COGNOMS:	**GRUP:**
NOM:	

EXAMEN FINAL D'EC
17 de gener de 2023

L'examen consta de 9 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 duració de l'examen és de 180 minuts. Les notes i la solució es publicaran al Racó el dia 23 de gener. La revisió es farà presencialment el 24 de gener, de 8 a 9 (preguntes 1 a 4) i de 9 a 10h (preguntes 5 a 9).

Pregunta 1. (1 punt)

Un processador que funciona a una freqüència de rellotge de 5Ghz disposa de 3 tipus d'instruccions diferents: A, B i C. La següent taula mostra quin és el nombre d'instruccions executades de cada tipus durant l'execució d'un programa de test, el CPI promig de cada tipus d'instrucció, i la potència dissipada durant l'execució de cada tipus d'instrucció, en Watts (Joules/segon).

Tipus d'instrucció	Nombre d'instruccions	CPI	Potència (W)
A	$9 \cdot 10^{10}$	2	10
B	$5 \cdot 10^{10}$	10	4
C	$4 \cdot 10^{10}$	3	5

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

t_{exe} = **160** s

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

E = **880** J

c) Al redissenyar el processador, la freqüència del rellotge es redueix a 4GHz i les instruccions de tipus B passen a tenir només 6 cicles per instrucció. Calcula el guany de rendiment (speed-up) obtingut després del redisseny, donant el resultat amb 2 xifres decimals de precisió.

Guany = **1,06**

Pregunta 2. (1,25 punts)

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

```
short a[] = {255, 11, -9};
char b[] = "2023";   /* s'usen tants bytes com calgui pel string */
short c = 0;
long long d = 18;
short *e = &c;
```

a) Tradueix-la al llenguatge assemblador del MIPS

```
        .data
a:      .half 255, 11, -9
b:      .asciiz "2023"
c:      .half 0
d:      .dword 18
e:      .word c
```

b) Completa la següent taula amb el contingut de memòria en hexadecimal (sense el prefix "0x") de les primeres 32 posicions de memòria. Tingues en compte que el codi ASCII del '0' és el 0x30. Les variables s'emmagatzemen a partir de l'adreça 0x10010000. Les posicions de memòria no ocupades es deixen en blanc.

@Memòria	Dada
0x10010000	**FF**
0x10010001	**00**
0x10010002	**0B**
0x10010003	**00**
0x10010004	**F7**
0x10010005	**FF**
0x10010006	**32**
0x10010007	**30**

@Memòria	Dada
0x10010008	**32**
0x10010009	**33**
0x1001000A	**00**
0x1001000B	
0x1001000C	**00**
0x1001000D	**00**
0x1001000E	
0x1001000F	

@Memòria	Dada
0x10010010	**12**
0x10010011	**00**
0x10010012	**00**
0x10010013	**00**
0x10010014	**00**
0x10010015	**00**
0x10010016	**00**
0x10010017	**00**

@Memòria	Dada
0x10010018	**0C**
0x10010019	**00**
0x1001001A	**01**
0x1001001B	**10**
0x1001001C	
0x1001001D	
0x1001001E	
0x1001001F	

c) Quin és el valor final de $t0 i $t1, en hexadecimal, després d'executar aquest fragment?

```
la      $t0, b
lbu     $t0, 3($t0)
addiu   $t0, $t0, '1'
sltu    $t1, $zero, $t0
```

$t0 = **0x 00000064**

$t1 = 0x **00000001**

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

```
*e = a[0] - a[2];
```

```
la      $t0, a
lh      $t1, 0($t0)
lh      $t2, 4($t0)
subu    $t1, $t1, $t2
la      $t2, e
lw      $t2, 0($t2)
sh      $t1, 0($t2)
```

COGNOMS:	**GRUP:**
NOM:	

Pregunta 3. (1 punt)

Donada la següent funció en C:

```
int f (int i, int M[][400]) {
        return M[i+2][i-4];
}
```

Completa els requadres del següent fragment de codi en assemblador MIPS per tal que sigui la traducció correcta de la funció anterior:

```
f:  li      $t0, 1604
    mult    $t0, $a0
    mflo    $t0
    addu    $t0, $t0, $a1
    lw      $v0, 3184($t0)
    jr      $ra
```

Pregunta 4. (1,50 punts)

Donat el següent codi en C:

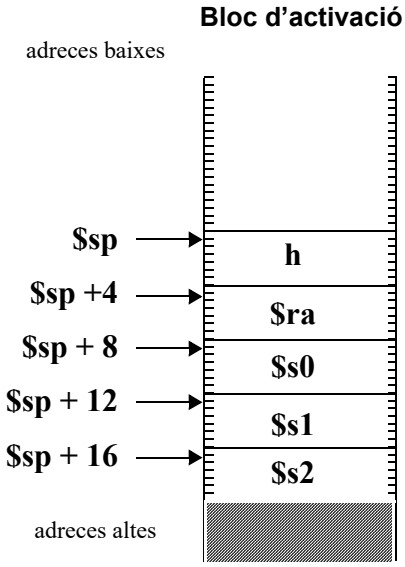
```
int *sub1(int *a, int b);
int sub2(int c, int *d);

int sub3(int *e, int f, int i) {
   int *g, h=0;
   while (h < f) {
          g = sub1(e, i);
          h = sub2(*g, &h);
   }
   return h;
}
```

a) Seguint les regles de l'ABI estudiades, determina quines variables, paràmetres o càlculs intermedis de la funció `sub3` cal guardar obligatòriament en registres segurs per garantir que no siguin alterats per les crides a `sub1` o `sub2`.

e, f, i

b) Dibuixa el bloc d'activació de `sub3`, especificant el nom i posició (relatius al $sp) de cada element que conté, així com la posició on apunta $sp un cop creat el bloc d'activació.



c) Tradueix a MIPS la subrutina `sub3`

```
S3:
          addiu   $sp, $sp, -20
          sw      $ra, 4($sp)
          sw      $s0, 8($sp)
          sw      $s1, 12($sp)
          sw      $s2, 16($sp)
          move    $s0, $a0
          move    $s1, $a1
          move    $s2, $a2
          li      $v0, 0
          sw      $zero, 0($sp)       # h=0
while:
          bge     $v0, $s1, endwhile
          move    $a0, $s0
          move    $a1, $s2
          jal     sub1                # g = s1(e, i);
          lw      $a0, 0($v0)
          move    $a1, $sp
          jal     sub2
          sw      $v0, 0($sp)         # h = s2(*g, &h);
          b       while
endwhile:
          lw      $ra, 4($sp)
          lw      $s0, 8($sp)
          lw      $s1, 12($sp)
          lw      $s2, 16($sp)
          addiu   $sp, $sp, 20
          jr      $ra
```

COGNOMS:	GRUP:
NOM:	

Pregunta 5. (1 punt)

a) Considera que treballem amb nombres enters representats en complement a 2 i en 16 bits. Indica en hexadecimal quin és el nombre *x* més petit tal que la resta 0x7FFF – *x* genera sobreeiximent (o *overfow*).

x = 0x **8000**

b) Considera que compilem el següent programa en C per al processador MIPS

```
main( ) {
short a = 0x789A;
short b = -16;
short c, d, e;
          c = a / b;
          d = a % b;
          e = a >> 4;
}
```

Indica en hexadecimal (amb 4 dígits) quin és el contingut de les variables *c*, *d*, *e* després d'executar l'anterior programa.

c = 0x **F877** d = 0x **000A** e = 0x **789A**

Pregunta 6. (1 punt)

Suposem que tenim un processador de 32 bits amb una memòria cache de dades de 256 bytes, on cada bloc té 16 bytes. Suposem que executem el següent programa.

```
int M[4][32];

void main() {
int i, j;                          // i, j en registre
    for (i=0; i<3; i++)
      for (j=0; j<32; j++)
         M[i][j] = M[i+1][j];
}
```

Calcula el nombre de fallades de la cache suposant que la memòria cache és inicialment buida. L'adreça base de la matriu M és 0.

a) Suposant que la cache és de correspondència directa i té la política d'**escriptura retardada amb assignació**.

fallades = **32** **3*8 lectures + 8 escript.**

b) Suposant que la cache és **associativa per conjunts de 4 vies** (algorisme de reemplaçament LRU), i que té la política d'**escriptura immediata sense assignació**.

fallades = **56** **3*8 lectures + 32 escript.**

Pregunta 7. (1 punt)

Considera que el contingut dels registres `$f4` i `$f6` és `$f4` = **0x40000031** i `$f6` = **0x4200000D**, i s'executa la instrucció MIPS: **`sub.s $f0,$f4,$f6`**. Suposant que el sumador/restador 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:

Quina és la mantissa (en binari) i l'exponent (en decimal) dels nombres que hi ha a `$f4` i `$f6`?

	mantissa (binari)	exponent (decimal)
`$f4:`	1,00000000000000000000001	1
`$f6:`	1,00000000000000000000101	5

Omple les caselles mostrant l'operació op (+/-), els bits a operar, i el resultat en valor absolut:

op			G	R	S
-	1,00000000000000000000101		0	0	0
	0,00010000000000000000011		0	0	1
	0,11110000000000000001001		1	1	1

Resultat en valor absolut després de normalitzar (si cal):

		G	R	S	exponent (decimal)
	1,11100000000000000010011	1	1	0	4

Resultat amb signe (+/-) després d'arrodonir:

signe		exponent (decimal)
-	1,11100000000000000010100	4

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

`$f0` = **0x C1F00014**

COGNOMS:	GRUP:
NOM:	

Pregunta 8. (1,25 punts)

La memòria virtual implementada en un sistema computador de 32 bits es caracteritza pels següents paràmetres:

- Pàgines de 256 bytes de mida.
- Un màxim de 5 pàgines carregades simultàniament a memòria física per aplicació.
- Reemplaçament de pàgines a memòria física seguint l'algorisme LRU.
- TLB totalment associatiu de 4 entrades amb reemplaçament LRU.

Donat el següent programa en C:

```
int V[384];
void main() {
        int i;
        for (i=0; i < 384; i++) {
                V[383 - i] = V[i];
        }
}
```

Considera que la variable local *i* s'emmagatzema en un registre, que el vector global V s'emmagatzema en memòria a partir de l'adreça `0x00000000`, i que el codi s'emmagatzema a partir de l'adreça `0x00000A00`, i ocupa menys d'una pàgina. El TLB i la memòria física estan inicialment buits. Es demana:

a) Quantes pàgines ocupa el vector `V`?

nombre de pàgines = **6**

b) Quantes fallades de TLB (codi i dades) es produiran en tota l'execució del programa?

fallades de TLB = **10**

c) Quantes fallades de pàgina (codi i dades) es produiran en tota l'execució del programa?

fallades de pàgina = **9**

d) Indica els VPN (en hexadecimal) de les cinc pàgines (codi i/o dades) que hi haurà carregades a memòria física quan s'acabi l'execució del programa.

VPNs = **0x0, 0x1, 0x4, 0x5, 0xA**

Pregunta 9. (1 punt)

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 0,1 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.-	En format de simple precisió IEEE-754 (32 bits), la codificació 0x00F00000 representa un número *denormal* (no normalitzat).		**X**
2.-	La suma de números en coma flotant té les propietats commutativa i associativa.		**X**
3.-	La codificació en excés de l'exponent dels nombres en coma flotant permet comparar les seves magnituds amb un simple comparador de naturals.	**X**	
4.-	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**	
5.-	Si el bit EXL val 1, les interrupcions seran ignorades.	**X**	
6.-	Durant el processament de l'excepció causada per una instrucció `syscall`, la RSE li suma 4 al registre EPC abans de retornar al programa.	**X**	
7.-	La rutina RSE de tractament d'excepcions del MIPS segueix les mateixes regles de l'ABI que s'estableixen per programar les subrutines.		**X**
8.-	La rutina de tractament de fallades de TLB està optimitzada de manera que no li cal accedir a la Taula de Pàgines.		**X**
9.-	Quan la CPU detecta una excepció, s'interromp la instrucció en curs per donar pas al tractament de l'excepció.	**X**	
10.-	Quan la CPU rep una petició d'interrupció, s'interromp la instrucció en curs per donar pas al tractament de la interrupció.		**X**

Cognoms: ..... Nom: .....
DNI: .....

