Parcial EC

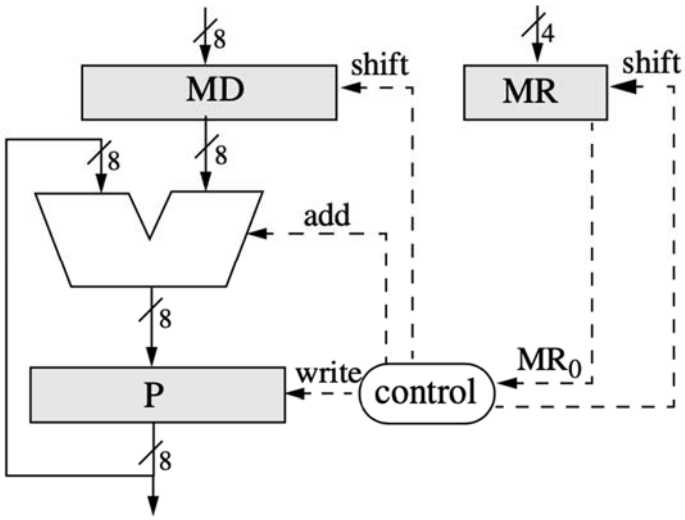
Cognoms: ..... Nom: .....
DNI: .....

EXAMEN PARCIAL D'EC
3 de novembre de 2022

- L'examen consta de 5 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 1:30 hores (90 minuts)
- Les notes i la solució es publicaran al Racó el dia 14 de novembre. La revisió es farà presencialment el 15 de novembre. De 8 a 9h (preguntes 1 i 2), i de 9 a 10h (preguntes 3, 4 i 5).

Pregunta 1 *(1,25 punts)*

Sigui el següent diagrama del multiplicador seqüencial de números naturals de 4 bits, anàleg a l'estudiat al curs, el qual calcula el producte en 8 bits:



Suposem que amb aquest circuit multipliquem els números binaris de 4 bits 1100 (multiplicand) i 1101 (multiplicador). Completa la següent taula, que mostra els valors en binari dels registres P, MD, i MR després de la inicialització i després de cada iteració, afegint tantes iteracions com facin falta:

Iter.	P (Producte)								MD (Multiplicand)								MR (Multiplicador)			
Init.	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**1**	**0**	**0**	**1**	**1**	**0**	**1**
1	**0**	**0**	**0**	**0**	**1**	**1**	**0**	**0**	**0**	**0**	**0**	**1**	**1**	**0**	**0**	**0**	**0**	**1**	**1**	**0**
2	**0**	**0**	**0**	**0**	**1**	**1**	**0**	**0**	**0**	**0**	**1**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**1**
3	**0**	**0**	**1**	**1**	**1**	**1**	**0**	**0**	**0**	**1**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**
4	**1**	**0**	**0**	**1**	**1**	**1**	**0**	**0**	**1**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
5																				
6																				
7																				
8																				

Pregunta 2 *(2,50 punts)*

Donat el següent codi en llenguatge C que utilitza accés seqüencial

```
void quadrat(int M[200][200]) {
   int i;
   for (i=0; i < 200; i+=4)
       M[i][200-i-1] = i*i;
}
```

Completa el codi MIPS que apareix a continuació:

```
quadrat:
   move  $t0, $zero
for:
   slti  $t1, $t0, 200
   beq   $t1, $zero, return
   mult  $t0, $t0
   mflo  $t1

   sw    $t1, [ 796 ] ($a0)

   addiu [ $a0, $a0, 3184 ]

   addiu $t0, $t0, 4
   b     for
return:
   jr    $ra
```

Considerant que:

- Les instruccions de memòria (lw/sw) tarden 4 cicles
- Els salts, si no salten, tarden 1 cicle
- Els salts, si salten, tarden 2 cicles
- La resta d'instruccions tarden 1 cicle
- La freqüència de rellotge del processador (f) és 2Ghz
- La potència dissipada per la CPU (P) és de 40W

Quantes instruccions s'executen en aquesta funció? **404**

Quants cicles de processador es consumeixen en aquesta funció? **606**

Calcula el temps d'execució (t_{exe}) d'aquesta funció en nanosegons (pots deixar l'expressió que el calcula sense avaluar) **303**

Calcula el CPI promig de la funció (pots deixar l'expressió que el calcula sense avaluar) **1,5**

Indica la fórmula que utilitzaries a partir de les dades proporcionades per calcular l'energia (E) consumida durant l'execució d'aquest programa. **$E = P \cdot t_{exe}$**

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

Pregunta 3 *(2,50 punts)*

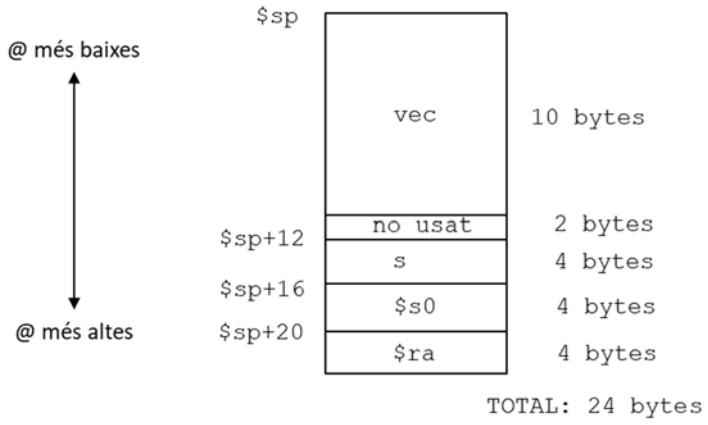
Donades les següents declaracions, en C:

```
int g(int p1, char *p2, int *p3);
int f(int par, char dat){
  char vec[10];
  int s;
      do {
            par = s + g(par, vec, &s);
      } while (vec[par] != dat);
      return par;
}
```

Seguint les regles de l'ABI de MIPS estudiades, quins elements de la funció f (variables locals, paràmetres, o càlculs intermedis) s'han d'emmagatzemar necessàriament en registres de tipus segur $s?

Element de `f` (en C)	Registre $s
dat	**$s0**

Dibuixa un esquema del bloc d'activació de `f`, 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 a $sp).



Tradueix a MIPS la següent sentència del cos de la subrutina `f`:

```
      par = s + g(par, vec, &s);
```

```
                              # par s'actualitza en $a0 en cada iteració
move    $a1, $sp              # vec
addiu   $a2, $sp, 12          # &s
jal     g
lw      $t0, 12($sp)          # s
addu    $a0, $t0, $v0         # par = s + f()
```

Pregunta 4 *(2,50 punts)*

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

```
char a = '3';
short b[4] = {-19, 7, -55, 1};
int c = 5;
short *d = b;
long long e;
char f[] = "2022";
short g[100];
```

Tradueix-la a llenguatge assemblador MIPS.

```
.data

a:      .byte   '3'
b:      .half   -19, 7, -55, 1
c:      .word   5
d:      .word   b
e:      .dword  0
f:      .asciiz "2022"
        .align  1
g:      .space  200
```

Completa la següent taula amb el contingut de memòria en hexadecimal (sense el prefix 0x), dels primers 48 bytes. 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 sense inicialitzar es deixen EN BLANC.

@Memòria	Dada	@Memòria	Dada	@Memòria	Dada
0x10010000	**33**	0x10010010	**02**	0x10010020	**32**
0x10010001		0x10010011	**00**	0x10010021	**30**
0x10010002	**ED**	0x10010012	**01**	0x10010022	**32**
0x10010003	**FF**	0x10010013	**10**	0x10010023	**32**
0x10010004	**07**	0x10010014		0x10010024	**00**
0x10010005	**00**	0x10010015		0x10010025	
0x10010006	**C9**	0x10010016		0x10010026	**00**
0x10010007	**FF**	0x10010017		0x10010027	**00**
0x10010008	**01**	0x10010018	**00**	0x10010028	**00**
0x10010009	**00**	0x10010019	**00**	0x10010029	**00**
0x1001000A		0x1001001A	**00**	0x1001002A	**00**
0x1001000B		0x1001001B	**00**	0x1001002B	**00**
0x1001000C	**05**	0x1001001C	**00**	0x1001002C	**00**
0x1001000D	**00**	0x1001001D	**00**	0x1001002D	**00**
0x1001000E	**00**	0x1001001E	**00**	0x1001002E	**00**
0x1001000F	**00**	0x1001001F	**00**	0x1001002F	**00**

Donat el següent codi en assemblador MIPS, indica quin és el valor final en hexadecimal del registre `$t0`:

```
la          $t1, b
lh          $t2, 0($t1)
lh          $t3, 2($t1)
div         $t2, $t3
mfhi        $t0
```

