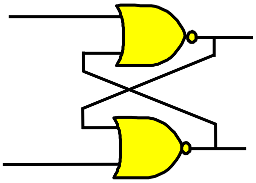
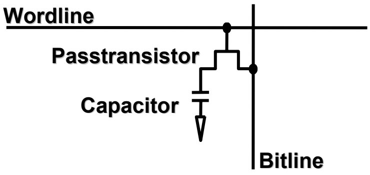
微算機系統 第七章
Large and Fast: Exploiting Memory Hierarchy

陳伯寧 教授
電信工程學系
國立交通大學

Memories: Review

- SRAM:
  - value is stored on a pair of inverting gates
  - very fast but takes up more space than DRAM (4 to 6 transistors)
- DRAM:
  - value is stored as a charge on capacitor (must be refreshed)
  - very small but slower than SRAM (factor of 5 to 10)



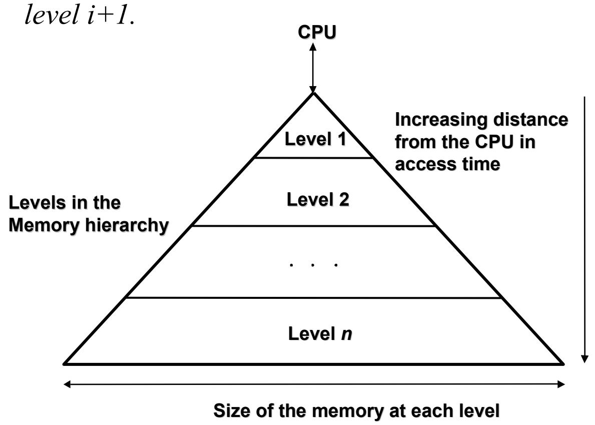


Exploiting memory hierarchy

- *Users want large and fast memories! (Information collected on 2004)*
  - *SRAM access times are 0.5~5ns at cost of US$4,000~$10,000 per Gbyte.*
  - *DRAM access times are 50~70ns at cost of US$100~200 per Gbyte.*
  - *Disk access times are 5 to 20 million ns at cost of US$0.5~$2 per Gbyte.*
- *A cost-effective approach*
  - *build a memory hierarchy*

Exploiting memory hierarchy

- *An unbreakable rule*
  - *The data cannot be present in level i unless it is present in level i+1.*



CPU

Level 1

Level 2

. . .

Level *n*

Levels in the Memory hierarchy

Increasing distance from the CPU in access time

Size of the memory at each level

Locality

- *Locality: A principle that makes having a memory hierarchy a good idea*
- *If an item is referenced,*

  temporal locality (locality in time) : it will tend to be referenced again soon

  spatial locality (locality in space): nearby items will tend to be referenced soon.
- *Our initial focus: Take two-level hierarchy as an example (upper, lower)*
  - *block: minimum unit of data transferring between two levels*
  - *hit: data requested is in the upper level*
  - *miss: data requested is not in the upper level*

Locality

- *Example of temporal locality in programs*
  - *Instructions in Loops*
- *Example of spatial locality in programs*
  - *Sequentially executed instructions*
- *Example of two-level hierarchy*
  - *Main memory (DRAM) – lower level*
  - *Cache (SRAM) – upper level*

Terminologies

- *Hit rate or hit ratio*
  - *Fraction of memory accesses found in the upper level*
- *Miss rate or miss ratio*
  - *Fraction of memory accesses not found in the upper level*
- *Hit time*
  - *The time to access the upper level of memory hierarchy, including the time to determine whether the access is a hit or a miss, but not including the retrieve time of a block from lower memory to higher memory when a miss occurs.*
  - *The block retrieve time is referred to as miss penalty.*

How to take advantage of program locality?

- *Keep more recently accessed data items closer to the processor (take advantage of temporal locality)*
- *Move blocks consisting of multiple continuous words in memory ((take advantage of spatial locality)*

Cache

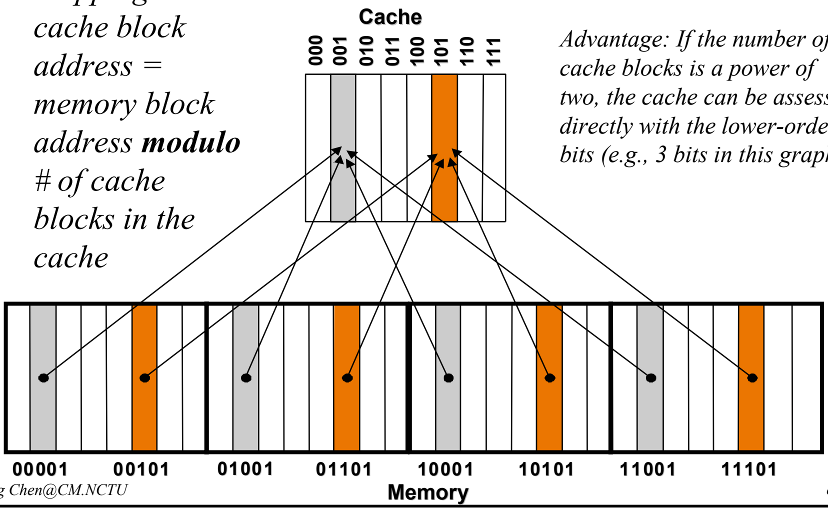
- *Two issues:*
  - *How do we know if a data item is in the cache?*
  - *If it is, how do we find it?*
- *Our first (simplified) example:*
  - *block size is one word of data*
  - *"direct mapped"*

**For each item of data at the lower level,
there is exactly one location in the cache where it might be.**

e.g., lots of items at the lower level share locations in the upper level

Direct mapped cache

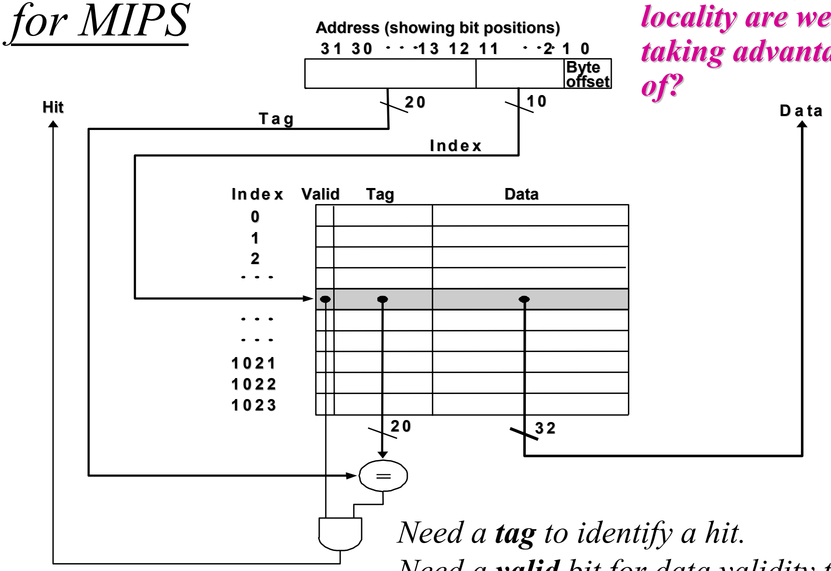
- *Mapping: cache block address = memory block address **modulo** # of cache blocks in the cache*



Advantage: If the number of cache blocks is a power of two, the cache can be assessed directly with the lower-order bits (e.g., 3 bits in this graph).

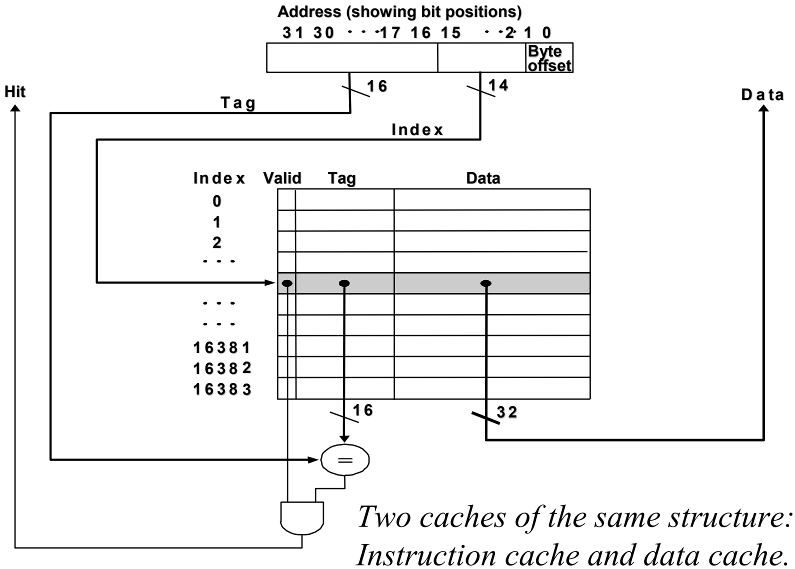
Direct mapped cache for MIPS

In this design, what kind of locality are we taking advantage of?