EXAMEN FINAL D'EC
13 de juny de 2022

- 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 3:00 hores (180 minuts)
- Les notes i la solució es publicaran al Racó el dia 23 de juny. La revisió es farà presencialment el 27 de juny a les 8:30h.

Pregunta 1 *(1 punt)*

Un computador funciona a una freqüència de rellotge de 3 GHz i dissipa una potència de 30W. Hem simulat un programa executant-se en aquest sistema, suposant que tingués una cache IDEAL (sempre encerta), i hem obtingut, per a cada tipus d'instrucció, el nombre d'instruccions executades i el seu CPI:

	N_{instr}	CPI
Load	$50 \cdot 10^9$	1
Store	$20 \cdot 10^9$	2
Mult	$1 \cdot 10^9$	32
Salts	$9 \cdot 10^9$	2
Aritmètico-lògiques	$70 \cdot 10^9$	1

Quin és el temps d'execució en segons del programa amb cache ideal?

70 s

Sabem que el computador real està equipat amb una cache d'escriptura immediata sense assignació, i que els seus temps representatius són $\mathbf{t_h}$ = 1 cicle (temps d'encert) i $\mathbf{t_{block}}$ = 14 cicles (còpia d'un bloc entre MP i MC o viceversa). Hi hem executat el mateix programa de test de l'apartat anterior i hem observat que el temps d'execució és 140 s. Quina ha estat la taxa de fallades?
Nota: Tingues en compte que les referències a cache inclouen el fetch d'instruccions i l'accés a dades.

7%

Suposem que alimentem el computador amb una bateria carregada amb 6000J, i que executem un programa amb un bucle infinit. Quant de temps (en segons) estarà el computador encès fins a exhaurir la bateria?

200 s

Pregunta 2 *(1,2 punts)*

Codifica els següents números en coma flotant de simple precisió i escriu els resultats en hexadecimal:

El número 12,825	**0x414D3333**
El menor número normalitzat positiu i no nul	**0x00800000**
El menor número denormal (no-normalitzat) positiu i no nul	**0x00000001**
El major número normalitzat positiu (diferent de +Inf)	**0x7F7FFFFF**
Un NaN amb signe positiu	**0x7F8xxxxx** amb **xxxxx** qualsevol excepte **00000**
El –Inf	**0xFF800000**

Cognoms: ..... Nom: .....
DNI: .....

Pregunta 3 *(1 punt)*

Donada la següent funció `foo` en llenguatge C:

```
char foo(char vec[16], char mat[][64], unsigned char k) {
      int i;
      int aux = 0;

      for (i=0; i<=k; i++)
            aux = aux + mat[63-i][k-i];

      return vec[aux%16];
}
```

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 `mat` s'accedeixen utilitzant la tècnica d'accés seqüencial usant el registre `$t0` com a punter. Aquest punter `$t0` s'inicialitza amb l'adreça del primer element del recorregut: `mat[63][k]`. Per calcular l'adreça de l'últim element del recorregut (`$t1`) ho fem a partir de l'adreça del primer element (`$t0`), del nombre d'iteracions que fa el bucle i del nombre de posicions de memòria que es desplaça a cada iteració.

```
foo:                                # $t0 : posició primer element ==> @mat[63][k]
      addiu $t0, $a1, 4032
      addu  $t0, $t0, $a2

                                    # $t1 : posició últim element
      li    $t2, -65
      mult  $a2, $t2
      mflo  $t2
      addu  $t1, $t0, $t2

      move  $t3, $zero              # $t3 : aux=0
loop: lb    $t4, 0($t0)
      addu  $t3, $t3, $t4
      addiu $t0, $t0, -65
      bgeu        $t0, $t1, loop
      andi  $t5, $t3, 0x000F
      addu  $t5, $a0, $t5
      lb    $v0, 0($t5)
      jr    $ra                     # Retorna
```

Pregunta 4 *(1 punt)*

Donades les següents declaracions de variables globals, emmagatzemades a partir de l'adreça 0x10010000:

```
a:    .word      0x10010004
b:    .half      0x00DE
c:    .dword     0xFFFF0000E2E05401
d:    .byte      0xDD
e:    .word      0xFF8406FF
f:    .half      0x0400
g:    .byte      0x40
```

Omple la següent taula amb el contingut de memòria **en hexadecimal**. Posa a ZERO les posicions de memòria sense inicialitzar.

@Memòria	Dada	@Memòria	Dada	@Memòria	Dada	@Memòria	Dada
0x10010000	**0x04**	0x10010008	**0x01**	0x10010010	**0xDD**	0x10010018	**0x00**
0x10010001	**0x00**	0x10010009	**0x54**	0x10010011	**0x00**	0x10010019	**0x04**
0x10010002	**0x01**	0x1001000A	**0xE0**	0x10010012	**0x00**	0x1001001A	**0x40**
0x10010003	**0x10**	0x1001000B	**0xE2**	0x10010013	**0x00**	0x1001001B	**0x00**
0x10010004	**0xDE**	0x1001000C	**0x00**	0x10010014	**0xFF**	0x1001001C	**0x00**
0x10010005	**0x00**	0x1001000D	**0x00**	0x10010015	**0x06**	0x1001001D	**0x00**
0x10010006	**0x00**	0x1001000E	**0xFF**	0x10010016	**0x84**	0x1001001E	**0x00**
0x10010007	**0x00**	0x1001000F	**0xFF**	0x10010017	**0xFF**	0x1001001F	**0x00**

Digues el contingut final **en hexadecimal** a `$t1` després d'executar el codi següent amb les dades declarades anteriorment. Indica també els canvis que es produeixen en l'estat del computador instrucció a instrucció.

```
la      $t0, a          # $t0 = 0x10010000
lw      $t1, 0($t0)     # $t1 = 0x10010004
lb      $t2, 0($t1)     # $t2 = 0xFFFFFFDE
sra     $t2, $t2, 2     # $t2 = 0xFFFFFFF7
sh      $t2, 4($t1)     # Mh[0x10010008] = 0xFFF7
lw      $t1, 4($t1)
```

$t1 = **0xE2E0FFF7**

Digues el contingut final **en hexadecimal** a `$t1` després d'executar el codi següent amb les dades declarades anteriorment. Indica també els canvis que es produeixen en l'estat del computador instrucció a instrucció.

```
la      $t0, d          # $t0 = 0x10010010
lb      $t1, 0($t0)     # $t1 = 0xFFFFFFDD
li      $t0, 1          # $t0 = 0x00000001
addiu   $t0, $t0, 1     # $t0 = 0x00000002
addiu   $t0, $t0, 1     # $t0 = 0x00000003
addu    $t1, $t1, $t0
```

$t1 = **0xFFFFFFE0**

Cognoms: ..... Nom: .....
DNI: .....

Pregunta 5 *(1 punt)*

En un programa MIPS les dades que s'usen poden situar-se en 4 zones ben diferenciades: .data, heap, pila i registres.

Donat el següent programa escrit en C:

```
int VG[100];
int *pG, sum;

int Test() {
   int VL[100];
   int *pL, *pD;

   ...
   pD = malloc(400);
   ...

}
```

On són les variables següents, suposant que els accessos indicats es fan dins la rutina `Test`? Contesta en cada cas **.data**, **heap**, **pila** o **registre**.

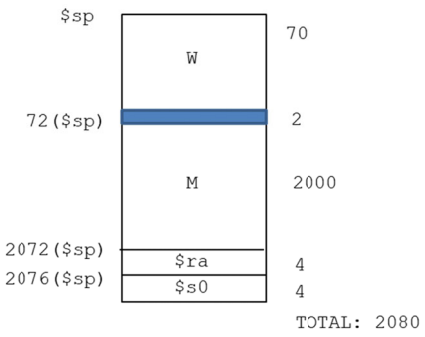
`… = VG[5];`	**.data**
`… = pL;`	**registre**
`… = *(pD+20);`	**heap**
`… = VL[6];`	**pila**
`… = pD;`	**registre**
`… = sum;`	**.data**
`… = pG;`	**.data**

Pregunta 6 *(1,2 punts)*

Donada la següent rutina escrita en C:

```
char Examen(int V[], int N, int k, int *p) {
   char W[70];
   int M[100][5];
   int i,j;                              // emmagatzemats a $t0 i $t1
   ...
   W[50] = Examen2(3, V, M[i][j]);       // sentència B
   ...
   return W[*p];                         // sentència C
}
```

Dibuixa el bloc d'activació de la rutina `Examen`, tenint en compte que a la rutina necessitem els registres segurs `$s0` i `$ra`. Heu d'indicar la mida de tots els elements que apareguin al Bloc d'Activació i els desplaçaments necessaris per accedir-hi. També cal indicar la mida total del Bloc d'Activació.



Tradueix a asemblador MIPS la sentència B.

```
li   $t2, 5
mult $t0, $t2
mflo $t2
addu $t2, $t2, $t1
sll  $t2, $t2, 2
addu $t2, $t2, $sp
lw   $a2, 72($t2)
move $a1, $a0
li   $a0, 3
jal  Examen2
sb   $v0, 50($sp)
```

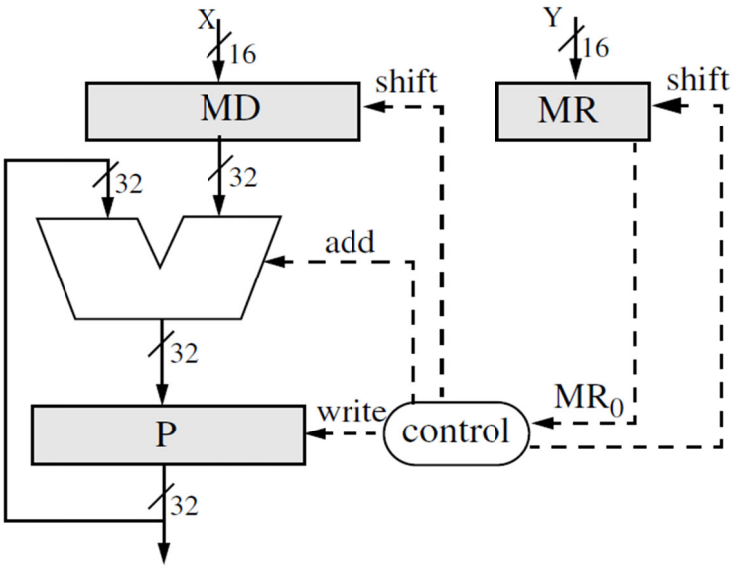
Tradueix a assemblador MIPS la sentència C, incloent-hi l'epíleg de la funció `Examen` (fins al `jr $ra`).

```
lw    $t3, 0($a3)
addu  $t3, $t3, $sp
lb    $v0, 0($t3)
lw    $ra, 2072($sp)
lw    $s0, 2076($sp)
addiu $sp,$sp,2080
jr    $ra
```

Cognoms: ..... Nom: .....
DNI: .....

