

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);

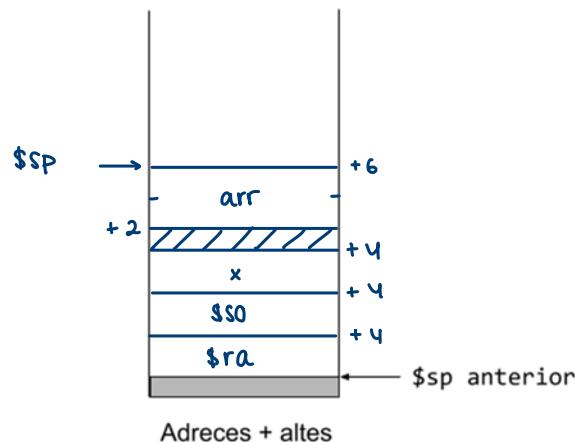
int g(int j, char *k) {
    short arr[3];
    int x, y;

    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).

Adreces + baixes



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) # $a2 = k
    addiu $a1, $sp, 8 # $a1 = x
    jal f
    addu $v0, $v0, $s0
    addu $v0, $v0, $v0

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

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]; → T = 4
void main(){
    int i, suma=0;
    for (i=N-1; i>0; i-=2)
        suma += m[i][N-i];
}
i = N-3
```

$M[i][j] = @N + i \cdot NC \cdot T + j \cdot T$

$i = N-1$

$M[N-1][N-(N-1)] = M[N-1][1]$

$M[N-1][1] = m + 4 \cdot N \cdot N - 1 + 4$

$M[N-3][N-(N-3)] = M[N-3][3] =$

Codi MIPS

```
main:    move   $t1, $zero    $t1 => suma
         li     $t0, (N-1)
         la     $t2, m+4·N·(N-1)+4
for:     lw     $t3, 0($t2) "M[0][0]" = $t3
         addu  $t1, $t1, $t3
         addiu $t0, $t0, -2
         addiu $t2, $t2, 8(N-1)
         bgt   $t0, $zero, for
         jr     $ra
```

$$\begin{aligned} & M[N-3][3] \\ & M[N-1][1] \\ \hline & M[-2][2] = -8 \cdot N + 8 \\ & \boxed{8(N-1)} \end{aligned}$$

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 → deixa espais fins arivar a una pos. múltiple de 22
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	0x 53	0x10010008	0x 48	0x10010010	0x FB	0x10010018	0x FF
0x10010001		0x10010009	0x 50	0x10010011	0x FF	0x10010019	0x ?F
0x10010002		0x1001000A	0x 54	0x10010012		0x1001001A	
0x10010003		0x1001000B	0x 00	0x10010013		0x1001001B	
0x10010004	0x 00	0x1001000C	0x 01	0x10010014	0x 07	0x1001001C	
0x10010005	0x 00	0x1001000D	0x 00	0x10010015	0x 00	0x1001001D	
0x10010006	0x 00	0x1001000E	0x 37	0x10010016	0x 01	0x1001001E	
0x10010007	0x 50	0x1001000F	0x 00	0x10010017	0x 10	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) → 0x37
la    $t2, a4
lh    $t2, 4($t2) → 0x FFFF
addu $t1, $t1, $t2 → 0x 0000 00031
sb    $t1, 3($t0)
move  $t3, $zero
li    $t4, 0xa → 0x 0a
la    $t0, a3
li    $t1, 3 → $t1 = 0x03
loop:
mult $t3, $t4
mflo $t3 → $t3=0x00 → 0x14 → 0x C8 → 0x 7E4
lb    $t5, 0($t0)
andi $t5, $t5, 0x0f
addu $t3, $t3, $t5 → 0x02 → 0x 14 → 0x CA → 0x 7E5
addiu $t1, $t1, -1 → 0x02 → 0x01 → 0x 00 → 0x FF
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

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 %

2. 10^9

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)?

$$CPI = \frac{8 \cdot 2 \cdot 10^9 + 2 \cdot 5 \cdot 10^9 + 4 \cdot 3 \cdot 10^9}{10 \cdot 10^9} = 2.8$$

Temps (s)	1.9s
Energia (J)	190W

$$T_{exe} = CPI \cdot N_i \cdot \frac{1}{f} = \frac{3.8 \cdot 10^9}{2 \cdot 10^9} = 1.9s$$

$$E = T_{exe} \cdot P = 1.9 \cdot 100 = 190W$$

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 ($CPI_{nou} = CPI_{vell} / 2$).

$$\text{mem } 8 \rightarrow \text{mem } 4 \Rightarrow CPI = \frac{4 \cdot 2 \cdot 10^9 + 2 \cdot 5 \cdot 10^9 + 4 \cdot 3 \cdot 10^9}{10 \cdot 10^9} = 3$$

$$T_{exe} = CPI \cdot N_i \cdot \frac{1}{f} = \frac{3 \cdot 10^9}{2 \cdot 10^9} = 1.5$$

- 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	19/15

$$\rightarrow \text{speed-up} = \frac{1.9}{1.5} = \frac{19}{15}$$

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)	??
Energia (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 Raccó el dia 3 de maig.

Pregunta 1 (1 punt)

Donada la següent subrutina en assemblador MIPS:

```

acces_aleatori:
    li    $t0, 400 * 100·4
    mult $t0, $a1 # 400·i
    mflo $t0 # x
    addu $t0, $t0, $a0 # @N+400i
    lw    $v0, -80($t0)
    jr    $ra

```

\downarrow
ens movem endarrera: $i-1, 80 : 100 - \frac{80}{4} = 100 - 20 = 80$

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 ];
}

```

$$\begin{aligned}
M[i][j] &= @M + i \cdot nc + j \cdot 4 \\
M[i][100] &= @M + i \cdot 100 \cdot 4 + 4 \cdot j \\
M[i-1][j] &= @M + i \cdot 100 \cdot 4 - 80 \\
M[i-1][j] &= @M + i \cdot 100 \cdot 4 - 20 \cdot 4 \\
\rightarrow M[i-1][100-20] &= M[i-1][80]
\end{aligned}$$

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[0][3] = @M + 0 + 12 = @M + 12
        M[i+1][i+3] = V[aux];
    }
}

```

$\hookrightarrow M[0][3] = @M + 1 \cdot 100 \cdot 4 + 3 \cdot 4 = @M + 400 + 12 = @M + 412$

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      #aux
li    $t1, 0      #i=0
li    $t2, 97
la    $t7, V      # $t7 = @V
bucle: bge   $t1, $t2, fibuc * i>=97?
        lw     $t3, 0($t0) # Primer valor d'aux
        sll   $t3, $t3, 2  # aux · tamV = aux · 4
        addu $t3, $t3, $t7 # V = @V + aux · 4
        lw     $t4, 0($t3) # Primer valor d'V
        sw     $t4, 400($t0)
        addiu $t0, $t0, 400 stride
        addiu $t1, $t1, 1
b bucle
fibuc:

```

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 mult. 1
v2: .half 0x80 mult. 2 → 1000 0000
v3: .dword 0xFEDCBA9876543210 mult. 8
v4: .ascii "EC" # codi ASCII 'A' = 0x41 mult. 1 acaba ox00
    .align 2 2 → mult. 4 42
v5: .space 8 → 8 espacios 43
v6: .word v1 mult. 4 44
                                45

```

- 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	FB	0x10010008	10	0x10010010	43	0x10010018	00
0x10010001		0x10010009	32	0x10010011	45	0x10010019	00
0x10010002	80	0x1001000A	54	0x10010012	00	0x1001001A	00
0x10010003	00	0x1001000B	76	0x10010013		0x1001001B	00
0x10010004		0x1001000C	98	0x10010014	00	0x1001001C	00
0x10010005		0x1001000D	BA	0x10010015	00	0x1001001D	00
0x10010006		0x1001000E	DC	0x10010016	00	0x1001001E	01
0x10010007		0x1001000F	FF	0x10010017	00	0x1001001F	10

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

```

la    $t1, v1    $t1 = 10010000
lb    $t1, 0($t1) $t1 = FB 11111111110111      FF > FB ✓
sra   $t2, $t1, 4 2^4 = 8 → $t2 = 1111 1111      -1 -5
sltu $t0, $t1, $t2

```

$$\$t0 = \boxed{0x\ 0000\ 0001}$$

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	$2 \cdot 10^9$	1
B	$1 \cdot 10^9$	1
C	$1 \cdot 10^9$	2

$$\begin{aligned} CPI &= \frac{1 \cdot 2 \cdot 10^9 + 1 \cdot 1 \cdot 10^9 + 2 \cdot 1 \cdot 10^9}{2 \cdot 10^9 + 1 \cdot 10^9 + 1 \cdot 10^9} = \frac{5}{4} \\ t_{exe} &= \frac{(2 \cdot 10^9 + 1 \cdot 10^9 + 1 \cdot 10^9) \cdot \frac{5}{4}}{1 \cdot 10^9} = \frac{4 \cdot 5}{4} = 5s \end{aligned}$$

- 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.

$$t_{exe} = \boxed{5s}$$

$$E = \boxed{500J} \quad E = P \cdot t_{exe} = 100 \cdot 5 = 500$$

- 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?.

$$\begin{aligned} \text{Guany} &= \boxed{5/4s} \quad f_r = 2 \cdot 10^9 \quad \text{Speedup} = \frac{5}{4}s \\ CPI &= \frac{1 \cdot 2 \cdot 10^9 + 4 \cdot 1 \cdot 10^9 + 2 \cdot 1 \cdot 10^9}{2 \cdot 10^9 + 1 \cdot 10^9 + 1 \cdot 10^9} = \frac{8}{4} = 2 \\ t_{exe} &= \frac{(2 \cdot 10^9 + 1 \cdot 10^9 + 1 \cdot 10^9) \cdot 2}{2 \cdot 10^9} = \frac{4 \cdot 10^9 \cdot 2}{2 \cdot 10^9} = \frac{8 \cdot 10^9}{2 \cdot 10^9} = 4s \end{aligned}$$

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++;
    }
}

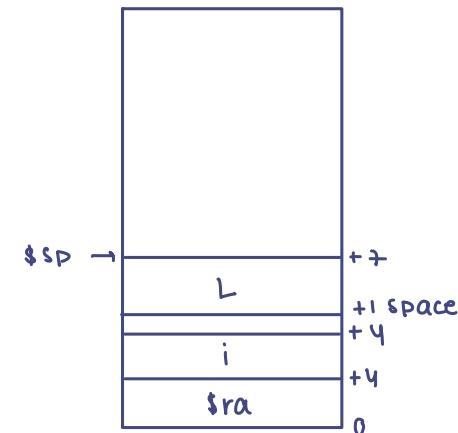
char func(char *a, char P[7], int b, char c) {
    P[b] = *a + c;
    return c + 1;
}
```

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 # $a0 = @G
    addu $a0, $a0, $s0 # @G+i → per anar a la següent G[i]
    lb $t0, 0($a0) # $t0 = G[i]
    beq $t0, $zero, endwhile # comparem
    move $a1, $sp
    move $a2, $s0
    move $a3, $v0
    jal func
    addiu $s0, $s0, 1 # ++i
    b while
endwhile:
    lw $s0, 8($sp)
    lw $ra, 12($sp)
    addiu $sp, $sp, 16
    jr $ra

func:
    lb $t0, 0($a0) # $t0 = *a
    addu $t0, $t0, $a3 # *a+c
    addu $t1, $a1, $a2 # P[b] = @P + b·1 = @P + b
    sb $t0, 0($t1) # P[b] = *a+c
    addiu $v0, $a3, 1 # c+1
    jr $ra
```

char ↗ G = \$a0
* \$t0 = *a



COGNOMS, NOM:

EXAMEN PARCIAL D'EC

4 de novembre de 2020

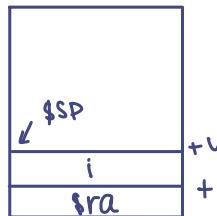
L'examen consta de 5 preguntes, que s'han de contestar als mateixos fulls de l'enunciat. No oblideu posar el nom i cognoms a tots els fulls. La durada de l'examen és de 90 minuts. Les notes, la solució i el procediment de revisió es publicaran al Racó.

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;
}
```

$$w[i] = @w + i \cdot 8^3$$



```
f1: addiu $sp, $sp, -8
    sw $s0, 0($sp)
    sw $ra, 4($sp)
    move $s0, $a1
    sll $t0, $s0, 2 # i·4
    sll $t1, $s0, 3 # i·8
    addu $a1, $a0, $t1 # @w + i · 8
    addu $t1, $a2, $s0 # p+i
    lw $a0, 0($t1)
    move $a2, $a3 # c
    move $a3, $s0 # i
    jal f2
    addu $v0, $v0, $t0 # f2 + i · 4
    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]; → half
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++;
    }
}
```

$\rightarrow N = @M + i \cdot NC \cdot T + j \cdot T$

$m[0][0] > 0$

$m[0][99] = @m + 0 + 99 \cdot 2 = @m + 198$

$\frac{m[i][i]}{m[0][0]}$

$m[1][1] = 100 \cdot 2 + 1 \cdot 2 = \underline{\underline{202}}$

$m[1][98]$

$\frac{-m[0][99]}{m[1][-1]} = 200 + (-2) = \underline{\underline{198}}$

```
.data
M : .space 100 * 100 * 2
      .text
      .globl main

main: la $t1, m # *N[i][i]
      addiu $t2, $t1, 198 # *N[i][99*2]
      lh $t3, 0($t1) # $t3 = N[i][i]
      ble $t3, $zero, fiwhile
while: lh $t4, 0($t2) # $t4 = N[i][99-i]
      ble $t4, $t3, fiif
      sh $t3, 0($t2) # m[i][99-i] = m[i][i]

fiif: addiu $t1, $t1, 202
      addiu $t2, $t2, 198
      lh $t3, 0($t1)
      bgt $t3, $zero, while

fiwhile: jr $ra
```

COGNOMS , NOM:

Problema 4. (1,5 punts)