*Need a **tag** to identify a hit.*
*Need a **valid** bit for data validity test.*

A true example: Caches in DECStation 3100



Two caches of the same structure:
Instruction cache and data cache.

Miss rate for DECStation 3100

Program	*Instruction miss rate*	*Data miss rate*	*Overall miss rate*
gcc	*6.1%*	*2.1%*	*5.4%*
spice	*1.2%*	*1.3%*	*1.2%*

How about a hit in memory write?

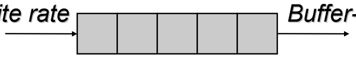
- *"Memory read" is straightforward for a cache system.*
  - *A miss causes a re-fill of the cache from the memory.*
- *How about a hit in memory write?*
  - *Can we write into the cache without changing the respective memory content?*
  - *Answer: Yes, but the "inconsistent" between the cache content and memory content should be explicitly handled.*
- *Approach taken by DECStation 3100:* ***Write-through cache***
  - ***Always write the data into both the memory and cache.***

How about a miss in memory write?

- *Approach taken by DECStation 3100 for a miss in memory write:* ***Write-through cache***
  - ***Always write the data into both the memory and cache.***
- *Observation*
  - *Poor performance of a write-through cache: A memory access is resulted for every write, regardless of whether it is a miss or a hit.*
  - *Example: Program gcc, whose CPI without considering memory miss is 1.2, requires 13% of memory write. If a memory-write requires 10 cycles, then the resultant CPI = 1.2 + 10 * 13% = 2.5—two times slower in performance.*

Performance improvement of write-through caches

- *Solution 1: Using write buffer*
  - *Writing to a quick write buffer instead of directly to memory.*
  - *Then "execution" and "buffer-to-memory-update" can be done simultaneously.*

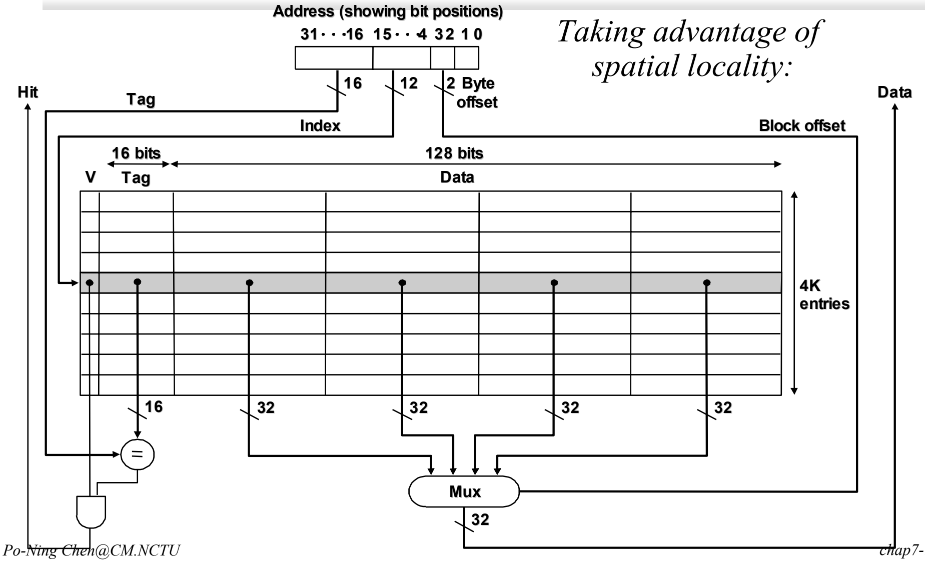


CPU-generate-write rate

Buffer-memory-update rate

- *Solution 2:* ***Write-back cache***
  - *When a write occurs, the new value is written only to the block in the cache.*
  - *The modified block is written to memory when a miss causes a replace in that block.*

Direct mapped cache considering spatial locality



Taking advantage of spatial locality:

Hit versus miss: A summary

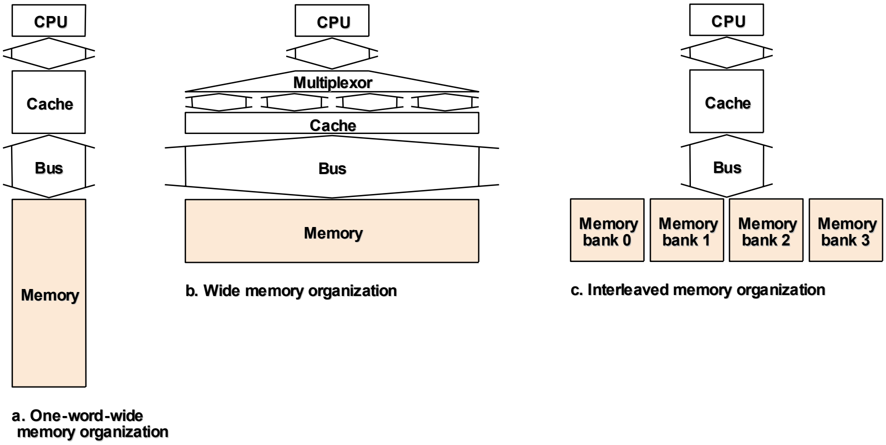
- *Read hits*
  - *this is what we want!*
- *Read misses*
  - *stall the CPU, fetch block from memory, deliver to cache, restart*
- *Write hits:*
  - *can replace data in cache and memory (write-through)*
  - *write the data only into the cache (write-back the cache later)*
- *Write misses:*
  - *read the entire block into the cache, then write the word*

Hit versus miss: A summary

- *"Block transfer latency" could be a penalty to large block size. Solutions:*
  - *Early restart*
    - *Resume execution as soon as the requested word of the block is returned.*
  - *Requested word first or critical word first*
    - *Memory transfer starts with the address of the requested word; then wraps around to the beginning of the block.*
  - *Large block transfer bandwidth*
    - *E.g., structures b and c in the next slide.*

Hardware issues

- *Make reading multiple words easier by using banks of memory*



Hardware issues

- How multiple-word access work?
  - Suppose:
    - A cache block = 4 words
    - x (e.g., 1 ns) clock to send the memory address
    - y (e.g., 15ns) clocks for memory access initialization
    - z (e.g., 1ns) clock for memory content access
  - Miss penalty of structure a = x + 4y + 4z
  - Miss penalty of structure b = x + y + z
  - Miss penalty of structure c = x + y + 4z

Memory device: Appendix B-Will not be covered in exam.

- Read-Only Memory (ROM) : data remain unchanged even if the power is turned off. (Can be possibly writable!)

- Static Random-Access Memory (SRAM)

- Dynamic Random-Access Memory (DRAM)

Memory device

- *Read-Only Memory (ROM)*
  - *PROM (Programmable ROM)*
    - *Fuse-structured*
    - *Data written by burning open the tiny silicon oxide fuse*
  - *EPROM (Erasable Programmable ROM)*
    - *Data erased by (about) 20-minute exposing to high-intensity ultraviolet light*
    - *Data written by EPROM writer*
  - *Read-mostly memory or flash ROM or EEPROM (electronically EPROM) or EAROM (electronic alterable ROM) or NOVRAM (nonvolatile ROM)*
    - *The time for write operations is usually in the range of milliseconds; while the time for read operations is only in the range of nanoseconds.*

Memory device

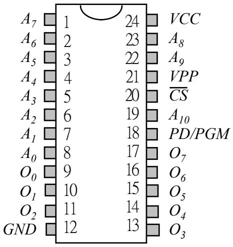
- *Naming Convention*

Names	*Size*	
*27**04***	*4/8=0.5K=512*	*512×8*
*27**08***	*8/8=1K*	*1K×8*
*27**16***	*16/8=2K*	*2K×8*
*27**32***	*32/8=4K*	*4K×8*
*27**64***	*64/8=8K*	*8K×8*
*27**128***	*128/8=16K*	*16K×8*
*27**256***	*256/8=32K*	*32K×8*
*27**512***	*512/8=64K*	*64K×8*
*27**1024***	*1024/8=128K*	*128K×8*