Pregunta 7 *(1 punt)*

Sigui el circuit seqüencial per a la multiplicació de números naturals de 16 bits, anàleg a l'estudiat al curs, el qual calcula el producte en 32 bits:



Suposem que volem calcular la multiplicació: 0x03D2 * 0x0026. Omple la següent taula (en hexadecimal) indicant quin és el valor dels registres P, MD i MR al final de les 3 primeres iteracions de l'algorisme que controla el circuit.

iter	**P (Producte)**	**MD (Multiplicand)**	**MR (Multiplicador)**
ini	0x00000000	0x000003D2	0x0026
1	**0x00000000**	**0x000007A4**	**0x0013**
2	**0x000007A4**	**0x00000F48**	**0x0009**
3	**0x000016EC**	**0x00001E90**	**0x0004**

Pregunta 8 *(0,6 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 0,1 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
A l'inici de la rutina genèrica de servei d'excepcions de MIPS (RSE) sols s'ha de salvar a la pila aquells registres segurs que es modifiquin durant l'execució de la RSE.		X
L'excepció per fallada de pàgina pot ser inhibida a través del camp *Interrupt Mask*.		X
Si l'accés a dades d'un *store* produeix un encert al TLB, però el bit D val 0, llavors es produeix una excepció.	X	
La rutina *TLBmiss* del processador MIPS copia tots els camps de l'entrada de la taula de pàgines al TLB excepte el bit de presència (bit V del TLB), que es posa sempre a 1.		X
La instrucció *eret* posa EXL a zero i copia al PC el contingut d'EPC.	X	
El tractament de les excepcions al MIPS es fa únicament en dues rutines de servei, una per a la fallada de TLB i una altra per a la resta d'excepcions.	X	

Cognoms: ..... Nom: .....
DNI: .....

Pregunta 9 *(1 punt)*

Suposem que tenim un processador de 32 bits amb una memòria cache de dades de 256 bytes, on cada bloc té 32 bytes. Suposem que executem els següents programes.

```
//programa A
int M[8][64];

void main() {
int i, j;              //en registres
  for (i=0;i<8;i++)
    for (j=0;j<64;j++)
      M[i][j] = 0;
}

//programa B
int M[8][64];

void main() {
int i, j;              //en registres
  for (j=0;j<64;j++)
    for (i=0;i<8;i++)
      M[i][j] = M[i][j]+1;
}
```

Calcula el nombre de fallades de la cache suposant que la memòria cache és inicialment buida. L'adreça base de la matriu M és 0.

Suposant que la cache és de correspondència directa i té la política d'escriptura retardada amb assignació.

Fallades A = **64**

Fallades B = **512**

Suposant que la cache és completament associativa (algorisme de reemplaçament LRU), i que té la política d'escriptura immediata sense assignació.

Fallades A = **512**

Fallades B = **64**

Pregunta 10 *(1 punt)*

Considera un processador MIPS de **32 bits** amb sistema de memòria virtual paginada. Les pàgines són de **4 Kbytes**, la memòria física de **16 Kbytes** i la política de reemplaçament de pàgines físiques és LRU. El sistema està complementat per un TLB completament associatiu **de dues entrades**, amb reemplaçament LRU.

Quins bits de l'adreça lògica serviran per conèixer el número de pàgina lògica (*virtual page numb*er o VPN)?

31 .. 12

Quants marcs de pàgina té la memòria física?

4

Quina és la grandària (en bits) d'una entrada de la taula de pàgines (TP)? i del TLB?

4 i 24, respectivament

Suposant que partim d'un estat inicial, amb la memòria física buida i el TLB també buit, emplena la següent taula que mostra una seqüència de referències a memòria (E: escriptura/ L: lectura):

Nota: *No cal considerar dins la memoria física l'espai que ocupa la pròpia taula de pàgines, que suposarem que s'emmagatzema en una memòria específica.*

adr. lògica	TLB miss	fallada de pàgina?	escriptura disc?	lect./esc. TP?*
L:0x10010A3F	*Sí*	*Sí*	*No*	*Sí*
L:0x10011001	*Sí*	*Sí*	*No*	*Sí*
L:0x10011433	**No**	**No**	*No*	**No**
E:0x10011C31	*No*	*No*	**No**	**Sí**
L:0x10012D63	*Sí*	*Sí*	*No*	*Sí*
L:0x10010464	**Sí**	**No**	**No**	*Sí*
L:0x10013801	*Sí*	*Sí*	*No*	*Sí*
L:0x10014F6B	*Sí*	*Sí*	**Sí**	*Sí*
L:0x10015432	*Sí*	*Sí*	**No**	*Sí*

**Si cal accedir (llegir o escriure) a la taula de pàgines (TP).*

Final EC

Cognoms: ..... Nom: .....
DNI: .....

EXAMEN FINAL D'EC
18 de gener de 2022

- L'examen consta de 9 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 3:00 hores (180 minuts)
- Les notes i la solució es publicaran al Racó el dia 24 de gener. La revisió es farà presencialment el 25 de gener a les 8:30h.

Pregunta 1 *(1 punt)*

Tenim dos processadors MIPS diferents, el model A i el model B, amb les característiques següents:

	A	**B**
CPI load/store	5	3
CPI resta instruccions	2	3
Freqüència	1 GHz	1 GHz
Potència dissipada	10 W	12 W

Ens han demanat que avaluem quin dels dos processadors seria millor per executar el programa següent, tant des del punt de temps d'execució com d'eficiència energética. Adona't que les instruccions en negreta són **macros**.

```
.data
A:      .space      400
S:      .word       35

.text
.globl main
main:
        addiu       $t0, $zero, 0
        la          $t1, A
        la          $t2, S
        lw          $t2, 0($t2)
for:
        sltiu       $t3, $t0, 100
        beq         $t3, $zero, fifor
        sll         $t3, $t0, 2
        addu        $t3, $t3, $t1
        lw          $t4, 0($t3)
        addu        $t4, $t4, $t2
        sw          $t4, 0($t3)
        addiu       $t0, $t0, 1
        beq         $zero, $zero, for
fifor:
        jr          $ra
```

Indica quants cicles trigarà cada processador a executar el programa, i per tant quin serà el més ràpid:

Cicles A: **2421** Cicles B: **2727** Millor: **A**

Indica quin dels dos processadors consumirà menys energia executant el programa? Per què?

El processador A. L'energia consumida ve donada pel producte del temps d'execució i la potència dissipada. A igual freqüència només cal comparar els productes dels cicles per les potències.

Pregunta 2 *(1 punt)*

Considera una funció següent que retorna al vector `suma` la suma de cada columna de la matriu `mat`:

```
void suma_per_columnes(int mat[M][N], int suma[N]) {
int i;
	for (i = 0; i < N; ++i) {
		suma[i] = 0;
		for (int j = 0; j < M; ++j) {
			suma[i] += mat[j][i];
		}
	}
}
```

Completa el següent codi en MIPS per a que sigui una traducció de la subrutina en C usant la tècnica d'accés seqüencial (deixa les respostes en funció de `N` i `M`):

```
suma_per_columnes:
	addiu		$t0, $a0,	0
	addiu		$t1, $a1,	0
	li		$t2, 0
	li		$t3, N
buc1:
	bge		$t2, $t3, fibuc1
	li		$t4, 0
	li		$t5, 0
	li		$t6, M
buc2:
	bge		$t5, $t6, fibuc2
	lw		$t7, 0($t0)
	addu		$t4, $t4, $t7
	addiu		$t0, $t0,	N*4
	addiu		$t5, $t5,	1
	b		buc2
fibuc2:
	sw		$t4, 0($t1)
	addiu		$t0, $t0,	-(M*N*4)+4
	addiu		$t1, $t1,	4
	addiu		$t2, $t2,	1
	b		buc1
fibuc1:
	jr		$ra
```

Cognoms: ..... Nom: .....
DNI: .....

Pregunta 3 *(1 punt)*

Donades les següents declaracions de variables globals en assemblador MIPS, que s'ubiquen a memòria a partir de l'adreça 0x10010000:

```
.data
A:      .word       1, 2
B:      .asciiz     "403"               # '0' és 0x30 en ASCII
C:      .float      1.0
D:      .byte       '5', 4, '3'
        .align      2
E:      .space      2
F:      .word       B
G:      .half       -1, -2
```

Omple la següent taula amb el contingut de memòria **en hexadecimal**. Les variables s'emmagatzemen a partir de l'adreça 0x10010000. Deixa EN BLANC les posicions de memòria sense inicialitzar.

@Memòria	Dada	@Memòria	Dada	@Memòria	Dada	@Memòria	Dada
0x10010000	**01**	0x10010008	**34**	0x10010010	**35**	0x10010018	**08**
0x10010001	**00**	0x10010009	**30**	0x10010011	**04**	0x10010019	**00**
0x10010002	**00**	0x1001000A	**33**	0x10010012	**33**	0x1001001A	**01**
0x10010003	**00**	0x1001000B	**00**	0x10010013		0x1001001B	**10**
0x10010004	**02**	0x1001000C	**00**	0x10010014	**00**	0x1001001C	**FF**
0x10010005	**00**	0x1001000D	**00**	0x10010015	**00**	0x1001001D	**FF**
0x10010006	**00**	0x1001000E	**80**	0x10010016		0x1001001E	**FE**
0x10010007	**00**	0x1001000F	**3F**	0x10010017		0x1001001F	**FF**

Digues el contingut final **en hexadecimal** a `$t0` després d'executar el codi següent amb les dades declarades a l'apartat anterior:

```
la      $t1, F
lw      $t1, 0($t1)
lb      $t1, 2($t1)
sll     $t1, $t1, 1
andi    $t0, $t1, 0x0F0F
```

`$t0 =` **0x00000006**

Digues el contingut final **en hexadecimal** a `$t1` després d'executar el codi següent:

```
li      $t0, 0xABCDABCD
sra     $t1, $t0, 9
slt     $t1, $t1, $t0
```

`$t1 =` **0x00000000**

Pregunta 4 *(1.5 punts)*

Considera les següents declaracions de funcions en C:

```
short f(int *a, short b, char *c);
short examen(short *x, int y[], char *z) {
      short w[3];
      int p;

      w[0] = f(&p, *(x+1), z);
      w[1] = f(y+2, *x, z);
      w[2] = 3;

      return w[0] + w[1] + w[2];
}
```

Per a totes les variables i arguments de la rutina `examen`, indica si els guardaries a la pila, en registres segurs o en registres temporals. Justifica la teva resposta.

`w` i `p` s'han de guardar a la pila (`w` és un vector i es demana calcular l'adreça de `p` dins el codi)

`x`, `y` i `z` s'han de guardar en registres segurs (s'ha de preservar el seu valor després d'una crida a subrutina)

la resta de variables es poden guardar en registres temporals

Tradueix la subrutina `examen` a MIPS, seguint les decisions de l'apartat anterior.

```
examen:
      addiu       $sp, $sp, -28
      sw          $s0, 12($sp)
      sw          $s1, 16($sp)
      sw          $s2, 20($sp)
      sw          $ra, 24($sp)
      move        $s0, $a0
      move        $s1, $a1
      move        $s2, $a2

      addiu       $a0, $sp, 8
      lh          $a1, 2($s0)
      jal         f
      sh          $v0, 0($sp)

      addiu       $a0, $s1, 8
      lh          $a1, 0($s0)
      move        $a2, $s2
      jal         f

      …
```

```
…

addiu       $v0, $v0, 3
lh          $t0, 0($sp)
addu        $v0, $v0, $t0

lw          $s0, 12($sp)
lw          $s1, 16($sp)
lw          $s2, 20($sp)
lw          $ra, 24($sp)
addiu       $sp, $sp, 28
jr          $ra
```

Cognoms: ..... Nom: .....
DNI: .....