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

0x10010000	FD	0x10010008	0D	0x10010010	00	0x10010018	
0x10010001	FF	0x10010009	00	0x10010011		0x10010019	
0x10010002	FF	0x1001000A	01	0x10010012		0x1001001A	
0x10010003	00	0x1001000B	10	0x10010013		0x1001001B	
0x10010004	01	0x1001000C	78	0x10010014	00	0x1001001C	
0x10010005	00	0x1001000D	79	0x10010015	00	0x1001001D	
0x10010006		0x1001000E	79	0x10010016	01	0x1001001E	
0x10010007		0x1001000F	80	0x10010017	08	0x1001001F	

$$\begin{array}{r} 1000\ 0000\ 0000\ 0000 \\ + 0000\ 0000\ 0000\ 0001 \\ \hline 1000\ 0000\ 0000\ 0001 \end{array} \rightarrow 8001$$

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

la	\$t0, lin2	10010008 1001000D		
lw	\$t0, 0(\$t0)	0x8079	0x00008079	10000000 01111001
lhu	\$t0, 1(\$t0)	0xA5A5	0x0000A5A5	10100101 10100101
lui	\$t1, 0xA5A5			10100101 11111101
or	\$t0, \$t0, \$t1			0x0000A5FD
sra	\$t0, \$t0, 8	→ 0x000000A5		
andi	\$t0, \$t0, -1			

\$t0 = 0x0000 00A5

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 pqrst 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 # i      32/2 = 16
      move $t2, $t1

bucle: andi $t3, $t1, 1  # mirem el bit de la dreta
      slt  $t4, $t2, $zero # mirem bit de l'esquerra
      bne $t3, $t4, fi_no # Si $t3 ≠ $t4 hem acabat
      srl  $t1, $t1, 1
      sll  $t2, $t2, 1
      addiu $t0, $t0, -1 # -- i
      bne $t0, $zero, bucle
      li   $t1, 1 # és capicua
      jr   $ra

fi_no: li $t1, 0 # no és
      jr $ra
```

COGNOMS:

NOM:

GRUP:

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

```
li $t0, 0 # i=0
la $t1, vecs
addiu $t1, $t1, 16 # @v+16
la $t2, mats
addiu $t2, $t2, 196 # @m+196
li $t3, 10
for: bge $t0,$t3, fifor
    lw $t4, 0($t2) # m[4][9]
    sw $t4, 0($t1)
    addiu $t1, $t1, 40
    addiu $t2, $t2, -4
    addiu $t0, $t0, 1 # ++i
    b for
fifor:
```

Pregunta 2. (2,25 punts)

Sigui el següent fragment de programa en C:

```
long long mat1[9][5];
long long *punterl;
main() {
    int i;                  /* guardat al registre $t0 */
    char q;                 /* guardat al registre $t1 */
    . . .
}
```

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

```
punterl = &mat1[i+1][4];
```

```
# m[i+1][4] = @m + (i+1) · 8 + 4 · 8 = @m + (i+1) · 40 + 32 = @m + 40i + 72
la $t1,mat1
addu $t1,$t1,72      # @m+72
li $t2,40
mult $t2,$t0           # 40·i
mflo $t2
addu $t1,$t1,$t2       # @m+72+40i
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 $t0,'0'
bge $t1,$t0,if
li $t2,'9'
bgt $t1,$t2,fiif
if : addiu $t1,$t1,-'0'
fiif :
```

- 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
sllu $t4,$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  0xFFFFFFFF00E2E05401

```

- 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	10010004	00D900DD	FF8406FE	00400000
0x10010010	E2E05401	FFFFFFFFFF		

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

```

la    $t0, a  0x10010000
lw    $t1, 0($t0) 0x10010004
lb    $t2, 2($t1) 0x D9
sra   $t2, $t2, 2  0xFFFFFFFF
sh    $t2, 4($t1) 0x FF84UFFFG
lw    $t1, 4($t1)

```

\$t1 = 0x **FF84UFFFG**

COGNOMS:

NOM:

GRUP:

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.

$$\text{Millora (speed-up)} = \boxed{1.5}$$

$$\begin{array}{ll} A & 10 \quad 30 \\ C & 30 \quad 6 \\ R & 60 \quad 2 \end{array} \quad \text{Speed-up} = \frac{\text{Texe}_0}{\text{Texe}_1} = \frac{n_{IPS} \cdot CPI_0 \cdot t_c}{n_{IPS} \cdot CPI_1 \cdot t_c} = \frac{CPI_0}{CPI_1} = \frac{6}{4} = \frac{3}{2} = 1.5$$

$$CPI_0 = \frac{10 \cdot 30 + 30 \cdot 6 + 60 \cdot 2}{100} = \frac{600}{100} = 6$$

$$CPI_1 = \frac{10 \cdot 10 + 30 \cdot 6 + 60 \cdot 2}{100} = \frac{400}{100} = 4$$

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];      11·4 = 44
    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:

```
addiu $sp, $sp, -52
sw $s0, 44($sp)
sw $ra, 48($sp)
move $s0, $a2 # x=m

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

sll $t0, $s0, 2 # 4i
addu $t0, $t0, $sp # @v+4i
lw $t0, 0($t0) # v[x]
addu $v0, $t0, $s0 # v[x]+x

lw $s0, 44($sp)
lw $ra, 48($sp)
addiu $sp, $sp, 52
jr $ra
```

Bloc d'activació

\$sp →

v

+44
\$s0=x
+4
\$ra
+4

COGNOMS:

NOM:

GRUP:

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)

$$G[0][63] = @G + 0 \cdot 64 \cdot 4 + 63 \cdot 4 = @G + 63 \cdot 4$$

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 quadres per tal que la traducció sigui correcta:

f: # Inicialitza el punter \$t0 <- @G[0][63-j]

```
la      $t0, G + 63 · 4
sll    $t4, $a1, 2 * j · 4
subu   $t0, $t0, $t4 # G + 63 · 4 - j · 4
# Inicialitza el punter $t1 <- @M[j][i]
sll  $t3, $a1, 8 * j · 64 · 4
sll  $t2, $a0, 2 * i · 4
addu $t2, $t2, $t3
addu $t1, $t2, $a2 * @m + j · 64 · 4 + i · 4
```

while:

```
bge   $a1, $t2, fi
lw    $t4, 0($t0) # G[0][63-j]
sw    $t4, 0($t1) * M[j][i]
```

```
addiu $t0, $t0, -4
```

```
addiu $t1, $t1, 64 · 4
```

```
addiu $a1, $a1, 1
```

```
b     while
```

```
jr    $ra
```

fi:

$$\begin{aligned} j &= 0 \\ M[j][i] &= M[0][i] = @m + 4 \cdot 64 \cdot j + 4 \cdot i \\ @m &\rightarrow \$a2 \end{aligned}$$

$$\begin{aligned} &G[0][62] \\ &- G[0][63] \\ &G[0][-1] = 0 \cdot 64 \cdot 4 + (-1) \cdot 4 = -4 \end{aligned}$$

$$\begin{aligned} &M[1][i] \\ &- M[0][i] \\ &M[1][0] = 1 \cdot 64 \cdot 4 + 0 \cdot 4 = 64 \cdot 4 \end{aligned}$$

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 +1
for:
    beq    $a1, $zero, fifor +100 # surt si N==0 +1
    lbu    $t2, 0($a0) +100      # 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) +100      # 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 +100
fifor:
    nor    $v0, $v0, $zero +1
    jr    $ra, +1
```

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

$$t_{\text{exe}} = \frac{3012}{2 \cdot 10^9} \text{ s}$$

$$\begin{aligned} \text{CPI} &= \frac{3012}{1204} \\ \text{t}_{\text{exe}} &= 1204 \cdot \frac{3012}{1204} \cdot \frac{1}{2 \cdot 10^9} = \frac{3012}{2 \cdot 10^9} \end{aligned}$$

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

$$E = \frac{3012}{2 \cdot 10^9} \cdot 100 \text{ J}$$

COGNOMS:

NOM:

GRUP:

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 $a \cdot b$ 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 * $a0 · $a1  
          mfhi $t1 # "overflow"  
          mflo $t0 # 32 bits de menys pes
```

```
sra $t0, $t0, 31 # $t0=bit signe  
subu $t0,$t0, $t1 # si $t0-$t1 = 0 llavors $t0 = $t1  
sltiu $v0, $zero, $t0
```

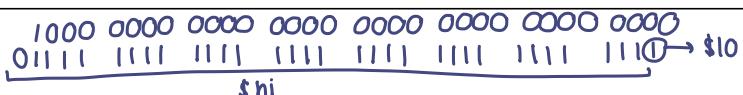
jr \$ra

- 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 = \$lo = overflow (Si/No) :

en què es coneix?

\$hi != 0



Pregunta 4. (0,70 punts)

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

```
int g(short *q, short val) {  
    $a0           $a1  
    return (*q == val); /* Retorna 1 si són iguals, 0 altrament */  
}
```

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

jr \$ra

Pregunta 5. (1,80 punts)

Donades les següents declaracions de funcions en C:

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

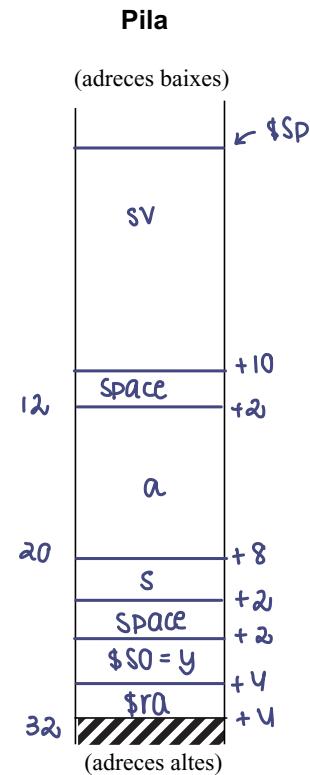
int f2(short *x, int y) {
    short sv[5]; 5·2=10
    int a[2], res; 4·2=8
    short s;
    s = *x + 3;
    res = y + f1(&sv[2], &s, a);
    return res;
}
```

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:

NOM:

GRUP:

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[2] = @a + 2 · 2 = @a + 4
```

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

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

31 | 2
| 1 | 15 | 2
| 1) | 1) | 7 | 2
| 1) | 3 | 2
| 1) | 1

- 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	FF
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 0x10010018
lw      $t1, 0($t1)
subu    $t0, $t0, $t1
```

1 0 0 9 → 0x1009

\$t0 = 0x00000004 ??

- d) (0,6 pts) Tradueix a llenguatge assemblador del MIPS la següent sentència en C:
 $*(\text{d} - 2) = *\text{d} + 5;$

```

la $t0, d
lw $t1, 0($t0) # @a
lh $t2, 0($t1) # el que hi ha adins d'a
addiu $t2, $t2, 5 # *d+5
sh $t2, -4($t1) # a·(-2) = -4
          ↑
          half

```

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	<code>\$t3, \$t0, 0xFFFF # \$t3 resultat xori</code>

etq1: <code>beq \$t3, \$zero,</code>	etq6
etq2: nor	<code>\$t4, \$t1, \$zero</code>
etq3: bgt	<code>\$t4, \$t0,</code>
etq4: <code>addu \$t2, \$t0, \$t1</code>	etq6
etq5: <code>b</code>	etq7
etq6: <code>xor \$t2, \$t2, \$t2</code>	
etq7:	

COGNOMS:

NOM:

GRUP:

EXAMEN PARCIAL D'EC

13 de novembre de 2017

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 **22** de novembre.

Pregunta 1. (1.0 punts)

Considera la següent acció `s1`, escrita en llenguatge C:

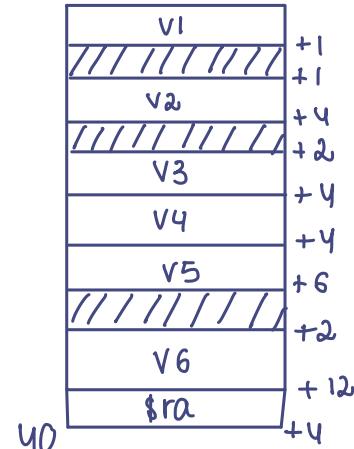
```
void s1() {
    char v1;
    short v2[2]; 2·2 = 4
    long long @v3; 4
    int v4; 4
    char v5[6]; 1·6 = 6
    int v6[3]; 4·3 = 12

    return b(&v1, v2, &v3, &v4, v5, v6);
}
```

Cal adonar-se que totes les variables locals s'han d'emmagatzemar a memòria, dins el bloc d'activació (BA) a la pila.

Indica per cada variable local quina és l'adreça de memòria on s'emmagatzema, amb el format `$sp+despl`, on `despl` és el desplaçament en bytes que hi ha des de l'inici del BA fins on comença a emmagatzemar-se la variable.

<code>@v1 = \$sp +</code>	<code>0 (\$sp)</code>
<code>@v2 = \$sp +</code>	<code>2 (\$sp)</code>
<code>@v3 = \$sp +</code>	<code>8 (\$sp)</code>
<code>@v4 = \$sp +</code>	<code>12 (\$sp)</code>
<code>@v5 = \$sp +</code>	<code>16 (\$sp)</code>
<code>@v6 = \$sp +</code>	<code>20 (\$sp)</code>



Indica l'espai total en bytes que ocupa el BA, incloent-hi l'espai dedicat a salvar registres.

mida BA =

`40`

Pregunta 2. (1,20 punts)

Escriu el valor final en hexadecimal del registre \$t0 en cada un dels següents apartats després d'executar el fragment de codi.

a) (0,50 pts.)

```
li      $t0, 0x0020A040
sw      $t0, 0($sp)    0x0020A040
lb      $t0, 1($sp)    0x00000000
srl     $t0, $t0, 4    0x00000001
ori     $t0, $t0, 0x0099
```

0000	1111	1111	1111	1111	1111	1111	1010
0000	0000	0000	0000	0000	0000	0000	1001 1001

0	F	F	F	F	F	F	1011
							B

\$t0 = 0x0FFFFFFB

b) (0,30 pts.)

