# Tema 4. Matrius

## Estructura de Computadors (EC)

Rubèn Tous

rtous@ac.upc.edu
Computer Architecture Department
Universitat Politecnica de Catalunya


UNIVERSITAT POLITÈCNICA
DE CATALUNYA
BARCELONATECH

# Índex

# 4.1 Les instruccions MIPS mult, mflo, mfhi

- Multiplicació de dos nombres enters de n i m bits dóna un resultat de (potencialment) n+m bits (excepte per al cas n o m=1, ja que no produeix carry).
- Multiplicació de dos nombres de 32 bits dóna un resultat de 64 bits.

```
mult rs, rt# $hi:$lo <- rs * rt
mflo rd# rd <- $lo
mfhi rd# rd <- $hi
```

# 4.2 Matrius

## Matriu

Agrupació multidimensional d'elements del mateix tipus i dentificats per un índex [0, N-1] en cada dimensió.

Estudiarem les matrius de 2 dimensions, però tot serà extrapolable a qualsevol nombre de dimensions.

# 4.2 Matrius

$$
\texttt{mat[NF][NC]} = \begin{vmatrix}
\texttt{mat[0][0]} & \texttt{mat[0][1]} & \ldots & \texttt{mat[0][NC-1]} \\
\texttt{mat[1][0]} & \texttt{mat[1][1]} & \ldots & \texttt{mat[1][NC-1]} \\
\ldots & \ldots & & \ldots \\
\texttt{mat[NF-1][0]} & \texttt{mat[NF-1][1]} & \ldots & \texttt{mat[NF-1][NC-1]}
\end{vmatrix}
$$

# 4.2.1 Declaració i emmagatzematge

En C:

```
1  ...
2  int mat[NF][NC];
3  ...
4  int mit[2][3]={{-1, 2, 0},{1, -12, 4}};
```

- En C, els elements es guarden a memòria per files".
- És un conveni (en Fortran, les matrius es guarden per columnes").

# 4.2.1 Declaració i emmagatzematge

En MIPS:

```
1         .data
2  ...
3         .align 2
4  mat:   .space NF*NC*4
5  ...
6  mit:   .word -1, 2, 0, 1, -12, 4
```

- Hem de respectar les regles d'alineació.

# 4.2.2 Accés a un element qualsevol (aleatori)

## mat[i][j]

```
@mat[i][j] = mat + (i*NC + j)*T
```

# 4.2.2 Accés a un element qualsevol (aleatori)

## Exemple general:

```
1  int mat[NF][NC];
2  void func() {
3      int i, j, k;
4      ...
5      k = mat[i][j];
6  }
```

## En MIPS:

```
1  la   $t3, mat
2  li   $t4, NC
3  mult $t4, $t0       # ... = i*NC
4  mflo $t4
5  addu $t4, $t4, $t1  # $t4 = i*NC + j
6  sll  $t4, $t4, 2    # $t4 = (i*NC + j)*4
7  addu $t3, $t3, $t4  # $t3 = mat + (i*NC + j)*4
8  lw   $t2, 0($t3)    # k = mat[i][j]
```

# 4.2.2 Accés a un element qualsevol (aleatori)

Exemple si la columna és constant::

```
1 k = mat[i][5];
```

En MIPS:

```
1 la   $t3, mat + 5*4
2 li   $t4, NC*4
3 mult $t4, $t0
4 mflo $t4
5 addu $t3, $t3, $t4
6 lw $t2, 0($t3)
```

# 4.2.2 Accés a un element qualsevol (aleatori)

Exemple si la fila és constant:

```
1 k = mat[3][j];
```

En MIPS:

```
1 la   $t3, mat + 3*NC*4
2 sll  $t4, $t1, 2
3 addu $t3, $t3, $t4
4 lw   $t2, 0($t3)
```

# 4.2.2 Accés a un element qualsevol (aleatori)

Exemple si fila i columna són constants:

```
1 k = mat[3][5];
```

En MIPS:

```
1 la $t3, mat + 3*NC*4 + 5*4
2 lw $t2, 0($t3)
```

# 4.3 Accés seqüencial a un vector

```
1  void clear1(int array[], int nelem)
2  {
3      int i;
4      for (i=0; i<nelem; i+=1)
5      array[i] = 0;
6  }
```

En MIPS, mitjançant **accés aleatori**:

```
1  clear1:
2      move  $t0, $zero      # i=0
3  loop1:
4      bge   $t0, $a1, end1
5      sll   $t1, $t0, 2
6      addu  $t2, $a0, $t1
7      sw    $zero, 0($t2)
8      addiu $t0, $t0, 1     # i++
9      b loop1
10 end1:
```

# 4.3 Accés seqüencial a un vector

Si les adreces de qualsevols dos elements consecutius estan separades per una **distància constant** podem fer servir la tècnica d'**accés seqüencial**:

```
1  clear2:
2      move   $t1, $a0          #punter = $t1 = &array[0]
3      move   $t0, $zero
4  loop2:
5      bge    $t0, $a1, end2
6      sw     $zero, 0($t1)     #*punter = 0
7      addiu  $t1, $t1, 4       #punter++
8      addiu  $t0, $t0, 1
9      b      loop2
10 end2:
```

Què hi hem guanyat? El bucle loop2 té una instrucció menys

# 4.3 Accés seqüencial a un vector

- La distància (constant) entre dos elements = **STRIDE**.
- $@n_{i+1} = @n_i$ + stride
- A l'exercici anterior l'STRIDE era 1 enter o 4 bytes si l'expressem en baix nivell.

