UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Departament d'Arquitectura de Computadors

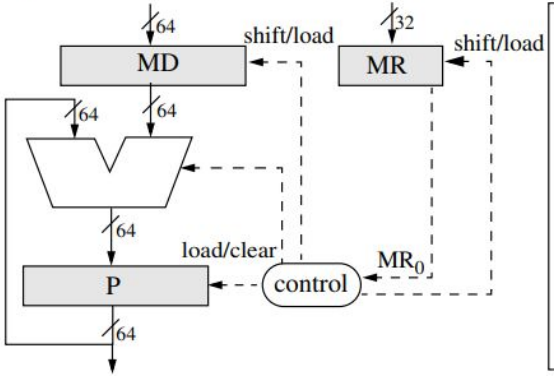
Repàs final

Estructura de Computadors (EC)
2023 - 2024 Q2
Adrià Armejach (adria.armejach@upc.edu)

Aritmètica d'enters

18-19Q2 - Exercici 6

A continuació es mostra la unitat de procés i l'algorisme que usa la unitat de control (UC) del multiplicador seqüencial de nombres naturals de 32 bits estudiat a classe, que calcula Z=X*Y:




```
MR = Y;
MD63:32= 0; MD31:0= X;
P63:32 = 0; P31:0= 0;
for (i=1; i<=32; i++)
{
        if (MR0 == 1)
                { P = P + MD; }
        MD = MD << 1;
        MR = MR >> 1;
}
Z = P;
```

Per implementar la UC cal considerar que té un senyal d'entrada MR_0 que correspon al bit de menys pes del registre `MR` i 6 sortides (`load_MD`, `load_MR`, `load_P`, `shift_MD`, `shift_MR`, `clear_P`). Els senyals *load* carreguen síncronament al registre corresponent el que tinguin a la seva entrada. El senyal `shift_MD` permet fer el desplaçament d'un bit a l'esquerra del registre `MD`, el senyal `shift_MR` permet fer el desplaçament d'un bit a la dreta del registre `MR`. El senyal `clear_P` posa a 0 el registre `P`. Tots aquests senyals són actius amb valor 1. Amb valor 0 són inactius. De les dues tasques que pot fer un registre, si les dues estan actives, sols executarà la càrrega per ser més prioritària. Suposem que l'ALU només realiza sumes en tot moment i, per tant no cal especificar la funció. També considerem que el control de les iteracions ja està implementat internament dins la unitat de control.

18-19Q2 - Exercici 6

a) Indica els valors que falten a la següent taula dels senyals de sortida durant la inicialització i durant cada iteració del processament (valen el mateix a totes les iteracions). Els valors que falten poden ser 1, 0 o X (*don't care*, és a dir, que sigui indiferent el valor que prengui).

	load_MD	load_MR	load_P	shift_MD	shift_MR	clear_P
inici						
cada iteració			MR_0			

b) Volem fer una optimització de la UC de forma que el bucle de processament acabi quan el registre `MR` no tingui cap bit a 1. Indica els canvis que s'haurien de realitzar a l'algorisme donat per reflectir aquesta optimització.

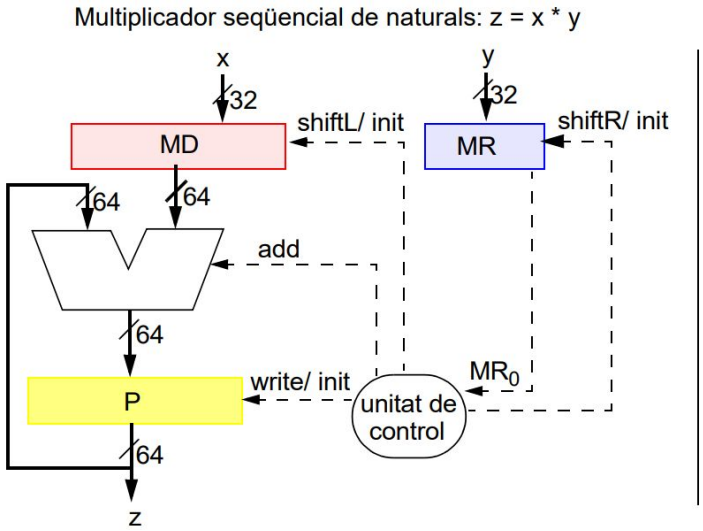
Multiplicació de naturals

- Números binaris (base 2)

```
   10                 1010      multiplicand
 × 13               × 1101      multiplicador
 ----               ------
                      1010  =    1010 × 1
                     0000   =   10100 × 0
                    1010    =  101000 × 1
                  + 1010    = 1010000 × 1
 = 130              10000010
```