```
addiu $t0, $zero, -1      0xFFFFFFFF
sltu  $t0, $zero, $t0 0x1  0x00000001
addiu $t0, $t0, -1
```

\$t0 = 0x00000000

c) (0,40 pts.) Escriu un fragment de codi en assemblador MIPS (de no més de 4 línies) que calculi el valor absolut d'un nombre enter guardat al registre \$t0 i deixi el resultat al registre \$t1

```
move $t1, $t0
bge $t1, $zero, fi
subu $t1, $zero, $t1
fi:
```

COGNOMS:

NOM:

GRUP:

Pregunta 3. (0,90 punts)

Considera la següent subrutina programada en assemblador MIPS:

```
func:      ble    $a0, $a2, etiq1  x <= z
           ble    $a1, $a2, etiq2  y <= z
etiq1:    beq    $a2, $zero, etiq3  z = 0 ; z != 0
etiq2:    li     $v0, 0
           b     etiq4
etiq3:    li     $v0, 1
etiq4:    jr     $ra
```

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

```
ao   al   a2
int func(int x, int y, int z)
{
    int res;
    if ( (( x > z ) && ( y <= z ) ) || ( z != 0 ) )
    {
        res = 0;
    } else
    {
        res = 1;
    }
    return res;
}
```

Pregunta 4. (1,50 punts)

Tradueix la següent funció en llenguatge C a una subrutina MIPS:

```
short s3(unsigned long long *p1) {  
    short v1;  
  
    if (*p1 != 0) /* p1 és un punter a una doble paraula */  
        v1 = 1 + s3(p1+1);  
    else  
        v1 = 0;  
    return v1;  
}
```

\$ra

+4

```
s3 : addiu $sp, $sp, -4  
      sw $ra, 0($sp)  
      lw $t0, 0($a0)  
      lw $t1, 4($a0)  
      or $t0, $t0, $t1  
      beq $t0, $zero, else  
      addiu $a0, $a0, 8  
      jal s3  
      addiu $v0, $v0, 1  
else : li $v0, 0  
fi : lw $ra, 0($sp)  
      addiu $sp, $sp, 4  
      jr $ra
```

COGNOMS:

GRUP:

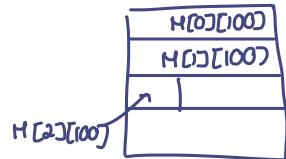
NOM:

Pregunta 5. (0,60 punts)

Donada la següent subrutina en assemblador MIPS:

acces_aleatori:

```
    li      $t0, 400
    mult   $t0, $a1
    mflo   $t0 * i·400
    addu   $t0, $t0, $a0 * @M + i·400 + j·4
    lw     $v0, 840($t0)
    jr     $ra 840 14 210
```



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+2][10];
```

Pregunta 6. (1,90 punts)

Sigui el següent fragment de programa en C, on hi ha una expressió E desconeguda:

```
int M[100][100];
main() {
    int i; /* es guarda en $t0 */
    ...
    for (i=2; i<10; i++)
        E = 0;
}
```

Hem traduït el bucle anterior a MIPS, usant la tècnica d'accés seqüencial i hem obtingut el següent codi, on hi falten els quadres A i B, ja que depenen de quina sigui l'expressió E:

```
la    $t2, A          # inicialitzem el punter
li    $t0, 2            # i=2
li    $t1, 10
for:
    bge   $t0, $t1, fi
    sw    $zero, 0($t2)
    addiu $t2, $t2, B
    addiu $t0, $t0, 1      # i++
    b     for
fi:
```

- a) (1,3 pts) Omple aquesta taula indicant els operands que cal escriure als quadres A i B perquè el programa en MIPS sigui la traducció correcta del programa en C, per a cada una de les possibles expressions d'E (si no es pot aplicar accés seqüencial, escriu "N/A" a les dues caselles).

E	A	B
M[i+1][0]	M + 1200	400
M[2][10*i]	M[2][20] = @M + 2 * 100 * 4 + 20 * 4 M + 880	40
M[2*i+2][10-i]	M[6][8] = @M + 6 * 100 * 4 + 8 * 4 M + 2432	796
M[5][i*i]	M[5][4] = @M + 5 * 100 * 4 + 4 * 4 M + 2016 N/A	20 N/A

- b) (0,60 pts) Escriu una versió del codi MIPS anterior per al cas que E sigui M[i+1][0], aplicant tan sols una optimització: avaluar la condició al final del bucle (convertir-lo en un do-while):

```
la $t2, M+1200
li $t0, 2
li $t1, 10
while: sw $zero, 0($t2)
       addiu $t2, $t2, 400
       addiu $t0, $t0, 1
       blt $t0, $t1, while
fi:
```

Pregunta 8. (1,70 punts)

Un processador P0 disposa de 3 tipus d'instruccions: A, B i C, amb CPIs que es detallen a la taula de sota. Hem creat un disseny millorat de l'anterior, el processador P1, amb idèntic joc d'instruccions, però amb diferents CPI, voltatge i freqüència de rellotge. La taula següent resumeix, per a P0 i P1, els CPI de cada tipus d'instrucció així com els seus voltatges d'alimentació i freqüències de rellotge.

	V (Volts)	f (Ghz)	CPI _A	CPI _B	CPI _C
P0	1	1,5	1	4	5
P1	1,5	2	1	5	5

- a) (1,0 pts.) Sabent que P0 dissipa una potència dinàmica de 20W, calcula la potència dinàmica dissipada (en watts) per P1, suposant que no ha variat la capacitància equivalent del circuit (C) ni el factor d'activitat (α).

$$\text{Potència} = \boxed{1,5^2 \cdot 2 \cdot 10^9 \cdot (20/1^2 \cdot 1,5 \cdot 10^9) \text{ W}}$$

$$P = V^2 \cdot f \cdot \alpha \cdot C$$

$$20 = 1^2 \cdot 1,5 \cdot 10^9 \cdot \alpha \cdot C$$

$$\alpha \cdot C = (20/1^2 \cdot 1,5 \cdot 10^9)$$

- b) (0,70 pts.) Calcula el guany de rendiment (speedup) de P1 respecte de P0 que s'obté en executar un programa de test que conté $80 \cdot 10^9$ instruccions de tipus A, $20 \cdot 10^9$ de tipus B i $4 \cdot 10^9$ de tipus C.

$$\text{Guany} = \boxed{\frac{120}{100} = 1.2}$$

$$\frac{(80 \cdot 1 + 20 \cdot 4 + 4 \cdot 5) \cdot 10^9}{1,5 \cdot 10^9} = \frac{80 + 80 + 20}{1,5}$$

$$= \frac{180}{1,5} = 120$$

$$\frac{(80 \cdot 1 + 20 \cdot 5 + 4 \cdot 5) \cdot 10^9}{2 \cdot 10^9} = \frac{80 + 100 + 20}{2}$$

$$= \frac{200}{2} = 100$$

contestades no es tenen en compte; cada resposta incorrecta resta 0,15 punts; i la puntuació total mínima és 0.

	Afirmació	V	F
1.-	El programa és erroni perque caldria salvar la variable <code>loc2</code> en un registre de tipus segur <code>\$s</code>		X
2.-	El programa és erroni perque caldria salvar la variable <code>loc3</code> en un registre de tipus segur <code>\$s</code>		X
3.-	Al bloc d'activació hi ha 3 bytes d'alineació entre la variable <code>loc1</code> i la <code>loc3</code>	X	
4.-	L'adreça de la variable <code>loc3</code> és el contingut del registre <code>\$sp</code>		X
5.-	El resultat de <code>f3</code> no cal guardar-lo en el registre <code>\$s2</code> , podria ser un registre <code>\$t</code>		X
6.-	La instrucció <code>lb \$t0, 8(\$sp)</code> (ressaltada en negreta) hauria de ser <code>lw \$t0, 8(\$sp)</code>	X	
7.-	Si a i b no s'haguessin salvat en registres de tipus segur <code>\$s</code> , el programa continuaria essent correcte.		X
8.-	La instrucció <code>lb \$t1, 0(\$t1)</code> (ressaltada en negreta) sobra		X
9.-	El bloc d'activació de la subrutina <code>f1</code> ocupa 28 bytes	X	
10.-	Si s'hagués traduït primer la crida a <code>f2</code> i després a <code>f3</code> farien falta menys registres segurs <code>\$s</code>	X	

Pregunta 2. (0,50 punts)

Donades les següents declaracions en C:

```
int w[100]; → @w+ui      ↘ $a1
char fun(char v[1000], int i){   V[i] = @v + i·1
    return v[w[i]];           V[w[i]] = @v + (@w + ui)·i
}
```

Tradueix a llenguatge assemblador del MIPS la funció `fun`.

```
la $t0, w
sll $t1, $a1, 2 * i
addu $t0, $t0, $t1 * w[i]
lw $t2, 0($t0) * primera pos w[i]
addu $t3, $t2, $a0
lb $v0, 0($t3)
jr $ra
```

COGNOMS:

NOM:

GRUP:

Pregunta 3. (1,50 punts) Accés a dades estructurades

Donada la següent funció `foo` en llenguatge C, correctament programada (no cal comprovar si se surt de rang), on `M` es declara com a variable global:

```
char M[100][100]; /* suposarem que està inicialitzada */
void foo() {
    int i; /* ocupa el registre $t0 */
    char result = 0;

    for (i=96; i>=3; i--) {
        if (M[i+3][i-3] > M[i-3][i+3]) {
            result += M[i+3][i-3];
        } else {
            result -= M[i-3][i+3];
        }
    }
    return result;
}
```

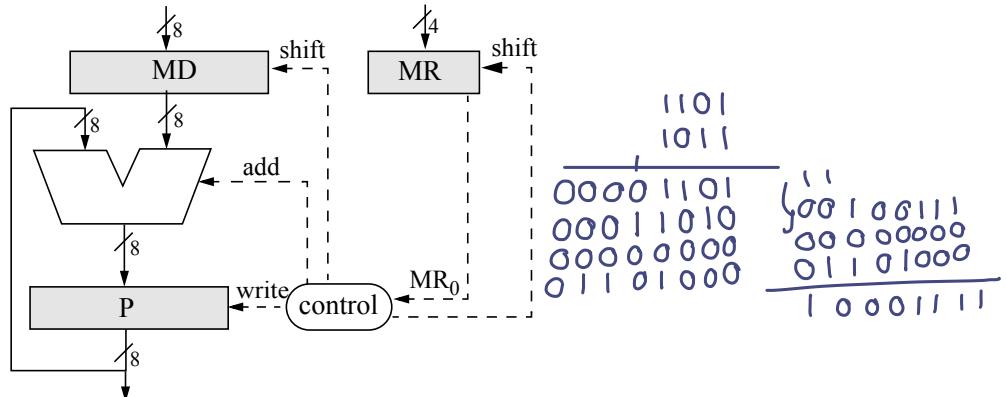
$$\begin{array}{r}
 M[93][99] = @m + 93 \cdot 100 + 99 \\
 M[99][93] = @m + 99 \cdot 100 + 93 \\
 \hline
 & 9993 \\
 & -9399 \\
 \hline
 & 0
 \end{array}$$

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, usant el registre `$t1` com a punter, i que aquest punter s'inicialitza amb l'adreça de l'element `M[96-3][96+3]`:

foo:	la	\$t1, M +	93·100 + 99	# @M[96-3][96+3]
	move	\$v0, \$zero		# result = 0
	li	\$t0, 96		# i = 96
	li	\$t2, 3		
for:	blt	\$t0, \$t2, return		
	lb	\$t4, [] (\$t1)	594	# M[i+3][i-3]
	lb	\$t5, [] (\$t1)	0	# M[i-3][i+3]
	ble	\$t4, \$t5, else		
	addu	\$v0, \$v0, \$t4		# result += M[i+3][i-3]
	b	endfor		
else:	subu	\$v0, \$v0, \$t5		# result -= M[i-3][i+3]
endfor:	addiu	\$t1, \$t1, []	-101	N[92][98]
	addiu	\$t0, \$t0, []	-1	N[93][99]
	b	for		M[-100-1] = -100+-1 = -101
return:	jr	\$ra		# i--

Pregunta 4. (1,50 punts) Aritmètica d'enters i naturals

Sigui el següent diagrama del multiplicador seqüencial de nombres naturals de 4 bits anàleg al que s'ha estudiat durant el curs, el qual calcula el producte amb 8 bits:



- a) (0,75 p) Suposem que amb aquest circuit multipliquem els números binaris de 4 bits 1101 (multiplicand) i 1011 (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, omplint tantes iteracions com facin falta:

iter.	P (Producte)								MD (Multiplicand)								MR (Multiplicador)			
ini	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	1	1	0	1	1
1	0	0	0	0	1	1	0	1	0	0	0	1	1	0	1	0	0	1	0	1
2	0	0	1	0	0	1	1	1	0	0	1	1	0	1	0	0	0	0	1	0
3	0	0	1	0	0	1	1	1	0	1	1	0	1	0	0	0	0	0	0	1
4	1	0	0	0	1	1	1	1	1	1	0	1	0	0	0	0	0	0	0	0

- b) (0,75 p) Suposem que els registres MIPS \$t1 i \$t2 valen \$t1=0x00000010 i \$t2=0xFFFFFFFF. Indica el contingut final en hexadecimal de \$hi i \$lo després d'executar `mult $t1,$t2` i després de `multu $t1,$t2`.

mult	\$t1, \$t2	# \$hi = 0x FFFFFFFFF	\$lo = 0x FFFF FFFF0
multu	\$t1, \$t2	# \$hi = 0x 0000000F	\$lo = 0x FFFF FFFF0

COGNOMS:

GRUP:

NOM:

Pregunta 5. (2,50 punts)

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

```
char a[] = "DECA";
int b = 0;
int *c = &b;
short d[3] = {-14, 43, 5};
long long e[1000];
```

- a) (0,50 p) Tradueix-la al llenguatge assemblador del MIPS

```
.data
a: .ascii "DECA"
b: .word 0
c: .word b
d: .half -14, 43, 5
e: .space 8000
```

