Q 1.4

Elemento	Endereço (Hexadecimal)
A [32,24]	1000 2060h
B [32,24]	1000 6060h
C [32,24]	1000 A060h
D [32,24]	1000 E060h
T1 [32,24]	1001 2060h
T2 [32,24]	1001 6060h

Q1.6

Tag	Index	Offset
22	6	4

Q1.7

Elemento	Endereço (Hexadecimal)	Tag	Index	Offset
A [32,24]	1000 2060h	**0001 0000 0000 0000 0010 00**	**00 0110**	**0000**
B [32,24]	1000 6060h	**0001 0000 0000 0000 0110 00**	**00 0110**	**0000**
C [32,24]	1000 A060h	**0001 0000 0000 0000 1010 00**	**00 0110**	**0000**
D [32,24]	1000 E060h	**0001 0000 0000 0000 1110 00**	**00 0110**	**0000**
T1 [32,24]	1001 2060h	**0001 0000 0000 0001 0010 00**	**00 0110**	**0000**
T2 [32,24]	1001 6060h	**0001 0000 0000 0001 0110 00**	**00 0110**	**0000**

Q1.8

O aumento de vias da cache teria um impacto positivo neste código, pois como pode ser visto pelo exercício 1.7 sempre que ele acede a uma posição que tem o mesmo Index de outra matriz, ele altera a linha toda guardada na cache e, portanto, perde essa linha fazendo com que o próximo acesso da matriz anterior origine um miss. Já com várias vias é possível guardar várias linhas com o mesmo Index sem ser preciso escrever por cima dos outros dados.

Por exemplo no código em baixo se só houver uma via então sempre que o i = j = k (ou seja coluna e linha das duas matrizes iguais, logo Index igual) vai fazer com que o "A" aceda a cache, escreva a linha toda e depois este será reescrito pelo B, o que resultará no novo miss assim que o A tentar aceder à coluna a seguir.

```
/* First part: T1 = A*B */
for (i=0;i<N;i++)
   for (j=0;j<N;j++){
      aux1=0;
      for (k=0;k<N;k++)
         T1[i][j] += A[i][k] * B[k][j];
   }
```

Q2.5

Nº de Hits:	256434	**Nº total de acessos à memória:**	540675
Nº de Misses:	284241	**Miss-Rate (total) [%]:**	52.57

Q2.6

Tempo médio de cada acesso:	1 + 0.5257 × 100 = 53.57 ns
Tempo total de acesso à memória:	53.57 × 540675 = 28963959 ns = 28, 963959 ms

Q2.7

Tempo médio de cada acesso:	100 ns
Tempo total de acesso à memória:	100 × 540675 = 54067500 ns = 54,067500 ms

Q2.8

Como esperado se retiramos a cache o tempo de acesso total aumenta consideravelmente, neste caso quase que dobra, pois todas a vezes que quer aceder a um dado tem de esperar os 100 ns da DRAM em vez dos apenas 1 ns da cache (em caso de hit).

Q3.2

O resultado da operação não é afetado pois é indiferente somar todos os parâmetros entre duas matrizes por linha ou por coluna.

Q3.3

A alteração influencia positivamente a taxa de hits ao acessar a cache porque como antes somávamos todos os elementos de cada coluna ele nunca conseguia aproveitar os endereços seguintes da matriz carregados no primeiro miss, pois são reescritos nas linhas de baixo e então estes nunca são aproveitados para a próxima coluna. Já no caso da alteração sempre que dá um miss ele carrega para a cache o endereço dos próximos 3 elementos da linha o que faz com que haja hits em todas as matrizes nas próximos 3 operações e depois volta a dar um miss, repetindo este procedimento até ao final.

Q3.4

A ordem mais favorável é a depois da alteração pois terá uma maior taxa de hits no acesso à cache o que levará a um menor tempo total de acesso à memória.

Q3.5

	Ordem i,j	Ordem j,i
Nº de acessos à memória (total):	540675	540675
Miss-Rate (total) [%]:	50.87	52.57
Tempo médio de cada acesso [ns]:	51.87	53.57
Tempo total de acesso à memória [ms]:	28.044812	28.963959

Q3.6

Vendo pelo ripes observamos que a maioria dos misses acontece na parte 1 (271953 miss de um total de 284241 miss (versão original)), logo já era expectável que a mudança dos for's da segunda parte não trouxesse ganhos tão significativos face à versão original.

Q4.1

```
addi t0, zero, 4
mul a3, s9, t0 # a3 -> 4*N
```

Alteração do local para antes da preliminar part para ser usado lá.

```
# Preliminary part #
add a4, zero, s6
add a1, zero, s2
mv t0, zero
mv t1, zero
for1:
addi t0, t0, 1
lw t6,0(a1)
sw t6, 0(a4)
addi a4, a4, 4
add a1, a1, a3
```

