Lecture 8: EITF20 Computer Architecture

Anders Ardö

EIT – Electrical and Information Technology, Lund University

December 2, 2014

Outline

Cache Performance

Execution Time =

$$IC * (CPI_{execution} + \frac{\text{mem accesses}}{\text{instruction}} * \textbf{miss rate} * \textbf{miss penalty}) * \mathbf{T_C}$$

Three(+) ways to increase performance:

- Reduce miss rate
- Reduce miss penalty
- Reduce hit time
- ... increase bandwidth

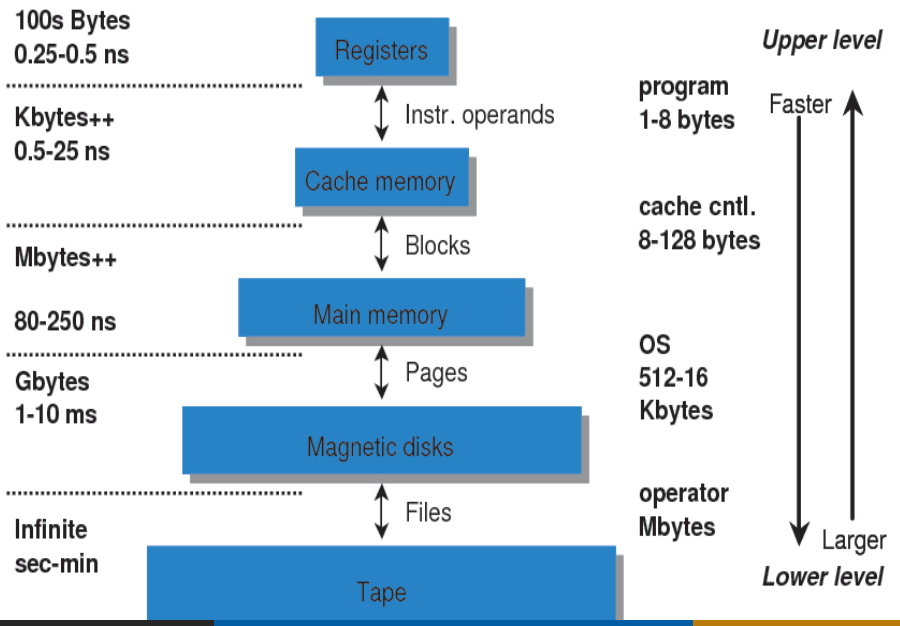
However, remember:

Execution time is the only true measure!

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
Non-blocking		+	+		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

Memory hierarchy

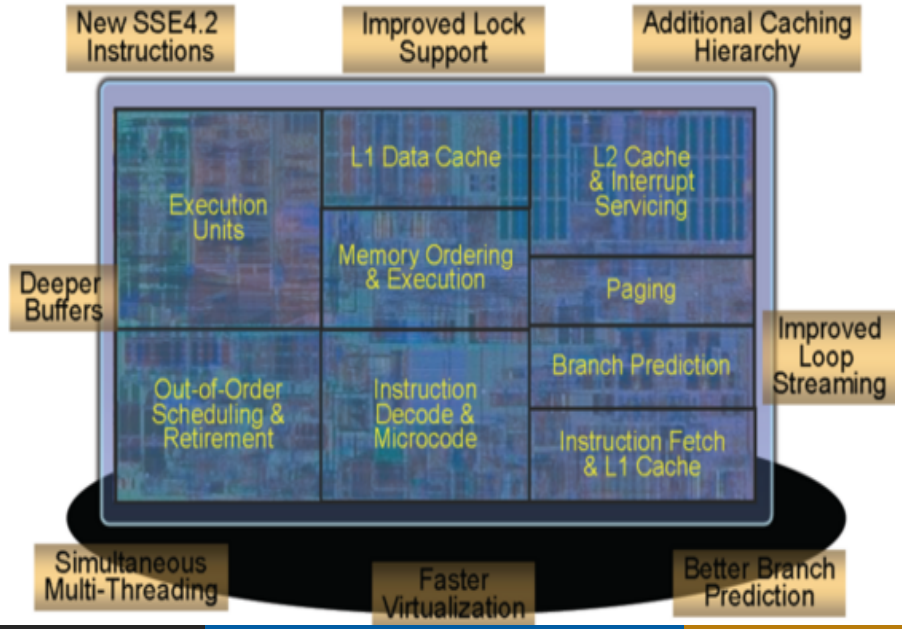


Questions!

QUESTIONS?

COMMENTS?

