COGNOMS: **GRUP:**

NOM:

EXAMEN PARCIAL D'EC
28 d'abril de 2020

L'examen consta de 6 preguntes, que s'han de contestar als mateixos fulls de l'enunciat. No oblidis posar el teu nom i cognoms a tots els fulls. La duració de l'examen és de **120 minuts**. Les notes, la solució i el procediment de revisió es publicaran al Racó properament.

Pregunta 1. (1,5 punts)

Sigui el següent programa en C:

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

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

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

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

vecs: @ inici = vecs + 16 Stride = 40

```
    la    $t4, mats + 196
    la    $t5, vecs + 16
    li    $t0, 0
    li    $t1, 10
  for:
    bge   $t0, $t1, fi_for
    lw    $t3, 0($t4)
    sw    $t3, 0($t5)
    addiu $t4, $t4, -4
    addiu $t5, $t5, 20
    addiu $t0, $t0, 1
    b     for
  fi_for:
```

Pregunta 2. (2,25 punts)

Sigui el següent fragment de programa en C:

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

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

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

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

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

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

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

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

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

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

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

COGNOMS:	GRUP:
NOM:	

Pregunta 3. (1,75 punts)

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

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

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

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)
0x10010000	0x10010004	0x00DE00DD	0xFF8406FE	0x00400400
0x10010010	0xE2E05401	0xFFFFFF00		

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

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

$t1 = 0xFF84FFF6

Pregunta 4. (1,5 punts)

Donat el següent fragment de codi:

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

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

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

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

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

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

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

$f8 = 0x41600404

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

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

COGNOMS:	**GRUP:**
NOM:	

Pregunta 5. (0,75 punts)

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

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

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

Millora (speed-up)= 1,5

Pregunta 6. (2,25 punts)

Donada la següent declaració de funcions en C:

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

int func(int W[], int n, int m)
{
   int V[10];
   int x = m;

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

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

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

Cal salvar $ra (per la crida). Cal salvar n i x (en $s1 i $s2) ja que després de la crida es necessiten els seus valors anteriors a la crida.

```
func:
  # Reserva pila i salva regs modificats
  addiu  $sp, $sp, -56
  sw     $s1, 44($sp)
  sw     $s2, 48($sp)
  sw     $ra, 52($sp)
  move   $s1, $a1       # copia n en registre $s

  # primera sentència
  move   $s2, $a2       # x = m

  # segona sentència
  move   $a1, $a0       # passem W
  move   $a0, $s1       # passem n
  jal    g
  addu   $s2, $s2, $v0  # x = x + g(...)

  # tercera sentència
  sll    $t0, $s1, 2    # 4*n
  addu   $t0, $sp, $t0  # @V + 4*n
  lw     $t0, 0($t0)    # V[n]
  addu   $v0, $t0, $s2  # return V[n] + x

  # Restaura regs i allibera pila
  lw     $s1, 44($sp)
  lw     $s2, 48($sp)
  lw     $ra, 52($sp)
  addiu  $sp, $sp, 56
  jr     $ra
```

Bloc d'activació