Memory device

- *Example. Pinout and Mode Selection of 2716 writable ROM.*

PIN Configuration



2716

Mode Selection

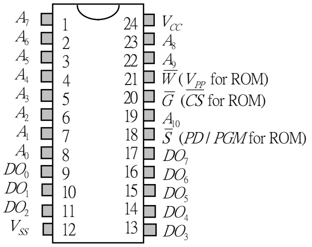
PINS MODE	PD/PGM	$\overline{CS}$	VPP	VCC	OUTPUTS
Read	V_L	V_L	+5	+5	Data out
Deselect	Don't care	V_H	+5	+5	High-Z
Power Down	V_H	Don't care	+5	+5	High-Z
Program	pulsed V_L to V_H	V_H	+25	+5	Data in
Program Verify	V_H	V_L	+25	+5	Data out
Program Inhibit	V_H	V_H	+25	+5	High-Z

Memory device

- *Static Random Access Memory (SRAM) -only requires a static DC power, and hence, is named Static RAM.*
  - *The data stored will disappear after power-off.*
  - *The speed of SRAM is usually faster than the speed of DRAM, but is more expensive though. Also, its size is often smaller than that of DRAM. Hence, it is often used as an "external cache memory."*

Memory device

- *Pinouts of 4016 (=4016=6116)*



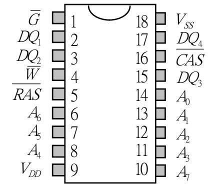
TMS4016

Memory device

- *Dynamic Random Access Memory (DRAM) - requires not only a static DC power, but also a dynamic refresh clock.*
  - *The data stored will disappear after power-off.*
  - *The data stored in DRAM will disappear if the data is not refreshed/rewritten within several milliseconds (i.e., if the inside capacitors are not re-charged within several milliseconds.)*
  - *Since the size of DRAM is often large, and the number of pins on its package are usually not enough. Hence, **multiplexing the pins** seems to be the only solution. This **complicates** the design of DRAM access circuit.*

Memory device

- *Example. TMS 4464.*
  - *Note : position of VDD (+5V) is not at pin 18. (convention of DRAM)*



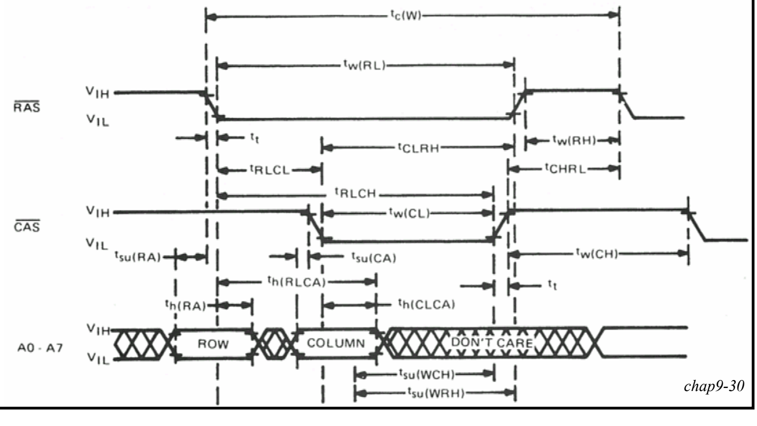
$\overline{CAS}$ \ $\overline{RAS}$	00H	01H	02H	03H	04H	...	FEH	FFH
00H	4bit	4bit	4bit	4bit	4bit	...	4bit	4bit
01H	4bit	4bit	4bit	4bit	4bit	...	4bit	4bit
02H	4bit	4bit	4bit	4bit	4bit	...	4bit	4bit
03H	4bit	4bit	4bit	4bit	4bit	...	4bit	4bit
04H	4bit	4bit	4bit	4bit	4bit	...	4bit	4bit
:	:	:	:	:	:	...	:	:
FEH	4bit	4bit	4bit	4bit	4bit	...	4bit	4bit
FFH	4bit	4bit	4bit	4bit	4bit	...	4bit	4bit

TMS4464 (64K× 4 DRAM)

- $\overline{RAS}$ *: Row Address Strobe* ($A_0 \sim A_7$)
- $\overline{CAS}$ *: Column Address Strobe* ($A_8 \sim A_{15}$)

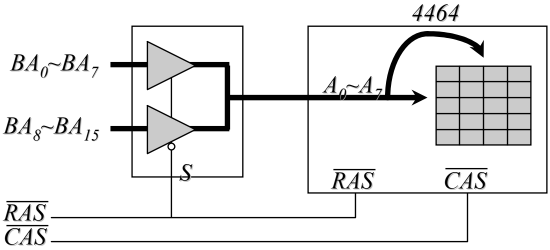
Memory device

- *Example. TMS 4464.*
  - *Timing Diagram for /RAS and /CAS.*



Memory device

- *Example. TMS 4464.*
  - *Multiplexing of address lines.*



Memory device

- *The name convention of DRAM is different from ROM and SRAM.*
  - *4464 = 64K × 4*
  - *41256 = 256K × 1*
  - *4C1024 = 1024K × 1*
- *Year mark convention of ICs*
  - *Example. 9021 = This IC was made in the 21th week of 1990.*
  - *Example. 8931 = This IC was made in the 31th week of 1989.*

Dynamic RAM

- *Refreshing*
  - *Data in DRAM needs to be refreshed within several milliseconds.*
  - *There are three ways to refresh the data:*
    - *Read (automatically refresh)*
    - *Write (automatically refresh)*
    - *A non-Read and non-Write refresh. (Sometimes referred to as hidden refresh, transparent refresh, or cycle stealing)*

Dynamic RAM

Normal memory Access 1
Normal memory Access 2
:
Normal memory Access 19
READ memory XXX00H (refresh 1st row)
Normal memory Access 20
:
Normal memory Access 38
READ memory XXX01H (refresh 2nd row)
Normal memory Access 39
:
Normal memory Access 57
READ memory XXX02H (refresh 3rd row)
Normal memory Access 58
:
Normal memory Access 4864
READ memory XXX03H (refresh 255th row)

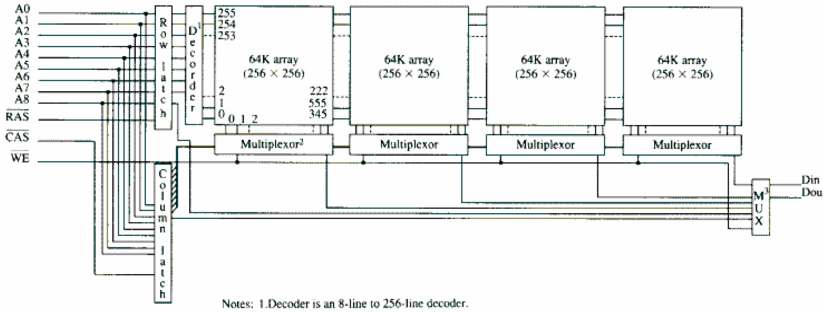
Will lose 5% of CPU time for refreshing !!!

Dynamic RAM

- *Hidden refresh.*
  - *To refresh part of DRAMs while the other parts of DRAMs are functioning (reading or writing) simultaneously.*
  - *This kind of refresh is (usually) not seen by the CPU, and hence, is named so. Its operation needs to be performed by an external (specially designed) circuit.*

Dynamic RAM

- *Hidden refresh = /RAS-only refers cycle. (Note that a true Read or Write cycle requires both /RAS and /CAS.)*



Notes: 1.Decoder is an 8-line to 256-line decoder.
2.Multiplexor is 256 to 1 line.
3.Multiplexor is 4 to 1 line.

Various DRAMs

- Page-mode (static-column-mode) DRAM
  - Provide the ability to access multiple bits out of a row by changing the column address only.
- EDO (extended-data-out) DRAM
  - The latest version of page-mode DRAM
- Nibble-mode DRAM
  - Internally generate the next three column addresses, thus providing 4 bits (called a nibble) for every row access.

Synchronous RAM

- Synchronous SRAM and Synchronous DRAM (SDRAM)
  - Provide ability to transfer a burst of data from a series of sequential addresses.
  - The burst is defined by “starting address” s (or even just the starting row address for page-mode-style SDRAM) and “burst length” b.
  - So using the notations in slide 7-21, the transfer will take $x + y + b * z$ rather $b * (x + y + z)$.
  - Very helpful in cache block transfer.