Pregunta 5 *(1 punt)*

Escriu un codi en assemblador de MIPS, **sense cap instrucció de salt**, per calcular el producte de dos números naturals representats en 32 bits i emmagatzemats a `$a0` i `$a1`. El resultat cal deixar-lo a `$v0`, que també s'haurà de posar a zero si el producte no es pot representar en 32 bits.

```
multu   $a0, $a1
mflo    $v0
mfhi    $t0
sltiu   $t0, $t0, 1
subu    $t0, $zero, $t0
and     $v0, $v0, $t0
```

Pregunta 6 *(1 punt)*

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 0,1 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
En un processador amb adreces de 32 bits, una cache associativa de 4 vies, d'1MB i blocs de 32 bytes, s'han de dedicar 14 bits a etiqueta (TAG), 13 a número d'entrada (conjunt) i 5 a desplaçament.	X	
Quan es produeix una interrupció s'atura la instrucció en marxa, i quan s'ha acabat el servei a la interrupció es torna a llançar l'esmentada instrucció.		X
En un sistema de paginació una fallada d'escriptura a l'espai de memòria virtual fa que ens portem la pàgina que falla de la memòria principal a l'àrea de swap (disc), i escrivim la dada modificada tant a l'àrea de swap (disc) com a la memòria principal.		X
Un processador sense TLB pot suportar memòria virtual.	X	
Es pot produir més d'una excepció per fallada de TLB en l'execució d'una mateixa instrucció.	X	
Quan es produeix una excepció o interrupció, el bit EXL canvia per prohibir les interrupcions, indicant al mateix temps que estem en mode usuari.		X
En les memòries cache que utilitzen escriptura immediata s'ha de posar el dirty bit a 1 només quan hi ha un encert d'escriptura.		X
El temps mitjà d'accés a memòria de les instruccions load i store en un computador que disposa de memòria virtual amb TLB, normalment és superior al temps d'accés de la memòria principal.		X
L'excepció per accés no alineat a memòria pot ser inhibida a través del camp Interrupt Mask.		X
La diferència entre les instruccions MIPS *add*/*addu* radica en què *addu* pot generar una excepció per sobreeiximent (overflow) a la suma d'enters mentre que *add* mai pot generar cap excepció.		X

Pregunta 7 *(1,50 punts)*

Suposem que tenim un processador de 32 bits amb una memòria cache de dades de 8KB associativa per conjunts de 2 vies, on cada bloc té 32 bytes, i que es segueix l'algorisme de reemplaçament LRU.

Calcula el nombre de fallades de la cache en els accessos a cada vector en executar el següent programa, suposant que la cache té la política d'**escriptura immediata sense assignació** (write-through, no-write-allocate) , i que la memòria cache és inicialment buida.

```
int A[1024], B[1024], C[1024];

void main() {
      int i;
      for (i=0;i<1024;i++)
            A[i]=B[i]+C[i];
}
```

Fallades A = **1024**

Fallades B = **128**

Fallades C = **128**

Es podria reduir el número de fallades modificant el número de blocs per conjunt? Justifica-ho i explica de quina manera, si és que es pot, juntament amb el nombre de fallades a cada vector que s'obtindrien.

No es pot reduir el nombre de fallades. Totes les fallades són obligades (compulsory) per carregar dades a memòria cache. Cap bloc fa fora a cap altre.

Repeteix el problema fent servir ara una política d'**escriptura retardada amb assignació** (write-back, write-allocate).

Fallades A = **1024**

Fallades B = **1024**

Fallades C = **1024**

Es podria reduir el número de fallades modificant el número de blocs per conjunt? Justifica-ho i explica de quina manera, si és que es pot, juntament amb el nombre de fallades a cada vector que s'obtindrien.

Sí que es pot reduir el nombre de fallades augmentant el nombre de blocs per conjunt, com a mínim fins a 3 blocs.

El nombre de fallades quedaria reduït a:

- **Fallades A: 128**
- **Fallades B: 128**
- **Fallades C: 128**

Cognoms: ..... Nom: .....
DNI: .....

Pregunta 8 *(1 punt)*

La memòria virtual implementada en un sistema computador de 32 bits es caracteritza pels següents paràmetres:

- Pàgines de 2KB de mida.
- Un màxim de 5 pàgines carregades simultàniament a memòria física per aplicació.
- Reemplaçament de pàgines a memòria física seguint l'algorisme LRU.
- TLB totalment associatiu de 16 entrades amb reemplaçament LRU.

Donat el següent codi en C:

```
int V[4096];
main() {
      int i;
      int suma = 0;
      for (i=0; i < 4096; i++) {
            suma += V[i] + V[4095- i];
      }
}
```

Considera que:

- les variables locals `i` i `suma` s'enmagatzemen en registres,
- el vector global `V` s'emmagatzema a memòria a partir de l'adreça 0x00000000,
- el codi s'emmagatzema a partir de l'adreça 0x00004000, i ocupa menys d'una pàgina.
- El TLB i la memòria física estan inicialment buits.

Es demana:

Quantes pàgines ocupa el vector V?

8

Quantes fallades de TLB (codi i dades) es produiran en tota l'execució del programa?

9 (1 pel codi i 8 per les pàgines del vector)

Quantes fallades de pàgina (codi i dades) es produiran en tota l'execució del programa?

13 (les 12 per dades en ordre de VPNs: 0,7,1,6,2,5,3,4,6,1,7,0)

Indica els VPN (en hexadecimal) de les pàgines (codi i/o dades) que hi haurà carregades a memòria física quan s'acabi l'execució d'aquest programa.

0x8 (codi)

0x0, 0x7, 0x1, 0x6 (les 4 darreres pàgines accedides del vector)

Pregunta 9 *(1 punt)*

Representa en coma flotant i **en hexadecimal**, segons l'estàndar IEEE 754 el nombre **3,2**.

0x404CCCCC

Calcula l'error absolut comès en la representació del nombre anterior i indica si aquest error és **més petit**, **igual** o **més gran** que 2^{-24}.

MÉS GRAN (l'error és 0,8 * 2^{-23})

Donat el nombre en coma flotant **0xc0180000** indica, **en decimal**, el valor d'aquest nombre real.

-2,375

Final EC

Cognoms: ..... Nom: .....
DNI: .....

EXAMEN FINAL D'EC
17 de juny de 2021

- 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 3:00 hores (180 minuts)
- Les notes, la solució i el procediment de revisió es publicaran al Racó el dia 25 de juny.

Pregunta 1 *(1 punt)*

Un processador d'última generació disposa de 4 tipus d'instruccions diferents: A, B, C i D. La següent taula mostra quin és el nombre d'instruccions executades per un programa sota consideració i el CPI de cada tipus d'instrucció. La freqüència de rellotge del processador és de 2GHz i la potència dissipada és de 251W.

Tipus d'instrucció	#instr.	CPI
A	$4 \cdot 10^9$	1
B	$3 \cdot 10^9$	3
C	$2 \cdot 10^9$	1
D	$1 \cdot 10^9$	3

Calcula el CPI mitjà del programa sota consideració

CPI = **1,8**

Indica quin és el temps d'execució en segons del programa

T_{exe} = **9** s

Calcula l'energia consumida en Joules durant l'execució del del programa

E = **2259** J

Indica quin seria el guany (speed-up) que s'obtindria si s'aconseguís reduir el CPI de les instruccions de tipus B a 1 cicle.

guany = **1,5**

Pregunta 2 *(0.5 punts)*

Quin serà el contingut final de `$t0` en hexadecimal després d'executar el següent codi?

```
li      $t0, -4
sra     $t1, $t0, 31
xor     $t0, $t0, $t1
subu    $t0, $t0, $t1
sll     $t1, $t1, 31
or      $t0, $t0, $t1
```

`$t0` = **0x80000004**

Pregunta 3 *(1 punt)*

Donades les següents declaracions de variables globals en assemblador MIPS, que s'ubiquen a memòria a partir de l'adreça 0x10010000:

```
      .data
a:    .ascii    "cba"                    # el codi ASCII de la 'a' és 0x61
b:    .half     -8
c:    .word     a
d:    .byte     0x05
      .align    1
e:    .byte     0xA5, 0xA4, 0xA3, 0xA2, 0xA1
f:    .float    1.25
```

Omple la següent taula amb el contingut de memòria **en hexadecimal**. Les variables s'emmagatzemen a partir de l'adreça 0x10010000. Deixa EN BLANC les posicions no ocupades per cap dada.

@Memòria	Dada	@Memòria	Dada	@Memòria	Dada	@Memòria	Dada
0x10010000	**63**	0x10010008	**00**	0x10010010	**A3**	0x10010018	
0x10010001	**62**	0x10010009	**00**	0x10010011	**A2**	0x10010019	
0x10010002	**61**	0x1001000A	**01**	0x10010012	**A1**	0x1001001A	
0x10010003		0x1001000B	**10**	0x10010013		0x1001001B	
0x10010004	**F8**	0x1001000C	**05**	0x10010014	**00**	0x1001001C	
0x10010005	**FF**	0x1001000D		0x10010015	**00**	0x1001001D	
0x10010006		0x1001000E	**A5**	0x10010016	**A0**	0x1001001E	
0x10010007		0x1001000F	**A4**	0x10010017	**3F**	0x1001001F	

Pregunta 4 *(1 punt)*

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 0,1 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
La traducció d'adreces virtuals a físiques en una instrucció `lw` pot arribar a produir fins a quatre excepcions.	**X**	
En un computador que disposa de memòria virtual amb TLB sempre cal fer dos accessos a memòria principal per cada referència: un a la Taula de Pàgines I l'altre a l'adreça de la referència.		**X**
Les fallades de TLB causen una excepció i es tracten per software. No obstant, les fallades de TLB invoquen la rutina *TLBmiss*, molt més curta d'executar que la rutina d'excepcions genèrica RSE.	**X**	
El TLB funciona com una memòria cache de la Taula de Pàgines, les etiquetes de la qual consisteixen en el número de pàgina virtual (VPN).	**X**	
En un processador amb adreces de 32 bits, una memòria cache associativa de 4 vies, de 32KB i blocs de 32 bytes, s'han de dedicar 20 bits a l'etiqueta, 7 al número de conjunt i 5 al desplaçament.		**X**
La taula de pàgines es pot modificar quan el bit EXL, del registre Status, val 0.		**X**
Després de servir una interrupció, de vegades, tornem a reexecutar la mateixa instrucció durant la qual ha arribat aquesta interrupció.		**X**
Podem desactivar les excepcions per *overflow* posant a 0 el bit corresponent a aquestes excepcions del camp IM del registre Status.		**X**
Les interrupcions es gestionen com un tipus concret d'excepció, i permeten al sistema operatiu interaccionar amb els dispositius externs de forma asíncrona.	**X**	
La instrucció `tlbwr` implementa la política de reemplaçament LRU pel TLB.		**X**

Cognoms: ..... Nom: .....
DNI: .....

Pregunta 5 *(0,75 punts)*

Donat el següent fragment de codi en llenguatge C

```
unsigned int x,y;

if (( x > y ) && ( y >= 10 )) {
      y++;
      x = x / 4;
} else {
      x = y;
}
```

L'hem traduït a assemblador MIPS sense fer servir macros o pseudoinstruccions. Completa els següents requadres amb els corresponents mnemònics i operands a fi que la traducció sigui correcta.

```
      sltu      $t3, $t1, $t0
      beq       $t3, $zero, else
      addiu     $t7, $zero, 10
      sltu      $t3, $t1 , $t7
      bne       $t3 , $zero , else
then:
      addiu     $t1, $t1, 1
      srl       $t0, $t0, 2
      beq       $zero, $zero, endif
else:
      addu  $t0, $t1, $zero
endif:
```

Pregunta 6 *(1 punt)*

Escriu un codi en assemblador de MIPS (màxim 4 instruccions) que multipliqui els valors de `$t0` i `$t1` (ambdós naturals) de forma que `$t3` tingui el resultat de la multiplicació i `$t7` valgui 0 si hi ha hagut sobreeiximent, i 1 altrament.

```
multu $t1, $t0
mflo  $t3
mfhi  $t7
sltiu $t7, $t7, 1
```

Pregunta 7 *(1 punt)*

Donat el següent fragment de codi en llenguatge C

```
int   vec[N];
short ind[N][N];

int i, sum = 0;
for (i=0; i<N; i++)
      sum += vec[ ind[i][i] ] + vec[ ind[N-i-1][2] ];
```

L'hem traduït a assemblador MIPS. Completa els següents requadres amb valors (en funció d'N si cal) a fi que la traducció sigui correcta.

```
      li          $t0, 0            # i
      li          $t7, N
      li          $v0, 0            # sum

      la          $t1, vec
      la          $t2, ind
      addiu       $t3, $t2, ((N-1)*N+2)*2

loop:
      beq         $t0, $t7, end_loop

      lh          $t5, 0($t2)
      sll         $t5, $t5, 2
      addu        $t5, $t5, $t1
      lw          $t5, 0($t5)
      addu        $v0, $v0, $t5

      lh          $t5, 0($t3)
      sll         $t5, $t5, 2
      addu        $t5, $t5, $t1
      lw          $t5, 0($t5)
      addu        $v0, $v0, $t5

      addiu       $t2, $t2, (N+1)*2
      addiu       $t3, $t3, -N*2
      addiu       $t0, $t0, 1
      b           loop
end_loop:
```

Cognoms: ..... Nom: .....
DNI: .....

