COGNOMS:	**GRUP:**
NOM:	

EXAMEN FINAL D'EC
11 de juny de 2019

L'examen consta de **10** preguntes, que s'han de contestar als mateixos fulls de l'enunciat. No oblidis posar el teu nom i cognoms a tots els fulls. La durada de l'examen és de 180 minuts. Les notes, la solució i el procediment de revisió es publicaran al Racó el dia **25** de juny.

Pregunta 1. (1 punt)

Considerem un computador amb un processador MIPS funcionant a una freqüència de 4GHz i que dissipa una potència de 100W. Sobre aquest processador s'executa un programa que consta de dues subrutines, *func1* i *func2*, executades una rere l'altra:

```
main()
{
        func1();
        func2();
}
```

La següent taula mostra, per a cada tipus d'instrucció, el seu CPI i el nombre d'instruccions executades tant a *func1* com a *func2,* referents a l'execució d'aquest programa:

Tipus d'instrucció	CPI	Nombre instruccions *func1*	Nombre instruccions *func2*
Aritmètiques	4	$3 \cdot 10^9$	$10 \cdot 10^9$
Salts	2	$2 \cdot 10^9$	$16 \cdot 10^9$
Altres	1	$4 \cdot 10^9$	$8 \cdot 10^9$

a) Calcula el temps d'execució de tot el programa, en segons

t_{exe} = **25** s

b) Calcula l'energia total consumida durant l'execució completa del programa, en Joules

E = **2500** J

c) Suposem que optimitzem el codi de la subrutina *func2*, reduint el nombre d'instruccions aritmètiques a la meitat aconseguint la mateixa funcionalitat. Quin serà el guany de rendiment (speedup) per al programa complet?

S_{total} = **1,25**

d) D'acord amb la llei d'Amdahl, quin serà el guany de rendiment (speedup) màxim que podem obtenir per al programa complet a l'optimitzar només la subrutina *func2*?

S_{max} = **5**

Pregunta 2. (0,9 punts)

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

```
char a = -1;
unsigned short b[2] = {3, 21};
unsigned short *c = &b[0];
char d[3] = "CA";
float e = 1.25;
```

a) Tradueix-la al llenguatge assemblador del MIPS.

```
        .data
a:      .byte -1
b:      .half 3, 21
c:      .word b
d:      .asciiz "CA"
e:      .float 1.25
```

b) Completa la següent taula amb el contingut de memòria en hexadecimal. Tingues en compte que el codi ASCII de la 'A' és el 0x41. Les variables s'emmagatzemen a partir de l'adreça 0x10010000. Les posicions de memòria sense inicialitzar es deixen en blanc.

@Memòria	Dada
0x10010000	**FF**
0x10010001	
0x10010002	**03**
0x10010003	**00**
0x10010004	**15**
0x10010005	**00**
0x10010006	
0x10010007	

@Memòria	Dada
0x10010008	**02**
0x10010009	**00**
0x1001000A	**01**
0x1001000B	**10**
0x1001000C	**43**
0x1001000D	**41**
0x1001000E	**00**
0x1001000F	

@Memòria	Dada
0x10010010	**00**
0x10010011	**00**
0x10010012	**A0**
0x10010013	**3F**
0x10010014	
0x10010015	
0x10010016	
0x10010017	

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

```
*(c + 1) = *c + 3;
```

```
        la      $t0, c
        lw      $t1, 0($t0)
        lhu     $t2, 0($t1)
        addiu   $t2, $t2, 3
        sh      $t2, 2($t1)
```

COGNOMS:	GRUP:
NOM:	

Pregunta 3. (1,5 punts)

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)?

64 KB

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	**1011**	**SI**	**1A**	**SI**	**SI**	**1010**
L	0x10100008	**1010**	**SI**	**20**	**SI**	**NO**	**77FF**
L	0x2800FFFC	**2800**	**NO**	**3F**	**NO**	**NO**	**-**
E	0x77FFA400	**77FF**	**SI**	**25**	**SI**	**SI**	**57C0**
E	0x1011A274	**1011**	**NO**	**1A**	**NO**	**NO**	**-**
L	0x101010A0	**1010**	**NO**	**20**	**NO**	**NO**	**-**

Pregunta 4. (0,5 punts)

Posa una X al costat de cada una de les següents afirmacions (a la columna V si és Verdadera o a la columna F si és Falsa). Suposem en tots els casos que es fa referència a un processador MIPS com l'estudiat a classe. Cada resposta correcta suma 0,05 punts; les respostes no contestades no es tenen en compte; cada resposta incorrecta resta 0,05 punts; i la puntuació total mínima és 0.