Error correction in memory

- *Parity Check*
  - *Example. Even Parity*
    - *Count the number of 1's (in a byte).*
    - *Check if this number is even.*
  - *Example. Odd Parity*
    - *Count the number of 1's (in a byte).*
    - *Check if this number is odd.*
  - *One-bit error can be detected; but two-bit error cannot be observed. (Again, 3-bit error can be detected, but 4-bit error cannot be observed.....)*
- *Need additional one-bit for parity; hence, the width of a byte becomes 9 bits instead of 8.*

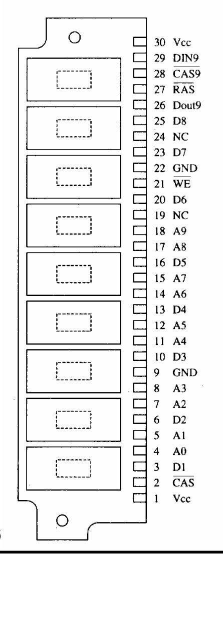
Error correction in memory

- *Error Correction Coding memory(ECC memory)*
  - *32 kinds of parities, instead of 2 (designed using modified Hamming code).*
  - *Need additional 5 bits; hence, a byte consists of 13 bits instead of 8.*
  - *Able to correct any one-bit error, and can detect any two-bit error. Notably, the erroneous bits includes the parity bits themselves.*

DRAM Interface

- DRAM Interface
  - Single in-line memory module (SIMM)
  - Dual in-line memory module (DIMM)
  - 30pin, 72pin, 168pin

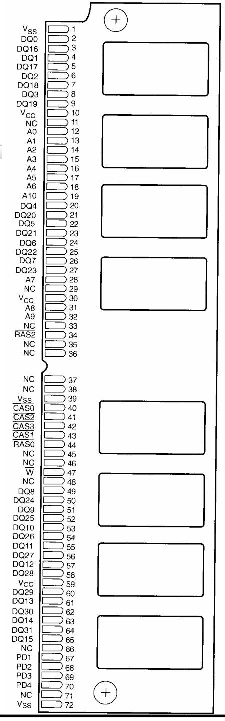
DRAM Interface



- 30pin SIMM
  - Din9(29), /CAS9(28), Dout9(26) : Parity check
  - At most 1M×9
  - Combined two NC pins to extend to 4M ×9.
    - A10(19)
    - A11(24)

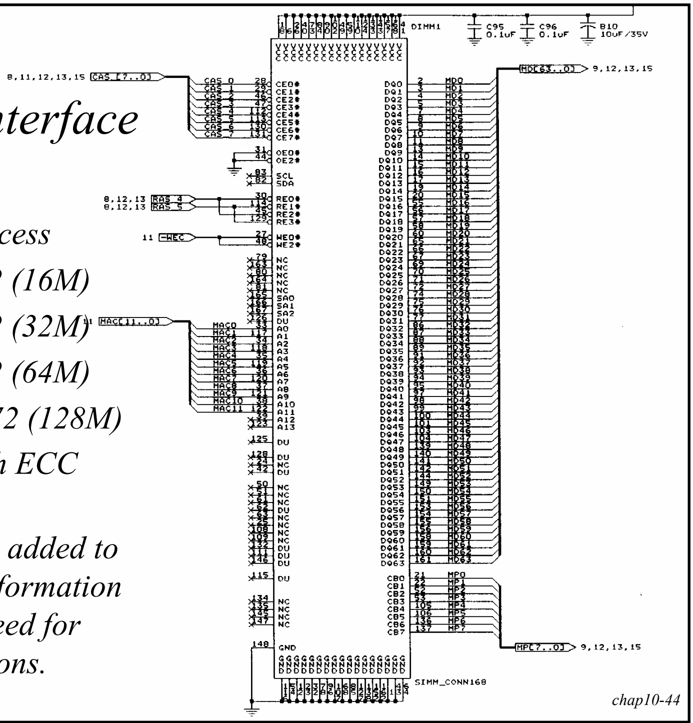
DRAM Interface

- 72-pin SIMM
  - 32-bit data access
  - 1M × 32 (4Mbytes)
  - 1M × 36 (4Mbytes)(with parity)
  - 2M × 32 or 36 (8Mbytes)
  - 4M × 32 or 36 (16Mbytes)
  - 8M × 32 or 36 (32Mbytes)



DRAM Interface

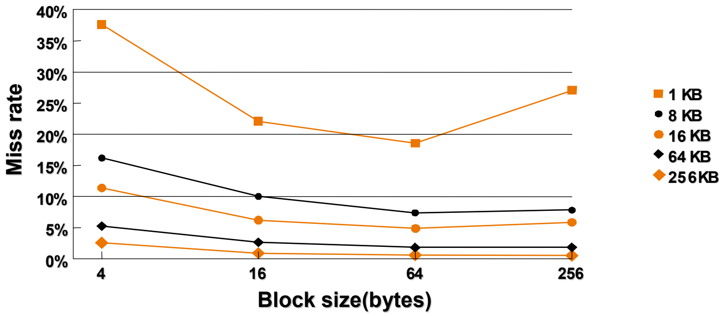
- 168pin SIMM
  - 64-bit data access
  - 2M × 64 or 72 (16M)
  - 4M × 64 or 72 (32M)
  - 8M × 64 or 72 (64M)
  - 12M × 64 or 72 (128M)
  - (possible) with ECC capability
  - An EPROM is added to provide the information of memory speed for PnP applications.



Now back to the exam covered subjects!

Performance

- *Increasing the block size tends to decrease miss rate:*



Performance

- *Use split caches because there is more spatial locality in code:*
    - *Notably in the next table, considering more spatial locality by increasing block size decreases the Instruction miss rate more.*

Program	Block size in words	Instruction miss rate	Data miss rate	Effective combined miss rate
gcc	1	6.1%	2.1%	5.4%
	4	2.0%	1.7%	1.9%
spice	1	1.2%	1.3%	1.2%
	4	0.3%	0.6%	0.4%

Performance

- *Simplified model:*

  execution time = (execution cycles + stall cycles) × cycle time

  *stall cycles = # of instructions × **miss ratio** × **miss penalty***

- *Two ways of improving performance:*
    - *decreasing the **miss ratio***
    - *decreasing the **miss penalty***

What happens if we increase block size?

Decreasing miss ratio, but possibly increasing miss penalty.

Example

- *Given*
  - *Instruction cache miss rate is 2%*
  - *Data cache miss rate is 4%*
  - *36% of instructions are load-and-store*
  - *Miss penalty = 100 cycles*
  - *CPI without memory stall = 2 cycles/instruction*
- *What is the performance degradation of such a system, if compared to a system with a perfect no-miss cache*

Example

- *Let I = # of instructions*
  - *Instruction memory-stall cycles = Instruction miss cycles = I * 2% * 100 = 2I*
  - *Data memory-stall cycles = Data miss cycles = I * 36% * 4% * 100 = 1.44 I*
  - *(total) Memory-stall cycles = 2I + 1.44I = 3.44I.*

$$\frac{\text{CPU Time}_{\text{with stall}}}{\text{CPU Time}_{\text{perfect cache}}}$$

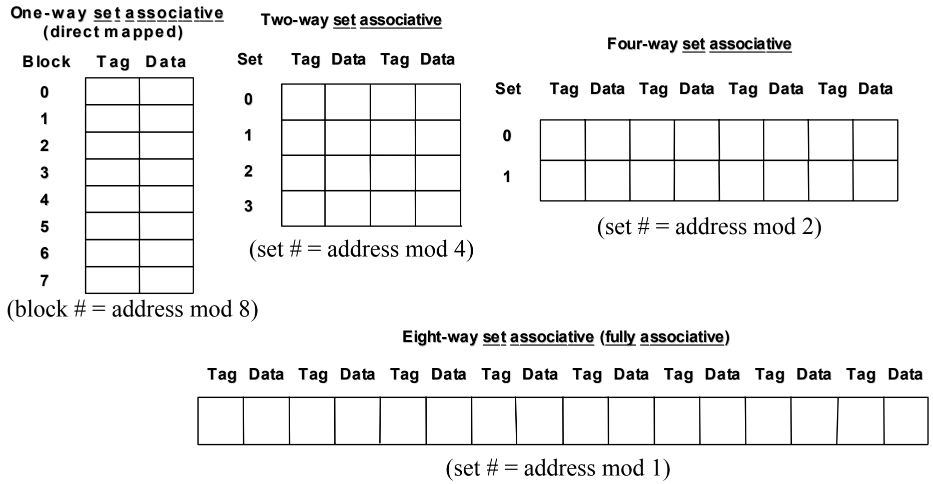
$$= \frac{(\text{CPU execution clock cycles} + \text{Memory stall clock cycles}_{\text{with stall}}) \times \text{Clock cycle time}}{(\text{CPU execution clock cycles} + \text{Memory stall clock cycles}_{\text{perfect cache}}) \times \text{Clock cycle time}}$$

$$= \frac{(2I + 3.44I) \times \text{Clock cycle time}}{(2I + 0I) \times \text{Clock cycle time}} = \frac{5.44}{2} = 2.72$$