- Productes parcials es redueixen a una decisió binària

Circuit multiplicador seqüencial



Pseudocodi

```
// Inicialització
MD0:31 = x; MD32:63 = 0;
P   = 0;
MR = y;

for (i=1; i<=32; i++)
{
      if (MR0 == 1)
            P = P + MD;
      MD = MD << 1;
      MR = MR >> 1;
}
z = P;
```

- Naturals de 32 bits amb resultat de 64 bits
  - Tarda 33 cicles a obtenir el producte …
  - … assumint que una suma de 64 bits tarda 1 cicle

Exemple de multiplicació: 1010 x 1101

iter.	P (Producte)								MD (Multiplicand)								MR (Multiplicador)			
inicial	0	0	0	0	0	0	0	0	0	0	0	0	**1**	**0**	**1**	**0**	**1**	**1**	**0**	**1**

Exemple de multiplicació: 1010 x 1101

iter.	P (Producte)								MD (Multiplicand)								MR (Multiplicador)			
inicial	0	0	0	0	0	0	0	0	0	0	0	0	**1**	**0**	**1**	**0**	**1**	**1**	**0**	**1**
1	0	0	0	**0**	**1**	**0**	**1**	**0**	0	0	0	**1**	**0**	**1**	**0**	0	0	**1**	**1**	**0**

Exemple de multiplicació: 1010 x 1101

iter.	P (Producte)								MD (Multiplicand)								MR (Multiplicador)			
inicial	0	0	0	0	0	0	0	0	0	0	0	0	**1**	**0**	**1**	**0**	**1**	**1**	**0**	**1**
1	0	0	0	**0**	**1**	**0**	**1**	**0**	0	0	0	**1**	**0**	**1**	**0**	0	0	**1**	**1**	**0**
2	0	0	**0**	**0**	**1**	**0**	**1**	**0**	0	0	**1**	**0**	**1**	**0**	0	0	0	0	**1**	**1**

Exemple de multiplicació: 1010 x 1101

iter.	P (Producte)								MD (Multiplicand)								MR (Multiplicador)			
inicial	0	0	0	0	0	0	0	0	0	0	0	0	**1**	**0**	**1**	**0**	**1**	**1**	**0**	**1**
1	0	0	0	**0**	**1**	**0**	**1**	**0**	0	0	0	**1**	**0**	**1**	**0**	0	0	**1**	**1**	**0**
2	0	0	**0**	**0**	**1**	**0**	**1**	**0**	0	0	**1**	**0**	**1**	**0**	0	0	0	0	**1**	**1**
3	0	**0**	**1**	**1**	**0**	**0**	**1**	**0**	0	**1**	**0**	**1**	**0**	0	0	0	0	0	0	**1**

Exemple de multiplicació: 1010 x 1101

iter.	P (Producte)								MD (Multiplicand)								MR (Multiplicador)			
inicial	0	0	0	0	0	0	0	0	0	0	0	0	**1**	**0**	**1**	**0**	**1**	**1**	**0**	**1**
1	0	0	0	**0**	**1**	**0**	**1**	**0**	0	0	0	**1**	**0**	**1**	**0**	0	0	**1**	**1**	**0**
2	0	0	**0**	**0**	**1**	**0**	**1**	**0**	0	0	**1**	**0**	**1**	**0**	0	0	0	0	**1**	**1**
3	0	**0**	**1**	**1**	**0**	**0**	**1**	**0**	0	**1**	**0**	**1**	**0**	0	0	0	0	0	0	**1**
4	**1**	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**1**	**0**	**1**	**0**	0	0	0	0	0	0	0	0

