

Enunciats de la sessió

Activitat 2.A: Operacions lògiques i desplaçaments

Les operacions lògiques i els desplaçaments ens permeten fer algunes operacions aritmètiques de forma més eficient donat que podem evitar, en alguns casos, l'ús d'operacions de divisió i multiplicació per potències de 2, o bucles iteratius.

A part de les intruccions d'operacions bit a bit explicades a la lectura prèvia, MARS disposa de les instruccions `sllv`, `srlv`, `srav` per fer desplaçaments d'un nombre variable de bits, especificant aquest número en un registre.

```
sllv $s1,$s2,$s3    # $s1<-$s2 despl. lògic esq. tants bits com indica $s3
srlv $s1,$s2,$s3    # $s1<-$s2 despl. lògic dreta tants bits com indica $s3
srav $s1,$s2,$s3    # $s1<-$s2 despl. aritm. dreta tants bits com indica $s3
```

Nota: aquestes 3 instruccions sols usen els 5 bits de menor pes del tercer operand (`$s3`).

La resta de bits s'ignoren (poden no ser zero).

Exercici 2.1: Escriviu un programa en MIPS que, donats dos enters X i Y que es troben als registres `$s0` i `$s1` respectivament, inverteixi (complementi) els X bits de menys pes de Y . Per fer-ho, podeu usar operacions bit a bit, però no podeu fer servir instruccions de salt ni de comparació, i us ha de sortir un programa amb menys de 5 instruccions.

Nota: podeu fer servir la següent sentència en C (on l'operador `^` expressa la *xor* bit a bit, i l'operació `1<<X` significa "el valor 0x01 desplaçat X bits a l'esquerra"):

$$Y = Y \wedge ((1 \ll X) - 1);$$

```
li $s2, 1
sllv $s3, $s2, $s0
subu $s4, $s3, $s2
xor $s5, $s4, $s1
```

Copieu el codi de l'exercici anterior al fitxer `s2a.s`. Comproveu que el codi és correcte mirant els valors inicial i final de `$s1`, per a diferents valors de X i Y .

Activitat 2.B: Sentències if-then-else

El codi següent comprova si el codi ASCII que conté la variable *num* correspon a un símbol alfabètic, a un dígit decimal, o a cap dels dos (pot ser de control, un símbol de puntuació, etc). Si és un símbol alfabètic, escriurem a la variable *result* el valor de *num*. Si és un dígit decimal, hi escriurem el seu valor en format d'enter. Altrament, hi escriurem un -1:

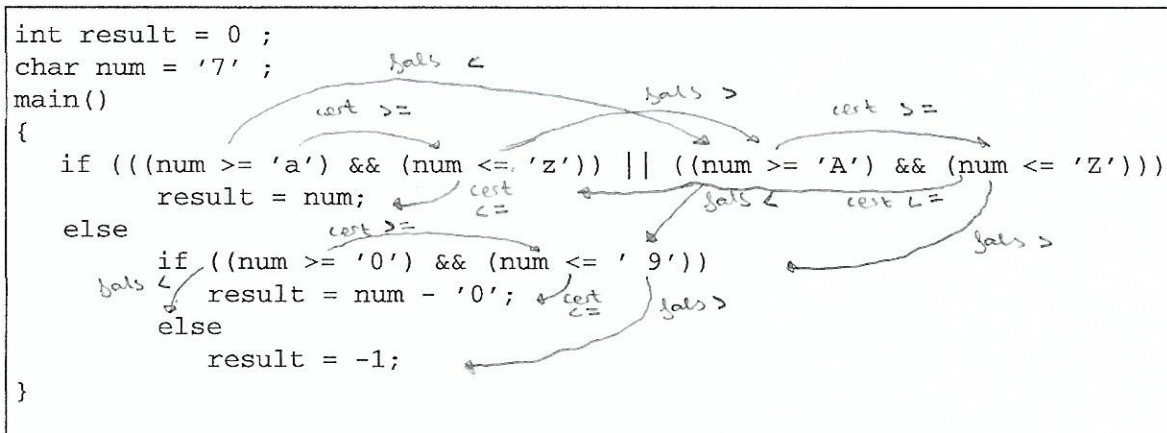


Figura 2.1: Programa que classifica un caràcter

Completa l'exercici 2.2 abans de continuar.