Performance

- *Final note on performance*
  - *Under a fixed miss penalty (i.e., a fixed memory access time) which machine should pay more attention to cache design? In other words, which machine will degrade more if an ill design in cache system is adopted.*
    - *CPU with low CPI and high clock rate*
    - *CPU with high CPI and low clock rate*
  - *Answer: See the examples on page 495.*

Decreasing miss rate by associativity



One-way set associative (direct mapped)

Block Tag Data

0 1 2 3 4 5 6 7

(block # = address mod 8)

Two-way set associative

Set Tag Data Tag Data

0 1 2 3

(set # = address mod 4)

Four-way set associative

Set Tag Data Tag Data Tag Data Tag Data

0 1

(set # = address mod 2)

Eight-way set associative (fully associative)

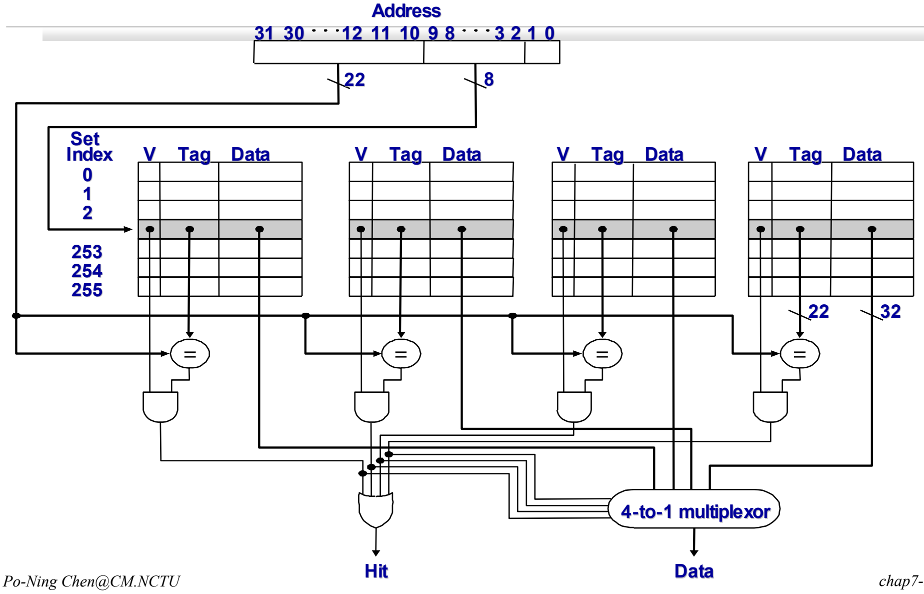
Tag Data Tag Data Tag Data Tag Data Tag Data Tag Data Tag Data Tag Data

(set # = address mod 1)

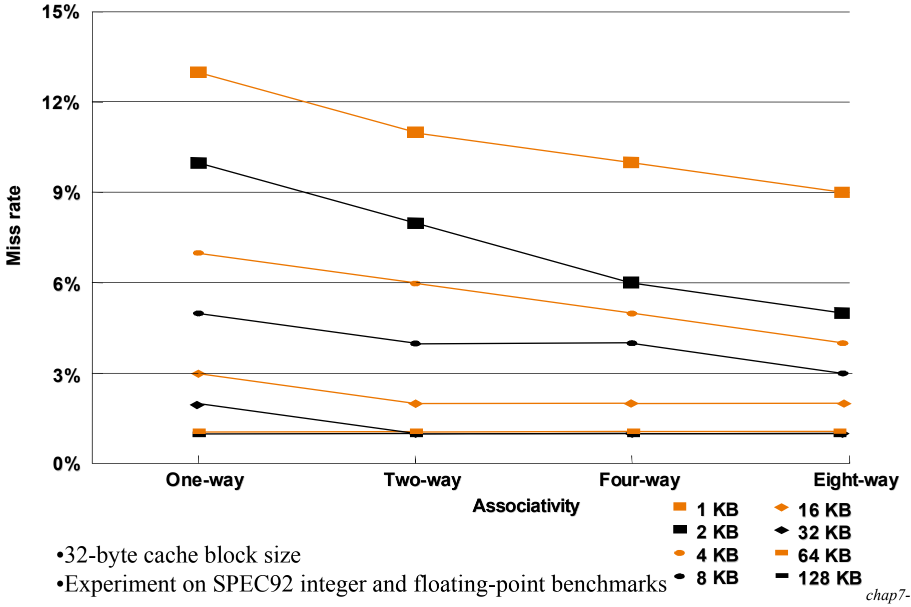
Decreasing miss rate by associability

- *n*-way set associative cache
  - A memory block is directly mapped into a set;
  - All the cache blocks in a set are searched for a hit match (which increase hit time).

Exemplified implementation



Performance of associability



- 32-byte cache block size
- Experiment on SPEC92 integer and floating-point benchmarks

Which block should be replaced in set-associate cache?

- *LRU (least recently used) scheme*
  - *The block replaced is the one that has been unused for the longest time.*

Decreasing miss penalty by multilevel cache

- *Add a second level cache:*
  - *often primary cache is on the same chip as the processor*
  - *use SRAMs to add another cache above primary memory (DRAM)*
  - *miss penalty goes down if data is in 2nd level cache*
- *Planning multilevel caches:*
  - *try and optimize the hit time on the 1st level cache*
  - *try and optimize the miss rate on the 2nd level cache*

Decreasing miss penalty by multilevel cache

- *Example:*
  - *CPI of 1.0 on a 5Ghz machine with a 2% miss rate, 100ns DRAM access*
  - *Adding 2nd level cache with 5ns access time decreases miss rate (of the 2nd level cache) to 0.5% (Note that the miss rate of the first cache is still 2%).*
  - *How much faster after adding the 2nd level cache?*

  - *1st-cache-miss-penalty-without-2nd-cache = 2nd-cache-miss-penalty = 100ns / 0.2 (ns/cycle) = 500 cycles*
  - *1st-cache-miss-penalty-with-2nd-cache = 5ns / 0.2 (ns/cycle) = 25 cycles*

$$\boxed{\text{Speed Up} = \frac{(I + 2\% \times 500I) \times \text{Clock cycle tim}}{(I + 2\% \times 25I + 0.5\% \times 500I) \times \text{Clock cycle time}} = \frac{11}{4} = 2.8}$$

* I is the number of Instructions of the benchmark program.