Coma flotant

18-19Q2 - Exercici 9

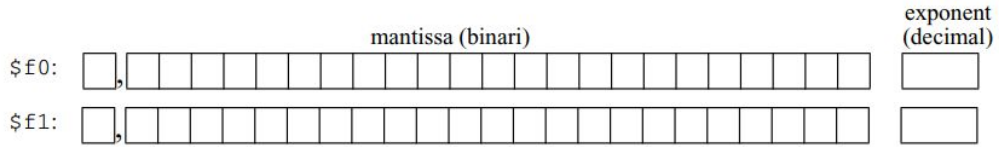
Donat el següent codi en C:

```
float res, i;
main() {
        res = 0.0;
        for (i = 1.0; i <= 10000.0; i = i + 1.0)
                res = res + i;
}
```

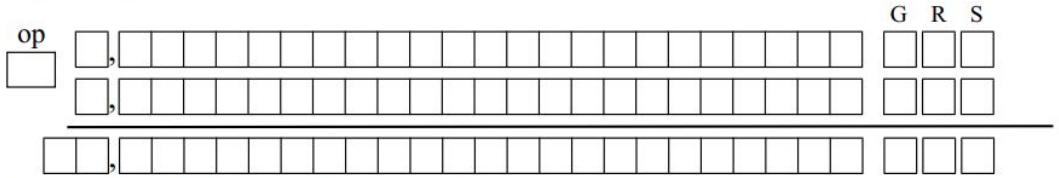
Abans de començar la darrera iteració del bucle (`i=10000.0`), ens trobem que el valor de `res` és **0x4C3EB530** i està guardat al registre `$f0`, i `i` val **0x461C4000**, que és la representació de 10000.0 en coma flotant, i està guardat al registre `$f1`. Volem executar la instrucció MIPS **add.s $f0, $f0, $f1** per obtenir el valor final de `res`. Suposant que el sumador té 1 bit de guarda, un d'arrodoniment i un de "sticky", i que arrodoneix al més pròxim (al parell en el cas equidistant), contesta les següents preguntes:

18-19Q2 - Exercici 9

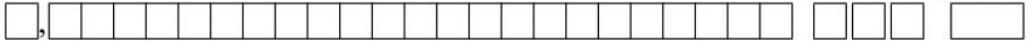
Quina és la mantissa (en binari) i l'exponent (en decimal) dels nombres que hi ha a `$f0` i `$f1`?



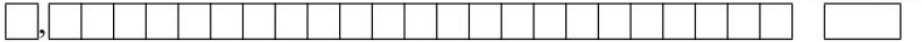
Omple les següents caselles mostrant l'operació op (+/-), les cadenes de bits a operar, i el resultat:



Resultat després de re-normalitzar (si cal):



Resultat després d'arrodonir:



Quin és el valor de `$f0` en hexadecimal després d'executar la instrucció ?

`$f0 =` `0x`

18-19Q2 - Exercici 9

Aquest programa suma els números de la sèrie 1.0, 2.0, 3.0, ..., 10000.0. La suma d'aquesta progressió aritmètica (per a N=10000.0) també es pot calcular amb la fórmula N*(N+1)/2, i el resultat exacte és 50 005 000.00 (**0x4C3EC102** en hexadecimal). Pots explicar breument per què no s'ha obtingut el valor correcte?

Si canviem el sentit del recorregut del bucle («`for (i = 10000.0; i > 0.0; i = i - 1.0)`»), el valor obtingut a `res` és **0x4C3EC4FC**. Pots explicar breument per quin motiu el resultat és diferent en canviar el sentit del recorregut?

Representació de nombres en coma flotant

- Representació binaria
  - xxx…x: dígits binaris de la mantissa
  - yyy…y: dígits binaris de l'exponent

$$v = \pm \underbrace{1,\overbrace{xxx \ldots x}^{\text{fracció(F)}}}_{\text{mantissa}} \times \underbrace{2}_{\text{base}}^{\underbrace{yyy \cdots y}_{\text{exponent(E)}}}$$

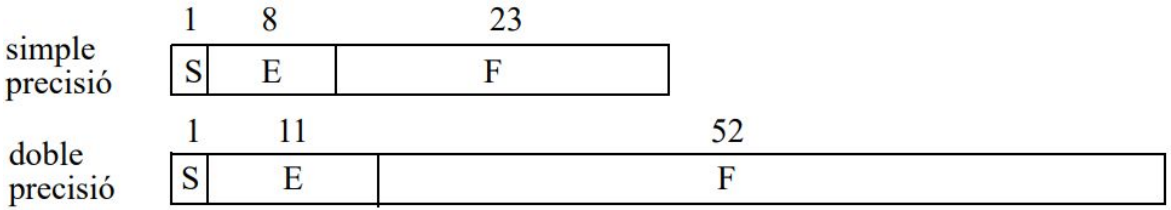
signe(S)

