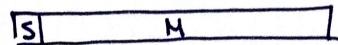
$$x_s = -x_{n-1} \cdot 2^{n-1} + \sum_{i=0}^{n-2} x_i \cdot 2^i$$

3) Enters (en Ca1)

$$x_s = x_u - x_{n-1} \cdot (2^n - 1)$$

Si és \oplus en igual,
si és \ominus en canviar
tots els bits.

4) Signe i magnitud



El bit de més pes indica el signe

5) En excess a β

$$x_s = x_u - \beta \quad \beta = 2^n - 1$$

CARÀCTERS

Taula de correspondència ASCII :

0 - 31	: control
32	: espai
48	: '0'
65	: 'A'
97	: 'a'

FORMAT D'INSTRUCCIONS

R (register) :

6	s	s	s	s	s	6
opcode	rs	rt	rd	shamt	funct	

opcode : codi d'operació

funct : extensió de opcode

shamt : shift amount,

immediat per a desplaçaments

I (immediate) :

6	opcode	rs	rt	imm16
---	--------	----	----	-------

J (jump) :

6	opcode	adress
---	--------	--------

VECTORS

- En memòria en posicions consecutives, alineats segons la seva mida en bytes
- Si es declaren posant .space n_elem * size_elem cae alinear-los prèviament

En C: short v1[5] = {1,2,3,4,5}; En Ass:

int v2[10];

int v3[3] = {1,-1,2};

.data

.half 1,2,3,4,5

.align 2

.space 40

.word 1,-1,2

@vec[i] = @vec[0] + i * size_elem

v1:

v2:

v3:



¿Puedo quedarme
este perro?

WUOLAH

STRINGS / CADENES DE CARÀCTERS

Sentinella: '\0'

char cadena [10] = {'U','n','a',' ','y','r','a','s','e','\0'}
char cadena [10] = "una frase"

En Ass.:

- . ascii "una frase"
- . byte 0
- . asciiiz "una frase"

PUNTERS

Si p és punter, alentorn p és una variable que conté una direcció de memòria, és de 32 bits i és semblant a un enter.

Un punter apunta al tipus de dada al qual ha estat declarat:

```
int k;  
char c;  
int *p;  
char *q;
```

→ CORRECTE: → INCORRECTE:

$p = \&k;$
 $q = \&c;$

$p = \&c;$
 $q = \&k;$

Iniciàlitzacio: int $p = \&k;$

Derreferència: int $l = *p;$ # l és la dada apuntada per p

Aritmètica de punters: S'ha de tenir en compte que els punters contenen adreces de memòria, ja que per sumar una posició (en un vec per ex.) caldrà sumar 1 (byte), 2 (half), 4 (word), ...

Punters vs. Vectors

És útil utilitzar punters per accedir a elements de vectors:

Si p apunta a $\text{vec}[0] \Rightarrow p + i$ apunta a $\text{vec}[i]$

TEMA 3: TRADUCCIÓ DE PROGRAMES

DESPLAZAMIENT DE BITS

(Amb registros)

- << → sll rd, rt, shamt → sllv rd, rs, rt → Mult per 2^n
 >> → srl rd, rt, shamt → srav " " " → Div per 2^n
 >> → sra rd, st, shamt → srav " " " → Div. enter 2^n

OPERACIÓNS LÒGQUES BIT A BIT

&	and	rd, rs, rt	andi	rd, rs, imm16	→ Porta AND	X ₁ , X ₀	AND	OR	XOR
	or	" "	ori	" "	→ Porta OR	0 0	0	0	0
^	xor	" "	xori	" "	→ Porta XOR	0 1	0	1	1
	nor	" "		→ negació del or		1 0	0	1	1
N	not	" "		→ negació bit a bit		1 1	1	1	0

L'and, l'or, etc serveixen per crear màscars (per ex. posar tots els bits d'un register a 0 excepte els de les pos 5,3 i 1)

OPERADORS BOOLEANOS

&& → And || → Or ! → Not

→ AVALUACIÓ "LAZY"

```
↳ if (cond1 op cond2) {
    then: ...
    else ...
}
fi
```

