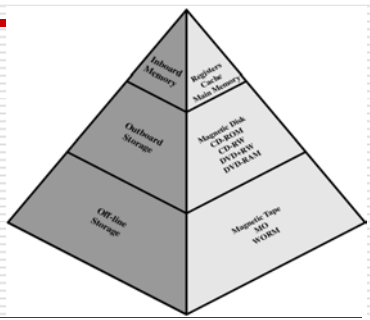
Memory Hierarchy

Contents

- Memory System Overview
- Cache Memory
- Internal Memory
- External Memory
- Virtual Memory

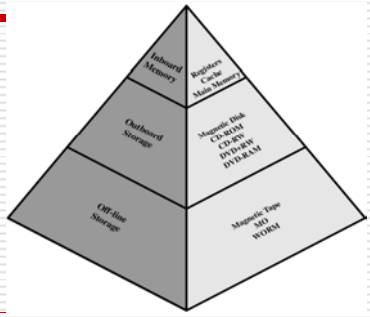
Memory Hierarchy

- Registers
  - In CPU
- Internal or Main memory
  - Cache
  - "RAM"
- External memory
  - Backing store

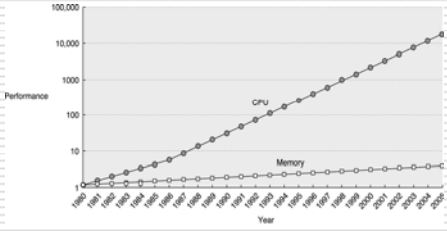


Memory Hierarchy

- Cost/bit
- Capacity
- Access time
- Frequency of access



Why memory hierarchy



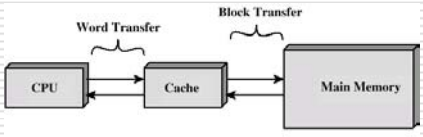
Memory-Processor Performance Gap

Why memory hierarchy

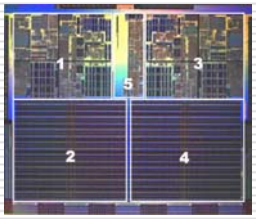
- Locality
  - Temporal locality
    - The currently required data are likely to need again in the near future
  - Spatial locality
    - There is high probability that the other data nearby will be need soon

Cache Memory

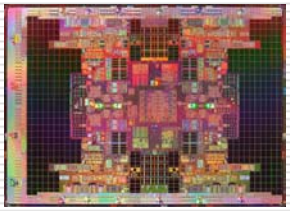
- Small amount of fast memory
- Sits between normal main memory and CPU
- May be located on CPU chip or module



Cache in processor

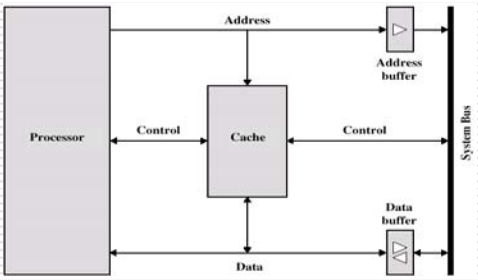


AMD Athlon 64 X2 (Dual Core)



Intel Tukwila (Four Core)

Typical Cache Organization



Cache operation

- CPU requests contents of memory location
- Check cache for this data
- If present, get from cache (fast)
- If not present, read required block from main memory to cache
- Then deliver from cache to CPU

Avg memory access time = Hit time$_{L1}$ + Miss rate$_{L1}$ x Miss penalty$_{L1}$

Miss Penalty$_{L1}$ = Hit time $_{L2}$ + Miss rate $_{L2\,x}$ Miss penalty $_{L2}$

Example

- Two levels of memory
- Level 1 access time 0.01 us, Level 2 access time 0.1us
- 95% memory accesses are found in level 1
- Average memory access time for each word?