`$t0` = **0xFFFFFFFB**

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

Donat el següent codi en assemblador MIPS, indica quin és el valor final en hexadecimal del registre `$t0`:

```
lui         $t1, 0x1001
addiu       $t1, $t1, 6
lh          $t2, 0($t1)
sra         $t0, $t2, 4
```

`$t0 =` **0xFFFFFFFC**

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

```
*(d+1) = *d + 5;
```

```
la          $t0, d
lw          $t1, 0($t0)
lh          $t2, 0($t1)
addiu       $t2, $t2, 5
sh          $t2, 2($t1)
```

Pregunta 5 *(1,25 punts)*

Donada la següent funció escrita en C:

```
int f(int x, int y)
{
      if (x>y && (x%16)==0)
            return x;
      else
            return -1;
}
```

Completa el següent fragment de codi en MIPS, que tradueix l'anterior funció, escrivint en cada calaix un mnemònic d'instrucció o macro, una etiqueta, o un registre.

```
f:    ble         $a0, $a1,    et5
et1:  andi        $t0, $a0, 15
et2:  bne         $t0, $zero,  et5
et3:  move        $v0, $a0
et4:  b           et6
et5:  li          $v0, -1
et6:  jr          $ra
```

Final EC

Cognoms: …… Nom: ……
DNI: ……

EXAMEN PARCIAL D'EC
5 d'abril de 2022

- L'examen consta de 6 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 1:30 hores (90 minuts)
- Les notes i la solució es publicaran al Racó el dia 19 d'abril. La revisió es farà presencialment el 21 d'abril a les 9:00h.

Pregunta 1 *(1,5 punts)*

Un processador ha estat dissenyat per poder funcionar correctament a les següents combinacions de freqüències i voltatges d'alimentació:

	Voltatge (V)	Freqüència (GHz)
A	2,0	1,25
B	2,3	2

Sabent que la potència dinàmica de la combinació A és de P_A = 80W, es demana que contestis les següents preguntes:

Quina és la potència dinámica dissipada per la combinació B en watts?

$P = alfa \cdot C \cdot V^2 \cdot f = 80/(2^2 \cdot 1,25 \cdot 10^9) \cdot 2,3^2 \cdot 2 \cdot 10^9 = 169,28\ W$

Quin és el guany de rendiment (o speedup) que s'obté amb la combinació B respecte de la combinació A, executant el mateix programa?

$Guany = t_{exeA} / t_{exeB} = f_B / f_A = 1,6$

Amb la combinació A, quin és el temps d'execució (en segons) d'un programa que executa $1,2 \cdot 10^{10}$ instruccions i que té un CPI promig de 4?

$t_{exeA} = n \cdot CPI / f_A = 38,4\ s$

Pregunta 2 *(1,5 punts)*

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

```
short a[3] = {-6, 8, 3};
unsigned int b;
long long c = -8;
char d[] = {0xC, 0xA, 0xF, 0xE};
char e[] = "CAFE";
short *f = &a[1];
```

Tradueix-la a llenguatge assemblador MIPS

```
.data
a: .half -6, 8, 3
b: .word 0
c: .dword -8
d: .byte 0xC, 0xA, 0xF, 0xE
e: .asciiz "CAFE"
f: .word a+2
```

Omple la següent taula amb el contingut de memòria **en hexadecimal** (sense el prefix 0x). 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	@Memòria	Dada	@Memòria	Dada	@Memòria	Dada	@Memòria	Dada
0x10010000	**FA**	0x10010008	**00**	0x10010010	**F8**	0x10010018	**0C**	0x10010020	**00**
0x10010001	**FF**	0x10010009	**00**	0x10010011	**FF**	0x10010019	**0A**	0x10010021	
0x10010002	**08**	0x1001000A	**00**	0x10010012	**FF**	0x1001001A	**0F**	0x10010022	
0x10010003	**00**	0x1001000B	**00**	0x10010013	**FF**	0x1001001B	**0E**	0x10010023	
0x10010004	**03**	0x1001000C		0x10010014	**FF**	0x1001001C	**43**	0x10010024	**02**
0x10010005	**00**	0x1001000D		0x10010015	**FF**	0x1001001D	**41**	0x10010025	**00**
0x10010006		0x1001000E		0x10010016	**FF**	0x1001001E	**46**	0x10010026	**01**
0x10010007		0x1001000F		0x10010017	**FF**	0x1001001F	**45**	0x10010027	**10**

Calcula el valor final del registre $t0 en hexadecimal després d'executar el següent codi.

```
la    $t1, d
lh    $t1, 2($t1)
sra   $t1, $t1, 8
addiu $t1, $t1, -12
lui   $t2, 0x1001
or    $t2, $t2, $t1
lb    $t0, -1($t2)
```

$t0 = **0xFFFFFFFF**

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

Pregunta 3 *(1,5 punts)*

Donada la següent funció escrita en alt nivell en C:

```
int func(int x, unsigned int y) {
    if ((y > 0) && (x!=0))
        x = 0;
    else if ((~x ^ 0x1111) != 0)
        x++;
    return x;
}
```

Completa el següent fragment de codi en MIPS, que tradueix l'anterior funció, escrivint en cada calaix un mnemònic d'instrucció o macro, etiqueta, registre o immediat.

```
        bleu / beq      $a1, $zero,   et4
et1:    beq             $a0, $zero,   et4
et2:    move            $a0, $zero
et3:    b               et8
et4:    nor             $t2, $a0,     $zero / $a0
et5:    xori            $t2, $t2,     0x1111
et6:    beq             $t2, $zero, et8
et7:    addiu           $a0, $a0, 1
et8:    move            $v0, $a0
```

Pregunta 4 *(2 punts)*

Donat el següent programa en llenguatge C:

```
int A[10][10];
int B[100][8];

main() {
  int i;                          /* ocupa el registre $t0 */
  for (i=0; i<10; i=i+1){
     B[i*i][5]=A[9-i][i];
  }
}
```

Considerant que les variables globals ja estan declarades, completa el següent codi MIPS omplint les caselles en blanc perquè sigui equivalent a l'anterior codi en alt nivell. Tingues en compte que els elements de la matriu A s'accedeixen utilitzant la tècnica d'accés seqüencial.

```
main:  li     $t0, 0                          # i = 0
       la     $t1, A + [360]
       la     $t2, B
       li     $t3, 10
       b      cond
do:    lw     $t4, 0($t1)
       mult   $t0, $t0
       mflo   $t5
       sll    $t5, $t5, [5]
       addu   $t5, $t2, $t5
       sw     $t4, [20] ($t5)
       addiu  $t1, $t1, [-36]
       addiu  $t0, $t0, 1
cond:  blt    $t0, $t3, do
       jr     $ra
```

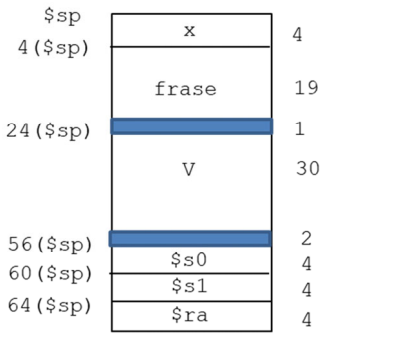
Cognoms: ..... Nom: .....
DNI: .....

Pregunta 5 *(2 punts)*

Donat el codi següent:

```
int examen(char c, char W[], int k) {
  int i, *j, x;
  char frase[19];
  short V[15]
  ...
}
```

Dibuixa el bloc d'activació de la rutina examen, indicant clarament la mida i el desplaçament necessari per accedir a cada element, sabent que utilitzarem els registres segurs `$s0`, `$s1` i `$ra` i que durant l'execució de la rutina s'utilitzarà `&x`.