# 4.3 Accés seqüencial a un vector

Podem calcular l'STRIDE de manera metòdica:

```
     @ni_next =    &array[i+1]= array + i + 1
    -@ni         -(&array[i]) = -(array + i)
    ------------------------------------------
STRIDE =                          1   element
STRIDE (bytes) =                  1*4 bytes
```

# 4.3 Accés seqüencial a un vector

1. Inicialitzar el punter:

```
move   $t1, $a0
```

2. Llegir/escriure l'element mitjançant el punter.

```
sw     $zero, 0($t1)
```

3. punter = punter + STRIDE

```
addiu $t1, $t1, 4
```

# 4.3 Accés seqüencial a un vector

Optimització: eliminació de la variable d'inducció:

```
1  clear3:
2      move   $t1, $a0
3      sll    $t2, $a1, 2
4      addu   $t3, $a0, $t2 # $t3 = array[nelem]
5  loop3:
6      bgeu   $t1, $t3, end3
7      sw     $zero, 0($t1)
8      addiu  $t1, $t1, 4
9      b      loop3
10 end3:
```

# 4.3 Accés seqüencial a un vector

Optimització: avaluació de la condició al final del bucle:

```
1  clear4:
2       move   $t1, $a0
3       sll    $t2, $a1, 2
4       addu   $t3, $a0, $t2
5       bgeu   $t1, $t3, end4
6  loop4:
7       sw     $zero, 0($t1)
8       addiu  $t1, $t1, 4
9       bltu   $t1, $t3, loop4
10 end4:
```

# 4.3 Accés seqüencial a un vector

Expressar l'accés seqüencial en alt nivell

```
1  void clear3(int *array, int nelem)
2  {
3      int *p;
4      for (p=array; p < &array[nelem]; p=p+1)
5          *p = 0;
6  }
```

Però el compilador pot usar accés seqüencial independenment de si el codi en alt nivell fa servir punters o no.

# 4.4 Accés seqüencial a una matriu

Una fila:

```
for (i=0; i<NC, i++)
    mat[1][i] = 0;
```

# 4.4 Accés seqüencial a una matriu

INIT:

```
punter = &mat[1][0]
```

STRIDE:

```
  @ni_next =    &mat[1][i+1] = mat + 1NC + i + 1
 -@ni      = -(&mat[1][i+1])= -(mat + 1NC + i)
-------------------------------------------------
STRIDE =                                1   element
STRIDE (bytes) =                        1*T bytes
```

# 4.4 Accés seqüencial a una matriu

Una columna:

```
for (j=0; j<NF, i++)
    mat[j][1] = 0;
```

# 4.4 Accés seqüencial a una matriu

INIT:

```
punter = &mat[0][1]
```

STRIDE:

```
   &mat[j+1][1] = mat + jNC + NC + 1
 -(&mat[j][1])= -(mat + jNC + 1)
-------------------------------------
STRIDE =                    NC  elements
STRIDE (bytes) =             NC*T  bytes
```

# 4.4 Accés seqüencial a una matriu

Diagonal principal:

```
for (i=0; i<NC, i++)
    mat[i][i] = 0;
```

# 4.4 Accés seqüencial a una matriu

INIT:

```
punter = &mat[0][0]
```

STRIDE:

```
   &mat[i+1][i+1] = mat + iNC + NC + i + 1
 -(&mat[i][i])= -(mat + iNC + i)
-------------------------------------------
STRIDE =                        NC+1  elements
STRIDE (bytes) =               (NC+1)*T  bytes
```

# 4.4 Accés seqüencial a una matriu

Diagonal secundària:

```
for (i=0; i<NC, i++)
    mat[i][NC-1-i] = 0;
```

# 4.4 Accés seqüencial a una matriu

INIT:

```
punter = &mat[0][NC-1]
```

STRIDE:

```
   &mat[i+1][NC-1-(i+1)] = mat+iNC+NC + NC-1-i-1
 -(&mat[i][NC-1-i])=     -(mat+iNC    + NC-1-i)
-------------------------------------------------
STRIDE =                              NC-1   elements
STRIDE (bytes) =                     (NC-1)*T  bytes
```