Chip floor-plan



Lecture 8 agenda

Appendix C.4 and Chapters 5.5-5.6 in "Computer Architecture"

1. Reiteration
2. Virtual memory
3. Case study AMD Opteron
4. Crosscutting issues
5. Summary

Outline

Memory hierarchy tricks

Use two "magic" tricks

- Make a small memory seem bigger (Without making it much slower) — **virtual memory**
- Make a slow memory seem faster (Without making it smaller) — cache memory

Memory tricks (techniques)

Use a hierarchy

	super-fast	Registers		
Memory	FAST	Cache	cache memory	(HW)
	CHEAP	Main	**virtual memory**	(SW)
	BIG	Disk		

Levels of the Memory Hierarchy

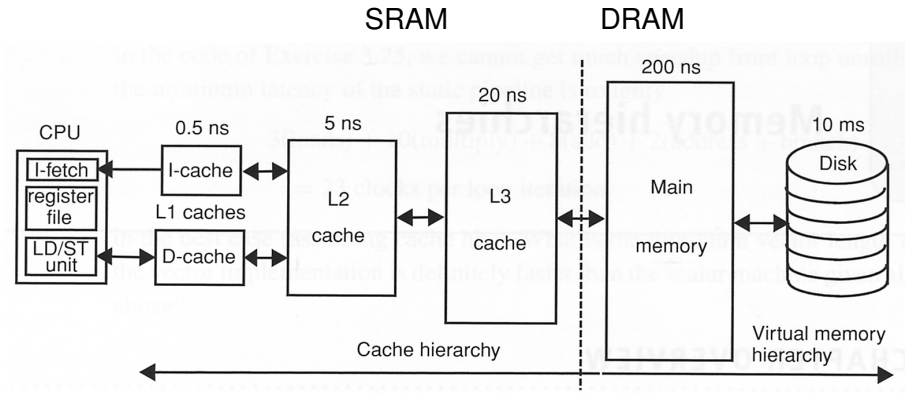


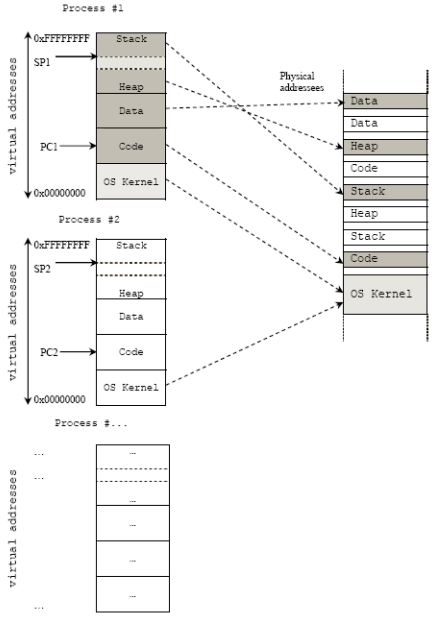
Figure 4.1. Typical memory hierarchy with three levels of caches.

(From Dubois, Annavaram, Stenström: "*Parallel Computer Organization and Design*", ISBN 978-0-521-88675-8)

OS Processes/Virtual memory

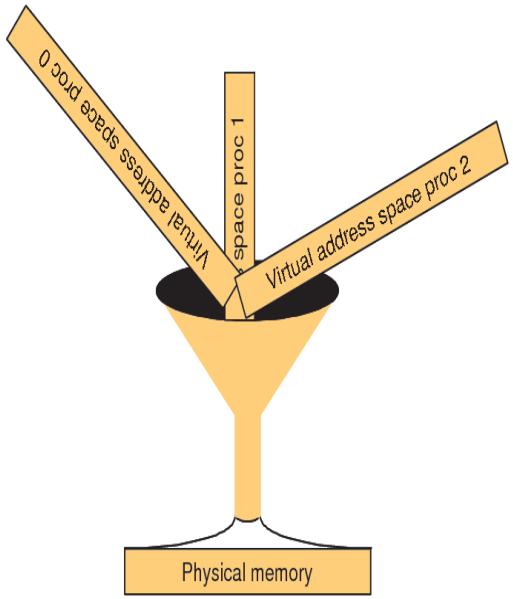
- Run several programs at the same time
- Each having the full address space available
- Sharing physical memory
- Reside and execute anywhere in memory - Relocation
- Program uses virtual memory address
- Virtual address is translated to physical address
- Should be completely transparent to program, minimal performance impact

Process Address Spaces



PC - Program counter.

Virtual memory – why?

