INSTITUTO SUPERIOR TÉCNICO

DEPARTAMENTO DE ENGENHARIA INFORMÁTICA

ORGANIZAÇÃO DE COMPUTADORES

LEIC

Conjunto de Exercícios VI

Caches (II)

Versão 3.0

2022/2023

Exercício 1

Consider a processor with the following parameters:

Parameter	Value
Base CPI, No Memory Stalls	1.5
Processor Speed	2 GHz
Main Memory Access Time	100 ns
L1 Cache Miss Rate per Instruction	7 %
L2 Cache, Direct-Mapped Speed	12 cycles
Global Miss Rate with L2 Cache, Direct-Mapped	3.5 %
L2 Cache, 8-Way Set Associative Speed	28 cycles
Global Miss Rate with L2 Cache, 8-Way Set Associative	1.5 %

Calculate the CPI for the processor in the table using:

a) no cache,

b) only a first level cache,

c) a first level cache and a second level direct-mapped cache, and

d) a first level cache and a second level eight-way set associative cache.

Exercício 2

The goal of this exercise is to analyze the execution of a small code segment in a 32-bit processor whose memory system has the following characteristics:

- Two caches, a program cache with 4 KB and a data cache with 16 KB.
- Both caches are direct mapped, have blocks with 16 bytes, as well as write-allocate and write-back strategies.
- The caches are non-blocking. A read operation to a memory address that is not present at the cache leads to its immediate loading from memory. As soon as the processor gets the value from the required memory address, block loading continues with the remaining positions that are required to completely fill the block.
- The data buses between the caches and the primary memory, as well as the data buses between the caches and the CPU are 32-bits wide.
- The address space has 2^{32} bytes and is byte addressable.
- Each access to a cache word requires 20 ns. The transfer of each word between the primary memory and the corresponding cache introduces a penalty in the access cycle of 100 ns.
- The internal processor cycle takes 20 ns.

a) For the data cache, identify the addressing bits (index), as well as the bits used for the tag and offset fields.

Consider the following code segment:

```
access() {
   register int i,j,k; /* variables i, j, k stored in registers */
   int a[128,128]; /* [column, row] */
   int b[128,128]; /* 32 bits integers */

   for(i=0; i < 128; i++)
   for(j=0; j < 128; j++)
   a[i,j] = b[i,j] + k;
}
```

Assume that initially all data cache entries are invalidated, the compiler does not implement any code optimization, variables are allocated in contiguous memory positions, using a growing addressing sequence, and by the same order as they were declared (matrices in major-row order: x[0,0], x[0,1], x[0,2], ... , x[1,0], ...).

b) What is the hit rate in the data cache when this program is executed?

c) Compute the data mean access time when this program is executed and compare it with a hypothetical situation where there isn't any cache and the accesses are directly issued to the primary memory.

d) Considering only the above code segment, what is the ideal cache block dimension?

e) Which modifications can be introduced by the programmer in this source code in order to reduce the data mean access time? Point out to what extent the algorithm, the matrix allocation strategy adopted by the compiler, and the cache structure may influence the performance of this matrix operation.

Exercício 3

The memory architecture of an embedded controller is being evaluated. The controller and the memory both have 32-bits words. It is known that the controller has a cache, but its size and internal structure are unknown.

In the first step of this process the following program was run and were recorded the times in the table.

```
#define REPEAT 10
#define MIN 1024 /* 1k */
#define MAX 65536 /* 64k */

register int i, n, range, s; /* allocated in registers; 32 bits integers */
int a[MAX];
register float t1, t2;

for(range= MIN, range<=MAX; range=2*range) { /* 1 - Scan several ranges */
   t1 = get_time (); /* Registers time */
   for (n = 0; n < REPEAT; n++) { /* 2 - In each range repeat sweeps */
      s = 0;
      for (i = 0; i < range; i++) { /* 3 - Sweep the array */
         s = s + a[i];
      }
   }
   t2 = get_time ();
   /* print - Sweep range, elapsed time, average access time */
   printf ( ... , range, t2-t1, (t2-t1)/(REPEAT*range));
}
```

