Computer Architecture

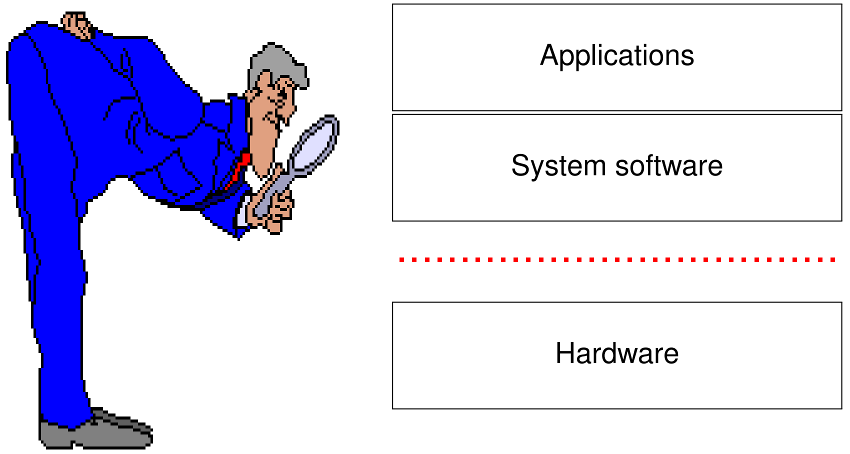
The Art of designing computers based on engineering principles and quantitative performance evaluations

Architecture:

- instruction set architecture
- implementation
  - organization
  - hardware

The role of the computer architect

Make design decisions across the interface between hardware and software in order to meet functional and performance goals.



What computer architecture?

