Concepts Introduced in Appendix B

- general principles of memory hierarchies
- caches
- virtual memory

Memory Hierarchy

- Exploits the principal of spatial and temporal locality.
- Smaller memories are faster, require less energy to access, and are more expensive per byte.
- Larger memories are slower, require more energy to access, and are less expensive per byte.
- Typically, data found in one level is also found in the level below, which is referred to as the inclusion principle.
- goals:
  - Catch most of the references in the fast memory.
  - Have the cost per byte almost as low as that at the slowest memory level.
- Will see that a memory hierarchy is used to implement protection schemes as well.

Memory Hierarchy Levels

Level	1	2	3	4
Name	Registers	Cache	Main memory	Disk storage
Typical size	<4 KiB	32 KiB to 8 MiB	<1 TB	>1 TB
Implementation technology	Custom memory with multiple ports, CMOS	On-chip CMOS SRAM	CMOS DRAM	Magnetic disk or FLASH
Access time (ns)	0.1–0.2	0.5–10	30–150	5,000,000
Bandwidth (MiB/sec)	1,000,000–10,000,000	20,000–50,000	10,000–30,000	100–1000
Managed by	Compiler	Hardware	Operating system	Operating system
Backed by	Cache	Main memory	Disk or FLASH	Other disks and DVD

Memory Hierarchy Terms

- Hit - item found in that level of the hierarchy
- Miss - item not found in that level of the hierarchy
- Hit Time - time required to access the desired item
- Miss Penalty - the additional time required to service the miss
- Miss Rate - fraction of accesses that are not in that level
- Block - the amount of information that is retrieved from the next lower level on a miss

Equations

$$miss\_rate = \frac{number\_of\_misses}{total\_references}$$

$$hit\_rate = \frac{number\_of\_hits}{total\_references}$$

$$avg\_mem\_access\_time = hit\_time + miss\_rate * miss\_penalty$$

$$total\_cycles = hit\_time * total\_references + miss\_penalty * total\_misses$$

Memory Hierarchy Evaluation Methods

- software approach
  - Generate traces of memory addresses by instrumentation, simulation, or traps.
  - Use a memory hierarchy simulator offline or on-the-fly.
- hardware approach
  - Use results from hardware performance counters.
  - Often cannot exactly reproduce results.

Memory Hierarchy Questions

- Where can a block be placed in the current level? (block placement)
- How is a block found if it is in the current level? (block identification)
- Which block should be replaced on a miss? (block replacement)
- What happens on a write? (write strategy)

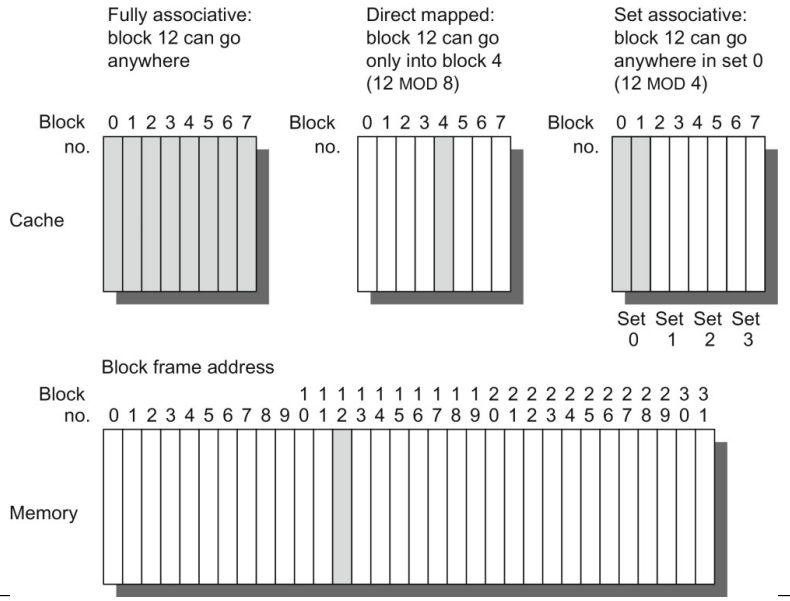
Cache Topics

- organization
- replacement policy
- write policy
- improving cache performance

Cache Organizations

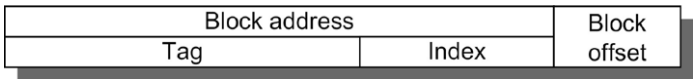
- direct mapped
  - A memory block can be placed in only one cache line.
  - *set = block_address MOD number_of_blocks_in_cache*
- fully associative
  - A memory block can be placed in any cache line.
- set associative
  - A memory block can be placed in any cache line within a single set of lines.
  - *set = block_address MOD number_of_sets_in_cache*

