# Virtual Machine, and Memory Management

1

# Virtual machines (VM)

Virtual machines (VM) were first developed in the mid-1960s, and they have remained an important part of mainframe computing over the years.

*System Virtual Machines*. IBM VM/370

Largely ignored in the single-user PC era in the 1980s and 1990s.

They have recently gained popularity due to

■ The increasing importance of isolation and security in modern systems

■ The sharing of a single computer among many unrelated users, for Cloud computing

A single computer runs multiple VMs and can support a number of different *operating systems, and also a number of different versions of the same operating systems*.

On a conventional platform, a single OS “owns” all the hardware resources, but with a VM, multiple OS all share the hardware resources.

2

# Hypervisor, Host and Guest

**Virtual machine monitor (VMM) or Hypervisor,** the software that supports VMs, determines how to map virtual resources to physical resources

**Host***,* the underlying hardware platform, and its resources are shared among the **guest** VMs.

VMs improve protection allow us to:

1. ***Managing software***.
   VMs provide an abstraction that can run the complete software stack, even including old operating systems like DOS.

**2.** ***Managing hardware*****.**
One reason for multiple servers is to have each application running with the compatible version of the operating system on separate computers, as this separation can improve dependability.

Processor virtualization produces overhead.

3

# As the VM works

To "virtualize" the processor, the VMM must control just about everything —access to privileged state, I/O, exceptions, and interrupts—even though the guest VM and OS presently running are temporarily using them.

For example, in the case of a timer interrupt,
the VMM would suspend the currently running guest VM, save its state, handle the interrupt, determine which guest VM to run next, and then load its state. Guest VMs that rely on a timer interrupt are provided with a virtual timer and an emulated timer interrupt by the VMM.

If a guest OS attempts to access or modify information related to hardware resources via a privileged instruction—for example, reading or writing a status bit that enables interrupts—it will trap to the VMM. The VMM can then affect the appropriate changes to corresponding real resources.

To be in charge, the VMM must be at a higher privilege level than the guest VM, which generally runs in user mode; this also ensures that the execution of any privileged instruction will be handled by the VMM.

4

# Overhead of processor virtualization

The overhead of processor virtualization depends on the workload.

User-level processor-bound programs have zero virtualization overhead, because the OS is rarely invoked, so everything runs at native speeds.

I/O-intensive workloads are generally also OS-intensive, executing many system calls and privileged instructions that can result in high virtualization overhead.

On the other hand, if the I/O-intensive workload is also I/O-bound, the cost of processor virtualization can be completely hidden, since the processor is often idle waiting for I/O.

5

# Basic system requirements

The basic system requirements to support VMMs are:

- At least two processor modes, system and user.
- A privileged subset of instructions that is available only in system mode, resulting in

a trap if executed in user mode; all system resources must be controllable just via these instructions.

Hence, if any instruction that tries to read or write such sensitive information traps when executed in user mode, the VMM can intercept it and support a virtual version of the sensitive information, as the guest OS expects.

A way to remap the address to physical memory.

6

# Virtual memory

The main memory can act as a “cache” for the secondary storage.
Differing historical roots have led to the use of different terminology. A virtual memory block is called a ***page***, and a virtual memory miss is called a **page fault**.

7

# Virtual Memory motivations

Historically, there were two major motivations:

1. to allow efficient and safe sharing of memory among several programs,
   - such as for the memory needed by multiple virtual machines for Cloud computing, and to
2. remove the programming burdens of a small, limited amount of main memory.

8

# Virtual Memory protection and requirements

- Protection via virtual memory
  - Keeps processes in their own memory space
    - to allow multiple virtual machines to share the same physical memory, we must be able to protect the virtual machines from each other, ensuring that a program can just read and write the portions of main memory that have been assigned to it.
- Role of architecture:
  - Provide user mode and supervisor mode
  - Protect certain aspects of CPU state
  - Provide mechanisms for switching between user mode and supervisor mode
  - Provide mechanisms to limit memory accesses
  - Provide TLB to translate addresses

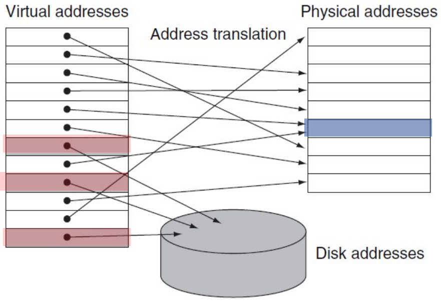
9

# Relocation of addresses

Virtual memory also simplifies loading the program for execution by providing relocation.

