

## Tema 6: LLENGUATGE D'ASSEMBLADOR

### Grau en Informàtica

## SOLUCIONS ALS EXERCICIS

6.1-Organització de la memòria	p.1
6.2-Joc d'instruccions	p.2
6.3-Programació en assemblador i codi màquina	p.3
6.4-Exercicis genèrics	p.15

### **Organització de la memòria**

1. Distribuiu les següents dades, de 4 bytes de grandària cada una, a les adreces de memòria corresponents.

Dada 1: 0xABCDEFFF, Dada 2: 0x01234567

<i>Little endian</i>				<i>Big endian</i>			
3	2	1	0	3	2	1	0
0xAB	0xCD	0xEF	0xFF	0xFF	0xEF	0xCD	0xAB
7	6	5	4	7	6	5	4
0x01	0x23	0x45	0x67	0x67	0x45	0x23	0x01

2. Donades les següents directives de dades, indiqueu quin serà el contingut de la memòria de dades, sabent que NULL representa el caràcter nul i que la màquina en qüestió emmagatzema les paraules segons el format Little Endian. Indiqueu clarament les zones de memòria de contingut desconegut.

```
.data 0x10000028
.byte 3
.ascii "ABC"
.float 1.5
.word 0xFFFF
.half 19,38
```

31	24	23	16	15	8	7	0	Adreça
'C'		'B'		'A'		0x03		0x10000028
0x3F		0xC0		0x00		0x00		0x1000002C
0x00		0x00		0xFF		0xFF		0x10000030
0x00		0x26		0x00		0x13		0x10000034
?		?		?		?		0x10000038
?		?		?		?		0x1000003C

3. Donades les següents directives de dades, indiqueu quin serà el contingut de la memòria de dades, sabent que NULL representa el caràcter nul i que la màquina en qüestió emmagatzema les paraules segons el format Little Endian. Indiqueu clarament les zones de memòria de contingut desconegut.

```
.data 0x1000000C
.space 2
.half 0xF0
.asciiz "050"
.double 1.5
```

31	24	23	16	15	8	7	0	Adreça
0x00	0xF0	0x00	0x00	0x00	0x00	0x00	0x00	0x1000000C
NULL	'0'	'5'	'0'	'0'	'0'	'0'	'0'	0x10000010
?	?	?	?	?	?	?	?	0x10000014
0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x10000018
0x3F	0xF8	0x00	0x00	0x00	0x00	0x00	0x00	0x1000001C
?	?	?	?	?	?	?	?	0x10000020

4. Donades les següents directives de dades, indiqueu quin serà el contingut de la memòria de dades, sabent que NULL representa el caràcter nul i que la màquina en qüestió emmagatzema les paraules segons el format Little Endian. Indiqueu clarament les zones de memòria de contingut desconegut.

```
.data 0x10000000
.half 5,3
.byte 3
.data 0x10000011
.byte 5
```

31	24	23	16	15	8	7	0	Adreça
0x00	0x03	0x00	0x05	0x00	0x05	0x05	0x05	0x10000000
?	?	?	?	?	?	?	?	0x10000004
?	?	?	?	?	?	?	?	0x10000008
?	?	?	?	?	?	?	?	0x1000000C
?	?	?	?	?	?	?	?	0x10000010
?	?	?	?	?	?	?	?	0x10000014

### Joc d'instruccions

5. Donat el contingut següent de la memòria de dades:

Memòria de dades	Adreça
0x 6C FF FF FF	0x10000000
0x AB 77 80 44	0x10000004
31	0

Quin valor tindran els registres \$5 i \$6 després d'executar-ne les instruccions següents?

```
lui $2, 0x1000
lh $5, 0($2)
lw $6, 4($2)
```

\$5=0xFFFFFFFF  
\$6=0xAB778044

6. Donat el contingut següent de la memòria de dades:

Memòria de dades	Adreça
0x 12 34 56 78	0x10010008
0x CB 00 88 00	0x1001000C
31	0

Quin valor tindran els registres \$4 i \$5 després d'executar-ne les següents instruccions?

```
lui $3, 0x1001
lw $4, 12($3)
lb $5, 9($3)
```