Block Placement



Physical Address Fields

- The *offset* indicates the starting byte within the block.
- The *index* is used to determine which cache set to access.
- The *tag* is used to verify that the correct block is resident in the set.



Block Replacement

- If more than one block in a cache set, then a block must be selected to be replaced on a cache miss.
  - random
    - easy to implement in hardware
    - may not be able to reproduce behavior
  - LRU
    - least-recently assessed block is chosen
    - reduces miss rates
    - can be expensive to implement for high levels of associativity
  - FIFO
    - block loaded first is chosen
    - approximates LRU, but is less complex to implement

Data Cache Misses per 1000 References

- For smaller caches LRU is the most effective.
- For larger caches, both the associativity level and replacement policy become less critical.

	Associativity								
	Two-way			Four-way			Eight-way		
Size	LRU	Random	FIFO	LRU	Random	FIFO	LRU	Random	FIFO
16 KiB	114.1	117.3	115.5	111.7	115.1	113.3	109.0	111.8	110.4
64 KiB	103.4	104.3	103.9	102.4	102.3	103.1	99.7	100.5	100.3
256 KiB	92.2	92.1	92.5	92.1	92.1	92.5	92.1	92.1	92.5

Write Policy

- write through
  - The data is written to both the current cache level and to the next level of the memory hierarchy.
  - Simpler to implement.
  - Can use write buffers to reduce stalls.
  - Cache and next memory hierarchy level are consistent.
- write back
  - The data is written to only the current cache level.
  - A modified cache block (dirty bit set) is written to the next memory hierarchy level when it is replaced.
  - Reduces traffic at the next memory hierarchy level.

Write Miss Policy

- write allocate
  - Block is loaded into the cache on a write miss.
  - Typically used with write back.
- no-write allocate
  - Block is not loaded into the cache on a write miss, but is updated in the next memory hierarchy level.
  - Typically used with write through.

Direct Mapped Example

- Assume a 16 byte line size, 128 cache sets, and an associativity of one (direct mapped).
- How is the physical address partitioned?

- Fill in the information for the following memory references.

Address	Tag	Index	Offset	Result
0x12c				
0x130				
0x4c0				
0x1134				
0x1138				
0x24c8				
0x128				
0x130				
0x4c4				
0x8c8				

Set Associative Example

- Assume a 2-way set associative cache with a 16 byte line size, 64 cache sets, and an LRU replacement policy.
- How is the physical address partitioned?

tag	index	offset

- Fill in the information for the following memory references.

Address	Tag	Index	Offset	Result
0x12c	0	18	12	
0x130	0	19	0	
0x4c0	1	12	0	
0x1134	4	19	4	
0x1138	4	19	8	
0x24c8	9	12	8	
0x128	0	18	0	
0x130	0	19	0	
0x4c4	1	12	4	
0x8c8	2	12	8	

Write-Through, No-Write Allocate

- Fill in what happens for each type of access for a write-through, no-write allocate cache.

Access Type	Update Cache	Access Next Mem Hier Level	Update Next Mem Hier Level
read hit			
read miss			
write hit			
write miss			

Write-Through, No-Write Allocate Example

- Assume a 2-way set associative cache with a 16 byte line size, 64 cache sets, an LRU replacement policy, and a write-through, no-write allocate policy.
- Fill in the information for the following memory references.

R/W	Address	Tag	Index	Offset	Result	Access Next Mem Hier Level	Update Cache
W	0x12c	0	18	12			
R	0x130	0	19	0			
R	0x1134	4	19	4			
W	0x1138	4	19	8			
W	0x2130	8	19	0			
R	0x2134	8	19	4			
R	0x130	0	19	0			

Write-Back, Write Allocate

- Fill in what happens for each type of access for a write-back, write allocate cache.