Exercici 2.2: Tradueix a assembleador MIPS el programa de la Figura 2.1

<pre> .data result: .word 0 num: .byte '7' .text .globl main main: li \$t0, 0x61 la \$s0, num lb \$s1, 0(\$s0) blt \$s1, \$t0, or1 li \$t1, 0x7a ble \$s1, \$t1, result or1: li \$t2, 0x30 blt \$s1, \$t2, else1 </pre>	<pre> li \$t3, 0x5A bgt \$s1, \$t3, else1 result: Move \$v1, \$s1 else1: li \$t4, 0x30 blt \$s1, \$t4, else2 li \$t5, 0x39 bgt \$s1, \$t5, else2 sub \$v1, \$s1, \$t4 b ends1 else2: li \$v1, -1 ends1: jr \$ra </pre>
---	---

Copia el programa de l'exercici anterior al fitxer `s2b.s` i comprova que al final de la seva execució el valor de *result* és 7. Fés la prova per a diferents valors de *num*: 'a', 'z', 'Z', '0' i ' ', per exemple. Els seus codis ASCII són 0x61, 0x7a, 0x5A, 0x30 i 0x3B, respectivament.

Activitat 2.C: Calcular el caràcter més freqüent d'un string

El següent programa (veure Figura 2.2) declara el vector global *w* del tipus string, format per 32 caràcters. Els 31 primers representen dígit numèrics (del '0' al '9', codificats amb valors del 48 al 57), i l'últim és el sentinella (caràcter `null` = '\0', codificat amb el valor 0).

La funció *moda* es crida una vegada des del *main* per tal que calculi quin és el caràcter més freqüent de la cadena *w*. Aquesta funció construeix, al vector local *histo* de 10 enters, un histograma que emmagatzema, per cada possible caràcter numèric, la seva freqüència d'aparició. A més a més, a cada pas, el caràcter més freqüent es guarda a la variable local *max*. La funció consta de dos bucles, un per inicialitzar les freqüències a zero, i l'altre per recórrer la cadena de caràcters, fent una crida a la funció *update* per cada caràcter visitat.

La funció *update* actualitza la freqüència del caràcter visitat en l'histograma, la compara amb la del caràcter *max*, i retorna el nou caràcter més freqüent. Dins d'aquesta funció hi ha una crida a l'acció *nofares*, que no fa res d'útil, i que està posada per facilitar la verificació de

```
char w[32] = "8754830094826456674949263746929";
char resultat;                                /* dígit ascii més freqüent */

main()
{
    resultat = moda(w, 31);
    print_character(resultat); /* consultar lectura prèvia sessió 1 */
}
char moda(char *vec, int num)
{
    int k, histo[10];
    char max;

    for (k=0; k<10; k++)
        histo[k] = 0;

    max = '0';
    for (k=0; k<num; k++)
        max = '0' + update(histo, vec[k]-'0', max-'0');

    return max;
}
void nofares();
char update(int *h, char i, char imax)
{
    nofares();
    h[i]++;
    if (h[i] > h[imax])
        return i;
    else
        return imax;
}
```

Figura 2.2: Càlcul de la moda.

la correctesa del codi que genereu.

Completa l'exercici 2.3 abans de continuar.