$\begin{array}{r} -14 \boxed{2} \\ 0) -7 \boxed{2} \\ \quad\quad\quad 1) -4 \boxed{2} \\ \quad\quad\quad 0) -2 \boxed{2} \\ \quad\quad\quad 0) -1 \end{array}$
 $\begin{array}{r} 5 \boxed{2} \\ 1) \boxed{2} \boxed{2} \\ \quad\quad\quad 0) \end{array}$
 $\begin{array}{r} 43 \boxed{2} \\ 03 \boxed{2} \boxed{2} \\ 1) 01 \boxed{10} \boxed{2} \\ \quad\quad\quad 1) 0) 5 \boxed{2} \\ \quad\quad\quad 1) 2 \boxed{2} \\ \quad\quad\quad 0) \end{array}$
 $\begin{array}{r} 0101 \\ \boxed{5} \end{array}$

- b) (0,50 p) Completa la següent taula amb el contingut de memòria en hexadecimal de les primeres 24 posicions de memòria. 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	44
0x10010001	45
0x10010002	43
0x10010003	41
0x10010004	00
0x10010005	
0x10010006	
0x10010007	

@Memòria	Dada
0x10010008	00
0x10010009	00
0x1001000A	00
0x1001000B	00
0x1001000C	08
0x1001000D	00
0x1001000E	01
0x1001000F	10

@Memòria	Dada
0x10010010	F2
0x10010011	FF
0x10010012	2B
0x10010013	00
0x10010014	05
0x10010015	00
0x10010016	
0x10010017	

- c) (0,50 p) Quin és el valor de \$t0 en hexadecimal, després d'executar el següent fragment de codi?
 $\$t0 = 0x\ 5555555F$

li	\$t1,	0x77777777
li	\$t2,	0x55555555
la	\$t3,	d + 2 \$t3 = 10010012
lh	\$t0,	0(\$t3) \$t0 = 0000002B
or	\$t0,	\$t0, \$t2
xor	\$t0,	\$t0, \$t1

d) (0,50 p) Tradueix a llenguatge assemblador del MIPS la següent sentència en C:

$*c = 18;$

```
li $t0, 18  
la $t1, c  
lw $t2, 0($t1)  
sw $t0, 0($t2)
```

e) (0,50 p) Tradueix a llenguatge assemblador del MIPS la següent sentència en C:

$e[5] = @e + 5 \cdot 8 = @e + 40$

```
la $t0, e  
sw $zero, u0($t0) } long long!  
sw $zero, uu($t0)
```

Pregunta 6. (1 punt) Condicional

Donada la següent sentència escrita en alt nivell en C:

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

Completa 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.

bne \$s0, \$zero, etiq1

bne \$s1, \$zero, etiq2

etiq1: ble \$s1, \$s0, etiq3

 bgt \$s0, \$zero, etiq3

etiq2: move \$s0, \$zero # x=0
 b etiq4

etiq3: li \$s0, 1 # x=1

etiq4:

COGNOMS:

NOM:

GRUP:

Pregunta 7. (1,50 punts) Rendiment i consum

Un processador disposa de 3 tipus d'instruccions (A, B, C), amb diferents CPI. A fi de caracteritzar el seu rendiment i consum hi executem un programa de test. La següent taula especifica el nombre d'instruccions executades de cada tipus, així com el seu CPI.

tipus d'instrucció	CPI	nombre d'instruccions
A	1	$8 \cdot 10^9$
B	4	$3 \cdot 10^9$
C	5	$2 \cdot 10^9$

- a) Suposant que la potència dissipada és $P=60$ W i que la freqüència de rellotge és de 1,2 GHz, calcula el temps d'execució en segons i l'energia consumida en Joules durant l'execució del programa de test.

$$t_{\text{exe}} = \boxed{25} \text{ s}$$

$$E = \boxed{1500} \text{ J}$$

- b) Hem redissenyat el processador, i ara la freqüència és de 1,8 GHz, però les instruccions de tipus B tenen un CPI=6. Suposant que no ha canviat cap més paràmetre arquitectònic o elèctric, calcula el temps d'execució en segons, el guany de rendiment (speedup) i la potència dissipada en Watts durant l'execució del programa de test.

$$t_{\text{exe}} = \boxed{20} \text{ s}$$

$$\text{speedup} = \boxed{1,25}$$

$$P = \boxed{90} \text{ W}$$

COGNOMS:

NOM:

GRUP:

EXAMEN PARCIAL D'EC

23 d'abril de 2015

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 el dia 7 de maig.

Pregunta 1. (1 punt)

Suposem que tenim la següent declaració de variables locals:

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

i que les variables es troben als registres \$t0 (a), \$t1 (b), \$t2(c) i \$t3(d).

Tenim el següent codi en llenguatge C:

```
if ( ($t0 == $t1) && ( ($t2 > $t3) || ($t0 > $t1) ) )  
    { a=b>>3; }  
else  
    { c=c/d; }
```

Ompliu les caselles en blanc per tal que el codi MIPS que hi ha a continuació equivalgui al codi C original.

bne	\$t0, \$t1,	sino	$a == b$
bgtu	\$t2, \$t3,	llavors	$c > d$
ble	\$t0, \$t1,	sino	$a > b$
llavors:			
sra \$t0, \$t1, 3			
b fsi			
sino:			
divu \$t2, \$t3			
mflo			
fsi:			

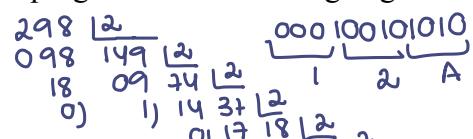
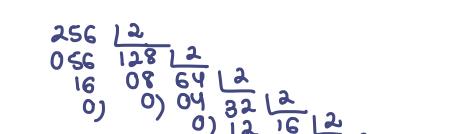
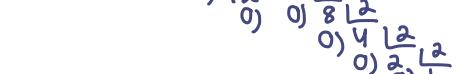
Pregunta 2. (2 punts)

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

```

int a= 298;
int *p=&a;
unsigned short s= 0xFFFF;
unsigned short *ps=&s;
char c[4]={ 'a' , 'b' , 'c' , 'd' } ; /* L'ascii de la lletra 'a' és 0x61 */
char *p2=c;
unsigned short s2= 0x12;
long long f=256;
int b;

```

Hem fet una representació de com quedarà la memòria i ens han sortit els següents valors en hexadecimal:

@Memòria	Dada
0x10010000	2A
0x10010001	01
0x10010002	00
0x10010003	00
0x10010004	00
0x10010005	00
0x10010006	01
0x10010007	10
0x10010008	FF
0x10010009	FF
0x1001000A	00
0x1001000B	00
0x1001000C	08
0x1001000D	00
0x1001000E	01
0x1001000F	10

@Memòria	Dada
0x10010010	61
0x10010011	62
0x10010012	63
0x10010013	64
0x10010014	10
0x10010015	00
0x10010016	01
0x10010017	10
0x10010018	12
0x10010019	00
0x1001001A	00
0x1001001B	00
0x1001001C	00
0x1001001D	01
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

(segueix al següent full)

COGNOMS:

NOM:

GRUP:

(continuació de la pregunta 2, prové del full anterior)

- a) Hi ha un error en la representació que hem fet. Quin és?

La variable longlong + no està alineada a una adreça múltiple de 8.

- b) Tradueix a llenguatge assemblador del MIPS les següents sentències en C

```
c[2] = 'a';  
C[2] = @C + 1·2 = @+2  
li    $t0, 'a'  
la    $t1, C  
sw    $t0, 2($t1)
```

```
if (s2<*ps) {  
    ps=&s2;  
}  
la    $t0, s2  
lhu   $t1, 0($t0)  
la    $t3, ps  
lw    $t3, 0($t1)  
lhu   $t4, 0($t3)  
bgeu $t0,$t4, fi  
sw    $t0, 0($t2)  
fi:
```

Pregunta 3. (2 punts)

Tenim un codi en MIPS que treballa sobre matrius. Hem fet dues versions del codi, una usant accés aleatori (AA) i una altra usant accés seqüencial (AS). El total d'instruccions utilitzades a cada versió les podeu trobar a la següent taula:

	Load / Store	Mult	Salts que salten	Salts que no salten	Altres instruccions
AA	500.000	250.000	250.000	250.000	3.750.000
AS	500.000	0	500.000	500.000	6.000.000

5 cicles 16 cicles 2 cicles 1 cicle 1 cicle

Suposem que a la nostra arquitectura, cada instrucció de memòria costa 5 cicles, cada multiplicació 16 cicles, els salts 1 o 2 cicles depenen de si salta (2) o no salta (1) i la resta d'instruccions 1 cicle.

- a) Quin seria el CPI de cada versió del programa? Quin seria el speed-up de la versió d'accés seqüencial respecte a la d'accés aleatori?

$$CPI_{AA}: 2.2$$

$$CPI_{AS}: 1.33$$

$$\text{Speed-up: } 1.1$$

$$\text{speedup} = \frac{t_{exec}}{t_{exe}} = \frac{CPI \cdot n^{\circ}ins \cdot 1/f}{CPI \cdot n^{\circ}ins \cdot 1/f} = \frac{CPI_{AA} \cdot n^{\circ}ins}{CPI_{AS} \cdot n^{\circ}ins}$$

- b) Quant de temps, en segons, triga en executar-se cada programa si el rellotge va a una freqüència de 2GHz?

$$\text{Temps}_{AA}: 5.5 \text{ ms}$$

$$\text{Temps}_{AS}: 5 \text{ ms}$$

- c) Si el processador dissipa 3 watts (Joules per segon) sigui quina sigui la instrucció executada, quin ha estat el consum total del processador en Joules de cada versió del programa?

$$\text{Consum}_{AA}: 16.5 \text{ mj}$$

$$\text{Consum}_{AS}: 15 \text{ mj}$$

- d) Ens diuen que es pot dissenyar una arquitectura alternativa on podríem reduir el temps de cicle de les operacions de multiplicació. Quin és el màxim nombre de cicles de multiplicació admissible per a la nova arquitectura per tal que el codi AA sigui al menys igual de ràpid que el codi AS?

12 cicles

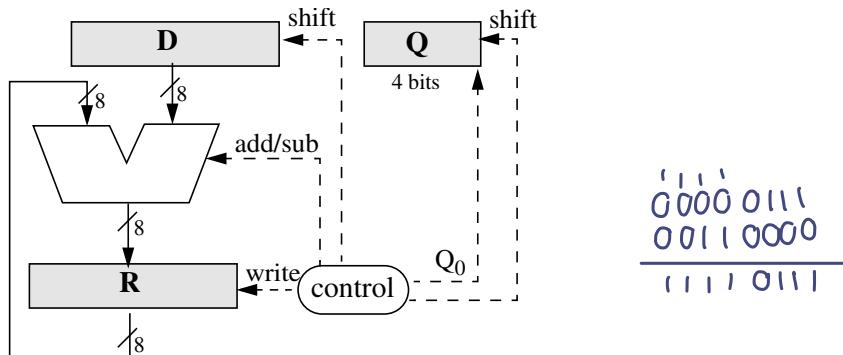
COGNOMS:

NOM:

GRUP:

Pregunta 4. (1 punt)

- a) Sigui el següent circuit seqüencial per a la divisió de números naturals de 4 bits, anàleg a l'estudiat a classe per a 32 bits:



Suposem que volem calcular la divisió (en base 2): 0111/0011.

Ompla la següent taula indicant quin és el valor final dels registres R, D i Q al final de cada iteració de l'algorithm que controla el circuit (omple tantes files com iteracions tingui l'algorisme).

iteració	R (Dividend/Residu)	D (Divisor)	Q (Quocient)
valor inicial	0 0 0 0 0 1 1 1 1	0 0 1 1 0 0 0 0	0 0 0 0
1	0 0 0 0 0 1 1 1	0 0 0 1 1 0 0 0	0 0 0 0
2	0 0 0 0 0 1 1 1	0 0 0 0 1 1 0 0	0 0 0 0
3	0 0 0 0 0 0 0 1	0 0 0 0 0 1 1 0	0 0 0 1
4	0 0 0 0 0 0 0 1	0 0 0 0 0 0 1 1	0 0 1 0

- b) Suposem que els registres MIPS \$t1 i \$t2 valen \$t1=0x12345678 i \$t2=0xFFFFFFFF. Indica el contingut final en hexadecimal de \$hi i \$lo després d'executar divu \$t1,\$t2 i després de div \$t1,\$t2.

divu: \$hi=	0x 12345678	\$lo=	0x 00000000
div: \$hi=	0x 00000000	\$lo=	0x FDCBA988

Pregunta 5. (2 punts)

La següent taula mostra una declaració d'una subrutina en alt nivell i la seva traducció a llençatge MIPS. El codi MIPS pot contenir errors, tot i que no necessàriament.

Declaració en alt nivell:	Traducció a MIPS:
<pre> int e2(short *r, int *s); int e1(char *b) { int x=0,y; char v1[5]; short v2[4]; while(*b != v1[x]) { x = x + e2(v2, &y); } return x; } </pre>	<pre> e1: addiu \$sp, \$sp, -32 sw \$s1, 20(\$sp) sw \$s0, 24(\$sp) sw \$ra, 28(\$sp) move \$s0, \$a0 #b li \$s1, 0 #x buc: addu \$t1,\$sp,\$s1 lb \$t1,0(\$t1) lb \$t2,0(\$s0) beq \$t1,\$t2,fibuc addiu \$a0, \$sp, 12 addiu \$a1, \$sp, 0 jal e2 addu \$s1, \$s1, \$v0 b buc </pre> <p style="text-align: right;"> </p>

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). Cada resposta correcta suma 0,2 punts; les respostes no contestades no es tenen en compte; cada resposta incorrecta resta 0,2 punts; i la puntuació total mínima és 0.

