7. The Memory Hierarchy (1)

The possibility of organizing the memory subsystem of a computer as a hierarchy, with levels, each level having a larger capacity and being slower than the precedent level, was envisioned by the pioneers of digital computers.

The main argument for having a memory hierarchy is economics: a unique, large memory, running at the CPU's speed, would be prohibitive, if possible at all. What makes the memory hierarchy idea work is the principle of locality.

Example 7.1 MEMORY CHIPS AND THEIR CAPACITY:

How many chips are necessary to implement a 4 MBytes memory:
1) using 64 Kbit SRAM;
2) using 1Mbit DRAM;
3) 64 KBytes using 64 Kbit SRAM and the rest using 1Mbit DRAM.

Answer:
The number of chips is computed as:

$$\frac{\text{Memory capacity (expressed in bits)}}{\text{Chip capacity (expressed in bits)}}$$

or as

$$\frac{\text{Memory capacity (expressed in bytes)}}{\text{Chip capacity (expressed in bytes)}}$$

1)n1 = $2^{22}/2^{13}$ = 512 chips.(using the second formula)
2)n2 = $2^{22}/2^{17}$ = 32 chips.
3)n3 = $2^{16}/2^{13}$ + floor((2^{22}-2^{16})/2^{17}) = 8 + 32(SRAM + DRAM)

SRAM circuits are designed to be very fast but they have smaller capacities than DRAM and are more expensive. DRAMs, on the other hand are slower (tens to hundreds of nanoseconds cycle time), but their capacity is very large. Using only SRAM results in a memory that matches very well the CPU's speed, but is very expensive and bulky, not to mention packaging, cooling and other problems. Using DRAMs results in a small dimension memory (only 16 chips) which is cheap but relatively slow: the CPU will have to wait at every memory access until the operation, read or write, is done.

What we would like is a memory that is fast like the SRAM and, at the same time, as cheap and compact as the DRAM version. With a proper organization solution 3 could be the needed compromise; obviously it won't be as fast as the pure SRAM memory, nor so cheap as the DRAM.

7.1 The Principle of Locality

In running a program the memory is accessed for two reasons:

- read instructions;
- read/write data.

Memory is not uniformly accessed; addresses in some region are accessed more often than others, and some addresses are accessed again shortly after the current access. In other words programs tend to favor parts of the address space at any moment of time.

- **temporal locality:** an access to a certain address tend to be repeated shortly thereafter;

- **spatial locality:** an access to a certain address tends to be followed by accesses to nearby addresses.

The numerical expression of this principle is given by the 90/10 rule of thumb: 90% of the running time of a program is spent accessing 10% of the address space of that program. If this is the case, then it is natural to think at a memory hierarchy: map somehow the most used addresses to a fast memory that needs to represent roughly only 10% of the address space, and the program will run for most of the time (90%) using that fast memory. The rest of the memory can be slower because is accessed less, it has to be larger though. This is not that difficult because slower memories are cheaper.

A memory hierarchy has several levels: the uppermost level is the closest to the CPU and it is the fastest (to match the processor's speed) and the smallest; as we go downwards to the bottom of the hierarchy, each level gets slower and larger as the previous one but with a lower price per bit.

As for the data different levels of the hierarchy hold, each level is a subset of the level below it, in that data in one level can be found in the level immediately below it. Memory items (they may represent instructions or data) are brought into the higher level when they are referred for the first time because there is a good chance they will be accessed again soon, and migrate back to the lower level when room must be made for newcomers.

7.2 Finite memory latency and performance

In Chapter 5 we discussed about the ideal CPI of instructions in the instruction set. At that moment we assumed the memory is fast enough to deliver the item being accessed without introducing wait states. If we look at Figure 5.1, we see that in state Q1 the MemoryReady signal is tested to determine if the memory cycle is complete or not; if not, then the Control Unit returns in state Q1, continuing to assert the control lines necessary for a memory access. Every clock cycle in which MemoryReady = No increases the CPI for that instruction with one. The same is true for load or store instructions. The bigger the memory's response time is, the higher the real CPI for that instruction is. Suppose an instruction has an ideal CPI of n (i.e. there is a sequence of n states in the state-diagram, corresponding to this instruction), and k of them are addressing cycles; k=2 for load/store instructions, and k=1 for all other ones in our instruction set. The ideal CPI for this instruction is:

$CPI_{ideal} = n$ (clock cycles per instruction)

If every addressing cycle introduces w waiting clock cycles, we have the real CPI for this instruction:

CPI_{real} = n + k * w (clock cycles per instruction)

Example 7.2 IDEAL AND REAL CPI:

We want to calculate the real CPI for our instruction set; assume that the ideal CPI is 4 (computed with some accepted instruction mix). Which is the real CPI if every memory access introduces one wait cycle? Loads and stores are 25% of the instructions being executed.