Pregunta 8 *(1.25 punts)*

Tenim un processador MIPS amb un co-processador de coma flotant, on els registres `$f4` i `$f6` contenen els valors `0x4136DB6D` i `0xBFB6DB6B`, respectivament. En aquest processador, s'executa la instrucció MIPS `add.s $f0, $f4, $f6`. La unitat de suma i resta utilitza els tres bits GRS de guarda i arrodoneix al més pròxim. Contesta les següents preguntes:

Quina és la mantissa (en binari) i l'exponent (en decimal) dels nombres que hi ha a `$f4` i `$f6`?

	mantissa (binari)	exponent (decimal)
`$f4:`	1,01101101101101101101101	3
`$f6:`	1,01101101101101101101011	0

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

		mantissa (binari)	G R S
op		1,01101101101101101101101	0 0 0
-		0,00101101101101101101101	0 1 1
	1	1,00111111111111111111111	1 0 1

Omple el resultat després de re-normalitzar i arrodonir:

mantissa (binari)	exponent (decimal)
1,01000000000000000000000	3

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

`$f0 =` **0x41200000**

Quin valor decimal representa `$f0` després d'executar la instrucció?

`$f0 =` **10,0**

Pregunta 9 *(1.25 punts)*

Considera un processador amb adreces de 32 bits i una memòria cache de dades amb les següents característiques:

- Capacitat: 1kB
- Mida del bloc: 64 bytes
- Correspondència associativa per conjunts
- Blocs per conjunt: 2
- Política de reemplaçament LRU
- Escriptura retardada amb assignació (write-back, write-allocate)

Suposem que la memòria cache és inicialment buida, i executem el següent programa al nostre processador:

```
int V[128];
int M[128][16];

void main() {
int i,j,tmp;              // a $t0, $t1 i $t2 respectivament

      for (i=0; i<128; i++)
            for (j=0; j<16; j++) {
                  tmp = M[i][j];
                  V[i] = V[i] + tmp;
            }
}
```

Considera addicionalment que el vector `V` comença a partir de l'adreça 0x10010000 i que tant el vector `V` com la matriu `M` están convenientment inicialitzats, tot i que no s'ha de tenir en compte aquesta inicialització en la resolució de l'exercici.

Quin serà el número del bloc de MP on es troba `M[0][0]`, en hexadecimal?

Bloc MP `M[0][0]` = **0x0400408**

Quantes referències a memòria per dades es realitzaran durant l'execució del programa, i quantes fallades de cache es produiran?

Referències = **6144** Fallades = **136**

Decidim experimentar canviant aquesta cache per una altra amb les mateixes característiques (1kB, blocs 64 bytes) però amb una política de correspondència directa.

Quina serà ara la quantitat de fallades que s'obtindrà en l'execució d'aquest codi?

Fallades = **384**

Pregunta 10 *(1,25 punts)*

Considera un processador MIPS amb un sistema de memòria virtual paginada. Les pàgines són de 4kB de mida i la política de reemplaçament és LRU. El sistema operatiu determina que la memòria física només mantindrà 4 pàgines per programa.

Per simplificar el problema només es consideren els accessos a dades, descartant la gestió dels accessos a instruccions.

L'estat de la taula de pàgines en un moment donat de l'execució d'un programa és el següent (només mostrem les entrades amb P=1):

VPN	PPN	P	D
0x10010	0x0	1	1
0x10020	0x3	1	1
0x1007F	0x2	1	0
0x10080	0x1	1	0

El sistema està complementat per un TLB completament associatiu de dues entrades, amb reemplaçament LRU. L'ordre de les últimes referències a memòria és el següent: 0x10020 (accés més llunyà) , 0x1007F, 0x10080, 0x10010 (accés més recent).

Considera la seqüència d'accessos a memòria de la taula següent. Per a cada accés, on s'indica l'adreça de l'accés en hexadecimal i si és Lectura (L) o Escriptura (E), omple el seu número de pàgina lògica (VPN), si provoca una fallada de TLB, si provoca una fallada de pàgina, si s'ha d'escriure a disc, quina pàgina lògica es reemplaça i quin és el PPN resultat de la traducció d'adreces. A les columnes amb preguntes booleanes escriu "SI" o "NO". A les columnes numèriques escriu la dada en hexadecimal.

L/E	Adreça (hex)	VPN (hex)	Fallada TLB?	Fallada Pàgina?	Escript. Disc?	VPN Reempl. (hex)	PPN (hex)
E	10080874	**10080**	**NO**	**NO**	**NO**	**-**	**1**
L	1009077C	**10090**	**SI**	**SI**	**SI**	**10020**	**3**
E	10020AA0	**10020**	**SI**	**SI**	**NO**	**1007F**	**2**
L	1007EFFC	**1007E**	**SI**	**SI**	**SI**	**10010**	**0**
L	10010998	**10010**	**SI**	**SI**	**SI**	**10080**	**1**

COGNOMS, NOM:

EXAMEN FINAL D'EC

19 de gener de 2021

L'examen consta de 10 preguntes, que s'han de contestar als mateixos fulls de l'enunciat. No oblideu posar el 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.

Problema 1. (1 punt)

Donat el següent codi MIPS

```
f1:  addiu     $sp, $sp, -16
     sw        $s0, 8($sp)
     sw        $ra, 12($sp)
     move      $s0, $a1
     move      $a1, $a0
     move      $a0, $sp
     jal       f2
     lw        $v0, 0($sp)
     lw        $t0, 4($sp)
     addu      $v0, $v0, $t0
     addu      $v0, $v0, $s0
     lw        $s0, 8($sp)
     lw        $ra, 12($sp)
     addiu     $sp, $sp, 16
     jr        $ra
```

i el prototipus en llenguatge C de la funció `f2`

```
void f2(int a[], int b);
```

Completeu el següent codi C amb una funció `f1`, que ha de correspondre amb la subrutina `f1` abans esmentada.

```
int f1(int a, int b) {
    int v[2];

    f2(v, a);

    return v[0] + v[1] + b;


}
```

COGNOMS, NOM:

Problema 2. (1 punt)

Donat el següent programa escrit en llenguatge C, traduïu el programa principal a codi MIPS de forma que s'optimitzi el nombre total d'instruccions executades. La matriu i el vector no cal declarar-los i considereu que ja estan inicialitzats.

```
int M[200][128];
int V[50];

main() {
int sum=0;
    for (i=0; i<50; i++) sum += M[V[i]][2*i];
}
```

```
main:
          li        $t0, 0          # sum
          la        $t1, V          # punter V: val ini
          la        $t2, V+200      # punter V: val fi
          la        $t3, M          # @base M
          li        $t4, 0          # despl. 2*i
bucle:
          lw        $t5, 0($t1)     # accés seq. V
          sll       $t5, $t5, 9
          addu      $t5, $t3, $t5
          addu      $t5, $t5, $t4
          lw        $t5, 0($t5)     # accés aleatori M
          addu      $t0, $t0, $t5
          addiu     $t1, $t1, 4     # act. punter V
          addiu     $t4, $t4, 8     # act. despl. 2*i
          bne       $t1, $t2, bucle

          jr        $ra
```

COGNOMS, NOM:

Problema 3. (1 punt)

Suposem un programa que s'executa sobre un procesador funcionant a una freqüència de 5 GHz, el qual dissipa una potencia de 100W. 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:

	Nombre d'instruccions	CPI
Memòria	$10 \cdot 10^{12}$	10
Salts	$5 \cdot 10^{12}$	4
Resta d'instr.	$35 \cdot 10^{12}$	2

Es demana que calculeu

a) El CPI promig de tot el programa

3,8

b) El temps d'execució del programa, en segons

$3{,}8 * 10^4$ s

c) L'energia total consumida durant l'execució del programa, en Joules

$3{,}8 * 10^6$ J

Volem millorar el rendiment del procesador optimitzant la gestió de les instruccions de memòria, amb tècniques més eficients de gestió de la memòria cache.

d) Quin hauria de ser el nou CPI promig de les instruccions de memòria per a obtener un guany de rendiment (speed-up) de 1.25x?

6,2

COGNOMS, NOM:

Problema 4. (1 punt)

Feu un programa en MIPS que modifiqui el registre `$t1` de la següent forma:

- Posi a 1 els bits del 0 al 7
- Intercanvïi els bytes de les posicions 23 a 16 i 15 a 8.
- Posi a 0 els bits del 24 al 31

Recordeu que els bits es numeren del 0 al 31, sent el de menys pes el 0 i el de més pes el 31.

Es valorarà que el programa s'executi amb el menor nombre possible d'instruccions.

```
main:
     andi      $t0, $t1, 0xFF00
     sll       $t0, $t0, 8
     sll       $t1, $t1, 8
     srl       $t1, $t1, 16
     ori       $t1, $t1, 0xFF
     or        $t1, $t0, $t1

     jr        $ra
```

COGNOMS, NOM:

Problema 5. (1 punt)

Un sistema computador amb procesador MIPS gestiona memoria virtual paginada on les pàgines són de 4KB de mida i el reemplaçament usat és LRU. Un programa pot tenir un màxim de 4 pàgines carregades a memoria física.

Considerem que s'està executant un programa que fa servir dades que ocupen 128 KB des de l'adreça base 0x10010000. En un moment donat de l'execució d'aquest programa es tenen les 4 pàgines permeses carregades a memoria física. El contingut de les 4 entrades de la TP que indiquen que la seva página virtual és carregada a memoria física són els següents:

VPN:	PPN	P	D
00400:	10000	1	0
10010:	10001	1	0
10011:	10002	1	1
10012:	10003	1	1

El TLB que s'usa en el sistema és de 2 entrades, completament associatiu i amb reemplaçament LRU. L'ordre en què s'han referenciat les pàgines presents a memoria física és (de la que fa més temps a la que menys i indicades per VPN): 0x10011, 0x10010, 0x10012, 0x00400.