	Afirmació	V	F
1.-	A la pila hi ha 3 bytes d'alignació	X	
2.-	Al \$sp cal restar-li 36 en lloc de 32		X
3.-	La instrucció lb \$t2,0(\$s0) sobra perquè la variable b ja la tenim a \$s0		X
4.-	La instrucció lb \$t1,0(\$t1) ha de ser lb \$t1,4(\$t1)	X	
5.-	La instrucció addiu \$a0,\$sp,12 ha de ser addiu \$a0, \$sp, 10	X	
6.-	La instrucció addiu \$a1,\$sp,0 ha de ser la \$a1,y		X
7.-	El resultat d'e2 no cal acumular-lo a \$s1 perquè es pot acumular al propi \$v0		X
8.-	El resultat que torna e1 no és el demanat	X	
9.-	La instrucció beq \$t1,\$t2,fibuc no es pot fer amb registres \$t, sinó amb \$s		X
10.-	La variable y es pot emmagatzemar en un registre segur \$s		X

COGNOMS:

NOM:

GRUP:

Pregunta 6. (2 punts)

Donat el següent programa en alt nivell correcte (no cal comprovar si se surt de rang), on m i v són declarades com a variables globals:

```

unsigned int M[100][100]; /* suposarem que està inicialitzada */
int V[100];
main() {
    int i;
    for (i=5; i<100; i+=2)
        V[M[i][i]] = M[i-1][i];
}

```

$M[2][2] = @m + 4 \cdot 2 \cdot 100 + 2 \cdot 4 = 808$
 $M[4][5] = @m + 4 \cdot 4 \cdot 100 + 4 \cdot 5 = @m + 1620$
 $M[5][5] = @m + 4 \cdot 5 \cdot 100 + 4 \cdot 5 = @m + 2020$

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**, i que s'hi ha aplicat l'optimització de conversió d'un bucle `for` en un `do_while`, i s'ha eliminat el comptador del bucle comprovant que no s'accedeixi a una adreça més enllà del darrer element a recórrer.

$$M[100][0] = @m + 4 \cdot 100 \cdot 100$$

```

la      $t0,M+ 2020
        # El punter $t0 apunta
        # als elements M[i][i]
la      $t1,V

la      $t2,M+ 40000
        # El punter $t2 apunta després de l'últim
        # element d'M
bucdo:
lw      $t3, -400 ($t0)  M[i-1][i]
lw      $t4, 0 ($t0)  M[i][i]
sll   $t4, $t4, 2
addu $t5, $t1, $t4
sw      $t3, 0 ($t5)  $t5 = V[M[i][i]]
addiu $t0,$t0, 808
bltu $t0,$t2,bucdo

```

Com és l'accés al vector v ? Per què s'ha de fer així?

Aleatori, com tenim una matriu a l'index no podem assegurar que ni hagi un stride constant.

COGNOMS:

GRUP:

NOM:

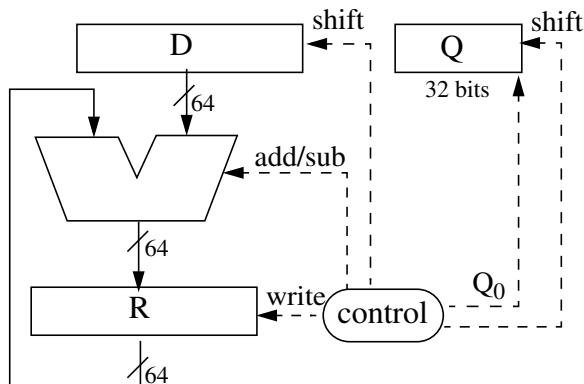
EXAMEN PARCIAL D'EC

3 de novembre de 2014

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 110 minuts. Les notes, la solució i el procediment de revisió es publicaran al Racó el dia 13 de novembre.

Pregunta 1. (1 punt)

Donat el següent diagrama que representa el divisor seqüencial de nombres naturals de 32 bits estudiad a classe i que realitza la divisió X/Y, calculant alhora el quocient i el residu, completa l'algorisme iteratiu que en descriu el funcionament cicle a cicle:



```

D63:32 = Y ; D31:0 = 0 ;
R63:32 = 0 ; R31:0 = X ;
Q = 0 ;
for (i = 1, i <= 32, i++) {
    D = D >> 1;
    Q = Q << 1;
    if (R >= D) {
        R = R - D;
        Q = Q | 0x1;
    }
}

```

Pregunta 2. (1,5 punts)

Considera el següent programa en assemblador MIPS, el qual suma els 4 elements de V:

```

.data
V:      .word 1,2,3,4          V[1] = @V+4
sum:    .word 0
.text
main:
    li    $t0,0   i           #1   +1
    li    $t1,4   v.size()    #1   +1
    li    $t5,0   "suma"     #1   +1
    la    $t4,V   @V         #2   +2
for:
    bge  $t0,$t1,fifor    #2   0>4   +(2*4) 8   +1
    sll  $t2,$t0,2
    addu $t3,$t4,$t2
    lw   $t3,0($t3) +4
    addu $t5,$t5,$t3
    addiu $t0,$t0,1
    b for
fifor:
    la $t0,suma          #2
    sw $t5,0($t0)  suma = "suma"  +1
    +

```

V[1] = @V+4
@V+4 +4
V[1] +4
suma = suma + V[i] +4
#1 +1 +4

a) (0,5 punts) Tradueix al llenguatge C el codi anterior

```

main()
{
    int i, aux; /* ocupen els registres $t0, $t5 */
    aux = 0;
    for (i=0; i<4; i++) {
        aux = aux + V[i];
    }
    suma = aux;
}

```

Als següents apartats volem calcular el temps d'execució del programa. Tingues en compte que:

- En el codi anterior s'han escrit en negreta les línies corresponents a macros i com a comentari d'aquestes línies hi ha el nombre d'instruccions amb què s'expandeix la macro.
- Les instruccions lw/sw s'executen en 2 cicles de rellotge, i tota la resta en 1 cicle.
- El processador funciona a una freqüència de rellotge de 1 GHz.

b) (0,5 punts) Indica quantes instruccions s'executen de cada tipus en el programa complet:

Nombre de lw i sw =

5

Nombre de la resta =

37

c) (0,5 punts) Calcula el nombre de cicles que triga el programa i el temps d'execució (t_{exe}), en segons:

Nombre de cicles =

$37 \cdot 1 + 5 \cdot 2 = 47$

$t_{exe} =$

$47 \cdot 10^{-9}$

$$CP1 = \frac{37 \cdot 1 + 5 \cdot 2}{42} = \frac{47}{42}$$

$$t_{exe} = \frac{47}{42} \cdot 42 \cdot \frac{1}{1 \cdot 10^9} = \frac{47}{1 \cdot 10^9}$$

COGNOMS:**GRUP:****NOM:**

Pregunta 3. (2 punts)

La següent taula mostra una declaració d'una subrutina en alt nivell i la seva traducció a llençatge MIPS. El codi MIPS pot contenir errors, tot i que no necessàriament.

Declaració en alt nivell:

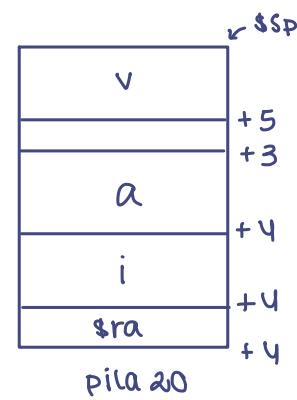
```
int subr1(char a[], int *b) {
    char v[5];
    int i, j;
    for (i = 0; i < 5; i++) {
        j = subr2(&v[i], &a[i]);
        *b = j;
    }
    return j;
}
```

Traducció a MIPS:

```
subr1:
    addiu $sp, $sp, -20
    sw $s0, 8($sp)
    sw $s1, 12($sp)
    sw $ra, 16($sp)
    addiu $sp, $sp, 20
    jr $ra

for:
    li $t0, 5
    bge $t2, $t0, fifor
    addu $a0, $sp, $t2
    addu $a1, $s0, $t2
    jal subr2
    sw $v0, 0($s1)
    addiu $t2, $t2, 1
    b for
fifor:
```

```
lw $s0, 8($sp)
lw $s1, 12($sp)
lw $ra, 16($sp)
addiu $sp, $sp, 20
jr $ra
```



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). Cada resposta correcta suma 0,2 punts; les respostes no contestades no es tenen en compte; cada resposta incorrecta resta 0,2 punts; i la puntuació total mínima és 0.

	Afirmació	V	F
1.-	Caldria salvar la variable <i>j</i> en un registre de tipus segur		X
2.-	Caldria salvar la variable <i>i</i> en un registre de tipus segur	X	
3.-	La traducció de <i>*b = j</i> és errònia ja que hi falta un <i>lw</i>	X	
4.-	No s'hauria de salvar el paràmetre <i>b</i> en un registre de tipus segur		X
5.-	Si tot fos correcte el bloc d'activació de subr1 ocuparia 24 bytes i no 20	X	
6.-	Si tot fos correcte el bloc d'activació de subr1 ocuparia 28 bytes i no 20		X
7.-	Tal i com s'ha traduït el programa funcionaria bé si subr2 no modifiqués <i>\$t2</i>	X	
8.-	La traducció és correcta		X
9.-	No s'hauria de guardar el vector <i>v</i> a la pila		X
10.-	No caldria salvar <i>\$ra</i> a la pila		X

COGNOMS:

NOM:

GRUP:

Pregunta 4. (2 punts)

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

```

int a= -12;           ,0x62
char b[2]={'d','b'};    /* el codi ascii de la 'a' és el 0x61 */
short c=1;             ,0x64
long long d=0xF0A40305;
→short e[3][2]={{-1,4},{2,-3},{0,7}};
unsigned long long f=32;

```

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

a: .word -12
b: .byte 'd', 'b'
c: .half 1
d: .dword 0xF0A40305
e: .half -1, 4, 2, -3, 0, 7
f: .dword 32

$12 \rightarrow 1100$
 $1100 \rightarrow \begin{array}{r} 0011 \\ +0001 \\ \hline 1110100 \end{array}$
 $32 \quad | \quad 2$
 $12 \quad | \quad 16 \quad | \quad 2$
 $0) \quad 0) \quad 8 \quad | \quad 2$
 $0) \quad 0) \quad 4 \quad | \quad 2$
 $0) \quad 0) \quad 2 \quad | \quad 2$
 $0) \quad 0) \quad 1 \quad | \quad 1$
 $0) \quad 0) \quad 0 \quad | \quad 0$
 $00100000 \quad 0$
 $2 \quad 2 \quad 2 \quad 2 \quad 2 \quad 2 \quad 2 \quad 0$

- b) (0,5 punts) Omple la següent taula amb el contingut de memòria en hexadecimal. Les variables s'emmagatzem 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	F4	0x10010010	FF	0x10010020	20	0x10010030	
0x10010001	FF	0x10010011	FF	0x10010021	00	0x10010031	
0x10010002	FF	0x10010012	04	0x10010022	00	0x10010032	
0x10010003	FF	0x10010013	00	0x10010023	00	0x10010033	
0x10010004	64	0x10010014	02	0x10010024	00	0x10010034	
0x10010005	62	0x10010015	00	0x10010025	00	0x10010035	
0x10010006	01	0x10010016	FD	0x10010026	00	0x10010036	
0x10010007	00	0x10010017	FF	0x10010027	00	0x10010037	
0x10010008	05	0x10010018	00	0x10010028		0x10010038	
0x10010009	03	0x10010019	00	0x10010029		0x10010039	
10 0x1001000A	A4	0x1001001A	07	0x1001002A		0x1001003A	
11 0x1001000B	F0	0x1001001B	00	0x1001002B		0x1001003B	
12 0x1001000C	00	0x1001001C		0x1001002C		0x1001003C	
0x1001000D	00	0x1001001D		0x1001002D		0x1001003D	
0x1001000E	00	0x1001001E		0x1001002E		0x1001003E	
0x1001000F	00	0x1001001F		0x1001002F		0x1001003F	

c) (0,5 punts) Tradueix a una única sentència d'alt nivell (C), el següent codi:

@c

la	\$t0,c	1	$e[i][j] = @e + i \cdot 2 \cdot 2 + j \cdot 2^2 = @e + 4i + 2j$
lh	\$t1,0(\$t0)		
addiu	\$t1,\$t1,1	2	
sll	\$t1,\$t1,1	C+1	
la	\$t2,e		
addu	\$t2,\$t2,\$t1		
sh	\$zero,0(\$t2)		

e[0][c+1] = 0

d) (0,5 punts) Quin és el valor final del registre \$t2, en hexadecimal, després d'executar el següent fragment de codi?

la	\$t0,d	0x 10010008	0x0000 00A4
lbu	\$t1,2(\$t0)	0x0000 00A4	0x 0000 8888
li	\$t2,0x00008888		1010 0100
or	\$t2,\$t1,\$t2		1000 1000
sll	\$t2,\$t2,4		1010 1100

→ 0x000088A0

\$t2 = **0x000088A0**

Pregunta 5. (1,5 punts)

Donada la següent sentència escrita en alt nivell en C:

```
if ((x%8!=0) || ((y<=x) && (y>0)))
    x=0;
else
    x=1;
```

Completa 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 unsigned int i estan inicialitzades i guardades als registres \$s0 i \$s1, respectivament.