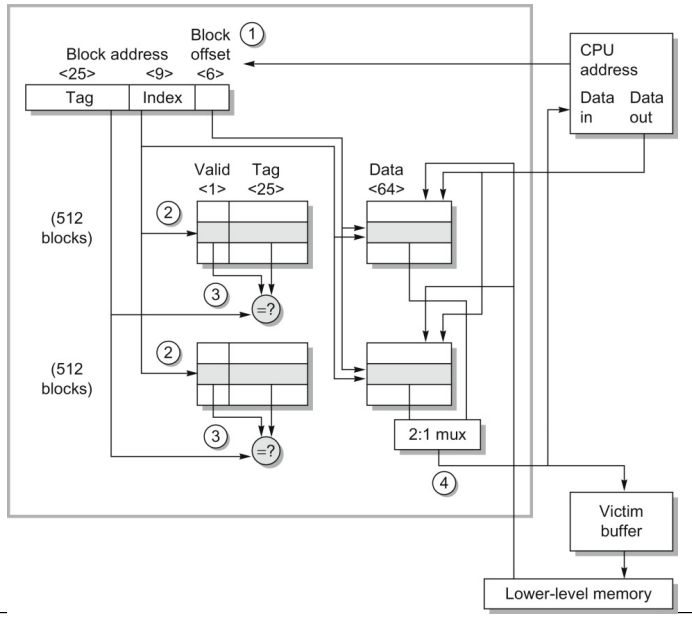
Access Type	Update Cache	Access Next Mem Hier Level	Update Next Mem Hier Level
read hit			
read miss			
write hit			
write miss			

Write-Back, Write Allocate Example

- Assume a 2-way set associative cache with a 16 byte line size, 64 cache sets, an LRU replacement policy, and a write-back, write allocate policy.
- Fill in the information for the following memory references.

R/W	Address	Tag	Index	Offset	Result	Access Next Mem Hier Level	Update Cache
W	0x12c	0	18	12			
R	0x130	0	19	0			
R	0x1134	4	19	4			
W	0x1138	4	19	8			
W	0x2130	8	19	0			
R	0x2134	8	19	4			
R	0x130	0	19	0			

Opteron Processor Data Cache Organization



Instruction, Data, and Unified Cache Misses per 1000 Insts

- Instruction caches have better locality than data.
- A unified cache can provide lower overall miss ratios.
- Typically L1 caches are separate and L2 and L3 caches are unified.

Size (KiB)	Instruction cache	Data cache	Unified cache
8	8.16	44.0	63.0
16	3.82	40.9	51.0
32	1.36	38.4	43.3
64	0.61	36.9	39.4
128	0.30	35.3	36.2
256	0.02	32.6	32.9

Improving Cache Performance

$$avg\_mem\_access\_time = hit\_time + miss\_rate * miss\_penalty$$

- reducing cache miss rate
- reducing cache miss penalty
- reducing cache hit time

Reducing Cache Miss Rate

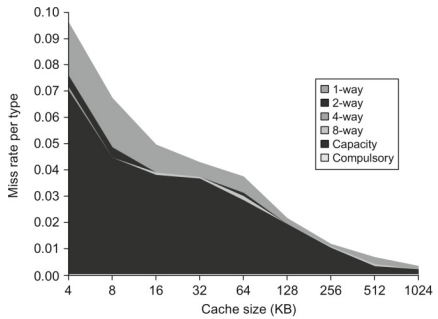
- larger block size
- larger cache size
- higher associativity

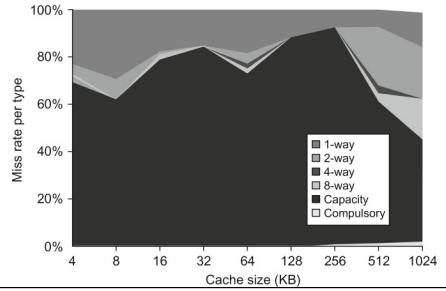
Cache Miss Categories

- compulsory - The first access to a block has to be loaded into cache.
- capacity - These blocks are replaced and later retrieved since the cache cannot contain all the blocks needed during execution.
- conflict - Occurs when too many blocks map to the same cache set.

