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