Solució comentada dels problemes proposats per al Tema 4

Problema 4.1

Donades les següents declaracions en C, on `NF` i `NC` són constants:

```
int mat[NF][NC];
int f() {
     int i, j;
     ... /* Aquí va el codi de cada apartat */
}
```

Escriu el codi MIPS pertanyent a la funció *f* per calcular les adreces dels següents elements fent servir el mínim nombre d'instruccions, i deixant el resultat en $v0. Suposem que *i* i *j* estan emmagatzemades en $t0 i $t1, respectivament.

Apartat (e). Calcular l'adreça de l'element: `mat[i+5][j-1]`

Comentaris:

```
@mat[i+5][j-1] = mat + (i+5)*NC*4 + (j-1)*4
               = mat + i*NC*4 + j*4 + (5*NC*4 + (-1)*4)
```

El primer i l'últim terme són constants

Solució:

```
la      $v0, mat + 5*NC*4 - 4
li      $t2, NC*4
mult    $t0, $t2
mflo    $t2
addu    $v0, $v0, $t2
sll     $t2, $t1, 2
addu    $v0, $v0, $t2
```

Apartat (f): Calcular l'adreça de l'element `mat[i*10+4][6]`

Comentaris:

```
@mat[i*10+4][6] = mat + (i*10+4)*NC*4 + 6*4
                = mat + i*10*NC*4 + (4*NC*4 + 6*4)
```

El primer i l'últim terme són constants

Solució:

```
la      $v0, mat + 16*NC*4 + 6*4
li      $t2, 10*NC*4
mult    $t2, $t0
mflo    $t2
addu    $v0, $v0, $t2
```

Problema 4.3

Donada la següent declaració d'una variable global (on N és una constant):

```
int M[N][N];
```

Tradueix a assemblador MIPS el següent fragment de codi en C, usant el mínim nombre d'instruccions, i suposant que les variables *i* i *suma* ocupen els registres $t0, $t1:

```
suma += M[i][i+1] - M[i+1][i];
```

Comentaris:

La solució simplista consistiria a calcular les dues adreces efectives per separat. Però és fàcil adonar-se que com que són adreces molt relacionades, tan sols cal calcular-ne una perquè l'altra està ubicada a una distància coneguda: una fila menys un element.

Alternativament, com que es tracta de dos elements adjacents a l'element `M[i][i]` de la diagonal, també podem calcular l'adreça d'aquest darrer element, i obtenir les altres dues adreces sumant-li distàncies conegudes: `M[i][i+1]` està 1 element més endavant, i `M[i+1][i]` està 1 fila més endavant.

```
@M[i][i] = M + i*N*4 + i*4 = M + i*(N+1)*4
```

Solució:

```
la      $t2, M
li      $t3, (N+1)*4
mult    $t0, $t3
mflo    $t3                 # i*(N+1)*4
addu    $t2, $t2, $t3       # @M[i][i]
lw      $t3, 4($t2)         # M[i][i+1]
lw      $t4, N*4($t2)       # M[i+1][i]
subu    $t3, $t3, $t4
addu    $t1, $t1, $t3
```

Problema 4.7

Donades les següents declaracions en C:

```
int mati[5][4];

void func(int veci[4]) {
int j;
     for (j=0; j<4; j++)
             veci[j] = j;
}

void main() {
int i;
     for (i=0; i<5; i++)
             func(&mati[i][0]);
}
```

Apartat (a). Tradueix a MIPS la subrutina *func* utilizant accés seqüencial al vector `veci` i suposant que la variable `j` ocupa el registre `$t0`.

Comentaris:

func és una subrutina uninivell, per tant no cal salvar registres si sols usem temporals. Com que tampoc té variables locals a la pila, no cal establir un bloc d'activació.

Per a l'accés seqüencial, hem de definir i inicialitzar un punter al primer element a recórrer. Si usem `$a0` com a punter, ja el tenim inicialitzat!

Com que recorrem tots els elements del vector en ordre creixent i són de tipus int, el stride és de 4 bytes.

Solució:

```
func:
     move    $t0, $zero          # j=0
     li      $t2, 4
for:
     bge     $t0, $t2, fifor     # j<4?
     sw      $t0, 0($a0)         # desreferenciem punter $a0
     addiu   $a0, $a0, 4         # punter+stride
     addiu   $t0, $t0, 1         # j++
     b       for
fifor:
     jr      $ra
```

Nota: Observem que a aquest bucle no se li pot aplicar l'eliminació de la variable d'inducció, ja que la necessitem per emmagatzemar-la a cada element

Apartat (b). Tradueix a MIPS la subrutina *main* utilitzant accés seqüencial a la matriu *mati*. Fes atenció a quins registres fas servir per guardar la variable *i* i el punter.

Comentaris:

El bucle recorre seqüencialment la columna 0 de la matriu, i el stride equival a una fila completa (4*4 = 16 bytes). La funció *main* és una subrutina multinivell. Com en totes les subrutines, cal respectar la regla de salvar registres segurs que es modifiquin. Per determinar quins registres segurs usarem, observem que la crida està dins d'un bucle, i cal conservar d'iteració en iteració els valors del comptador i del punter. Per tant, usarem els registres `$s0` i `$s1` respectivament.