Cache Design

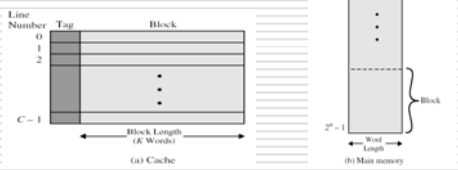
- Size
- Mapping Function
- Replacement Algorithm
- Write Policy
- Block Size
- Number of Caches

Size does matter

- Cost
  - More cache is expensive
- Speed
  - More cache is faster (up to a point)
  - Checking cache for data takes time

Mapping

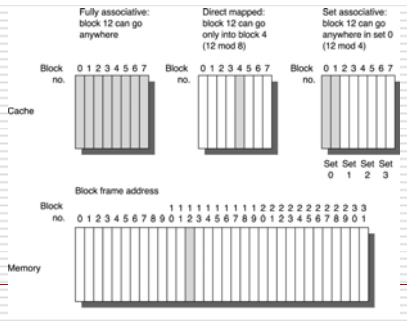
- Memory blocks
- Cache Lines (blocks)
- Cache Tag



Mapping

- Direct
- Associative
- Set associative

Mapping Example



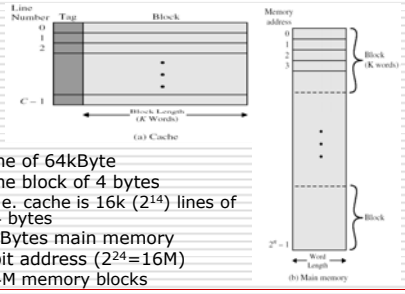
Address structure

- Address is in two parts (**s+w**)
- Most Significant **s** bits specify one memory block
  - *Block address*: which block should be read or written
  - The **s** bits are split into a cache line field **r** and a tag of **s-r** (most significant)
- Least Significant **w** bits identify unique byte
  - *Block shift:* within the block, which byte should be read or written

Direct Mapping

- Each block of main memory maps to only one cache line
  - i.e. if a block is in cache, it must be in one specific place
- How
  - i=j modulo m
    - i: cache line number
    - j: memory block number
    - m: number of total lines in the cache

Example



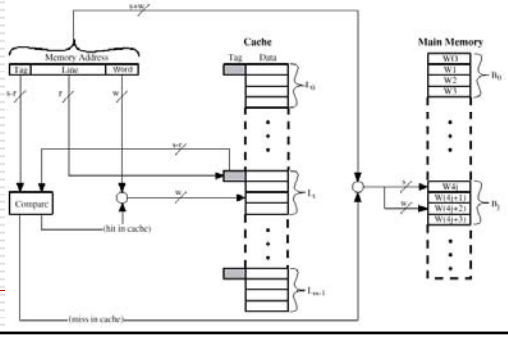
- Cache of 64kByte
- Cache block of 4 bytes
  - i.e. cache is 16k (2^{14}) lines of 4 bytes
- 16MBytes main memory
- 24 bit address (2^{24}=16M)
  - 4M memory blocks

Ex:Direct Mapping Address Structure

Tag s-r	Line of block index r	Word w
8	14	2

- 24 bit address
- 2 bit word identifier (4 byte block)
- Total cache lines: 2^{14} (or the number of blocks in caches)
- 22 bit block identifier
  - 8 bit tag (=22-14)

Direct Mapping Cache Organization



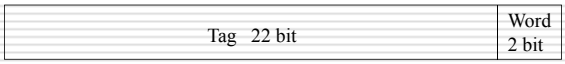
Direct Mapping pros & cons

- Simple and fast
- Inexpensive
- Fixed location for given block
  - If a program accesses 2 blocks that map to the same line repeatedly, cache misses are very high

Fully Associative Mapping

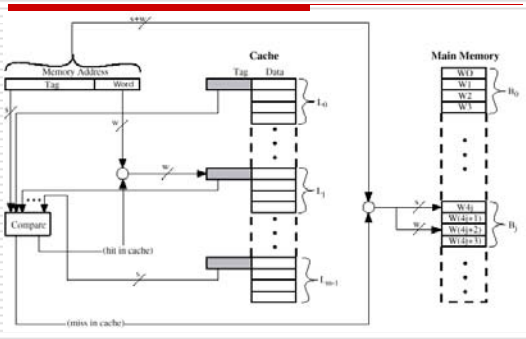
- A main memory block can load into any line of cache
- Memory address is interpreted as tag and word
- Tag uniquely identifies block of memory
- Every line's tag is examined for a match
- Flexibility in placing data blocks
- Cache searching gets expensive

