Repàs: Examen parcial 2017-18 Q2

Joan Manuel Parcerisa

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

FIB

Pregunta 1. Matrius

Donada la següent funció `foo` en llenguatge C:

```
void foo(short M[][64], unsigned int k) {
        int i;
        short aux = M[k][8];
        for (i=5; i<45; i+=5){
                M[45-i][i] = aux;
        }
}
```

Completa el següent codi MIPS omplint les caselles en blanc perquè sigui equivalent a l'anterior codi en alt nivell, tenint en compte que els elements de la matriu M s'accedeixen utilitzant la tècnica **d'accés seqüencial** sempre que es pot, usant el registre `$t1` com a punter. Aquest punter `$t1` s'inicialitza amb l'adreça de l'element `M[40][5]`. Al codi se li ha aplicat l'optimització de conversió d'un bucle `for` en un `do_while` i l'eliminació de la variable d'inducció.

```
        sll     $t0, $a1, [          ]
        addu    $t0, $t0, $a0
        lh      $t2, [          ]($t0)          # aux = M[k][8];
        addiu   $t1, $a0, [          ]          # @M[40][5]
        addiu   $t3, $a0, [          ]          # adreça final del punter $t1
        b       cond
bucle:  sh      $t2, [          ]($t1)
        addiu   $t1, $t1, [          ]
cond:   bgtu    $t1, $t3, bucle
        jr      $ra
```

Pregunta 1. Matrius

```
void foo(short M[][64], unsigned int k) {
        int i;
        short aux = M[k][8];
        for (i=5; i<45; i+=5){
                M[45-i][i] = aux;
        }
}
```

```
sll     $t0, $a1, 7
addu    $t0, $t0, $a0
lh      $t2, 16($t0)        # aux = M[k][8];
addiu   $t1, $a0,           # @M[40][5]
addiu   $t3, $a0,           # adreça final del punter $t1
```

$$@M[k][8] = @M + k*64*2 + 8*2$$

$$= \$a0 + \$a1*128 + 16$$

Pregunta 1. Matrius

```
void foo(short M[][64], unsigned int k) {
        int i;
        short aux = M[k][8];
        for (i=5; i<45; i+=5){
            M[45-i][i] = aux;
        }
}
```

```
sll     $t0, $a1, 7
addu    $t0, $t0, $a0
lh      $t2, 16($t0)        # aux = M[k][8];
addiu   $t1, $a0, 5130      # @M[40][5]
addiu   $t3, $a0,           # adreça final del punter $t1
```

$$@M[k][8] = @M + k*64*2 + 8*2$$

$$= \$a0 + \$a1*128 + 16$$

$$@M[40[5] = @M + 40*64*2 + 5*2$$

$$= \$a0 + \mathbf{5130}$$

Pregunta 1. Matrius

```
void foo(short M[][64], unsigned int k) {
        int i;
        short aux = M[k][8];
        for (i=5; i<45; i+=5){
            M[45-i][i] = aux;
        }
}
```

```
sll     $t0, $a1, 7
addu    $t0, $t0, $a0
lh      $t2, 16($t0)        # aux = M[k][8];
addiu   $t1, $a0, 5130      # @M[40][5]
addiu   $t3, $a0, 90        # adreça final del punter $t1
```

$$@M[k][8] = @M + k*64*2 + 8*2$$

$$= \$a0 + \$a1*128 + 16$$

$$@M[40[5] = @M + 40*64*2 + 5*2$$

$$= \$a0 + 5130$$

$$\text{Adreça final del punter (per a i=45)} = @M[0][45]$$

$$= @M + 45*2$$

$$= \$a0 + \mathbf{90}$$

Pregunta 1. Matrius

```
void foo(short M[][64], unsigned int k) {
        int i;
        short aux = M[k][8];
        for (i=5; i<45; i+=5){
            M[45-i][i] = aux;
        }
}
```

```
            b       cond
bucle:      sh      $t2, 0 ($t1)
            addiu   $t1, $t1,
cond:       bgtu    $t1, $t3, bucle
            jr      $ra
```

Emmagatzemar aux en M[45-i][i] (és on apunta $t1)

```
sh      $t2, 0 ($t1)
```

Pregunta 1. Matrius

```
void foo(short M[][64], unsigned int k) {
        int i;
        short aux = M[k][8];
        for (i=5; i<45; i+=5){
                M[45-i][i] = aux;
        }
}
```

```
                b       cond
bucle:          sh      $t2, 0 ($t1)
                addiu   $t1, $t1, -630
cond:           bgtu    $t1, $t3, bucle
                jr      $ra
```

Emmagatzemar aux en M[45-i][i] (és on apunta $t1)

```
        sh      $t2, 0 ($t1)
```

Incrementar punter pel valor del stride és decrementar 5 files i incrementar 5 columnes

```
stride = -5*64*2 + 5*2
       = -630
```

Pregunta 2. Subrutines

Donades les següents declaracions en C:

```
unsigned int func2(int x, char M[][4], int *y);
unsigned int *pglob;
int func1(int par1, char par2[][4], int par3, int *par4) {
        int loc1;
        loc1 = par1 + par3;
        *pglob = func2 (*par4, &par2[3][0], par4);
        return *par4 - loc1;
}
```

A continuació es mostra una traducció de la funció `func1` a llenguatge MIPS que està incompleta. Llegiu-la amb atenció, a fi de contestar correctament a les preguntes de més avall.

```
func1:
        addiu   $sp, $sp, -12
        sw      $ra, 8($sp)
        sw      $s0, 4($sp)
        sw      $s1, 0($sp)
        move    $s1, $a3                        # copia del punter par4

        # loc1 = par1 + par3
        addu    $s0, $a0, $a2

        # *pglob = func2 (*par4, &par2[3][0], par4);
          # CAIXA 1
          # (pas de paràmetres a func2)
        jal func2
          # CAIXA 2
          # (emmagatzema resultat de func2)
        # return *par4 - loc1;
          # CAIXA 3
          # (sentència return)

        lw      $ra, 8($sp)
        lw      $s0, 4($sp)
        lw      $s1, 0($sp)
        addiu   $sp, $sp, 12
        jr      $ra
```

a) Completa la CAIXA 1, en llenguatge assemblador de M crida a la funció `func2` corresponent a la següent sentèn

b) Completa la CAIXA 2, en llenguatge assemblador de crida i el seu emmagatzematge a l'adreça apuntada següent fragment de sentència del cos de la subrutina fu

c) Completa la CAIXA 3, en llenguatge assemblador de M sentència: `return *par4 - loc1;`

Pregunta 2. Subrutines

Donades les següents declaracions en C:

```
unsigned int func2(int x, char M[][4], int *y);
unsigned int *pglob;
int func1(int par1, char par2[][4], int par3, int *par4) {
        int loc1;
        loc1 = par1 + par3;
        *pglob = func2 (*par4, &par2[3][0], par4);
        return *par4 - loc1;
}
```

- Guardem les variables locals a la pila o en registres?
  - `loc1`: és escalar, i no apareix enlloc darrera l'operador & → en registre

Pregunta 2. Subrutines

Donades les següents declaracions en C:

```
unsigned int func2(int x, char M[][4], int *y);
unsigned int *pglob;
int func1(int par1, char par2[][4], int par3, int *par4) {
        int loc1;
        loc1 = par1 + par3;
        *pglob = func2 (*par4, &par2[3][0], par4);
        return *par4 - loc1;
}
```

- Guardem les variables locals a la pila o en registres?
  - `loc1`: és escalar, i no apareix enlloc darrera l'operador & → en registre
- Quines variables locals, paràmetres i resultats intermedis van en registres segurs $s?
  - `loc1`: en registre segur

Pregunta 2. Subrutines

Donades les següents declaracions en C:

```
unsigned int func2(int x, char M[][4], int *y);
unsigned int *pglob;
int func1(int par1, char par2[][4], int par3, int *par4) {
        int loc1;
        loc1 = par1 + par3;
        *pglob = func2 (*par4, &par2[3][0], par4);
        return *par4 - loc1;
}
```

- Guardem les variables locals a la pila o en registres?
  - loc1: és escalar, i no apareix enlloc darrera l'operador & → en registre
- Quines variables locals, paràmetres i resultats intermedis van en registres segurs $s?
  - `loc1`: en registre segur
  - `par4`: en registre segur

Pregunta 2. Subrutines

Donades les següents declaracions en C:

```
unsigned int func2(int x, char M[][4], int *y);
unsigned int *pglob;
int func1(int par1, char par2[][4], int par3, int *par4) {
        int loc1;
        loc1 = par1 + par3;
        *pglob = func2 (*par4, &par2[3][0], par4);
        return *par4 - loc1;
}
```

- Guardem les variables locals a la pila o en registres?
  - loc1: és escalar, i no apareix enlloc darrera l'operador & → en registre
- Quines variables locals, paràmetres i resultats intermedis van en registres segurs $s?
  - `loc1`: en registre segur
  - `par4`: en registre segur
  - `pglob`: variable global → resideix a la memòria

Pregunta 2. Subrutines

Donades les següents declaracions en C:

```
unsigned int func2(int x, char M[][4], int *y);
unsigned int *pglob;
int func1(int par1, char par2[][4], int par3, int *par4) {
        int loc1;
        loc1 = par1 + par3;
        *pglob = func2 (*par4, &par2[3][0], par4);
        return *par4 - loc1;
}
```

- Guardem les variables locals a la pila o en registres?
  - loc1: és escalar, i no apareix enlloc darrera l'operador & → en registre
- Quines variables locals, paràmetres i resultats intermedis van en registres segurs $s?
  - `loc1`: en registre segur → **`$s0`**
  - `par4`: en registre segur → **`$s1`**
  - `pglob`: variable global → resideix a la memòria
- Bloc d'activació: sols conté còpies de registres
  - $s0, $s1, $ra (12 bytes en total)