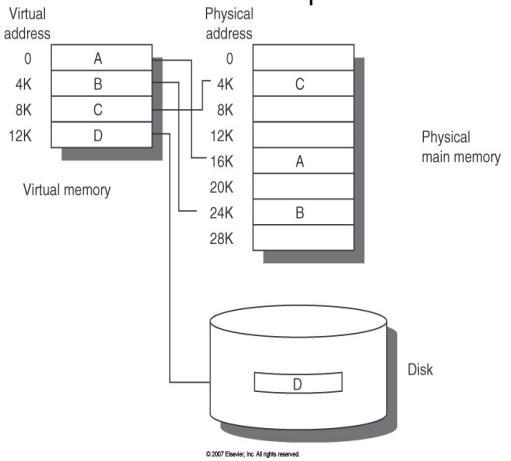


Reasons to use VM:

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

Virtual memory – concepts

is a 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

Virtual memory parameters

	Regs	L1	L2	Main memory	Disk
Access (ns)	0.2	0.5	7	100	10 000 000
Capacity (kB)	1	64	1 000	32 000 000	6 000 000 000
Block size (B)	8	64	128	4 000 - 16 0000	

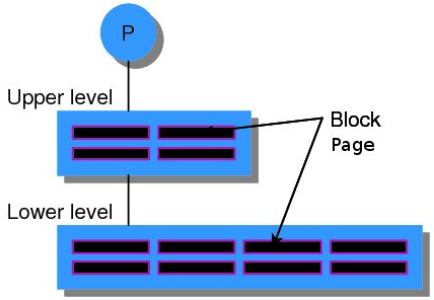
- Replacement in cache handled by HW
- Replacement in VM handled by SW
- Size of VM determined by no of address bits
- The backing store for VM (***paging (swap) partition*** on disk) is shared with the file system

More Virtual memory parameters

Parameter	*First-level cache*	*Virtual memory*
Block (page) size	16-128 bytes	4K-64K bytes
Hit time	1-3 clock cycles	50-150 clock cycles
Miss penalty	8-150 clock cycles	1 M – 10 M clock cycles
(Access time)	(6-130 clock cycles)	(800K-8000K clock cycles)
(Transfer time)	(2-20 clock cycles)	(200K-2000K clock cycles)
Miss rate	0.1%-10%	0.00001%-0.001%
Data memory size	16 Kbyte - 1 Mbyte	32 Mbyte - 1000 Gbyte

4 questions for the Memory Hierarchy

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



VM: Page placement

Where can a page be placed in main memory?

- Cache access: $\sim$ ns
- Memory access: $\sim$ 100 ns
- Disk access: $\sim 10,000,000$ ns

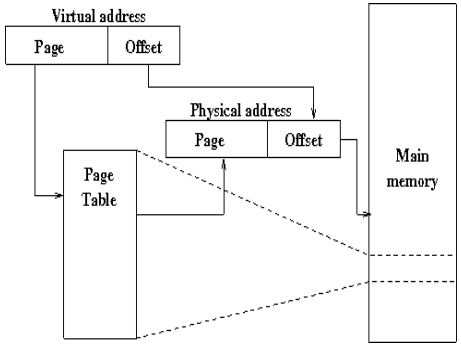
$\Longrightarrow$ **HIGH miss penalty**

The high miss penalty makes it

- necessary to **minimize miss rate**
- possible to use software solutions to implement a **fully associative address mapping**

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)

Virtual memory access

- CPU issues a load for virtual address
- Split into *page, offset*
- Look-up in the page table (main memory) to translate *page*
- Concatenate translated *page* with *offset* $\rightarrow$ physical address
- A read is done from the main memory at physical address
- Data is delivered to the CPU

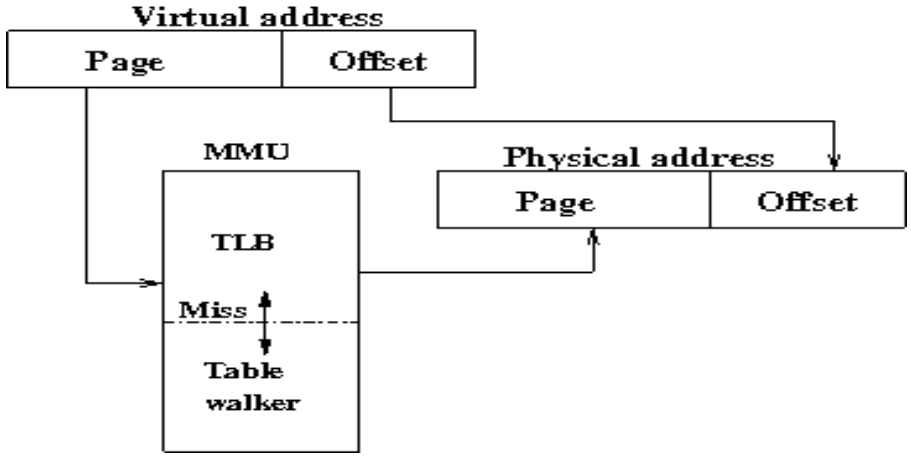
2 memory accesses!