Designing

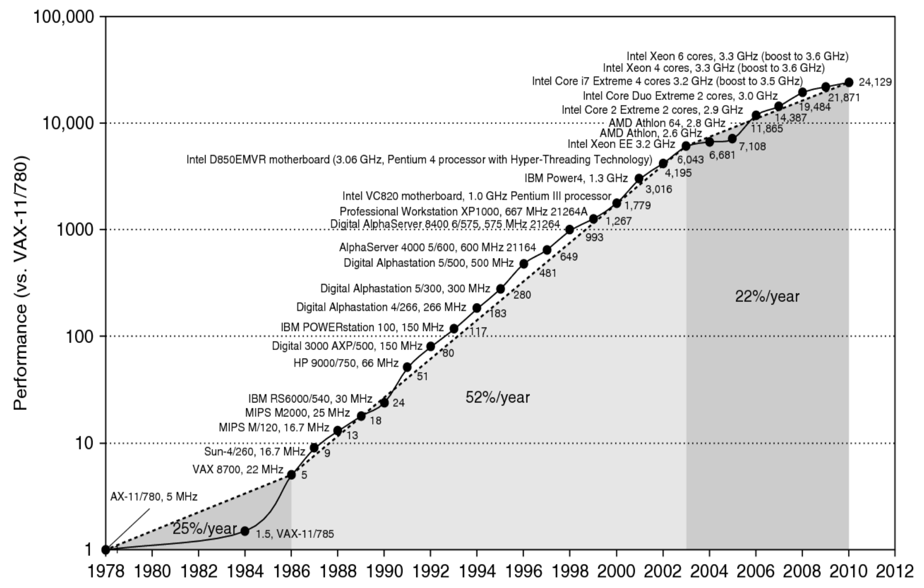
- ISA
- Organization
- Hardware

to meet

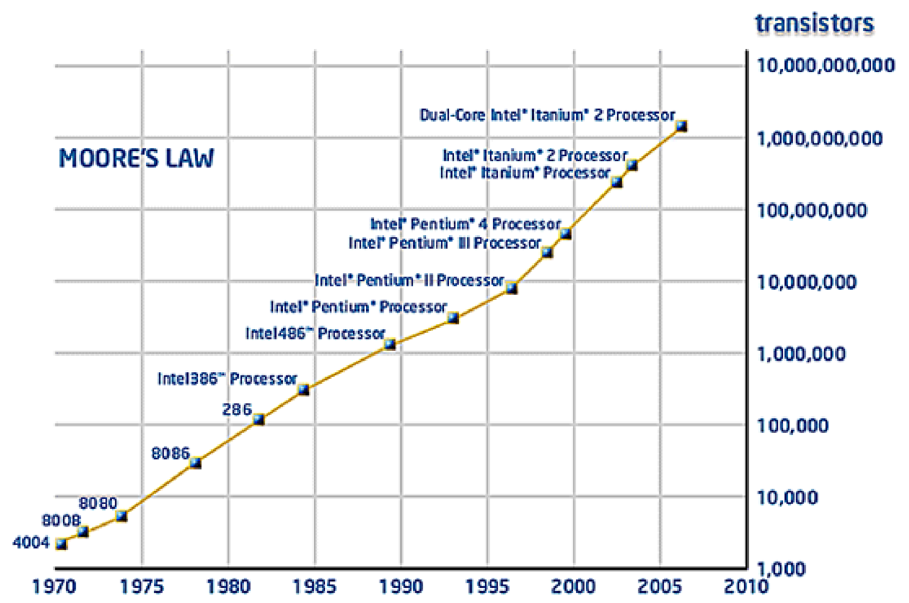
- functional requirements (applications, standards, ...)
- price
- power
- performance
- availability
- dependability

Outline

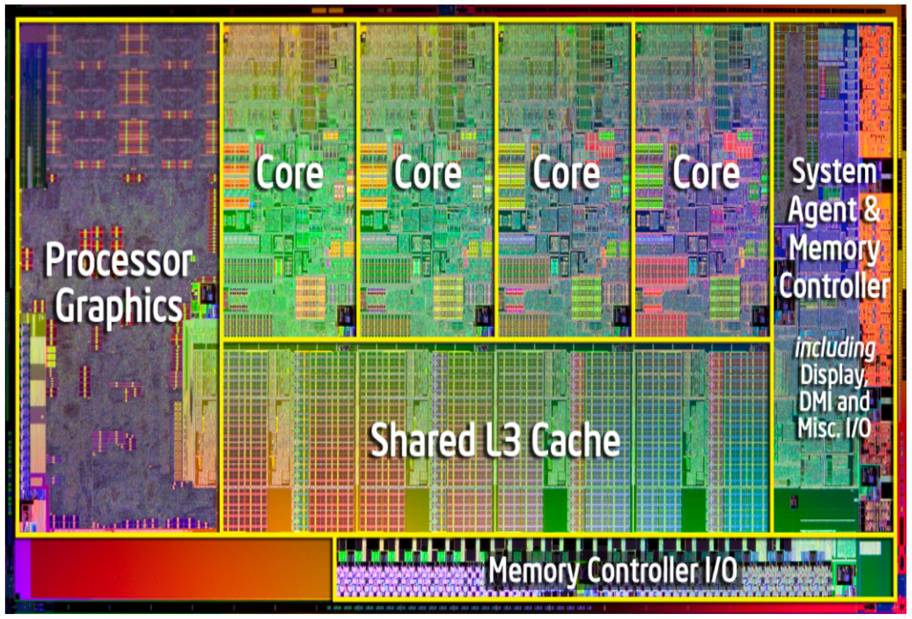
Performance of processors



Transistors in a CPU



Transistors in a CPU



Metrics of performance

- Time to complete a task (T_{exe})
  - Execution time, response time, latency
- Task per day, hour, second, ... (Performance)
  - Throughput, bandwidth

Application	$\Longleftarrow$	Answers/month
Programming language	$\Longleftarrow$	Response time (seconds)
	$\Longleftarrow$	Operations/second
Compiler		
Instruction set	$\Longleftarrow$	MIPS/MFLOPS
Data-path control	$\Longleftarrow$	Megabytes/second
Functional units		
Transistors, wires, pins	$\Longleftarrow$	Cycles per second (clock rate)

MIPS = millions of instructions per second
MFLOPS = millions of FP operations per second