Fully Associative Address Structure



- 22 bit tag stored with each 4 bytes of data
- Compare tag field with tag entry in cache to check for hit
- Least significant 2 bits of address identify which byte is required from the 4 byte block

Fully Associative Cache Organization



Set Associative Mapping

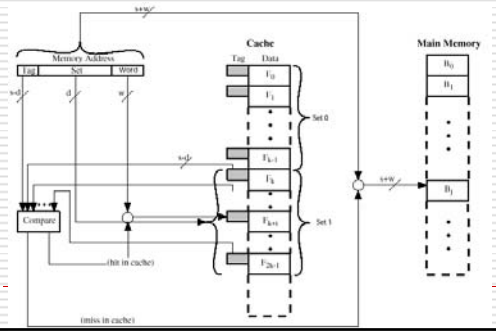
- Cache lines is divided into a number of groups, i.e., ***sets***
- A given block maps to only one set, but can be any line in the set
- How
  - i=j modulo v
    - v: number of total **sets** in the cache
- Trade off between direct mapping and fully associative mapping

Ex: 8-way Set Associative Mapping Address Structure

Tag s-r	Set Index (r)	Word w
8	14	2

- 24 bit address
- 2 bit word identifier (4 byte block)
- 22 bit block identifier
- Total number of sets: $2^{14}/8 = 2^{11}$
  - 11 bit tag (=22-11)
  - 11 bit set index

K-Way Set Associative Cache Organization



Replacement Algorithms

- For Associative & Set Associative
  - Least Recently used (LRU)
  - First in first out (FIFO)
  - Least frequently used
  - Random

Write Policy

- Why
  - Must not overwrite a cache block unless main memory is up to date
  - Multiple CPUs may have individual caches
  - I/O may address main memory directly
- How
  - Write through
  - Write back

Write through

- All writes go to main memory as well as cache
  - Multiple CPUs can monitor main memory traffic to keep local (to CPU) cache up to date
  - Lots of traffic
  - Slows down writes

Write back

- Updates initially made in cache only
- If block is to be replaced, write to main memory only if updated
  - Other caches get out of sync
  - I/O must access main memory through cache

Line Size

- Larger blocks reduce the number of blocks that fit into a cache.
- Larger blocks make each additional word is less likely to be needed in the near future.

Number of Caches

- Single or two level
  - Logic density increased
  - On-chip cache reduces the processor's external bus activity
  - Off-chip/external cache is still desirable
- Unified or split

Contents

- Memory System Overview
- Cache Memory
- Internal Memory
- External Memory
- Virtual Memory

Internal Memory

- Internal memory types
- DRAM vs. SRAM

Semiconductor Memory Types

Memory Type	Category	Erasure	Write Mechanism	Volatility
Random-access memory (RAM)	Read-write memory	Electrically, byte-level	Electrically	Volatile
Read-only memory (ROM)	Read-only memory	Not possible	Masks	Nonvolatile
Programmable ROM (PROM)			Electrically	
Erasable PROM (EPROM)	Read-mostly memory	UV light, chip-level		
Electrically Erasable PROM (EEPROM)		Electrically, byte-level		
Flash memory		Electrically, block-level		

Dynamic RAM

- Bits stored as charge in capacitors
- Charges leak -> Need refreshing (and therefore refresh circuits) even when powered
- Simpler construction
- Smaller per bit
- Less expensive
- Slower
- Main memory

Static RAM

- Bits stored as on/off switches
- No charges to leak and therefore no refreshing needed when powered
- More complex construction
- Larger per bit
- More expensive
- Faster
- Cache

SRAM vs DRAM

- Both volatile
  - Power needed to preserve data
- Dynamic cell
  - Simpler to build, smaller, more dense
  - Less expensive
  - Needs refresh
  - Larger memory units
- Static
  - Faster
  - Cache

