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^((1<<X)-1);
```

```
        .data

        .text
        .globl main
main:
        li  $s1, 23 # Y
        li  $s0, 8  # X

        li  $t0, 1
        sllv $t0, $t0, $s0
        addiu $t0, $t0, -1
        xor $s1, $s1, $t0


        jr $ra
```

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:

```
int result = 0 ;
char num = '7' ;
main()
{
   if (((num >= 'a') && (num <= 'z')) || ((num >= 'A') && (num <= 'Z')))
         result = num;
   else
         if ((num >= '0') && (num <= '9'))
            result = num - '0';
         else
            result = -1;
}
```

Figura 2.1: Programa que classifica un caràcter

Completa l'exercici 2.2 abans de continuar.

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

```
        .data
result: .word 0
num:    .byte '7'

        .text
        .globl main
main:
        la $t0, num
        lbu $t0, 0($t0)
        li $t1, 'a'
        blt $t0, $t1, cmp
        li $t1, 'z'
        ble $t0, $t1, then

cmp:
        li $t1, 'A'
        blt $t0, $t1, else
        li $t1, 'z'
        bgt $t0, $t1, else

then:
        la $t2, result
        sw $t0, 0($t2)
        b fi
```

```
else:
        li $t1, '0'
        blt $t0, $t1, else_2
        li $t1, '9'
        bgt $t0, $t1, else_2
        addiu $t0, $t0, -48
        la $t2, result
        sw $t0, 0($t2)
        b fi

else_2:
        la $t2, result
        li $t0, -1
        sw $t0, 0($t2)

fi:
        jr $ra
```

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 dígits 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] = "87548300948264566749492637469 29";
char resultat;                    /* dígit ascii més frequent */

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 assemblador 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*

```
moda:
        # Bloc de activació
        addiu $sp, $sp, -60     # Reservem espai de 60 bytes
        sw $s0, 40($sp)         # punter vec
        sw $s1, 44($sp)         # num
        sw $s2, 48($sp)         # k
        sw $s3, 52($sp)         # max
        sw $ra, 56($sp)         # $ra

        move $s0, $a0           # $s0 = vec
        move $s1, $a1           # $s1 = num
        li $s3, '0'             # max = '0'

        # Primer for
        li $s2, 0               # incialitzem k = 0
        li $t0, 10

for:    bge $s2, $t0, max       # k < 10
        sll $t1, $s2, 2         # k = k * tipus (4bytes)
        addu $t1, $sp, $t1      # $t1 = $sp + k
        sw  $zero, 0($t1)       # histo[k] = 0
        addiu $s2, $s2, 1       # k++
        b for

        # Segon for
max:    li $s2, 0               # incialitzem k = 0


for2:   bge $s2, $s1, end       # k < num (està a $s1)

        # Pas de parametres
        move $a0, $sp           # Passem la direcció de histo ($sp)

        addu $t0, $s0, $s2      # $t0 = vec + k
        lb $a1, 0($t0)          # $a1 = vec[k]
        addiu $a1, $a1, -48     # Passem vec[k] - '0'

        addiu $a2, $s3, -48     # Passem max - '0'

        jal update              # Cridem a update

        addiu $s3, $v0, 48      # $v0 += '0'
        sb $v0, 52($sp)         # max = $v0
        addiu $s2, $s2, 1       # k++
        b for2
end:    move $v0, $s3           # Tornem max

        # Epileg
        lw $s0, 40($sp)         # punter vec
        lw $s1, 44($sp)         # num
        lw $s2, 48($sp)         # k
        lw $s3, 52($sp)         # max
        lw $ra, 56($sp)         # $ra
        addiu $sp, $sp, 60      # Alliberem espai de 60 bytes
```

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'`.

Traducció funció ***update***

```
update:
        # Bloc de activació
        addiu   $sp, $sp, -16
        sw $a0, 0($sp)          # $s0 = *h
        sw $a1, 4($sp)          # $s1 = i
        sw $a2, 8($sp)          # $s2 = imax
        sw $ra, 12($sp)

        jal nofares             # Crida a nofares

        # Restore
        lw $a0, 0($sp)
        lw $a1, 4($sp)
        lw $a2, 8($sp)
        lw $ra, 12($sp)

        sll $t0, $a1, 2         # i * tipus (4bytes)
        addu $t0, $a0, $t0      # h[i]

        lw $t1, 0($t0)          #
        addiu $t1, $t1, 1       # h[i]++
        sw $t1, 0($t0)          #

        sll $t2, $a2, 2         # imax * tipus (4bytes)
        addu $t2, $a0, $t2      # @h[imax]

        lw $t2, 0($t2)          # h[imax]

        ble $t1, $t2, else
        move $v0, $a1
        b fi_u

else:   move $v0, $a2

fi_u:   addiu   $sp, $sp, 16
        jr $ra
```