Miss Rate Distributions for the Three C's

- actual miss rate
- percentage in each category



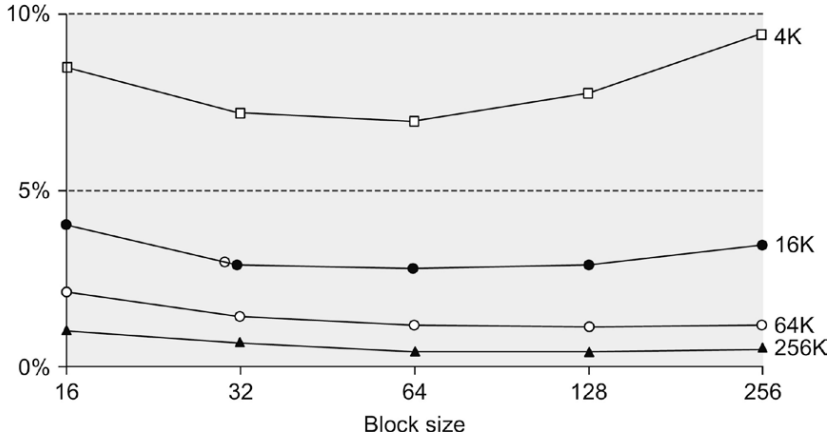


Larger Block Size

- advantages
  - Exploits spatial locality.
  - Reduces compulsory misses.
- disadvantages
  - Lengthens miss penalty.
  - May waste bandwidth bringing in bytes that are unused.

Miss Rate vs Block Size

- Increasing the block size too much can increase conflict misses.
- It is possible that a significant fraction of the words in a large block may not be referenced before the block is evicted.



Average Memory Access Time (AMAT) vs Block Size

- Results assume main memory access requires 80 cycles and then delivers 16 bytes every 2 cycles.
- The best AMATs are shown in **bold**.

		Cache size			
Block size	**Miss penalty**	**4K**	**16K**	**64K**	**256K**
16	82	8.027	4.231	2.673	1.894
32	84	**7.082**	3.411	2.134	1.588
64	88	7.160	**3.323**	**1.933**	**1.449**
128	96	8.469	3.659	1.979	1.470
256	112	11.651	4.685	2.288	1.549

Larger Cache Size

- advantages
  - Reduces capacity and conflict misses.
- disadvantages
  - Requires more space.
  - May increase hit time.
  - Requires more power for each access.

Higher Associativity

- advantages
  - Reduces miss rate.
- disadvantages
  - May increase hit time.
  - Requires more space and power.
    - logic for additional comparators
    - more tag bits
    - LRU bits
    - logic for a multiplexor

AMAT vs Associativity Level

- Assumes hit time for direct-mapped is 1 cycle, 2-way is 1.36 cycles, 4-way is 1.44 cycles, and 8-way is 1.52 cycles.
- The **bold** numbers means the AMAT got worse than the number to the left of it.

	Associativity			
Cache size (KiB)	1-way	2-way	4-way	8-way
4	3.44	3.25	3.22	**3.28**
8	2.69	2.58	2.55	**2.62**
16	2.23	**2.40**	**2.46**	**2.53**
32	2.06	**2.30**	**2.37**	**2.45**
64	1.92	**2.14**	**2.18**	**2.25**
128	1.52	**1.84**	**1.92**	**2.00**
256	1.32	**1.66**	**1.74**	**1.82**
512	1.20	**1.55**	**1.59**	**1.66**

Reducing Cache Miss Penalty

- multi-level caches
- Give priority to read misses before write misses.

Multi-Level Caches

- Became popular as the miss penalty for primary caches continued to increase.
- Most general purpose machines have 3 or more levels of cache and now are all on the same chip as the processor(s).
- L2 (L3) caches are typically much bigger than L1 (L2) caches with larger block sizes and higher associativity levels.
- L2 and L3 caches tend to be unified.

$$avg\_access\_time = hit\_time\_L1 + miss\_rate\_L1 * miss\_penalty\_L1$$
$$miss\_penalty\_L1 = hit\_time\_L2 + miss\_rate\_L2 * miss\_penalty\_L2$$
$$local\_miss\_rate = \frac{misses\_in\_cache}{accesses\_to\_cache}$$
$$global\_miss\_rate = \frac{misses\_in\_cache}{accesses\_to\_L1\_cache}$$

Victim Caches

- A victim cache is a small fully associative cache that contains recently replaced blocks from a primary cache.
- A block evicted from the primary cache is placed in the victim cache.
- The victim cache is checked in parallel with the primary cache.
- A block that hits in a victim cache can be available in the same cycle as a block that hits in the primary cache.
- A hit in the victim cache causes the block where the data item was found to be swapped with the block in the primary cache that would be evicted.
- Provides performance benefits for machines using primary caches with lower levels of associativity.