Escriu les 4 primeres instruccions de la rutina `examen`.

```
examen:     addiu   $sp, $sp, -68
            sw      $s0, 56($sp)
            sw      $s1, 60($sp)
            sw      $ra, 64($sp)
```

Tradueix la següent sentència:

```
frase[*j] = W[k];
```

suposant que està dins la rutina examen, i que les variables locals `i` i `j` estan als registres `$s0` i `$s1` respectivament.

```
            addu    $t0, $a1, $a2
            lb      $t0, 0($t0)
            lw      $t1, 0($s1)
            addu    $t1, $t1, $sp
            sb      $t0, 4($t1)
```

Pregunta 6 *(1,5 punts)*

Donat el següent fragment de codi, en C

```
int func() {
        long long x, y;
        ...
        return (x < y);
}
```

Omple els requadres per tal que el següent fragment de codi MIPS sigui la traducció de la sentència visible de la funció `func()`. Suposem que:

`x` està guardat en `$t1` (part alta) i `$t0` (part baixa)
`y` està guardat en `$t3` (part alta) i `$t2` (part baixa)

```
        slt     $v0, [ $t1 ] , [ $t3 ]
    [ bne ]     $v0, $zero, fi
        bgt     $t1, $t3, fi
    [ sltu ]    $v0, $t0, $t2
fi:
```

Cognoms: …… Nom: ……
DNI:……

EXAMEN PARCIAL D'EC
4 de novembre de 2021

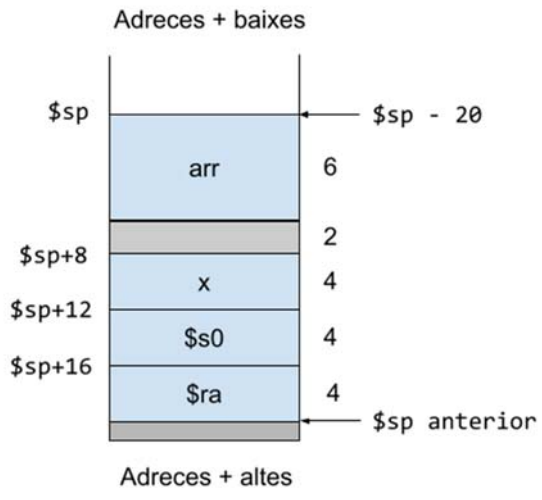
- L'examen consta de 5 preguntes, que s'han de contestar als mateixos fulls de l'enunciat. No oblidis posar el teu nom i cognoms a tots els fulls.
- La durada de l'examen és de 1:30 hores (90 minuts)
- Les notes, la solució i el procediment de revisió es publicaran al Racó el dia 12 de novembre.

Pregunta 1 *(2,50 punts)*

Donades les següents declaracions de funcions en C:

```
int f(short *a, int *b, char c);

int g(int j, char *k) {
    short arr[3];
    int x, y;

    y = j + f(&arr[1], &x, *k);
    return y * 2;
}
```

a) Dibuixa el bloc d'activació (stack frame) de `g`, indicant a quina posició apunta el registre `$sp` un cop reservat l'espai necessari a la pila, així com el nom de cada registre o variable que es guardi a la pila i la seva posició respecte a `$sp` (`$sp` + n bytes).



b) Tradueix el codi de la subrutina g.

```
g:
    addiu   $sp, $sp, -20
    sw      $s0, 12($sp)
    sw      $ra, 16($sp)
    move    $s0, $a0

    addiu   $a0, $sp, 2
    lb      $a2, 0($a1)
    addiu   $a1, $sp, 8
    jal     f
    addu    $v0, $v0, $s0
    addu    $v0, $v0, $v0

    lw      $ra, 16($sp)
    lw      $s0, 12($sp)
    addiu   $sp, $sp, 20
    jr      $ra
```

Cognoms: …… Nom: ……
DNI:……

Pregunta 2 *(1. 75 punts)*

Considera la següent subrutina programada en assemblador MIPS:

```
func:   ble   $a2, $a1, et1
        bgt   $a0, $a1, et2
et1:    blt   $a2, $zero, et4
et2:    move  $v0, $a0
et3:    b     et5
et4:    move  $v0, $a2
et5:    jr    $ra
```

Completa el següent codi escrit en C omplint les caselles en blanc perquè sigui equivalent a l'anterior codi en assemblador:

```
int func(int x, int y, int z) {
    int res;
    if ( (( z > y )&&( x > y )) || ( z >= 0 ) ) {
        res = x ;
    } else {
        res = z ;
    }
    return res;
    }
```

Pregunta 3 *(1.75 punts)*

Donat el següent codi en C que es tradueix a MIPS just a sota es demana que omplis les caselles buides del codi MIPS en funció de la constant N.

Codi C

```
#define N 1000
int m[N][N];

void main(){
  int i, suma=0;

  for (i=N-1; i>0; i-=2)
        suma += m[i][N-i];
}
```

Codi MIPS

```
main:   move  $t1, $zero
        li    $t0, N - 1
        la    $t2, m + 4 * N * (N-1) + 4
for:    lw    $t3, 0($t2)
        addu  $t1, $t1, $t3
        addiu $t0, $t0, -2
        addiu $t2, $t2, 8 * (1-N)
        bgt   $t0, $zero, for
        jr    $ra
```

Pregunta 4 *(2 punts)*

Donada la següent declaració de variables globals, que s'ubica a memòria a partir de l'adreça 0x10010000:

```
.data
a1:   .byte   '5'            # el codi ascii de '0' és 48
      .align  2
a2:   .space  3
a3:   .asciiz "2026"
a4:   .half   1, 0x37, -5
a5:   .word   a3
a6:   .half   0x7fff
```

a) Omple la següent taula amb el contingut de memòria **en hexadecimal**. Les posicions de memòria sense inicialitzar es deixen en blanc.

@Memòria	Dada	@Memòria	Dada	@Memòria	Dada	@Memòria	Dada
0x10010000	**0x35**	0x10010008	**0x30**	0x10010010	**0xFB**	0x10010018	**0xFF**
0x10010001		0x10010009	**0x32**	0x10010011	**0xFF**	0x10010019	**0x07**
0x10010002		0x1001000A	**0x36**	0x10010012		0x1001001A	
0x10010003		0x1001000B	**0x00**	0x10010013		0x1001001B	
0x10010004	**0x00**	0x1001000C	**0x01**	0x10010014	**0x07**	0x1001001C	
0x10010005	**0x00**	0x1001000D	**0x00**	0x10010015	**0x00**	0x1001001D	
0x10010006	**0x00**	0x1001000E	**0x37**	0x10010016	**0x01**	0x1001001E	
0x10010007	**0x32**	0x1001000F	**0x00**	0x10010017	**0x10**	0x1001001F	

b) Donat el següent codi que fa referència a l'anterior declaració:

```
main:
   la    $t0, a5
   lw    $t0,0($t0)
   lb    $t1,3($t0)
   la    $t2, a4
   lh    $t2, 4($t2)
   addu  $t1, $t1, $t2
   sb    $t1,3($t0)
   move  $t3, $zero
   li    $t4, 0x0a
   la    $t0, a3
   li    $t1, 3
loop:
   mult  $t3, $t4
   mflo  $t3
   lb    $t5, 0($t0)
   andi  $t5, $t5, 0x0f
   addu  $t3, $t3, $t5
   addiu $t1, $t1, -1
   addiu $t0, $t0, 1
   ble   $zero, $t1, loop
   jr    $ra
```

Omple la següent taula amb el valor en decimal dels registres $t1 i $t3 just ABANS d'executar la instrucció en negreta (cal usar una fila de la taula per cada iteració que es faci) i els valors dels mateixos registres en sortir del bucle:

	$t1	**$t3**
1a iter.:	**3**	**0**
2a iter.:	**2**	**2**
...	**1**	**20**
	0	**202**
en sortir:	**-1**	**2021**

Cognoms: …… Nom: ……
DNI:……

Pregunta 5 *(2 punts)*

Hem executat un programa que estem analitzant en un nou processador MIPS que funciona a una freqüència de 2 GHz. El programa té la següent distribució d'instruccions segons el seu CPI:

Tipus	CPI	% Instruccions
Accés a memòria (load/store)	8	20 %
Aritmètiques (add/sub/…)	2	50 %
Branches	4	30 %

Sabem que el nombre total d'instruccions del programa és de 10^9, i que la potència que dissipa el processador és de 100W.

a) Quin temps d'execució té el nostre programa en aquesta CPU (en segons), i quina quantitat d'energia necessitarà la seva execució (en Joules)?

Temps (s)	**1,9 s**
Energia (J)	**190 J**

Els nostres enginyers diuen que abans de llençar el nou processador MIPS al mercat hi ha temps per reduir el CPI d'un dels tipus d'instrucció a la meitat ($CPI_{nou} = CPI_{vell} / 2$).

b) Quin CPI hauríem de reduir per obtenir el màxim speedup en l'execució del nostre programa? Quin speedup obtindríem (pots deixar-ho en format de fracció)?

Instrucció a millorar	**Memòria**
Speedup obtingut	**~1,27 (38/30 o 19/15)**

Com a resultat de millorar el CPI, per mantenir la mateixa freqüència els enginyers han estimat que caldrà augmentar el voltatge del processador un 10%.

c) Suposant que la potència estàtica que dissipa el processador és zero (abans i després de la millora), quina serà la nova potència dissipada pel processador? Quanta energia consumirem ara en executar el nostre programa?

Potència (W)	**121 W**
Energia (J)	**181,5 J**

EXAMEN PARCIAL D'EC
20 d'abril de 2021

- 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 1:30 hores (90 minuts)
- Les notes, la solució i el procediment de revisió es publicaran al Racó el dia 3 de maig.