$\left\{ \begin{array}{l} \text{And: si cond1 cert} \rightarrow \text{salta al then} \\ \quad \text{si fals} \rightarrow \text{salta al else} \\ \text{or: si cond1 cert} \rightarrow \text{comprueba cond2} \\ \quad \text{cert} \rightarrow \text{then} \\ \quad \text{fals} \rightarrow \text{salta al else} \\ \text{else} \rightarrow \text{else} \end{array} \right.$

OPERACIÓNS DE COMPARACIÓ

slt rd, rs, rt slt u rd, rs, rt

slti rd, rs, imm16 sltiu rd, rs, rt

Exemples:

$$c = a < b \rightarrow \text{slt } c, a, b$$

$$c = a != b \rightarrow \text{subu } c, a, b$$

$\hookrightarrow (a-b) > 0$

unsigned

$\text{sltu } c, 0, c$

WUOLAH

OPERACIONS DE SALT

Salt condicional

$$PC \leftarrow PC_{up}(PC+4) + \text{SigExt}(\text{offset } 16 \cdot 4)$$

beq rs, rt, etiq # branch if equal

bne rs, rt, etiq # branch if not equal

bgt rs, rt, etiq # branch if greater than

Macros: blt, bge, ble, bgtu, bltu, bgeu, bleu

Salt incondicional

b etiq # branch to etiq

Salts absoluts

j target # jump $PC \leftarrow \text{target}$

jal target # jump and link $\begin{cases} PC \leftarrow \text{target} \\ \$ra \leftarrow PC_{up} \end{cases}$

jr rs # jump at register $PC \leftarrow rs$

jaer rs, rd # jump and link register $\begin{cases} PC \leftarrow rs \\ rd \leftarrow PC_{up} \end{cases}$

* A EC utilitzem jae i jr,
en principi jal per fer
cridar a subrutines i jr per
acabar el programa, jr \$ra
(o la subrutina corresponent)

SUBRUTINES

Paràmetres i resultats

Els paràmetres en passen als registres \$a0...\$a3
El resultat en passa al registre \$v0 (enter)
\$f0 (float)

Exemple:

```
void main() {
    int x, y, z; // $t0, $t1, $t2
    z = suma(x, y);
}

int suma(int a, int b) {
    return a+b;
}
```

```
main: move $a0, $t0
      move $a1, $t1
      jal suma
      move $t2, $v0
      jr $ra
```

suma:

```
addu $v0, $a0, $a1
```

Bloc d'activació i la pila

Algunes funcions requereixen guardar variables locals en memòria. Aquestes es guarden a la pila. El registre \$sp apunta al cim de la pila. La pila creix des de direccions altes fins direccions baixes.

Gestió de la pila

A l'inici → es decrementa \$sp per reservar espai

Al final → s'incrementa \$sp per alliberar aquest espai

Regles (ABi) per variables locals

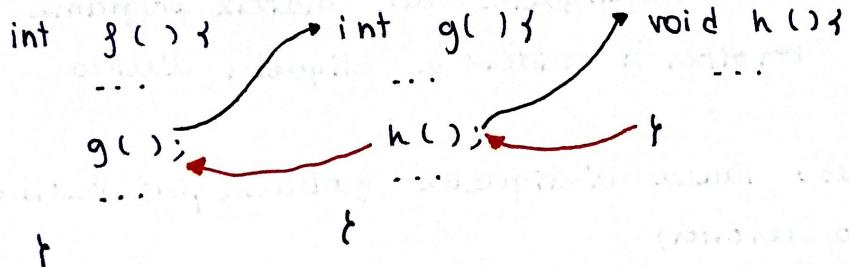
- Escalars → Registres

Excepte si : → Un escalar està afectat per l'operand &
→ No queden més registres

- Dades estructurades (vect., matr.,...) → Al B.A.

A l'hora de guardar dades a la pila, primer guardem els dades locals i després les còpies de seguretat dels registres

Subrutines multinivell



Es important que els dades que hi ha als registres a la funció } no quedin modificats en fer la crida a g(); Per això salvarem els registres segurs.

Exemple:

addiu \$sp, \$sp, -60	}
sw \$s0, 44(\$sp)	
sw \$s1, 48(\$sp)	
sw \$s2, 52(\$sp)	
sw \$ra, 56(\$sp)	
...	
lw \$s0, 44(\$sp)	}
lw \$s1, 48(\$sp)	
lw \$s2, 52(\$sp)	
lw \$ra, 56(\$sp)	
addiu \$sp, \$sp, 60	
jr \$ra	

Pròleg: guardem 60 bytes, els primers 44 per variable locals, i la resta per reg. segurs.