Pregunta 2. Subrutines

A continuació es mostra una traducció de la funció `func1` a llenguatge MIPS que està incompleta. Llegiu-la amb atenció, a fi de contestar correctament a les preguntes de més avall.

```
func1:
        addiu   $sp, $sp, -12
        sw      $ra, 8($sp)
        sw      $s0, 4($sp)
        sw      $s1, 0($sp)
        move    $s1, $a3                        # copia del punter par4

        # loc1 = par1 + par3
        addu    $s0, $a0, $a2

        # *pglob = func2 (*par4, &par2[3][0], par4);

        # CAIXA 1
        # (pas de paràmetres a func2)

        jal func2

        # CAIXA 2
        # (emmagatzema resultat de func2)

        # return *par4 - loc1;

        # CAIXA 3
        # (sentència return)

        lw      $ra, 8($sp)
        lw      $s0, 4($sp)
        lw      $s1, 0($sp)
        addiu   $sp, $sp, 12
        jr      $ra
```

Pregunta 2. Subrutines

A continuació es mostra una traducció de la funció `func1` a llenguatge MIPS que està incompleta. Llegiu-la amb atenció, a fi de contestar correctament a les preguntes de més avall.

```
func1:
        addiu   $sp, $sp, -12
        sw      $ra, 8($sp)
        sw      $s0, 4($sp)
        sw      $s1, 0($sp)
        move    $s1, $a3        # copia del punter par4

        # loc1 = par1 + par3
        addu    $s0, $a0, $a2

        # *pglob = func2 (*par4, &par2[3][0], par4);

          # CAIXA 1
          # (pas de paràmetres a func2)

        jal func2

          # CAIXA 2
          # (emmagatzema resultat de func2)

        # return *par4 - loc1;

          # CAIXA 3
          # (sentència return)

        lw      $ra, 8($sp)
        lw      $s0, 4($sp)
        lw      $s1, 0($sp)
        addiu   $sp, $sp, 12
        jr      $ra
```

Pròleg:
Salvar $s0, $s1, $ra a la pila

Copiar `par4` a $s1

`loc1` es guarda en $s0

Epíleg:
Restaurar registres i tornar

Pregunta 2. Subrutines

```
int func1(int par1, char par2[][4], int par3, int *par4) {
```

a) Completa la CAIXA 1, en llenguatge assemblador de MIPS, amb el pas de paràmetres per a la crida a la funció `func2` corresponent a la següent sentència del cos de la subrutina `func1`.

```
... = func2 (*par4, &par2[3][0], par4);
```

```
jal    func2
```

- 1er paràmetre: `*par4` (desreferenciar punter)

```
lw          $a0, 0($a3)
```

Pregunta 2. Subrutines

```
int func1(int par1, char par2[][4], int par3, int *par4) {
```

a) Completa la CAIXA 1, en llenguatge assemblador de MIPS, amb el pas de paràmetres per a la crida a la funció `func2` corresponent a la següent sentència del cos de la subrutina `func1`.

```
... = func2 (*par4, &par2[3][0], par4);
```

```
jal    func2
```

- 1er paràmetre: `*par4` (desreferenciar punter)

```
lw          $a0, 0($a3)
```

- 2on paràmetre: `&par2[3][0]` (calcular adreça)

```
addiu       $a1, $a1, 12
```

Pregunta 2. Subrutines

```
int func1(int par1, char par2[][4], int par3, int *par4) {
```

a) Completa la CAIXA 1, en llenguatge assemblador de MIPS, amb el pas de paràmetres per a la crida a la funció `func2` corresponent a la següent sentència del cos de la subrutina `func1`.

```
... = func2 (*par4, &par2[3][0], par4);
```

```
lw      $a0, 0($a3)
addiu   $a1, $a1, 12
move    $a2, $a3

jal   func2
```

- 1er paràmetre: `*par4` (desreferenciar punter)

```
lw          $a0, 0($a3)
```

- 2on paràmetre: `&par2[3][0]` (calcular adreça)

```
addiu       $a1, $a1, 12
```

- 3er paràmetre: `par4` (copiar tal qual)

```
move        $a2, $a3
```

Pregunta 2. Subrutines

```
int func1(int par1, char par2[][4], int par3, int *par4) {
```

b) Completa la CAIXA 2, en llenguatge assemblador de MIPS, amb la recollida del resultat de la crida i el seu emmagatzematge a l'adreça apuntada per la variable `pglob`, corresponent al següent fragment de sentència del cos de la subrutina `func1`: `*pglob = func2( ...`

```
jal    func2
```

1. Carreguem el punter de la memòria en el registre `$t1`:

```
la      $t0, pglob
lw      $t1, 0($t0)
```

Pregunta 2. Subrutines

```
int func1(int par1, char par2[][4], int par3, int *par4) {
```

b) Completa la CAIXA 2, en llenguatge assemblador de MIPS, amb la recollida del resultat de la crida i el seu emmagatzematge a l'adreça apuntada per la variable `pglob`, corresponent al següent fragment de sentència del cos de la subrutina `func1`: `*pglob = func2( ...`

```
jal    func2
```

1. Carreguem el punter de la memòria en el registre `$t1`:

```
la      $t0, pglob
lw      $t1, 0($t0)
```

2. Desreferenciem el punter `$t1` (escrivint el resultat de `func2` a memòria)

```
sw      $v0, 0($t1)
```

Pregunta 2. Subrutines

```
int func1(int par1, char par2[][4], int par3, int *par4) {
```

b) Completa la CAIXA 2, en llenguatge assemblador de MIPS, amb la recollida del resultat de la crida i el seu emmagatzematge a l'adreça apuntada per la variable `pglob`, corresponent al següent fragment de sentència del cos de la subrutina `func1`: `*pglob = func2( ...`

```
jal    func2
```

```
la      $t0, pglob
lw      $t1, 0($t0)
sw      $v0, 0($t1)
```

1. Carreguem el punter de la memòria en el registre `$t1`:

```
la      $t0, pglob
lw      $t1, 0($t0)
```

2. Desreferenciem el punter `$t1` (escrivint el resultat de `func2` a memòria)

```
sw      $v0, 0($t1)
```

Pregunta 2. Subrutines

```
int func1(int par1, char par2[][4], int par3, int *par4) {
```

c) Completa la CAIXA 3, en llenguatge assemblador de MIPS, amb la traducció de la darrera sentència: `return *par4 - loc1;`

- Desreferenciem `par4` (que el tenim en `$s1`)

```
lw      $v0, 0($s1)
```

Pregunta 2. Subrutines

```
int func1(int par1, char par2[][4], int par3, int *par4) {
```

c) Completa la CAIXA 3, en llenguatge assemblador de MIPS, amb la traducció de la darrera sentència: `return *par4 - loc1;`

```
lw      $v0, 0($s1)
subu    $v0, $v0, $s0
```

- Desreferenciem `par4` (que el tenim en `$s1`)

```
lw      $v0, 0($s1)
```

- I li restem `loc1` (que el tenim en `$s0`)

```
subu    $v0, $v0, $s0
```

Pregunta 3. if-then-else

Donada la següent sentència escrita en alt nivell en C:

```
if ((a%2==0)||(b!=0)) && ((b<=a)||(a>0)))
        z=a;
else
        z=b;
```

Completa el següent fragment de codi MIPS, que tradueix l'anterior sentència, escrivint en cada calaix un mnemònic d'instrucció o macro, etiqueta, registre o immediat. Les variables `a`, `b` i `z` són de tipus `int` i estan inicialitzades i guardades als registres `$t0`, `$t1` i `$t2`, respectivament.

- Calcular `a%2` equival a seleccionar el bit 0 amb una and bit a bit:

  `andi $t3, $t0, 1`

```
        andi    $t3, $t0,  1
etq1:   beq     $t3, $zero,
etq2:           $t1, $zero,
etq3:           $t1, $t0,
etq4:   ble     $t0, $zero,
etq5:   move    $t2, $t0
        b
etq6:   move    $t2, $t1
etq7:
```

Pregunta 3. if-then-else

Donada la següent sentència escrita en alt nivell en C:

```
if ((a%2==0)||(b!=0)) && ((b<=a)||(a>0)))
        z=a;
else
        z=b;
```

Completa el següent fragment de codi MIPS, que tradueix l'anterior sentència, escrivint en cada calaix un mnemònic d'instrucció o macro, etiqueta, registre o immediat. Les variables `a`, `b` i `z` són de tipus `int` i estan inicialitzades i guardades als registres `$t0`, `$t1` i `$t2`, respectivament.

- Calcular `a%2` equival a seleccionar el bit 0 amb una and bit a bit:
  `andi $t3, $t0, 1`
- En cas de ser `a%2=0`, ja no s'avalua `b!=0`, però sí que cal avaluar la part dreta de la AND: `b<=a` → saltem a **etq3**

```
        andi    $t3, $t0,       1
etq1:   beq     $t3, $zero,     etq3
etq2:           $t1, $zero,
etq3:           $t1, $t0,
etq4:   ble     $t0, $zero,
etq5:   move    $t2, $t0
        b
etq6:   move    $t2, $t1
etq7:
```

Pregunta 3. if-then-else

Donada la següent sentència escrita en alt nivell en C:

```
if (((a%2==0)||(b!=0)) && ((b<=a)||(a>0)))
        z=a;
else
        z=b;
```

Completa el següent fragment de codi MIPS, que tradueix l'anterior sentència, escrivint en cada calaix un mnemònic d'instrucció o macro, etiqueta, registre o immediat. Les variables `a`, `b` i `z` són de tipus `int` i estan inicialitzades i guardades als registres `$t0`, `$t1` i `$t2`, respectivament.