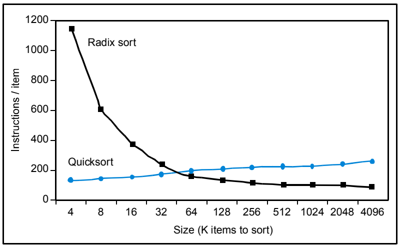
Decreasing miss penalty by multilevel cache

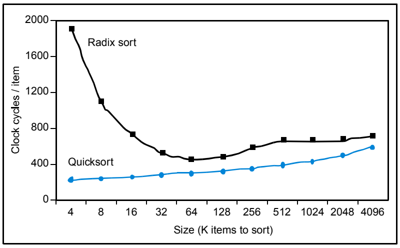
- *Design principle*
  - *To minimize the **hit time** of the first cache*
  - *To minimize the **miss rate** of the second cache*

- *Final note*
  - ***Local miss rate** of the second-level cache*
    - *The number of misses in the second-level cache divided by the number of accesses to the second-level cache*
  - ***Global miss rate** of the second-level cache*
    - *The number of misses in the second-level cache divided by the number of accesses to the entire cache system*
  - *What is the **local miss rate** of the second cache in the previous example?*
    - *(0.5% * I) / (2% * I) = 25%*

Cache complexity

- *Not always easy to understand implications of caches:*



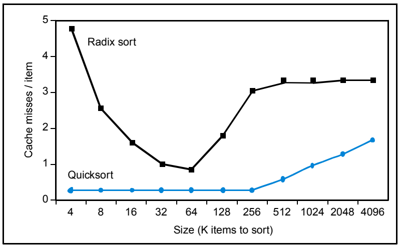


Theoretical behavior of Radix sort vs. Quicksort

Observed behavior of Radix sort vs. Quicksort

Cache complexity

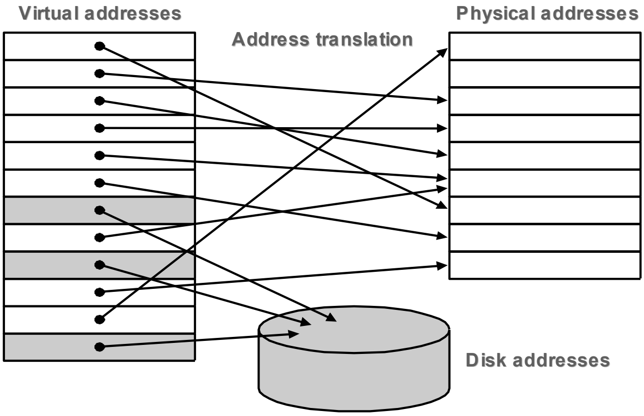
- *Here is why:*



- *Memory system performance is often critical factor*
  - *multilevel caches, pipelined processors, make it harder to predict outcomes*
  - *Compiler optimizations to increase locality sometimes hurt ILP*
- *Difficult to predict best algorithm: need experimental data*

Virtual memory

- *Main memory can act as a* ***cache*** *for the* ***secondary storage****, i.e.,* ***disk.***



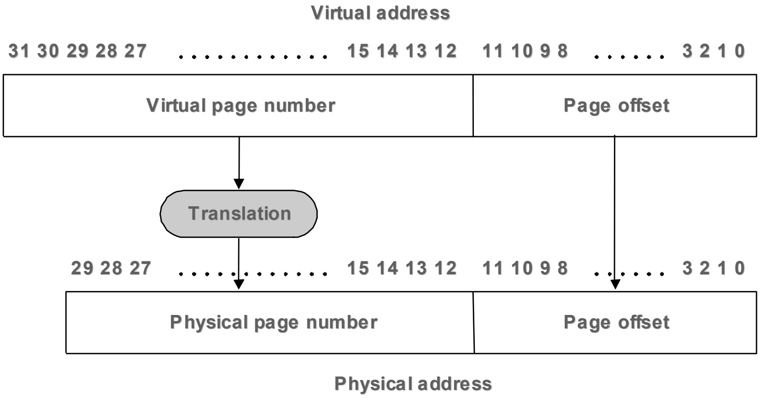
Virtual memory

- *Advantages:*
  - *illusion of having more physical memory (to programmer)*
  - *program relocation (multiple programs efficiently share the physical memory)*
    - *The total memory required by all programs may be much larger than the amount of main memory available.*
    - *However, main memory is only required to contain the active portion of each program at a time.*
  - *Protection*
    - *A program can only read and write the portions of the main memory that have assigned to it.*

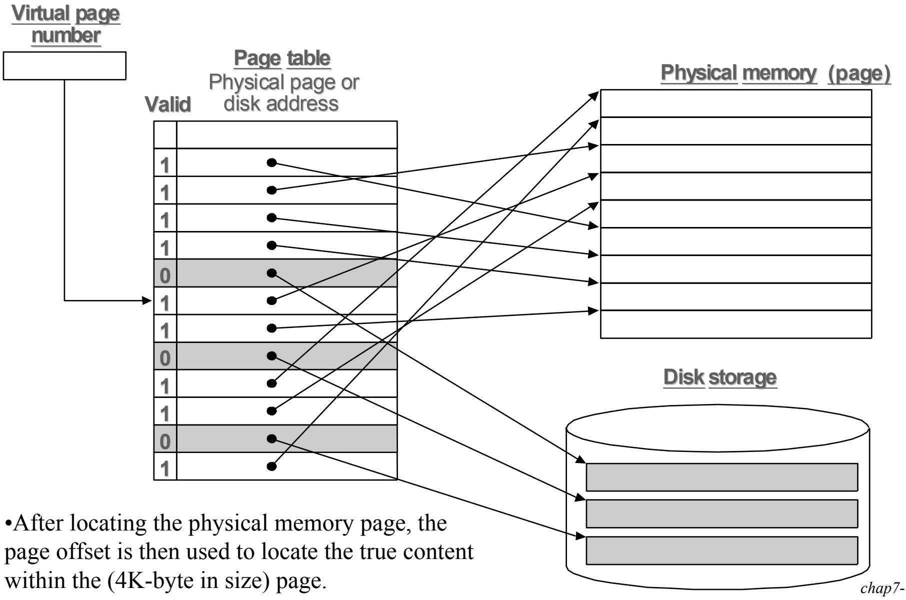
Virtual memory blocks = pages

- *Page faults: the data is not in memory, retrieve it from disk*
  - *huge miss penalty, thus pages should be fairly large (e.g., 4KB)*
  - *reducing page faults is important (LRU is worth the price)*
    - *LRU is worth the price*
    - *Full-associativity becomes an attractive choice*
  - *can handle the faults in software instead of hardware*
    - *Compared to large page-fault penalty (miss penalty), software handling overhead can be reasonably neglected.*
    - *By software, efficient but complicated algorithm for page handling becomes justifiable.*
  - *using write-through is too expensive so we use* ***writeback***

Virtual memory blocks = pages

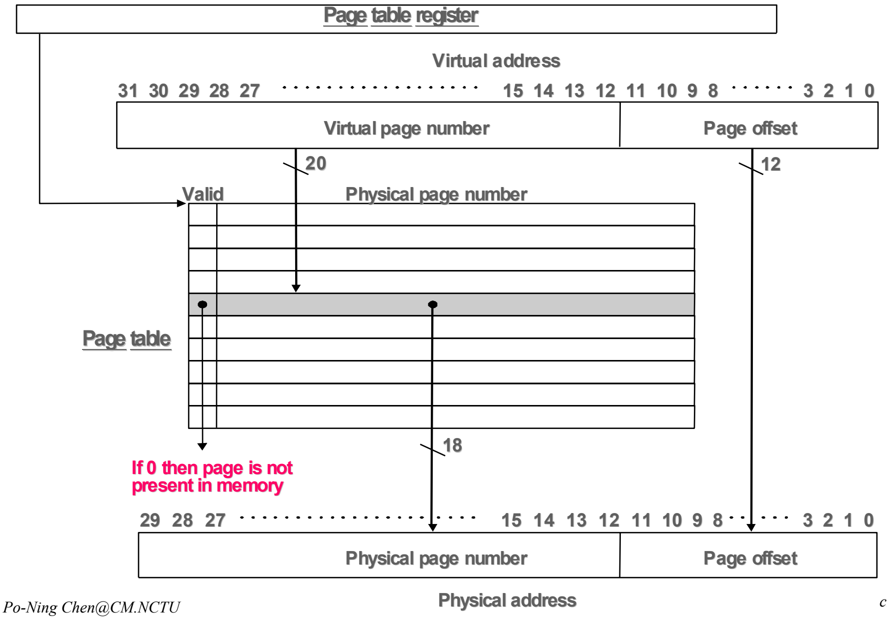


Page tables



•After locating the physical memory page, the page offset is then used to locate the true content within the (4K-byte in size) page.

