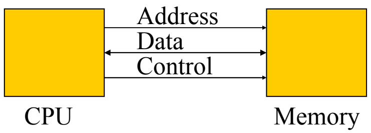
The Main Memory Unit

- 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

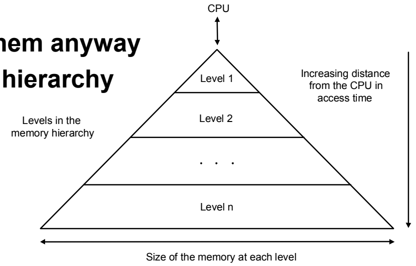
- Cost: c=C/S ($/bit)
- Performance:
  - Read access time (Ta), access rate (1/Ta)
- Access Mode: random access, serial, semi-random
- Alterability:
  - R/W, Read Only, PROM, EPROM, EEPROM
- Storage:
  - Destructive read out, Dynamic, Volatility
- Hierarchy:
  - Tape, Disk, DRUM, CCD, CORE, MOS, BiPOLAR

Exploiting Memory Hierarchy

- **Users want large and fast memories!**

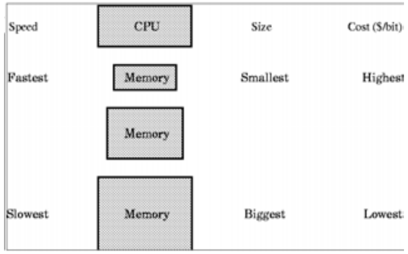
SRAM access times are 1 - 25ns at cost of $100 to $250 per Mbyte.
DRAM access times are 60-120ns at cost of $5 to $10 per Mbyte.
Disk access times are 10 to 20 million ns at cost of $.10 to $.20 per Mbyte.

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


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

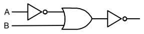
Advantage of Memory Hierarchy

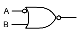
- **Decrease cost/bit**
- **Increase capacity**
- **Improve average access time**
- **Decrease frequency of accesses to slow memory**

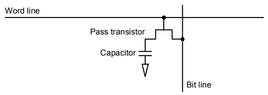


Memories: Review

- **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)**



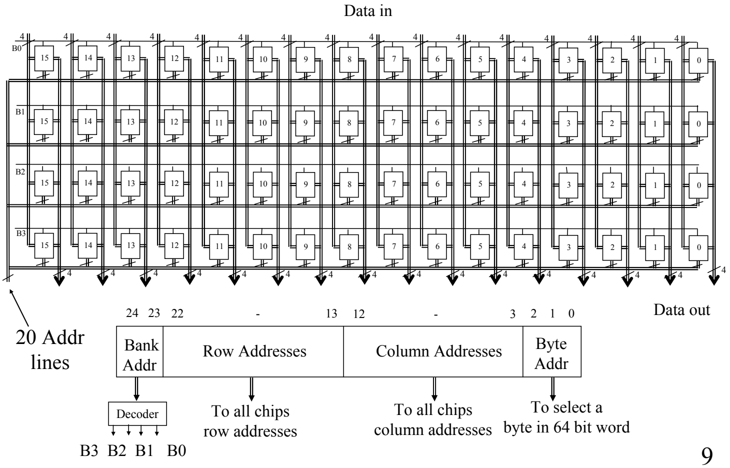




Memories: Array Organization

- **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

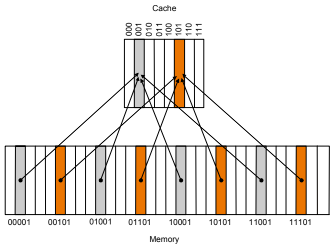
- Two issues:
  - How do we know if a data item is in the cache?
  - If it is, how do we find it?
- Our first example:
  - block size is one word of data
  - "direct mapped"

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

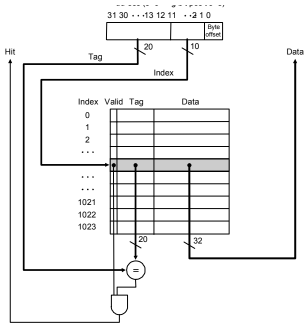
Direct Mapped Cache

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



Direct Mapped Cache

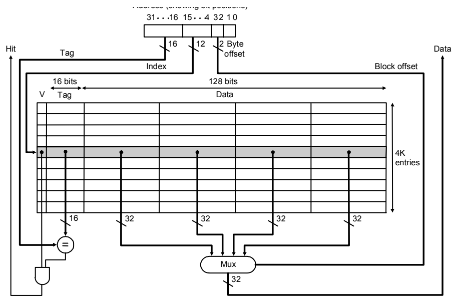
- For MIPS:



What kind of locality are we taking advantage of?

Direct Mapped Cache

- Taking advantage of spatial locality:

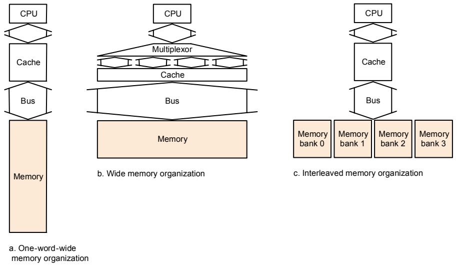


Hits vs. Misses

- **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**



a. One-word-wide memory organization

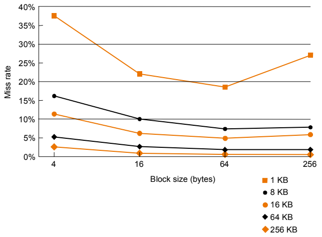
b. Wide memory organization

c. Interleaved memory organization

- **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

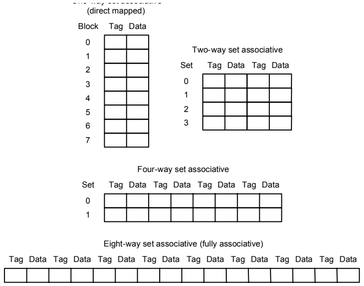
- **Simplified model:**

  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

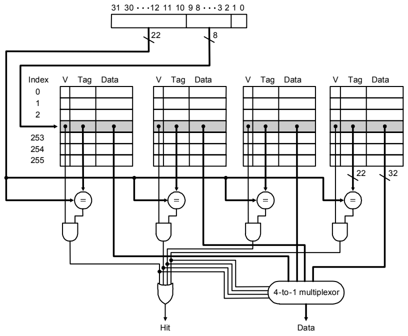

(direct mapped)
Block Tag Data
0
1
2
3
4
5
6
7
Two-way set associative
Set Tag Data Tag Data
0
1
2
3
Four-way set associative
Set Tag Data Tag Data Tag Data Tag Data
0
1
Eight-way set associative (fully associative)
Tag Data Tag Data Tag Data Tag Data Tag Data Tag Data Tag Data Tag Data

Compared to direct mapped, give a series of references that:

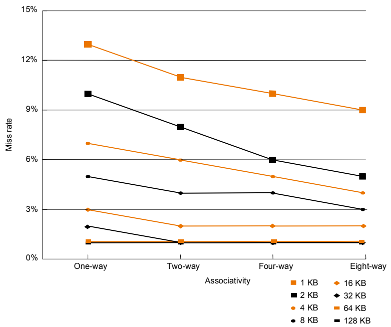
- *results in a lower miss ratio using a 2-way set associative cache*
- *results in a higher miss ratio using a 2-way set associative cache*

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**