- Calcular `a%2` equival a seleccionar el bit 0 amb una and bit a bit:
  `andi $t3, $t0, 1`
- En cas de ser `a%2=0`, ja no s'avalua `b!=0`, però sí que cal avaluar la part dreta de la AND: `b<=a` → saltem a **etq3**
- En cas contrari avaluem `b!=0`, i en cas de ser **fals**, ja no s'avaluarà la part dreta de la AND, sinó que saltarem a *else:*
  `beq $t1,$zero,etq6`

```
        andi    $t3, $t0,      1
etq1:   beq     $t3, $zero,    etq3
etq2:   beq     $t1, $zero,    etq6
etq3:           $t1, $t0,
etq4:   ble     $t0, $zero,
etq5:   move    $t2, $t0
        b
etq6:   move    $t2, $t1
etq7:
```

Pregunta 3. if-then-else

Donada la següent sentència escrita en alt nivell en C:

```
if (((a%2==0)||(b!=0)) && ((b<=a)||(a>0)))
        z=a;
else
        z=b;
```

Completa el següent fragment de codi MIPS, que tradueix l'anterior sentència, escrivint en cada calaix un mnemònic d'instrucció o macro, etiqueta, registre o immediat. Les variables `a`, `b` i `z` són de tipus `int` i estan inicialitzades i guardades als registres `$t0`, `$t1` i `$t2`, respectivament.

- Avaluem `b<=a`, i en cas de ser **cert** ja no s'avalua `a>0` sinó que se salta al "then":
  `z=a:`
  `ble` `$t1,$t0,`**`etq5`**

```
        andi    $t3, $t0,     [ 1 ]
etq1:   beq     $t3, $zero,   [etq3]
etq2:   [beq]   $t1, $zero,   [etq6]
etq3:   [ble]   $t1, $t0,     [etq5]
etq4:   ble     $t0, $zero,   [   ]
etq5:   move    $t2, $t0
        b       [   ]
etq6:   move    $t2, $t1
etq7:
```

Pregunta 3. if-then-else

Donada la següent sentència escrita en alt nivell en C:

```
if (((a%2==0)||(b!=0)) && ((b<=a)|| (a>0)))
        z=a;
else
        z=b;
```

Completa el següent fragment de codi MIPS, que tradueix l'anterior sentència, escrivint en cada calaix un mnemònic d'instrucció o macro, etiqueta, registre o immediat. Les variables `a`, `b` i `z` són de tipus `int` i estan inicialitzades i guardades als registres `$t0`, `$t1` i `$t2`, respectivament.

- Avaluem `b<=a`, i en cas de ser **cert** ja no s'avalua `a>0` sinó que se salta al "then":
  `z=a`
  `ble` `$t1,$t0,`**`etq5`**
- En cas contrari, avaluem `a>0`, i en cas de ser **fals**, és a dir que `a<=0`, salta a "else": **etq6**

```
        andi    $t3, $t0,      1
etq1:   beq     $t3, $zero,    etq3
etq2:   beq     $t1, $zero,    etq6
etq3:   ble     $t1, $t0,      etq5
etq4:   ble     $t0, $zero,    etq6
etq5:   move    $t2, $t0
        b
etq6:   move    $t2, $t1
etq7:
```

Pregunta 3. if-then-else

Donada la següent sentència escrita en alt nivell en C:

```
if (((a%2==0)||(b!=0)) && ((b<=a)|| (a>0)))
        z=a;
else
        z=b;
```

Completa el següent fragment de codi MIPS, que tradueix l'anterior sentència, escrivint en cada calaix un mnemònic d'instrucció o macro, etiqueta, registre o immediat. Les variables `a`, `b` i `z` són de tipus `int` i estan inicialitzades i guardades als registres `$t0`, `$t1` i `$t2`, respectivament.

- Avaluem `b<=a`, i en cas de ser **cert** ja no s'avalua `a>0` sinó que se salta al "then":
  `z=a`
  `ble` `$t1,$t0,`**`etq5`**
- En cas contrari, avaluem `a>0`, i en cas de ser **fals**, és a dir que `a<=0`, salta a "else": **etq6**
- Un cop executat el "then" cal saltar al final: **etq7**

```
        andi    $t3, $t0,   1
etq1:   beq     $t3, $zero, etq3
etq2:   beq     $t1, $zero, etq6
etq3:   ble     $t1, $t0,   etq5
etq4:   ble     $t0, $zero, etq6
etq5:   move    $t2, $t0
        b       etq7
etq6:   move    $t2, $t1
etq7:
```

Pregunta 4. Rendiment i Consum

Suposem un programa que s'executa sobre un processador funcionant a una freqüència de 2GHz, el qual dissipa una potència de 40W. La següent taula mostra, per a cada tipus d'instrucció, el nombre d'instruccions executades i el CPI, referents a l'execució d'aquest programa:

Tipus d'instr.	Nombre. d'instr.	CPI
Memòria	$1{,}5 \times 10^9$	3
Salts	$1{,}0 \times 10^9$	4
Resta	$2{,}5 \times 10^9$	1

a) Calcula el CPI promig de tot el programa

CPI =

Pregunta 4. Rendiment i Consum

Suposem un programa que s'executa sobre un processador funcionant a una freqüència de 2GHz, el qual dissipa una potència de 40W. La següent taula mostra, per a cada tipus d'instrucció, el nombre d'instruccions executades i el CPI, referents a l'execució d'aquest programa:

Tipus d'instr.	Nombre. d'instr.	CPI
Memòria	$1{,}5 \times 10^9$	3
Salts	$1{,}0 \times 10^9$	4
Resta	$2{,}5 \times 10^9$	1

a) Calcula el CPI promig de tot el programa

CPI = ____

- CPI promig = nombre_total_de_cicles / nombre_total_instruccions
  - n_{cicles}
    $= (1{,}5 \times 10^9) \times 3 + (1{,}0 \times 10^9) \times 4 + (2{,}5 \times 10^9) \times 1$
    $= (1{,}5 \times 3 + 1{,}0 \times 4 + 2{,}5 \times 1) \times 10^9$
    $= 11{,}0 \times 10^9$

Pregunta 4. Rendiment i Consum

Suposem un programa que s'executa sobre un processador funcionant a una freqüència de 2GHz, el qual dissipa una potència de 40W. La següent taula mostra, per a cada tipus d'instrucció, el nombre d'instruccions executades i el CPI, referents a l'execució d'aquest programa:

Tipus d'instr.	Nombre. d'instr.	CPI
Memòria	$1,5 \times 10^9$	3
Salts	$1,0 \times 10^9$	4
Resta	$2,5 \times 10^9$	1

a) Calcula el CPI promig de tot el programa

CPI = **2,2**

- CPI promig = nombre_total_de_cicles / nombre_total_instruccions
  - n_{cicles}
    $= (1,5 \times 10^9) \times 3 + (1,0 \times 10^9) \times 4 + (2,5 \times 10^9) \times 1$
    $= (1,5 \times 3 + 1,0 \times 4 + 2,5 \times 1) \times 10^9$
    $= 11,0 \times 10^9$
  - $n_{instruccions}$
    $= (1,5 + 1,0 + 2,5) \times 10^9$
    $= 5,0 \times 10^9$
- CPI = $(11,0 \times 10^{9)} / (5,0 \times 10^9) =$ **2,2**

Pregunta 4. Rendiment i Consum

Suposem un programa que s'executa sobre un processador funcionant a una freqüència de 2GHz, el qual dissipa una potència de 40W. La següent taula mostra, per a cada tipus d'instrucció, el nombre d'instruccions executades i el CPI, referents a l'execució d'aquest programa:

Tipus d'instr.	Nombre. d'instr.	CPI
Memòria	$1{,}5 \times 10^9$	3
Salts	$1{,}0 \times 10^9$	4
Resta	$2{,}5 \times 10^9$	1

b) Calcula el temps d'execució del programa, en segons

t_{exe} = ____ s

- $t_{exe} = n_{cicles} / f_{clock}$
  - n_{cicles} l'hem calculat a l'apartat (a) $= 11{,}0 \times 10^9$
  - $f_{clock} = 2$ GHz

Pregunta 4. Rendiment i Consum

Suposem un programa que s'executa sobre un processador funcionant a una freqüència de 2GHz, el qual dissipa una potència de 40W. La següent taula mostra, per a cada tipus d'instrucció, el nombre d'instruccions executades i el CPI, referents a l'execució d'aquest programa:

Tipus d'instr.	Nombre. d'instr.	CPI
Memòria	$1{,}5 \times 10^9$	3
Salts	$1{,}0 \times 10^9$	4
Resta	$2{,}5 \times 10^9$	1

b) Calcula el temps d'execució del programa, en segons

t_{exe} = **5,5** s

- $t_{exe} = n_{cicles} / f_{clock}$
  - n_{cicles} l'hem calculat a l'apartat (a) $= 11{,}0 \times 10^9$
  - $f_{clock} = 2$ GHz
- $t_{exe} = (11{,}0 \times 10^9) / (2 \times 10^9) =$ **5,5 s**

Pregunta 4. Rendiment i Consum

Suposem un programa que s'executa sobre un processador funcionant a una freqüència de 2GHz, el qual dissipa una potència de 40W. La següent taula mostra, per a cada tipus d'instrucció, el nombre d'instruccions executades i el CPI, referents a l'execució d'aquest programa:

Tipus d'instr.	Nombre. d'instr.	CPI
Memòria	$1{,}5 \times 10^9$	3
Salts	$1{,}0 \times 10^9$	4
Resta	$2{,}5 \times 10^9$	1

c) Calcula el consum d'energia del programa, en Joules

E = _____ J

- $E = P \times t_{exe}$

Pregunta 4. Rendiment i Consum

Suposem un programa que s'executa sobre un processador funcionant a una freqüència de 2GHz, el qual dissipa una potència de 40W. La següent taula mostra, per a cada tipus d'instrucció, el nombre d'instruccions executades i el CPI, referents a l'execució d'aquest programa:

Tipus d'instr.	Nombre. d'instr.	CPI
Memòria	$1{,}5 \times 10^9$	3
Salts	$1{,}0 \times 10^9$	4
Resta	$2{,}5 \times 10^9$	1

c) Calcula el consum d'energia del programa, en Joules

E = **220** J

- $E = P \times t_{exe}$
- $E = 40 \times 5{,}5$ **= 220 J**

Pregunta 4. Rendiment i Consum

Suposem un programa que s'executa sobre un processador funcionant a una freqüència de 2GHz, el qual dissipa una potència de 40W. La següent taula mostra, per a cada tipus d'instrucció, el nombre d'instruccions executades i el CPI, referents a l'execució d'aquest programa:

Tipus d'instr.	Nombre. d'instr.	CPI
Memòria	$1{,}5 \times 10^9$	3
Salts	$1{,}0 \times 10^9$	4
Resta	$2{,}5 \times 10^9$	1

d) Suposem una nova versió del programa en què reduïm el nombre de Salts a la meitat, però és a costa d'augmentar el seu CPI de 4 a 6. Quin guany de rendiment (speed-up) s'ha produït?

guany = ____

- Guany = $t_{exe\_original} / t_{exe\_millorat}$

  $= n_{cicles\_original} / n_{cicles\_millorat}$ (a igual f_{clock})

Pregunta 4. Rendiment i Consum

Suposem un programa que s'executa sobre un processador funcionant a una freqüència de 2GHz, el qual dissipa una potència de 40W. La següent taula mostra, per a cada tipus d'instrucció, el nombre d'instruccions executades i el CPI, referents a l'execució d'aquest programa:

Tipus d'instr.	Nombre. d'instr.	CPI
Memòria	$1{,}5 \times 10^9$	3
Salts	$1{,}0 \times 10^9$	4
Resta	$2{,}5 \times 10^9$	1

d) Suposem una nova versió del programa en què reduïm el nombre de Salts a la meitat, però és a costa d'augmentar el seu CPI de 4 a 6. Quin guany de rendiment (speed-up) s'ha produït?

guany =

- Guany = $t_{exe\_original} / t_{exe\_millorat}$

  $= n_{cicles\_original} / n_{cicles\_millorat}$ (a igual f_{clock})

- $n_{cicles\_millorat}$

  $= (1{,}5 \times 10^9) \times 3 + (\mathbf{0{,}5} \times 10^9) \times \mathbf{6} + (2{,}5 \times 10^9) \times 1$

  $= (1{,}5 \times 3 + 0{,}5 \times 6 + 2{,}5 \times 1) \times 10^9$

  $= 10{,}0 \times 10^9$

Pregunta 4. Rendiment i Consum

Suposem un programa que s'executa sobre un processador funcionant a una freqüència de 2GHz, el qual dissipa una potència de 40W. La següent taula mostra, per a cada tipus d'instrucció, el nombre d'instruccions executades i el CPI, referents a l'execució d'aquest programa:

Tipus d'instr.	Nombre. d'instr.	CPI
Memòria	$1{,}5 \times 10^9$	3
Salts	$1{,}0 \times 10^9$	4
Resta	$2{,}5 \times 10^9$	1

d) Suposem una nova versió del programa en què reduïm el nombre de Salts a la meitat, però és a costa d'augmentar el seu CPI de 4 a 6. Quin guany de rendiment (speed-up) s'ha produït?

guany = **1,1**

- Guany = $t_{exe\_original} / t_{exe\_millorat}$

  $= n_{cicles\_original} / n_{cicles\_millorat}$ (a igual f_{clock})

- $n_{cicles\_millorat}$

  $= (1{,}5 \times 10^9) \times 3 + (\mathbf{0{,}5} \times 10^9) \times \mathbf{6} + (2{,}5 \times 10^9) \times 1$

  $= (1{,}5 \times 3 + 0{,}5 \times 6 + 2{,}5 \times 1) \times 10^9$

  $= 10{,}0 \times 10^9$

- Guany = $(11{,}0 \times 10^9) / (10{,}0 \times 10^9) = \mathbf{1{,}1}$

Pregunta 5. Declaracions

Donada la següent declaració de variables globals d'un programa escrit en llenguatge C:

```
char a[5]    = "DADA";
char *b      = &a[2];
long long c = -4;
char d[2]    = {14,16};
unsigned int e[100];
```

a) Tradueix-la al llenguatge assemblador del MIPS

```
.data
```

Pregunta 5. Declaracions

Donada la següent declaració de variables globals d'un programa escrit en llenguatge C:

```
char a[5]    = "DADA";
char *b      = &a[2];
long long c = -4;
char d[2]    = {14,16};
unsigned int e[100];
```

a) Tradueix-la al llenguatge assemblador del MIPS

```
        .data
a:      .asciiz "DADA"
```

Pregunta 5. Declaracions

Donada la següent declaració de variables globals d'un programa escrit en llenguatge C:

```
char a[5]    = "DADA";
char *b      = &a[2];
long long c = -4;
char d[2]    = {14,16};
unsigned int e[100];
```

a) Tradueix-la al llenguatge assemblador del MIPS

```
        .data
a:      .asciiz "DADA"
b:      .word a+2
```

Pregunta 5. Declaracions

Donada la següent declaració de variables globals d'un programa escrit en llenguatge C:

```
char a[5]    = "DADA";
char *b      = &a[2];
long long c = -4;
char d[2]    = {14,16};
unsigned int e[100];
```

a) Tradueix-la al llenguatge assemblador del MIPS

```
        .data
a:      .asciiz "DADA"
b:      .word a+2
c:      .dword -4
```

Pregunta 5. Declaracions

Donada la següent declaració de variables globals d'un programa escrit en llenguatge C:

```
char a[5]    = "DADA";
char *b      = &a[2];
long long c = -4;
char d[2]    = {14,16};
unsigned int e[100];
```

a) Tradueix-la al llenguatge assemblador del MIPS

```
        .data
a:      .asciiz "DADA"
b:      .word a+2
c:      .dword -4
d:      .byte 14, 16
```

Pregunta 5. Declaracions

Donada la següent declaració de variables globals d'un programa escrit en llenguatge C:

```
char a[5]    = "DADA";
char *b      = &a[2];
long long c = -4;
char d[2]    = {14,16};
unsigned int e[100];
```

a) Tradueix-la al llenguatge assemblador del MIPS

```
        .data
a:      .asciiz "DADA"
b:      .word a+2
c:      .dword -4
d:      .byte 14, 16
        .align 2
e:      .space 400
```

Pregunta 5. Declaracions

```
a:      .asciiz "DADA"
```

b) Completa la següent taula amb el contingut de les 48 posicions de memòria representades, en hexadecimal (sense el prefix "0x"). Les variables globals s'emmagatzemen a partir de l'adreça 0x10010000. Recorda que el codi ASCII de la 'A' és el 0x41, i que les variables globals no inicialitzades valen 0x00. Les posicions de memòria no ocupades es deixen en blanc.

@Memòria	Dada
0x10010000	44
0x10010001	41
0x10010002	44
0x10010003	41
0x10010004	00
0x10010005	
0x10010006	
0x10010007	
0x10010008	
0x10010009	
0x1001000A	
0x1001000B	
0x1001000C	
0x1001000D	
0x1001000E	
0x1001000F	

@Memòria	Dada
0x10010010	
0x10010011	
0x10010012	
0x10010013	
0x10010014	
0x10010015	
0x10010016	
0x10010017	
0x10010018	
0x10010019	
0x1001001A	
0x1001001B	
0x1001001C	
0x1001001D	
0x1001001E	
0x1001001F	

@Memòria	Dada
0x10010020	
0x10010021	
0x10010022	
0x10010023	
0x10010024	
0x10010025	
0x10010026	
0x10010027	
0x10010028	
0x10010029	
0x1001002A	
0x1001002B	
0x1001002C	
0x1001002D	
0x1001002E	
0x1001002F	

Pregunta 5. Declaracions

```
a:      .asciiz "DADA"
b:      .word a+2
```

b) Completa la següent taula amb el contingut de les 48 posicions de memòria representades, en hexadecimal (sense el prefix "0x"). Les variables globals s'emmagatzemen a partir de l'adreça 0x10010000. Recorda que el codi ASCII de la 'A' és el 0x41, i que les variables globals no inicialitzades valen 0x00. Les posicions de memòria no ocupades es deixen en blanc.

@Memòria	Dada
0x10010000	44
0x10010001	41
0x10010002	44
0x10010003	41
0x10010004	00
0x10010005	
0x10010006	
0x10010007	
→ 0x10010008	02
0x10010009	00
0x1001000A	01
0x1001000B	10
0x1001000C	
0x1001000D	
0x1001000E	
0x1001000F	

@Memòria	Dada
0x10010010	
0x10010011	
0x10010012	
0x10010013	
0x10010014	
0x10010015	
0x10010016	
0x10010017	
0x10010018	
0x10010019	
0x1001001A	
0x1001001B	
0x1001001C	
0x1001001D	
0x1001001E	
0x1001001F	

@Memòria	Dada
0x10010020	
0x10010021	
0x10010022	
0x10010023	
0x10010024	
0x10010025	
0x10010026	
0x10010027	
0x10010028	
0x10010029	
0x1001002A	
0x1001002B	
0x1001002C	
0x1001002D	
0x1001002E	
0x1001002F	

Pregunta 5. Declaracions

```
a:      .asciiz "DADA"
b:      .word a+2
c:      .dword -4
```

b) Completa la següent taula amb el contingut de les 48 posicions de memòria representades, en hexadecimal (sense el prefix "0x"). Les variables globals s'emmagatzemen a partir de l'adreça 0x10010000. Recorda que el codi ASCII de la 'A' és el 0x41, i que les variables globals no inicialitzades valen 0x00. Les posicions de memòria no ocupades es deixen en blanc.