Answer: Using the above formulae we have:
n = 4
w = 1
k = 1in f1=75% of cases
k = 2in f2=25% of cases (the loads and stores)

We get:
CPI_{real} = 4 + (f1*1 + f2*2)*w
CPI_{real} = 4 + (0.75*1 + 0.25*2)*1
CPI_{real} = 4 + 1.25 = 5.25

A machine that had an ideal memory would run faster then the one in our problem by:

$$\frac{CPI_{real}}{CPI_{ideal}} - 1 = \frac{5.25}{4} - 1 = 0.31 = 31\%$$

The above example should make clear what a big difference is between the expectations and the reality. We must have a really fast memory close to the CPU to get full advantage of the CPU's performance. As a final comment, it may happen that the read and write behave differently, in that they require different clock cycles to conclude; the formula giving CPI_{real} will be then slightly modified.

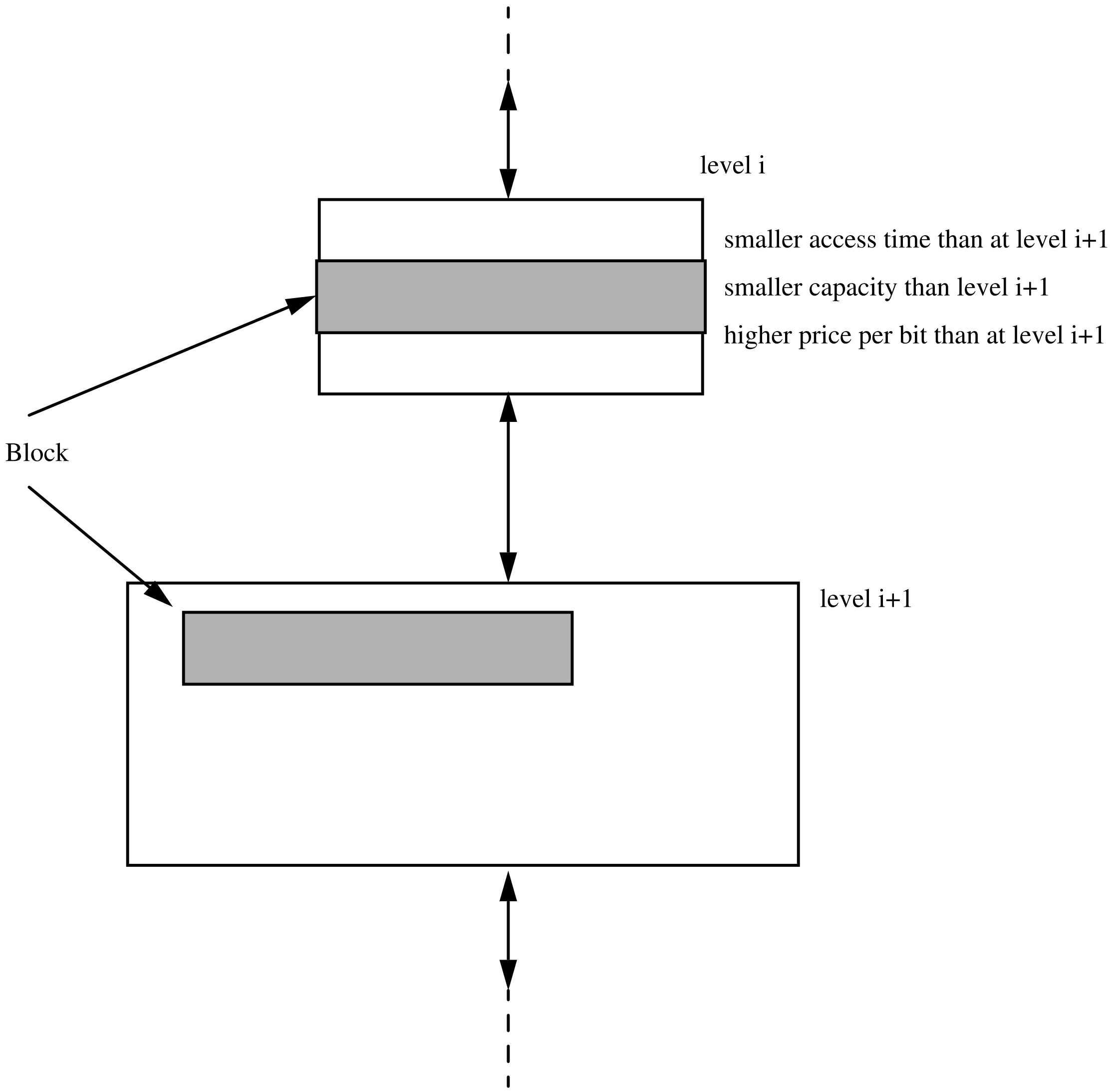

FIGURE 7.1 Two levels in a memory hierarchy. The unit of information that is transferred between levels of hierarchy is called **block.**