\$4=0xCB008800  
\$5=0x00000056

## Programació en assemblador i codi màquina

7. Donat el codi en llenguatge d'assemblador MIPS R2000 que es mostra a continuació.

```
.data 0x100000A0
.byte 1,2,3
.half 4
dada1:.word 8
dada2:.word 2
.space 5
.word 9
.text 0x00400000
.globl __start

__start:
    la $2, dada1
    la $3, dada2
    lw $8, 0($2)
    lw $4, -4($3)
    add $9, $4, $8
    sw $9, 0($2)
    .end
```

A. Quin és el contingut de la memòria de dades abans de l'execució del programa?

31	24	23	16	15	8	7	0	Adreça
?		0x03		0x02		0x01		0x100000A0
?		?		0x00		0x04		0x100000A4
0x00		0x00		0x00		0x08		0x100000A8
0x00		0x00		0x00		0x02		0x100000AC
0x00		0x00		0x00		0x00		0x100000B0
?		?		?		0x00		0x100000B4

B. Quin és el contingut de la memòria de dades després de l'execució del programa?

31	24	23	16	15	8	7	0	Adreça
?		0x03		0x02		0x01		0x100000A0
?		?		0x00		0x04		0x100000A4
0x00		0x00		0x00		0x10		0x100000A8
0x00		0x00		0x00		0x02		0x100000AC
0x00		0x00		0x00		0x00		0x100000B0
?		?		?		0x00		0x100000B4

C. Quin serà el valor emmagatzemat en els registres següents després de l'execució del programa?

Registre	Valor
\$2	0x100000A8
\$3	0x100000AC
\$8	0x00000008
\$4	0x00000008
\$9	0x00000010

D. Indiqueu la seqüència d'instruccions per les quals es traduiria la pseudoinstrucció `la $2, dada1`

```
lui $1, 4096 / lui $1, 0x1000
ori $2, $1, 168 / ori $2, $1, 0x00A8
```

E. Codifiqueu la instrucció `lw $4, -4 ($3)`

0x8C64FFFC

8. En un algorisme d'encriptació per blocs, es van xifrant blocs d'una grandària concreta. Per a augmentar la seguretat de l'algoritme, se solen fer operacions prèvies entre blocs. Un dels modes d'operar es coneix com CBC i consisteix a realitzar una OR exclusiva entre el bloc de text a xifrar i el bloc precedent. El codi següent implementa el funcionament descrit.

```

.data 0x10000000
gran:      .half 8          # Grandària del bloc
BlocA:     .asciiz "or bloqu" # Bloc precedent
BlocB:     .asciiz "es, se v"  # Bloc a xifrar
BlocF:     .space 8          # Espai per al resultat

.globl __start
.text 0x00400000
__start:
    # Lectura de les dades inicials
    la $10, BlocA      # Llegim l'adreça del bloc precedent
    la $11, BlocB      # Llegim l'adreça del bloc a xifrar
    la $12, BlocF      # Llegim l'adreça del bloc resultat
    la $13, gran        # Llegim la grandària del bloc
    lh $14, 0($13)

    # bucle de l'algorisme
bucle:
    beq $14, $0, fi
    lb $20, 0($10)
    lb $21, 0($11)
    xor $22, $20, $21
    sb $22, 0($12)
    addi $10, $10, 1
    addi $11, $11, 1
    addi $12, $12, 1
    addi $14, $14, -1
    j bucle
fi:    # Final de l'algorisme
.end

```

A. Indiqueu quin contingut tindrà la memòria de dades abans d'executar el programa.

31	24	23	16	15	8	7	0	Adreça
0x72	0x6F	0x00	0x08	0x10000000				
0x6F	0x6C	0x62	0x20	0x10000004				
0x65	NULL	0x75	0x71	0x10000008				
0x73	0x20	0x2C	0x73	0x1000000C				
NULL	0x76	0x20	0x65	0x10000010				
0x00	0x00	0x00	0x00	0x10000014				
0x00	0x00	0x00	0x00	0x10000018				

B. Indiqueu quin contingut tindrà la memòria de dades després d'executar el programa.

31	24	23	16	15	8	7	0	Adreça
0x72	0x6F	0x00	0x08	0x10000000				
0x6F	0x6C	0x62	0x20	0x10000004				
0x65	NULL	0x75	0x71	0x10000008				
0x73	0x20	0x2C	0x73	0x1000000C				
NULL	0x76	0x20	0x65	0x10000010				
0x42	0x0C	0x01	0x0A	0x10000014				
0x03	0x51	0x0A	0x1F	0x10000018				