Reg. segurs:

Reg. temporals:

\$t0...t9, \$v0, \$v1, \$a0...\$a3, \$f0...\$f9

→ contingut de la subroutine

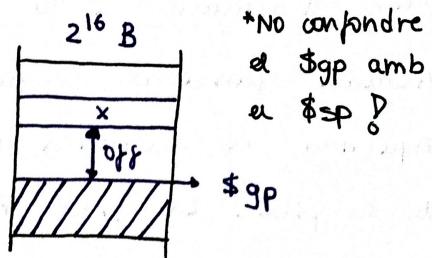
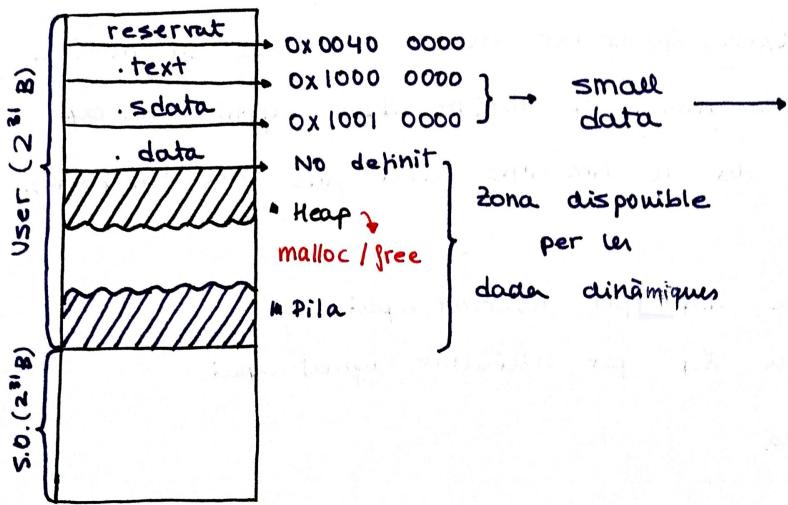
Epòleg: Hem de guardar els dades que hi havia quan hem fet la crida per si han estat modificades. Per sortir de la funció fem jr \$ra



¿Puedo quedarme
este perro?

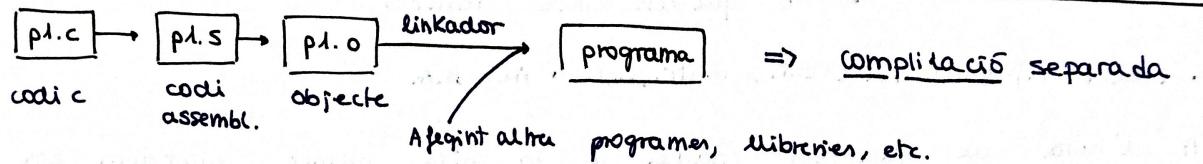
WUOLAH

ESTRUCTURA DE LA MEMORIA



Posicions de memòria on es pot accedir a les variables amb una sola instrucció:
`lw $t0, off($gp)`

COMPILACIÓ, ASSEMBLATGE, MUNTATGE I CÀRREGA D'UN PROGRAMA



Assemblatge: En unir diferents programes certes adreces quedarán modificades → cal arreglar-ho:

- Llista de reubicació: criden a etiquetes del mateix programa
- Llista de referències no resoltres: criden a etiquetes d'altres programes.
- Taula de símbols globals: Taula d'etiquetes globals per resoldre les referències no resoltres

El fitxer objecte conté la capçalera, la selecció de text i dades, la U.R., la U.R.N.R., la T.S.G., i informació de la decompiler.

Per fer el muntatge cal: 1) Resoldre les referències, 2) combinar els mòduls i 3) corregir les adreces de memòria.

A partir d'aquí ja es pot crear el fitxer executable.