How do we make the page table look-up faster?

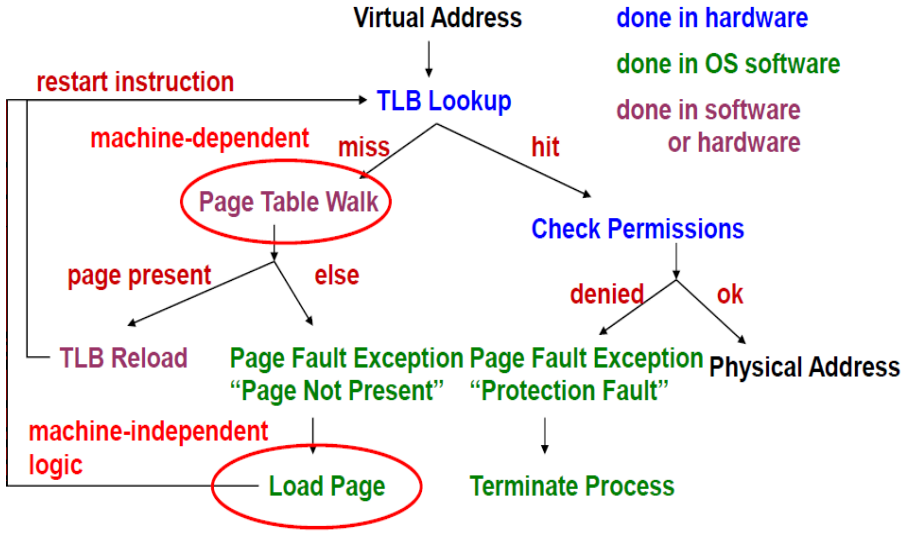
Fast address translation

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

- Cache address translations – ***Translation Look-aside Buffer*** **(TLB)**

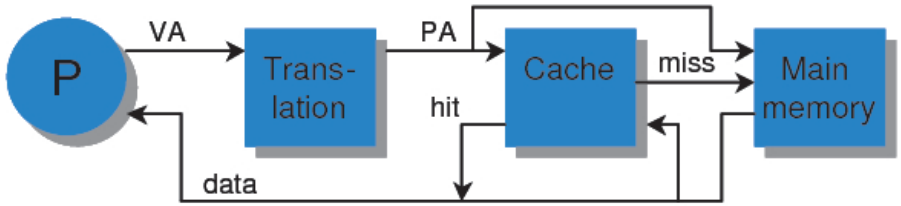


Address translation and TLB

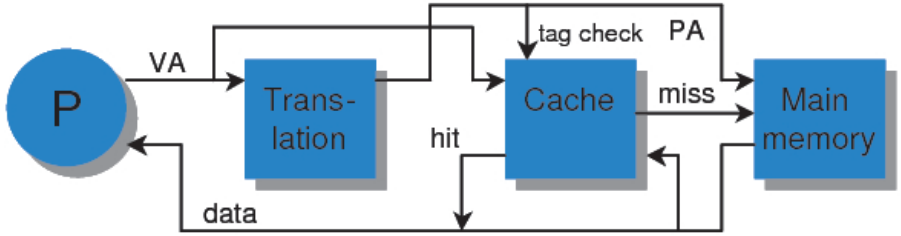


Reduce hit time 2: Address translation

- Processor uses *virtual addresses* (VA) while caches and main memory use *physical addresses* (PA)