Range [words]	**1k**	**2k**	**4k**	**8k**	**16k**	**32k**	**64k**
t2-t1 [ms]	0.20	0.31	0.60	1.20	15.66	32.78	65.55
Number of accesses to a[]							
Average access time [ns]	20	15	15	15	96	100	100

a) For each range calculate the total amount of accesses to the array a[].

b) Explain the need for the outer loop cycle (performed REPEAT times).

c) Given the obtained time results, estimate the capacity of the cache.

d) Is it possible to estimate the hit time, the miss time, or the miss penalty?

Exercício 4

The goal of this exercise is to analyze the execution of a small code segment in a 32-bit processor family with a Load-Store architecture, and with an address space with 2^{32} bytes, with 32-bit words, byte addressable. There are several different implementations of the memory system with different cache organizations.

Consider the following code segment:

```
access() {
   register int i,j; /* variables i, j stored in registers */
   int a[128,128]; /* [column, row] */
   int b[128,128];
   int c[128,128];

   for(i=0; i < 128; i++)
      for(j=0; j < 128; j++)
         a[i,j] = b[i,j] + c[i,j];
}
```

The compiler does not implement any code optimization; standard precision integers have 32 bits; variables are allocated in contiguous memory positions using a growing addressing sequence and by the same order as they were declared (matrices in major-row order: x[0,0], x[0,1], x[0,2], ... , x[0,127], x[1,0], ...).

a) Assuming that the compiler allocates logical space to the global variables starting at address 0h. Determine the base addresses of arrays a, b, and c.

b) Determine the sequence of accesses to data in memory generated by the processor during the execution of the first three iterations of the inner loop. For each access identify its type (read, or write) and its address.

The memory system has the following characteristics:

- Two caches - a program cache with 4 KB and a data cache with 16 KB.
- Both caches are 2-way set associative, have blocks with 16 bytes, and implement Least Recently Used (LRU) substitution.
- Write-allocate and write-back strategies.
- Non-blocking and critical-word-first loading policies.

- The data buses between the caches and the primary memory, as well as the data buses between the caches and the CPU are 32-bits wide.
- Each access to a cache word requires 20 ns. The transfer of each word between the primary memory and the corresponding cache introduces a penalty in the access cycle of about 100 ns. The internal processor cycle takes 20 ns.

Assume that, initially, all data cache entries are invalidated.

c) For the data cache, identify the addressing bits (index), as well as the bits used for the tag and offset fields.

d) What is the hit rate in the data cache when this program is executed?

e) Based on this hit rate compute the data mean access time and compare this memory system with a simpler system without any cache.

f) Determine the sequence of accesses to data in memory generated by the cache during the execution of the first two iterations of the inner loop. For each access identify its type (read, or write) and its address.

g) By only considering the above code segment, what is the ideal cache block size?

h) Which modifications can be introduced by the programmer in this source code in order to reduce the data mean access time?

i) For the original program evaluate the performance of a memory system with an identical data cache but with write-through no allocate strategy.

j) For the original program evaluate the performance of a memory system with the original data cache enhanced by the inclusion of a fully associative victim cache with two blocks. A miss in the normal cache followed by an hit in the victim cache has a penalty of 20 ns.

References

[1] David Patterson and John Hennessy. *Computer Organization and Design: The Hardware/Software Interface*. Morgan Kaufmann, 4th edition, 2011.