- S'emmagatzemen el signe (S), la fracció (F) i l'exponent (E)
- No s'emmagatzemen
  - El dígit enter de la mantissa, sempre és 1 (bit ocult)
  - La base, sempre és 2

$$v = (-1)^s \times (1 + 0, f) \times 2^e$$

L'estàndard IEEE-754

- Defineix el format dels nombres en coma flotant (1985)



- Signe (S)
  - 0: positiu 1: negatiu
- Fracció (F)
  - Part fraccionària de la mantissa. El bit ocult (part entera) no es representa.
- Exponent (E)
  - Codificat en excés 127 (simple precisió) o 1023 (doble precisió)

Suma/resta en coma flotant - simple precisió

- Sumar `z = x + y;` on `x = 0x3F40000D, y = 0xC0800004`

1. **Decodificar signe, exponent i mantissa**

```
x = 0 01111110 10000000000000000001101
E = 126 - 127 = -1
x = +1,10000000000000000001101 × 2^-1

y = 1 10000001 00000000000000000100
E = 129 - 127 = 2
y = -1,00000000000000000100 × 2^2
```

Suma/resta en coma flotant - simple precisió

- Sumar `z = x + y;` on `x = 0x3F40000D, y = 0xC0800004`

2. **Igualar exponents (al major)**

$x = +1,10000000000000000001101 \times 2^{-1}$
$y = -1,00000000000000000000100 \times 2^{2}$

$x = +0,00110000000000000000001101 \times 2^{2}$

Suma/resta en coma flotant - simple precisió

- Sumar `z = x + y;` on `x = 0x3F40000D, y = 0xC0800004`

3. **Sumar o restar les mantisses**

Signe x	Signe y	Operació	Signe del resultat
+	+	Suma	+
-	-	Suma	-
+ -	- +	Al de major magnitud se li resta el de menor magnitud	Signe del de major magnitud

Suma/resta en coma flotant - simple precisió

- Sumar `z = x + y;` on `x = 0x3F40000D, y = 0xC0800004`

3. **Sumar o restar les mantisses**

$x = +0,00110000000000000000001101 \times 2^2$
$y = -1,00000000000000000000100 \times 2^2$

Signe diferent: S'han de restar les mantisses
y té major magnitud: **y - x, signe del resultat negatiu**

$$
\begin{array}{rl}
 & |y| = 1,00000000000000000000100 \\
- & |x| = 0,00110000000000000000001101 \\
\hline
= & |z| = 0,11010000000000000000010011
\end{array}
$$

Suma/resta en coma flotant - simple precisió

- Sumar `z = x + y;` on `x = 0x3F40000D, y = 0xC0800004`

4. **Normalitzar la mantissa**

$|z| = 0,11010000000000000000010011 \times 2^{2}$

$|z| = 1,10100000000000000000010011 \times 2^{1}$

Si E < E min : underflow

Si E > E max : overflow

Suma/resta en coma flotant - simple precisió

- Sumar `z = x + y;` on `x = 0x3F40000D, y = 0xC0800004`

5. **Arrodonir la mantissa**

$|z| = 1,10100000000000000000100\,11 \times 2^{1}$

```
   1,10100000000000000000100
+  0,00000000000000000000001
  ---------------------------
   1,10100000000000000000101
```

$|z| = 1,10100000000000000000101 \times 2^{1}$

A vegades és necessari tornar a normalitzar i arrodonir la mantissa

Suma/resta en coma flotant - simple precisió

- Sumar `z = x + y;` on `x = 0x3F40000D, y = 0xC0800004`

6. **Codificar signe, exponent i fracció**

$z = -1,10100000000000000000101 \times 2^{1}$

$E = 1 + 127 = 128 = 10000000_2$

`z = 1 10000000 10100000000000000000101`

`z = 0xC0500005`

Memòria cache

18-19Q2 - Exercici 10

Un sistema disposa d'un processador de 32 bits (adreces i dades de 32 bits). La cache d'instruccions podem suposar que és ideal (sempre encerta). La cache de dades (MC) té 512 bytes i la següent organització:

- Correspondència associativa per conjunts, de grau 2 (2 blocs per conjunt)
- Blocs de 16 bytes
- Reemplaçament LRU
- Escriptura immediata sense assignació.