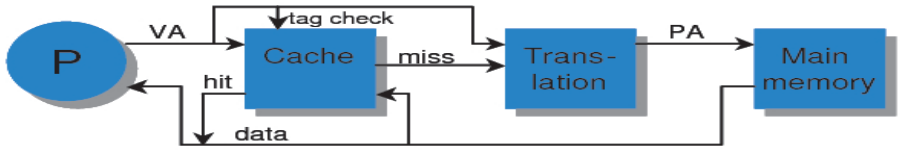


Use the virtual address to index the cache in parallel



Reduce hit time 3: Address translation

- Use virtual addresses to both index cache and tag check



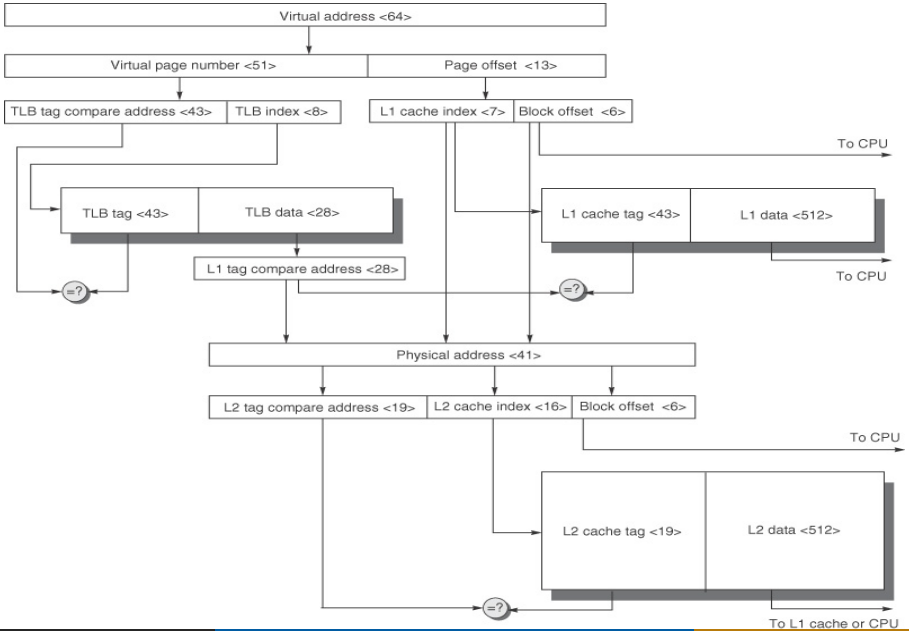
Processes have different virtual address spaces

- Flush cache between context switches or
- Extend tag with process-identifier tag (PID)

Two virtual addresses may map to the same physical address – synonyms or aliases

- HW guarantees that every cache block have a unique physical address
- Some OS uses “page colouring” to ensure that the cache index is unique

Address translation cache and VM



VM: Page replacement

Most important: ***minimize number of page faults***

Handled in SW

Page replacement strategies:

- FIFO – First-In-First-Out
- LRU – Least Recently Used
  - Approximation
  - Each page has a ***reference bit*** that is set on a reference
  - The OS periodically resets the reference bits
  - When a page needs to be replaced, a page with a reference bit that is not set is chosen

VM: Write strategy

Write back or Write through?

- **Write back**! + dirty bit
- Write through is impossible to use:
  - Too long access time to disk
  - The write buffer would need to be *very* large
  - The I/O system would need an extremely high bandwidth

Page size

Larger page size

advantages

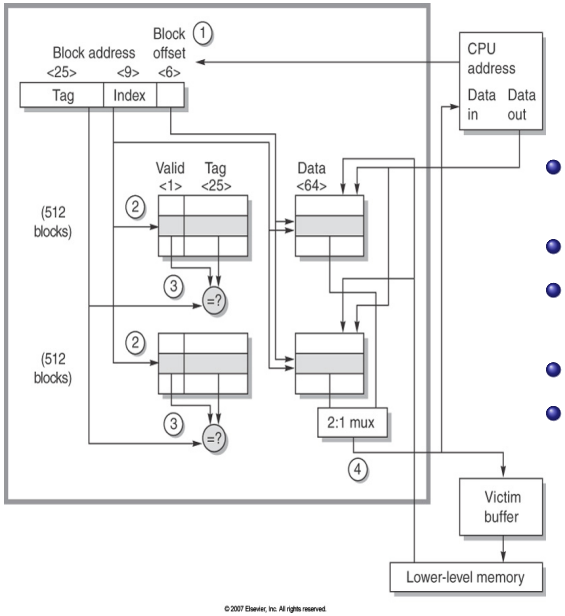
- Size of page table $= k * \frac{2^{addrbits}}{2^{pagebits}} \sim \frac{1}{page\ size}$
- More efficient to transfer large pages
- More memory can be mapped $\rightarrow$ reducing TLB misses