Exercici 2.3: Tradueix a ensamblador MIPS el codi de les funcions *moda* i *update* de la Figura 2.2. No oblidis posar dins d'*update* la crida a la subrutina *nofares*.

```

.data
w: .asciiz "87548300948264566
      34949263746929"
resultat: .byte 0

.text
.globl main

main:
    addiu $sp, $sp, -4
    sw $ra, 0($sp)
    la $a0, w
    li $a1, 31
    jal moda
    la $s0, resultat
    sb $v0, 0($s0)
    move $a0, $v0
    li $v0, 11
    syscall
    lw $ra, 0($sp)
    addiu $sp, $sp, 4
    jr $ra

nofares:
    li $t0, 0x12345678
    move $t1, $t0
    move $t2, $t0
    move $t3, $t0
    move $t4, $t0
    move $t5, $t0
    move $t6, $t0
    move $t7, $t0
    move $t8, $t0
    move $t9, $t0
    move $a0, $t0
    move $a1, $t0
    move $a2, $t0
    move $a3, $t0
    jr $ra

moda:
    addiu $sp, $sp, -60
    sw $s0, 40($sp)
    sw $s1, 44($sp)
    sw $s2, 48($sp)
    sw $s3, 52($sp)
    sw $ra, 56($sp)

    li $t0, 0
    li $t1, 10
    move $t2, $sp
for:  beq $t0, $t1, $i
      sw $zero, 0($t2)
      addiu $t2, $t2, 4
      addiu $t0, $t0, 1
      b for

$i:   move $s0, $a0
      move $s1, $a1
      li $s2, 0
      li $s3, '0'

for2: beq $s2, $s1, $i2
      move $a0, $sp
      addu $a1, $s0, $s2
      lb $a2, 0($a1)
      addiu $a1, $a1, -48
      addiu $a2, $s3, -48
      jal update
      addiu $s3, $v0, '0'
      addiu $s2, $s2, 1
      b for2

$i2:  move $v0, $s3
      lw $s0, 40($sp)
      lw $s1, 44($sp)
      lw $s2, 48($sp)
      lw $s3, 52($sp)
      lw $ra, 56($sp)
      addiu $sp, $sp, 60
      jr $ra

* Continua

```

Copia el codi anterior al fitxer `s2c.s`. Comprova amb el simulador que al final de l'execució, el valor de la variable *resultat* és '4'.

(*) Sieve

Update:

```
addiu $sp, $sp, -16
sw     $s0, 0($sp)
sw     $s1, 4($sp)
sw     $s2, 8($sp)
sw     $ra, 12($sp)
```

```
move   $s0, $a0
move   $s1, $a1
move   $s2, $a2
```

```
jal     nojars
```

```
sll     $t0, $s1, 2
addu    $t0, $s0, $t0
lw      $t1, 0($t0)
addiu   $t1, $t1, 1
sw      $t1, 0($t0)
```

```
sll     $t2, $s2, 2
addu    $t2, $s0, $t2
lw      $t2, 0($t2)
```

```
ble     $t1, $t2, else
move    $v0, $s1
b       $i3
```

```
else:   move    $v0, $s2
```

```
$i3:   lw      $s0, 0($sp)
        lw      $s1, 4($sp)
        lw      $s2, 8($sp)
        lw      $ra, 12($sp)
        addiu   $sp, $sp, 16
        jr      $ra
```


Activitat 2.D: (Opcional) Depuració de codi erroni en ensamblador.

En aquest apartat estudiarem el programa de la Figura 2.3. El vector *alfabet* és un string que conté la llista ordenada de les lletres majúscules i, com és costum en els strings, acaba amb un byte sentinella que val 0. La funció *codifica* fa el següent: donat un string d'entrada, genera un string de sortida on cada lletra ha estat intercanviada, de la següent manera: una 'A' es convertirà en una 'Z' o viceversa; una 'B' en una 'Y' o viceversa, etc. El programa principal fa dues crides a *codifica*. La primera vegada li passa un string d'entrada *w1* = "ARQUITECTURA", i retorna un string de sortida *w2*. En la segona crida li passa com a entrada *w2*, i retorna un string de sortida *w3*:

```
char alfabet[27] = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
char w1[16] = "ARQUITECTURA";
char w2[16];
char w3[16];
int count=0;

main()
{
    count = codifica(w1, w2);
    count += codifica(w2, w3);
    print_string(w2);      /* consultar lectura prèvia sessió 1 */
    print_string(w3);      /* consultar lectura prèvia sessió 1 */
}

char g(char alfa[], char *pfrase)
{
    return alfa[25 - (*pfrase - 'A')];
}

int codifica(char *pfrasein, char *pfraseout)
{
    int i;
    i = 0;
    while (*pfrasein != 0)
    {
        *pfraseout = g(alfabet, pfrasein);
        pfrasein++;
        pfraseout++;
        i++;
    }

    *pfraseout = 0;
    return i;
}
```