En aquest moment el MIPS és a punt d'executar la instrucció `lw $t0, -8($t1)`, situada a l'adreça 0x00400140. Respecte l'execució d'aquesta instrucció, es demana que indiqueu el contingut en hexadecimal de `$t1` que compleixi la condició establerta en cada apartat. Si hi ha més d'un posible valor indiqueu l'adreça **més baixa** possible. Si no n'hi ha cap de possible, poseu **CAP**.

a) Es detecta un encert de TLB

0x10010008

b) No provoca una fallada de página

0x10010008

c) No es produeix una lectura de página de disc cap a memòria física

0x10010008

d) Es produeix una escriptura de página de memoria física a disc

0x10013008

COGNOMS, NOM:

Problema 6. (1 punt)

Considera la declaració de les variables `u`, `w`, `x`, `y` guardades en `$t1`, `$t2`, `$t3`, `$t4` respectivament:

```
int u, w, x, y;
```

i el següent fragment de codi en llenguatge C:

```
if (u < w)
     y = (u >= x);
else
     y = (u != x);
```

L'hem traduït a assemblador MIPS sense fer servir macros. Completa els següents requadres amb els corresponents mnemònics i operands a fi que la traducció sigui correcta. Tingues en compte que, per la semàntica del llenguatge C, el resultat final de la variable *y* només pot ser 0 o 1.

```
     slt      [ $t0 ] , $t1, $t2
    [ beq  ]  $t0, $zero, else
if:
     slt      $t0, $t1, $t3
    [ sltiu ] $t4, $t0, [ 1 ]
     b        endif
else:
    [ subu ]  $t0, $t1, $t3
     sltu     $t4, [ $zero ] , $t0
endif:
```

COGNOMS, NOM:

Problema 7. (1 punt)

Disposem d'un processador de 32 bits (tant per adreces com per a dades) amb memòria principal adreçable a nivell de byte i dues memòries cache associatives per conjunts, una per a instruccions (MCI) i una altra per a dades (MCD), amb els següents paràmetres:

- Capacitat de la cache de dades: **512 KB**
- Capacitat de la cache d'instruccions: **1024 KB**
- Blocs per conjunt, en totes dues caches: **4 blocs**
- Mida de bloc: **32 paraules**
- Freqüència del rellotge: **200 MHz**
- Temps d'accés a memòria cache, per totes dues caches, en cas d'encert: **2 cicles**
- Temps d'accés a memòria principal per llegir/escriure blocs: **18 cicles**
- Temps d'accés a memòria principal per llegir/escriure paraules: **10 cicles**
- Memòria cache de dades amb una política **d'escriptura retardada amb assignació**.

Quan s'executen un conjunt de programes representatius (benchmark) observem que:

- CPI_{ideal}: **2,5**
- Taxa d'encerts en memòria cache d'instruccions**: 95%**
- Taxa d'encerts en memòria cache de dades: **70%**
- De cada 3 blocs reemplaçats a memòria cache de dades se n'ha modificat 1
- En mitjana, les instruccions que s'executen són
  - load: 10%
  - store: 5%
  - moviment de dades (immediats i registres): 20%
  - aritmètico-lògiques: 35%
  - salt: 20%
  - comparació: 10%

a) Quants bits ocuparà l'etiqueta (tag) dels blocs emmagatzemats a memòria cache de dades?

15

b) En mitjana, quants cicles per instrucció de penalització són motivats per les fallades a la memòria cache d'instruccions? (Indiqueu el nombre arrodonit a dècimes)

1,0

c) En mitjana, quants cicles per instrucció de penalització són motivats per les fallades a la memòria cache de dades? (Indiqueu el nombre arrodonit a dècimes)

1,2

d) Quin és el CPI promig d'aquest sistema processador-memòria obtingut en executar el benchmark? (Indiqueu el nombre arrodonit a dècimes)

3,7

COGNOMS, NOM:

Problema 8. (1 punt)

Considera la següent declaració MIPS de variables globals:

```
a:   .word 0xc4000000
b:   .word 0x43800000
c:   .float 128.0
```

Suposant que s'executa el següent codi:

```
la   $t0, a
lwc1 $f0, 0($t0)
la   $t0, b
lwc1 $f2, 0($t0)
la   $t0, c
lwc1 $f4, 0($t0)
```

Es demana que contesteu quin serà el valor final a `$f6` en hexadecimal després de l'execució dels següents codis:

a)

```
add.s $f6, $f4, $f4
add.s $f6, $f6, $f2
add.s $f6, $f6, $f0
```

$f6 => **0x00000000**

b)

```
add.s $f6, $f2, $f2
sub.s $f6, $f6, $f0
```

$f6 => **0x44800000**

COGNOMS, NOM:

Problema 9. (1 punt)

Donades les següents declaracions de variables globals en assemblador del MIPS:

```
.data                       # adreça base = 0x10010000
a:  .half -1, -7, 8
b:  .word a
.align 3
x:  .space 4
d:  .asciiz "abcde"         # codi ascii ‘a’ = 0x61
e:  .word 256
```

a) Ompliu la següent taula amb el contingut de la memòria, indicant el valor de cada byte EN HEXADECIMAL, i deixant EN BLANC les posicions no ocupades per cap dada.

@Memòria	Dada
0x10010000	**FF**
0x10010001	**FF**
0x10010002	**F9**
0x10010003	**FF**
0x10010004	**08**
0x10010005	**00**
0x10010006	
0x10010007	

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

@Memòria	Dada
0x10010010	**00**
0x10010011	**00**
0x10010012	**00**
0x10010013	**00**
0x10010014	**61**
0x10010015	**62**
0x10010016	**63**
0x10010017	**64**

@Memòria	Dada
0x10010018	**65**
0x10010019	**00**
0x1001001A	
0x1001001B	
0x1001001C	**00**
0x1001001D	**01**
0x1001001E	**00**
0x1001001F	**00**

b) Quin és el valor de `$t0` en hexadecimal després d’executar el següent codi?

```
la   $t0, b
lw   $t0, 0($t0)
lh   $t0, 2($t0)
```

$t0 => 0xFFFFFFF9

c) Quin és el valor final de `$t0` i de `$t1` en hexadecimal després d’executar el següent codi?

```
li   $t0, -7
li   $t1, 3
div  $t0, $t1
mflo $t0
mfhi $t1
```

$t0 => 0xFFFFFFFE
$t1 => 0xFFFFFFFF

d) Quin és el valor final de `$t0` i de `$t1` en hexadecimal després d’executar el següent codi?

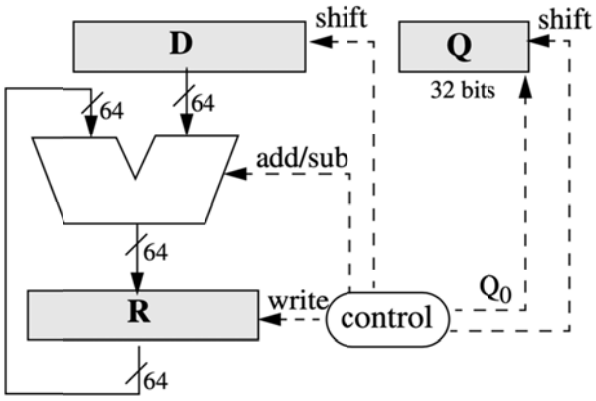
```
li   $t0, 1056
sra  $t1, $t0, 5
xor  $t0, $t0, $t1
subu $t0, $t0, $t1
```

$t0 => 0x000003E0
$t1 => 0x00000021

COGNOMS, NOM:

Problema 10. (1 punt)

Donat el següent diagrama que representa el divisor seqüencial de nombres naturals de 32 bits estudiat a classe i que realitza la divisió X/Y, calculant alhora el quocient i el residu, completeu l'algorisme iteratiu que en descriu el funcionament cicle a cicle:



$D_{63:32}$ = **Y** ;

$D_{31:0}$ = **0** ;

Q = **0** ;

$R_{63:32}$ = **0** ;

$R_{31:0}$ = **X** ;

```
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;
}
```

```
}
```

COGNOMS:	**GRUP:**
NOM:	

EXAMEN FINAL D'EC
9 de gener de 2020

L'examen consta de 10 preguntes, que s'han de contestar als mateixos fulls de l'enunciat. Posa el teu nom i cognoms a tots els fulls. La duració de l'examen és de 180 minuts. Les notes es publicaran al Racó el dia 20 de gener. La revisió es farà presencialment el 21 de gener de 11 a 12h.

Pregunta 1. (0,9 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 0,1 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.-	Per obtenir el resultat en n bits de la multiplicació de dos nombres enters negatius representats en Ca2 només cal fer el seu producte amb un multiplicador de nombres naturals, sense fer cap tractament de signes a priori ni a posteriori.	**X**	
2.-	Si es permetés que l'RSE pogués tractar excepcions llavors el sistema entraria en un bucle infinit en el moment en què es produís una excepció mentre s'executa l'RSE.		**X**
3.-	La divisió entera per una potència de 2 és sempre equivalent a un desplaçament aritmètic a la dreta de tants bits com indiqui l'exponent de la potència.		**X**
4.-	L'excepció per accés no alineat a memòria pot ser inhibida a través del camp Interrupt Mask.		**X**
5.-	Una excepció no pot ser atesa fins que la instrucció en curs hagi finalitzat.		**X**
6.-	Al format de coma flotant IEEE 754 de simple precisió, l'exponent 255 representa tant infinit, com menys infinit com el concepte NaN (Not a Number).	**X**	
7.-	En un sistema amb memòria virtual que té una mida de pàgina fixa, la mida de la taula de pàgines augmentarà tant si s'augmenta la mida de l'espai d'adreçament virtual com si s'augmenta la mida de l'espai d'adreçament físic.	**X**	
8.-	Per a una fallada de lectura, una cache d'escriptura immediata sense assignació té el mateix temps de penalització que una cache d'escriptura retardada amb assignació.		**X**
9.-	Un accés a memòria mai canviarà l'estat de la taula de pàgines si es produeix un encert al TLB a una entrada amb el bit de validesa a 1.		**X**

Pregunta 2. (1,1 punts)

Tenim la instrucció **`div.s $f0, $f2, $f4`**
Sabem que després d'executar la instrucció `$f0` = 0x40100001.
També sabem que `$f4` = 0x3FC00000.

Calcula un valor (en hexadecimal) de `$f2` que compleixi aquestes condicions.

`$f2` = **0x40580002 (també 0x40580001)**

Pregunta 3. (1 punt)

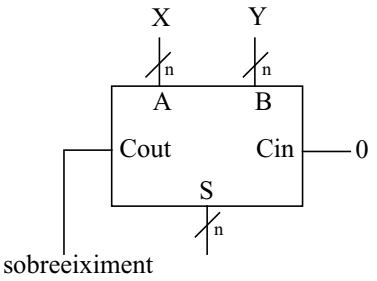
Per a cadascun dels apartats següents, on S, X i Y són nombres d'n bits, indica textualment i de forma concisa com es pot saber si la suma S = X+Y produeix sobreeiximent i dissenya, també, un circuit que ho implementi.

NOTA: Pots usar tots els blocs combinacionals i les portes que creguis convenient. En cas d'usar un sumador aritmètic, considera que les seves entrades són A i B d'n bits i Cin d'1 bit i té les sortides S d'n bits i Cout d'1 bit.

a) [0,5 punts] S, X i Y són nombres naturals

i) Hi ha transport en la suma aritmètica d'X i Y en n bits

Altres possibilitats:
ii) Sent S = X+Y, si S < X (o també si S < Y)
iii) Si X > $\overline{Y}$ (o també si Y>$\overline{X}$)



b) [0,5 punts] S, X i Y són nombres enters representats en Ca2

i) Es sumen dos nombres del mateix signe i el signe del resultat és oposat.

A partir del mateix circuit anterior caldria agafar els bits de més pes d'X, Y i S (X_{n-1}, Y_{n-1} i S_{n-1}, respectivament) i fer:

$$\text{sobreeiximent} = X_{n-1} * Y_{n-1} * \overline{S_{n-1}} + \overline{X_{n-1}} * \overline{Y_{n-1}} * S_{n-1}$$

Altres possibilitats (sent C_{n-1} el transport que hi ha d'entrada a la suma dels bits n-1):
ii) Cout * (X_{n-1} XOR Y_{n-1})
iii) Cout XOR C_{n-1}
iv) X_{n-1} XOR Y_{n-1} XOR C_{n-1}

COGNOMS:	GRUP:
NOM:	

Pregunta 4. (0,7 punts)

Donada la següent funció en C:

```
short func(short M[][1000], int i) {
       return M[i][i+3] + M[i+2][i];
}
```

Completa els requadres del següent fragment de codi en assemblador MIPS per tal que sigui la traducció correcta de la funció anterior:

```
unc:    li      $t0,  2002
        mult    $t0,  $a1
        mflo    $t0
        addu    $t0, $t0,  $a0
        lh      $t1,  6     ($t0)
        lh      $t2,  4000  ($t0)
        addu    $v0, $t1, $t2
        jr      $ra
```

Pregunta 5. (1,1 punts)

Un sistema computador treballa a una freqüència de rellotge de 1 GHz, un voltatge de 1 V i disposa d'una memòria cache (MC) que utilitza una política d'escriptura **immediata amb assignació**. Els temps representatius del sistema de memòria són $t_h = 1$ cicle i $t_{block} = 199$ cicles. En aquest computador s'executa un programa de $3 \cdot 10^9$ instruccions, les quals generen $2 \cdot 10^9$ referències a memòria. S'ha observat que la taxa d'encerts a MC és d'un 90%. El temps total d'execució del programa és 52 segons i la potència dissipada és de 2 W.

a) [0,5 punts] Determina quin seria el CPI_{ideal} d'aquest sistema computador.

