COGNOMS:	**GRUP:**
NOM:	

EXAMEN PARCIAL D'EC
7 de novembre de 2019

L'examen consta de **7** 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 120 minuts. Les notes, la solució i el procediment de revisió es publicaran al Racó el dia **18** de novembre.

Pregunta 1. (1,70 punts)

Considera les següents declaracions en C:

```
int G[64][64];
void f(int i, int j, int M[][64]) {
        while (j<64) {
                M[j][i] = G[0][63-j];
                j++;
        }
}
```

El següent fragment de codi conté la traducció de la funció f, aplicant-li l'optimització d'*accés seqüencial*. En concret, s'han usat els registres `$t0` i `$t1` com a punters per a recórrer els accessos a `G[0][63-j]`, i `M[j][i]`, respectivament. Completa les instruccions que falten als requadres per tal que la traducció sigui correcta:

```
f:      # Inicialitza el punter $t0 <- @G[0][63-j]
        la      $t0, G + 252
        sll     $t4, $a1, 2
        subu    $t0, $t0, $t4
        # Inicialitza el punter $t1 <- @M[j][i]
        sll     $t1, $a1, 8     # j*64*4
        sll     $t2, $a0, 2     # i*4
        addu    $t1, $t1, $t2
        addu    $t1, $t1, $a2
        li      $t2, 64
while:
        bge     $a1, $t2, fi
        lw      $t4, 0($t0)
        sw      $t4, 0($t1)
        addiu   $t0, $t0, -4
        addiu   $t1, $t1, 256
        addiu   $a1, $a1, 1
        b       while
        jr      $ra
fi:
```

Pregunta 2. (1,50 punts)

Considera el següent prototip de la funció crc32, que calcula el Codi de Redundància Cíclica de 32 bits per a un missatge `M` compost de `N` bytes, fent servir una taula auxiliar `LUT` de 256 words.

```
unsigned int crc32(unsigned char M[], int N, unsigned int LUT[]);
```

El codi corresponent en assemblador MIPS és el següent:

```
crc32:
        nor     $v0, $zero, $zero
for:
        beq     $a1, $zero, fifor   # surt si N==0
        lbu     $t2, 0($a0)         # carrega un element de M
        andi    $t3, $v0, 0xFF
        xor     $t3, $t3, $t2
        sll     $t3, $t3, 2
        addu    $t4, $a2, $t3
        lw      $t5, 0($t4)         # carrega un element de LUT
        srl     $v0, $v0, 8
        xor     $v0, $v0, $t5
        addiu   $a0, $a0, 1
        addiu   $a1, $a1, -1        # N = N-1
        b       for
fifor:
        nor     $v0, $v0, $zero
        jr      $ra
```

Suposem que el missatge `M` consta de `N`=100 bytes, i que la funció `crc32` s'executa en un processador que dissipa 100W de potència, que funciona amb un rellotge de 2GHz i que té els següents CPI segons el tipus d'instrucció:

TIPUS	salt (salta)	salt (no salta)	load/store	altres
CPI	5	1	8	1

a) (0,6 pts) Calcula quantes instruccions de cada tipus s'executen en la funció crc32, i quants cicles tarden. Calcula també el total d'instruccions i el total de cicles.

TIPUS	salt (salta)	salt (no salta)	load/store	altres	TOTAL
Núm. d'instruccions	102	100	200	802	1204
Núm. de cicles	510	100	1600	802	3012

b) (0,6 pts) Calcula el temps d'execució de crc32 en segons

t_{exe} = $1{,}506 \cdot 10^{-6}$ s

c) (0,3 pts) Calcula l'energia total consumida durant l'execució de `crc32`, en Joules

E = $1{,}506 \cdot 10^{-4}$ J

COGNOMS:	**GRUP:**
NOM:	

Pregunta 3. (1,20 punts)

Considera el següent prototip de funció en C:

```
int overflow(int a, int b);
```

a) (0,4 pts) Un cop programada la funció en MIPS, la primera instrucció és: "`mult $a0,$a1`" (multiplicació d'enters). Explica de la manera més concisa possible quina condició han de complir els registres resultants `$hi` i `$lo`, per afirmar que el producte `a*b` no és representable amb 32 bits (és a dir, que causa *overflow*).

`$hi` conté algun bit diferent del bit de signe de `$lo`

b) (0,4 pts) Completa el següent codi MIPS de la funció `overflow`, amb les 3 instruccions que falten, de manera que el resultat sigui un 1 en cas d'overflow, o bé un 0 altrament.

```
overflow:  mult  $a0, $a1
           mfhi  $t1
           mflo  $t0
```

```
sra   $t0, $t0, 31
subu  $t0, $t0, $t1
sltu  $v0, $zero, $t0
```

```
           jr    $ra
```

c) (0,4 pts) Suposant que `$t0`=0x80000000 i `$t1`=0xFFFFFFFF, calcula (en hexadecimal) el resultat en `$hi` i `$lo` després d'executar la instrucció "`multu $t0,$t1`" (multiplicació de naturals), i digues si s'ha produït overflow (no representable amb 32 bits), i en què es coneix.