C. Quin serà el valor emmagatzemat en els següents registres després de l'execució del programa?

Registre	Valor
\$10	0x1000000A
\$11	0x10000013
\$12	0x1000001C
\$13	0x10000000
\$14	0x00000000

D. Codifiqueu la instrucció `j bucle`

0x08100008

E. Codifiqueu la instrucció `beq $14, $0, fi`

0x11C0000A

9. Donat el programa següent en llenguatge assemblador del MIPS R2000:

```
.data 0x10000000
estat:    .byte 25
zonaA:    .word 0xffffffff,0xffffffff, 0xffffffff, 0xffffffff
grandaria: .word 4

.text 0x400400
.globl __start
__start:
    la $6, estat
    lb $6, 0($6)
    la $7, zonaA
    la $8, grandaria
    lw $9, 0($8)
    li $10, 0x00000000
    li $11, 0xaaaaaaaa
    beq $6,$0,accio0
accio1:
    beq $9,$0,fi
    sw $11, 0($7)
    addi $7,$7,4
    addi $9,$9,-1
    j accio1
accio0:
    beq $9,$0,fi
    sw $10, 0($7)
    addi $7,$7,4
    addi $9,$9,-1
    j accio0
fi:
.end
```

A. Quin és el contingut de la memòria de dades abans de l'execució del programa?

31	24	23	16	15	8	7	0	Adreça
?	?	?	?	?	0x19	?	?	0x10000000
0xFF	0xFF	0xFF	0xFF	0xFF	0xFF	0xFF	0xFF	0x10000004
0xFF	0xFF	0xFF	0xFF	0xFF	0xFF	0xFF	0xFF	0x10000008
0xFF	0xFF	0xFF	0xFF	0xFF	0xFF	0xFF	0xFF	0x1000000C
0xFF	0xFF	0xFF	0xFF	0xFF	0xFF	0xFF	0xFF	0x10000010
0x00	0x00	0x00	0x00	0x00	0x04	?	?	0x10000014

B. Quin és el contingut de la memòria de dades després de l'execució del programa?

31	24	23	16	15	8	7	0	Adreça
?	?	?	?	?	0x19	?	?	0x10000000
0xAA	0xAA	0xAA	0xAA	0xAA	0xAA	0xAA	0xAA	0x10000004
0xAA	0xAA	0xAA	0xAA	0xAA	0xAA	0xAA	0xAA	0x10000008
0xAA	0xAA	0xAA	0xAA	0xAA	0xAA	0xAA	0xAA	0x1000000C
0xAA	0xAA	0xAA	0xAA	0xAA	0xAA	0xAA	0xAA	0x10000010
0x00	0x00	0x00	0x00	0x00	0x04	?	?	0x10000014

C. Codifiqueu la instrucció `j accio0`

0x08100110
------------

D. Codifiqueu la instrucció `beq $6, $0, accio0`

0x10C00006
------------

**10.** Com a part d'un algoritme de realitat virtual, es vol calcular el volum d'un prisma regular i comprovar si aquest volum supera un cert llindar. En el cas que el volum supere el llindar, el programa escriu un 1 en el byte de memòria etiquetat com "res"; en qualsevol altre cas escriu un 0. Així mateix, el volum calculat s'emmagatzema en l'adreça de memòria etiquetada com "vol". El codi proposat és el següent:

```

.data 0x10000000
llargA:    .half 100    # Aresta A
llargB:    .half 50     # Aresta B
llargC:    .half 25     # Aresta C
llindar:   .word 1500   # Llindar per a la comparació
res:       .byte 0      # Espai per al resultat de la comparació
vol:       .word 0      # Espai per al volum del prisma

.globl __start
.text 0x00400000
__start:
    # Lectura dels valors de les arestes
    la $20, llargA
    lh $10, 0($20)
    la $20, llargB
    lh $11, 0($20)
    la $20, llargC
    lh $12, 0($20)

    # Càlcul de volum
    mult $10, $11
    mflo $13
    mult $12, $13
    mflo $13

    # Emmagatzemem el volum en la memòria
    la $20, vol
    sw $13, 0($20)

    # Comparació amb el llindar
    la $20, llindar
    lw $14, 0($20)
    slt $15, $14, $13
    # Emmagatzemem el resultat
    la $20, res
    sb $15, 0($20)
.end