@Memòria	Dada
0x10010000	44
0x10010001	41
0x10010002	44
0x10010003	41
0x10010004	00
0x10010005	
0x10010006	
0x10010007	
0x10010008	02
0x10010009	00
0x1001000A	01
0x1001000B	10
0x1001000C	
0x1001000D	
0x1001000E	
0x1001000F	

→

@Memòria	Dada
0x10010010	FC
0x10010011	FF
0x10010012	FF
0x10010013	FF
0x10010014	FF
0x10010015	FF
0x10010016	FF
0x10010017	FF
0x10010018	
0x10010019	
0x1001001A	
0x1001001B	
0x1001001C	
0x1001001D	
0x1001001E	
0x1001001F	

@Memòria	Dada
0x10010020	
0x10010021	
0x10010022	
0x10010023	
0x10010024	
0x10010025	
0x10010026	
0x10010027	
0x10010028	
0x10010029	
0x1001002A	
0x1001002B	
0x1001002C	
0x1001002D	
0x1001002E	
0x1001002F	

Pregunta 5. Declaracions

```
a:      .asciiz "DADA"
b:      .word a+2
c:      .dword -4
d:      .byte 14, 16
```

b) Completa la següent taula amb el contingut de les 48 posicions de memòria representades, en hexadecimal (sense el prefix "0x"). Les variables globals s'emmagatzemen a partir de l'adreça 0x10010000. Recorda que el codi ASCII de la 'A' és el 0x41, i que les variables globals no inicialitzades valen 0x00. Les posicions de memòria no ocupades es deixen en blanc.

@Memòria	Dada
0x10010000	44
0x10010001	41
0x10010002	44
0x10010003	41
0x10010004	00
0x10010005	
0x10010006	
0x10010007	
0x10010008	02
0x10010009	00
0x1001000A	01
0x1001000B	10
0x1001000C	
0x1001000D	
0x1001000E	
0x1001000F	

@Memòria	Dada
0x10010010	FC
0x10010011	FF
0x10010012	FF
0x10010013	FF
0x10010014	FF
0x10010015	FF
0x10010016	FF
0x10010017	FF
0x10010018	0E
0x10010019	10
0x1001001A	
0x1001001B	
0x1001001C	
0x1001001D	
0x1001001E	
0x1001001F	

@Memòria	Dada
0x10010020	
0x10010021	
0x10010022	
0x10010023	
0x10010024	
0x10010025	
0x10010026	
0x10010027	
0x10010028	
0x10010029	
0x1001002A	
0x1001002B	
0x1001002C	
0x1001002D	
0x1001002E	
0x1001002F	

Pregunta 5. Declaracions

```
a:      .asciiz "DADA"
b:      .word a+2
c:      .dword -4
d:      .byte 14, 16
        .align 2
e:      .space 400
```

b) Completa la següent taula amb el contingut de les 48 posicions de memòria representades, en hexadecimal (sense el prefix "0x"). Les variables globals s'emmagatzemen a partir de l'adreça 0x10010000. Recorda que el codi ASCII de la 'A' és el 0x41, i que les variables globals no inicialitzades valen 0x00. Les posicions de memòria no ocupades es deixen en blanc.

@Memòria	Dada
0x10010000	44
0x10010001	41
0x10010002	44
0x10010003	41
0x10010004	00
0x10010005	
0x10010006	
0x10010007	
0x10010008	02
0x10010009	00
0x1001000A	01
0x1001000B	10
0x1001000C	
0x1001000D	
0x1001000E	
0x1001000F	

@Memòria	Dada
0x10010010	FC
0x10010011	FF
0x10010012	FF
0x10010013	FF
0x10010014	FF
0x10010015	FF
0x10010016	FF
0x10010017	FF
0x10010018	0E
0x10010019	10
0x1001001A	
0x1001001B	
0x1001001C	00
0x1001001D	00
0x1001001E	00
0x1001001F	00

@Memòria	Dada
0x10010020	00
0x10010021	00
0x10010022	00
0x10010023	00
0x10010024	00
0x10010025	00
0x10010026	00
0x10010027	00
0x10010028	00
0x10010029	00
0x1001002A	00
0x1001002B	00
0x1001002C	00
0x1001002D	00
0x1001002E	00
0x1001002F	00

Pregunta 5. Anàlisi

c) Donat el següent codi en MIPS, indica quin és el valor final en hexadecimal del registre `$t1`:

```
la      $t1, d+1
lb      $t1, 0($t1)
lui     $t2, 0x1001
or      $t1, $t2, $t1
lhu     $t1, 0($t1)
```

```
la    $t1, d+1        # $t1 =  0x 10010019
```

@Memòria	Dada
0x10010010	FC
0x10010011	FF
0x10010012	FF
0x10010013	FF
0x10010014	FF
0x10010015	FF
0x10010016	FF
0x10010017	FF
d: 0x10010018	0E
0x10010019	10
0x1001001A	
0x1001001B	
0x1001001C	00
0x1001001D	00
0x1001001E	00
0x1001001F	00

Pregunta 5. Anàlisi

c) Donat el següent codi en MIPS, indica quin és el valor final en hexadecimal del registre `$t1`:

```
la      $t1, d+1
lb      $t1, 0($t1)
lui     $t2, 0x1001
or      $t1, $t2, $t1
lhu     $t1, 0($t1)
```

```
la    $t1, d+1
lb    $t1, 0($t1)     # $t1 =  0x 00 00 00 10
```

Extensió de signe (positiu)

	@Memòria	Dada
	0x10010010	FC
	0x10010011	FF
	0x10010012	FF
	0x10010013	FF
	0x10010014	FF
	0x10010015	FF
	0x10010016	FF
	0x10010017	FF
d:	0x10010018	0E
	0x10010019	10
	0x1001001A	
	0x1001001B	
	0x1001001C	00
	0x1001001D	00
	0x1001001E	00
	0x1001001F	00

Pregunta 5. Anàlisi

c) Donat el següent codi en MIPS, indica quin és el valor final en hexadecimal del registre `$t1`:

```
la      $t1, d+1
lb      $t1, 0($t1)
lui     $t2, 0x1001
or      $t1, $t2, $t1
lhu     $t1, 0($t1)
```

```
la   $t1, d+1
lb   $t1, 0($t1)     # $t1 =  0x 00 00 00 10
lui  $t2, 0x1001     # $t2 =  0x 10 01 00 00
```

Zeros a la part baixa

d:

@Memòria	Dada
0x10010010	FC
0x10010011	FF
0x10010012	FF
0x10010013	FF
0x10010014	FF
0x10010015	FF
0x10010016	FF
0x10010017	FF
0x10010018	0E
0x10010019	10
0x1001001A	
0x1001001B	
0x1001001C	00
0x1001001D	00
0x1001001E	00
0x1001001F	00

Pregunta 5. Anàlisi

c) Donat el següent codi en MIPS, indica quin és el valor final en hexadecimal del registre `$t1`:

```
la      $t1, d+1
lb      $t1, 0($t1)
lui     $t2, 0x1001
or      $t1, $t2, $t1
lhu     $t1, 0($t1)
```

```
la   $t1, d+1
lb   $t1, 0($t1)     # $t1 =  0x 00 00 00 10
lui  $t2, 0x1001     # $t2 =  0x 10 01 00 00
or   $t1, $t2, $t1   # $t1 =  0x 10 01 00 10
```

d:

@Memòria	Dada
0x10010010	FC
0x10010011	FF
0x10010012	FF
0x10010013	FF
0x10010014	FF
0x10010015	FF
0x10010016	FF
0x10010017	FF
0x10010018	0E
0x10010019	10
0x1001001A	
0x1001001B	
0x1001001C	00
0x1001001D	00
0x1001001E	00
0x1001001F	00

Pregunta 5. Anàlisi

c) Donat el següent codi en MIPS, indica quin és el valor final en hexadecimal del registre `$t1`:

```
la      $t1, d+1
lb      $t1, 0($t1)
lui     $t2, 0x1001
or      $t1, $t2, $t1
lhu     $t1, 0($t1)
```

```
la   $t1, d+1
lb   $t1, 0($t1)     # $t1 =  0x 00 00 00 10
lui  $t2, 0x1001     # $t2 =  0x 10 01 00 00
or   $t1, $t2, $t1   # $t1 =  0x 10 01 00 10
lhu  $t1, 0($t1)
```

d:

@Memòria	Dada
0x10010010	FC
0x10010011	FF
0x10010012	FF
0x10010013	FF
0x10010014	FF
0x10010015	FF
0x10010016	FF
0x10010017	FF
0x10010018	0E
0x10010019	10
0x1001001A	
0x1001001B	
0x1001001C	00
0x1001001D	00
0x1001001E	00
0x1001001F	00

Pregunta 5. Anàlisi

c) Donat el següent codi en MIPS, indica quin és el valor final en hexadecimal del registre `$t1`:

```
la      $t1, d+1
lb      $t1, 0($t1)
lui     $t2, 0x1001
or      $t1, $t2, $t1
lhu     $t1, 0($t1)
```

```
la    $t1, d+1
lb    $t1, 0($t1)     # $t1 =  0x 00 00 00 10
lui   $t2, 0x1001     # $t2 =  0x 10 01 00 00
or    $t1, $t2, $t1   # $t1 =  0x 10 01 00 10
lhu   $t1, 0($t1)     # $t1 =  0x 00 00 FF FC
```