Inicialização de auxiliares de B e de T2 e função que transpõe a matriz B para T2

```
blt t0, s9, for1

add a1, zero, s2
addi t1, t1, 1
addi t0, zero, 4
mul t0, t0, t1
add a1, a1, t0
add t0, zero, zero
blt t1, s9, for1
```

```
# Inicializacao de auxiliares:
add a0, zero, s1 # a0 -> *A
add a1, zero, s2 # a1 -> *B
add t0, zero, s6 ----> Mudança para o endereço de T2 em vez de B
add a2, zero, s5 # a2 -> *T1

add t1, zero, zero
f1loop1: add t2, zero, zero
f1loop2: add t3, zero, zero
add t4, zero, zero
f1loop3: lw t5, 0(a0)
lw t6, 0(t0)
mul t5, t5, t6
add t4, t4, t5
addi a0, a0, 4
addi t0, t0, 4 ----------> Em vez de saltar uma linha salta uma coluna
addi t3, t3, 1
blt t3, s9, f1loop3
sw t4, 0(a2)
addi a2, a2, 4
sub a0, a0, a3
addi t2, t2, 1
blt t2, s9, f1loop2
add a0, a0, a3
add t0, zero, s6 ----------> Rebobina para o início da matriz
addi t1, t1, 1
blt t1, s9, f1loop1
```

Q4.3

Nº de Hits:	473090	**Nº total de acessos à memória:**	548867
Nº de Misses:	75777	**Miss-Rate (total) [%]:**	13.81

Q4.4

A alteração influencia positivamente a taxa de acessos à cache porque como antes multiplicávamos cada linha de A por cada coluna de B ele nunca conseguia aproveitar os endereços seguintes da matriz B carregados no primeiro miss, pois são reescritos à medida que usamos as linhas de baixo e então estes nunca são aproveitados para a próxima coluna. Já no caso da alteração sempre que dá um miss ele carrega para a cache o endereço dos próximos 3 elementos das linhas da matriz A e B o que faz com que haja hits nas duas matrizes nas próximos 3 operações e depois volta a dar um miss, seguindo assim até ao final.

Q4.5

Tempo médio de cada acesso:	1 + 0.1381 × 100 = 14.81 ns
Tempo total de acesso à memória:	14.81 × 548867 = 8128720 ns = 8, 128720 ms
Speedup:	$\frac{28,963959}{8,128720} = 3.563$

Q5.2

Nº de Hits:	343298	**Nº total de acessos à memória:**	548867
Nº de Misses:	205569	**Miss-Rate (total) [%]:**	37.45

Tempo médio de cada acesso:	1 + 0.3745 × 100 = 38.45 ns
Tempo total de acesso à memória:	38.45 × 548867 = 21103936 ns = 21, 103936 ms
Speedup:	$\frac{21,103936}{28,963959} = 0.7286$

Q5.3

Como explicado no exercício 1.8 quando existe apenas uma via e o programa tenta aceder a um dado de uma matriz X, o que ele faz é alterar a linha, guardando todos os dados dessa matriz com o mesmo Index na cache. Já quando outra matriz, por exemplo Y, acede a esse mesmo Index esta reescreve os dados que lá estavam e, portanto, perde se essa linha fazendo com que se o próximo acesso da matriz X origine um miss. Como este código está sempre a usar dados de mais de uma matriz para a mesmas contas então este problema é bastante relevante.

Ex do lab

Q6.1

```
add a0, zero, s1 # a0 -> *A
add a1, zero, s2 # a1 -> *B
add t0, zero, s6 # t0 -> *auxB = B
add a2, zero, s5 # a2 -> *T1

add a5, zero, s3 # a0 -> *C
add a6, zero, s4 # a1 -> *D
```

Inicialização das variáveis auxiliares que estavam na parte 2 com as devidas alterações

```
add t1, zero, zero
f1loop1: add t2, zero, zero
f1loop2: add t3, zero, zero
add t4, zero, zero
f1loop3: lw t5, 0(a0)
lw t6, 0(t0)
mul t5, t5, t6
add t4, t4, t5
addi a0, a0, 4
addi t0, t0, 4
addi t3, t3, 1
blt t3, s9, f1loop3
sub a0, a0, a3
addi t2, t2, 1

lw t6, 0(a5)
mv t3, t4
mul t4, s7, t3
mul t5, s8, t6
add t5, t5, t4
sw t5, 0(a6)
addi a5, a5, 4
addi a6, a6, 4
blt t2, s9, f1loop2
```

Adição do código da parte 2, com a devidas alterações das variáveis

```
add a0, a0, a3
add t0, zero, s6
addi t1, t1, 1
blt t1, s9, f1loop1

li a7, 1
ecall
```

Q6.3

Nº de Hits:	466946	**Nº total de acessos à memória:**	540675
Nº de Misses:	73729	**Miss-Rate (total) [%]:**	13.64

Q6.4

Ao unificar as duas partes do código num único código deixamos de necessitar de guardar os valores na matriz T1 (e por consequência deixamos de dar loads), o que reduz significativamente o nº total de acessos à memória cache.

Q6.5

Tempo médio de cada acesso:	1 + 0.1364 × 100 = 14.64 ns
Tempo total de acesso à memória:	14.64 × 540675 = 7915482 ns = 7. 915482 ms
Speedup:	$\frac{28.044812}{7.915482} = 3.543$

Q6.7

Tempo médio de cada acesso:	1 + 0.1364(20 + 0.1 × 100) = 5.092 ns
Tempo total de acesso à memória:	5.092 × 540675 = 2753117 ns = 2. 753117ms
Speedup:	$\frac{28.044812}{2.753117} = 10.187$