7.3 Some Definitions

Information has to migrate between the levels of an hierarchy. The higher a level is in the hierarchy, the smaller its capacity is: it can accommodate only a a small part of the logical address space. Transfers between levels take place in amounts called blocks, as can be seen in Figure 7.1.

A block may be:

- **fixed size**
- variable size.

Fixed size blocks are the most common; in this case the size of the memory is a multiple of the block size.

Note that it is not necessary that blocks between different memory levels all have the same size. It is possible that transfers between level i and i+1 are done with blocks of size b_i, while transfers between level i+1 and i+2 is done with a different block size b_{i+1}. Generally the block size is a power of 2 number of bytes, but there is no rule for this, and, as a matter of fact, deciding the block size is a difficult problem as we shall discuss soon.

The reason for having a memory hierarchy is that we want a memory that behaves like a very fast one and is cheap as a slower one. For this to happen, most of the memory accesses must be found in the upper level of the hierarchy. In this case we say we have a **hit**. Otherwise, if the addressed item is in a lower level of the hierarchy, we have a **miss;** it will take longer until the addressed item gets to the CPU.

The **hit time** is the time it takes to access an item in the upper level of the memory hierarchy; this time includes the time spent to determine if there is a hit or a miss. In the case of a miss there is a **miss penalty** because the item accessed has to be brought from the lower level in memory into the higher level, and then the sought item delivered to the caller (this is usually the CPU). The miss penalty includes two times:

- the **access time** for the first element of a block into the lower level of the hierarchy;
- the **transfer time** for the remaining parts of the block; in the case of a miss a whole block is replaced with a new one from the lower level.

The **hit rate** is the fraction of the memory accesses that hit. The miss rate is the fraction of the memory accesses that miss:

miss rate = 1 - hit rate

The hit rate (or the miss rate if you prefer) does not characterize the memory hierarchy alone; it depends both upon the memory organization and the program being run on the machine. For a given program and machine the hit rate can be experimentally determined as follows: run the program and count how many times the memory is accessed, say this number is N, and how many times accesses are hits, say this number is N_h; then the hit ratio (H) is given by:

$$H = \frac{N_h}{N}$$

The cost of a memory hierarchy can be computed if we know the price per bit C_i and the capacity S_i of every level in the hierarchy. Then the average cost per bit is given by:

$$C = \frac{C_1 * S_1 + C_2 * S_2 + \ldots + C_n * S_n}{S_1 + S_2 + \ldots + S_n}$$

7.4 Defining the performance for a memory hierarchy

The goal of the designer is a machine as fast as possible. When it comes to the memory hierarchy, we want an average access time from the memory as small as possible. The average access time can't be smaller than the access time of the memory in the highest level of the hierarchy, t_{A1}.

For a two level memory hierarchy we have:

t_{av} = hit_time + miss_rate * miss_penalty

where t_{av} is the average memory access time. Do not forget that the hit time is basically the access time of the memory at the first level in the hierarchy, t_{A1}, plus the time to detect if it is a hit or a miss.

Example 7.3 HIT TIME AND ACCESS TIME:

The hit time for a two level memory hierarchy is 40ns, the miss penalty is 400ns, and the hit rate is 99%. Which is the average access time for this memory?

Answer:
The miss rate is:

miss_rate = 1 - hit_rate
miss_rate = 1 - 0.99 = 0.01

The average access time is:

t_{av} = 40 + 0.01 * 400 = 44ns

greater by 10% than the hit time.