```

A. Indiqueu quin és l'estat del segment de dades abans que s'execute el programa.

31	24	23	16	15	8	7	0	Adreça
0x00		0x32			0x00		0x64	0x10000000
?		?			0x00		0x19	0x10000004
0x00		0x00			0x05		0xDC	0x10000008
?		?			?		0x00	0x1000000C
0x00		0x00			0x00		0x00	0x10000010
?		?			?		?	0x10000014



- B. Feu una proposta de nova declaració de dades que reduísca els espais de memòria no usats a causa de l'alineament

Amb l'actual disposició es perden 5 bytes a causa de l'alineament. Per a solucionar-ho es poden moure les etiquetes de manera que les dades que provoquen un alineament s'agrupen, d'esta manera la declaració quedarà com:

```
llindar:      .word 1500
vol:         .word 0
llargA:      .half 100
llargB:      .half 50
llargC:      .half 25
res:         .byte 0
```

Existixen més solucions que estalvien un espai, encara que la més eficient és la que s'ha exposat.

- C. Quin serà el valor emmagatzemat en els següents registres després de l'execució del programa? En tot cas, indiqueu en quina base estan expressats els valors numèrics

Registre	Valor
\$10	0x00000064
\$11	0x00000032
\$12	0x00000019
\$13	0x0001E848
\$14	0x000005DC
\$15	0x00000001
\$20	0x1000000C

- D. Codifiqueu la instrucció `slt $15, $14, $13`

0x01CD782A

11. El codi següent conté error: identifiqueu-los.

```
.data 0x10000000
vector: .byte 0x333, 0x88, 0x54, 0x77
        .word 0x55
        .half 0x44445555
        .space 18
        .text 0x400000
        .globl __start
__start:
        lui $10, 0x1000
        ori $10, $10, 0x0003
        lw $11, 0($10)
        lw $12, 1($10)
.end
```

*Error 1: la dada 0x333 no és un BYTE, hauria de declarar-se amb una altra directiva com .HALF o .WORD*

*Error 2: la dada 0x44445555 no és mitja paraula (HALF), hauria de declarar-se amb la directiva .WORD*

*Error 3: l'adreça d'accés a memòria utilitzada per la instrucció `lw $11, 0($10)` que accedeix a una paraula en memòria no és una adreça múltiple de 4, per tant l'execució d'esta instrucció finalitzaria el programa amb error. L'adreça que ocasiona l'error d'execució és: 0x10000003 que no és una adreça vàlida per a accedir a una paraula en memòria.*

## 12. Quin és el resultat d'executar el codi següent?

```
.text 0x400800
.globl __start
__start:
bucle: lui $10, 0xFFFF
        andi $10, $10, 0xFFFF
        beq $10, $zero, bucle
        li $10, 0x12345678
.end
```

```
.globl __start
__start:
bucle: lui $10, 0xFFFF           # $10=0xFFFF0000
        andi $10, $10, 0xFFFF    # $10=0xFFFFFFFF
        beq $10, $zero, bucle     # $10<>0, per tant, no salta
        li $10, 0x12345678        # $10=0x12345678
.end
```

## 13. Escriviu un programa en llenguatge d'assemblador del MIPS R2000 que implemente les operacions següents:

$$\text{resultat} = \text{dadaa} - \text{dadab} + \text{dadac} - \text{dadad}$$

Heu de tenir en compte el següent:

- Les dades “dadaa”, “dadab”, “dadac” i “dadad” es defineixen com enters de 32 bits ubicats a partir de l'adreça de memòria 0x10002000.
- Heu de reservar espai per a emmagatzemar el resultat com enter de 32 bit a partir de l'adreça de memòria 0x10001000.
- El programa emmagatzemarà el resultat de les operacions en “resultat”.