disadvantages

- More wasted storage, internal fragmentation
- Long start-up times

Outline

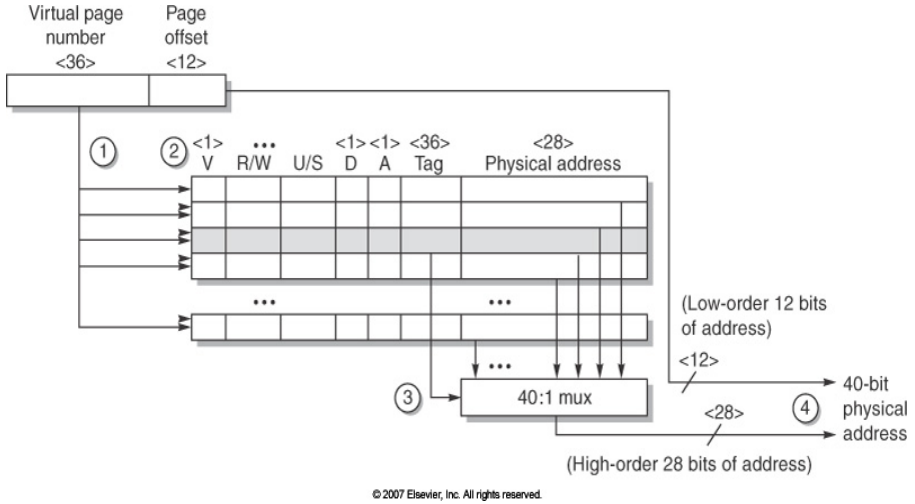
Basic L1 data cache



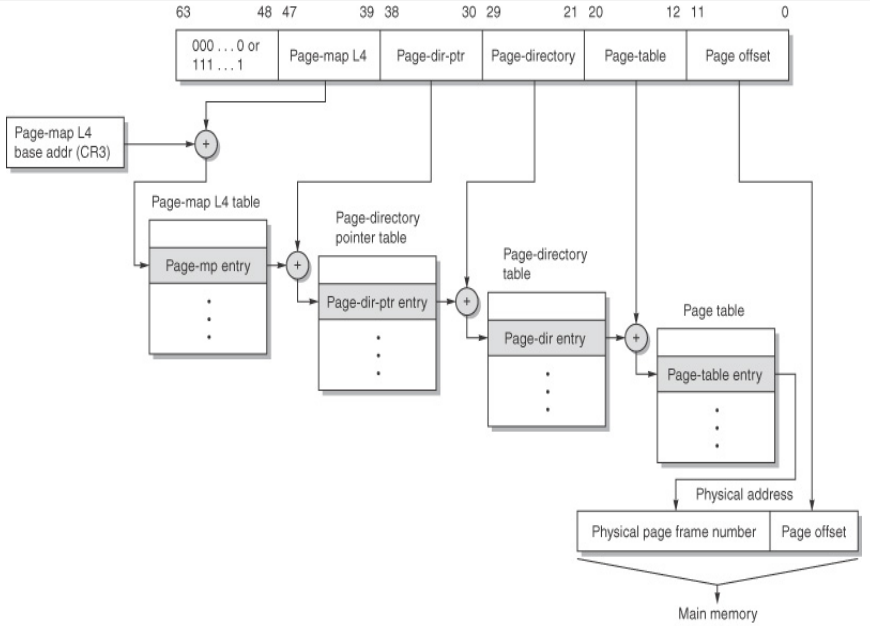
- 64 Kbyte, 64 byte block size $\Longrightarrow$ 1024 blocks
- write-back, write allocate
- 2-way set associative $\Longrightarrow$ 512 sets
- 8 block write buffer (victim)
- LRU - 1 bit

Data TLB

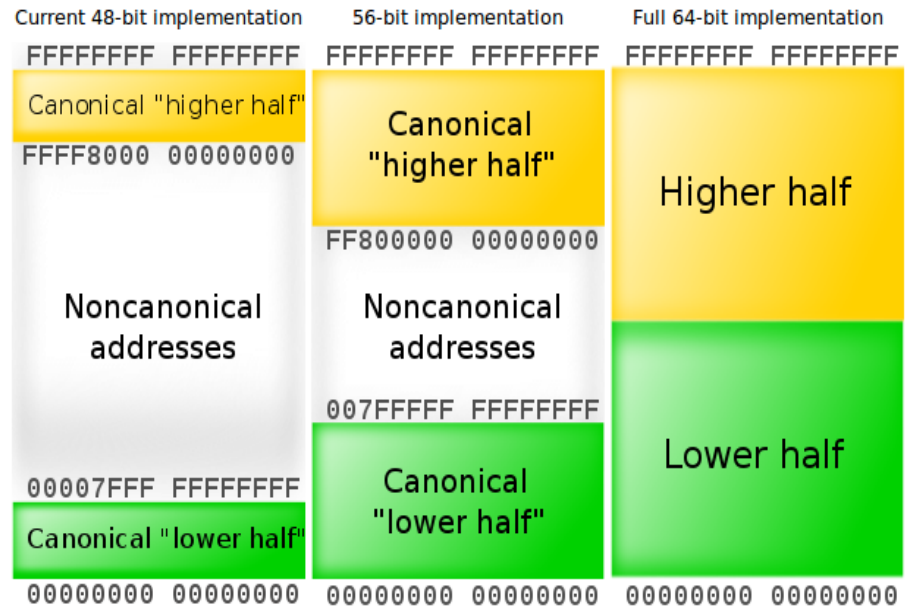
- 40 page table entries
- Fully associative
- Valid bit, kernel & user read/write permissions, protection



