COGNOMS:	**GRUP:**
NOM:	

EXAMEN PARCIAL D'EC
10 de maig de 2018

L'examen consta de **8** 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 **21** de maig.

Pregunta 1. (1,8 punts)

Donada la següent funció `foo` en llenguatge C:

```
void foo(short M[][64], unsigned int k) {
        int i;
        short aux = M[k][8];
        for (i=5; i<45; i+=5){
                M[45-i][i] = aux;
        }
}
```

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 M s'accedeixen utilitzant la tècnica **d'accés seqüencial** sempre que es pot, usant el registre `$t1` com a punter. Aquest punter `$t1` s'inicialitza amb l'adreça de l'element `M[40][5]`. Al codi se li ha aplicat l'optimització de conversió d'un bucle `for` en un `do_while` i l'eliminació de la variable d'inducció.

```
        sll     $t0, $a1, 7
        addu    $t0, $t0, $a0
        lh      $t2, 16($t0)          # aux = M[k][8];
        addiu   $t1, $a0, 5130        # @M[40][5]
        addiu   $t3, $a0, 90          # adreça final del punter $t1
        b       cond
bucle:  sh      $t2, 0($t1)
        addiu   $t1, $t1, -630
cond:   bgtu    $t1, $t3, bucle
        jr      $ra
```

Pregunta 2. (1,8 punts)

Donades les següents declaracions en C:

```
unsigned int func2(int x, char M[][4], int *y);
unsigned int *pglob;
int func1(int par1, char par2[][4], int par3, int *par4) {
        int loc1;
        loc1 = par1 + par3;
        *pglob = func2 (*par4, &par2[3][0], par4);
        return *par4 - loc1;
}
```

A continuació es mostra una traducció de la funció `func1` a llenguatge MIPS que està incompleta. Llegiu-la amb atenció, a fi de contestar correctament a les preguntes de més avall.

```
func1:
        addiu   $sp, $sp, -12
        sw      $ra, 8($sp)
        sw      $s0, 4($sp)
        sw      $s1, 0($sp)
        move    $s1, $a3                        # copia del punter par4

        # loc1 = par1 + par3
        addu    $s0, $a0, $a2

        # *pglob = func2 (*par4, &par2[3][0], par4);

          # CAIXA 1
          # (pas de paràmetres a func2)

        jal func2

          # CAIXA 2
          # (emmagatzema resultat de func2)

        # return *par4 - loc1;

          # CAIXA 3
          # (sentència return)

        lw      $ra, 8($sp)
        lw      $s0, 4($sp)
        lw      $s1, 0($sp)
        addiu   $sp, $sp, 12
        jr      $ra
```

a) Completa la CAIXA 1, en llenguatge assemblador de MIPS, amb el pas de paràmetres per a la crida a la funció `func2` corresponent a la següent sentència del cos de la subrutina `func1`.

```
        ... = func2 (*par4, &par2[3][0], par4);
```

```
lw      $a0, 0($a3)
addiu   $a1, $a1, 12
move    $a2, $a3
```

```
jal   func2
```

COGNOMS:	GRUP:
NOM:	

b) Completa la CAIXA 2, en llenguatge assemblador de MIPS, amb la recollida del resultat de la crida i el seu emmagatzematge a l'adreça apuntada per la variable `pglob`, corresponent al següent fragment de sentència del cos de la subrutina `func1`: `*pglob = func2( ...`

```
jal    func2
la     $t0, pglob
lw     $t1, 0($t0)
sw     $v0, 0($t1)
```

c) Completa la CAIXA 3, en llenguatge assemblador de MIPS, amb la traducció de la darrera sentència: `return *par4 - loc1;`

```
lw     $v0, 0($s1)
subu   $v0, $v0, $s0
```

Pregunta 3. (0,8 punts)

Donada la següent sentència escrita en alt nivell en C:

```
if (((a%2==0)||(b!=0)) && ((b<=a)||(a>0)))
        z=a;
else
        z=b;
```

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.

```
        andi    $t3, $t0,    1
etq1:   beq     $t3, $zero,  etq3
etq2:   beq     $t1, $zero,  etq6
etq3:   ble     $t1, $t0,    etq5
etq4:   ble     $t0, $zero,  etq6

etq5:   move    $t2, $t0
        b       etq7

etq6:   move    $t2, $t1

etq7:
```

Pregunta 4. (1 punt)

Suposem un programa que s'executa sobre un processador funcionant a una freqüència de 2GHz, el qual dissipa una potència de 40W. 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:

Tipus d'instr.	Nombre. d'instr.	CPI
Memòria	$1{,}5 \times 10^9$	3
Salts	$1{,}0 \times 10^9$	4
Resta	$2{,}5 \times 10^9$	1

a) Calcula el CPI promig de tot el programa

CPI = **2,2**

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

t_{exe} = **5,5** s

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

E = **220** J

d) Suposem una nova versió del programa en què reduïm el nombre de Salts a la meitat, però és a costa d'augmentar el seu CPI de 4 a 6. Quin guany de rendiment (speed-up) s'ha produït?

guany = **1,1**

COGNOMS:	GRUP:
NOM:	

Pregunta 5. (2 punts)

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

```
char a[5]    = "DADA";
char *b      = &a[2];
long long c = -4;
char d[2]    = {14,16};
unsigned int e[100];
```

a) Tradueix-la al llenguatge assemblador del MIPS

```
            .data
a:          .asciiz  "DADA"
b:          .word    a+2
c:          .dword   -4
d:          .byte    14, 16
            .align   2
e:          .space 400
```

b) Completa la següent taula amb el contingut de les 48 posicions de memòria representades, en hexadecimal (sense el prefix "0x"). Les variables globals s'emmagatzemen a partir de l'adreça 0x10010000. Recorda que el codi ASCII de la 'A' és el 0x41, i que les variables globals no inicialitzades valen 0x00. Les posicions de memòria no ocupades es deixen en blanc.