Pregunta 1 *(1 punt)*
Donada la següent subrutina en assemblador MIPS:

```
acces_aleatori:
        li      $t0, 400
        mult    $t0, $a1
        mflo    $t0
        addu    $t0, $t0, $a0
        lw      $v0, -80($t0)
        jr      $ra
```

Sabem que és la traducció de la següent funció en C (de la qual desconeixem els valors dels requadres). Completa els requadres amb les expressions vàlides en C per tal que la traducció sigui correcta:

```
int acces_aleatori (int M[][100], int i)
{
        return M[ i-1 ][ 80 ];
}
```

Pregunta 2 *(1.25 punts)*
Donada la següent funció `foo` en alt nivell correcte (no cal comprovar si se surt de rang), on `M` i `V` són declarades com a variables globals:

```
int M[100][100]; /* suposarem que està inicialitzada */
int V[100];
void foo() {
        int i,aux; /* ocupen els registres $t1 i $t3 respectivament */
        for (i=0; i<97; i++) {
                aux = M[i][i+3];
                M[i+1][i+3] = V[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 s'accedeixen utilitzant la tècnica **d'accés seqüencial**, usant el registre `$t0` com a punter per accedir els elements de la matriu.

```
        la      $t0, M + 12
        li      $t1, 0
        li      $t2, 97
        la      $t7, V
bucle:  bge     $t1, $t2, fibuc
        lw      $t3, 0($t0)     #hi ha variants: aquesta casella i la primera han de sumar 12
        sll     $t3, $t3, 2
        addu    $t3, $t3, $t7
        lw      $t4, 0($t3)
        sw      $t4, 400($t0)   #hi ha variants: aquesta casella i la primera han de sumar 412
        addiu   $t0, $t0, 404
        addiu   $t1, $t1, 1
        b bucle
fibuc:
```

Pregunta 3 *(0.75 punts)*
Donades les següents declaracions de variables locals

```
unsigned int a, b, c;
int d;
```

i el següent codi en alt nivell

```
if (((a<=b) && (a>c)) || (d!=0))
      d = -1;
else
      d = 0;
```

Sabent que les variables a, b, c, d estan guardades en els registres `$t0`, `$t1`, `$t2`, `$t3` respectivament, completa el següent fragment de codi MIPS omplint les caselles en blanc perquè sigui equivalent a l'anterior codi en alt nivell (*nota: posa atenció al tipus de les variables!*):

```
et1:   bgtu  $t0, $t1, et3
et2:   bgtu  $t0, $t2, et4
et3:   beq   $t3, $zero, et6
et4:   li    $t3, -1
et5:   b     et7
et6:   move  $t3, $zero
et7:
```

Pregunta 4 *(2 punts)*
Donat el següent fragment de codi en assemblador MIPS:

```
li     $t1, 0xC0580000
mtc1   $t1, $f2
li     $t2, 0x42000002
mtc1   $t2, $f4
li     $t3, 0xC0000009
mtc1   $t3, $f6
add.s  $f8, $f4, $f6
```

a) Quin valor representa, en decimal en notació científica normalitzada, el contingut que hi ha a `$f2`? *(0.5 punts)*

-3.375

b) Quina és la mantissa (en binari) i l'exponent (en decimal) dels nombres que hi ha a `$f4` i `$f6`? *(0.25 punts)*

		mantissa(binari)	exponent (decimal)
$f4:	1 ,	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0	5
$f6:	- 1 ,	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1	1

c) Volem fer la suma entre els registres `$f4` i `$f6`. Què valdrien els bits GRS i el seu càlcul en la operació? Omple les caselles que falten dels bits GRS amb els valors corresponents. Els valors de les operacions ja estan ajustats al nombre amb major exponent. *(0.25 punts)*

	G	R	S
1 , 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0	0	0	0
– 0 , 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	1	0	1
x x , x x x x x x x x x x x x x x x x x x x x x x x x	0	1	1

d) Quin és el contingut de `$f8` en hexadecimal després d'executar la instrucció `add.s`? *(1 punt)*

`$f8 =` 0x**41F00003**

Pregunta 5 *(1.5 punts)*
Donada la següent declaració de variables globals, que s'ubica a memòria a partir de l'adreça 0x10010000:

```
     .data
v1: .byte    -5
v2: .half    0x80
v3: .dword   0xFEDCBA9876543210
v4: .asciiz  "EC"  # codi ASCII 'A' = 0x41
    .align   2
v5: .space   8
v6: .word    v1
```

a) Omple la següent taula amb el contingut de memòria **en hexadecimal**. Les variables s'emmagatzemen a partir de l'adreça 0x10010000. Les posicions de memòria sense inicialitzar es deixen en blanc.

@Memòria	Dada	@Memòria	Dada	@Memòria	Dada	@Memòria	Dada
0x10010000	**0xFB**	0x10010008	**0x10**	0x10010010	**0x45**	0x10010018	**0x00**
0x10010001		0x10010009	**0x32**	0x10010011	**0x43**	0x10010019	**0x00**
0x10010002	**0x80**	0x1001000A	**0x54**	0x10010012	**0x00**	0x1001001A	**0x00**
0x10010003	**0x00**	0x1001000B	**0x76**	0x10010013		0x1001001B	**0x00**
0x10010004		0x1001000C	**0x98**	0x10010014	**0x00**	0x1001001C	**0x00**
0x10010005		0x1001000D	**0xBA**	0x10010015	**0x00**	0x1001001D	**0x00**
0x10010006		0x1001000E	**0xDC**	0x10010016	**0x00**	0x1001001E	**0x01**
0x10010007		0x1001000F	**0xFE**	0x10010017	**0x00**	0x1001001F	**0x10**

b) Calcula el valor final del registre $t0 en hexadecimal després d'executar el següent codi.

```
la   $t1, v1
lb   $t1, 0($t1)
sra  $t2, $t1, 4
sltu $t0, $t1, $t2
```

$t0 = 0x**00000001**

Pregunta 6 *(1 punt)*
S'executa un programa de test en un ordinador que té 3 tipus d'instruccions (A,B,C), amb CPI diferents. La següent taula especifica el nombre d'instruccions de cada tipus que executa el programa i el seu CPI:

Tipus d'instrucció	Nombre d'instruccions	CPI
A	$2*10^9$	1
B	$1*10^9$	1
C	$1*10^9$	2

a) Sabem que la freqüència de rellotge és de 1GHz i que la potència dissipada és P=100W. Calcula el temps d'execució en segons i l'energia consumida en Joules durant l'execució del programa de test.

t_{exe} = **5 s**

E = **500 J**

b) S'implementen unes millores al CPU que permeten incrementar la freqüència de rellotge a 2GHz. No obstant, el CPI de les instruccions de tipus B passa a ser de 4 cicles. Quin és el guany (speedup) de rendiment obtingut?.

Guany = **1.25**

Pregunta 7 *(2.5 punts)*
Donat el següent programa en C:

```
char G[7] = "EXAMEN";

int main() {
  char L[7];
  int i = 0;
  char x = 1;
  while(G[i] != '\0') {
    x = func(&G[i], L, i, x);
    i++;
  }
}

char func(char *a, char P[7], int b, char c) {
    P[b] = *a + c;
    return c + 1;
}
```

Completa la següent traducció a llenguatge assemblador MIPS omplint les caselles en blanc:

```
.data
G: .asciiz "EXAMEN"
.text
main:
  addiu $sp, $sp, -16
  sw    $s0, 8($sp)
  sw    $ra, 12($sp)
  li    $s0, 0
  li    $v0, 1
while:
  la    $a0, G
  addu  $a0, $a0, $s0
  lb    $t0, 0($a0)
  beq   $t0, $zero, endwhile
  move  $a1, $sp
  move  $a2, $s0
  move  $a3, $v0
  jal   func
  addiu $s0, $s0, 1
  b     while
endwhile:
  lw    $s0, 8($sp)
  lw    $ra, 12($sp)
  addiu $sp, $sp, 16
  jr $ra


func:
   lb    $t0, 0($a0)
   addu  $t0, $t0, $a3
   addu  $t1, $a1, $a2
   sb    $t0, 0($t1)
   addiu $v0, $a3, 1
   jr    $ra
```

COGNOMS, NOM:

EXAMEN PARCIAL D'EC

4 de novembre de 2020

L'examen consta de 5 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 90 minuts. Les notes, la solució i el procediment de revisió es publicaran al Racó.

Problema 1. (2,5 punts)

Donades les següents declaracions en llenguatge C, traduïu a assemblador MIPS la funció `f1` com una subrutina.