	Afirmació	V	F
1.-	Si el bit EXL val 1, les interrupcions seran ignorades.	**X**	
2.-	Una excepció no pot ser atesa fins que la instrucció que l'ha causada hagi finalitzat.		**X**
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.	**X**	
4.-	La divisió d'enters codificats en el format de Ca2 no pot produir overflow.		**X**
5.-	En format de simple precisió IEEE-754 (32 bits), la codificació 0x00F00000 representa un número normalitzat.	**X**	
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.		**X**
7.-	En una subrutina, una variable local de tipus enter sempre es guardarà en un registre.		**X**
8.-	La rutina RSE de tractament d'excepcions del MIPS segueix les regles de l'ABI que s'estableixen per programar les subrutines.		**X**
9.-	Al MIPS es detecta que un accés a memòria causa una fallada de pàgina consultant el bit V en el TLB.	**X**	
10.-	La codificació en excés de l'exponent en el format de coma flotant simplifica les operacions de comparació.	**X**	

COGNOMS:	**GRUP:**
NOM:	

Pregunta 5. (0,9 punts)

Considera el següent fragment de codi MIPS que forma part d'un programa principal:

```
        sltu   $t0, $t2, $t1
        beq    $t0, $zero, sino
        slti   $t0, $t3, 5
        sltiu  $t4, $t0, 1
        b      fisi
sino:
        slt    $t4, $zero, $t3
fisi:
```

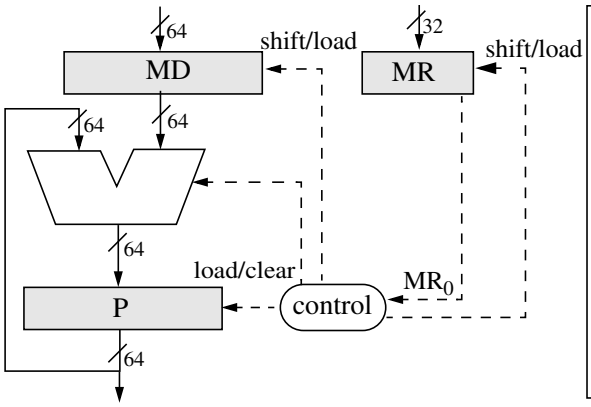
Completa les següents caselles per tal que reflecteixin un programa C equivalent a l'anterior. Heu de posar els tipus que falten de les variables locals (`int` o `unsigned int`) i les parts de la sentència condicional que falten (condició, codi del llavors i codi del sinó).

```
main() {
  unsigned int          a; /* variable local a $t1 */
  unsigned int          b; /* variable local a $t2 */
  int                   c; /* variable local a $t3 */
  int                   d; /* variable local a $t4 */

        if ( b<a          )
             d = (c>=5)   ;
        else
             d = (0<c)    ;
}
```

Pregunta 6. (0,8 punts)

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.

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	**1**	**1**	**0**	**X**	**X**	**1**
cada iteració	**0**	**0**	MR_0	**1**	**1**	**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.

Solament cal canviar el `for` per un `while`, així:

```
while (MR > 0)
{
        ...
}
```

COGNOMS:	GRUP:
NOM:	

Pregunta 7. (1,2 punts)

Donat el següent codi en llenguatge C:

```
int M[100][100];
int I[100];

main() {
int x;        /* variable local a $t0 */
       for (x=0; x<100; x++)
              M[I[x]][I[x]] = M[x][x];
}
```

Considerant que les variables globals ja estan declarades i inicialitzades, completa el codi del programa principal en MIPS.

Aplica la tècnica d'accés seqüencial sempre que sigui possible utilitzar-la.

```
main:   la      $t2, M              # $t2: punter a M[x][x]
        move    $t3, $t2            # $t3: adreça base d'M
        la      $t4, I              # $t4: punter a I[x]
        li      $t5, 404

        li      $t0, 0
        li      $t1, 100
for:    bge     $t0, $t1, fifor

        lw      $t6, 0($t2)         # accés seqüencial a M[x][x]
        lw      $t7, 0($t4)         # accés seqüencial a I[x]
        mult    $t7, $t5
        mflo    $t7                 # $t7: I[x]*404
        addu    $t7, $t7, $t3       # $t7: adreça M[I[x]][I[x]]
        sw      $t6, 0($t7)         # accés aleatori a M[I[x]][I[x]]
        addu    $t2, $t2, $t5       # increment stride accés M[x][x]
        addiu   $t4, $t4, 4         # increment stride accés I[x]

        addiu   $t0, $t0, 1
        b       for
fifor:  jr      $ra
```