TEMA 4: MATRÍUS

Multiplicació d'enters

$$\begin{array}{r}
 \boxed{\$t0} \\
 \times \quad \boxed{\$t1} \\
 \hline
 \boxed{\$hi} \quad \boxed{\$lo}
 \end{array}$$

Per fer multiplicacions fem:

mult \$t0, \$t1 (per multiplicar)

mflo \$t2 (per agafar els 32 bits menors)

mfhi \$t3 (per agafar els 32 bits majors)

MATRÍUS

Estudiem matríus de dues dimensions

NF (nombre de files)

NC (nombre de columnes)

Dades en C:

```
int mat1[NF][NC];
```

```
int mat2[2][3] = {{-1, 2, 0}, {1, -12, 4}}
```

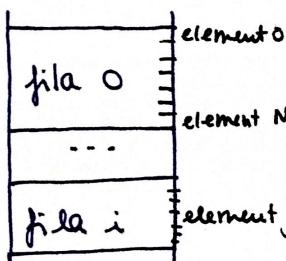
↓

.data

mat1: .space NF·NC·4

mat2: .word -1, 2, 0, 1, -12, 4

Accés a un element aleatori (Elements mida T i mat[NF][NC])



mat[i][j] = @mat[0][0] + (i · NC + j) · T

* Hi pot haver casos on sabem que val i/j (perquè són constants), alleshores podrem fer un accés seqüencial

ACCÉS SEQÜENCIAL (a vector)

Optimització per bucles que recorren els elements d'un vector o matríu. Es pot fer quan la distància en bytes (stride) entre les direccions de dos elements consecutius és constant.

El que hem de fer és:

- Inicialitzar un punter al 1r element

En vectors

- Accedir al element dienterferenciant el punter

el stride
és normalment

- Incrementem el punter pel valor del stride

T.

$$@array[i] = @array[0] + i \cdot T = @array[i-1] + \text{stride}$$

Optimitzacó: Eliminar els usos de la variable d'índic.

Els que fem és calcular la posició final a la qual volem accedir i, en el seu de controlar les iteracions amb la variable "i" (en un `for (...)`) ho controlem comparant la pos. de memòria en què estem amb la qual volem acabar.

`@limit = @array[0] + n_elem · T`

↳ Ens evita una instrucció al bucle.

(Si es vol fer més òptim → convertir a sentència do-while)

ACCÉS SEQÜENCIAL (en matrícies)

Per fer l'accés seqüencial amb matrícies cal saber el stride:

- Accés per files → $\text{stride} = T$
- Accés per columnes → $\text{stride} = NC \cdot T$
- Accés per diagonal principal → $\text{stride} = (NC+1) \cdot T$
- Accés per diagonal secundària → $\text{stride} = (NC-1) \cdot T$

Un cop sabem l'stride podem aplicar el mateix criteri que en vectors.

TEMA 5: ARITMÈTICA D'ENTERS I COMA FLOTANT

OVERFLOW

a, b representables en n bits

• $s = a + b$, si $s \notin [-2^{n-1}, 2^{n-1}-1]$ \Rightarrow hi ha overflow

Hi ha overflow quan a i b són del mateix signe i s de l'altra signe.

• $d = a - b \rightarrow a = d + b$

Hi ha overflow si la diferència és del mateix signe que "b" però diferent que "a".

• $v = \underline{(A_{31} + B_{31})} \wedge (S_{31} + A_{31}) = \begin{cases} 1, & \text{si hi ha overflow en la suma} \\ 0, & \text{si no hi ha overflow en la suma} \end{cases}$

MULTIPLICACIÓ D'ENTERS

1) Calcular magnituds 2) Multiplicar 3) Determinar el signe

Pseudocodi multiplicació

$P = 0;$

$MD_{31,0} = A$

$MD_{63,32} = 0$

$MR = B$

for (int i = 1; i < 32; ++i) {

 if ($MR_0 == 1$) $P = P + MD;$

$MD = MD \ll 1;$

$MR = MR \gg 1;$

}

Latència multiplicació:

1 cicle inicial + 32 cicles = 33 cicles

En MIPS:

mult \$t0,\$t1 } Hi ha overflow si
mflo \$t2 } el número no es
 pot representar en
 32 bits

Per multiplicar naturals:

multiu \$t0,\$t1



¿Puedo quedarme
este perro?