Unsigned: extensió de zeros

d:

@Memòria	Dada
0x10010010	FC
0x10010011	FF
0x10010012	FF
0x10010013	FF
0x10010014	FF
0x10010015	FF
0x10010016	FF
0x10010017	FF
0x10010018	0E
0x10010019	10
0x1001001A	
0x1001001B	
0x1001001C	00
0x1001001D	00
0x1001001E	00
0x1001001F	00

`$t1 =` **0x0000FFFC**

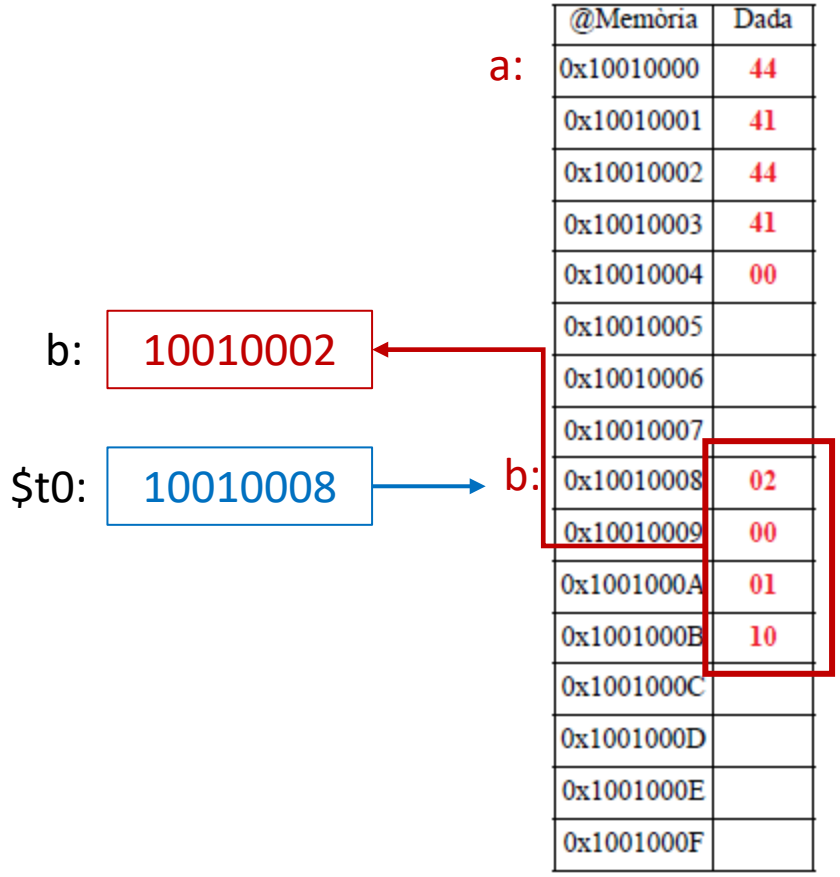
Pregunta 5. Anàlisi

d) Tradueix a llenguatge assemblador del MIPS la següent sentència en C:

```
*b = *(b-2);
```

- Carreguem la variable b al registre $t1

```
la    $t0, b
lw    $t1, 0($t0)
```



Pregunta 5. Anàlisi

d) Tradueix a llenguatge assemblador del MIPS la següent sentència en C:

```
*b = *(b-2);
```

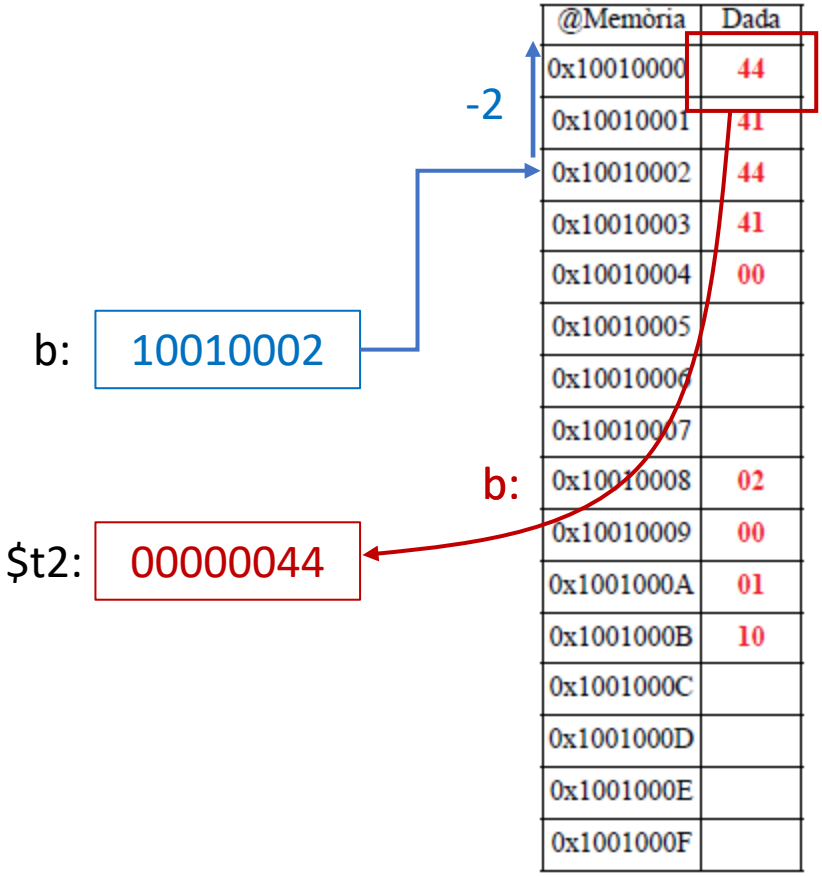
- Carreguem la variable b al registre $t1

```
la    $t0, b
lw    $t1, 0($t0)
```

- Carreguem la dada apuntada per b-2

(b està declarat *punter a char)*

```
lb    $t2, -2($t1)
```



Pregunta 5. Anàlisi

d) Tradueix a llenguatge assemblador del MIPS la següent sentència en C:

```
*b = *(b-2);
```

- Carreguem la variable b al registre $t1

```
la    $t0, b
lw    $t1, 0($t0)
```

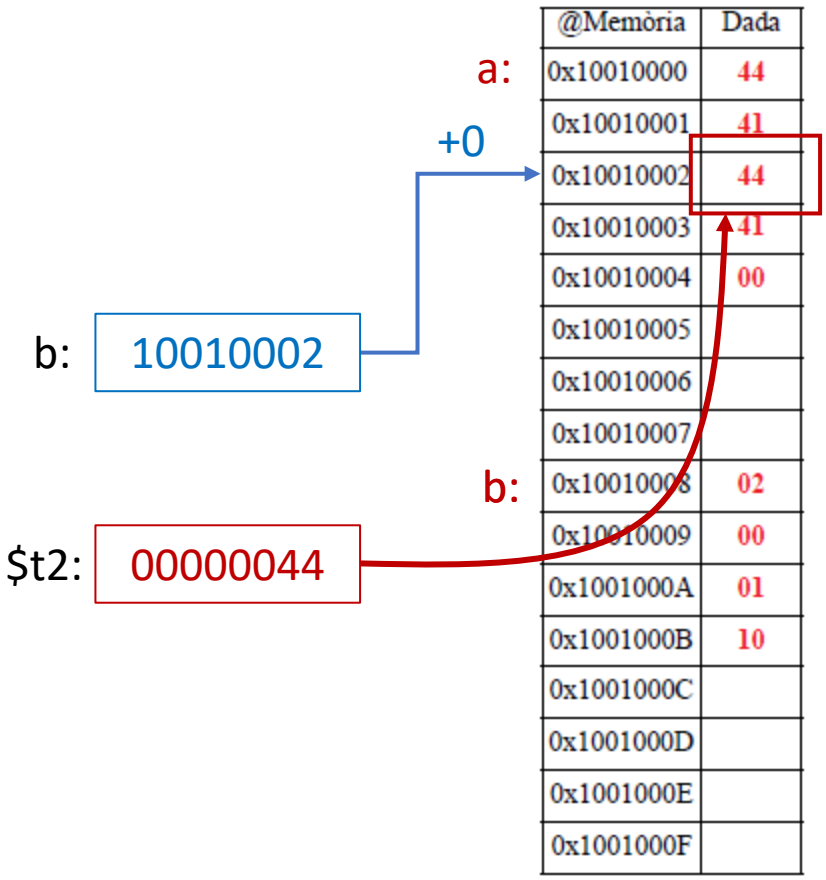
- Carreguem la dada apuntada per b-2

(b està declarat *punter a char)*

```
lb    $t2, -2($t1)
```

- La guardem en memòria a on apunti b

```
sb    $t2, 0($t1)
```



@Memòria	Dada
0x10010000	44
0x10010001	41
0x10010002	44
0x10010003	41
0x10010004	00
0x10010005	
0x10010006	
0x10010007	
0x10010008	02
0x10010009	00
0x1001000A	01
0x1001000B	10
0x1001000C	
0x1001000D	
0x1001000E	
0x1001000F	

```
la    $t0, b
lw    $t1, 0($t0)
lb    $t2, -2($t1)
sb    $t2, 0($t1)
```

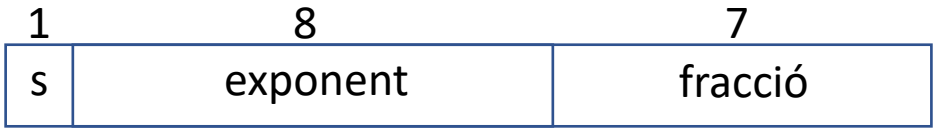
Pregunta 6. Coma flotant

Suposem que definim un format de coma flotant de 16 bits, similar a l'estàndard de simple precisió, excepte que té 7 bits de fracció en comptes de 23. La resta de camps es codifiquen igual.