Giving Priority to Read Misses over Write Misses

- Priority is given to read misses over write misses by using a write buffer.
- The CPU can check the addresses in the write buffer on a read miss in case the desired item is in the buffer.
- advantages
  - The CPU is much more likely to use the value read from memory sooner than a value written to memory.
  - The CPU can initiate the read miss to the next memory hierarchy level without waiting for the write buffer to drain.
  - The read access to the next memory hierarchy level can be avoided if it can be obtained from the write buffer.
  - Write buffers can be used to make write-back caches more effective.
    - First, copy the dirty block to a write buffer.
    - Second, load the block from the next memory hierarchy level to cache.
    - Third, write the dirty block to the next memory hierarchy level.
- Disadvantage is the additional complexity.

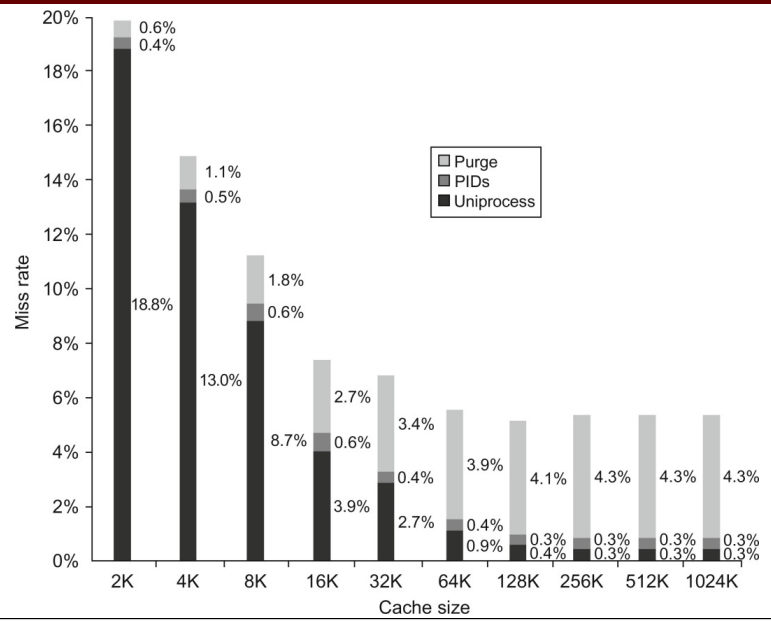
Reducing Cache Hit Time

- virtual caches
- virtually indexed, physically tagged caches

Virtual Caches

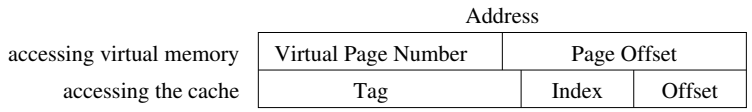
- Access the cache with a virtual instead of a physical address.
- Benefit is that address translation is not on the critical path.
- problems
  - Have to purge the cache on a context switch unless include the PID as part of the tag.
  - Can have synonyms (aliases), which are multiple copies of the same physical data accessed by different processes with different virtual addresses.
  - Page protection information from the DTLB would have to be added to each virtual cache line.
  - I/O and L2/L3 accesses are typically performed using physical addresses and would require mapping to virtual addresses to deal with the virtually addressed cache for I/O and L2/L3 block replacements.

Miss rates for Virtually Addressed Caches



Virtually Indexed, Physically Tagged (VIPT) Caches

- Index into the cache with the page offset.
- Do the tag comparison after the virtual to physical address translation.
- Advantage is that the access to the data in the cache can start sooner.
- Limitation is that one way of a VIPT cache can be no larger than the page size.



