EXAMEN PARCIAL D'EC
29 de març de 2020

Pregunta 1 *(0,5 punts)*
Donada la següent subrutina en assemblador MIPS:

```
acces_aleatori:
        li      $t0, 400
        mult    $t0, $a1
        mflo    $t0
        addu    $t0, $t0, $a0
        lw      $v0, 480($t0)
        jr      $ra
```

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

```
int acces_aleatori (int M[][100], int i)
{
        return M[ i+1 ][ 20 ];
}
```

Pregunta 2 *(1 punt)*
Donada la següent funció `foo` en alt nivell correcte (no cal comprovar si se surt de rang), on `M` i `V` són declarades com a variables globals:

```
int M[100][100]; /* suposarem que està inicialitzada */
int V[100];
void foo() {
        int i,aux; /* ocupen els registres $t1 i $t3 respectivament */
        for (i=0; i<97; i++) {
                aux = M[i][i+3];
                M[i+1][i+3] = V[aux];
        }
}
```

Completa el següent codi MIPS omplint les caselles en blanc perquè sigui equivalent a l'anterior codi en alt nivell, tenint en compte que els elements de la matriu s'accedeixen utilitzant la tècnica **d'accés seqüencial**, usant el registre `$t0` com a punter per accedir els elements `M[i][i+3]`.

```
        la      $t0, M + 12
        li      $t1, 0
        li      $t2, 97
        la      $t7, V
bucle:  bge     $t1, $t2, fibuc
        lw      $t3, 0($t0)
        sll     $t3, $t3, 2
        addu    $t3, $t3, $t7
        lw      $t4, 0($t3)
        sw      $t4, 400($t0)
        addiu   $t0, $t0, 404
        addiu   $t1, $t1, 1
        b bucle
fibuc:
```

Pregunta 3 *(2 punts)*
Ens han demanat que dissenyem un processador compatible amb l'ISA del MIPS per a una aplicació específica. Per a obtenir un millor rendiment, aquesta aplicació necessita un format per a coma flotant en simple precisió una mica lleuger a l'estàndard IEEE-754. El nou format serà similar excepte que té 6 bits per l'exponent, codificat en excés 31, i 25 bits per la mantissa.

Donat el següent fragment de codi en assemblador MIPS per aquest nou processador:

```
li     $t1, 0x48040002
mtc1   $t1, $f4
li     $t2, 0xC0400009
mtc1   $t2, $f6
add.s  $f8, $f4, $f6
```

Volem calcular, pas a pas, el resultat de la darrera instrucció en el codi anterior (`add.s`).

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

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

Omple les següents caselles mostrant l'operació op (+/−), les cadenes de bits a operar, i el resultat:

		G	R	S
op	1, 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0	0	0	0
−	0, 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	1	0	1
	0 0, 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1	0	1	1

Resultat després de re-normalitzar (si cal):

	G	R	S	exponent (decimal)
1, 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0	1	1	0	4

Resultat després d'arrodonir:

	exponent (decimal)
1, 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1	4

Quin és el valor de $f8 en hexadecimal després d'executar la instrucció `add.s`?

`$f8 =` 0x**47C00003**

Pregunta 4 *(0.75 punts)*
Donades les següents declaracions de variables locals

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

i el següent codi en alt nivell

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

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

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

Pregunta 5 *(2 punts)*
Donada la següent declaració de variables globals d'un programa escrit en llenguatge C:

```
short a[] = {513, 17, -5};
long long b = 1030;
short c = 0;
char d[] = "COVID19"; /* s'usen tants bytes com calgui per l'string*/
short *e = &c;
```

a) Tradueix-la al llenguatge assemblador del MIPS

```
.data
        a: .half 513, 17, -5
        b: .dword 1030
        c: .half 0
        d: .asciiz "COVID19"
        e: .word c
```

b) Completa la següent taula amb el contingut de memòria en hexadecimal. Tingues en compte que el codi ASCII de la '0' és el 0x30 i 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	**02**
0x10010002	**11**
0x10010003	**00**
0x10010004	**FB**
0x10010005	**FF**
0x10010006	
0x10010007	

@Memòria	Dada
0x10010008	**06**
0x10010009	**04**
0x1001000A	**00**
0x1001000B	**00**
0x1001000C	**00**
0x1001000D	**00**
0x1001000E	**00**
0x1001000F	**00**