This relocation allows us to load the program anywhere in main memory.

All virtual memory systems in use today relocate the program as a set of fixed-size blocks (pages), thereby eliminating the need to find a contiguous block of memory to allocate to a program; instead, the operating system needs only to find enough pages in main memory.



**A virtual page may be absent from main.**
In that case, the page resides on disk.

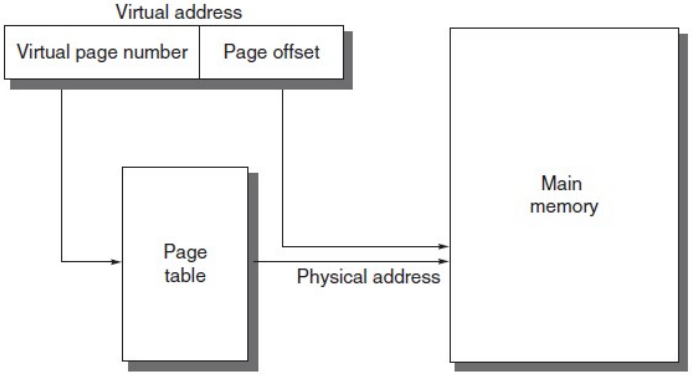
**Physical pages can be shared by having two virtual addresses point to the same physical address.**
This capability is used to allow two different programs to share data or code.

10

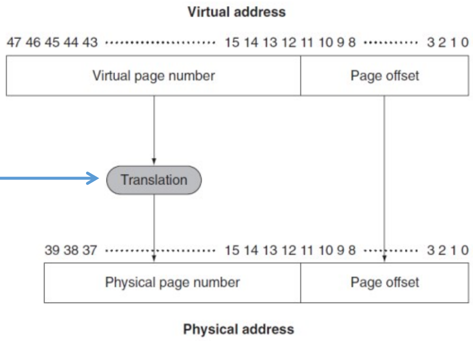
# Virtual Memory

Main memory need contain only the active portions of the many virtual machines, just as a cache contains only the active portion of one program.

The principle of locality enables virtual memory as well as caches, and virtual memory allows us to share the processor efficiently as well as the main memory.



# Table size



**ARMv8**

- It has a 64-bit **virtual address**, the upper 16 bits are not used, so the address to be mapped is 48 bits.
- The **physical memory** is 1 TiB, or $2^{40}$ bytes, which needs a 40-bit address.
- Page size = $2^{12}$ bytes = 4 kbytes
- Number of Virtual pages = $2^{(48-12)}$ = $2^{36}$= 64 G virtual pages
- Number of Virtual pages = $2^{(40-12)}$ = $2^{28}$= 256 M physical pages

Having a larger number of virtual pages than physical pages is the basis for the illusion of an essentially unbounded amount of virtual memory.

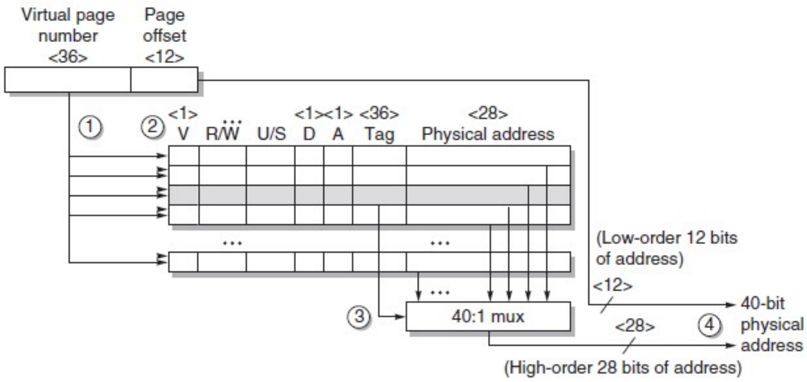
# Page Fault cost

A page fault to disk will take millions of clock cycles to process.
This enormous miss penalty, dominated by the time to get the first word for typical page sizes, leads to several key decisions in designing virtual memory systems:

- Pages should be large (4 Kbytes to 64 Kbytes ) enough to try to amortize the high access time.
- Organizations that reduce the page fault rate are attractive.
- Page faults can be handled in software because the overhead will be small compared to the disk access time.
  - In addition, software can afford to use smart algorithms for choosing how to place pages because even little reductions in the miss rate will pay for the cost of such algorithms.
- Write-through will not work for virtual memory, since writes take too long.
  - Instead, virtual memory systems use write-back.

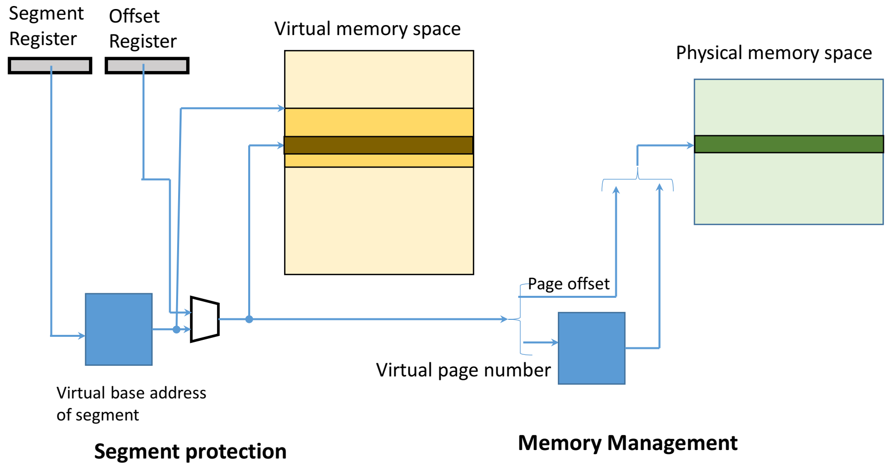
# Virtual Memory

ARM8  supports three options for granule size: 4, 16, and 64 Kbyte.



# Segmentation

In segmentation, an address consists of two parts: a segment number and a segment offset.



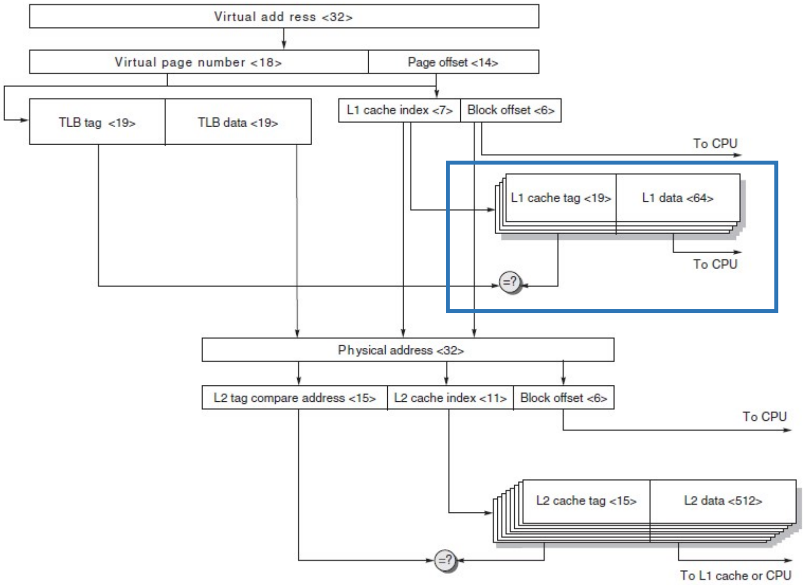
# ARM Cortex-A8

## Organization of a two-level cache hierarchy using virtually indexed caches

- The Cortex-A8 is a configurable core that supports the ARMv7 instruction set architecture
- Cortex-A8 IP core is used in the Apple iPad and smartphones by several manufacturers including Motorola and Samsung.
- Two-level cache hierarchy with the first level being a pair of caches (for I & D), each 16 KB or 32 KB.
- The optional second-level cache when present is eight-way set associative and can be configured with 128 KB up to 1 MB