Summary Impact of Cache Optimizations

Technique	Hit time	Miss penalty	Miss rate	Hardware complexity	Comment
Larger block size		–	+	0	Trivial; Pentium 4L2 uses 128 bytes
Larger cache size	–		+	1	Widely used, especially for L2 caches
Higher associativity	–		+	1	Widely used
Multilevel caches		+		2	Costly hardware; harder if L1 block size $\neq$ L2 block size; widely used
Read priority over writes		+		1	Widely used
Avoiding address translation during cache indexing	+			1	Widely used

- - means the attribute gets worse
- + means the attribute gets better
- blank means the attribute has little or no effect
- 0 represents the least hardware complexity
- 2 represents the most hardware complexity

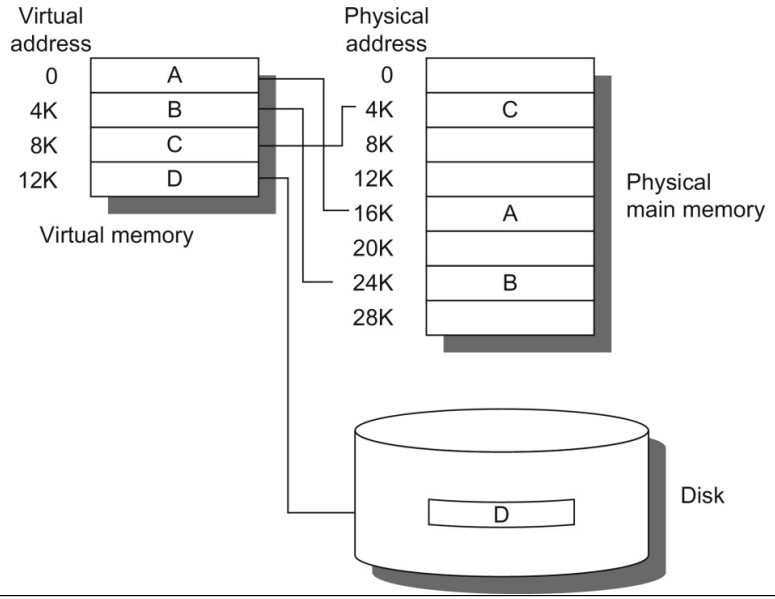
Overlays

- Overlays were used to allow a program to exceed the size of physical memory and still be able to execute.
- A programmer divides their programs into pieces.
- The programmer determines which pieces are never needed to be used at the same time.
- A portion of the program loads from disk or stores to disk these pieces during execution.
- The programmer ensures that the maximum number of program pieces used at the same time fits into physical memory.

Virtual Memory

- Each process has its own separate virtual address space.
- Divides physical memory into blocks and allocates these blocks to different processes.
- Provides a mapping between blocks in physical memory and blocks on disk.
- Allows a process to execute with only portions of the process being in main memory.
- Also reduces program startup time.
- Provides protection to prevent processes from inappropriately accessing these blocks of data or instructions.

Virtual to Physical Page Mappings

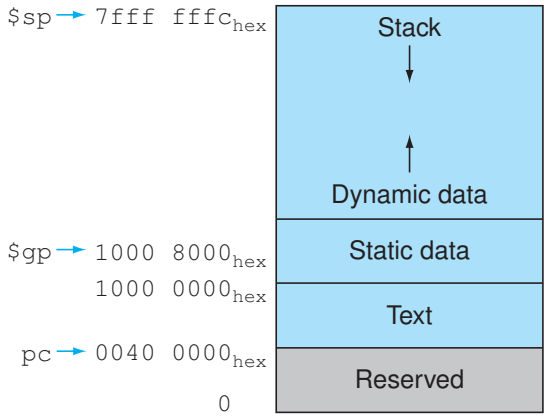


Virtual Memory Terms

- *Page* is the name used for a block.
- *Page fault* is the name when a referenced block is not resident in physical memory.
- *Virtual address* is the address produced by a CPU.
- *Physical address* is the address used to access main memory and typically cache as well.
- *Page table* is the data structure containing the mappings between virtual and physical addresses.
- *Translation Lookaside Buffer* is a cache that contains a subset of the page table entries.

Virtual Memory Process Organization

- Below is a common virtual memory process organization.
- Virtual memory gives the illustion that the memory of a process is contiguous.