```
int f2(char a, long long *b, char c, int d);

int f1(long long w[], int i, char *p, char c) {
      return f2(*(p+i), &w[i], c, i) + 4*i;
}
```

```
f1:
      addiu  $sp, $sp, -8
      sw     $s0, 0($sp)
      sw     $ra, 4($sp)
      move   $s0, $a1          # guardem la i

      sll    $t0, $s0, 3
      addu   $a1, $a0, $t0     # 2n param: l'adreça és w + i*8
      addu   $a0, $a2, $s0     #
      lw     $a0, 0($a0)       # 1r param: contingut de l'@ p+i*1
      move   $a2, $a3          # 3r param: c
      move   $a3, $s0          # 4t param: i
      jal    f2
      sll    $t0, $s0, 2       # 4 * i
      addu   $v0, $v0, $t0     # retorna f2() + 4*i

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

COGNOMS, NOM:

Problema 2. (1,5 punts)

Donada la següent sentència escrita en alt nivell en llenguatge C:

```
if (((x==0) || (y!=0)) && ((y<x) || (x>0)))
            y=0;
else
            y=1;
```

Completeu el següent fragment de codi MIPS, que tradueix l'anterior sentència, escrivint en cada calaix un mnemònic d'instrucció o macro, una etiqueta, o un registre. Les variables `x` i `y` són de tipus `int` i estan inicialitzades i guardades als registres `$s0` i `$s1`, respectivament.

```
et1:        beq     $s0, $zero,  et3
et2:        beq     $s1, $zero,  et7
et3:        blt     $s1, $s0,    et5
et4:        ble     $s0, $zero,  et7
et5:        li      $s1, 0                  # y = 0
et6:        b       et8
et7:        li      $s1, 1                  # y = 1
et8:
```

COGNOMS, NOM:

Problema 3. (2,5 punts)

Donat el següent programa escrit en llenguatge C, traduïu-lo a assemblador MIPS amb el menor nombre possible de línies de codi a cada iteració del bucle.

```
short m[100][100];
main() {
int i = 0;
  while (m[i][i] > 0) {
      if (m[i][99-i] > m[i][i]) {
            m[i][99-i] = m[i][i];
      }
      i++;
  }
}
```

```
.data
m:  .space 100*100*2

.text
.globl main
main:
    la     $t1, m               # $t1: punter a m[i][i]
    addiu  $t2, $t1, 99*2       # $t2: punter a m[i][99-i]
    lh     $t3, 0($t1)
    ble    $t3, $zero, fbucle
bucle:
    lh     $t4, 0($t2)
    ble    $t4, $t3, fsi
    sh     $t3, 0($t2)
fsi:
    addiu  $t1, $t1, 101*2      # acumulem stride a $t1
    addiu  $t2, $t2, 99*2       #
    lh     $t3, 0($t1)
    bgt    $t3, $zero, bucle
fbucle:
    jr     $ra
```

COGNOMS, NOM:

Problema 4. (1,5 punts)

a) Mostreu el contingut de memòria a nivell de byte (amb les posicions d'alineament en blanc) corresponent a aquesta declaració:

```
.data                                   # comença a 0x10010000
lin1:    .half    -3, 0xff, 1
lin2:    .word    lin4
lin3:    .byte    'x'                   # el codi ascii de la 'x' és 120
lin4:    .asciiz  "xyz"
lin5:    .align   2
lin6:    .space   2
lin7:    .half    -0x7fff
```

0x10010000	**0xFD**	0x10010008	**0x0D**	0x10010010	**0x00**	0x10010018	
0x10010001	**0xFF**	0x10010009	**0x00**	0x10010011		0x10010019	
0x10010002	**0xFF**	0x1001000A	**0x01**	0x10010012		0x1001001A	
0x10010003	**0x00**	0x1001000B	**0x10**	0x10010013		0x1001001B	
0x10010004	**0x01**	0x1001000C	**0x78**	0x10010014	**0x00**	0x1001001C	
0x10010005	**0x00**	0x1001000D	**0x78**	0x10010015	**0x00**	0x1001001D	
0x10010006		0x1001000E	**0x79**	0x10010016	**0x01**	0x1001001E	
0x10010007		0x1001000F	**0x7A**	0x10010017	**0x80**	0x1001001F	

b) Indiqueu el valor final a `$t0` després d'executar aquest codi que fa referència a l'anterior declaració:

```
la    $t0, lin2
lw    $t0, 0($t0)
lhu   $t0, 1($t0)
lui   $t1, 0xA5A5
or    $t0, $t0, $t1
sra   $t0, $t0, 8
andi  $t0, $t0, -1
```

`$t0` = **0x0000A57A**

`$t0` = **0xFFA5A57A**

Nota: La darrera instrucció conté un error d'assemblatge de MARS, que decideix que hi ha sobreeiximent en representar la constant -1. Per tant, EL PROGRAMA NO ES POT EXECUTAR AL MARS. En qualsevol cas, considerem també correctes tant la solució que interpreta que s'executa aquesta darrera instrucció fent el producte lògic del register $t0 amb la màscara 0x0000FFFF (operació que es pretenia fer amb el codi de l'enunciat) com també amb la màscara 0xFFFFFFFF.

COGNOMS, NOM:

Problema 5. (2 punts)

Feu un programa que detecti si el registre `$t1` conté un valor capicua a nivel de bit, és a dir que el seu contingut es pot expressar com `$t1`: `abcd efgh ijkl mnop ponm lkji hgfe dcba`, per a qualsevol valor binari de les variables des de l'a fins la `p`. El valor final a `$t1` s'ha de codificar com CERT=1 i FALS=0.

```
main:
    li    $t0, 16
    move  $t2, $t1
bucle:
    andi  $t3, $t1, 1           # mirem bit de la dreta
    slt   $t4, $t2, $zero       # mirem bit de l'esquerra
    bne   $t3, $t4, fi_no       # si difereixen, hem acabat
    srl   $t1, $t1, 1
    sll   $t2, $t2, 1
    addiu $t0, $t0, -1
    bne   $t0, $zero, bucle
    li    $t1, 1         # és capicua
    jr    $ra
fi_no:
    li    $t1, 0         # no és capicua
    jr    $ra
```

COGNOMS: **GRUP:**

NOM:

EXAMEN PARCIAL D'EC
28 d'abril de 2020

L'examen consta de 6 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 **120 minuts**. Les notes, la solució i el procediment de revisió es publicaran al Racó properament.

Pregunta 1. (1,5 punts)

Sigui el següent programa en C:

```
int vecs[100];
int mats[5][10];

main() {
        int i;                  /* guardat al registre $t0 */
        for (i=0; i<10; i++)
              vecs[i*10 + 4] = mats[4][9-i];
}
```

Volem traduir el bucle d'aquest programa (sols el bucle) a assemblador MIPS, fent servir la tècnica d'accés seqüencial, tant per al recorregut de *vecs* com per al recorregut de *mats*. Determina, per a cada cas, la fórmula per calcular l'adreça del primer element del recorregut i el valor de l'stride. A continuació escriu, fent servir aquests valors, la traducció del bucle suposant que la variable *i* ocupa el registre $t0:

mats: @ inici = mats + 196 Stride = -4

vecs: @ inici = vecs + 16 Stride = 40

```
    la    $t4, mats + 196
    la    $t5, vecs + 16
    li    $t0, 0
    li    $t1, 10
  for:
    bge   $t0, $t1, fi_for
    lw    $t3, 0($t4)
    sw    $t3, 0($t5)
    addiu $t4, $t4, -4
    addiu $t5, $t5, 40
    addiu $t0, $t0, 1
    b     for
  fi_for:
```

Pregunta 2. (2,25 punts)

Sigui el següent fragment de programa en C:

```
long long matl[9][5];
long long *punterl;
main() {
        int i;              /* guardat al registre $t0 */
        char q;             /* guardat al registre $t1 */
        . . .               /* sentències dels apartats */
}
```

a) Tradueix a assemblador MIPS la següent sentència, suposant que i ocupa el registre $t0:

```
punterl = &matl[i+1][4];
```

```
# @matl[i+1][4] = matl + (i*5 + 1*5 + 4)*8 = matl + i*40 + 72

    la   $t1, matl + 72
    li   $t2, 40
    mult $t2, $t0
    mflo $t2
    addu $t1, $t1, $t2
    la   $t3, punterl
    sw   $t1, 0($t3)
```

b) Tradueix a assemblador MIPS la següent sentència, suposant que q ocupa el registre $t1:

```
if ((q >= '0') || (q <= '9'))
       q = q - '0';
```

```
    li   $t2, '0'
    bge  $t1, $t2, if
    li   $t2, '9'
    bgt  $t1, $t2, fi_if
if:
    addiu $t1, $t1, -'0'
fi_if:
```

c) Tradueix a assemblador MIPS la següent sentència (compte!! punterl és un punter a un enter de doble precisió):

```
*punterl = *punterl + 8;
```

```
    la    $t0, punterl
    lw    $t0, 0($t0)
    lw    $t1, 0($t0)
    lw    $t2, 4($t0)
    addiu $t3, $t1, 8
    sltu  $t4, $t3, $t1
    addu  $t2, $t2, $t4
    sw    $t3, 0($t0)
    sw    $t2, 4($t0)
```