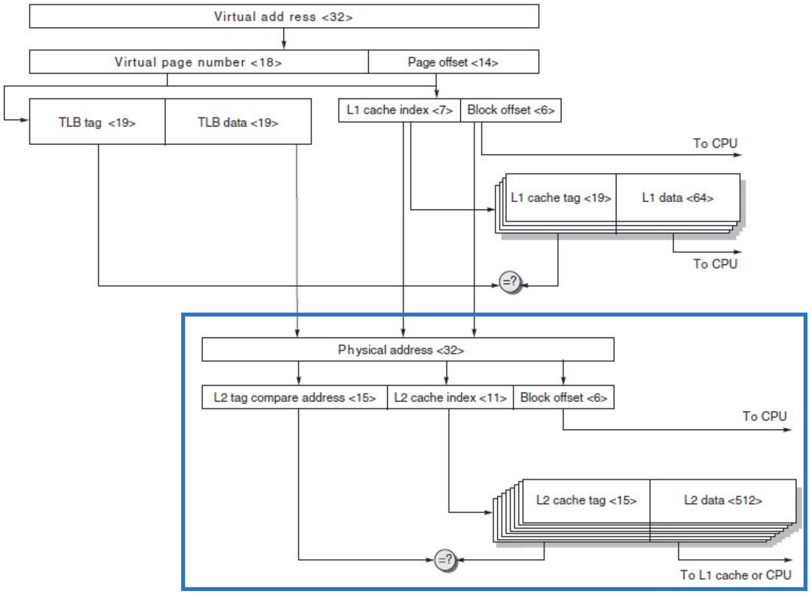
17

## ARM Cortex-A8



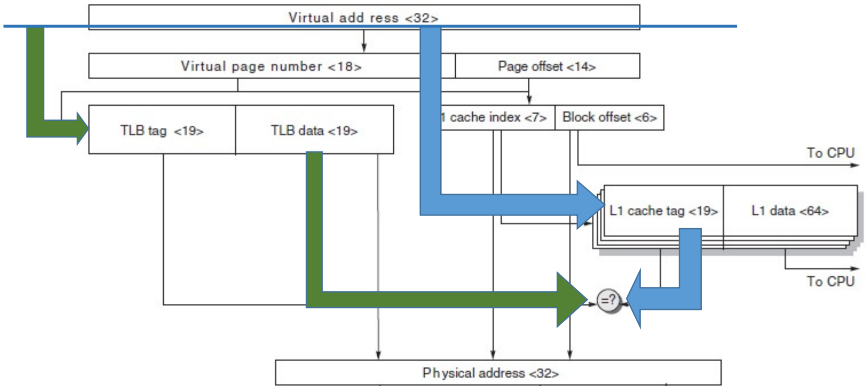
18

# ARM Cortex-A8



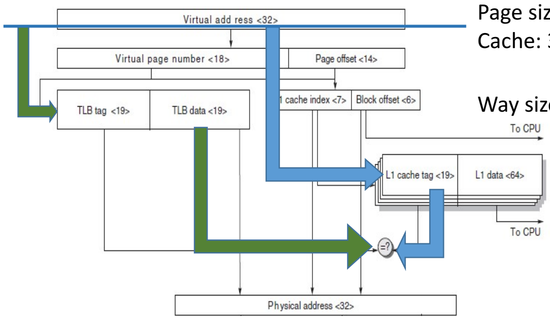
19

# ARM Cortex-A8



20

# ARM Cortex-A8



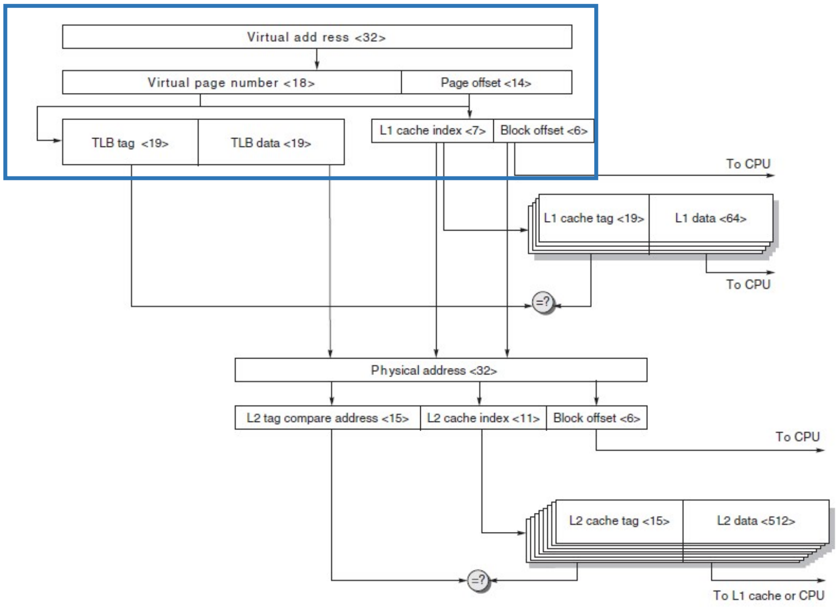
Page size of vitual memory: 16 kbytes
Cache: 32 kbytes, 4-way, 64-byte block