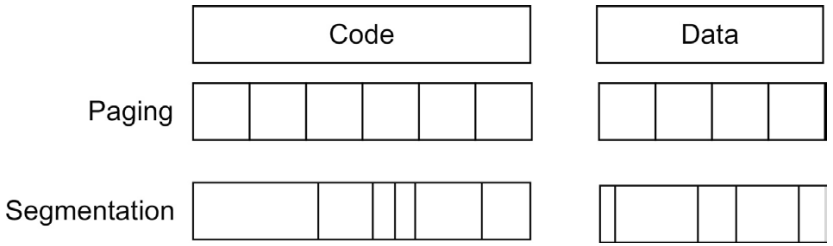


Typical Ranges of Cache and Virtual Memory Parameters

Parameter	First-level cache	Virtual memory
Block (page) size	16–128 bytes	4096–65,536 bytes
Hit time	1–3 clock cycles	100–200 clock cycles
Miss penalty	8–200 clock cycles	1,000,000–10,000,000 clock cycles
(access time)	(6–160 clock cycles)	(800,000–8,000,000 clock cycles)
(transfer time)	(2–40 clock cycles)	(200,000–2,000,000 clock cycles)
Miss rate	0.1%–10%	0.00001%–0.001%
Address mapping	25–45-bit physical address to 14–20-bit cache address	32–64-bit virtual address to 25–45-bit physical address

Example Paging and Segmentation

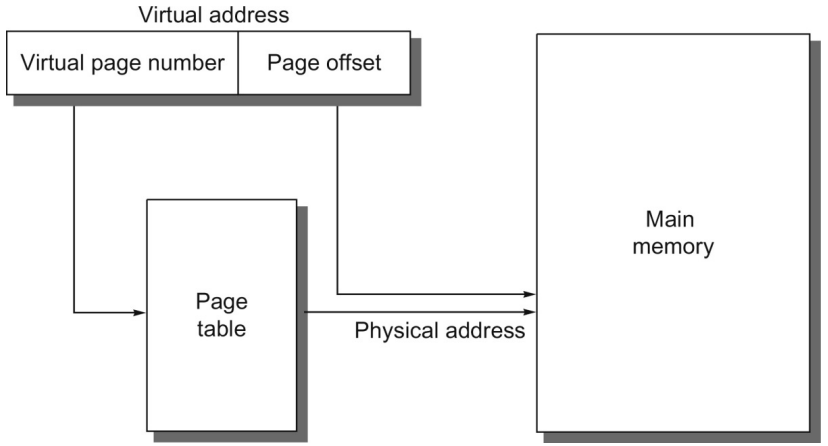
- Pages are fixed in size and segments can vary in size.
- Most systems today use paging.



Paging vs Segmentation

	Page	Segment
Words per address	One	Two (segment and offset)
Programmer visible?	Invisible to application programmer	May be visible to application programmer
Replacing a block	Trivial (all blocks are the same size)	Difficult (must find contiguous, variable-size, unused portion of main memory)
Memory use inefficiency	Internal fragmentation (unused portion of page)	External fragmentation (unused pieces of main memory)
Efficient disk traffic	Yes (adjust page size to balance access time and transfer time)	Not always (small segments may transfer just a few bytes)

Virtual to Physical Address Mapping via a Page Table



Page Tables

- Number of entries in page table is equal to the number of virtual pages.
- Number of virtual pages is the size of the virtual address space divided by the page size.
- Each process has its own page table.
- A page table entry will typically contain a physical page number, resident bit, dirty bit, use bit, protection field (e.g. read only), disk address.

Virtual Memory Questions

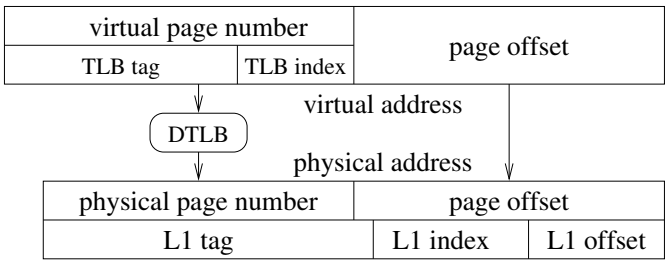
- *Where can a block be placed in main memory?* It can be placed anywhere (fully associative) to reduce the miss rate.
- *How is a block found if it is in main memory?* The page table is indexed by the virtual page number and the page table entry contains the physical page number for that virtual page.
- *Which block should be replaced on a virtual memory miss?* Use bits are used to approximate LRU.
- *What happens on a write?* A write-back, write-allocate policy is always used.