Tradueix aquí x%8 (màxim 2 instruccions) i deixa el resultat a \$t0:

andi \$t0, \$s0, 7

bne	\$t0, \$zero,	etiq2	
bgtu	\$s1, \$s0,	etiq3	
etiq1:	bleu	\$s1, \$zero,	etiq3
etiq2:	move	\$s0, \$zero	# x=0
	b	etiq4	
etiq3:	li	\$s0,1	# x=1
etiq4:			

COGNOMS:

GRUP:

NOM:

Pregunta 6. (2 punts)

Donat el següent programa en alt nivell, on M és una matriu declarada com a variable global i N és una constant ($N < 100$):

```
int M[N][N];      /* suposarem que està inicialitzada */
main() {           N[N-1][N-1] = @m + N-1 · N · 4 + N-1 · 4 = @m + 4N2 - 4N + 4N - 4 = @m + 4N2 - 4
    int i;
    for (i=0; i<N; i++)
        M[N-1-i][N-1-i] = M[i][N-1-i];
}
```

$M[0][N-1] = @m + 4N - 4$

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 amb dos pointers**, i que s'hi ha aplicat l'optimització de conversió d'un bucle `for` en un `do_while`. Deixa el resultat en funció d' N , si cal.

la	\$t0,M +	$4N^2 - 4$	# El punter \$t0 apunta # als elements M[N-1-i][N-1-i]
la	\$t1,M +	$4N - 4$	# El punter \$t1 apunta # als elements M[i][N-1-i]
li	\$t2,0		# La variable i es guarda en \$t2
li	\$t3,N		# N és una constant
buc:			
lw	\$t7,	0	$M[-1][N-2]$ $M[0][N-1]$
sw	\$t7,	0	$M[-1][-1] = -UN - 4$
addiu	\$t0,\$t0,	-UN - 4	$M[1][N-2]$ $M[0][N-1]$
addiu	\$t1,\$t1,	$UN - 4$	$M[1][-1] = UN - 4$
addiu	\$t2,\$t2,1		
blt	\$t2,\$t3,buc		

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 dinàmica dissipada per la combinació B en watts?

$$P = \alpha \cdot C \cdot V^2 \cdot f = \frac{80 \cdot 2,3^2 \cdot 2 \cdot 10^9}{2^2 \cdot 1,25 \cdot 10^9} = 169,2$$

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

$$\text{Speedup} = t_{\text{exeA}} / t_{\text{exeB}} = \frac{f_B}{f_A} = 1.6$$

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?

$$t_{\text{exeA}} = \frac{n \cdot \text{CPI}}{f_A} = 38.4s$$

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

```
a: .half -6,8,3
b: word 0
c: .dword -8
d: .byte 0xC, 0xA, 0xF, 0xE
e: .asciiz "CAFE"
f: .half a+1
```

A 0x41
B 0x42
C 0x43
D 0x45
E 0x46 F 0x47

6 | 2
0) 3 | 2
|) 1 | 2
) 0
0 110
1001
0001
1010

8 | 2
0) 4 | 2
|) 2 | 2
) 1
1000
b | 1 | 1
+ 0001

11111000

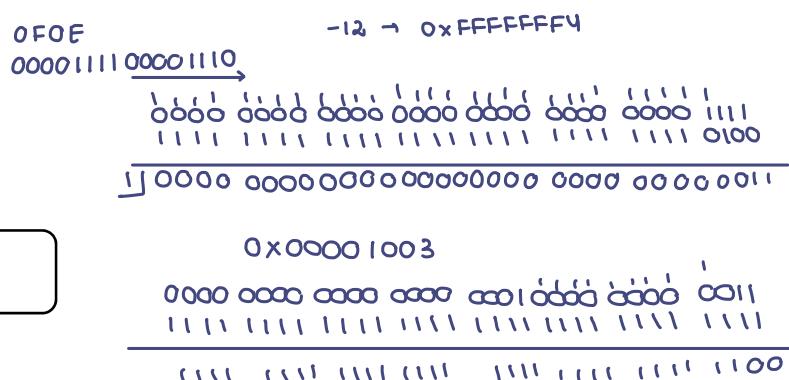
Omple la següent taula amb el contingut de memòria **en hexadecimal** (sense el prefix 0x). Tingues en compte que el codi ASCII de la 'A' és el 0x41. Les variables s'emmagatzem a partir de l'adreça 0x10010000. Les posicions de memòria sense inicialitzar es deixen en blanc.

@Memòria	Dada	@Memòria	Dada	@Memòria	Dada	@Memòria	Dada	@Memòria	Dada
0x10010000	FA	0x10010008	00	0x10010010 ¹⁶	F8	0x10010018 ¹⁷	0C	0x10010020 ¹⁸	00
0x10010001	FF	0x10010009	00	0x10010011 ¹⁹	FF	0x10010019 ²⁰	0A	0x10010021 ²¹	
0x10010002	08	0x1001000A	00	0x10010012 ²²	FF	0x1001001A ²³	0F	0x10010022 ²⁴	
0x10010003	00	0x1001000B	00	0x10010013 ²⁵	FF	0x1001001B ²⁶	0E	0x10010023 ²⁷	
0x10010004	03	0x1001000C ²⁸		0x10010014 ²⁹	FF	0x1001001C ³⁰	U3	0x10010024 ³¹	02
0x10010005	00	0x1001000D ³²		0x10010015 ³³	FF	0x1001001D ³⁴	U1	0x10010025 ³⁵	00
0x10010006		0x1001000E ³⁶		0x10010016 ³⁷	FF	0x1001001E ³⁸	U7	0x10010026 ³⁹	01
0x10010007		0x1001000F ⁴⁰		0x10010017 ⁴¹	FF	0x1001001F ⁴²	U6	0x10010027 ⁴³	10

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

```
la    $t1, d  0x 10010018
lh    $t1, 2($t1) 0x 0FOE
sra   $t1, $t1, 8 0x 0000000F
addiu $t1, $t1, -12 0x 00000003
lui   $t2, 0x1001 0x 00001001
or    $t2, $t2, $t1
lb    $t0, -1($t2)
```

\$t0 = 0x FFFFFFFF



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		et4
et1:	beq	\$a1, \$zero,	et4
et2:	move	\$a0, \$zero,	et4
et3:	b	\$a0, \$zero	et8
et4:	nor	\$t2, \$a0,	\$a0
et5:	xori	\$t2, \$t2,	0x 1111
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;
    for (i=0; i<10; i=i+1){
        B[i*i][5]=A[9-i][i];
    }
}
```

$i = 0$
 $B[0][5] = @B + 4 \cdot 8 \cdot 0 + 4 \cdot 5 = @B + 20$
 $A[9][0] = @A + 4 \cdot 9 \cdot 10 + 0 = @A + 360$
 $\underline{A[8][1]}$
 $\underline{B[1]} = 4 \cdot 10 \cdot -1 + 4 = -40 + 4 = -36$

/* ocupa el registre \$t0 */

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.

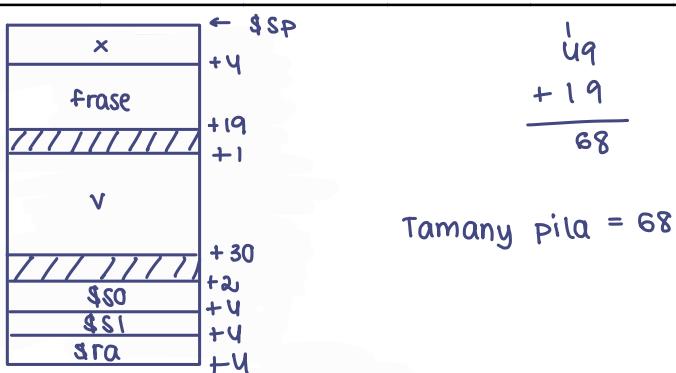
```
main: li      $t0, 0          # i = 0
      la      $t1, A +
      la      $t2, B
      li      $t3, 10    i < 10
      b       cond
do:   lw      $t4, 0($t1) A[9][0]
      mult   $t0, $t0    i*i
      mflo   $t5
      sll    $t5, $t5, 5
      addu   $t5, $t2, $t5  @B + __
      sw      $t4, 20    ($t5)
      addiu  $t1, $t1, -36
      addiu  $t0, $t0, 1
cond: blt    $t0, $t3, do
      jr      $ra
```

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.

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

Tradueix la següent sentència:

$frase[*j] = W[k];$ $frase[i] = @frase + 1 \cdot i$
 $W[i] = @W + 1 \cdot i$ $i \quad j$
 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 # @W + k
lb $t0, 0($t0) # $t0 = W[k]
lw $t1, 0($s1) # @j
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		
slt	\$v0, \$t0, \$t2		

fi:

COGNOMS:

NOM:

GRUP:

EXAMEN PARCIAL D'EC

27 d'abril de 2017

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 9 de maig.

Pregunta 1. (1,5 punts)

Donada la següent funció `foo` en llenguatge C, correctament programada, on `M` es declara com a variable global:

$$M[50] = @m + 2 \cdot 100 \cdot 51 + 50 \cdot 2 = @m + 10300$$

$$\text{short } M[100][100]; \quad M[49][50] = @m + 2 \cdot 100 \cdot 49 + 50 \cdot 2 = @m + 9900$$

```
void foo() {
    int i; /* ocupa el registre $t0 */
    for (i=50; i>=2; i=i-2) {
        M[i+1][100-i] = M[i-1][100-i];
    }
}
```

$M[47][52] - M[49][50]$
 $\underline{M[-2][2]} = 2 \cdot 100 \cdot -2 + 2 \cdot 2$
 $= -400 + 6 = -396$

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, usant el registre `$t1` com a punter, i que aquest punter s'inicialitza amb l'adreça de l'element `M[49][50]`:

<pre> foo: la \$t1, M + li \$t0, 50 li \$t2, 2 for: blt \$t0, \$t2, endfor lh \$t4, [](\$t1) # M[i-1][100-i] sh \$t4, [](\$t1) # M[i+1][100-i]=... addiu \$t1, \$t1, -396 addiu \$t0, \$t0, -2 endfor: jr \$ra </pre>	$ \begin{array}{r} 10000 \\ - 9000 \\ \hline 1000 \\ - 900 \\ \hline 100 \\ + 300 \\ \hline 400 \end{array} $
--	--

Pregunta 2 (1 punt)

$$\begin{aligned} \text{suma}[i] &= @\text{suma} + 4 \cdot i \\ \text{mat}[j][i] &= @\text{mat} + 4 \cdot j \cdot N + 4 \cdot i \end{aligned}$$

Considera una funció següent que retorna al vector `suma` la suma de cada columna de la matriu `mat`:

```
void suma_per_columnes(int mat[M][N], int suma[N]) {
    int i;
    for (i = 0; i < N; ++i) {
        suma[i] = 0;
        for (int j = 0; j < M; ++j) {
            suma[i] += mat[j][i];
        }
    }
}
```

$\text{mat}[0][0] = @\text{mat}$
 $\text{mat}[1][0] = @\text{mat} + 4 \cdot N + 4 \cdot 0 = @\text{mat} + 4N$
 $\text{mat}[0][1]$
 $\text{mat}[N][0]$
 $\text{mat}[M][1] = 4N \cdot (-M) + 4 = -4NM + 4$

Completa el següent codi en MIPS per a que sigui una traducció de la subrutina en C usant la tècnica d'accés seqüencial (deixa les respostes en funció de `N` i `M`):

```
suma_per_columnes:
    addiu      $t0, $a0,
    addiu      $t1, $a1,
    li         $t2, 0
    li         $t3, N

buc1:
    bge      $t2, $t3, fibuc1  i < N
    li         $t4, 0
    li         $t5, 0
    li         $t6, M

buc2:
    bge      $t5, $t6, fibuc2  j < M
    lw         $t7, 0($t0)
    addu      $t4, $t4, $t7
    addiu      $t0, $t0,
    addiu      $t5, $t5,
    b         buc2               +1

fibuc2:
    sw         $t4, 0($t1)
    addiu      $t0, $t0,
    addiu      $t1, $t1,
    addiu      $t2, $t2,
    b         buc1               # next pos suma

fibuc1:
    jr      $ra
```

0
0
4N
1
-4NM + 4
4
1

Cognoms: Nom:
DNI:

Pregunta 3 (1 punt)

Donades les següents declaracions de variables globals en assemblador MIPS, que s'ubiquen a memòria a partir de l'adreça 0x10010000:

Omple la següent taula amb el contingut de memòria **en hexadecimal**. Les variables s'emmagatzem a partir de l'adreça 0x10010000. Deixa EN BLANC les posicions de memòria sense inicialitzar.

@Memòria	Dada	@Memòria	Dada	@Memòria	Dada	@Memòria	Dada
0x10010000	01	0x10010008	34	0x10010010	35	0x10010018	08
0x10010001	00	0x10010009	30	0x10010011	34	0x10010019	00
0x10010002	00	0x1001000A	* 33	0x10010012	33	0x1001001A	01
0x10010003	00	0x1001000B	00	0x10010013		0x1001001B	10
0x10010004	02	0x1001000C	00	0x10010014	00	0x1001001C	FF
0x10010005	00	0x1001000D	00	0x10010015	00	0x1001001D	FF
0x10010006	00	0x1001000E	80	0x10010016		0x1001001E	FE
0x10010007	00	0x1001000F	3F	0x10010017		0x1001001F	FF

Diges el contingut final en hexadecimal a \$t0 després d'executar el codi següent amb les dades declarades a l'apartat anterior:

```

la    $t1, F $t1 = 0x10010018
lw    $t1, 0($t1) $t1 = 0x10010008
lb    $t1, 2($t1) $t1 = 0x0000 0033
sll   $t1, $t1, 1 $t1 = 0x000C 0066
andi $t0, $t1, 0xF0F

```

0 0011 00110 ← 1
 6 6

0000	0000	0110	0110
0000	1111	0000	1111
<hr/>			
0000	0000	0000	0110

Diges el contingut final en hexadecimal a \$t1 després d'executar el codi següent:

```
li      $t0, 0xABCDABCD  
sra    $t1, $t0, 9  
slt    $t1, $t1, $t0
```

Yardwork
F E R F E 6 R 5

`0xFFFFSEGDS < 0xABCDABCD ? NO!`

\$t1 = 0x0000 0000

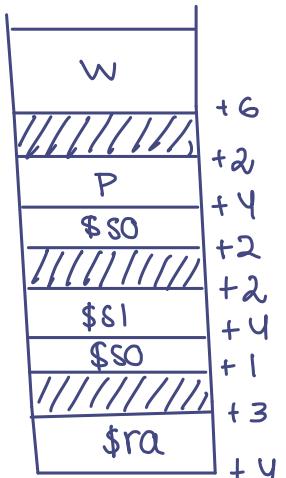
Pregunta 4 (1.5 punts)

Considera les següents declaracions de funcions en C:

```

short f(int *a, short b, char *c);
short examen(short *x, int y[], char *z) {
    short w[3];
    int p;
    ↙ vector
    w[0] = f(&p, *(x+1), z);
    w[1] = f(y+2, *x, z);
    w[2] = 3;
    ↙ Punter
    &P, *(x+1), z
    ↙ w[i] = @w+2·i
    return w[0] + w[1] + w[2];
}

```



Per a totes les variables i arguments de la rutina examen, indica si els guardaries a la pila, en registres segurs o en registres temporals. Justifica la teva resposta.

- * w i p es guardarien en una pila ja que w es un vector i p un punter del que demanen l'adreça
- * x, y, z es guarden a registres segurs ja que hem de preservar el seu valor
- * La resta poden estar en registres temporals

Tradueix la subrutina examen a MIPS, seguint les decisions de l'apartat anterior.

examen :

```

addiu $sp, $sp, -28
sw $so, 12($sp)
sw $s1, 16($sp)
sw $s2, 20($sp)
sw $ra, 24($sp)
move $so, $a0
move $s1, $a1
move $s2, $a2

addiu $a0, $sp, 8 # $a0 = &P
lh $a1, 2($so) # $a1 = *(x+1)
jal f
sh $v0, 0($sp)

```

```

addiu $a0, $a1, 8 # $a0 = y+2
lh $a1, 0($so) # $a1 = x
move $a2, $s2
jal f

```

\$v0 = w[1]

```

addiu $v0, $v0, 3 # w[1]+w[2]
lh $t0, 0($sp) # w[0]
addiu $v0, $v0, $t0 # w[0]+w[1]+w[2]

lw $so, 12($sp)
lw $s1, 16($sp)
lw $s2, 20($sp)
lw $ra, 24($sp)
addiu $sp, $sp, 28
jr $ra

```

Pregunta 5 (1 punt)

Escriu un codi en assemblador de MIPS, **sense cap instrucció de salt**, per calcular el producte de dos números naturals representats en 32 bits i emmagatzemats a \$a0 i \$a1. El resultat cal deixar-lo a \$v0, que també s'haurà de posar a zero si el producte no es pot representar en 32 bits.