The hit time as well as the miss time can be expressed as absolute time, as in the example above, or in clock cycles as in the example 7.4

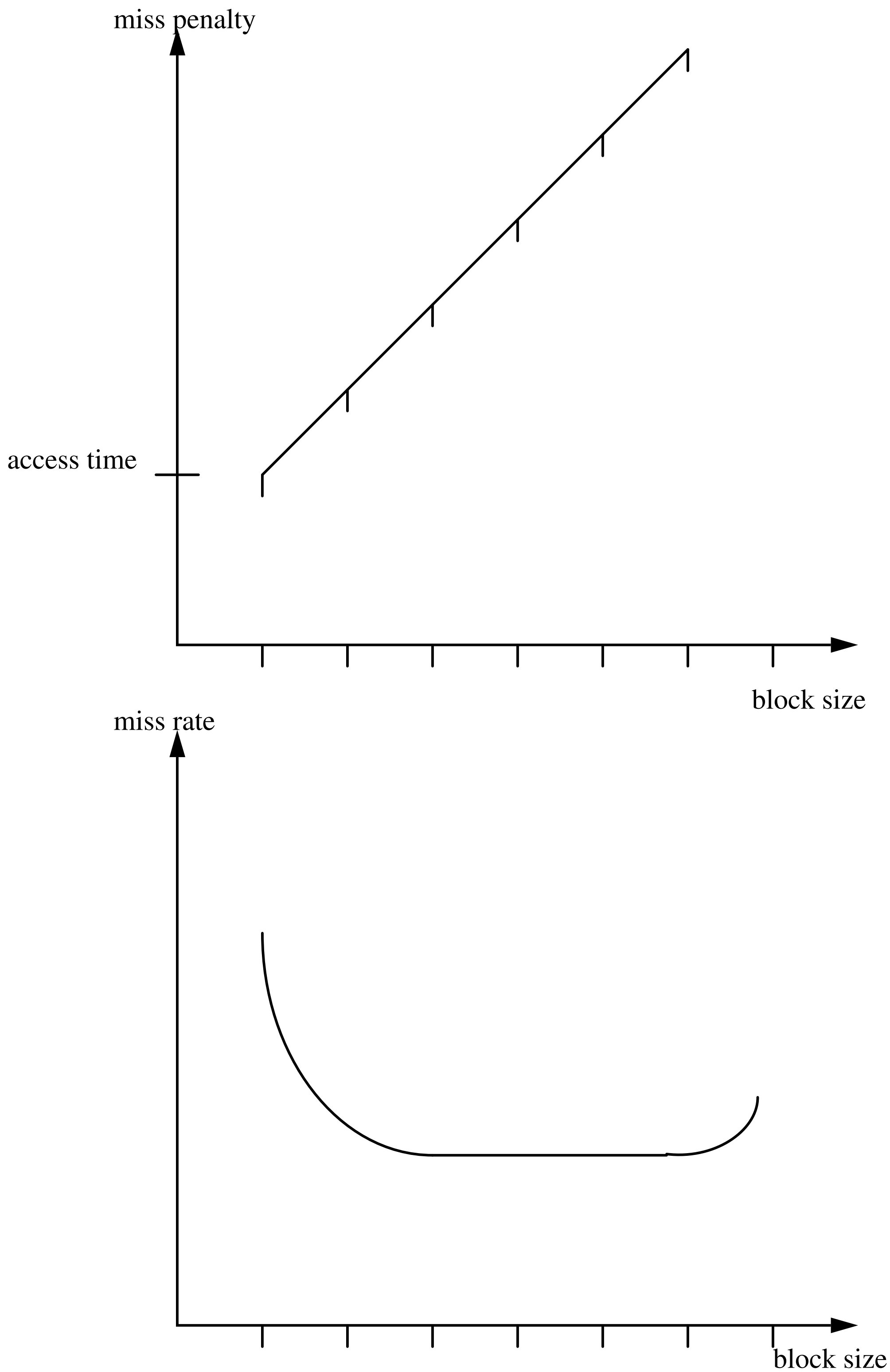

FIGURE 7.2 The relation between block size and miss penalty / miss rate.

Example 7.4 HIT TIME AND ACCESS TIME:

The hit time for a memory is 1 clock cycles, and the miss penalty is 20 clock cycles. What should be the hit rate to have an average access time of 1.5 clock cycles.

Answer:

$$\text{miss\_rate} = \frac{t_{av} - \text{hit\_time}}{\text{miss\_penalty}}$$

$$\text{miss\_rate} = \frac{1.5 - 1}{20} = 0.0025 = 2.5\%$$

hit_rate = 1 - miss_rate = 97.5%

Figure 7.2 presents the general relation between the miss penalty and the block size as well as the general appearance of a relation between the miss rate and the block size, for a given two level memory hierarchy. The minimum value of the miss penalty equals the access time of the memory in the lower level of the memory hierarchy; this happens if there is only one item transferred from the lower level. As the block size increases, the miss penalty increases also, every supplementary item being transferred taking the same amount of time.