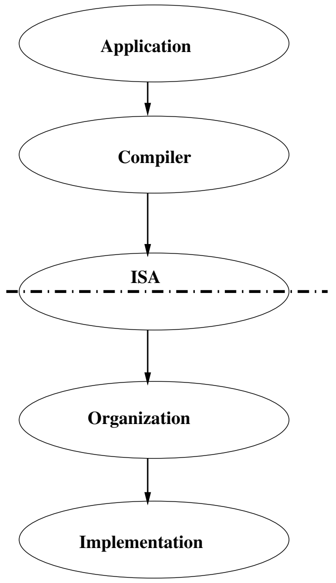
Quantitative principles

- Take advantage of parallelism
- Principle of locality
- Focus on the common case
- **Amdahl's law**
  Enhancement E accelerates a fraction F of a program by a factor S
  $T_{exe}(with\ E) = T_{exe}(without\ E) * [(1 - F) + F/S]$
  $Speedup(E) = \frac{T_{exe}(without\ E)}{T_{exe}(with\ E)} = \frac{1}{(1-F)+F/S}$
- Processor performance equation
  Execution time =
  $seconds/program =$

$$\underbrace{instr./program}_{IC} * \underbrace{cycles/instr.}_{\overline{CPI}} * \underbrace{seconds/cycle}_{T_c}$$

Outline

Instruction Set Architecture - ISA



Instruction Set Architecture (ISA)

ISA (Instruction Set Architecture) is the
interface between software and hardware

A good interface:

- Lasts through many implementations (portability, compatibility)
- Can be used in many different ways (generality)
- Provides sufficient functionality to higher levels
- Permits an efficient implementation at lower levels

RISC - CISC

$$CPUtime = T_c * \underset{RISC}{\overline{CPI}} * \underset{CISC}{IC}$$

RISC (Reduced Instruction Set Computing)

- simple instructions
- MIPS, ARM, ...
- easier to design, build
- less power
- larger code size
- easier for compiler

CISC (Complex Instruction Set Computing)

- complex instructions
- VAX, Intel 80x86 (now RISC-like internally), ...

Addressing modes

Are all these addressing modes needed?

- Register
- Immediate
- Displacement
- Register indirect
- Indexed
- Direct or absolute
- Memory indirect
- Auto-increment
- Auto-decrement
- Scaled

Instruction formats

- A variable instruction format yields compact code but the instruction decoding is more complex and thus slower
  Examples: VAX, Intel 80x86

Operation # operands	Address specifier 1	Address field 1	...	Address specifier x	Address field x

- A fixed instruction format is easy and fast to decode but gives large code size
  Examples: Alpha, ARM, MIPS, PowerPC, SPARC

Operation	Address field 1	Address field 2	Address field 3

How can you aid the compiler?

- Rules of thumb when designing an instruction set:
  - Regularity (operations, data types and addressing modes should be orthogonal)
  - Provide primitives, not high-level constructs or solutions. Complex instructions are often too specialized.
  - Simplify trade-offs among alternatives
  - Provide instructions that bind quantities known at compile time as constants

Summary - ISA

- The instruction set architecture have importance for the performance
- The important aspects of an ISA are:
  - register model
  - addressing modes
  - types of operations
  - data types
  - encoding
- Benchmark measurements can reveal the most common case
- Interaction compiler - ISA important

Outline

Pipelining Lessons

- Pipelining doesn't help latency of a single instruction, it helps throughput of the entire worklo ad
- Pipeline rate is limited by the slowest pipeline stage
- Multiple instructions are executing simultaneously
- Potential speedup = Number of pipe stages
  - Unbalanced lengths of pipe stages reduces speedup
  - Time to fill pipeline and time to drain reduces speedup
  - Hazards reduces speedup

Summary Pipelining - Methods

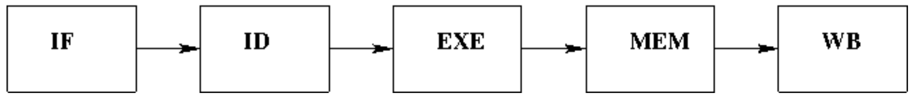
Dependencies are properties of the ***program***
Whether a dependency leads to a hazard or not is a property of the ***pipeline implementation***