Pregunta 8. (0,8 punts)

Donada la següent funció en C, tradueix-la a lleguatge MIPS:

```
float inc_f(float *x) {
        return *x+1.0;
}
```

```
inc_f:   lwc1    $f0, 0($a0)
         lui     $t0, 0x3F80
         mtc1    $t0, $f1
         add.s   $f0, $f0, $f1
         jr      $ra
```

Pregunta 9. (1,2 punts)

Donat el següent codi en C:

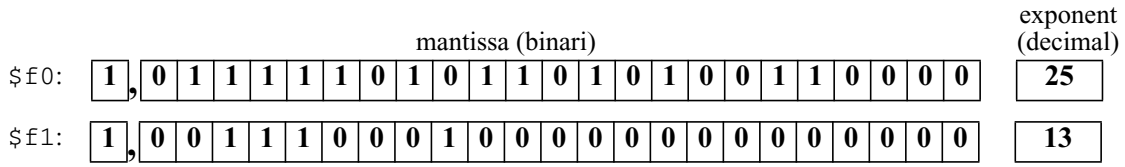
```
float res, i;
main() {
        res = 0.0;
        for (i = 1.0; i <= 10000.0; i = i + 1.0)
                res = res + i;
}
```

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:

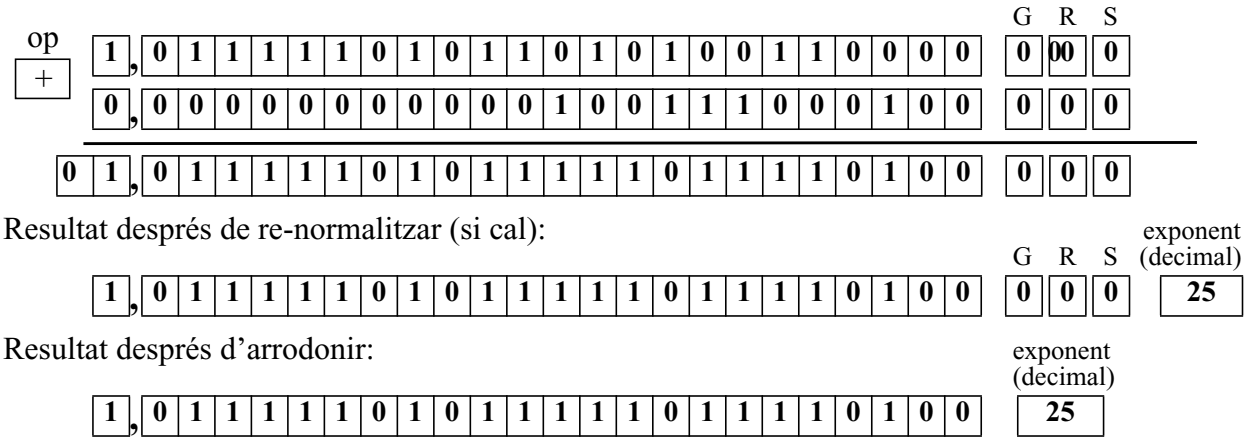
COGNOMS: **GRUP:**

NOM:

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:



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

`$f0 =` **0x4C3EBEF4**

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?

Per errors de precisió produïts en els arrodoniments dels càlculs en iteracions anteriors del bucle.

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?

La suma en coma flotant no és una operació associativa, a causa dels arrodoniments

Pregunta 10. (1,2 punts)

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.

L/E	adreça (hex)	núm. de conjunt	encert (e)/ fallada (f)	bytes de MP	
				llegits	escrits
L	0x100110F8	**15**	**F**	**16**	
L	**0x10010000**	**0**	**F**	**16**	
E	**0x10010000**	**0**	**E**		**4**
L	**0x100110FC**	**15**	**E**		
L	**0x10010000**	**0**	**E**		
E	**0x10010000**	**0**	**E**		**4**
L	**0x10011100**	**0**	**F**	**16**	
L	**0x10010000**	**0**	**E**		
E	**0x10010000**	**0**	**E**		**4**