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**.

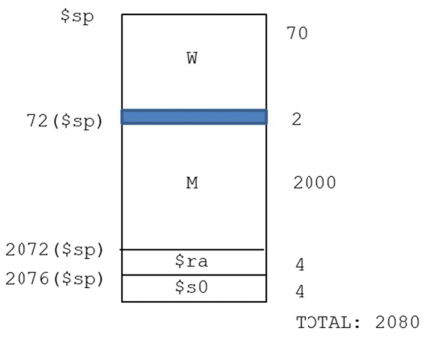
`… = VG[5];`	**.data**
`… = pL;`	**registre**
`… = *(pD+20);`	**heap**
`… = VL[6];`	**pila**
`… = pD;`	**registre**
`… = sum;`	**.data**
`… = pG;`	**.data**

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)
```

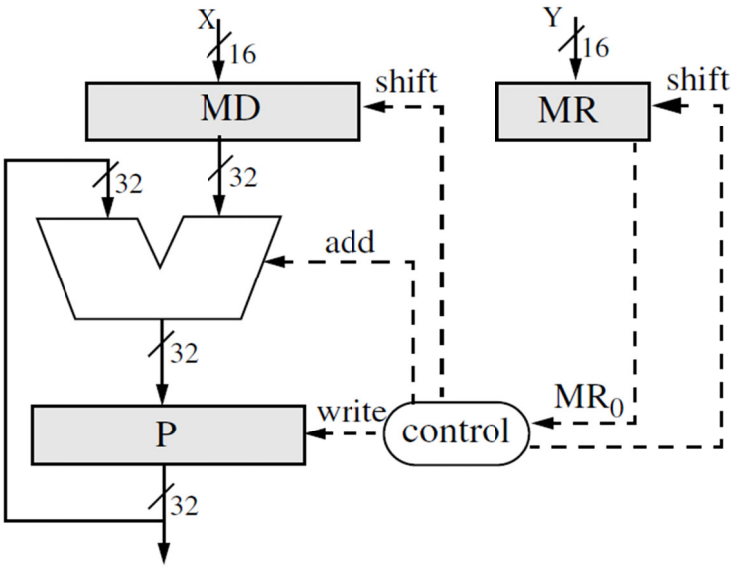
Tradueix a assemblador MIPS la sentència C, incloent-hi l'epíleg de la funció `Examen` (fins al `jr $ra`).

```
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).*