COGNOMS:	GRUP:
NOM:	

Pregunta 3. (1,75 punts)

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

```
a:      .word  0x10010004
b:      .byte  0xDD
c:      .half  0x00D9
d:      .word  0xFF8406FE
e:      .half  0x0400
f:      .byte  0x40
g:      .dword 0xFFFFFF00E2E05401
```

a) Indica el contingut inicial de la memòria, representat en hexadecimal, segons el format que utilitza el simulador MARS, suposant que les posicions de memòria no inicialitzades son 0 (noteu que el simulador mostra el contingut en grups de paraules de 32 bits en hexadecimal):

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)
0x10010000	0x10010004	0x00D900DD	0xFF8406FE	0x00400400
0x10010010	0xE2E05401	0xFFFFFF00		

b) Quin és el valor final del registre $t1, en hexadecimal, després d'executar el següent fragment de codi?

```
la    $t0, a
lw    $t1, 0($t0)
lb    $t2, 2($t1)
sra   $t2, $t2, 2
sh    $t2, 4($t1)
lw    $t1, 4($t1)
```

$t1 = 0xFF84FFF6

Pregunta 4. (1,5 punts)

Donat el següent fragment de codi:

```
        li    $t1, 0x415C0403
        mtc1  $t1, $f6
        li    $t2, 0xBE800018
        mtc1  $t2, $f4
        sub.s $f8, $f6, $f4
```

a) Indica quin és el valor final de $f8 després d'executar el codi anterior, expressat en hexadecimal (escriu els càlculs intermedis al requadre gran, amb bona lletra)

0x415C0403 = 0**100 0001 0**101 1100 0000 0100 0000 0011
= + 1,01111000000010000000011 x 2^3

0xBE800018 = 1**011 1110 1**000 0000 0000 0000 0001 1000
= - 1,00000000000000000011000 x 2^{-2}

1,01111000000010000000011 x 2^3
+ 0,00001000000000000000011000 x 2^3
1,11000000000010000000011**110** x 2^3 (en negreta els bits GRS)

Abans de l'arrodoniment: 1,11000000000010000000011**110** x 2^3
Després de l'arrodoniment: 1,11000000000010000000100 x 2^3
Codificació: 0 10000010 11000000000010000000100 = 0x41600404

Càlcul de l'error en l'arrodoniment (diferència):
0,00000000000000000000001 x 2^3 = 1,0 x 2^{-22}

$f8 = 0x41600404

b) Indica si es comet error per pèrdua de precisió en l'operació de l'apartat anterior. En cas afirmatiu, calcula aquest error, i expressa'l en notació científica.

Error = 1,0 x 2^{-22}

COGNOMS:	GRUP:
NOM:	

Pregunta 5. (0,75 punts)

En un processador sota estudi, s'han obtingut les següents mesures:

- Freqüència de la instrucció FPSQR (arrel quadrada): 10% (sobre el total d'instruccions).
- Freqüència de les instruccions de coma flotant (excloent la instrucció FPSQR): 30%
- CPI mitjà de la instrucció FPSQR: 30
- CPI mitjà de les instruccions de coma flotant (excloent la instrucció FPSQR): 6
- CPI mitjà de la resta d'instruccions (que no són de coma flotant): 2

a) Indica la millora global de rendiment si reduïm el CPI de la instrucció FPSQR a 10.

Millora (speed-up)= 1,5

Pregunta 6. (2,25 punts)

Donada la següent declaració de funcions en C:

```
int g(int A, int *B);       /* prototip */

int func(int W[], int n, int m)
{
   int  V[11];
   int x = m;

   x = x + g(n, W);
   return V[x] + x;
}
```

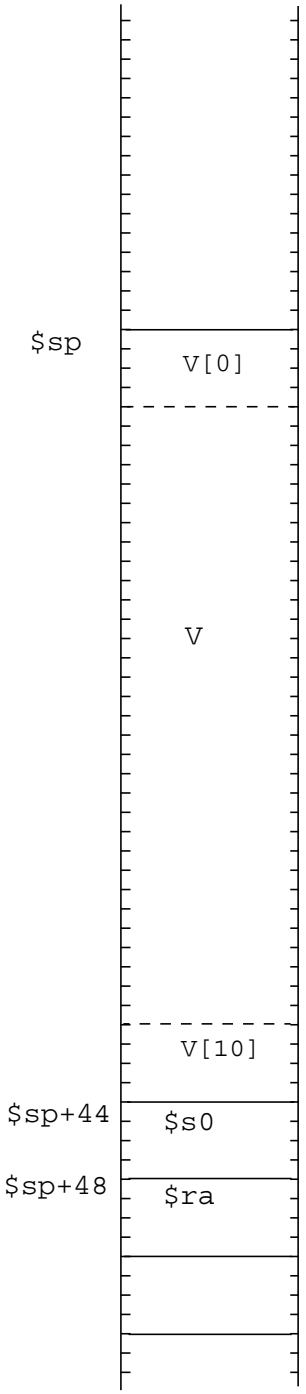
a) Examina el codi de la subrutina *func* i determina el nombre mínim de registres que s'hauran de salvar a la pila durant l'execució. Dibuixa el bloc d'activació de la subrutina, especificant la posició on apunta el registre $sp un cop reservat l'espai corresponent a la pila, així com el nom de cada registre i/o variable, indicant clarament per a cada un la seva posició (desplaçament relatiu al $sp).

b) Tradueix a assemblador MIPS la rutina `func`:

Cal salvar $ra (per la crida). Cal salvar x (en $s0) ja que després de la crida es necessita el seu valor anterior a la crida.

```
func:
  # Reserva pila i salva context
  addiu  $sp, $sp, -52
  sw     $s0, 44($sp)
  sw     $ra, 48($sp)

  # primera sentència
  move   $s0, $a2       # x = m

  # segona sentència
  move   $t0, $a0

  move   $a0, $a1       #  passem  n
  move   $a1, $t0       # passem W
  jal    g
  addu   $s0, $s0, $v0  # x = x + g(...)

  # tercera sentència
  sll    $t0, $s0, 2    # 4*x
  addu   $t0, $sp, $t0  # @V + 4*x
  lw     $t0, 0($t0)    # V[x]
  addu   $v0, $t0, $s0  # return V[x] + x

  # Restaura regs i allibera pila
  lw     $s0, 44($sp)
  lw     $ra, 48($sp)
  addiu  $sp, $sp, 52
  jr     $ra
```

Bloc d'activació



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:

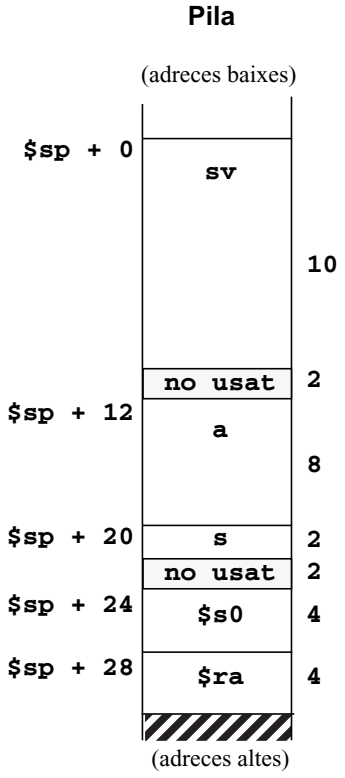
```
int f1(short *ps1, short *ps2,
       int *pi);

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:
```

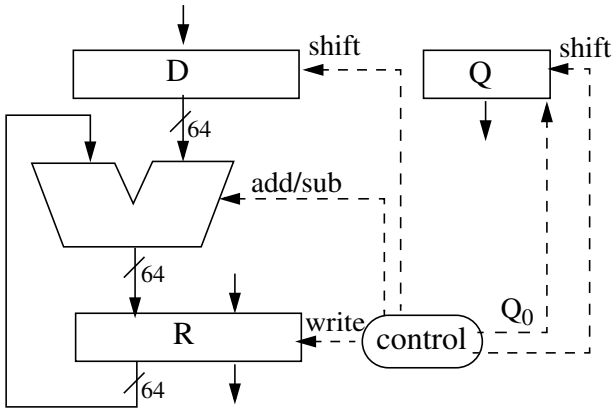
COGNOMS:	**GRUP:**
NOM:	

EXAMEN PARCIAL D'EC
6 de maig de 2019

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 **20** de maig.