Read Only Memory (ROM)

- Permanent storage
  - Nonvolatile
- Usage
  - Library subroutines
  - Systems programs (BIOS)
  - Function tables

Types of ROM

- Written during manufacture
- Programmable (once)
  - PROM
  - Needs special equipment to program
- Read "mostly"
  - Erasable Programmable (EPROM)
    - Erased by UV
  - Electrically Erasable (EEPROM)
    - Takes much longer to write than read
  - Flash memory

Contents

- Memory System Overview
- Cache Memory
- Internal Memory
- External Memory
- Virtual Memory

Contents

- Memory System Overview
- Cache Memory
- Internal Memory
- External Memory
- Virtual Memory

External Memory

- Types of external memory

Types of External Memory

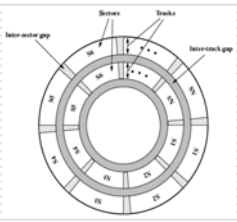
- Magnetic Disk
  - RAID (Redundant Array of Independent Disk)
  - Removable
- Optical
  - CD-ROM
  - CD-Recordable (CD-R)
  - CD-R/W
  - DVD
- Magnetic Tape

Magnetic Disk

- Read and Write Mechanisms
  - Recording and retrieval via conductive coil called a head
- May be single read/write head or separate ones
- During read/write, head is stationary, platter rotates
- Write
  - Current through coil produces magnetic field
  - Pulses sent to head
  - Magnetic pattern recorded on surface below

Data Organization and Formatting

- Concentric rings or tracks
  - Gaps between tracks
  - Reduce gap to increase capacity
  - Same number of bits per track (variable packing density)
- Tracks divided into sectors



Performance

- Seek time
  - Moving head to correct track (t_s)
- Average (Rotational) latency
  - Waiting for data to rotate under head
  - t_r= 1/2r (r: rounds per second)
- Transfer time: t_t=b/(rN)
  - b: number of bytes to be transferred
  - r: rounds per second
  - N: number of bytes per track
- Total disk access time = $t_s+t_r+t_t$

Example

- Seek time = 4ms
- Rotation speed = 7500rpm
- 512 bytes/sector, 500 sectors/track
- A file with total 2500 sectors
- Assume:
  - Sequentially stored
  - Randomly stored

1. Ts= 4ms, Max Tr = 60/7500=8ms
   2500 sectors → 5 tracks Tt for each track = b/rN =8ms
   Total disk access time = (4 + 8 + 8) + 8 * 4 = 52 ms
   (assume no seek and rotation time except for the first track)
2. Ts = 4ms, Avg Tr= 4ms,
   Tt for each sector =
   (1x 512) * 60 /(7500 * (500 x 512)) = 8/500 ms
   Total disk access time = 2500 x (4 + 4 + 8/500) = 20.04 sec

Contents

Virtual Memory

- Virtual Memory
  - A technique of using secondary storage such as disks to extend the size limit of physical memory

Memory Paging

- Split memory into equal sized, small chunks - page frames
- Allocate the required number page frames to a process
- Operating System maintains list of free frames
- A process does not require contiguous page frames
- Use page table to keep track

Virtual Memory

- Demand paging
  - Do not require all pages of a process in memory
  - Bring in pages as required
- Page fault
  - Required page is not in memory
  - Operating System must swap in required page
  - May need to swap out a page to make space
  - Select page to throw out based on recent history

Bonus

- We do not need all of a process in memory for it to run
- We can swap in pages as required
- So - we can now run processes that are bigger than total memory available!

- Main memory is called real memory
- User/programmer sees much bigger memory - virtual memory

Thrashing

- Too many processes in too little memory
- Operating System spends all its time swapping
- Little or no real work is done
- Disk light is on all the time

- Solutions
  - Good page replacement algorithms
  - Reduce number of processes running
  - Fit more memory

Translation Lookaside Buffer (TLB)

- Two physical memory accesses/virtual memory reference
  - Page table
  - Desired data
- TLB
  - A special cache for page table entries

TLB Operation