CPI_{ideal} = **4**

b) [0,6 punts] Si la freqüència del computador s'incrementés a 2 GHz i el voltatge s'incrementés a 1.2 V, suposant que no ha canviat la capacitància (C) ni el factor d'activitat (α), ni el CPI_{ideal} ni cap paràmetre del sistema de memoria, calcula quin seria el temps d'execució del mateix programa i la potència dissipada.

t_{exe} = **26 s**

P = **5.76 W**

Pregunta 6. (1 punt)

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

```
char a = 'A';
float b = 1.5;
short c[3] = {-2, -1, 7};
float *d = &b;
long long e[100];
```

a) [0,3 punts] Tradueix-la al llenguatge assemblador del MIPS.

```
        .data
a:      .byte 'A'
b:      .float 1.5
c:      .half -2, -1, 7
d:      .word b
        .align 3
e:      .space 800
```

b) [0,3 punts] 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	**41**
0x10010001	
0x10010002	
0x10010003	
0x10010004	**00**
0x10010005	**00**
0x10010006	**C0**
0x10010007	**3F**

@Memòria	Dada
0x10010008	**FE**
0x10010009	**FF**
0x1001000A	**FF**
0x1001000B	**FF**
0x1001000C	**07**
0x1001000D	**00**
0x1001000E	
0x1001000F	

@Memòria	Dada
0x10010010	**04**
0x10010011	**00**
0x10010012	**01**
0x10010013	**10**
0x10010014	
0x10010015	
0x10010016	
0x10010017	

@Memòria	Dada
0x10010018	**00**
0x10010019	**00**
0x1001001A	**00**
0x1001001B	**00**
0x1001001C	**00**
0x1001001D	**00**
0x1001001E	**00**
0x1001001F	**00**

c) [0,4 punts] Donat el següent codi en assemblador MIPS, indica quin és el valor final en hexadecimal del registre `$t0`:

```
la      $t0, c
lh      $t1, 2($t0)
lh      $t2, 4($t0)
sll     $t2, $t2, 4
xor     $t0, $t1, $t2
```

$t0 = **0xFFFFFF8F**

COGNOMS:	GRUP:
NOM:	

Pregunta 7. (0,5 punts)

Donada la següent sentència escrita en alt nivell en C:

```
if ((x > y) && (y < 0))
    x++;
else if ((x <= 0) || (y <= 0))
    y++;
```

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

```
        ble     $t0, $t1,    etq4
etq1:   bge     $t1, $zero,  etq4
etq2:   addiu   $t0, $t0, 1
etq3:   b       etq7
etq4:   ble     $t0, $zero,  etq6
etq5:   bgt     $t1, $zero,  etq7
etq6:   addiu   $t1, $t1, 1
etq7:
```

Pregunta 8. (1,1 punts)

Donades les següents declaracions de funcions en C:

```
float g(int *v, char *b);

float f(float *v) {
        int w[10];
        char a;
        return g(&w[2], &a) + v[2];
}
```

Tradueix a MIPS la funció f:

```
f:  addiu  $sp, $sp, -52
    sw     $s0, 44($sp)
    sw     $ra, 48($sp)
    move   $s0, $a0
    addiu  $a0, $sp, 8
    addiu  $a1, $sp, 40
    jal    g
    lwc1   $f2, 8($s0)
    add.s  $f0, $f0, $f2
    lw     $s0, 44($sp)
    lw     $ra, 48($sp)
    addiu  $sp, $sp, 52
    jr     $ra
```

COGNOMS:	GRUP:
NOM:	

Pregunta 9. (1,3 punts)

Tenim un programa que, donada una matriu `M`, calcula la seva matriu trasposada `T`. El codi del programa és el següent:

```
int M[32][64], T[64][32];

for (int i=0; i<64; i++)
        for (int j=0; j<32; j++)
                T[i][j] = M[j][i];
```

Al compilar el programa, el compilador emmagatzema la matriu `M` a l'adreça 0x10010000 i la matriu `T` immediatament a continuació, mentre que les variables `i` i `j` s'emmagatzemen a registres del processador.

El programa s'executa a un processador de 32 bits amb una memòria cache amb les següents característiques:

- Capacitat de 64 blocs de 32 Bytes cadascun
- Completament associativa amb algorisme de reemplaçament LRU
- Escriptura retardada amb assignació

Es demana:

a) [0,8 punts] Quantes fallades de cache es produiran durant l'execució del programa?. Pots argumentar la resposta.

fallades = **512**

Els accessos a la matriu T causen 4 fallades a cada iteració del bucle 'i'. Els accessos a la matriu M causen 32 fallades cada 8 iteracions del bucle 'i'. En total tenim 4*64+32*64/8=512 fallades.

b) [0,5 punts] Quantes fallades de cache es produiran durant l'execució del programa si només es canvia la política d'escriptura de la cache a escriptura immediata sense assignació?

fallades = **2304**

Pregunta 10. (1,3 punts)

Tenim un processador amb memòria virtual basada en paginació amb les següents característiques:

- Adresses lògiques de 20 bits
- Pàgines de 16KB

A continuació es mostra l'estat inicial de la taula de pàgines d'un programa en execució. L'ordre temporal en que s'ha accedit a les pàgines virtuals és 2,4,1 (la pàgina 2 és la més antiga). :

VPN	P	D	PPN
0	0	0	
1	1	0	2
2	1	1	0
3	0	0	
4	1	0	1
5	0	0	
6	0	0	
7	0	0	
...			
63	0	0	

El programa pot tenir a memòria física un màxim de 4 pàgines simultàniament que es mapegen sempre als marcs de pàgina física 0, 1, 2 i 3. Tal com mostra la taula de pàgines, a l'estat inicial la memòria física té 3 pàgines virtuals carregades als marcs de pàgina física 0, 1 i 2, i el marc de pàgina física 3 està inicialment lliure. En cas que una pàgina s'hagi de sacrificar per deixar lloc a una altra, es fa servir l'algorisme de reemplaçament LRU.

A partir d'aquest estat inicial s'executen una seqüència de sis accessos a memòria. Emplena la següent taula indicant, per a cada accés a memòria, el seu número de pàgina virtual (VPN), si hi ha fallada de pàgina, si es llegeix la pàgina de disc, quina pàgina virtual es reemplaça, quina pàgina virtual s'escriu a disc (si cal), i quin és el marc de pàgina física (PPN) resultant de la traducció.

L/E	Adr. lògica (hex)	VPN (hex)	Fallada de pàgina? (SI/NO)	Lectura de disc (SI/NO)	VPN reempl. (hex)	VPN escrita a disc (hex)	PPN (hex)
L	0x1C124	**7**	**SI**	**SI**	-	-	**3**
E	0x12ACB	**4**	**NO**	**NO**	-	-	**1**
L	0x0D972	**3**	**SI**	**SI**	**2**	**2**	**0**
L	0x08448	**2**	**SI**	**SI**	**1**	-	**2**
E	0x1E666	**7**	**NO**	**NO**	-	-	**3**
L	0x07A34	**1**	**SI**	**SI**	**4**	**4**	**1**

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:
```

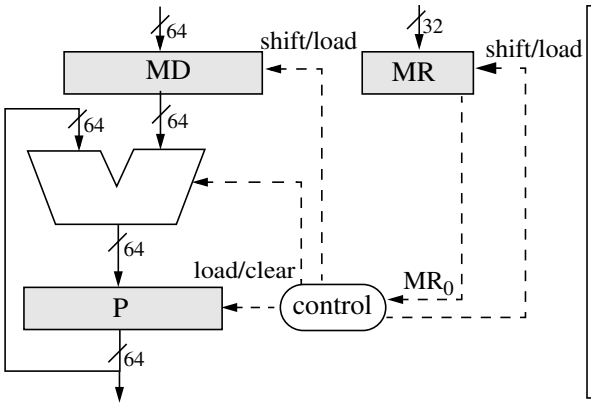
Completa les següents caselles per tal que reflecteixin un programa C equivalent a l'anterior. Heu de posar els tipus que falten de les variables locals (`int` o `unsigned int`) i les parts de la sentència condicional que falten (condició, codi del llavors i codi del sinó).

```
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:

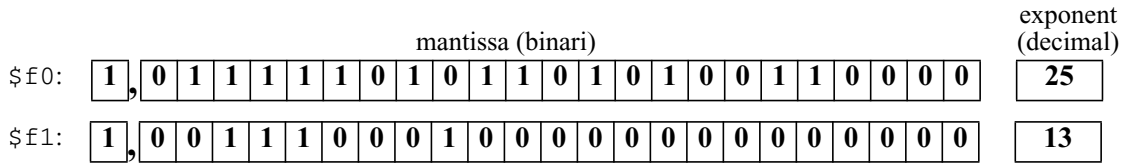
```
float res, i;
main() {
        res = 0.0;
        for (i = 1.0; i <= 10000.0; i = i + 1.0)
                res = res + i;
}
```

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:

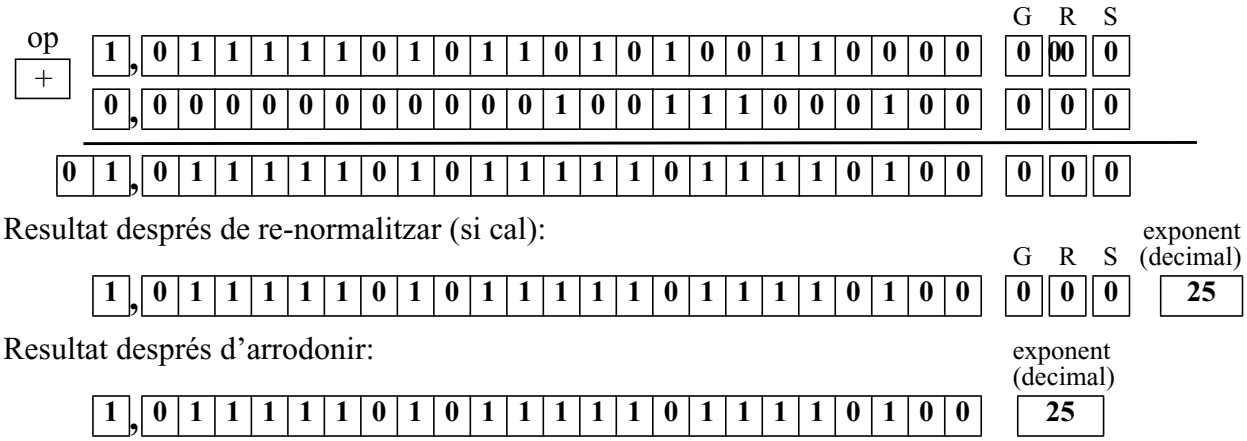
COGNOMS: **GRUP:**

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**

COGNOMS:	GRUP:
NOM:	

EXAMEN FINAL D'EC
10 de gener de 2019

L'examen consta de 9 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 abans del dia 17 de gener.