A la pila caldrà salvar `$s0`, `$s1` i `$ra` (12 bytes)

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

     la      $s1, mati     # inicialitzem el punter @mati[0][0]
     move    $s0, $zero    # i=0
for:
     li      $t0, 5
     bge     $s0, $t0, fifor
     move    $a0, $s1      # passem parà metre &mati[i][0]
     jal     func
     addiu   $s1, $s1, 16  # incrementem punter + stride
     addiu   $s0, $s0, 1
     b       for
fifor:
     lw      $s0, 0($sp)
     lw      $s1, 4($sp)
     lw      $ra, 8($sp)
     addiu   $sp, $sp, 12
     jr      $ra
```

Problema 4.8

Donades les següents declaracions en C (on `N` és una constant):

```
void func(int A[N][N]) {
     int i, j, suma=0;
              ... /* aquí va la sentència de cada apartat */
     }
```

Tradueix a MIPS les següents sentències utilitzant la tècnica d'accés seqüencial per als accessos a la matriu A, suposant que pertanyen a la funció *func*, i que les variables *i*, *j*, i *suma* ocupen els registres $t0, $t1, $t2:

Apartat (d):

```
for (i=0; i<N; i+=3)       /* Atenció: la i va de 3 en 3 */
              suma += A[i][N-1-i];
```

Comentari:

Per a aquest recorregut, l'adreça inicial del punter (que posarem en $t3) és:

```
@A[0][N-1] = $a0 + (N-1)*4
```

Com que recorre l'antidiagonal, el punter avança `(N-1)*4` bytes per cada fila, i com avança les files de 3 en 3, el stride és:

```
stride = 3*(N-1)*4 = (N-1)*12
```

Solució:

```
     move    $t0, $zero
     addiu   $t3, $a0, (N-1)*4   # punter inicial
     li      $T4, n
for:
     bge     $t0, $t4, fifor
     lw      $t5, 0($t3)
     addiu   $t3, $t3, 12*(N-1)  # punter + stride
     addu    $t2, $t2, $t5       # suma += ...
     addiu   $t0, $t0, 3         # i+=3
     b       for
fifor:
```

Nota: Aquesta solució admet les optimitzacions d'eliminació de la variable d'inducció, i també de conversió while en do-while. Resulta molt recomanable intentar aplicar-les.

Problema 4.14

La funció *fila_dispersa* escriu els elements de la fila *i* de la matriu dispersa representada pels vectors *A*, *IA* i *JA* (que rep com a paràmetres), en les corresponents posicions de la matriu *mat* (variable global):

```
int mat[N][M];
void fila_dispersa(int *A, int *IA, int *JA, int i) {
     int k;
     for (k=IA[i]; k<IA[i+1]; k++)
              mat[i][JA[k]] = A[k];
}
```

Apartat (a): Dels següents accessos a memòria que fa aquest bucle, ¿quins es poden traduir fent servir la tècnica d'accés seqüencial, i quins sols es poden traduir amb accés aleatori?

```
IA[i], IA[i+1], JA[k], mat[i][JA[k]], A[k]
```

Resposta:

Accés seqüencial: `JA[k]`, `A[k]`, ja que fan recorreguts amb stride constant (=4 bytes)

Accés aleatori: `mat[i][JA[k]]`, ja que l'índex `JA[k]` no asegura un creixement lineal

`JA[i]`, `JA[i+1]`, ja que no fan cap recorregut (i no varia)

Apartat (b): Tradueix la funció *fila_dispersa* usant la tècnica d'accés seqüencial per als recorreguts que ho permetin, suposant que *k* es guarda al registre $t0.

Comentari:

Usarem un punter `$a0` per recórrer `A[k]`, i un punter `$a2` per recórrer `A[k]`

Si denominem `X` al valor `JA[k]`, l'adreça de `mat[i][X]` es pot escriure:

```
@mat[i][X] = @mat[0][0] + i*M*4 + X*4
```

D'aquesta fórmula, els 2 primers termes no varien en tot el bucle, i els podem extraure fora del bucle com a *invariants*. El tercer terme varia d'una iteració a una altra.

Solució:

```
        # Carreguem en $t0 i $t1 els valors IA[i] i IA[i+1]
        sll     $t2, $a3, 2         # i*4
        addu    $t2, $t2, $a1       # @IA[i] = @IA[0] + i*4
        lw      $t0, 0($t2)         # k = IA[i] (comptador k inicial)
        lw      $t1, 4($t2)         # IA[i+1]   (là-mit del comptador k)

        # Inicialitzem els punters $a0 = @A[k], $a2 = @JA[k]:
        sll     $t2, $t0, 2         # k*4
        addu    $a0, $a0, $t2       # @A[k] = @A[0] + k*4
        addu    $a2, $a2, $t2       # @JA[k] = @A[0] + k*4

        # Calculem la part invariant de @mat[i][JA[k]]: @mat[0][0] + i*M*4
        la      $t3, mat
        li      $t2, M*4
        mult    $t2, $a3
        mflo    $t2                 # i*M*4
        addu    $t2, $t2, $t3       # @mat[0][0] + i*M*4
for:
        bge     $t0, $t1, fifor

        # Part esquerra de l'assignació
        lw      $t4, 0($a0)         # carreguem A[k] desreferenciant punter $a0

        # Part dreta de l'assignació
        lw      $t3, 0($a2)         # carreguem X = JA[k] desreferenciant  a2
        addiu   $a0, $a0, 4         # punter $a0 + stride
        addiu   $a2, $a2, 4         # punter $a2 + stride
        sll     $t3, $t3, 2         # X*4 (part variable de l'adreça de mat[i][X]
        addu    $t3, $t3, $t2       # adreça @mat[i][X]
        sw      $t4, 0($t3)         # escriu A[k] a la posició mat[i][X]

        # Final del bucle
        addiu   $t0, $t0, 1# k++
        b       for
fifor:
        jr      $ra
```