Una possible solució:

```
.data 0x10002000
dadaa: .word 9      # Es poden iniciar amb altres valors
dadab: .word 2
dadac: .word 3
dadad: .word 1

.data 0x10001000
resultat: .word 0

.globl __start
.text 0x00400000
```

```

__start:
    # Càrrega de dades des de memòria a registres
    la $8, dadaa
    lw $8, 0($8)      # $8 = dadaa
    la $9, dadab
    lw $9, 0($9)      # $9 = dadab
    la $10, dadac
    lw $10, 0($10)    # $10 = dadac
    la $11, dadad
    lw $11, 0($11)    # $11 = dadad

    # Càlculs aritmètics, acumulant en el registre $2
    sub $2, $8, $9     # $2 = dadaa-dadab
    add $2, $2, $10     # $2 = (dadaa-dadab)+dadac
    sub $2, $2, $11     # $2 = (dadaa-dadab+dadac)-dadad

    # Escriptura del resultat final des de registre a memòria
    la $7, resultat
    sw $2, 0($7)       # resultat = (dadaa-dadab+dadac)-dadad

.end

```

- 14.** Codifiqueu en llenguatge assemblador del MIPS R2000 un programa que realitzi la suma de dues variables de tipus *short int* (16 bits) anomenades “dadaa” i “dadab”, i emmagatzeme el resultat de la suma en “dadac”. Heu de fer la reserva de dades necessària per a accedir a les tres variables, i escriure les instruccions que adients per a realitzar les operacions de suma i emmagatzematge. El codi d'alt nivell següent mostra el que es demana:

```

short int dadaa = 9;
short int dadab =12;
short int dadac;
dadac = dadaa+dadab;

```

Una possible solució:

```

.data 0x10000000
dadaa: .half 9    # Es podrien iniciar amb altres valors
dadab: .half 12
dadac: .half 0

.globl __start
.text 0x00400000
__start: # Càrrega des de memòria a registres
    la $2, dadaa
    lh $2, 0($2)      # $2 = dadaa
    la $3, dadab
    lh $3, 0($3)      # $3 = dadab

    #Operació de suma
    add $2, $2, $3     # $2 = dadaa + dadab

    # Escriptura del resultat, des de registre a memòria
    la $3, dadac
    sh $2, 0($3)      # c = (dadaa + dadab)

.end

```

15. Escriviu un programa en llenguatge d'assemblador del MIPS R2000 que calcule l'operació  $\text{dadac} = \text{dada} * \text{dadab}$ . Heu de tenir en compte les especificacions següents:

- “dadaa” i “dadab” han de definir-se com enters de 16 bits a partir de l'adreça de memòria 0x10000000.
- Heu de reservar espai per a emmagatzemar el resultat en “dadac” com enter de 32 bit a partir de l'adreça de memòria 0x10001000.

Una possible solució:

```
.data 0x10000000
dadaa: .half 9      # Podeu triar altres valors
dadab: .half 12
      .data 0x10001000
dadac: .word 0

      .globl __start
      .text 0x00400000
__start:
    # Càrrega de dades des de memòria a registres
    la $2, dadaa
    lh $2, 0($2)      # $2 = dadaa
    la $3, dadab
    lh $3, 0($3)      # $3 = dadab
    # Operació de multiplicació
    mult $2, $3       # HI=0 i LO = dadaa * dadab
                        # per que dadaa i dadab són de 16 bits
    mflo $2           # $2 = LO = dadaa * dadab
    # Escriptura del resultat, des de registres a memòria
    la $3, dadac
    sw $2, 0($3)      # dadac = (dadaa*dadab)
.end
```

16. Escriviu un programa en llenguatge d'assemblador del MIPS R2000 que realitzi l'operació de divisió entera  $\text{dada} / \text{dadab}$ . Heu de tenir en compte les especificacions següents:

- En primer lloc cal comprovar que dadab és distint de zero per continuar amb la divisió. En cas contrari, el programa ha de saltar a l'etiqueta “divisioperzero”.
- El quocient de la divisió ha de guardar-se en la variable “quocient”.
- El residu de la divisió ha de guardar-se en la variable “residu”.
- Heu de declarar totes les variables “dadaa”, “dadab”, “quocient” i “residu” com enters de 32 bit ubicats a partir de l'adreça de memòria 0x10000000.