Way size = $\frac{cache\ size}{number\ of\ ways}$ = $\frac{32\ kbytes}{4}$ = 8 kbytes

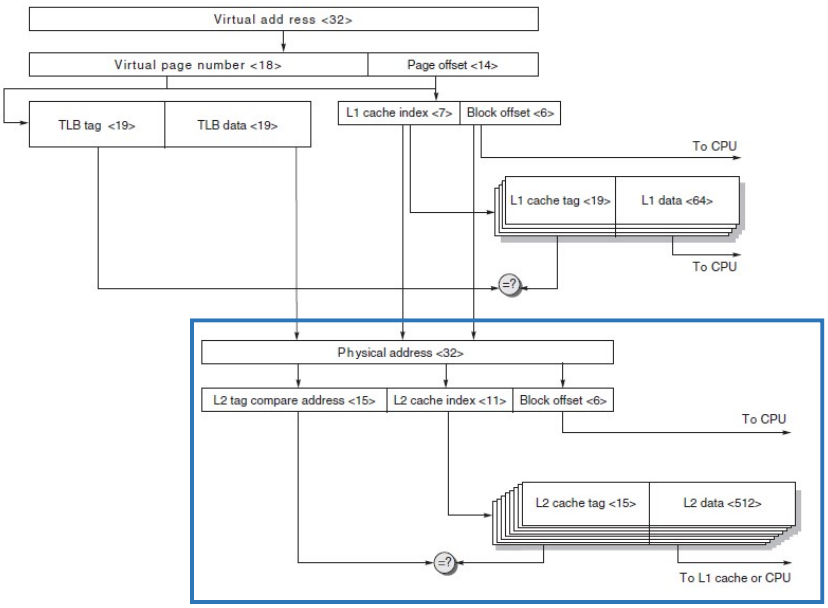
$$page\ size\ of\ virtual\ memory \geq size\ of\ the\ cache\ way$$

$$16\ kbytes \geq 8\ kbytes$$

# ARM Cortex-A8



# ARM Cortex-A8



23

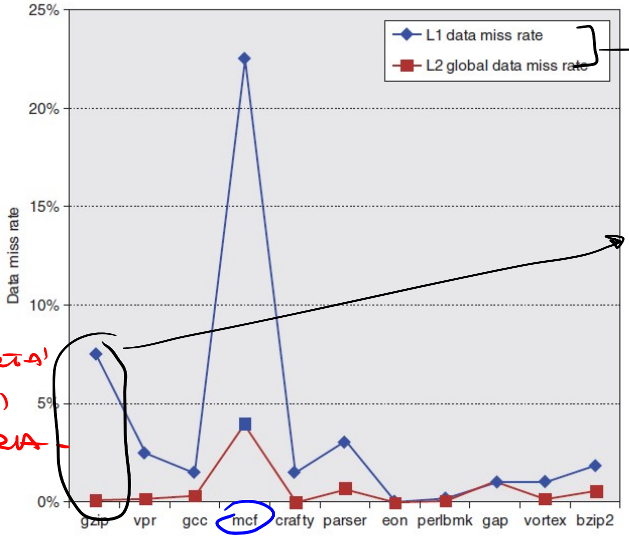
# *Performance of the Cortex-A8 Memory Hierarchy*

