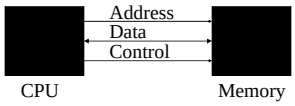
The Main Memory Unit

- CPU and memory unit interface



- CPU issues address (and data for write)
- Memory returns data (or acknowledgment for write)

Memories: Design Objectives

- Provide adequate storage capacity
- Four ways to approach this goal
  - Use of number of different memory devices with different cost/performance ratios
  - Automatic space-allocation methods in hierarchy
  - Development of virtual-memory concept
  - Design of communication links

Memories: Characteristics

- Location: Inside CPU, Outside CPU, External
- Performance: Access time, Cycle time, Transfer rate
- Capacity: Word size, Number of words
- Unit of Transfer: Word, Block
- Access: Sequential, Direct, Random, associative
- Physical Type: Semiconductor, Magnetic, Optical

Memories: Basic Parameters

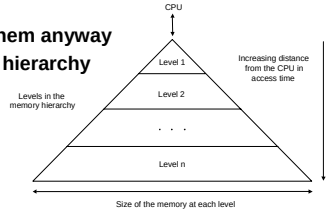
- Cost: c=C/S ($/bit)
- Performance:
  - Read access time (Ta), access rate (1/Ta)
- Access Mode: random access, serial, semi-random
- Alterability:
  - R/W, Read Only, PROM, EPROM, EEPROM
- Storage:
  - Destructive read out, Dynamic, Volatility
- Hierarchy:
  - Tape, Disk, DRUM, CCD, CORE, MOS, BiPOLAR

Exploiting Memory Hierarchy

- Users want large and fast memories!

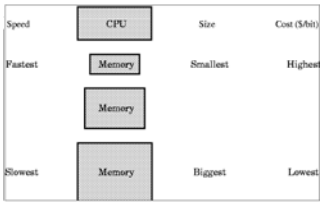
SRAM access times are 1 - 25ns at cost of $100 to $250 per Mbyte.
DRAM access times are 60-120ns at cost of $5 to $10 per Mbyte.
Disk access times are 10 to 20 million ns at cost of $.10 to $.20 per Mbyte.

- Try and give it to them anyway
  - build a memory hierarchy



Advantage of Memory Hierarchy

- Decrease cost/bit
- Increase capacity
- Improve average access time
- Decrease frequency of accesses to slow memory



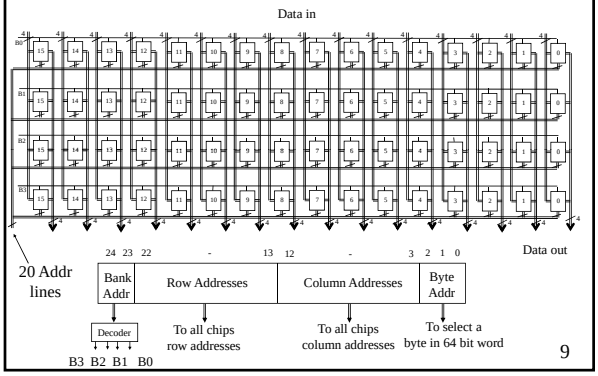
Memories: Review

- SRAM:
  - value is stored on a pair of inverting gates
  - very fast but takes up more space than DRAM
- DRAM:
  - value is stored as a charge on capacitor
  - very small but slower than SRAM (factor of 5/10)

Memories: Array Organization

- Storage cells are organized in a rectangular array
- The address is divided into row and column parts
- There are M (=2^r) rows of N bits each
- The row address (r bits) selects a full row of N bits
- The column address (c bits) selects k bits out of N
- M and N are generally powers of 2
- Total size of a memory chip = M*N bits
  - It is organized as A=2^{r+c} addresses of k-bit words
- To design an R addresses W-bit words memory, we need |R/A| * |W/k| chips

4Mx64-bit Memory using 1Mx4 memory chip :



Locality

- A principle that makes memory hierarchy a good idea
- If an item is referenced
  - temporal locality: it will tend to be referenced again soon
  - spatial locality: nearby items will tend to be referenced soon.
- *Why does code have locality?*
- Our initial focus: two levels (upper, lower)
  - block: minimum unit of data
  - hit: data requested is in the upper level
  - miss: data requested is not in the upper level