```

multu $a0, $a1
mflo $v0
mfhi $t0
slli $t0, $t0, 1
subu $t0, $zero, $t0
and $v0, $v0, $t0

```

Pregunta 6 (1 punt)

Posa una X al costat de cada una de les següents afirmacions (a la columna V si és Verdadera o a la columna F si és Falsa). Suposem en tots els casos que es fa referència a un processador MIPS com l'estudiat a classe. Cada resposta correcta suma 0,1 punts; les respostes no contestades no es tenen en compte; cada resposta incorrecta resta 0,1 punts; i la puntuació total mínima és 0.

Afirmació	V	F
En un processador amb adreces de 32 bits, una cache associativa de 4 vies, d'1MB i blocs de 32 bytes, s'han de dedicar 14 bits a etiqueta (TAG), 13 a número d'entrada (conjunt) i 5 a desplaçament.	X	
Quan es produeix una interrupció s'atura la instrucció en marxa, i quan s'ha acabat el servei a la interrupció es torna a llançar l'esmentada instrucció.		X
En un sistema de paginació una fallada d'escriptura a l'espai de memòria virtual fa que ens portem la pàgina que falla de la memòria principal a l'àrea de swap, i escrivim la dada modificada tant a l'àrea de swap com a la memòria principal.		X
Un processador sense TLB pot suportar memòria virtual.	X	
En un sistema MIPS es pot produir més d'una excepció per fallada de TLB en l'execució d'una mateixa instrucció.	X	
Quan es produeix una excepció o interrupció, el bit EXL canvia per prohibir les interrupcions, indicant al mateix temps que estem en mode usuari.		X
En les memòries cache que utilitzen escriptura immediata s'ha de posar el dirty bit a 1 només quan hi ha un encert d'escriptura.		X
El temps mitjà d'accés a memòria de les instruccions load i store en un computador que disposa de memòria virtual amb TLB, normalment és superior al temps d'accés de la memòria principal.		X
L'excepció per accés no alineat a memòria pot ser inhibida a través del camp Interrupt Mask.		X
La diferència entre les instruccions MIPS add/addu radica en què addu pot generar una excepció per sobreeiximent (overflow) a la suma d'enters mentre que add mai pot generar cap excepció.		X

$$2^3 \cdot 2^{10} = 2^{13}$$

Pregunta 7 (1,50 punts)

Suposem que tenim un processador de 32 bits amb una memòria cache de dades de 8KB associativa per conjunts de 2 vies, on cada bloc té 32 bytes, i que es segueix l'algorisme de reemplaçament LRU.

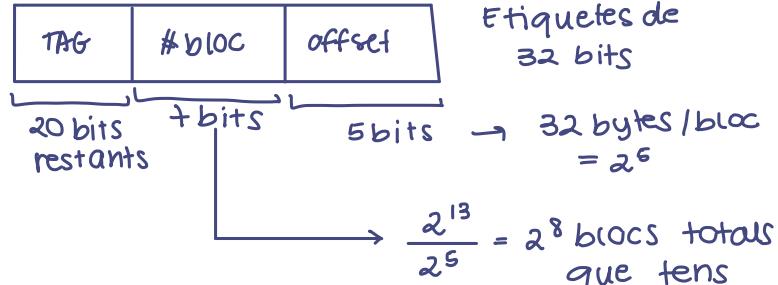
→ 2^6 bytes

Calcula el nombre de fallades de la cache en els accessos a cada vector en executar el següent programa, suposant que la cache té la política d'**escriptura immediata sense assignació** (write-through, no-write-allocate), i que la memòria cache és inicialment buida.

```
int A[1024], B[1024], C[1024];

void main() {
    int i;
    for (i=0; i<1024; i++)
        A[i]=B[i]+C[i];
}
```

SW LW LW



Fallades A = 1024 → sempre fallas a cache per tant escrius a principal

Fallades B = 128

Fallades C = 128

$$\frac{2^8}{2^1} = 2^7$$

Es podria reduir el número de fallades modificant el número de blocs per conjunt? Justifica-ho i explica de quina manera, si és que es pot, juntament amb el nombre de fallades a cada vector que s'obtindrien.

No, no es pot reduir el num de fallades, totes són necessàries per carregar les dades a caché.

Repeteix el problema fent servir ara una política d'**escriptura retardada amb assignació** (write-back, write-allocate).

Fallades A = 1024

Fallades B = 1024

Fallades C = 1024

Es podria reduir el número de fallades modificant el número de blocs per conjunt? Justifica-ho i explica de quina manera, si és que es pot, juntament amb el nombre de fallades a cada vector que s'obtindrien.

Si que es pot augmentant el número de blocs a 3 mínim.

Uanors tindriem 128 fallades per cada vector

EXAMEN FINAL D'EC

17 de juny de 2021

- 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, la solució i el procediment de revisió es publicaran al Raccò el dia 25 de juny.

Pregunta 1 (1 punt)

Un processador d'última generació disposa de 4 tipus d'instruccions diferents: A, B, C i D. La següent taula mostra quin és el nombre d'instruccions executades per un programa sota consideració i el CPI de cada tipus d'instrucció. La freqüència del rellotge del processador és de 2GHz i la potència dissipada és de 251W.

Tipus d'instrucció	#instr.	CPI
A	$4 \cdot 10^9$	1
B	$3 \cdot 10^9$	3
C	$2 \cdot 10^9$	1
D	$1 \cdot 10^9$	3

Calcula el CPI mitjà del programa sota consideració

$$\text{CPI} = \frac{1 \cdot 4 \cdot 10^9 + 3 \cdot 3 \cdot 10^9 + 1 \cdot 2 \cdot 10^9 + 3 \cdot 1 \cdot 10^9}{4 \cdot 10^9 + 3 \cdot 10^9 + 2 \cdot 10^9 + 1 \cdot 10^9} = \frac{19 \cdot 10^9}{10 \cdot 10^9} = 1.9$$

Indica quin és el temps d'execució en segons del programa

$$T_{\text{exe}} = \boxed{9} \text{ s} \quad t_{\text{exe}} = \text{nºins} \cdot \text{CPI} = \text{nºins} \cdot \frac{\sum \text{CPI} \cdot \text{nºins}}{\text{nºins}} \cdot \frac{1}{f} = \frac{18 \cdot 10^9}{2 \cdot 10^9} = 9$$

Calcula l'energia consumida en Joules durant l'execució del del programa

$$E = \boxed{2259} \text{ J} \quad E = P \cdot t = 251 \cdot 9 \quad \frac{4}{251} \times \frac{9}{2259}$$

Indica quin seria el guany (speed-up) que s'obtindria si s'aconseguís reduir el CPI de les instruccions de tipus B a 1 cicle.

$$\text{guany} = \frac{\text{CPI}_{\text{ant}}}{\text{CPI}_{\text{act}}} = \frac{1.8}{1.2} = 1.5$$

$$\text{CPI}_{\text{act}} = \frac{1 \cdot 4 \cdot 10^9 + 1 \cdot 3 \cdot 10^9 + 1 \cdot 2 \cdot 10^9 + 3 \cdot 1 \cdot 10^9}{10 \cdot 10^9} = \frac{12 \cdot 10^9}{10 \cdot 10^9} = 1.2$$

Pregunta 2 (0.5 punts)

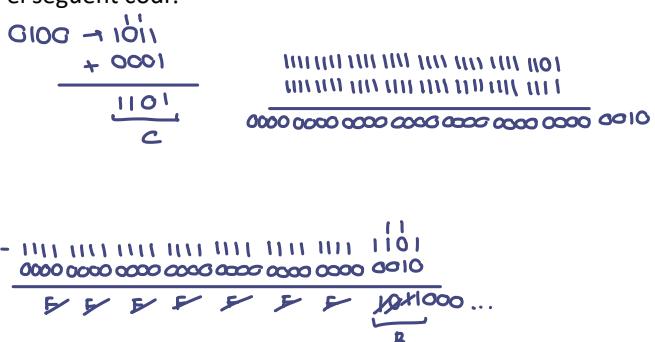
Quin serà el contingut final de \$t0 en hexadecimal després d'executar el següent codi?

```

li    $t0, -4 → $t0 = 0x FFFF FFFF
sra   $t1, $t0, 31 $t1 = 0x FFFFFFFF
xor   $t0, $t0, $t1 $t0 = 0x 0000 0002
subu  $t0, $t0, $t1 $t0 = 0x 0000 0004
sll   $t1, $t1, 31 $t1 = 0x 8000 0000
or    $t0, $t0, $t1

```

$$\$t0 = \boxed{0x 8000 0004}$$



Pregunta 3 (1 punt)

Donades les següents declaracions de variables globals en assemblador MIPS, que s'ubiquen a memòria a partir de l'adreça 0x10010000:

Omple la següent taula amb el contingut de memòria en hexadecimal. Les variables s'emmagatzem a partir de l'adreça 0x10010000. Deixa EN BLANC les posicions no ocupades per cap dada.

@Memòria	Dada	@Memòria	Dada	@Memòria	Dada	@Memòria	Dada
0x10010000	0x63	0x10010008	00	0x10010010	16	A3	0x10010018
0x10010001	0x62	0x10010009	00	0x10010011	17	A2	0x10010019
0x10010002	0x61	0x1001000A	01	0x10010012	18	A1	0x1001001A
0x10010003		0x1001000B	10	0x10010013	19		0x1001001B
0x10010004	F8	0x1001000C	05	0x10010014	20	00	0x1001001C
0x10010005	FF	0x1001000D	19	0x10010015	21	00	0x1001001D
0x10010006		0x1001000E	A5	0x10010016	22	A0	0x1001001E
0x10010007		0x1001000F	A4	0x10010017	23	3F	0x1001001F

Pregunta 4 (1 punt)

Posa una X al costat de cada una de les següents afirmacions (a la columna V si és Verdadera o a la columna F si és Falsa). Suposem en tots els casos que es fa referència a un processador MIPS com l'estudiat a classe. Cada resposta correcta suma 0,1 punts; les respostes no contestades no es tenen en compte; cada resposta incorrecta resta 0,1 punts; i la puntuació total mínima és 0.

Afirmació	V	F
La traducció d'adreces virtuals a físiques en una instrucció <code>lw</code> pot arribar a produir fins a quatre excepcions.	X	
En un computador que disposa de memòria virtual amb TLB sempre cal fer dos accessos a memòria principal per cada referència: un a la Taula de Pàgines i l'altre a l'adreça de la referència.		X
Les fallades de TLB causen una excepció i es tracten per software. No obstant, les fallades de TLB invoquen la rutina <code>TLBmiss</code> , molt més curta d'executar que la rutina d'excepcions genèrica RSE.	X	
El TLB funciona com una memòria cache de la Taula de Pàgines, les etiquetes de la qual consisteixen en el número de pàgina virtual (VPN).	X	
En un processador amb adreces de 32 bits, una memòria cache associativa de 4 vies, de 32KB i blocs de 32 bytes, s'han de dedicar 20 bits a l'etiqueta, 7 al número de conjunt i 5 al desplaçament.		X
La taula de pàgines es pot modificar quan el bit EXL, del registre Status, val 0.		X
Després de servir una interrupció, de vegades, tornem a reexecutar la mateixa instrucció durant la qual ha arribat aquesta interrupció.		X
Podem desactivar les excepcions per <i>overflow</i> posant a 0 el bit corresponent a aquestes excepcions del camp IM del registre Status.	X	
Les interrupcions es gestionen com un tipus concret d'excepció, i permeten al sistema operatiu interaccionar amb els dispositius externs de forma asíncrona.		X
La instrucció <code>tlbwr</code> implementa la política de reemplaçament LRU pel TLB.		X

Pregunta 5 (0,75 punts)

Donat el següent fragment de codi en llenguatge C

```
unsigned int x,y;

if (( x > y ) && ( y >= 10 )) {
    y++;
    x = x / 4;
} else {
    x = y;
}
```

L'hem traduït a assemblador MIPS sense fer servir macros o pseudoinstruccions. Completa els següents quadres amb els corresponents mnemònics i operands a fi que la traducció sigui correcta.

sltu	y	x	\$t3, \$t1, \$t0
beq			\$t3, \$zero, else \$t3 = 0 → else
addiu	\$t7, \$zero,		10
sltu	\$t3,		\$t1 , \$t7
bne	\$t3		, \$zero , else

then:

addiu	\$t1, \$t1, 1
srl	\$t0, \$t0, 2
beq	\$zero, \$zero, endif

else:

addu	\$t0, \$t1, \$zero
------	---------------------------

endif:

Pregunta 6 (1 punt)

Escriu un codi en assemblador de MIPS (màxim 4 instruccions) que multipliqui els valors de \$t0 i \$t1 (ambdós naturals) de forma que \$t3 tingui el resultat de la multiplicació i \$t7 valgui 0 si hi ha hagut sobreiximent, i 1 altrament.

multu	\$t0, \$t1
mflo	\$t2
mfhi	\$t3
shfu	\$t3, \$t3, 1

$$\text{ind}[i][i] = @\text{ind} + 2 \cdot i \cdot N + 2 \cdot i$$

Pregunta 7 (1 punt)

Donat el següent fragment de codi en llenguatge C

```
int    vec[N];
short ind[N][N];
int i, sum = 0;
for (i=0; i<N; i++)
    sum += vec[ ind[i][i] ] + vec[ ind[N-i-1][2] ];
```

$$\begin{array}{r}
 \text{ind}[i][i] \\
 - \text{ind}[0][0] \\
 \hline
 \text{ind}[i][i] = @\text{ind} + 2N + 2 \\
 \text{ind}[N-2][2] \\
 - \text{ind}[N-1][2] \\
 \hline
 \text{ind}[i][i] = 2 \cdot (-1)N + 20 = -2N \\
 \\
 \text{ind}[N-1][2] = @\text{ind} + 2 \cdot N - i \cdot N + 2 \cdot 2 \\
 = 2(N-i) \cdot N + 4
 \end{array}$$

L'hem traduït a assemblador MIPS. Completa els següents quadres amb valors (en funció d' N si cal) a fi que la traducció sigui correcta.