$hi = 0x 7FFFFFFF $lo = 0x 80000000 overflow (Si/No) : SI

en què es coneix? `$hi` no és zero

Pregunta 4. (0,70 punts)

Completa la traducció a assemblador del MIPS de la funció g:

```
int g(short *q, short val) {
        return (*q == val);     /* Retorna 1 si són iguals, 0 altrament */
}
```

```
g:
        lh      $t0, 0($a0)
        subu    $v0, $t0, $a1
        sltiu   $v0, $v0, 1

        jr      $ra
```

Pregunta 5. (1,80 punts)

Donades les següents declaracions de funcions en C:

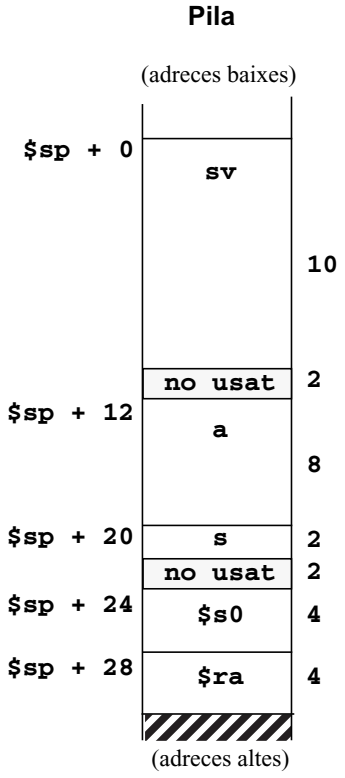
```
int f1(short *ps1, short *ps2,
       int *pi);

int f2(short *x, int y) {
       short sv[5];
       int a[2], res;
       short s;
       s = *x + 3;
       res = y + f1(&sv[2], &s, a);
       return res;
}
```

Contesta els següents apartats que hi fan referència:

a) (0,5 pts) Completa la taula següent indicant on s'han d'emmagatzemar cadascun dels elements de f2: a la pila, a un registre temporal o a un registre segur. Posa una X a la columna corresponent, només pots triar una opció per a cada element de f2.

element de `f2 (en C)`	pila	registre temporal	registre segur
`sv`	**X**		
`a`	**X**		
`res`		**X**	
`s`	**X**		
`x`		**X**	
`y`			**X**

b) (0,5 pts) Dibuixa el bloc d'activació de `f2`, especificant-hi la posició on apunta el registre `$sp` un cop reservat l'espai corresponent a la pila, així com el nom de cada registre i/o variable, i la seva posició (desplaçament relatiu al `$sp`).



c) (0,8 pts) Tradueix a assemblador de MIPS la següent sentència del cos de la subrutina `f2`:

```
res = y + f1(&sv[2], &s, a);
```

```
addiu   $a0, $sp, 4         # &sv[2]
addiu   $a1, $sp, 20        # &s
addiu   $a2, $sp, 12        # a
jal     f1
addu    $v0, $s0, $v0       # res = y + f1(...)
```

COGNOMS:	GRUP:
NOM:	

Pregunta 6. (2,20 punts)

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

```
short a[3] = {1, 31, -2};
long long int b = -31
char c[5] = "ACDC";
short *d = &a[2];
```

a) (0,5 pts) Tradueix-la al llenguatge assemblador del MIPS.

```
        .data
a:      .half   1, 31, -2
b:      .dword  -31
c:      .asciiz "ACDC"
d:      .word   a+4
```

b) (0,5 pts) Completa la següent taula amb el contingut de memòria en hexadecimal (sense el prefix 0x). Recorda 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	**01**
0x10010001	**00**
0x10010002	**1F**
0x10010003	**00**
0x10010004	**FE**
0x10010005	**FF**
0x10010006	
0x10010007	

@Memòria	Dada
0x10010008	**E1**
0x10010009	**FF**
0x1001000A	**FF**
0x1001000B	**FF**
0x1001000C	**FF**
0x1001000D	**FF**
0x1001000E	**FF**
0x1001000F	**FF**

@Memòria	Dada
0x10010010	**41**
0x10010011	**43**
0x10010012	**44**
0x10010013	**43**
0x10010014	**00**
0x10010015	
0x10010016	
0x10010017	

@Memòria	Dada
0x10010018	**04**
0x10010019	**00**
0x1001001A	**01**
0x1001001B	**10**
0x1001001C	
0x1001001D	
0x1001001E	
0x1001001F	

c) (0,6 pts) Donat el següent codi en assemblador MIPS, indica quin és el valor final en hexadecimal del registre `$t0`:

```
lui     $t0, 0x1001
addiu   $t0, $t0, 8
la      $t1, d
lw      $t1, 0($t1)
subu    $t0, $t0, $t1
```

`$t0 =` 0x **00000004**

d) (0,6 pts) Tradueix a llenguatge assemblador del MIPS la següent sentència en C:

```
*(d - 2) = *d + 5;
```

```
la      $t0, d
lw      $t1, 0($t0)         # d
lh      $t2, 0($t1)         # *d
addiu   $t2, $t2, 5         # *d + 5
sh      $t2, -4($t1)
```

Pregunta 7. (0,90 punts)

Donada la següent sentència escrita en alt nivell en C:

```
if ((a ^ 0xFFFF) && ((~b) <= a))
        z = a + b;
else
        z = 0;
```

Completa el següent fragment de codi MIPS, que tradueix l'anterior sentència, escrivint en cada calaix un mnemònic d'instrucció o macro, etiqueta, registre o immediat. Les variables `a`, `b` i `z` són de tipus `int` i estan inicialitzades i guardades als registres `$t0`, `$t1` i `$t2`, respectivament.

```
        xori    $t3, $t0, 0xFFFF
etq1:   beq     $t3, $zero, etq6
etq2:   nor     $t4, $t1, $zero
etq3:   bgt     $t4, $t0,  etq6
etq4:   addu    $t2, $t0, $t1
etq5:   b       etq7
etq6:   xor     $t2, $t2, $t2
etq7:
```