Figura 2.3: Programa que codifica i decodifica un string

Nosaltres hem fet una traducció a MIPS d'aquest programa (Figura 2.4), però conté tres errors. Els errors es troben localitzats en el codi de la subrutina *codifica*.

```

.data
alfabet: .ascii "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
w1:      .ascii "ARQUITECTURA"
w2:      .space 17
w3:      .space 17
count:   .word 0

.text
.globl main

main:
    addiu $sp, $sp, -4
    sw    $ra, 0($sp)
    la    $s0, count
    la    $a0, w1
    la    $a1, w2
    jal   codifica
    sw    $v0, 0($s0)
    la    $a0, w2
    la    $a1, w3
    jal   codifica
    lw    $s1, 0($s0)
    addu   $s1, $s1, $v0
    sw    $s1, 0($s0)
    li    $v0, 4
    la    $a0, w2
    syscall                                # print_string(w2)
    la    $a0, w3
    syscall                                # print_string(w3)
    lw    $ra, 0($sp)
    addiu $sp, $sp, 4
    jr    $ra
codifica:
    addiu $sp, $sp, -16
    sw    $ra, 0($sp)
    sw    $s0, 4($sp)
    sw    $s1, 8($sp)
    sw    $s2, 12($sp)
    move  $s2, $zero
    move  $s0, $a0
    move  $s1, $a1
while:
    lb    $t0, 0($s0)
lb    $t0, 0($t0)
    beq   $t0, $zero, fi_while
    la    $a0, alfabet
    move  $a1, $s1
    jal   g
    sb    $v0, 0($s0)
    addiu $s0, $s0, 1
    addiu $s1, $s1, 1
    addiu $s2, $s2, 1
    b     while
fi_while:
    sb    $zero, 0($s1)
    move  $v0, $s2
    lw    $ra, 0($sp)
    lw    $s0, 4($sp)
    lw    $s1, 8($sp)
    lw    $s2, 12($sp)
    addiu $sp, $sp, 16
    jr    $ra

```

Figura 2.4: Traducció del programa, amb errors (continua a la pàg. següent)

```

g:
    lb      $t0, 0($a1)
    li      $t1, 'A'
    addiu   $t1, $t1, 25
    subu    $t1, $t1, $t0
    addu    $t0, $a0, $t1
    lb      $v0, 0($t0)
    jr      $ra

```

Figura 2.4: Traducció del programa, amb errors (continuació)

Completa l'exercici 2.4 abans de continuar.

Exercici 2.4: Explica breument quins són els **3 errors** del programa, i com s'haurien de corregir:

-El primer error es troba al principi del while, on es repeteix una instrucció i per tant és innecessària.

```

while:
    lb      $t0, 0($s0)
lb      $t0, 0($s0)

```

-Després tant el segon com el tercer error són deguts a una confusió amb els registres on estaven guardats els paràmetres de la funció. Es troba en el mateix while a continuació del primer error.

```

beq      $t0, $zero, fi_while
la       $a0, alphabet
move     $a1, $s0 $s0
jal      0
sb       $v0, 0($s0) $s1

```

Per comprovar-ho, carrega el fitxer s2d.s, que conté el programa de la Figura 2.4, i corregeix els errors. Verifica que després d'executar-se, el programa imprimeix els strings "ZIJ-FRGVXGFIZ" i "ARQUITECTURA", i que la variable global *count* guardada a la memòria val 24.

Si no és així, depura el programa: executa'l pas a pas verificant a cada pas si fa el que s'espera que faci. La depuració requereix paciència i concentració!