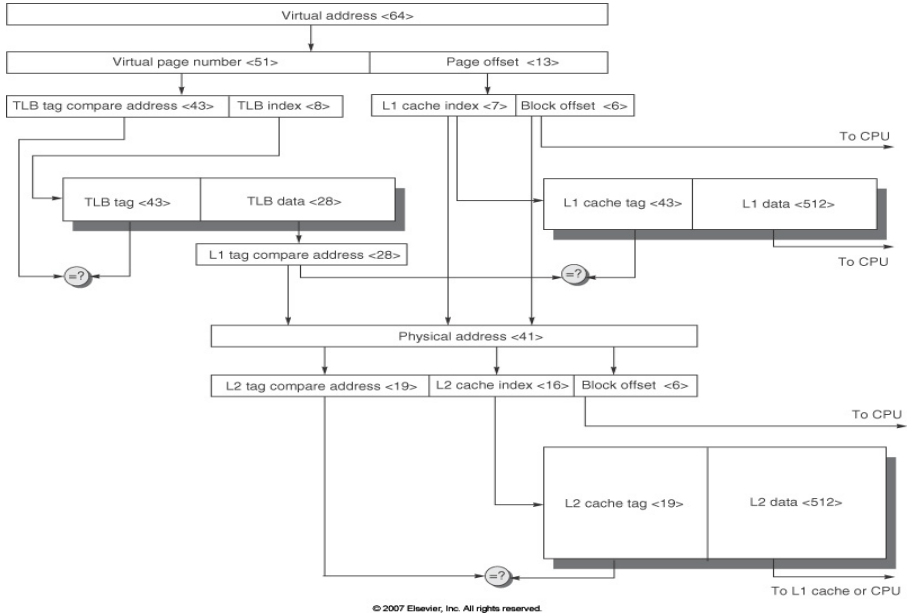
Page table structure



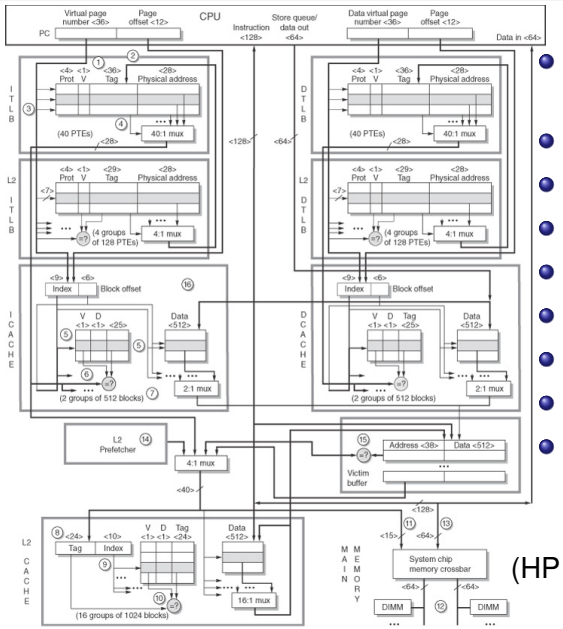
AMD64 Memory space



AMD Opteron cache + TLB



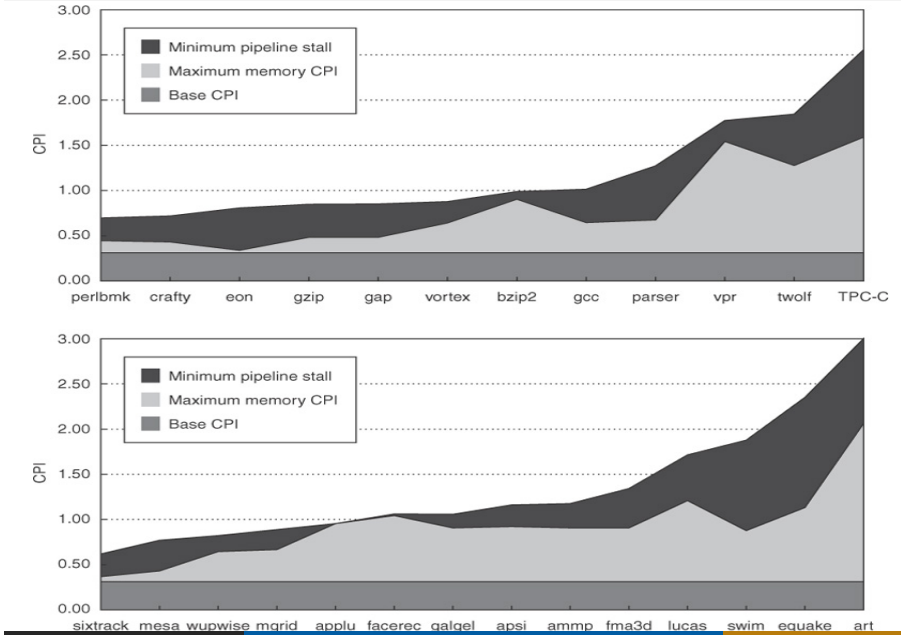
The memory hierarchy of AMD Opteron



- 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)

