Exercici 3.1: Donada la següent declaració de variables globals en C, tradueix-la a assemblador MIPS. Recorda que en C els elements d'una matriu es guarden per files. Utilitza la directiva `.align` per garantir l'alineació dels elements, allà on calgui.

```
int mat1[5][6];
char mat2[3][5];
long long mat3[2][2];
int mat4[2][3] = {{2, 3, 1}, {2, 4, 3}};
```

```
.space  5·6·4
.space  3·5
.align  3
.space  2·2·8
.align  2
.word   2,3,1,2,4,3
```

mat 1 = 0x0000 0000
mat 2 0x0000 0078
mat 3 - 0x0000 0087
- 0x0000 0098
mat 4 - 0x0000 009c
- 0x0000 009c

Exercici 3.2: Calcula l'adreça de memòria de cada un dels següents accessos a les matrius declarades en l'apartat anterior. Pots deixar la resposta en funció de les adreces *mat1*, *mat2*, *mat3* i *mat4*.

`@mat1[4][3]` =	@mat1 + 4·6·4 + 3·4
`@mat2[2][4]` =	@mat2 + (2·5 + 4)
`@mat3[1][0]` =	@mat3 + 8(2·1 + 0)
`@mat4[0][2]` =	@mat4 + 4(0·3+2)

Exercici 3.5: Donada la següent declaració de la matriu *mat*, de 4 files per 6 columnes, de nombres enters:

```
int mat[4][6];
```

Escriu la fórmula per calcular l'adreça de l'element *mat*[*i*][*2*], en funció de l'adreça de *mat* i del valor enter *i*:

`@mat[i][2] =`	$@mat + (i \cdot 6 + 2) \cdot 4 = i \cdot 2^3 \cdot 3 + 8$

Fent servir aquesta fórmula, calcula la distància en bytes (stride) entre les adreces de dos elements consecutius d'una mateixa columna:

`@mat[i+1][2] - @mat[i][2] =`	~~$i \cdot 2^3 \cdot 3 + 8$~~ $(i+1) \cdot 2^3 \cdot 3 + 8 - i \cdot 2^3 \cdot 3 - 8$ $2^3 \cdot 3 (i+1-i) = 2^3 \cdot 3$ $= 24$

Exercici 3.3: Tradueix a llenguatge assemblador MIPS les declaracions de variables globals i la funció *main* de la Figura 3.2. Recorda que aquesta s'ha de programar com una subrutina, seguint totes les regles estudiades (p. ex., s'ha de preservar el registre *$ra* si el *main* crida altres subrutines). Observa que la declaració de les variables globals *mat1* i *mat4* són les mateixes que en l'exercici 3.1, i que les adreces dels elements `mat1[4][3]` i `mat4[0][2]` són les que ja has calculat a l'exercici 3.2.

```
mat1 :  .space 120
mat4:   .word  2,3,1,2,4,3
col:    .word  2

main:
la $s0, mat1
la $a0, mat4
lw $a1, 8($a0)
la $a2, 0($a2)
jal subr
sw $v0, 108($s0)
la $a0, mat4
li $a1, 1
li $a2, 1
jal subr
sw $v0, 0($s0)
jr $ra
```

Exercici 3.4: Tradueix a assemblador MIPS la subrutina *subr* de la Figura 3.2

```
la   $t0, mat1
sll  $t1, $a2, 4
sll  $t2, $a2, 3
addu $t1, $t1, $t2
addiu $t1, $t1, 20
addu $t0, $t0, 20
addu $t0, $t0, $t1
sll  $t1, $a1, 3
addu $t2, $a1, $a2
sll  $t2, $t2, 2
addu $t1, $t1, $t2
addu $t1, $t1, $a0
lw   $t1, 0($t1)
sw   $t1, 0($t0)
move $v0, $a1
jr $ra
```

Exercici 3.6: Tradueix a MIPS el programa de la Figura 3.3 usant la tècnica d'accés seqüencial per recórrer la matriu (versió que apareix a la Figura 3.4). Recorda que la funció *main* s'ha de programar com una subrutina, seguint les regles pertinents.

```
main:
addiu   $sp, -4($sp)
sw  $ra, 0($sp)
la  $s0, result
la  $a0, mat
jal  suma_columna
sw  $v0, 0($sp)
lw  $ra, 0($sp)
addiu  $sp, $sp, 4
jr  $ra

suma_col
    li  $v0, 0
    addiu  $t0, $a0, 8
    addiu  $t7, $a0, 80
while:
    lw  $t1, 0($t0)
    addu  $v0, $v0, $t1
    addiu  $t0, $t0, 24
    ble  $t0, $t7, while
    jr  $ra
```