VOGLIAMO RIDURRE L'OVERHEAD DOVUTO AL COSTO DELLE MISS -

MEGLIO LAVORARE SUL PRIMO LIVELLO O SUL SECONDO?

VOGLIAMO RIDURRE LA PENALITA' MEDIA VERSO IL TEMPO DI ACCESSO IN MEMORIA -

SUL PRIMO LIVELLO



KILLER APPLICATION

2 LIVELLI DI CACHE

PERCHE' HA QUESTE PERFORMANCE?

GZIP

SOURCE

DESTINATION

SPATIAL LOCALITY ✓

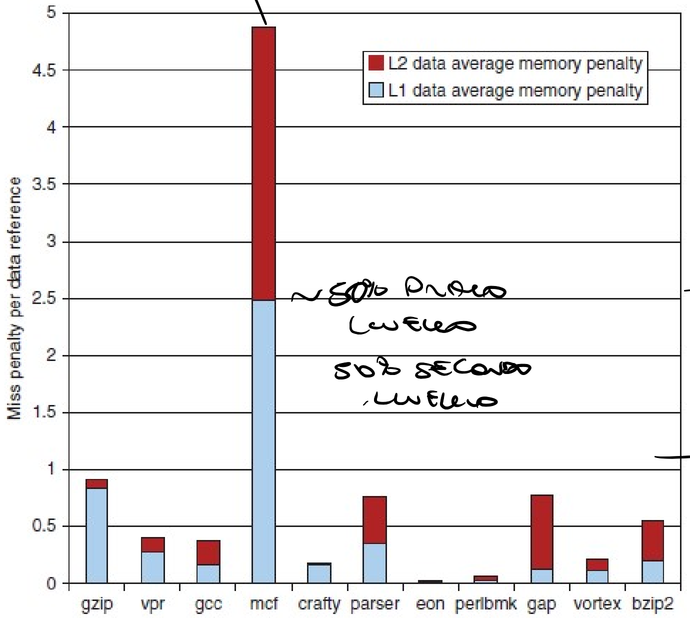
TEMPORAL LOCALITY

LEGGO SOLO UNA VOLTA -

24

KILLER APPLICATION PER LE CACHE

## The average access penalty per data memory reference for L1 and L2



~50% PRIMO LIVELLO

50% SECONDO LIVELLO

**L1 miss penalty** for a 1 GHz Cortex-A8 is 11 clock cycles → NON E' UNA BUONA IDEA AUMENTARE IL NUMERO DI LIVELLI

↓

AUMENTA I CONSUMI

**L2 miss penalty** is 60 clock cycles

# The Processor Performance Equation

CPU time CPU **=** clock cycles for a program **×** Clock cycle time

$$\text{CPI} = \frac{\text{CPU clock cycles for a program}}{\text{Instruction count}}$$

$$\text{CPU time} = \left( \sum_{i=1}^{n} \text{IC}_i \times \text{CPI}_i \right) \times \text{Clock cycle time}$$

where IC*i* represents the number of times instruction *i* is executed in a program and CPI*i* represents the average number of clocks per instruction for instruction *i*.