Dependency	**Hazard**	**Method**
Data Name	RAW WAR, WAW	Forwarding, Scheduling Register Renaming
Control	Control	Branch Prediction, Speculation, Delayed branch
	Structural	More hardware
Precise exceptions		in-order commit
ILP		Scheduling, Loop unrolling

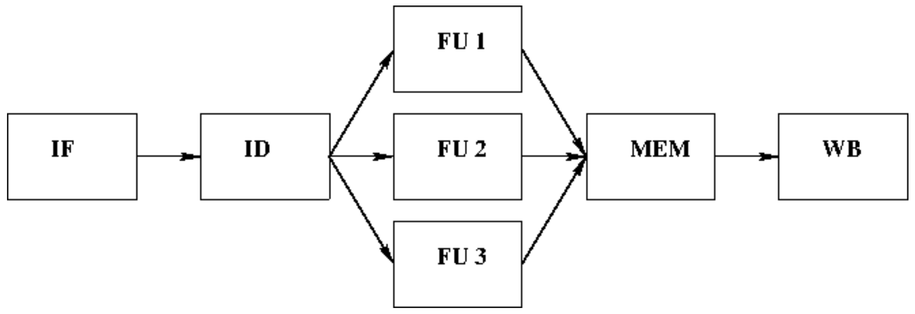
Summary Pipelining - Implementation

Problem	**Simple**	**Scoreboard**	**Tomasulo**	**Tomasulo + Speculation**
	Static Sch	Dynamic Scheduling		
RAW	forwarding stall	wait (Read)	CDB stall	CDB stall
WAR	-	wait (Write)	Reg. rename	Reg. rename
WAW	-	wait (Issue)	Reg. rename	Reg. rename
Exceptions	precise	?	?	precise, ROB
Issue	in-order	in-order	in-order	in-order
Execution	in-order	out-of-order	out-of-order	out-of-order
Completion	in-order	out-of-order	out-of-order	in-order
Structural hazard	-	many FU stall	many FU, CDB, stall	many FU, CDB, stall
Control hazard	Delayed br., stall	Branch prediction	Branch prediction	Br. pred, speculation