@Memòria	Dada
0x10010010	**00**
0x10010011	**00**
0x10010012	**43**
0x10010013	**4F**
0x10010014	**56**
0x10010015	**49**
0x10010016	**44**
0x10010017	**31**

@Memòria	Dada
0x10010018	**39**
0x10010019	
0x1001001A	
0x1001001B	
0x1001001C	**10**
0x1001001D	**00**
0x1001001E	**01**
0x1001001F	**10**

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

```
la $t0, d
lbu $t0, 5($t0)
addiu $t0, $t0, -0x32
sltu $t1, $zero, $t0
```

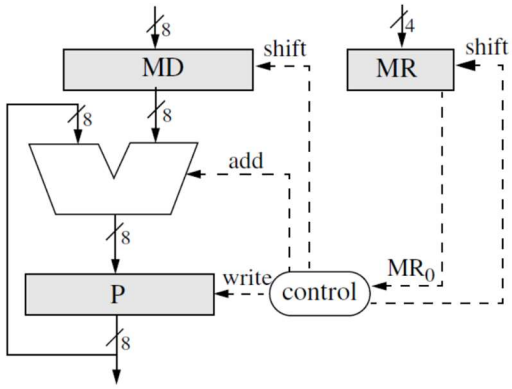
$t1 = 0x**00000001**

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

```
*e = a[0] - a[2]
```

```
la   $t0, a
lh   $t1, 0($t0)    # a[0]
lh   $t2, 4($t0)    # a[2]
subu $t1, $t1, $t2
la   $t2, e
lw   $t2, 0($t2)    # contingut punter
sh   $t1, 0($t2)
```

Pregunta 6 *(1,5 punts)*
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) Suposem que amb aquest circuit multipliquem els números binaris de 4 bits 1011 (multiplicand) i 1010 (multiplicador). Completa la següent taula, que mostra els valors en binari dels registres `P`, `MD`, i `MR` després de la inicialització i després de cada iteració, afegint tantes iteracions com facin falta:

Iter.	P (producte)								MD (Multiplicand)								MR (Multiplicador)			
ini	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1	1	0	1	0
1	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**1**	**1**	**0**	**0**	**1**	**0**	**1**
2	**0**	**0**	**0**	**1**	**0**	**1**	**1**	**0**	**0**	**0**	**1**	**0**	**1**	**1**	**0**	**0**	**0**	**0**	**1**	**0**
3	**0**	**0**	**0**	**1**	**0**	**1**	**1**	**0**	**0**	**1**	**0**	**1**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**1**
4	**0**	**1**	**1**	**0**	**1**	**1**	**1**	**0**	**1**	**0**	**1**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**

b) Suposem que els registres MIPS `$t1` i `$t2` valen `$t1=0x00000008` i `$t2=0xFFFFFFFF`. Indica el contingut final en hexadecimal de `$hi` i `$lo` després d'executar `multu $t1,$t2` i després de `mult $t1,$t2`.

`multu:` `$hi=` **0x00000007** `$lo=` **0xFFFFFFF8**

`mult:` `$hi=` **0xFFFFFFFF** `$lo=` **0xFFFFFFF8**

Pregunta 7 *(1 punt)*
Donades les següents declaracions de funcions en C:

```
float func1(int *w, int m);/* prototip de la funció func1 */
float func2(int n) {
       float s, t;
       int v[100];
       for (s=0.0; s >= 0.0; n++) {
              t = func1(v, n);
              s = t + func1(v, n);
       }
       return s;
}
```

a) D'acord amb les regles de l'ABI estudiades, ¿quins elements de la funció `func2` (paràmetres, variables locals i càlculs intermedis) s'han de guardar necessàriament en registres de tipus segur `$s` o `$f`? Especifica el registre segur concret que tries en aquells casos que en cal un.(Omple tantes entrades de la taula següent com et calguin):

Element de `func2` (en C)	Registre
t	**$f20**
n	**$s0**

b) De quina mida en bytes seria el bloc d'activació de un cop traduïda a MIPS la funció `func2`?

Mida = **412** bytes

Pregunta 8 *(1,25 punts)*

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=60W i que la freqüència de rellotge és de 1,2GHz, calcula el temps d'execució en segons i l'energia consumida en Joules durant l'execució del programa de test.

t_{exe} = **25** s

E = **1500** 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 electrònic, 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_{exe} = **20** s

speedup = **1,25**

P = **90** W