Un programa executa el següent bucle en C, en què les dades accedides són totes de tipus `int`:

```
for (i = 0; i < 3; i++)
        res = res + vec[i];
```

Assumint que `i` es guarda en un registre temporal, `res` és una variable global a l'adreça **0x10010000**, i `vec` és un vector global guardat en una adreça no consecutiva (hi ha altres variables globals al codi sencer), completa la seqüència de referències a dades de memòria segons s'indica a la següent taula. A la taula apareixen les adreces en hexadecimal i si són lectures o escriptures (L/E). Completa les columnes que falten indicant, per a cada referència: el número de conjunt de MC; si és encert (e) o fallada (f); i el nombre de bytes de Memòria Principal (MP) llegits i/o escrits. Podeu assumir que inicialment la MC està buida.

18-19Q2 - Exercici 10

L/E	adreça (hex)	núm. de conjunt	encert (e)/ fallada (f)	bytes de MP	
				llegits	escrits
L	0x100110F8				
L					
E					
L					
L					
E					
L					
L					
E					

Memòria virtual

18-19Q2 - Exercici 3

Disposem d'un computador que gestiona memòria virtual paginada amb pàgines de 64KB. El processador permet adreçar fins a 4GB de memòria virtual però només pot tenir 4MB de memòria física. El sistema operatiu limita a 4 el nombre de marcs de pàgina disponibles per cada procés i la seva política de reemplaçament és LRU.

a) Quants bytes ocuparà la taula de pàgines tenint en compte que a cada entrada hi ha un bit de presència (P) i un bit de pàgina modificada (D) a part del número de pàgina física (PPN)?

18-19Q2 - Exercici 3

Durant l'execució d'un cert programa la seva taula de pàgines és en el següent estat:

VPN (hex)	PPN (hex)	P	D
...			
1010	1A	1	1
...			
2800	3F	1	0
...			
57C0	25	1	1
...			
77FF	20	1	0
...			

Els darrers accessos que ha realitzat aquest programa han estat en el següent ordre de VPNs: 0x1010, 0x77FF, 0x2800 i 0x57C0.

b) Omple la següent taula per a cada referència a memòria que es realitza posteriorment a l'estat indicat anteriorment.

adr. lògica (hex)		VPN (hex)	fallada de pàgina? (SI/NO)	PPN (hex)	lectura de disc (SI/NO)	escriptura a disc (SI/NO)	VPN pàgina reemplaçada (hex)
E	0x1011C5F4						
L	0x10100008						
L	0x2800FFFC						
E	0x77FFA400						
E	0x1011A274						
L	0x101010A0						

Excepcions i interrupcions

18-19Q2 - Exercici 4

	Afirmació	V	F
1.-	Si el bit EXL val 1, les interrupcions seran ignorades.		
2.-	Una excepció no pot ser atesa fins que la instrucció que l'ha causada hagi finalitzat.		
3.-	En un sistema amb memòria virtual, la mida total d'un programa i les seves dades poden excedir la capacitat de la memòria física.		
4.-	La divisió d'enters codificats en el format de Ca2 no pot produir overflow.		
5.-	En format de simple precisió IEEE-754 (32 bits), la codificació 0x00F00000 representa un número normalitzat.		
6.-	En una memòria cache amb política d'escriptura immediata sense assignació, un accés a la memòria cache pot implicar dos accesos a memòria principal.		
7.-	En una subrutina, una variable local de tipus enter sempre es guardarà en un registre.		
8.-	La rutina RSE de tractament d'excepcions del MIPS segueix les regles de l'ABI que s'estableixen per programar les subrutines.		
9.-	Al MIPS es detecta que un accés a memòria causa una fallada de pàgina consultant el bit V en el TLB.		
10.-	La codificació en excés de l'exponent en el format de coma flotant simplifica les operacions de comparació.		