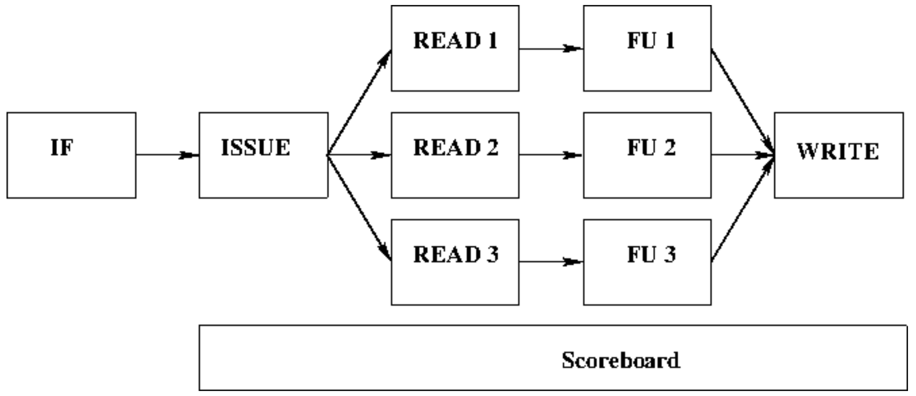
Basic 5 stage pipeline



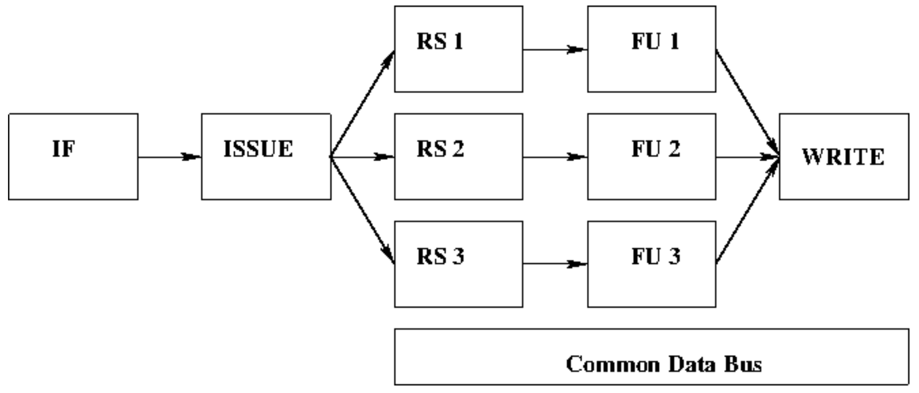
Pipeline with several Functional units



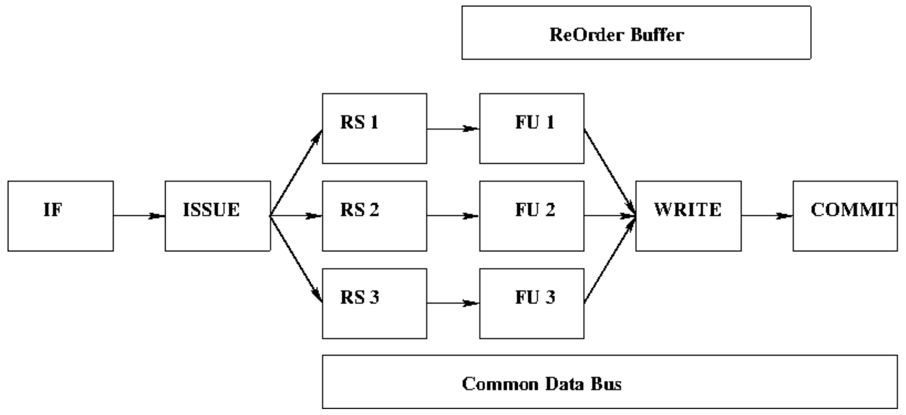
Scoreboard pipeline



Basic Tomasulo pipeline



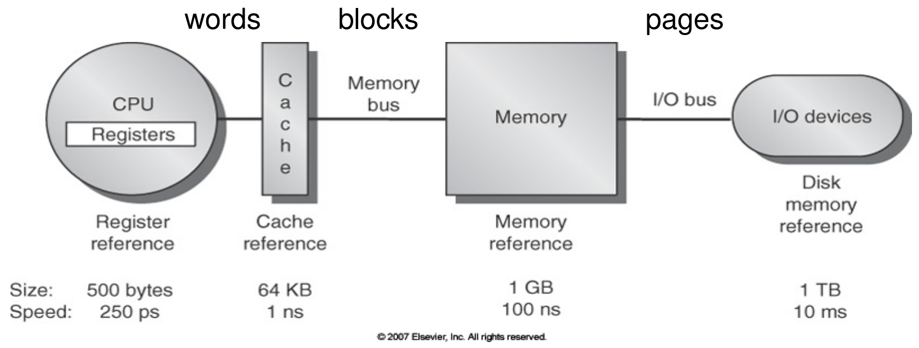
Tomasulo pipeline with speculation



Outline

Memory Hierarchy Functionality

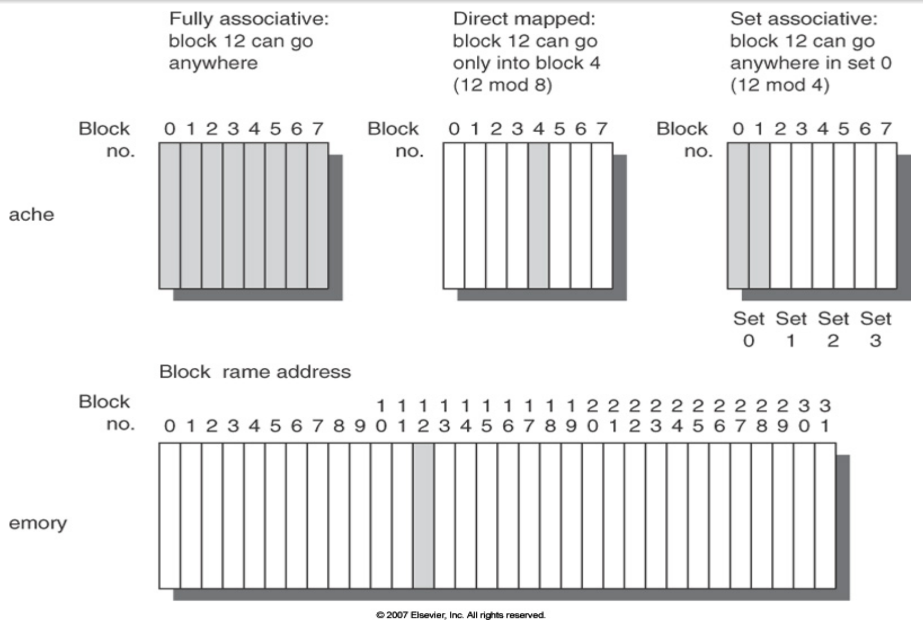
- CPU tries to access memory at address A. If A is in the cache, deliver it directly to the CPU
- If not – transfer a **block of memory words**, containing A, from the memory to the cache. Access A in the cache.
- If A not present in the memory – transfer a **page of memory blocks**, containing A, from disk to the memory, then transfer the block containing A from memory to cache. Access A in the cache.