WUOLAH

Divisió d'enters

- 1) calcular magnituds
- 2) dividir
- 3) Determinar signe

Exemple de divisió:

$$\begin{array}{r} 1011 \\ -0000 \\ \hline 1011 \end{array}$$

$$\begin{array}{r} -0010 \\ \hline 0011 \end{array}$$

$$\begin{array}{r} -0000 \\ \hline 0011 \end{array}$$

$$\begin{array}{r} -0010 \\ \hline 0001 \end{array}$$

$$\begin{array}{r} 0010 \\ 0101 \\ \hline \end{array}$$

Quocient = 101

Residu = 1

MIPS:

div \$t0, \$t1

\$t0 ← \$t0 / \$t1

\$hi ← \$t0 % \$t1

Pseudocodi divisió (amb restauració)

$Q = 0;$

$R_{63,32} = 0; R_{31,0} = X;$

$D_{63,32} = y; D_{31,0} = 0;$

for (int i = 1; i <= 32; ++i) {

 D = D >> 1;

 R = R - D;

 if (R < 0) {

 R = R + D;

 Q = Q << 1;

 } else {

 Q = (Q << 1) | 1;

}

- Divisió amb restauració: En cada resta es guarda la diferència en R. Si R és negatiu, es restauren R i el Quocient.

REPRESENTACIÓ DE REALS EN COMA FLOTANT

Codificar números reals de forma aproximada:

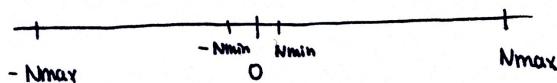
$$X = \pm 1^{\text{signe}} \cdot \underbrace{f \dots}_{\text{fracció (F)}} \cdot \underbrace{2^E}_{\text{base exponent (E)}} \cdot \underbrace{1 + F}_{\text{potència}}$$

mantissa

Magnitud

$$\left. \begin{array}{l} S = \begin{cases} 1, & \ominus \\ 0, & \oplus \end{cases} \\ X = (-1)^S \cdot (1 + F) \cdot 2^E \\ \boxed{\begin{array}{|c|c|c|} \hline S & E & F \\ \hline \end{array}} \end{array} \right\}$$

Rang: $0 \leq |X| < 2^{E+1}$



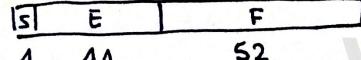
$$\begin{aligned} |N_{\max}| &= \pm 1^{\text{signe}} 111\dots 11 \cdot 2^{E_{\max}} & \Rightarrow |X| > N_{\max} \rightarrow \text{OVERFLOW} \\ |N_{\min}| &= \pm 1^{\text{signe}} 00\dots 00 \cdot 2^{E_{\min}} & \Rightarrow |X| < N_{\min} \rightarrow \text{UNDERFLOW} \end{aligned}$$

Format IEEE - 754

Simple precision: 32 bits →



Double precision: 64 bits →



Exponent: En format excess a 127 $x_5 = x_u - 127$

Casos reservats del format IEEE-754:

1) zero $\boxed{\begin{matrix} S & E & F \end{matrix}}$ $\rightarrow \pm 0^0$
zeros zeros

2) Infinit $\boxed{\begin{matrix} S & E & F \end{matrix}}$ $\rightarrow \pm \infty$
uns zeros

3) NaN (Not a Number) $\boxed{\begin{matrix} S & E & F \end{matrix}}$ $\rightarrow \frac{\infty}{\infty}, \frac{0}{0}, \sqrt{-1}$
uns dif. de zero

4) Denormals: Aquells números tq $|x| \in (0, 2^{E_{min}})$ $\rightarrow \boxed{\begin{matrix} S & E & F \end{matrix}}$
zeros dif de zero

Arrodoniment i Error de Precisió

$$x = 1^1 00\dots 00 101 \quad \begin{cases} N_1 = 1^1 00\dots 00 \\ N_2 = 1^1 00\dots 01 \end{cases}$$

a) Truncament: $N_1 \rightarrow \epsilon_{max} = |N_2 - N_1| = 0^1 00\dots 01 \cdot 2^E = 2^{E-23} = 1 \text{ ULP}$

b) Al més proxim: $\epsilon_{max} = \frac{|N_2 - N_1|}{2} = \frac{1}{2} \text{ ULP}$

c) Cap al +∞: $\rightarrow N_2$

d) Cap al -∞: $\rightarrow N_1$

* Algorisme per arrodonir

$1, [....] x --$ $\begin{cases} x = 1 \rightarrow 1^1 [...+1] & \text{Si } x=1 \text{ però segueixen zeros,} \\ x = 0 \rightarrow 1^1 [...] & \text{jarem que acabi en zero} \end{cases}$
el resultat

UNDERFLOW I NOMBRES DE NORMALS

Per valors entre $-2^{E_{min}}$ i $2^{E_{max}}$ l'error màxim de precisió és constant, és sempre: $\eta_{max} = 2^{-23}$.