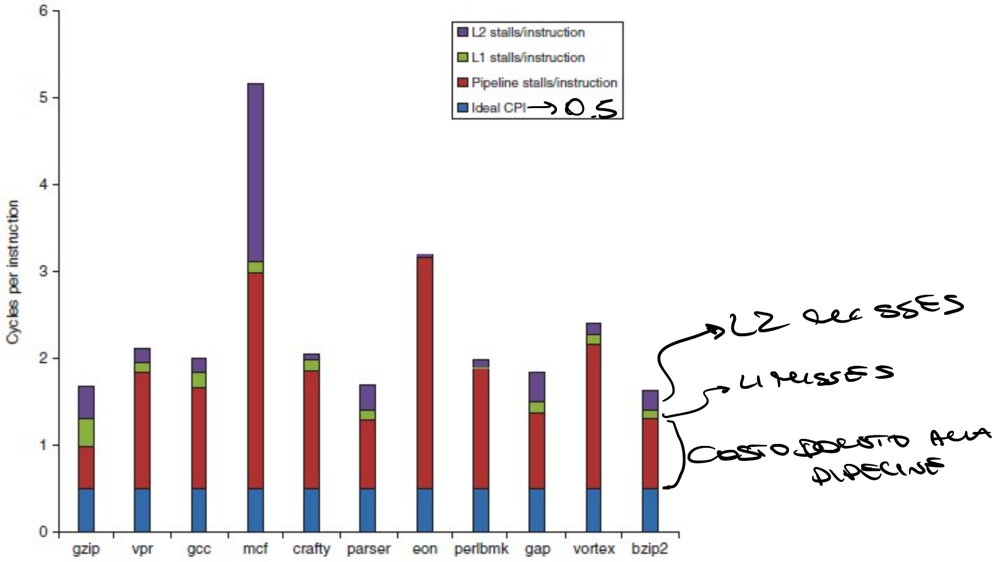
CPU time = Instruction count × Cycles per instruction × Clock cycle time

*Clock cycle time*—Hardware technology and organization

*CPI*—Organization and instruction set architecture

*Instruction count*—Instruction set architecture and compiler technology

# The average memory access penalty per data memory reference



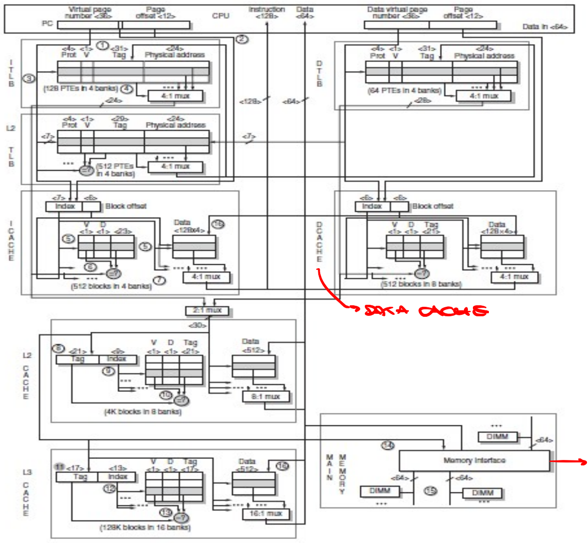
27

# The Intel Core i7

- i7 supports the x86-64 instruction set architecture, a 64-bit extension of the 80x86 architecture.
- I7 can execute up to four 80x86 instructions per clock cycle.
- i7 uses 48-bit virtual addresses and 36-bit physical addresses.
- Memory management is handled with a two level TLB

28

# The Intel i7 memory hierarchy



DATA CACHE

ULTIMA CACHE CONDIVISA

29

# I7: the caches

| Characteristic | L1 | L2 | L3 |
|---|---|---|---|
| Size | 32 KB I/32 KB D | 256 KB | 2 MB per core |
| Associativity | 4-way I/8-way D | 8-way | 16-way |
| Access latency | 4 cycles, pipelined | 10 cycles | 35 cycles |
| Replacement scheme | Pseudo-LRU | Pseudo-LRU | Pseudo-LRU but with an ordered selection algorihtm |

PER RIDURRE IL NUMERO
DI BIT NECESSARI

OBIETTIVO; RIDURRE CPI DOVUTO A
INSTRUCTION ACCESS - 4 WAY
DATA ACCESS - 8 WAY

PER RISOLVERE I PROBLEMI DOVUTI A UNA ACCESS AL [illegible] CLUSTER AUMENTO # WAY

30

## Characteristics of the i7's TLB structure

PER ACCEDERE ALLA CACHE CONFRONTO 16 TAG.