Una possible solució:

```
.data 0x10000000
dadaa: .word 12      # Es poden triar altres valors
dadab: .word 6
quocient: .word 0
residu:   .word 0
```

```

        .globl __start
        .text 0x00400000
__start:
        # Càrrega de dades des de memòria a registres
        la $2, dadaa
        lw $2, 0($2)          # $2 = dadaa
        la $3, dadab
        lw $3, 0($3)          # $3 = dadab

        # Divisió
        beq $3,$0, divisioperzero
        div $2, $3             # HI=resto i LO = $2/$3

        # Escriptura del quocient a memòria
        mflo $2                # $2 = quocient
        la $3, quocient
        sw $2,0($3)

        # Escriptura del residu a memòria
        mfhi $2                # $2 = residu
        la $3, residu
        sw $2,0($3)
        j fi
divisioperzero: # Mostrar missatge d'error?
fi:
        .end

```

**17.** Codifiqueu en llenguatge d'assemblador del MIPS R2000 un programa que realitzi l'operació `resultat = dada+dadab`, comprovant si es produeix o no desbordament (overflow) i indicant-ho en la variable “`hihadesdordament`”. Heu de tenir en compte les especificacions següents:

- Heu de declarar totes les variables “`dadaa`”, “`dadab`”, “`resultat`” i “`desbordament`” con enters de 32 bit a partir de l'adreça de memòria `0x10000000`.
- Després de fer l'operació, s'emmagatzemarà un 1 en “`hihadesbordament`” si el resultat està fora de rang. En cas contrari s'emmagatzemarà un 0. en “`hihadesbordament`”.
- Per detectar el desbordament heu de comprovar si el bit de signe dels operands és el mateix però diferent del signe del resultat, el que indica que hi ha desbordament.

Una possible solució:

```

        .data 0x10000000
dadaa: .word 9      # Podeu gastar altres valors
dadab: .word 2
resultat: .word 0
hihadesbordament: .byte 0

```

```

        .globl __start
        .text 0x00400000
__start:
        # Càrrega de dades des de memòria a registres
        la $8, dadaa
        lw $8, 0($8)          # $8 = dadaa
        la $9, dadab
        lw $9, 0($9)          # $9 = dadab

        # Càlculs aritmètics acumulant en $2
        add $2, $8, $9        # $2 = dadaa+dadab

        # Escriptura del resultat final des de registre a memòria
        la $7, resultat
        sw $2, 0($7)          # resultat =dadaa-dadab

        # Comprovar si hi ha desbordament. En la suma
        # d'enters en complement a dos, no més pot haver
        # desbordament si els dos operands tenen el mateix
        # signe.
        li $3, 0x80000000     # Màscara de bit de signe
        and $10, $8, $3       # Filtre bit de signe en $10 de
                                # de dadaa
        and $11, $9, $3       # Filtre bit de signe en $11 de
                                # de dadab
        bne $10, $11, fi      # Si són de diferent signe, la suma
                                # no pot produir desbordament
        # Si tenen el mateix signe, el signe del resultat ha
        # de ser també.
        # Si no són iguals és que hi ha desbordament
        and $12, $2, $3       # Filtre bit de signe en $12 de resultat
        beq $12, $10, fi      # Signes iguals, no hi ha desbordament
        # Si execute per ací és que hi ha desbordament
        la $3, hihadesbordament
        li $2, 1
        sw $2, 0($3)
        j eixir

fi: # No hi ha desbordament
    la $3, hihadesbordament
    sw $0, 0($3)

eixir:.end

```

**18.** Codifiqueu en llenguatge d'assemblador del MIPS R2000 un programa que realitzi l'operació  $\text{resultat} = \text{dada} - \text{dadab}$ , comprovant si es produeix o no desbordament (overflow) i indicant-ho en la variable “desbordament”. IMPORTANT, el desbordament no es detecta de la mateixa manera que en la suma. Heu de tenir en compte les especificacions següents:

- Heu de declarar totes les variables “dadaa”, “dadab”, “resultat” i “desbordament” com enters de 32 bit a partir de l'adreça de memòria 0x10000000.

- Després de fer l'operació, s'emmagatzemarà un 1 en “desbordament” si el resultat està fora de rang. En cas contrari s'emmagatzemarà un 0. en “desbordament”.
- Per detectar el desbordament heu de comprovar si el bit de signe dels operands és igual. En aquest cas no pot haver-hi desbordament. En cas contrari, hi ha desbordament si el bit de signe del minuend és diferent del bit de signe del resultat.