Memory Hierarchy and Access Time

- ti is time for access at level i
  - on-chip cache, off-chip cache, main memory, disk, tape
- N accesses
  - ni satisfied at level i
  - a higher level can always satisfy any access that is satisfied by a lower level
  - N = n1 + n2 + n3 + n4 + n5
- Hit Ratio
  - number of accesses satisfied/number of accesses made
  - Could be confusing
  - For example for level 3 is it n3/N or (n1+n2+n3)/N or n3/(N-n1-n2)
  - We will take the second definition

Average Access Time

- ti is time for access at level i
- ni satisfied at level i
- hi is hit ratio at level i
  - hi = (n1 + n2 + …+ ni) /N
- We will also assume that data are transferred from level i+1 to level i before satisfying the request
- Total time = n1*t1 + n2*(t1+t2) + n3*(t1+t2+t3) + n4*(t1+t2+t3+t4) + n5*(t1+t2+t3+t4+t5)
- Average time = Total time/N
- t(avr) = t1+t2*(l-h1)+t3*(1-h2)+t4*(1-h3)+t5*(1-h4)
- Total Cost = C1*S1+C2*S2+C3*S3+C4*S4+C5*S5

Cache

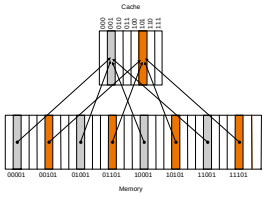
- Two issues:
  - How do we know if a data item is in the cache?
  - If it is, how do we find it?
- Our first example:
  - block size is one word of data
  - "direct mapped"

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

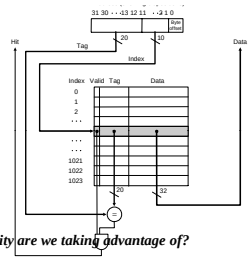
Direct Mapped Cache

- Mapping:
  - address is modulo the number of blocks in the cache



Direct Mapped Cache

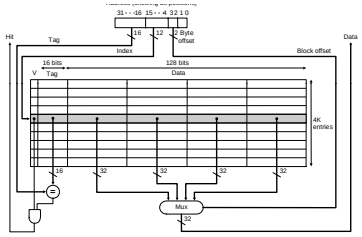
- For MIPS:



What kind of locality are we taking advantage of?

Direct Mapped Cache

- Taking advantage of spatial locality:

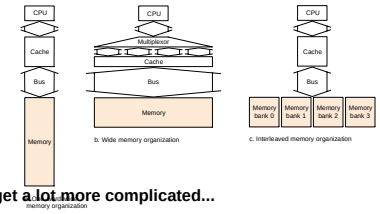


Hits vs. Misses

- 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

Hardware Issues

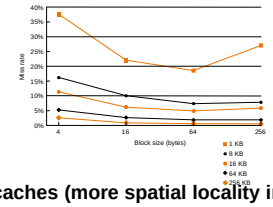
- Make reading multiple words easier by using banks



- It can get a lot more complicated...

Performance

- **Increase in block size tend to decrease miss rate:**



- **Use split caches (more spatial locality in code)**

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

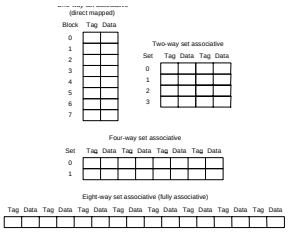
- **Simplified model:**

  execution time=(execution cycles + stall cycles)∗cct
- **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?

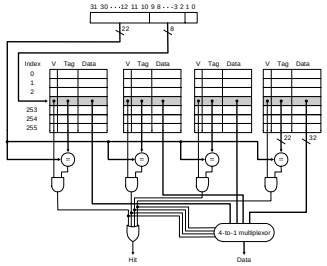
Decreasing miss ratio with associativity



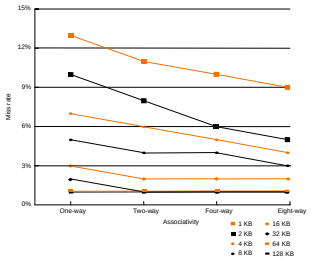
Compared to direct mapped, give a series of references that:
- ***results in a lower miss ratio using a 2-way set associative cache***
- ***results in a higher miss ratio using a 2-way set associative cache***

assuming we use the "least recently used" replacement strategy

An implementation



Performance



Decreasing miss penalty with multilevel caches

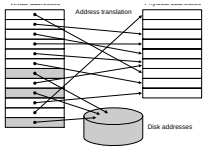
- **Add a second level cache:**
  - **often primary cache is on the same chip as the processor**
  - **use SRAMs to add another cache above primary memory (DRAM)**
  - **miss penalty goes down if data is in 2nd level cache**