Pregunta 1. (1 punt)

El següent esquema representa un circuit seqüencial que realitza la divisió dels nombres naturals de 32 bits X/Y. Aquest circuit calcula alhora el quocient QUO i el residu RES seguint el procediment de divisió amb restauració que has estudiat a classe. Completa l'algorisme en pseudocodi de la dreta (anàleg al llenguatge C) que en descriu el funcionament cicle a cicle.




```
R63:32 = [ 0 ] ; R31:0 = [ X ] ;
D63:32 = [ Y ] ; D31:0 = [ 0 ] ;
Q      = [ 0 ] ;
for (i=1; i<= [ 32 ] ; i++)
{
    D = D>>1;
    R = R-D;
    if (R<0)
        { R = R+D; Q = Q<<1; }
    else
        { Q = Q<<1 | 1; }
}
QUO  = [ Q ] ;
RES  = [ R31:0 ] ;
```

Pregunta 2. (1 punt)

Suposem els valors inicials dels registres $f2 = 0x41000001, i $f4 = 0xBF40000F. Calcula el valor de $f0 en hexadecimal després d'executar la instrucció `add.s $f0,$f2,$f4`.

$f0 = [0x**40E80000**]

Pregunta 3. (2 punts)

S'executa un programa de test en un ordinador que té 3 tipus d'instruccions (A,B,C), amb CPI diferents. La següent taula especifica el nombre d'instruccions de cada tipus que executa el programa i el seu CPI.

Tipus d'instrucció	Nombre d'instruccions	CPI
A	$8 \cdot 10^9$	7
B	$6 \cdot 10^9$	5
C	$4 \cdot 10^9$	4

a) Sabem que la freqüència de rellotge és de 1,5GHz i que la potència dissipada és P=100W. Calcula el temps d'execució en segons i l'energia consumida en Joules durant l'execució del programa de test.

t_{exe} = **68** s

E = **6800** J

b) Es vol modernitzar l'equipament comprant un nou ordinador amb una CPU amb ISA diferent. Aquesta nova CPU funciona amb el mateix voltatge però te més transistors, així que la seva capacitància agregada (C) és un 50% més gran i el factor d'activitat (α) un 20% superior. A més a més, els seu ISA té 4 tipus d'instruccions (A,B,C,D). Un cop recompilat el programa, el nombre d'instruccions de cada tipus que executa i el seu CPI són les indicades en la següent taula:

Tipus d'instrucció	Nombre d'instruccions	CPI
A	$5 \cdot 10^9$	2
B	$3 \cdot 10^9$	4
C	$2 \cdot 10^9$	5
D	$2 \cdot 10^9$	1

Sabem que amb aquesta nova CPU obtenim un *speedup* de 2. A quina freqüència (en GHz) va la nova CPU? ¿Quina potència dissipada en Watts consumeix l'execució del programa (podem suposar que la potència estàtica és zero tant en la CPU original com en la nova)?

Freq = **1** GHz

P = **120** W

COGNOMS:	**GRUP:**
NOM:	

Pregunta 4. (2 punts)

Donada la següent funció `foo` en llenguatge C, correctament programada:

```
void foo(unsigned char k, char M[][100]) {
        int i;
        for (i=2; i<48; i+=4){
                M[50-i][i*2] = M[k][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 M s'accedeixen utilitzant la tècnica **d'accés seqüencial** sempre que es pot, usant el registre `$t1` com a punter per a la matriu M. Aquest punter `$t1` s'inicialitza amb l'adreça de l'element `M[48][4]`. Al codi s'hi ha aplicat l'optimització de conversió d'un bucle `for` en un `do_while` i l'eliminació de la variable d'inducció.

```
        li    $t4, 100
        mult  $a0,$t4                 # k*100
        mflo  $t4
        addu  $t4, $a1, $t4           # @M[k][0]
        lb    $t4, [ 16 ]($t4)        # $t4=M[k][16]
        addiu $t1, $a1, [ 4804 ]      # @ inicial del punter: $t1 = @M[48][4]
        addiu $t2, $a1, [ 492 ]       # @ final del punter
        b     cond
bucle:  sb    $t4, [ 0 ]($t1)
        addiu $t1, $t1, [ -392 ]
cond:   bgeu  $t1, $t2, bucle
        jr    $ra
```

Pregunta 5. (1 punt)

Donada la següent sentència escrita en alt nivell en C:

```
if (((a<=b)&&(b!=0)) || (((b%8)^0x0005)>0))
        z=5;
else
        z=a-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.

```
        [bgt]   $t0, $t1,   [etq2]
etq1:   [bne]   $t1, $zero, [etq5]
etq2:   andi    $t3, $t1,   [7]
etq3:   [xori]  $t5, $t3,   [5]
etq4:   ble     $t5, $zero, [etq7]
etq5:   li      $t2, 5
etq6:   b       [etq8]
etq7:   subu    $t2, $t0, $t1
etq8:
```

Pregunta 6. (0,5 punts)

Donades les següents declaracions de funcions en C:

```
float g(int *w, int m);                 /* prototip de la funció g */
float f(int n) {
        float s, t;
        int v[100];

        for (s=0.0; s >= 0.0; n++) {
                t = g(v, n);
                s = t + g(v, n);
        }
        return s;
}
```

D'acord amb les regles de l'ABI estudiades, ¿quins elements de la funció f (paràmetres, variables locals i càlculs intermedis) s'han de guardar necessàriament en registres de tipus segur $s o $f? Especifica el registre segur concret que tries en aquells casos que en cal un. (Omple tantes entrades de la taula següent com et calguin):

Element de f (en C)	Registre
t	$f20
n	$s0

COGNOMS:	**GRUP:**
NOM:	

Pregunta 7. (1,5 punts)

Donades les següents declaracions de funcions en C:

```
void g(short *a, char *b, int c);
char f(int n) {
        char c;
        short v[20];

        g(v, &c, n);
        return c;
}
```

Tradueix a MIPS la funció f:

```
f:      addiu $sp, $sp, -48
        sw    $ra, 44($sp)

        move  $a2, $a0
        addiu $a0, $sp, 2
        move  $a1, $sp
        jal g
        lb    $v0, 0($sp)

        lw    $ra, 44($sp)
        addiu $sp, $sp, 48
        jr    $ra
```

Pregunta 8. (1 punt)

Un programa està compost de dos mòduls que es compilen i assemblen separadament per generar sengles fitxers objecte. Per a generar l'executable cal enllaçar-los després amb el muntador. El codi en C de les funcions `main()` i `func()` dels dos mòduls és el següent:

MÒDUL 1: `int main() { i=5; do { func(VA,i); i--} while(i!=0); }`
MÒDUL 2: `void func(int *W,int i) { VB[i]=W[i]+X; }`

Les variables `VA,X,VB` són globals de tipus `int` i estan definides en el mòdul 1. Hem traduït els dos fitxers a MIPS amb el següent resultat (hem afegit a l'esquerra els números de línia per facilitar les respostes posteriors):

MÒDUL 1

```
1          .data
2
3
4   VA:    .word -1, 0, -2, 0, -3
5   X:     .word 7
6   VB:    .word 0, 0, 0, 0, 0
7
8          .text
9          .globl main
10
11  main:  addiu  $sp, $sp, -8
12         sw     $s0, 0($sp)
13         sw     $ra, 4($sp)
14         li     $s0, 5
15  do:
16         la     $a0, VA
17         move   $a1, $s0
18         jal    func
19         addiu  $s0, $s0, -1
20         bne    $s0, $zero, do
21         lw     $s0, 0($sp)
22         lw     $ra, 4($sp)
23         addiu  $sp, $sp, 8
24         jr     $ra
```

MÒDUL 2

```
1          .data
2
3
4          .text
5
6   func:  sll    $t0, $a1, 2
7          addu   $t1, $a0, $t0
8          lw     $t2, 0($t1)
9
10         la     $t3, X
11         lw     $t4, 0($t3)
12         addu   $t4, $t4, $t2
13
14         la     $t5, VB
15         addu   $t6, $t5, $t0
16         sw     $t4, 0($t6)
17         jr     $ra
```

a) Quan hem intentat enllaçar els dos fitxers objecte generats per l'assemblador, l'enllaçador ha detectat diversos errors del tipus "referència no-resolta". ¿Com caldrà corregir o completar el codi MIPS perquè no torni a fallar? (fes servir tantes files de la taula com necessitis)

mòdul	núm. línia	línia de codi
1	2	`.globl X`
1	3	`.globl VB`
2	5	`.globl func`

b) Contesta les següents preguntes suposant que s'han tornat a assemblar els dos fitxers després de corregir els errors i que totes les instruccions conserven la numeració de línies original:

Pregunta	MÒDUL 1	MÒDUL 2
Quines etiquetes conté la Taula de Símbols Globals en cada fitxer objecte?	X, VB, main	func
Quines instruccions en cada fitxer objecte (sols el número de línia) requereixen **adreces absolutes** (de dades o de codi)?	16, 18	10, 14
Quines instruccions en cada fitxer objecte (sols el número de línia) contenen **referències no-resoltes** (a dades o a codi definits en l'altre mòdul)?	18	10, 14

COGNOMS:	**GRUP:**
NOM:	

EXAMEN PARCIAL D'EC
8 de novembre 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 **20** de novembre.