Una possible solució:

```
.data 0x10000000
dadaa: .word 9      # Podeu gastar altres valors
dadab: .word 2
resultat: .word 0
hihadesbordament: .byte 0

.globl __start
.text 0x00400000
__start:
    # Càrrega de dades des de memòria a registres
    la $8, dadaa
    lw $8, 0($8)      # $8 = dadaa
    la $9, dadab
    lw $9, 0($9)      # $9 = dadab

    # Càlculs aritmètics acumulant en $2
    sub $2, $8, $9    # $2 = dadaa-dadab

    # Escriptura del resultat final des de registre a memòria
    la $7, resultat
    sw $2, 0($7)      # resultat =dadaa-dadab

    # Comprovar si hi ha desbordament. En la suma
    # d'enters en complement a dos, no més pot haver
    # desbordament si els dos operands tenen el mateix
    # signe. Però en el cas de la resta és l'inrevés,
    # si tenen diferent signe podria haver-hi
    # desbordament
    li $3, 0x80000000  # Màscara de bit de signe
    and $10, $8, $3    # Filtre bit de signe en $10 de
                        # de dadaa
    and $11, $9, $3    # Filtre bit de signe en $11 de
                        # de dadab
    beq $10, $11, fi   # Si són del mateix signe, la resta
                        # no pot produir desbordament

    # Si tenen els signes diferents, el signe del resultat ha
    # de ser el mateix que el signe de dadaa (el minuend).
    # Si no són iguals és que hi ha desbordament
    and $12, $2, $3    # Filtre bit de signe en $12 de resultat
    beq $12, $10, fi   # Signes iguals, no hi ha desbordament
    # Si executa per ací és que hi ha desbordament
    la $3, hihadesbordament
    li $2, 1
    sw $2, 0($3)
    j eixir

fi: # No hi ha desbordament
```

```

        la $3, hihadesbordament
        sw $0, 0($3)

eixir:.end

```

19. Considerant les directives de dades que es mostren a continuació, escriviu un programa en llenguatge assemblador del MIPS R2000 que guardi en “tira\_res” la mateixa cadena de caràcters emmagatzemada en la variable “tira”, però convertint cadascuna de les lletres a minúscules. En la taula ASCII es pot observar que la diferència entre el codi d’una lletra majúscula i el codi de la mateixa lletra minúscula és el valor del bit 5, que per les majúscules és 0 i per les minúscules és 1. Per tant, podem convertir majúscules a minúscules i a l’inrevés sumant-li i restant-li 32, o ficant a 1 i 0 el bit 5.

```

.data 0x10000000
tira: .asciiz "ABC"
tira_res: .space 3

```

Una possible solució:

```

        .data 0x10000000

tira:    .asciiz "ABC"
tira_res: .space 3

        .globl __start
        .text 0x00400000

__start: la $10, tira      # Emmagatzemem en $10 l'adreça de tira
        la $11, tira_res  # Emmagatzemem en $11 l'adreça tira_res
bucle:   lb $12, 0($10)    # Carrega en $12 el codi ASCII del
                        # del caràcter
        beq $12, $0, fi    # si és el caràcter 0 termina el bucle
        addi $12, $12, 32  # suma 32 per obtenir minúscula
        sb $12, 0($11)     # escriu el caràcter en tira_res
        addi $10, $10, 1   # apuntem al següent caràcter de tira
        addi $11, $11, 1   # apuntem al següent byte de tira_res
        j bucle

fi:
        .end

```

## Exercicis genèrics

20. Tenint en compte el format d’instrucció vist en les transparències, quantes instruccions de tipus R, I i J pot tindre el MIPS?

*Els formats I i J, no tenen codi de funció, per la qual cosa el total d'instruccions que es poden tindre són 63 instruccions de tipus I i J (ja que el codi 0 ho emprenen les instruccions de tipus R), i 64 de tipus R ja que el camp funció és de 6 bits. És a dir 127 instruccions.*



**21.** Responen a les qüestions següents sobre les instruccions de tipus R

- A. Si totes les instruccions de tipus R tenen el codi d'operació 0 quantes instruccions de tipus R com a màxim pot tindre el MIPS?