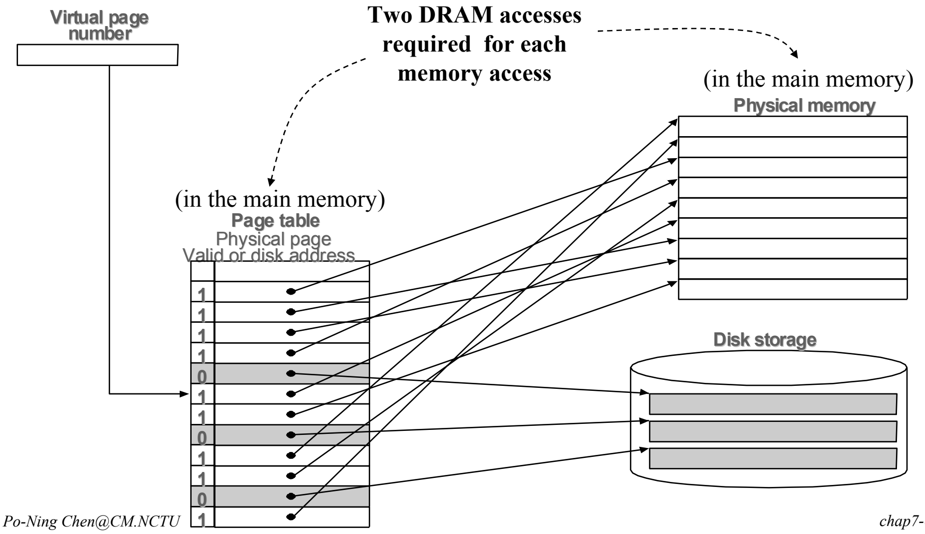
Page table register *and* ***page table***



Flags for paging system

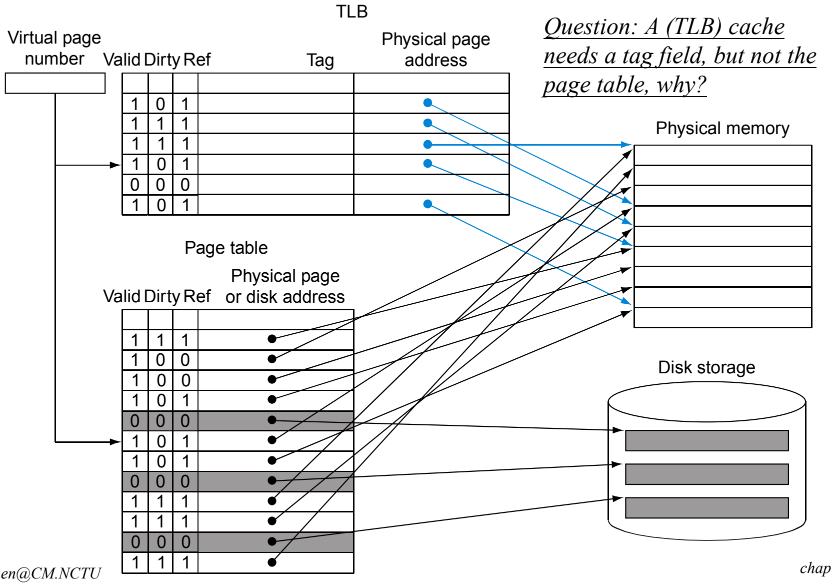
- *Reference or use bit*
  - *An exact LRU scheme is expensive in implementation.*
  - *So a simple approximate LRU scheme is:*
    - *Periodic clears the* ***reference bit*** *of each page.*
    - *Set the* ***reference bit*** *when a page is accessed.*
    - *Select a page to replace, whose* ***reference bit*** *is clear, whenever necessary.*
- *Dirty bit*
  - *A write-back operation is costly.*
  - *So a dirty-bit is added to the page table, which is set when the page is first written, and which indicates whether the page needs to be written out.*

Making address translation fast



Two DRAM accesses required for each memory access

- *Translation-lookaside buffer (TLB) = A cache that keep the LRU entry in the page table*
- *Also need additional flags, such as reference bit.*



TLB

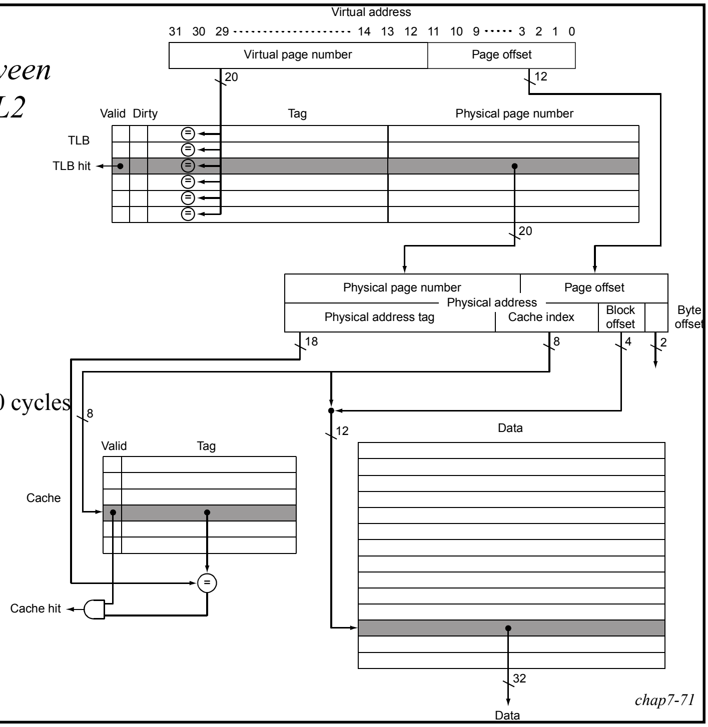
Valid	Dirty	Ref	Tag	Physical page address
1	0	1		
1	1	1		
1	1	1		
1	0	1		
0	0	0		
1	0	1		

Page table

Valid	Dirty	Ref	Physical page or disk address
1	1	1	●
1	0	0	●
1	0	0	●
1	0	1	●
0	0	0	●
1	0	1	●
1	0	1	●
0	0	0	●
1	1	1	●
1	1	1	●
0	0	0	●
1	1	1	●

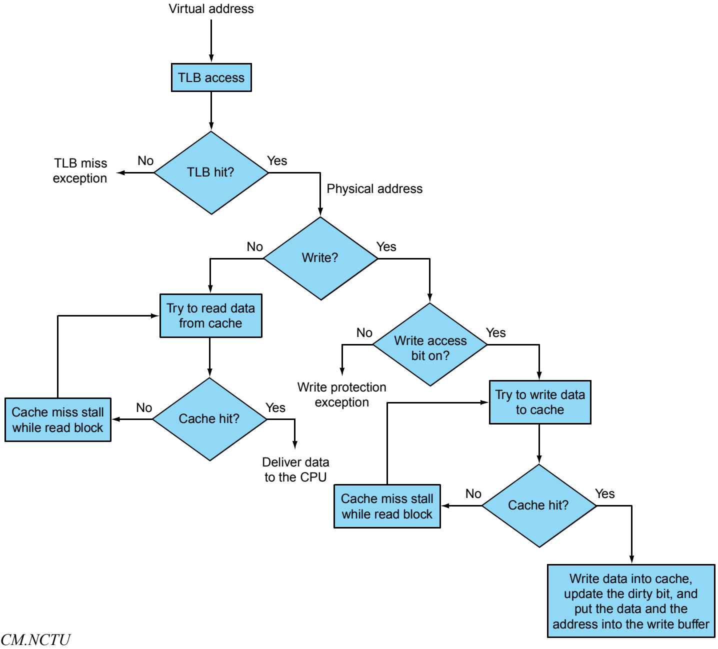
Question: A (TLB) cache needs a tag field, but not the page table, why?

- *Relation between TLB and L1/L2 cache*



Typical values of TLB:
16-512 entries,
miss-rate: .01% - 1%
miss-penalty: 10 – 100 cycles

TLB and (L1/L2) cache



Virtual address
TLB access
TLB hit?
No
TLB miss exception
Yes
Physical address
Write?
No
Yes
Try to read data from cache
Write access bit on?
No
Write protection exception
Yes
Try to write data to cache
Cache hit?
No
Cache miss stall while read block
Yes
Deliver data to the CPU
Cache hit?
No
Cache miss stall while read block
Yes
Write data into cache, update the dirty bit, and put the data and the address into the write buffer

Possible combinations

Cache	*TLB*	*Page table*	*Comments*
miss	*miss*	*miss*	*TLB misses, followed by a page fault; after retry, data must miss in cache*
miss	*miss*	*hit*	*TLB misses, but entry found in page table; after re-try, data misses in cache*
miss	*hit*	*miss*	***Impossible to occur**; cannot have a translation in TLB if page is not present in memory*
miss	*hit*	*hit*	***Possible**, although the page table is never really checked if TLB bit.*
hit	*miss*	*miss*	***Impossible to occur**; data cannot be allowed in ache if the page is not in memory (see the next slide)*
hit	*miss*	*hit*	*TLB misses, but entry found in page table; after re-try, data is found in cache*
hit	*hit*	*miss*	***Impossible to occur**; cannot have a translation in TLB if page is not present in memory*
hit	*hit*	*hit*	***Excellent situation***