@Memòria	Dada
0x10010000	44
0x10010001	41
0x10010002	44
0x10010003	41
0x10010004	00
0x10010005	
0x10010006	
0x10010007	
0x10010008	02
0x10010009	00
0x1001000A	01
0x1001000B	10
0x1001000C	
0x1001000D	
0x1001000E	
0x1001000F	

@Memòria	Dada
0x10010010	FC
0x10010011	FF
0x10010012	FF
0x10010013	FF
0x10010014	FF
0x10010015	FF
0x10010016	FF
0x10010017	FF
0x10010018	0E
0x10010019	10
0x1001001A	
0x1001001B	
0x1001001C	00
0x1001001D	00
0x1001001E	00
0x1001001F	00

@Memòria	Dada
0x10010020	00
0x10010021	00
0x10010022	00
0x10010023	00
0x10010024	00
0x10010025	00
0x10010026	00
0x10010027	00
0x10010028	00
0x10010029	00
0x1001002A	00
0x1001002B	00
0x1001002C	00
0x1001002D	00
0x1001002E	00
0x1001002F	00

c) Donat el següent codi en MIPS, indica quin és el valor final en hexadecimal del registre `$t1`:

```
la      $t1, d+1
lb      $t1, 0($t1)
lui     $t2, 0x1001
or      $t1, $t2, $t1
lhu     $t1, 0($t1)
```

$t1 = 0x**0000FFFC**

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

```
*b = *(b-2);
```

```
la      $t0, b
lw      $t1, 0($t0)
lb      $t2, -2($t1)
sb      $t2, 0($t1)
```

COGNOMS:	**GRUP:**
NOM:	

Pregunta 6. (1,2 punts)

Suposem que definim un format de coma flotant de 16 bits, similar a l'estàndard de simple precisió, excepte que té 7 bits de fracció en comptes de 23. La resta de camps es codifiquen igual.

a) Codifica el número x = -27,16 en el nou format de 16 bits aplicant l'arrodoniment al més pròxim, i expressa el resultat en hexadecimal:

x = 0x **C1D9**

b) Calcula l'error de precisió comès en l'anterior apartat, expressant-lo en base 10, amb 2 dígits significatius (per exemple: error = 0,000XX, o bé: 0,0XX, etc.):

error = **0,035**

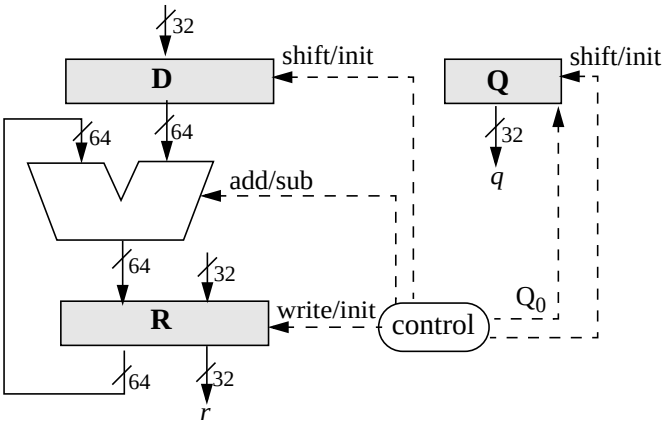
Pregunta 7. (0,5 punts)

Les següents afirmacions fan referència al format de simple precisió IEEE-754 (32 bits). Posa una X per a cada una d'elles (a la columna V si és Verdadera o a la columna F si és Falsa). 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.-	Es produeix *underflow* quan els bits fraccionaris de la mantissa són tots zeros		**X**
2.-	Es produeix *overflow* quan l'exponent del resultat d'una operació és major que +126		**X**
3.-	La codificació 0x7FFFFFFF representa el major número positiu no-nul		**X**
4.-	La codificació 0x00800000 representa un número normalitzat	**X**	
5.-	La codificació 0x00000001 representa un número denormal	**X**	

Pregunta 8. (0,9 punts)

Sigui el següent circuit seqüencial per a la divisió de números naturals de 32 bits, que calcula el quocient $q = x/y$, i el residu $r = x\%y$ (els senyals d'entrada x, i y s'hi han omès expressament):



Completa el següent algorisme en pseudocodi, que expressa la seqüència d'operacions que realitza l'anterior circuit divisor:

```
R63:32 =  0;                          # part alta de R
R31:0  =  x;                          # part baixa de R
D63:32 =  y;                          # part alta de D
D31:0  =  0;                          # part baixa de D
Q      =  0;
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;
        }
}
q      =  Q;                          # quocient
r      =  R31:0;                      # residu
```