Conjunto de Exercícios VI

Caches (II)

Soluções

Exercício 1

a) CPI = 200

b) CPI = 15.5

c) CPI = 9.34

d) CPI = 6.46

Exercício 2

a) Data cache: 10 bits for index, 4 bits for offset (16 bytes block), and 18 bits for tag.

b) Hit rate = 0%.

c) The memory access time with cache = 120ns. Memory access time without cache = 100 ns.

d) Block size = 1 word.

e) Misalign the arrays.

```
int a[128,128];
int d[BLOCKSIZE];
int b[128,128];
```

or merge the arrays a and b:

```
access() {
   register int i,j,k; /* variables i, j, k stored in registers */
   struct ab
   {
      int a;
      int b;
   };
   struct ab m[128][128];

   for(i=0; i < 128; i++)
      for(j=0; j < 128; j++)
         m[i,j].a = m[i,j].b + k;
}
```

Exercício 3

a) Number of accesses to a[]:

Range [words]	**1k**	**2k**	**4k**	**8k**	**16k**	**32k**	**64k**
t2-t1 [ms]	0.20	0.31	0.60	1.20	15.66	32.78	65.55
Number of accesses to a[]	10,240	20,480	40,960	81,920	163,840	327,680	655,360
Average access time [ns]	20	15	15	15	96	100	100

b) The repetition of the sweep attenuates the contribution of the first accesses to the array which may generate a pattern of hits/misses very different from subsequent sweeps.

c) Cache capacity = 8 K words = 32 K bytes.

d) t_hit around 15 ns to 20 ns, and t_miss ≈ 100 ns
miss_penalty ≈ 85.

Exercício 4

a) base of a = 0h

base of b = 0h + 64 K = 00010000h

base of c = 0h + 64 k + 64 K = 128 K = 00020000h

b) read b[0,0] at 0001 0000h
read c[0,0] at 0002 0000h
write a[0,0] at 0000 0000h
read b[0,1] at 0001 0004h
read c[0,1] at 0002 0004h
write a[0,1] at 0000 0004h
read b[0,2] at 0001 0008h
read c[0,2] at 0002 0008h
write a[0,2] at 0000 0008h

c) 9 bits for index (A4 - A12), 4 bits for offset (A0 - A3), and 19 bits for tag (A13 - A31).

d) Hit rate = 0%.

e) The memory access time (the sum of the cache and memory access times) is 120 ns.
S = t_without cache / t_with cache = 100 ns / 120 ns = 0.83.

f) read b[0,0] miss, to way 0
read 0001 0000h, read 0001 0004h, read 0001 0008h, read 0001 000Ch
read c[0,0] miss, to way 1
read 0002 0000h, read 0002 0004h, read 0002 0008h, read 0002 000Ch
write a[0,0] miss, to way 0
read 0000 0000h, read 0000 0004h, read 0000 0008h, read 0000 000Ch
read b[0,1] miss, to way 1
read 0001 0004h, read 0001 0008h, read 0001 000Ch, read 0001 0000h
read c[0,1] miss, to way 0, write-back
write 0000 0000h, write 0000 0004h, write 0000 0008h, write 0000 000Ch
read 0002 0004h, read 0002 0008h, read 0002 000Ch, read 0002 0000h
write a[0,1] miss, to way 1
read 0000 0004h, read 0000 0008h, read 0000 000Ch, read 0000 0000h

g) 1 word (4 bytes).

h) Misalign one of the arrays:

```
int a[128,128];
int d[BLOCKSIZE];
int b[128,128];
int b[128,128];
```

or merge matrices b and c:

```
access() {
   register int i,j; /* variables i, j stored in registers */
   int a[128,128]; /* [column, row] */
   struct bc {
      int b;
      int c;
   };
   struct bc m[128][128];

   for(i=0; i < 128; i++)
      for(j=0; j < 128; j++)
         a[i,j] = m[i,j].b + m[i,j].c;
}
```

i) Miss rate = 50%.
average memory access time (AMAT) = 63.3 ns
Comparing with the original cache:
Speedup = 120 / 63,3 = 1.90

If we considered that a write miss is a direct access to memory and if the cache is enhanced with a write buffer:
AMAT = 36.7 ns
Speedup = 120 / 36.7 = 3.27

j) The victim cache will store blocks replaced in the "main cache".
average memory access time (AMAT) = 60 ns