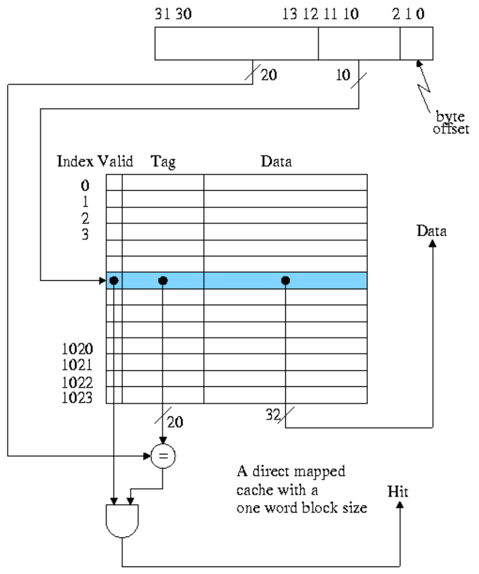
4 questions for the Memory Hierarchy

- Q1: Where can a block be placed in the upper level? ***(Block placement)***
- Q2: How is a block found if it is in the upper level? ***(Block identification)***
- Q3: Which block should be replaced on a miss? ***(Block replacement)***
- Q4: What happens on a write? ***(Write strategy)***

Block placement


Fully associative: block 12 can go anywhere
Direct mapped: block 12 can go only into block 4 (12 mod 8)
Set associative: block 12 can go anywhere in set 0 (12 mod 4)
Block no. 0 1 2 3 4 5 6 7
ache
Set 0 Set 1 Set 2 Set 3
Block rame address
Block no. 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
emory
© 2007 Elsevier, Inc. All rights reserved.

Block identification



Block replacement

- Direct mapped caches don't need a block replacement policy (why?)
- Primary strategies:
  - Random (easiest to implement)
  - LRU – Least Recently Used (best)
  - FIFO – Oldest

Cache micro-ops sequencing – read

- **1**: Address division
- **2**: Set/block selection
- **2a**: Tag read
- **3**: Tag/Valid bit checking
- **4**: Hit: Data out
  Miss: Signal cache miss; initiate replacement

Cache Performance

$$\text{CPU execution Time} = IC * (CPI_{execution} + \frac{\textit{mem accesses}}{\textit{instruction}} * \textit{miss rate} * \textit{miss penalty}) * T_C$$

Three ways to increase performance:

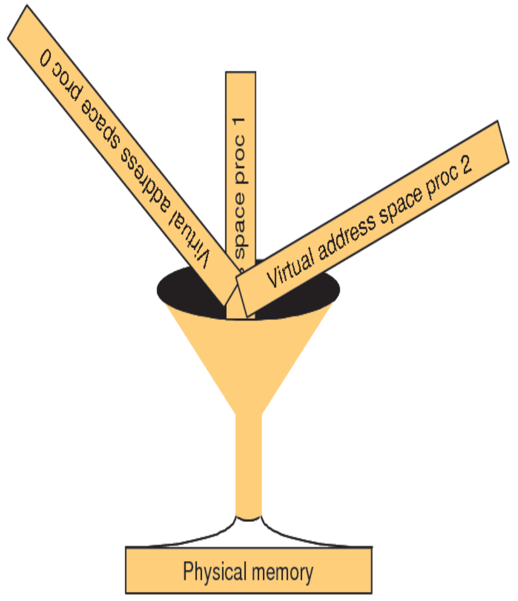
- Reduce miss rate
- Reduce miss penalty
- Reduce hit time (improves T_C)

$$\text{Average memory access time} = \textit{hit time} + \textit{miss rate} * \textit{miss penalty}$$