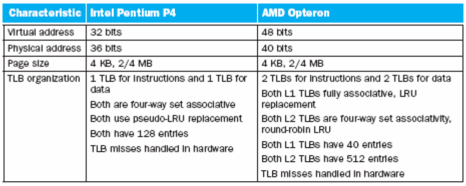
Final note

- *When the system decides to migrate a page to disk, the page shall be flushed from the cache.*
  - *Notably, to put a page in cache, which in not in main memory (be migrated to disk), does not make sense; so the situation should be prevented.*
- *The TLB access and cache access can possibly be **pipelined**, where TLB access translated a virtual address to physical address first, followed by cache access that locates the true content based on the physical address.*
- *Two processes can share the same data by mapping two virtual addresses to the same physical addresses; in such case, protection against malicious usage of the data becomes an additional concern.*

Summary

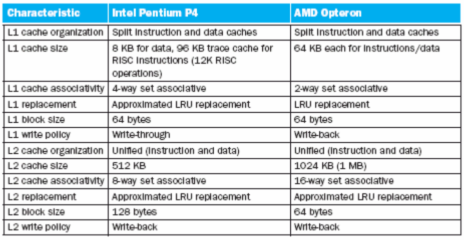
- *You shall know:*
  - ***Directed map** versus **set associative** versus **fully associative***
  - *Cache block founding through **indexing** (as directed mapped), **limited search** (as set-associative), **full search** (as fully associative) and **separate lookup table** (as page table)*
  - *Replacing blocks using **LRU** or **random***
  - ***Write through** or **write back** (copy back)*

Modern Systems



Characteristic	Intel Pentium P4	AMD Opteron
Virtual address	32 bits	48 bits
Physical address	36 bits	40 bits
Page size	4 KB, 2/4 MB	4 KB, 2/4 MB
TLB organization	1 TLB for instructions and 1 TLB for data Both are four-way set associative Both use pseudo-LRU replacement Both have 128 entries TLB misses handled in hardware	2 TLBs for instructions and 2 TLBs for data Both L1 TLBs fully associative, LRU replacement Both L2 TLBs are four-way set associativity, round-robin LRU Both L1 TLBs have 40 entries Both L2 TLBs have 512 entries TLB misses handled in hardware

FIGURE 7.34 Address translation and TLB hardware for the Intel Pentium P4 and AMD Opteron. The word size sets the maximum size of the virtual address, but a processor need not use all bits. The physical address size is independent of word size. The P4 has one TLB for instructions and a separate identical TLB for data, while the Opteron has both an L1 TLB and an L2 TLB for instructions and identical L1 and L2 TLBs for data. Both processors provide support for large pages, which are used for things like the operating system or mapping a frame buffer. The large-page scheme avoids using a large number of entries to map a single object that is always present.

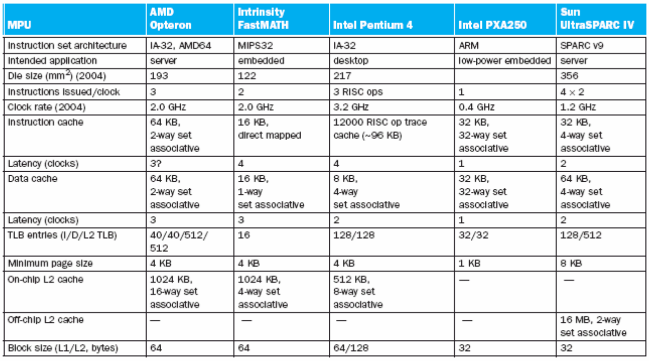


Characteristic	Intel Pentium P4	AMD Opteron
L1 cache organization	Split instruction and data caches	Split instruction and data caches
L1 cache size	8 KB for data, 96 KB trace cache for RISC instructions (12K RISC operations)	64 KB each for instructions/data
L1 cache associativity	4-way set associative	2-way set associative
L1 replacement	Approximated LRU replacement	LRU replacement
L1 block size	64 bytes	64 bytes
L1 write policy	Write-through	Write-back
L2 cache organization	Unified (instruction and data)	Unified (instruction and data)
L2 cache size	512 KB	1024 KB (1 MB)
L2 cache associativity	8-way set associative	16-way set associative
L2 replacement	Approximated LRU replacement	Approximated LRU replacement
L2 block size	128 bytes	64 bytes
L2 write policy	Write-back	Write-back

FIGURE 7.35 First-level and second-level caches in the Intel Pentium P4 and AMD Opteron. The primary caches in the P4 are physically indexed and tagged; for a discussion of the alternatives, see the Elaboration on page 527.

Modern systems

- *Things are getting complicated!*

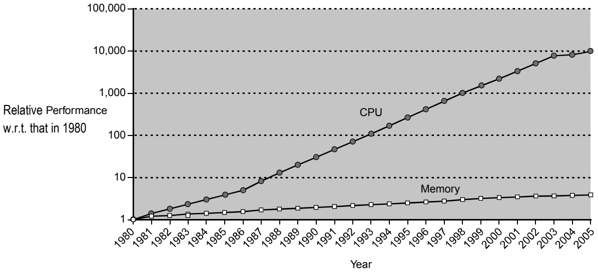


MPU	AMD Opteron	Intrinsity FastMATH	Intel Pentium 4	Intel PXA250	Sun UltraSPARC IV
Instruction set architecture	IA-32, AMD64	MIPS32	IA-32	ARM	SPARC v9
Intended application	server	embedded	desktop	low-power embedded	server
Die size (mm^2) (2004)	193	122	217		356
Instructions issued/clock	3	2	3 RISC ops	1	4 × 2
Clock rate (2004)	2.0 GHz	2.0 GHz	3.2 GHz	0.4 GHz	1.2 GHz
Instruction cache	64 KB, 2-way set associative	16 KB, direct mapped	12000 RISC op trace cache (~96 KB)	32 KB, 32-way set associative	32 KB, 4-way set associative
Latency (clocks)	3?	4	4	1	2
Data cache	64 KB, 2-way set associative	16 KB, 1-way set associative	8 KB, 4-way set associative	32 KB, 32-way set associative	64 KB, 4-way set associative
Latency (clocks)	3	3	2	1	2
TLB entries (I/D/L2 TLB)	40/40/512/ 512	16	128/128	32/32	128/512
Minimum page size	4 KB	4 KB	4 KB	1 KB	8 KB
On-chip L2 cache	1024 KB, 16-way set associative	1024 KB, 4-way set associative	512 KB, 8-way set associative	—	—
Off-chip L2 cache	—	—	—	—	16 MB, 2-way set associative
Block size (L1/L2, bytes)	64	64	64/128	32	32

FIGURE 7.36 Desktop, embedded, and server microprocessors in 2004. From a memory hierarchy perspective, the primary differences between categories is the L2 cache. There is no L2 cache for the low-power embedded, a large on-chip L2 for the embedded and desktop, and 16 MB off chip for the server. The processor clock rates also vary: 0.4 GHz for low-power embedded, 1 GHz or higher for the rest. Note that UltraSPARC IV has two processors on the chip.

Some issues

- *Processor speeds continue to increase very fast*
  — much faster than either DRAM or disk access times



- *Design challenge: dealing with this growing disparity*
  - *Prefetching? 3rd level caches and more? Memory design?*

How important is the cache consideration?

- *Example: CPU run time on a Silicon Graphics Challenge L (MIPS R4000 with 1MB secondary cache)*

77.2 seconds

```
for (i=0; i!=500; i=i+1)
    for (j=0; j!=500; j=j+1)
        for (k=0; k!=500; k=k+1)
            x[i][j] = x[i][j] + y[i][k] * z[k][j];
```

44.2 seconds

```
for (k=0; k!=500; k=k+1)
    for (j=0; j!=500; j=j+1)
        for (i=0; i!=500; i=i+1)
            x[i][j] = x[i][j] + y[i][k] * z[k][j];
```

Key: To re-organize the program to enhance its spatial and temporal locality.

Suggestive exercises

- *7.1, 7.2, 7.3, 7.4, 7.5, 7.6, 7.7, 7.8, 7.9*
- *7.11, 7.12, 7.15, 7.16, 7.17*
- *7.20, 7.21, 7.22, 7.38, 7.45*