a) Codifica el número x = -27,16 en el nou format de 16 bits aplicant l'arrodoniment al més pròxim, i expressa el resultat en hexadecimal:

x = 0x

- Desglossament dels bits:



- Part entera: 27 = 11011 (ja tenim 5 dels 8 bits de mantissa)

Pregunta 6. Coma flotant

Suposem que definim un format de coma flotant de 16 bits, similar a l'estàndard de simple precisió, excepte que té 7 bits de fracció en comptes de 23. La resta de camps es codifiquen igual.

a) Codifica el número x = -27,16 en el nou format de 16 bits aplicant l'arrodoniment al més pròxim, i expressa el resultat en hexadecimal:

x = | 0x |

- Desglossament dels bits:

1	8	7
s	exponent	fracció

- Part entera: 27 = 11011 (ja tenim 5 dels 8 bits de mantissa)
- Fracció: 0,16 =

  0,16 × 2 = **0**,32 0,32 × 2 = **0**,64 0,64 × 2 = **1**,28 (ja tenim 5+3 = 8 bits de mantissa)

Pregunta 6. Coma flotant

Suposem que definim un format de coma flotant de 16 bits, similar a l'estàndard de simple precisió, excepte que té 7 bits de fracció en comptes de 23. La resta de camps es codifiquen igual.

a) Codifica el número x = -27,16 en el nou format de 16 bits aplicant l'arrodoniment al més pròxim, i expressa el resultat en hexadecimal:

x = 0x

- Desglossament dels bits:

1	8	7
s	exponent	fracció

- Part entera: 27 = 11011 (ja tenim 5 dels 8 bits de mantissa)
- Fracció: 0,16 =

  0,16 × 2 = **0**,32 0,32 × 2 = **0**,64 0,64 × 2 = **1**,28 (ja tenim 5+3 = 8 bits de mantissa)

  Afegim alguns bits extra per a l'arrodoniment:

  0,28 × 2 = **0**,56 0,56 × 2 = **1**,12 (bits **01** → suficients per arrodonir avall)

- Número en binari: -11011,001 **01**

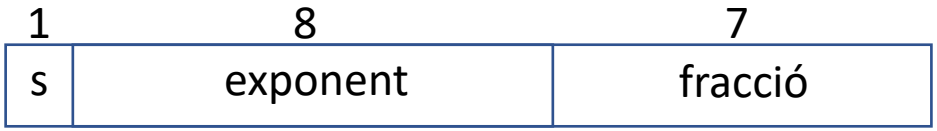
Pregunta 6. Coma flotant

Suposem que definim un format de coma flotant de 16 bits, similar a l'estàndard de simple precisió, excepte que té 7 bits de fracció en comptes de 23. La resta de camps es codifiquen igual.

a) Codifica el número x = -27,16 en el nou format de 16 bits aplicant l'arrodoniment al més pròxim, i expressa el resultat en hexadecimal:

x = 0x

- Desglossament dels bits:



- Número en binari: -11011,001 **01**
- Normalitzem: **-1**,1011001 **01** × 2^4

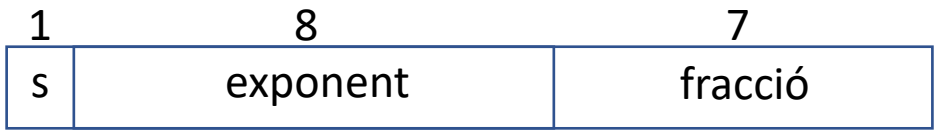
Pregunta 6. Coma flotant

Suposem que definim un format de coma flotant de 16 bits, similar a l'estàndard de simple precisió, excepte que té 7 bits de fracció en comptes de 23. La resta de camps es codifiquen igual.

a) Codifica el número x = -27,16 en el nou format de 16 bits aplicant l'arrodoniment al més pròxim, i expressa el resultat en hexadecimal:

x = 0x

- Desglossament dels bits:



- Número en binari: -11011,001 **01**
- Normalitzem: **-1**,1011001 **01** × 2^4
- Arrodonim (truncant): **-1**,1011001 × 2^4

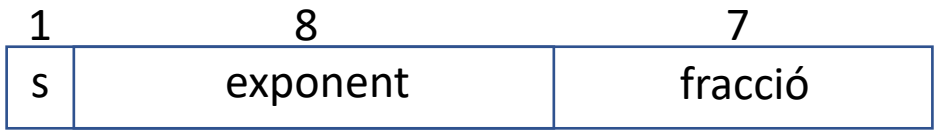
Pregunta 6. Coma flotant

Suposem que definim un format de coma flotant de 16 bits, similar a l'estàndard de simple precisió, excepte que té 7 bits de fracció en comptes de 23. La resta de camps es codifiquen igual.

a) Codifica el número x = -27,16 en el nou format de 16 bits aplicant l'arrodoniment al més pròxim, i expressa el resultat en hexadecimal:

x = 0x

- Desglossament dels bits:



- Número en binari: -11011,001 **01**
- Normalitzem: **-1**,1011001 **01** × 2^4
- Arrodonim (truncant): **-1**,1011001 × 2^4
- Exponent (en excés a 127): 4+127 = 131 = 10000011

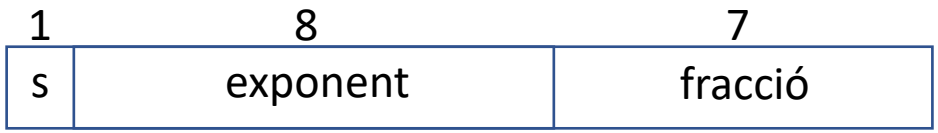
Pregunta 6. Coma flotant

Suposem que definim un format de coma flotant de 16 bits, similar a l'estàndard de simple precisió, excepte que té 7 bits de fracció en comptes de 23. La resta de camps es codifiquen igual.

a) Codifica el número x = -27,16 en el nou format de 16 bits aplicant l'arrodoniment al més pròxim, i expressa el resultat en hexadecimal:

x = 0x

- Desglossament dels bits:



- Número en binari: -11011,001 **01**
- Normalitzem: **-1**,1011001 **01** × 2^4
- Arrodonim (truncant): **-1**,1011001 × 2^4
- Exponent (en excés a 127): 4+127 = 131 = 10000011
- Signe: 1
- Ajuntem signe, exponent i fracció codificats, eliminant el bit ocult

  1 10000011 1011001

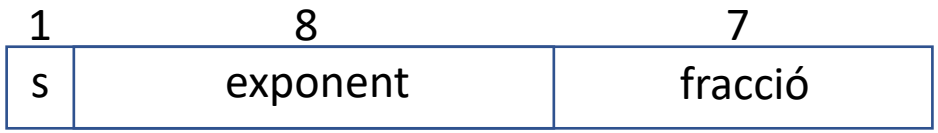
Pregunta 6. Coma flotant

Suposem que definim un format de coma flotant de 16 bits, similar a l'estàndard de simple precisió, excepte que té 7 bits de fracció en comptes de 23. La resta de camps es codifiquen igual.

a) Codifica el número x = -27,16 en el nou format de 16 bits aplicant l'arrodoniment al més pròxim, i expressa el resultat en hexadecimal:

x = 0x **0xC1D9**

- Desglossament dels bits:



- Número en binari: -11011,001 **01**
- Normalitzem: **-1**,1011001 **01** × 2^4
- Arrodonim (truncant): **-1**,1011001 × 2^4
- Exponent (en excés a 127): 4+127 = 131 = 10000011
- Signe: 1
- Ajuntem signe, exponent i fracció codificats, eliminant el bit ocult

  1 10000011 1011001
- Hexadecimal: 0xC1D9

Pregunta 6. Coma flotant

Suposem que definim un format de coma flotant de 16 bits, similar a l'estàndard de simple precisió, excepte que té 7 bits de fracció en comptes de 23. La resta de camps es codifiquen igual.

b) Calcula l'error de precisió comès en l'anterior apartat, expressant-lo en base 10, amb 2 dígits significatius (per exemple: error = 0,000XX, o bé: 0,0XX, etc.):

error = ☐

- Valor exacte inicial: $v = -27{,}16_{10}$
- Resultat arrodonit: $v_0 = -1{,}1011001_2 \times 2^4$

  $= -11011{,}001_2 = -27{,}125_{10}$ (en decimal)

Pregunta 6. Coma flotant

Suposem que definim un format de coma flotant de 16 bits, similar a l'estàndard de simple precisió, excepte que té 7 bits de fracció en comptes de 23. La resta de camps es codifiquen igual.

b) Calcula l'error de precisió comès en l'anterior apartat, expressant-lo en base 10, amb 2 dígits significatius (per exemple: error = 0,000XX, o bé: 0,0XX, etc.):

error = **0,035**

- Valor exacte inicial: $v = -27{,}16_{10}$
- Resultat arrodonit: $v_0 = -1{,}1011001_2 \times 2^4$

  $= -11011{,}001_2 = -27{,}125_{10}$ (en decimal)
- Error de precisió: $\varepsilon = |v - v_0|$

  $= |-27{,}16 - (-27{,}125)|$

  $= \mathbf{0{,}035}$

Pregunta 7. Coma flotant

Les següents afirmacions fan referència al format de simple precisió IEEE-754 (32 bits). Posa una X per a cada una d'elles (a la columna V si és Verdadera o a la columna F si és Falsa). Cada resposta correcta suma 0,1 punts; les respostes no contestades no es tenen en compte; cada resposta incorrecta resta 0,1 punts; i la puntuació total mínima és 0.