Translation Lookaside Buffers

- Most machines use special caches called TLBs that contain a portion of the entries in a page table.
- Each entry in the TLB contains a valid bit, a tag (portion of the virtual page number), and most of the information in a page table entry.
- TLBs are typically invalidated when a context switch is performed.
- TLBs are typically quite small to provide a very fast translation.
- There are typically separate TLBs for instructions and data to support simultaneous access due to pipelining.
- Many processors have multiple levels of TLBs, where the L1 TLBs are separate and the higher levels are unified.

Virtual to Physical Address Translation

- The virtual page number consists of a tag and index for the TLB.
- The page offset remains unchanged during the virtual to physical address translation.

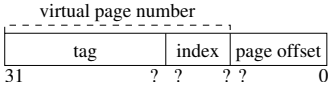


Virtual Memory Supports Multiprogramming

- *Multiprogramming* means that several processes can concurrently share a computer.
- A *process* is the code, data, and any state information used in the execution of a program.
- A *context switch* means transfering control of the machine from one process to another.
- Proper protection must be provided to prevent a process from inappropriately affecting another process or itself.
- Virtual memory systems keep bits in the page table and TLB entries to indicate the type and level of access that the process has to each of the pages.

TLB and Page Table Example

- Page size is 512 bytes. TLB is direct-mapped and has 64 sets.



TLB			
Index	Page #	Tag	V
0	?	?	?
...	...	...	...
7	?	?	0
8	2	0	1
...	...	...	...
63	?	?	?

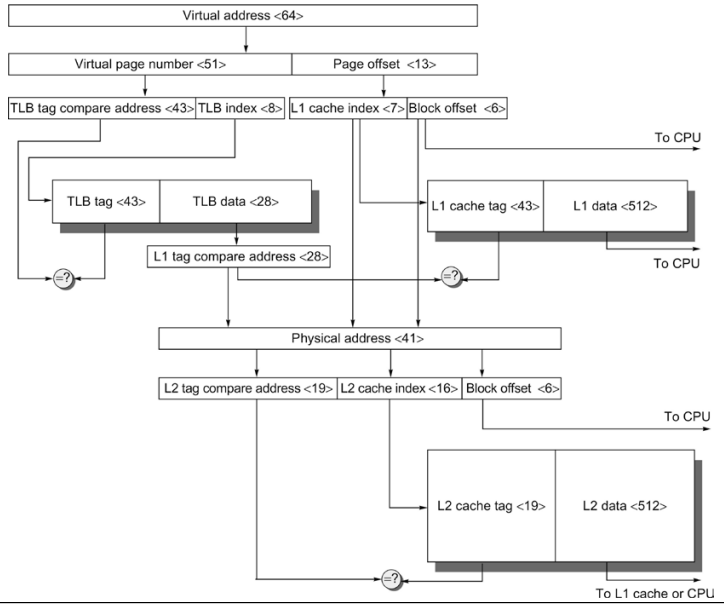
Page Table					
Index	Page #	Resident	Dirty	Disk Addr	...
0	?	?	?	?	...
...	...	...	...	...	...
71	9	Yes	?	?	...
...	...	...	...	...	...

- Given virtual address 0x8FDF, what is the physical address?
- Given virtual address 0x10DF, what is the physical address?

Selecting a Page Size

- Advantages of Using a Large Page Size
  - The bigger the page size, the fewer the entries in the page table and the smaller the page table.
  - Larger pages may allow the cache access to occur in parallel with the virtual to physical address translation (VIPT).
  - Transferring larger pages to/from disk is more efficient since fewer seeks are required.
  - Will probably result in fewer TLB misses since there are fewer distinct pages referenced.
- Advantages of Using a Smaller Page Size
  - Will waste less space.
  - The miss penalty for a page fault and program startup time will decrease.

Memory Hierarchy from Virtual Address to L2 Cache Access



Fallacies and Pitfalls

- Pitfall: Too small an address space.
- Pitfall: Ignoring the impact of the operating system on the performance of the memory hierarchy.