- **Example:**
  - **CPI of 1.0 on a 500Mhz machine with a 5% miss rate, 200ns DRAM access**
  - **Adding 2nd level cache with 20ns access time decreases miss rate to 2%**

- **Using multilevel caches:**
  - **try and optimize the hit time on the 1st level cache**
  - **try and optimize the miss rate on the 2nd level cache**

Virtual Memory

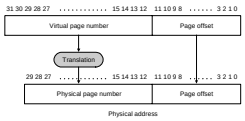
- Main memory can act as a cache for the secondary storage (disk)



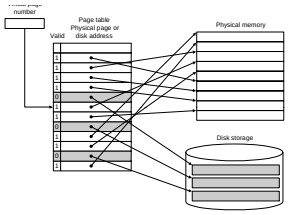
- Advantages:
  - illusion of having more physical memory
  - program relocation
  - protection

Pages: virtual memory blocks

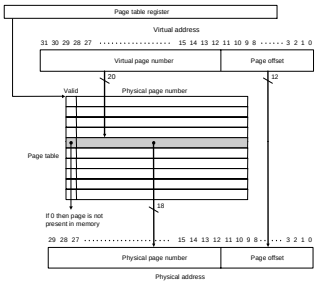
- Page faults: the data is not in memory, retrieve it from disk
  - huge miss penalty, thus pages should be fairly large (e.g., 4KB)
  - reducing page faults is important (LRU is worth the price)
  - can handle the faults in software instead of hardware
  - using write-through is too expensive so we use writeback



Page Tables

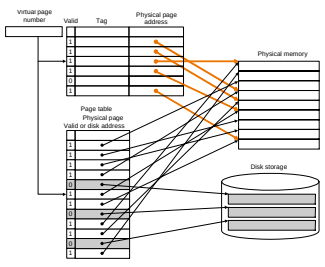


Page Tables

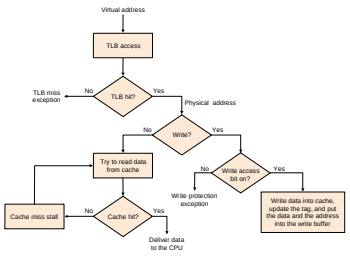


Making Address Translation Fast

- A cache for address translations: translation lookaside buffer



TLBs and Caches



Replacement Policies

- Replacement Policies in Multi-way Set Associative caches
  - Random: Replace any line arbitrarily
  - Least Recently Used (LRU): Find the least recently used line to replace
  - Keep Most Recently Used (MRU): Keep the last used line in the set and replace any other randomly
- LRU performs the best
- MRU does equally well

LRU Scheme

- We explain LRU with an example of a 4-way set associative cache
- Associate a 2-bit counter with each line (log k bit for k-way cache)
- Initially all lines are invalid
- For a miss bring a new line in an invalid line, make it valid, set its counter to zero, increment all other counters
  - If no invalid line, replace the line with counter value = 3, set its counter to zero, increment all other counters
- For a hit, set the accessed line's counter to zero and increment counters of those lines whose values is smaller than the accessed line
- Try this algorithm for an examples where lines read are 0, 64, 128, 64, 192, 256, 128, 0, 256, 192, 64…
  - There are 64 lines in each cache and it is 4-way set associative

Reading or Writing a Memory word

- Check the address in TLB
- If not there, get the physical translation and also store the entry in TLB
  - Penalty 40-50 cycles
- If page itself is not present, page fault occurs
  - Read the page, update page table and TLB
  - Penalty 100's of thousands cycles
- Once physical address is there If there, perform read or write in cache
- If cache miss
  - Read the line in cache for read
  - May need to replace a dirty or clean line
    - Penalty 20-40 cycles
  - For Write read the line if write allocate, else write around
- If cache hit read or write in cache
  - Also write in main memory if write through