Cache optimizations

	Hit time	**Band-width**	**Miss penalty**	**Miss rate**	**HW complexity**
Simple	+			-	0
Addr. transl.	+				1
Way-predict	+				1
Trace	+				3
Pipelined	-	+			1
Banked		+			1
Nonblocking		+	+		3
Early start			+		2
Merging write			+		1
Multilevel			+		2
Read priority			+		1
Prefetch			+	+	2-3
Victim			+	+	2
Compiler				+	0
Larger block			-	+	0
Larger cache	-			+	1
Associativity	-			+	1

Virtual memory – why?

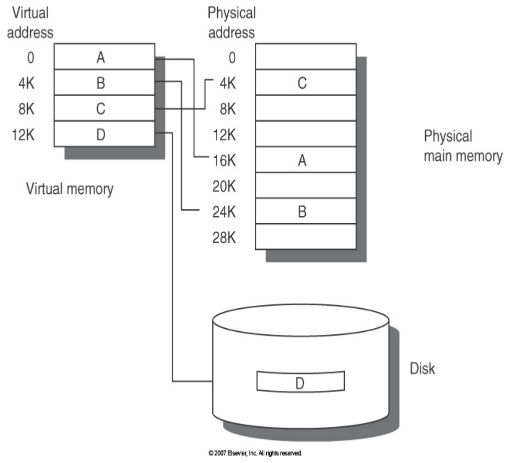


Reasons to use VM:

- Replaces overlays
- Large address space
- Several processes sharing the same physical memory
- Protection of memory
- Relocation

Virtual memory – concepts

Part of the memory hierarchy:

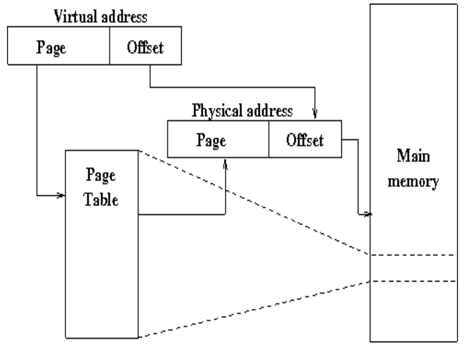


- The virtual address space is divided into ***pages***
- The physical address space is divided into ***page frames***
- A miss is called a ***page fault***
- Pages not in main memory are stored on ***disk***

- The CPU uses **virtual addresses**
- We need an **address translation** (memory mapping) mechanism

VM: Page identification

Use a **page table** stored in main memory:



- Suppose 4 KB pages, 32 bit virtual address, 4 bytes per entry
- Page table takes $\frac{2^{32}}{2^{12}} * 4 = 2^{22} = 4\ Mbyte$
- 64 bit virtual address, 16 KB pages $\rightarrow$ $\frac{2^{64}}{2^{14}} * 4 = 2^{52} = 2^{12}TB$
- Per process