	Afirmació	V	F
1.-	Es produeix *underflow* quan els bits fraccionaris de la mantissa són tots zeros		X
2.-	Es produeix *overflow* quan l'exponent del resultat d'una operació és major que +126		
3.-	La codificació 0x7FFFFFFF representa el major número positiu no-nul		
4.-	La codificació 0x00800000 representa un número normalitzat		
5.-	La codificació 0x00000001 representa un número denormal		

1. FALS. Contraexemple: $1{,}000....0 \times 2^3$

   L'underflow es produeix quan la magnitud d'un resultat és menor que el mínim dels valors normalitzats (que és $1{,}000...00 \times 2^{Emin}$), i s'identifica fàcilment perquè si el normalitzem, el seu exponent és $E < -126 = E_{min}$

Pregunta 7. Coma flotant

Les següents afirmacions fan referència al format de simple precisió IEEE-754 (32 bits). Posa una X per a cada una d'elles (a la columna V si és Verdadera o a la columna F si és Falsa). Cada resposta correcta suma 0,1 punts; les respostes no contestades no es tenen en compte; cada resposta incorrecta resta 0,1 punts; i la puntuació total mínima és 0.

	Afirmació	V	F
1.-	Es produeix *underflow* quan els bits fraccionaris de la mantissa són tots zeros		X
2.-	Es produeix *overflow* quan l'exponent del resultat d'una operació és major que +126		X
3.-	La codificació 0x7FFFFFFF representa el major número positiu no-nul		
4.-	La codificació 0x00800000 representa un número normalitzat		
5.-	La codificació 0x00000001 representa un número denormal		

2. FALS. Contraexemple: $1{,}000....0 \times 2^{127}$ és representable

   El major exponent és E_{max} i es codifica en excés com 11111110_2, que representa l'enter 254 – 127 = 127

Pregunta 7. Coma flotant

Les següents afirmacions fan referència al format de simple precisió IEEE-754 (32 bits). Posa una X per a cada una d'elles (a la columna V si és Verdadera o a la columna F si és Falsa). Cada resposta correcta suma 0,1 punts; les respostes no contestades no es tenen en compte; cada resposta incorrecta resta 0,1 punts; i la puntuació total mínima és 0.

	Afirmació	V	F
1.-	Es produeix *underflow* quan els bits fraccionaris de la mantissa són tots zeros		X
2.-	Es produeix *overflow* quan l'exponent del resultat d'una operació és major que +126		X
3.-	La codificació 0x7FFFFFFF representa el major número positiu no-nul		X
4.-	La codificació 0x00800000 representa un número normalitzat		
5.-	La codificació 0x00000001 representa un número denormal		

3. FALS. Aquest valor conté l'exponent i la fracció amb tots els bits a 1, i no representa un "número", sinó el valor Infinit.

   El "major número positiu no-nul" és $1,1111...1_2 \times 2^{127}$ i es codifica com: 0-11111110-11111111111111111111111 = 0x7F7FFFFF

Pregunta 7. Coma flotant

Les següents afirmacions fan referència al format de simple precisió IEEE-754 (32 bits). Posa una X per a cada una d'elles (a la columna V si és Verdadera o a la columna F si és Falsa). Cada resposta correcta suma 0,1 punts; les respostes no contestades no es tenen en compte; cada resposta incorrecta resta 0,1 punts; i la puntuació total mínima és 0.

	Afirmació	V	F
1.-	Es produeix *underflow* quan els bits fraccionaris de la mantissa són tots zeros		X
2.-	Es produeix *overflow* quan l'exponent del resultat d'una operació és major que +126		X
3.-	La codificació 0x7FFFFFFF representa el major número positiu no-nul		X
4.-	La codificació 0x00800000 representa un número normalitzat	X	
5.-	La codificació 0x00000001 representa un número denormal		

4. VERITAT. 0x00800000 = 0 00000001 00000000000000000000000
   El seu exponent és 00000001, el qual correspon a números normalitzats

   En concret, és el número $1{,}0 \times 2^{-126}$

Pregunta 7. Coma flotant

Les següents afirmacions fan referència al format de simple precisió IEEE-754 (32 bits). Posa una X per a cada una d'elles (a la columna V si és Verdadera o a la columna F si és Falsa). Cada resposta correcta suma 0,1 punts; les respostes no contestades no es tenen en compte; cada resposta incorrecta resta 0,1 punts; i la puntuació total mínima és 0.

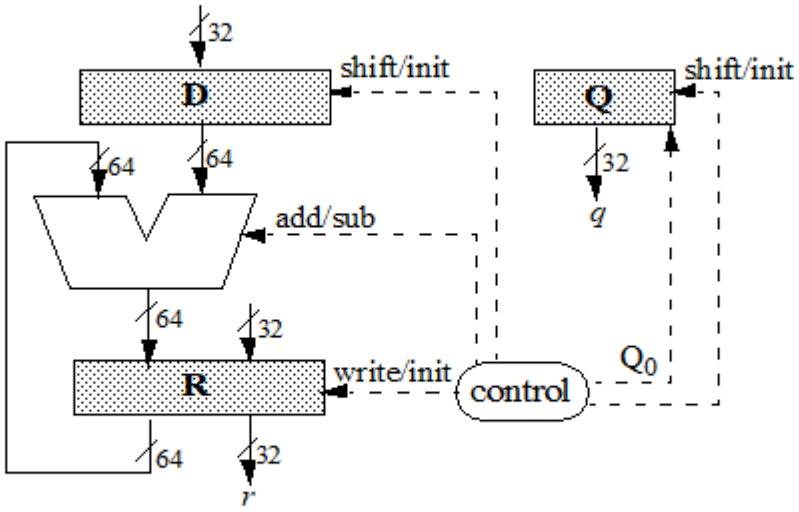
	Afirmació	V	F
1.-	Es produeix *underflow* quan els bits fraccionaris de la mantissa són tots zeros		X
2.-	Es produeix *overflow* quan l'exponent del resultat d'una operació és major que +126		X
3.-	La codificació 0x7FFFFFFF representa el major número positiu no-nul		X
4.-	La codificació 0x00800000 representa un número normalitzat	X	
5.-	La codificació 0x00000001 representa un número denormal	X	

5. VERITAT. 0x00000001 = 0 00000000 00000000000000000001
   El seu exponent és 00000000, el qual correspon a números denormals

   En concret, és el número $0{,}00000000000000000000001 \times 2^{-126}$
   $= 1{,}0 \times 2^{-126-23} = 1{,}0 \times 2^{-149}$

Pregunta 8. Divisió d'enters

Sigui el següent circuit seqüencial per a la divisió de números naturals de 32 bits, que calcula el quocient $q = x/y$, i el residu $r = x\%y$ (els senyals d'entrada x, i y s'hi han omès expressament):

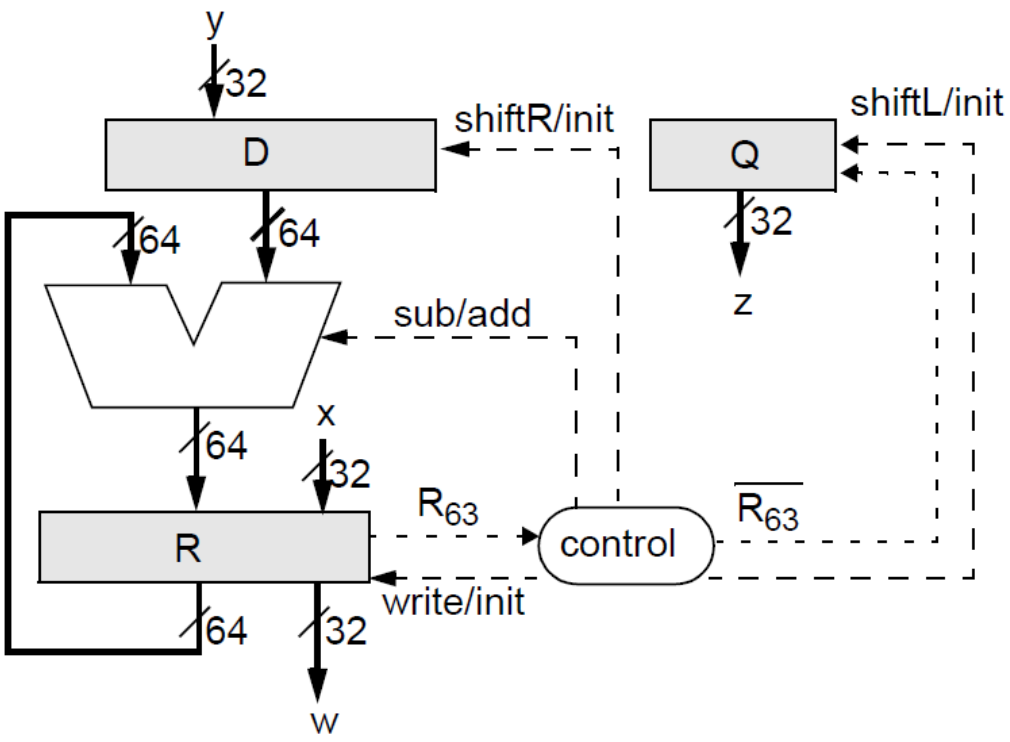


Completa el següent algorisme en pseudocodi, que expressa la seqüència d'operacions que realitza l'anterior circuit divisor:

Pregunta 8. Divisió d'enters

- Solució (la teniu als apunts del Tema 5, secció 3.3, pàgina 11) !!

Divisor seqüencial de naturals



Pseudocodi

```
// Inicialització
R63:32 = 0; R31:0 = x;
D63:32 = y; D31:0 = 0;
Q  = 0;
for (i=1; i<=32; i++) {
     D = D >> 1;
     R = R - D;
     if (R63 == 0)
          Q = (Q << 1) | 1;
     else {
          R = R + D;
          Q = Q << 1;
     }
}
```