A Big Example

- Instruction Frequency: LW(20%), SW(10%), R(50%), BR(15%), J(5%)
- Branch Penalty: 3 cycles on 20% mis-predictions = 15*0.20*3 = 9 cycles
- Data Cache 1: Miss rate 10% (of load/store), write back, write around, 50% dirty replacement, penalty for reading or writing a line 20 cycles
  - Load penalty = 20*0.10*0.50*20 + 20*0.10*0.50*(20+20) = 60 cycles
  - Store Penalty = 0 (because of write around, otherwise will be 30)
- Data Cache 2: Miss rate 5% (of load/store), write back, write allocation, 50% dirty replacement, penalty for reading or writing a line 100 cycles
  - Load penalty = 20*0.05*0.5*100 + 20*0.05*0.5*(100+100) = 150 cycles
  - Store Penalty = 10*0.05*0.5*100 + 10*0.05*0.5*(100+100) = 75 cycles
- TLB: Miss Rate 2% (of load/store), Miss Penalty 100 cycles
  - Total Penalty = (20+10)*0.02*100 = 60 cycles
- Page faults: 0.01% (of load/store), Penalty 300,000 cycles
  - Total Penalty = (20+10)*0.0001*300,000 = 900 cycles
- Total Time = 100+9+60+150+75+60+900 = 1354 cycles, or CPI=13.54
- Notice that miss rates can be spacified per instruction or per load/store

Misses and Replacement Policies

- 3 C Misses
  - Compulsory: Miss will have to occur on first read (or write)
  - Capacity: A line is replaced and then brought back
  - Conflict: a miss occurs as some other line is occupying that line
- Example Suppose we read line a first time (no line is in cache), then read line b that replaces line a, and then read line a again
- The first and second misses are compulsory, second miss is also capacity and conflict, and the third miss is capacity (and also conflict)
- The terminology can be confusing here
  - The first read is always classified as compulsory
  - The replacement and read back is conflict if there was place in cache elsewhere but you had to bring it at that place due to mapping
  - If there was no place at all then it is capacity miss (like cache is full in a fully associative cache)

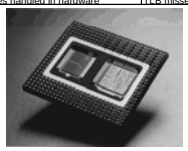
Virtual Memory: Other Translation Schemes

- In a single-level translation
  - 32 bit virtual address
  - 4KB Page size (12 bit address in each page)
  - Leaves 20-bit page address => 1 Million Pages =>4MB for Table
- One alternate is to only have a limited size page table with Hi and Lo Checks
  - But program use many addresses segments
- Alternate is to have a two level page table
- Divide page addresses in two parts of 10 bits each
  - There are 1K tables of 1K entries each (total is still 1M entries)
  - Most significant 10 bits points to a table (with 1K entries, each 4 bytes long, a total of 4KB that fits in a page) that contains the address of that part of table
  - Least significant 10 bits are used to access a particular entry in the selected table
- We only need to keep the first table (pointing to real tables) and some of the second level tables in memory

Modern Systems

- **Very complicated memory systems:**

Characteristic	Intel Pentium Pro	PowerPC 604
Virtual address	32 bits	52 bits
Physical address	32 bits	32 bits
Page size	4 KB, 4 MB	4 KB, selectable, and 256 MB
TLB organization	A TLB for instructions and a TLB for data Both four-way set associative Pseudo-LRU replacement Instruction TLB: 32 entries Data TLB: 64 entries TLB misses handled in hardware	A TLB for instructions and a TLB for data Both two-way set associative LRU replacement Instruction TLB: 128 entries Data TLB: 128 entries TLB misses handled in hardware



Characteristic	Intel Pentium Pro	PowerPC 604
Cache organization	Split instruction and data caches	Split intruction and data caches
Cache size	8 KB each for instructions/data	16 KB each for instructions/data
Cache associativity	Four-way set associative	Four-way set associative
Replacement	Approximated LRU replacement	LRU replacement
Block size	32 bytes	32 bytes
Write policy	Write-back	Write-back or write-through

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**
- **Trends:**
  - **synchronous SRAMs (provide a burst of data)**
  - **redesign DRAM chips to provide higher bandwidth or processing**
  - **restructure code to increase locality**
  - **use prefetching (make cache visible to ISA)**