Però en l'intervall $(0, 2^{E_{min}})$, els nombres normalitzats tenen un error molt superior, $\eta_{max} = 1$. En els nombres denormals, l'error augmenta conforme ens aproguem a $2^{E_{min}-23}$ tq $\eta_{max} = \frac{2^{E_{min}-23}}{|x|}$

CONVERSIÓ DE BASE 10 A BASE 2

$$x = -2053 \cdot 68.$$

$$2053 = 1000\ 0000\ 0101$$

0'68	$\rightarrow 1'36$	$1'76$	$0'16$	$0'56$	(calculen 12 bits + 3)
	$0'72$	$1'52$	$0'32$	$1'12$	Els 3 bits de més
	$1'44$	$1'04$	$0'64$	$0'24$	s'agafen per saber
	$0'88$	$0'08$	$1'28$		si s'ha d'arredonir

$$0'68 = 1010\ 1110\ 0001\ldots010$$

Ara els unim (24 bits en total)

$$x = -1,000,000,0,0101,1010,1110,0001 \cdot 2^0$$

$$x = -1,000,0000,0101,1010,1110,0001 \cdot 2^11$$

$$E_{10} = 11 \rightarrow 11 + 127 = 138 \rightarrow 138_2 = 10001010$$

$$\hookrightarrow x = .1\ 1000\ 1010\ 0000\ 0000; 0101, 1010, 1110, 0001,$$

$$x = 0x C5005A E1$$

CONVERSIÓ DE BASE 2 A BASE 10

$$x = 0x 4581 4140 = 1000\ 0101\ 1000\ 0001\ 0100\ 0001\ 0100\ 0000$$

$$E = 1000\ 1011 = 139 \rightarrow 139 - 127 = 12 \rightarrow E = 12$$

$$x = 1,000,0001,0100,0001,0100,0000 \cdot 2^{12}$$

$$x = 1000\ 0001\ 0100\ 0,001\ 0100\ 0000$$

$$\text{Part entera: } 1000000101000 = 8 + 32 + 4096 = 4036$$

$$\text{Part fraccionària: } 0'00101 = 101 \cdot 2^{-5} = \frac{5}{32} \rightarrow \frac{50}{32} \underline{132} \\ 190\ 0'15625$$

$$\text{Resultat: } x = +4036'15625$$

BITS DE GUARDA

G → Guarda

R → Round

S → Sticky

1' [....] GRS

Sticky es un bit or de tots els bits de la sèrie dreta.

Casos:

1' [...] ORS → Si G = 0, es deixa igual → 1' [...]

1' [...] 101 → Si G = 1, R = 0, S = 1 → Arrodonim 1' [... + 1]

1' [...] 100 → Si G = 1, R = 0, S = 0 → Arrodonim a fi que l'últim bit sigui 0 → 1' [... 0]

COMA FLOTANT MIPS

Còpia entre registres

mov.s fd, fs mfc1 rt, fs mtcl rt, fs
↳ fd = fs ↳ rt = fs ↳ fs = rt

Accés a memòria

lwcl ft, off16(rs) float swcl ft, off16(rs) float
ldcl ft, off16(rs) double sdc1 ft, off16(rs) double

Aritmètiques

add.s fd, fs, ft sub.s fd, fs, ft mul.s fd, fs, ft

div.s fd, fs, ft , i en corresponents ambd

Comparacions

c. xx.s fs, ft → Si es compleix la condició posa el bit de control i status a 1. Si no, a 0.
xx = {eq, lt, le}

↓ ↓
cc = 1 cc = 0

Salts

bclt etiq → salta si cc = 1

bclf etiq → salta si cc = 0

PARÀMETRES I RESULTATS

Paràmetres → \$j12 ... \$j14

Resultat → \$j0

Registers segurs → \$j20 ... \$j31

- Per passar números de coma flotant:

Ex: 1'0 → 0x 3F80 0000

li \$t0, 0x 3F80 0000

mtcl \$t0, \$j16

.data

num: .float 1.0

.text

la \$t0, num

lwcl \$j16, 0(\$t0)