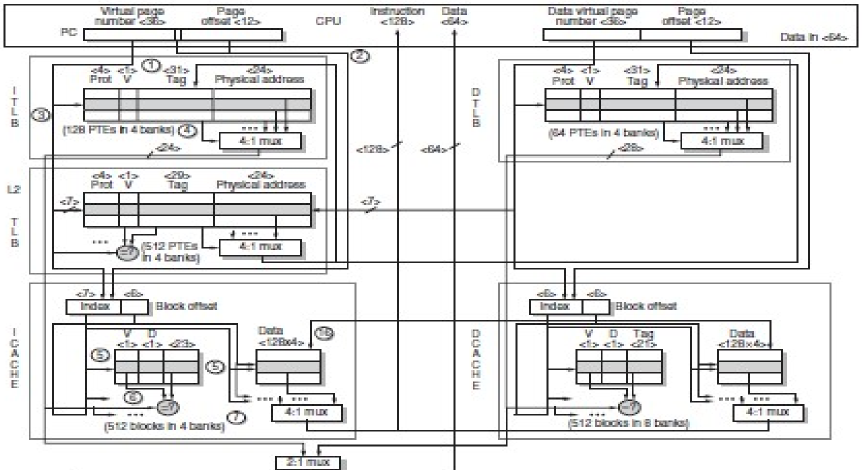
NUMERO REGISTRI = 16, FULL ASSOCIATIVE CACHE.

| Characteristic | Instruction TLB | Data DLB | Second-level TLB |
|---|---|---|---|
| Size | 128 | 64 | 512 |
| Associativity | 4-way | 4-way | 4-way |
| Replacement | Pseudo-LRU | Pseudo-LRU | Pseudo-LRU |
| Access latency | 1 cycle | 1 cycle | 6 cycles |
| Miss | 7 cycles | 7 cycles | Hundreds of cycles to access page table |

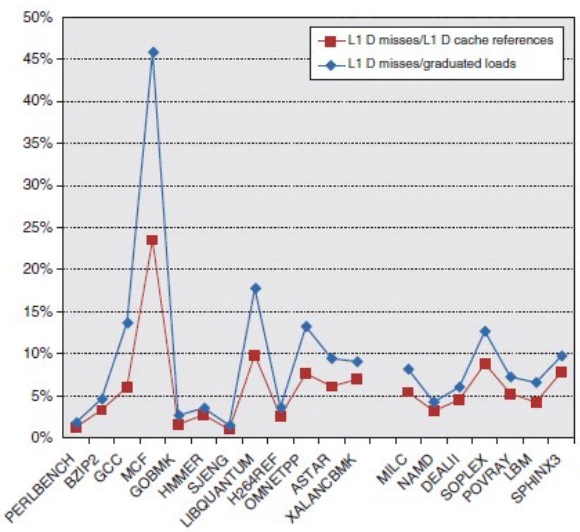
OCCORRE RIDURRE [illegible]!

AUMENTO DIMENSIONI
CACHE USATA PER
ISTRUZIONI

## The Intel i7 memory hierarchy

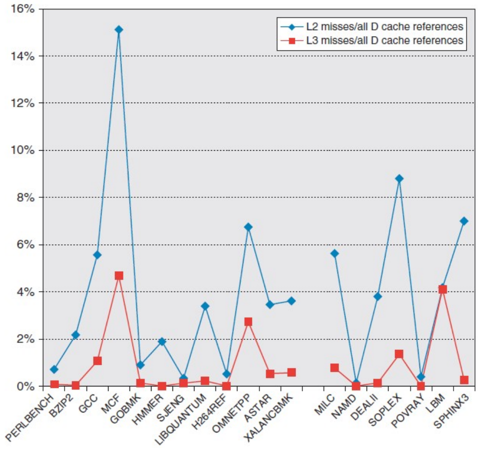


# The L1 data cache miss rate for 17 SPECCPU2006 benchmarks



33

# The L2 and L3 data cache miss rates for 17 SPECCPU2006 benchmarks



34