Performance



Outline

Crosscutting issues

- Protection needs support in the ISA
- Speculative execution $\Longrightarrow$
  - Non-blocking caches
  - Suppress exceptions
- Embedded computers $\Longrightarrow$
  - Real time performance predictability
  - Power limitations

Outline

Summary memory hierarchy

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

Cache memories:

- Fast, small - Close to CPU
- Hardware
- TLB
- CPU performance equation
- Average memory access time
- Optimizations

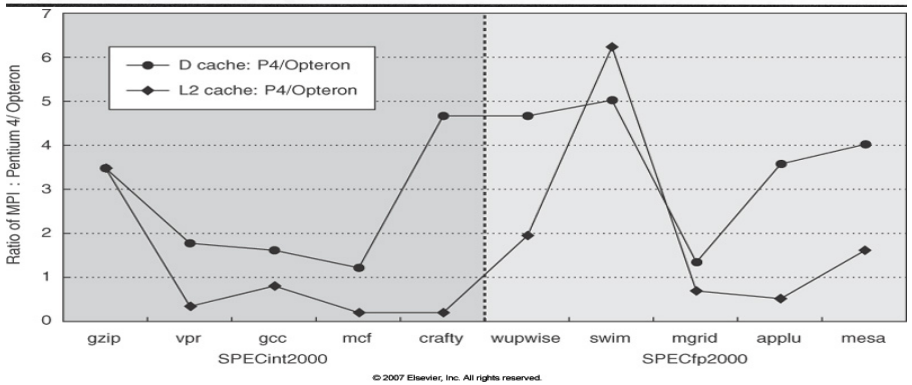
Virtual memory:

- Slow, big - Close to disk
- Software
- TLB
- Page-table
- Very high miss penalty $\Longrightarrow$ miss rate must be low
- Also facilitates: relocation; memory protection; and multiprogramming

Same 4 design questions - Different answers

Program behavior vs cache organization

Processor	Pentium 4 (3.2 GHz)	Opteron (2.8 GHz)
Data cache	8-way associative, 16 KB, 64-byte block	2-way associative, 64 KB, 64-byte block
L2 cache	8-way associative, 2 MB, 128-byte block, inclusive of D cache	16-way associative, 1 MB, 64-byte block, exclusive of D cache
Prefetch	8 streams to L2	1 stream to L2



Example organizations

MPU	AMD Opteron	Intel Pentium 4	IBM Power 5	Sun Niagara
Instruction set architecture	80x86 (64b)	80x86	PowerPC	SPARC v9
Intended application	desktop	desktop	server	server
CMOS process (nm)	90	90	130	90
Die size (mm^2)	199	217	389	379
Instructions issued/clock	3	3 RISC ops	8	1
Processors/chip	2	1	2	8
Clock rate (2006)	2.8 GHz	3.6 GHz	2.0 GHz	1.2 GHz
Instruction cache per processor	64 KB, 2-way set associative	12000 RISC op trace cache (~96 KB)	64 KB, 2-way set associative	16 KB, 1-way set associative
Latency L1 I (clocks)	2	4	1	1
Data cache per processor	64 KB, 2-way set associative	16 KB, 8-way set associative	32 KB, 4-way set associative	8 KB, 1-way set associative
Latency L1 D (clocks)	2	2	2	1
TLB entries (I/D/L2 I/L2 D)	40/40/512/512	128/54	1024/1024	64/64
Minimum page size	4 KB	4 KB	4 KB	8 KB
On-chip L2 cache	2 x 1 MB, 16-way set associative	2 MB, 8-way set associative	1.875 MB, 10-way set associative	3 MB, 2-way set associative
L2 banks	2	1	3	4
Latency L2 (clocks)	7	22	13	22 I, 23 D
Off-chip L3 cache	—	—	36 MB, 12-way set associative (tags on chip)	—
Latency L3 (clocks)	—	—	87	—
Block size (L1I/L1D/L2/L3, bytes)	64	64/64/128/—	128/128/128/256	32/16/64/—
Memory bus width (bits)	128	64	64	128
Memory bus clock	200 MHz	200 MHz	400 MHz	400 MHz
Number of memory buses	1	1	4	4