```

li      $t0, 0          # i
li      $t7, N
li      $v0, 0          # sum

la      $t1, vec
la      $t2, ind
addiu   $t3, $t2, 2(N-1) · N + 4

loop:
beq    $t0, $t7, end_loop

lh      $t5, 0($t2)
sll    $t5, $t5, 2
addu   $t5, $t5, $t1
lw      $t5, 0($t5)
addu   $v0, $v0, $t5

lh      $t5, 0($t3)
sll    $t5, $t5, 2
addu   $t5, $t5, $t1
lw      $t5, 0($t5)
addu   $v0, $v0, $t5

addiu  $t2, $t2, 2N+2
addiu  $t3, $t3, -2N
addiu  $t0, $t0, 1
b      loop

end_loop:
```

vec[i] = @vect + i · N

Cognoms:

DNI:

Pregunta 8 (1.25 punts)

Nom:

Tenim un processador MIPS amb un co-processador de coma flotant, on els registres \$f4 i \$f6 contenen els valors 0x4136DB6D i 0xBFB6DB6B, respectivament. En aquest processador, s'executa la instrucció MIPS add.s \$f0, \$f4, \$f6. La unitat de suma i resta utilitza els tres bits GRS de guarda i arrodoneix al més pròxim. Contesta les següents preguntes:

Quina és la mantissa (en binari) i l'exponent (en decimal) dels nombres que hi ha a §f4 i §f6?

Omple les següents caselles mostrant l'operació (+/-), els nombres a operar, els bits de guarda i el resultat:

Omple el resultat després de re-normalitzar i arrodonir:

Quin és el contingut de \$f0 en hexadecimal després d'executar la instrucció?

\$f0 = 0x41200000

Quin valor decimal representa \$f0 després d'executar la instrucció?

$$\$f0 = \boxed{10.0}$$

0100 0001 0010 0000 0000 0000 0000 0000
 ↓, mantissa $2^e = 1, m \cdot 2^3 = 1.25 \cdot 8 = 10.0$

Pregunta 9 (1.25 punts)

Considera un processador amb adreces de 32 bits i una memòria cache de dades amb les següents característiques:

- Capacitat: 1kB → 2^{10}
 - Mida del bloc: 64 bytes → 2^6 offset
 - Correspondència associativa per conjunts
 - Blocs per conjunt: 2
 - Política de reemplaçament LRU
 - Escriptura retardada amb assignació (write-back, write-allocate)

$\frac{2^{10}}{2^6} = 2^4 \text{ bytes} \times \text{bloc} = 16 \text{ pos. / bloc}$

$\frac{2^9}{2^5} = 2^3 = 8 \rightarrow \forall \text{ ocupa } 8 \text{ blocs}$

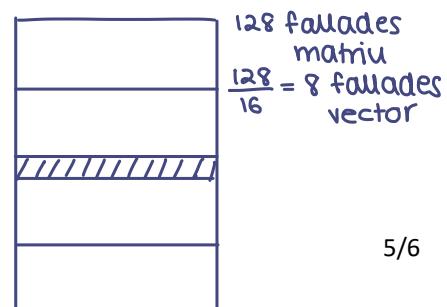
Suposem que la memòria cache és inicialment buida, i executem el següent programa al nostre processador:

```

memòria cache es inicialment buida, i executem el següent programa al nostre processador

int V[128];           → 128 → 0000 0010 0000 0000
int M[128][16];       → 4   → 0x0200
void main() {
    int i,j,tmp;      // a $t0, $t1 i $t2 respectivament

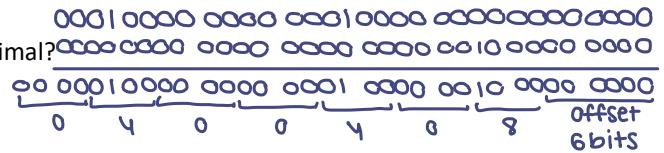
    for (i=0; i<128; i++)
        for (j=0; j<16; j++) {
            tmp = M[i][j]; → 128 · 16
            V[i] = V[i] + tmp;
        }
    }                   → 128 · 16 · 2
                        ↓
                        escriure
                        y negar!
```



Considera addicionalment que el vector V comença a partir de l'adreça 0x10010000 i que tant el vector V com la matriu M estan convenientment inicialitzats, tot i que no s'ha de tenir en compte aquesta inicialització en la resolució de l'exercici.

Quin serà el número del bloc de MP on es troba $M[0][0]$, en hexadecimal?

Bloc MP $M[0][0] = \boxed{0x0400408}$



Quantes referències a memòria per dades es realitzaran durant l'execució del programa, i quantes fallades de cache es produiran?

$$\text{Referències} = \boxed{128 \cdot 16 \times 2 + 128 \cdot 16}$$

$$\text{Fallades} = \boxed{128 + 8 = 136}$$

Decidim experimentar canviant aquesta cache per una altra amb les mateixes característiques (1kB, blocs 64 bytes) però amb una política de correspondència directa.

Quina serà ara la quantitat de fallades que s'obtindrà en l'execució d'aquest codi?

$$\text{Fallades} = \boxed{384}$$

128 per carregar la matrícula

$$16 \cdot 8 \cdot 2 = 256 \text{ fallades vector}$$

Pregunta 10 (1,25 punts)

Considera un processador MIPS amb un sistema de memòria virtual paginada. Les pàgines són de 4kB de mida i la política de reemplaçament és LRU. El sistema operatiu determina que la memòria física només mantindrà 4 pàgines per programa.

Per simplificar el problema només es consideren els accessos a dades, descartant la gestió dels accessos a instruccions.

L'estat de la taula de pàgines en un moment donat de l'execució d'un programa és el següent (només mostrem les entrades amb P=1):

	VPN	PPN	P	D	
0x1007E	0x10010	0x0	1	✓	0
0x10090	0x10020	0x3	1	✓	0
0x10020	0x1007F	0x2	1	0	
0x10010	0x10080	0x1	1	✓	1

El sistema està complementat per un TLB completament associatiu de dues entrades, amb reemplaçament LRU. L'ordre de les últimes referències a memòria és el següent: 0x10020 (accés més llunyà), 0x1007F, 0x10080, 0x10010 (accés més recent).

Considera la seqüència d'accisos a memòria de la taula següent. Per a cada accés, on s'indica l'adreça de l'accés en hexadecimal i si és Lectura (L) o Escriptura (E), omple el seu número de pàgina lògica (VPN), si provoca una fallada de TLB, si provoca una fallada de pàgina, si s'ha d'escriure a disc, quina pàgina lògica es reemplaça i quin és el PPN resultat de la traducció d'adreses. A les columnes amb preguntes booleanes escriu "SI" o "NO". A les columnes numèriques escriu la dada en hexadecimal.

L/E	Adreça (hex)	VPN (hex)	Fallada TLB?	Fallada Pàgina?	Escript. Disc?	VPN Reempl. (hex)	PPN (hex)
E	10080874	10080	NO	NO	NO	-	0x1
L	1009077C	10090	Sí	Sí	Sí	0x10020	0x3
E	10020AA0	10020	Sí	Sí	NO	0x1007F	0x2
L	1007EFFC	1007E	Sí	Sí	Sí	0x10080	0x1
L	10010998	10010	Sí	Sí	Sí	0x10010	0x0

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'enunci.
- 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
Aritmètic-lògiques	$70 \cdot 10^9$	1

Quin és el temps d'execució en segons del programa amb cache ideal?

70 s

$$CPI = \frac{(1 \cdot 50 + 2 \cdot 20 + 32 \cdot 1 + 2 \cdot 9 + 1 \cdot 70) \cdot 10^9}{(50 + 20 + 1 + 9 + 70) \cdot 10^9} = \frac{210}{150}$$

$$t_{exe} = (150 \cdot CPI) / 3 = 70$$

Sabem que el computador real està equipat amb una cache d'escriptura immediata sense assignació, i que els seus temps representatius són $t_h = 1$ cicle (temps d'encert) i $t_{block} = 14$ 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 %

$$\begin{aligned} \text{taxa fallades} &= 1 - \text{taxa d'encerts} \\ \frac{7}{140} &= 0.5 \rightarrow 14 \cdot 0.5 = 7 \end{aligned}$$

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?

200s

$$E = 6000 \text{ J}$$

$$E = P \cdot t$$

$$t = \frac{6000}{30} = 200 \text{ s}$$

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

$1.0 \cdot 2^{-126}$

0x0080 0000

El menor número denormal (no-normalitzat) positiu i no nul

$0.1 \cdot 2^{-127}$

0x0000 0001

El major número normalitzat positiu (diferent de +Inf)

0x7F7FFFFF

Un NaN amb signe positiu

0x7F8X XXXX

on XXXXX != 0000

El -Inf

0x7F80 0000

12,825

$$+1100,1101\overline{0011} \rightarrow +1,1001101\overline{0011} \cdot 2^3$$

$$0.825 \times 2 = 1.65 \quad 1$$

$$0.65 \times 2 = 1.3 \quad 1$$

$$0.3 \times 2 = 0.6 \quad 0$$

$$0.6 \times 2 = 1.2 \quad 1$$

$$0.2 \times 2 = 0.4 \quad 0$$

$$0.4 \times 2 = 0.8 \quad 0$$

$$0.8 \times 2 = 1.6 \quad 1$$

$$0.6 \times 2 = 1.2 \quad 1$$

$$0.2 \times 2 = 0.4 \quad 0$$

$$0.4 \times 2 = 0.8 \quad 0$$

$$0.8 \times 2 = 1.6 \quad 1$$

$$0.6 \times 2 = 1.2 \quad 1$$

01000001010011010011001100110011
4 1 4 D 3 3 3 3

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];
}
```

$\mathtt{mat[63][0]} = \mathtt{@mat} + 63 \cdot 64 + 0 = 4032$
 $\mathtt{mat[62][k-1]}$
 $\mathtt{mat[63][k]}$
 $\mathtt{mat[-1][-1]} = 64 \cdot (-1) - 1 = -65$

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
    → aux
    move $t3, $zero          # $t3 : aux=0
loop: lb $t4, 0($t0)
      addu $t3, $t3, $t4
      addiu $t0, $t0, -65
      b geu $t0, $t1, loop → mira cuando llegas al último elem
      andi $t5, $t3, 0x000F → aux %16
      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	04	0x10010008	01	0x10010010 ¹⁶	DD	0x10010018 ²⁴	00
0x10010001	00	0x10010009	54	0x10010011 ¹⁷		0x10010019 ²⁵	04
0x10010002	01	0x1001000A ¹⁸	F0	0x10010012 ¹⁹		0x1001001A ²⁶	40
0x10010003	10	0x1001000B ¹⁹	E2	0x10010013 ¹⁹		0x1001001B ²⁷	
0x10010004	DE	0x1001000C ²⁰	00	0x10010014 ²⁰	FF	0x1001001C ²⁸	
0x10010005	00	0x1001000D ²¹	00	0x10010015 ²¹	06	0x1001001D ²⁹	
0x10010006		0x1001000E ¹⁹	FF	0x10010016 ²¹	84	0x1001001E ²⁹	
0x10010007		0x1001000F ¹⁵	FF	0x10010017 ²³	FF	0x1001001F ²¹	

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.

		111011100
la	\$t0, a	\$t0 = 0x10010000
lw	\$t1, 0(\$t0)	\$t1 = 0x10010004
lb	\$t2, 0(\$t1)	\$t2 = 0xFFFF FFDE
sra	\$t2, \$t2, 2	\$t2 = 0xFFFF FFFF7
sh	\$t2, 4(\$t1)	sh (0x10010008) = 0xFFFF7
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	0x10010010
lb	\$t1, 0(\$t0)	\$t1 = 0xFFFFFD
li	\$t0, 1	\$t0 = 0x0000 0001
addiu	\$t0, \$t0, 1	\$t0 = 0x0000 0002
addiu	\$t0, \$t0, 1	\$t0 = 0x0000 0003
addu	\$t1, \$t1, \$t0	
		FFFFFD + 00000003 ----- FFFFFEE0
\$t1 =	0xFFFFFEE0	

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);          alloc   free
    ...
}

}

```

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];	<i>.data</i>
... = pL;	<i>registre</i>
... = *(pD+20);	<i>heap</i>
... = VL[6];	<i>pila</i>
... = pD;	<i>registre</i>
... = sum;	<i>.data</i>
... = pG;	<i>.data</i>

$$\frac{2^{14}}{2^{12}} = 2^2 = 4$$

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.

$$2^4 \cdot 2^{10} = 2^{14} \rightarrow 16 / 4 = 4 \text{ marcs}$$

$$2^2 \cdot 2^{10} = 2^{12}$$

Quins bits de l'adreça lògica serviran per conèixer el número de pàgina lògica (*virtual page number* 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, 24

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).