*El codi de funció és de 6 bits, per la qual cosa el nombre de funcions possibles és de  $2^6$  és a dir 64 instruccions.*

- B. En les instruccions *sll* i *srl*, quin és el desplaçament màxim que es pot posar? Seria interessant tindre més desplaçament?

*El màxim desplaçament és de 32 bits, és a dir  $2^5$ , ja que 5 és la grandària de camp de desplaçament. Com els registres són de 32 bits, no cal desplaçar més.*

- C. Si el banc de registres del MIPS tinguera 64 registres de 32 bits, quins canvis implicaria en el format de les instruccions de tipus R?

*El problema seria que els camps que defineixen el nombre de registre haurien de tindre 6 bits, com és necessari definir 3 registres, serien necessaris 18 bits per als registres, per la qual cosa caldria canviar la grandària d'alguns dels camps restants, o fer major la grandària de la instrucció.*

- D. Si la grandària dels registres del banc de registres del MIPS passara a ser de 64 bits, mantenint-se 32 registres, quins canvis implicaria en el format de les instruccions de tipus R?

*Només afectaria les instruccions de desplaçament, que haurien de contemplar la possibilitat de desplaçar 64 bits interns dins d'un registre.*

**22.** Responen a les següents qüestions sobre les instruccions de tipus I

- A. Si el banc de registres del MIPS tinguera 64 registres de 32 bits, quins canvis implicaria en el format de les instruccions de tipus I?

*Serien necessaris 6 bits per a emmagatzemar el nombre de registre què es referència, la qual cosa deixaria menys espai per a la dada immediata.*

- B. Quina és la distància màxima a què pot arribar un salt condicional?

*La distància vindrà definida pel nombre de paraules que es poden emmagatzemar en el camp immediat. Com és de 16 bits, i està en complement a dos, els salts poden anar des de 32767 instruccions més avant fins a 32768 instruccions més arrere.*

- C. Per què es codifica en complement a dos la dada immediata en les instruccions de salt condicional?

*Complement a dos simplifica el càlcul de l'adreça destí, i permet nombres enters positius i negatius, per la qual cosa el salt pot ser cap avant o cap arrere*

- D. Per què es codifica el salt en paraules en les instruccions de salt condicional? A quina distància màxima s'arribaria si es codificara en bytes i no en paraules?

*El que es calcula és l'adreça de la instrucció a què s'ha de saltar, i les instruccions estan sempre emmagatzemades en adreces múltiple de quatre. Si es codificara en bytes, simplement es reduiria l'abast del salt.*

- E. Quin és el desplaçament màxim teòric on es pot arribar amb una instrucció de càrrega o emmagatzemament?

*Teòricament no es pot desplaçar més de 32767 o -32768, encara que hi ha mètodes que permeten desplaçaments majors (provar amb el PCSPIM).*

- F. Per què les instruccions de càrrega i emmagatzemament tenen el desplaçament codificat en bytes?

*Perquè el desplaçament s'usa per a calcular una adreça de memòria de dades, i el tipus de dades byte, o ASCII permet estar alineat en qualsevol adreça, és a dir no es pot suposar que siga adreça múltiple de dos o quatre.*

### 23. Responen a les següents qüestions sobre les instruccions de tipus J

- A. El fet de no poder saltar a una adreça que no comence per 0x0.... fa que el MIPS es deixi fora part de la zona de memòria destinada al codi? Per què?

*No passa res, ja que les adreces de memòria d'instruccions, que són les adreces a què es pot saltar, van des de la 0x00400000 a la 0xFFFFFFF, en el dit rang, els quatre primers bits, sempre són 0.*

- B. Per què la instrucció *jr* no es considera de tipus J?

*Per que no precisa emmagatzemar l'adreça de memòria dins de la instrucció, ja que l'adreça de memòria a què s'ha de saltar està en un registre.*

- C. La grandària d'adreça del MIPS és de 32 bits. En què afectaria les instruccions de tipus J un augment de grandària d'adreça del MIPS a 64 bits?

*Codificar l'adreça seria molt complicat, ja que només es tenen 26 bits per a emmagatzemar l'adreça de salt. S'hauria de buscar una altra manera de codificar les instruccions de salt.*