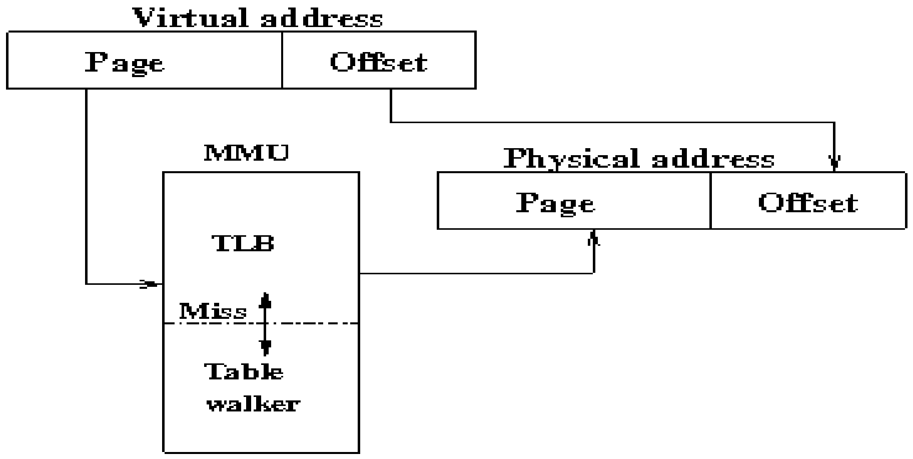
Solutions

- Multi–level page table
- (Inverted page table)

Fast address translation

How do we avoid two (or more) memory references for each original memory reference?

- Cache address translations – ***Translation Lookaside Buffer*** **(TLB)**



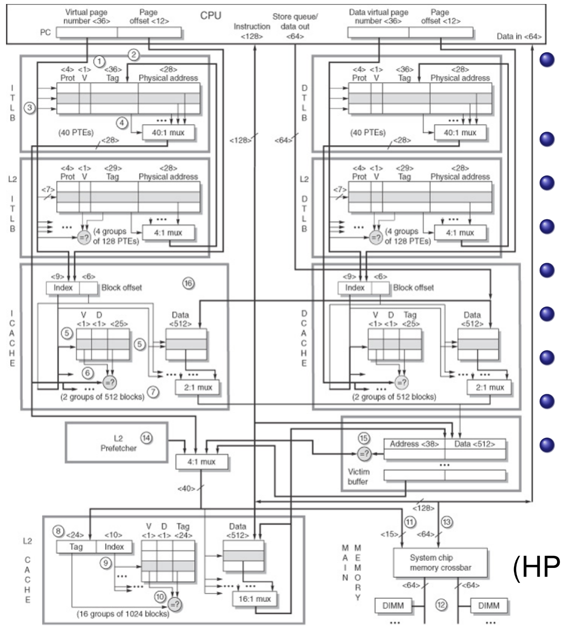
Summary memory hierarchy

Hide CPU - memory performance gap
Memory hierarchy with several levels
Principle of locality

Cache memories:	**Virtual memory:**
Fast, small - Close to CPU	Slow, big - Close to disk
Hardware	Software
TLB	TLB
CPU performance equation	Page-table
Average memory access time	Very high miss penalty $\Longrightarrow$ miss rate must be low
Optimizations	Also facilitates: relocation; memory protection; and multiprogramming

Same 4 design questions - Different answers

The memory hierarchy of AMD Opteron



- Separate Instr & Data TLB and Caches
- 2-level TLBs
- L1 TLBs fully associative
- L2 TLBs 4 way set associative
- Write buffer (and Victim cache)
- Way prediction
- Line prediction - prefetch
- hit under 10 misses
- 1 MB L2 cache, shared, 16 way set associative, write back

(HP fig 5.19) (complex - no details)

Outline

RAID types

	RAID level	Failures tolerated	Overhead 8 data disks	comment
0	striped	0	0	JBOD, common
1	mirrored	1-8	8	high overhead
2	ECC	1	4	not used
3	bit parity	1	1	synchronized drives
4	block parity	1	1	
5	block parity distributed	1	1	common
6	row-diagonal dual parity	2	2	high availability
01	mirrored stripes	1-8	8	
10	striped mirrors	1-8	8	

Summary I/O

I/O:

- I/O performance is important!
- The task of the I/O system designer:
  - meet performance needs
  - cost-effective
  - reliability, availability
- I/O system parts
  - CPU interface
  - Interconnect technology
  - Device performance

Disks:

- Disks have moving parts leading to long service times
- RAID disk arrays provide high bandwidth, high capacity disk storage at a reasonable cost
- SSD is faster and more expensive

Embedded processors

- Important, found everywhere, high volume
- General purpose, application specific, single purpose
- Design of hardware and software together
- Cover several areas
  - microelectronics
  - real time
  - software + hardware
  - SoC