On the other hand, the miss penalty decreases for a while when the block size increases, This is due to the spatial locality: a larger block size increases the probability that neighboring items will be found in the upper level of the memory. Above a certain block size, the miss rate starts to increase; as the block size increases the upper level of the hierarchy can accommodate fewer and fewer blocks: when the block being transferred contains more information than needed for the spatial locality properties of the program, it means that time is being spent for useless transfers, and that blocks containing useful information, which could be accessed soon (temporal locality), are replaced in the upper level of the hierarchy.

As the goal of the memory hierarchy is to provide the best access time, the designer must find the minimum of the product:

miss_rate * miss_penalty

7.5 Hardware/Software Support for a Memory Hierarchy

As we have already mentioned, the hit time includes the time necessary to determine if the item being accessed is in the upper level of the memory hierarchy (a hit) or not (a miss). Because this decision must take as little time as possible, it has to be implemented in hardware.

A block transfer occurs at every miss. If the block transfer is short (tens of clock cycles) then it is hardware handled. If the block transfer is large (hundreds to thousands of clock cycles) then it can be software controlled. Which could be the reason for such long lasting transfers? Basically this happens when the difference between memory access times at two levels of memory hierarchy are very large.

Example 7.5 ACCESS TIME AND CLOCK-RATE:

The typical access time for a hard-disk is 10ms. The CPU is running at a 50MHz clock rate. How many clock cycles does the access time represent? How many clock cycles are necessary to transfer a 4KB block at a rate of 10MB/s?

Answer:
The clock cycle is given by:

$$T_{ck}\,[ns] = \frac{1000}{\text{clock\_rate[Mhz]}}$$

$$T_{ck} = \frac{1000}{50} = 20ns$$

The number of clock cycles the access time represent is n_A:

$$n_A = \frac{t_A}{T_{ck}} = \frac{10 * 10^6\,[ns]}{20\,[ns]} = 500,000 \text{ clock cycles}$$

The transfer time is t_T:

$$t_T\,[s] = \frac{\text{block\_size [Bytes]}}{\text{transfer\_rate[Bytes/s]}}$$

$$t_T = \frac{4 * 10^3}{10 * 10^6} = 4 * 10^{-3}\,s = 4ms$$

The number of clock cycles the transfer represents is n_T:

$$n_T = \frac{t_T\,[ns]}{T_{ck}\,[ns]} = \frac{4 * 10^6}{20} = 200,000 \text{ clock cycles}$$

This example clearly shows that a block transfer from the disk can be resolved in software, in the sense that it is the CPU that takes all necessary actions to start the disk accessing process; a few thousand clock cycles are around 1% of the disk access time.

When block transfers are short, up to tens of clock cycles, the CPU waits until the transfer is complete. On the other hand, for long transfers, it would be a waste to let the CPU wait until the transfer is complete; in this case it is more appropriate to switch to another task (process) and works until an interrupt from the accessed devices informs the CPU that the transfer is complete; then the instruction that caused the miss can be restarted (obviously there must be hardware + software support to restart instructions).

7.6 How Does Data Migrate Between the Hierarchy's Levels

At every memory access it must be determined somehow if the access is a hit or miss; as such the question that can be asked is:

- **how is a block identified if it is or not in the upper level of the hierarchy?**

In the case of a miss data has to be brought from a lower level in the hierarchy into a higher level; the question here is:

- where can the block be placed in the upper level?

Bringing a new block into the upper level means that it has to replace some other block here; the question is:

- **which block should be replaced on a miss?**

As we mentioned, a lower level of the hierarchy contains the whole information in its upper level; for this to be true we must know what happens when a write takes place in the upper level:

- **what is the write strategy?**

These questions may be asked for any two neighbor levels of the hierarchy, and they will help us take the proper decisions in design.

Exercises

7.1 Consider a n-level memory hierarchy; the hit-rate H_i for the i-th level is defined as the probability to find the information requested by CPU in level i. The information in memory at level i also appears in the level i+1: hence $H_i < H_{i+1}$. The bottom most level of the hierarchy (disk or tape usually) contain the whole information, such that the hit ratio at this level is 1 (H_n=1). Derive an expression for the average access time t_{av} in this memory.

7.2 Consider the following characteristics of a 3-level memory hierarchy:

Level i	S_i [KB]	C_i [$/bit]	t_{Ai} [ns]	H_i
1 (cache)	2	0.05	20	0.95
2 (main memory)	2000	0.005	200	0.9999
3 (disk)	200,000	0.00001	5,000,000	1

Which is the average memory access time? You may assume that the block transfer time from level i is roughly t_{Ai}, and the hit time at level i is roughly t_{Ai}.