Pregunta 1. (1,6 punts)

Donades les següents declaracions de variables globals en assemblador del MIPS, que s'ubiquen en memòria a partir de l'adreça 0x10010000:

```
        .data
a:      .asciiz "abcd"                  # codi ASCII 'a' = 0x61
b:      .half  10, -7
c:      .word  b
d:      .byte  0x03
        .align 1
e:      .space 4
f:      .word  256
```

a) (0,4 pts) Omple la següent taula amb el contingut de la memòria, indicant el valor de cada byte EN HEXADECIMAL, i deixant EN BLANC les posicions no ocupades per cap dada.

@Memòria	Dada
0x10010000	61
0x10010001	62
0x10010002	63
0x10010003	64
0x10010004	00
0x10010005	
0x10010006	0A
0x10010007	00

@Memòria	Dada
0x10010008	F9
0x10010009	FF
0x1001000A	
0x1001000B	
0x1001000C	06
0x1001000D	00
0x1001000E	01
0x1001000F	10

@Memòria	Dada
0x10010010	03
0x10010011	
0x10010012	00
0x10010013	00
0x10010014	00
0x10010015	00
0x10010016	
0x10010017	

@Memòria	Dada
0x10010018	00
0x10010019	01
0x1001001A	00
0x1001001B	00
0x1001001C	
0x1001001D	
0x1001001E	
0x1001001F	

b) (0,4 pts) Quin és el valor de `$t0` en hexadecimal després d'executar el següent codi?

```
la      $t0, c
lw      $t0, 0($t0)
lh      $t0, 2($t0)
```

`$t0 =` 0xFFFFFFF9

c) (0,4 pts) Quin és el valor final de `$t0` i de `$t1` en hexadecimal després d'executar el següent codi?

```
li      $t0, 2563
li      $t1, 10
div     $t0, $t1
mflo    $t0
mfhi    $t1
```

`$t0 =` 0x00000100

`$t1 =` 0x00000003

d) (0,4 pts) Quin és el valor final de `$t0` en hexadecimal després d'executar el següent codi?

```
li      $t0, -3
sra     $t1, $t0, 31
xor     $t0, $t0, $t1
subu    $t0, $t0, $t1
sll     $t1, $t1, 31
or      $t0, $t0, $t1
```

`$t0 =` 0x80000003

Pregunta 2. (1,3 punts)

Considerem un computador amb un processador MIPS funcionant a una freqüència de 0,5Ghz, i que dissipa una potència de 10 W. Suposem que la cache d'instruccions és ideal (sempre encerta), i que la cache de dades té un temps de servei en cas d'encert t_h = 1 cicle. El temps necessari per copiar un bloc de memòria principal a cache és t_{block} = 99 cicles. Els CPI dels diversos tipus d'instruccions (en absència de fallades) són:

	Salts	Loads	Resta d'instruccions
CPI	3	5	2

A través de simulacions amb un programa de test hem mesurat una taxa de fallades de la cache de dades del 4,4% (és a dir, m = 0,044). Totes les referències a memòria són lectures. El nombre d'instruccions executades és:

	Salts	Loads	Resta d'instruccions
núm. instr.	$1 \cdot 10^9$	$3 \cdot 10^9$	$6 \cdot 10^9$

a) (0,4 pts) Calcula el CPI_{ideal} del programa (CPI promig amb cache ideal sense fallades)

CPI_{ideal} = 3

b) (0,4 pts) Calcula, en segons, el temps d'execució (incloent-hi fallades de cache)

t_{exe} = 86,4 s

c) (0,1 pts) Calcula l'energia total consumida durant l'execució del programa, en Joules

E = 864 J

d) (0,4 pts) Calcula, en cicles, el temps d'accés mitjà a memòria dels loads per a aquest programa

t_{am} = 5,4 cicles

COGNOMS:	**GRUP:**
NOM:	

Pregunta 3. (1,4 punts)

Donades les següents declaracions en C:

```
int *pglob;                              /* variable global  */
int f2(int x, char *y, char *z);         /* prototipus de f2 */
char f1(int a, char b[][5], int c, int *d) {
        char v[10];
        if ((c < 0)||(c >= a))
                c = 0;
        *pglob = f2(*d, &b[3][0], v);
        return v[c];
}
```

A continuació es mostra una traducció de la funció `f1` a llenguatge MIPS que està incompleta. Llegiu-la amb atenció, i completeu les caixes per tal que la traducció sigui correcta.

```
f1:     addiu   $sp, $sp, -20
        sw      $s0, 12($sp)
        sw      $ra, 16($sp)
        blt     $a2, $zero, then        # c<0?
        blt     $a2, $a0, endif         # c>=a?
then:
        move    $a2, $zero
endif:
        move    $s0, $a2                # copiar en registre segur
        # Passar paràmetres: f2(*d, &b[3][0], v)
        lw      $a0, 0($a3)
        addiu   $a1, $a1, 15
        move    $a2, $sp
        jal f2
        # Emmagatzemar resultat: *pglob = f2(...)
        la      $t0, pglob
        lw      $t1, 0($t0)
        sw      $v0, 0($t1)
        # Sentència final: return v[c];
        addu    $t0, $sp, $s0
        lb      $v0, 0($t0)
        lw      $s0, 12($sp)
        lw      $ra, 16($sp)
        addiu   $sp, $sp, 20
        jr      $ra
```

Pregunta 4. (1,2 punts)

Considera la següent declaració MIPS de variables globals:

```
a:      .word  0xCC800000
b:      .word  0x4C800000
c:      .float 1.0
```

Suposant que s'executa el següent codi:

```
la      $t0, a
lwc1    $f0, 0($t0)
la      $t0, b
lwc1    $f2, 0($t0)
la      $t0, c
lwc1    $f4, 0($t0)
```

Es demana que contesteu quin serà el valor final a `$f6` en hexadecimal després de l'execució dels següent codis:

a) (0,6 pts)

```
add.s   $f6, $f0, $f2
add.s   $f6, $f6, $f4
```

$f6 = 0x3f800000

b) (0,6 pts)

```
add.s   $f6, $f2, $f4
add.s   $f6, $f0, $f6
```

$f6 = 0x00000000

COGNOMS:	GRUP:
NOM:	

Pregunta 5. (0,8 punts)

Considerant la declaració de la matriu global A

```
int A[N][N];
```

el següent codi en llenguatge C copia la triangular superior cap a la inferior d'aquesta matriu quadrada:

```
int i,j;

for (i=0; i<N; i++)
   for (j=i+1; j<N; j++)
       A[j][i] = A[i][j];
```

A continuació tenim un codi incomplert també en llenguatge C que és una optimització del codi anterior:

```
int *pdi,*pei,*pdj,*pej;

pdi = &A[0][1];
pei = &A[1][0];
do {
       pdj = pdi;
       pej = pei;
       do {
              *pej = *pdj;

              pdj += [ 1 ] ;

              pej += [ N ] ;

       } while (pej <= &A[N-1][N-1]);

       pdi += [ N+1 ] ;

       pei += [ N+1 ] ;

} while (pdi <= &A[N-1][N-1]);
```

S'hi ha aplicat l'optimització de convertir un bucle `while` en un `do_while`, d'eliminar variables d'inducció i de fer accés seqüencial tant per l'accés a `A[i][j]` com per a `A[j][i]`.

Es demana que completeu el codi anterior.

Pregunta 6. (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 0,1 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.-	En un programa escrit en assemblador MIPS, que consta de 2 mòduls A i B que es compilen per separat, i on el mòdul A invoca una funció `func` que està declarada al mòdul B, l'assemblatge del mòdul A fallarà sense generar cap fitxer objecte.		X
2.-	Si l'accés a dades d'una instrucció produeix un encert al TLB, però el bit V val 0, llavors la instrucció causarà una excepció de fallada de pàgina.	X	
3.-	Una mateixa instrucció pot causar durant la seva execució 2 fallades de pàgina.	X	
4.-	La instrucció `tlbwr` escriu una entrada de la taula de pàgines a l'entrada del TLB que resulta d'aplicar un algorisme de reemplaçament RANDOM.	X	
5.-	Si el bit EXL (Exception Level) del registre Status val 1, el processador està en mode sistema i totes les excepcions resten inhibides.		X

COGNOMS:	GRUP:
NOM:	

Pregunta 7. (1,2 punts)

La memòria virtual implementada en un sistema computador de 32 bits es caracteritza pels següents paràmetres:

- Pàgines de 4 KB de mida.
- Un màxim de 5 pàgines carregades simultàniament a memòria física per aplicació.
- Reemplaçament de pàgines a memòria física seguint l'algorisme LRU.
- TLB totalment associatiu de 8 entrades amb reemplaçament LRU.

Donat el següent codi en C:

```
int V[8192];

main() {
        int i;
        int sum = 0;

        for (i=0; i < 8192; i++) {
                sum += V[i] + V[8191 - i];
        }
}
```

Considera que les variables locals *i* i *sum* s'enmagatzemen en registres, que el vector global V s'emmagatzema a memòria a partir de l'adreça `0x00000000`, i que el codi s'emmagatzema a partir de l'adreça `0x0000C000`, i ocupa menys d'una pàgina. El TLB i la memòria física estan inicialment buits. Es demana:

a) (0,3 pts) Quantes pàgines ocupa el vector `V`?

`nombre de pàgines` = **8**

b) (0,3 pts) Quantes fallades de TLB (codi i dades) es produiran en tota l'execució del programa?

fallades de TLB = **10**

c) (0,3 pts) Quantes fallades de pàgina (codi i dades) es produiran en tota l'execució del programa?

fallades de pàgina = **13**

d) (0,3 pts) Indica els VPN (en hexadecimal) de les cinc pàgines (codi i/o dades) que hi haurà carregades a memòria física quan s'acabi l'execució d'aquest programa.

VPNs = **0x0, 0x1, 0x6, 0x7, 0xC**

Pregunta 8. (1,0 punts)

a) (0,4 pts) Escriu 4 formes equivalents d'expandir la macro `li $t0,1` usant únicament 1 instrucció.

```
addiu  $t0,$zero,1
```

```
ori    $t0,$zero,1
```

```
xori   $t0,$zero,1
```

```
sltiu  $t0,$zero,1
```

b) (0,3 pts) Escriu en decimal i en notació científica normalitzada de base 2 el menor número real positiu tal que en l'estàndard IEEE-754 de simple precisió es codifiqui com un denormal.

$1{,}0 \cdot 2^{-149}$

c) (0,3 pts) Donat el següent codi en C que usa la variable `a` de tipus `int`:

```
a = (((a & 1) && 1) || 1) | 1;
```

i considerant que la variable `a` està emmagatzemada al registre `$t0`, escriu un codi MIPS equivalent amb 2 línies com a màxim.

```
li   $t0, 1
```

Pregunta 9. (1,0 punts)

Un sistema disposa d'un processador MIPS (32 bits d'adreces i mida de paraula de 4 bytes), i una memòria cache (MC) de 64 Kbytes amb la següent organització:

- Correspondència directa
- Blocs de 256 bytes
- Escriptura immediata sense assignació

Estant la cache inicialment buida, un programa fa una seqüència de referències a memòria segons s'indica a la següent taula, on apareixen les adreces en hexadecimal i si són lectures o escriptures (L/E). Completa les columnes que falten indicant, per a cada referència: l'índex de MC; si és encert (e) o fallada (f); i el nombre de bytes de Memòria Principal (MP) llegits i/o escrits.

L/E	adreça (hex)	índex MC	encert (e)/ fallada (f)	bytes de MP	
				llegits	escrits
E	00010000	**0**	**f**	**-**	**4**
L	00010008	**0**	**f**	**256**	**-**
E	00010016	**0**	**e**	**-**	**4**
L	00F40316	**3**	**f**	**256**	**-**
E	03100004	**0**	**f**	**-**	**4**
L	00010024	**0**	**e**	**-**	**-**
L	03200308	**3**	**f**	**256**	**-**
L	00F403A8	**3**	**f**	**256**	**-**