Pregunta 1. (1,0 punts)

Donada la següent sentència escrita en alt nivell en C:

```
if (((a & 0xFFFF) != 0) && ((a ^ 0xAAAA) > 0))
    a = 5;
else
    a = 1;
```

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. La variable `a` és de tipus `int` i està inicialitzada i guardada al registre `$t0`.

```
        [andi] $t1, $t0, 0xFFFF
        [beq]  $t1, $zero, [et5]
etq1:   [xori] $t1, $t0, 0xAAAA
etq2:   [ble]  $t1, $zero, [et5]
etq3:   li     $t0, 5
etq4:   b      et6
etq5:   li     $t0, 1
etq6:
```

Pregunta 2. (1,0 punts)

Suposem que els registres MIPS `$t1` i `$t2` valen `$t1=0x12341234` i `$t2=0xFFFFFFFF`. Indica el contingut final en hexadecimal de `$hi` i `$lo` després d'executar la instrucció `div $t1,$t2` i després de la instrucció `divu $t1,$t2`.

div $t1,$t2	$hi =	0x **0000 0000**	$lo =	0x **EDCB EDCC**
divu $t1,$t2	$hi =	0x **1234 1234**	$lo =	0x **0000 0000**

Pregunta 3. (1,5 punts)

Tradueix la funció `f2` de llenguatge C a una subrutina MIPS:

```
void f1(int v[], int x);
int f2(int a, int b)
{
    int tmp[2];

    f1(tmp, a);
    return tmp[0] + tmp[1] + b;
}
```

```
f2:     addiu  $sp, $sp, -16
        sw     $s0, 8($sp)
        sw     $ra, 12($sp)
        move   $s0, $a1

        move   $a1, $a0
        move   $a0, $sp
        jal    f1
        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
```

COGNOMS:	GRUP:
NOM:	

Pregunta 4. (1,5 punts)

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

```
char a[3] = "AB";
long long int b = -3;
char *c = &a[1];
char d = 'C';
int e[2] = {1025, 17};
```

a) Tradueix-la al llenguatge assemblador del MIPS.

```
        .data
a:      .asciiz "AB"
b:      .dword  -3
c:      .word   a+1
d:      .byte   'c'
e:      .word  1025, 17
```

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	**41**
0x10010001	**42**
0x10010002	**00**
0x10010003	
0x10010004	
0x10010005	
0x10010006	
0x10010007	

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

@Memòria	Dada
0x10010010	**01**
0x10010011	**00**
0x10010012	**01**
0x10010013	**10**
0x10010014	**43**
0x10010015	
0x10010016	
0x10010017	

@Memòria	Dada
0x10010018	**01**
0x10010019	**04**
0x1001001A	**00**
0x1001001B	**00**
0x1001001C	**11**
0x1001001D	**00**
0x1001001E	**00**
0x1001001F	**00**

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

```
la      $t0, c
lw      $t0, 0($t0)
lb      $t0, 0($t0)
sra     $t0, $t0, 4
```

`$t0 =` 0x **0000 0004**

Pregunta 5. (1,5 punts)

Donada la següent declaració en C:

```
void func(int mat[][40], int i) {
        mat[i][20-i] = mat[20-i][i];
}
```

Completa els requadres del següent fragment de codi per tal que sigui la traducció optimitzada de la funció func (pista: determina les dues adreces de memòria en funció de `i`, i observa si contenen algun terme en comú):

```
func:   li      $t2,  156
        mult    $t2, $a1
        mflo    $t2
        subu    $t1, $a0, $t2
        addu    $t0, $a0, $t2
        lw      $t3,  3200   ($t1)
        sw      $t3,  80     ($t0)
        jr      $ra
```

Pregunta 6. (1,0 punts)

Donades les següents declaracions en C:

```
int vec[100];
void f(short *p) {
        *(p+5) = *p + 3;
}

int *g(int i) {
        return &vec[i];
}
```

a) Tradueix a MIPS la funció f

```
f:      lh    $t0, 0($a0)
        addiu $t0, $t0, 3
        sh    $t0, 10($a0)
        jr    $ra
```

b) Tradueix a MIPS la funció g

```
g:      la    $v0, vec
        sll   $t0, $a0, 2
        addu  $v0, $v0, $t0
        jr    $ra
```

COGNOMS:	**GRUP:**
NOM:	

Pregunta 7. (1,5 punts)

Considera la següent funció f, que rep com paràmetres en $a0 i $a1 dos vectors de 1000 enters, i que retorna el seu producte escalar.

```
f:      addiu   $t0, $zero, 0
        addiu   $v0, $zero, 0
do:
        lw      $t1, 0($a0)
        lw      $t2, 0($a1)
        mult    $t1, $t2
        mflo    $t3
        addu    $v0, $v0, $t3
        addiu   $a0, $a0, 4
        addiu   $a1, $a1, 4
        addiu   $t0, $t0, 1
        slti    $t4, $t0, 1000
        bne     $t4, $zero, do      # salta si $t0<1000

        jr      $ra
```

Suposem que s'executa en un processador amb un rellotge de 2GHz i amb els següents CPI segons el tipus d'instrucció:

TIPUS	multiplicació	salt (salta)	salt (no salta)	load/store	altres
CPI	9	5	1	5	1

Els càlculs que es demanen a continuació sobre l'execució de la funció `f` s'han d'expressar amb nombres decimals amb un màxim de 2 dígits fraccionaris de precisió (es poden obtenir fàcilment sense calculadora).

a) Calcula el temps d'execució de `f` en microsegons ($1\mu s = 10^{-6}$ s)

temps = **15** μs

b) Calcula el CPI promig

CPI = **3**

c) Calcula el guany de rendiment (speedup) que obtindríem si optimitzem el disseny de la CPU de manera que les multiplicacions tardin 4 cicles en lloc de 9.

guany = **1,20**

Pregunta 8. (1,0 punts)

Un programa està compost de dos mòduls que es compilen i assemblen separadament per generar sengles fitxers objecte. Per a generar l'executable cal enllaçar-los després amb el muntador. El codi en C dels dos mòduls és el següent:

MODUL 1: `int main() { f(V[X]); }`

MODUL 2: `void f(int i) { Y=i; }`

Les variables `V,X,Y` són globals. Hem traduït els dos fitxers a MIPS amb el següent resultat (hem afegit a l'esquerra els números de línia per facilitar les respostes posteriors):

MÒDUL 1

```
1         .data
2         .globl V
3   V:    .word 1, 2, 3, 4, 5
4   Y:    .word 0

5         .text
6         .globl main
7   main:addiu  $sp, $sp, -4
8         sw    $ra, 0($sp)

9         la    $t0, X
10        lw    $t1, 0($t0) # $t1 <- X
11        la    $t2, V
12        sll   $t3, $t1, 2
13        addu  $t4, $t2, $t3
14        lw    $a0, 0($t4) # $a0 <- V[X]
15        jal   f

16        lw    $ra, 0($sp)
17        addiu $sp, $sp, 4
18        jr    $ra
```

MÒDUL 2

```
1         .data
2         .globl  X
3   X:    .word   3

4         .text
5         .globl  f
6   f:    la    $t0, Y
7         sw    $a0, 0($t0) # Y <- $a0
8         jr    $ra
```

a) Quan hem intentat enllaçar els dos fitxers objecte generats per l'assemblador, l'enllaçador ha detectat un error. Com caldrà corregir el codi MIPS perquè no torni a fallar?

mòdul: **1**

codi corregit: **canviar la línia 2 per: .globl Y**

b) Contesta les següents 3 preguntes suposant que s'ha corregit l'error anterior i que conservem la numeració de línies original:

Pregunta	MÒDUL 1	MÒDUL 2
Quines etiquetes conté la Taula de Símbols Globals de cada fitxer objecte?	**Y,main**	**X,f**
Quines línies de codi en cada fitxer objecte (sols el número) contenen referències **no-resoltes** (referències creuades)?	**9,15**	**6**
Quines línies de codi en cada fitxer objecte (sols el número) contenen **adreces absolutes** (i necessiten ser reubicades)?	**9,11,15**	**6**