Computer Organization & Architecture
Lecture #16

Cache Memory

Computer memory is organized into a hierarchy. At the highest level (closest to the processor) are the processor registers. Next comes one or more levels of cache. When multiple levels are used, they are denoted L1, L2, etc… Next comes main memory, which is usually made out of a dynamic random-access memory (DRAM). All of these are considered internal to the computer system. The hierarchy continues with external memory, with the next level typically being a fixed hard disk, and one or more levels below that consisting of removable media such as ZIP cartridges, optical disks, and tape.

As one goes down the memory hierarchy, one finds decreasing cost/bit, increasing capacity, and slower access time. It would be nice to use only the fastest memory, but because that is the most expensive memory, we trade off access time and cost by using more of the slower memory. The trick is to organize the data and programs in memory so that the memory words needed are usually in the fastest memory.

In general, it is likely that most future accesses to main memory by the processor will be to locations recently accesses. So the cache automatically retains a copy of some of the recently used words from the DRAM. If the cache is designed properly, then most of the time the processor will request memory words that are already in the cache.

Computer Memory System Overview

Characteristics of Memory Systems

Location
- Processor
- Internal – main memory
- External – secondary memory

Capacity
- Word size – natural unit or organization
- Number of words – number of bytes

Unit of Transfer

- Internal
  - Usually governed by bus width
- External
  - Usually a block which is much larger than a word
- Addressable unit
  - Smallest location which can be uniquely addressed
  - Cluster on external disk

Access Methods

- Sequential – tape
  - Start at the beginning and read through in order
  - Access time depends on location of data and previous location
- Direct – disk
  - Individual blocks have unique address
  - Access is by jumping to vicinity plus sequential search
  - Access time depends on location of data and previous location
- Random - RAM
  - Individual addresses identify location exactly
  - Access time is independent of data location and previous location
- Associative – cache
  - Data is located by a comparison with contents of a portion of the store
  - Access time is independent of data location and previous location

Performance

- Access time (latency)
  - The time between presenting an address and getting access to valid data
- Memory Cycle time – primarily random-access memory
  - Time may be required for the memory to “recover” before the next access
  - Access time plus recovery time
- Transfer rate
  - The rate at which data can be transferred into or out of a memory unit

Physical Types

- Semiconductor – RAM
- Magnetic – disk and tape
- Optical – CD and DVD
- Magneto-optical

Physical Characteristics

- Volatile/non-volatile
- Erasable/non-erasable
- Power requirements

Organization

- The physical arrangement of bits to form words
- The obvious arrangement is not always used

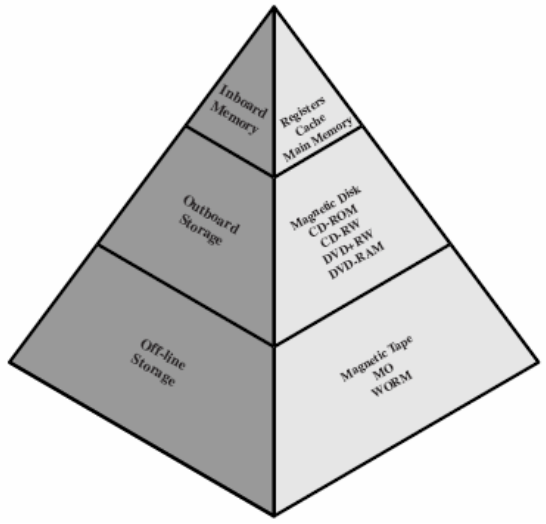
The Memory Hierarchy

- How much?
  - If the capacity is there, applications will be developed to use it.
- How fast?
  - To achieve performance, the memory must be able to keep up with the processor.
- How expensive?
  - For a practical system, the cost of memory must be reasonable in relationship to other components

There is a trade-off among the three key characteristics of memory: cost, capacity, and access time.

- Faster access time – greater cost per bit
- Greater capacity – smaller cost per bit
- Greater capacity – slower access time

The way out of this dilemma is not to rely on a single memory component or technology. Employ a memory hierarchy.



As one goes down the hierarchy: (a) decreasing cost per bit; (b) increasing capacity; (c) increasing access time; (d) decreasing frequency of access of the memory by the processor.

Thus smaller, more expensive, faster memories are supplemented by larger, cheaper, slower memories. The key to the success of this organization is item (d).

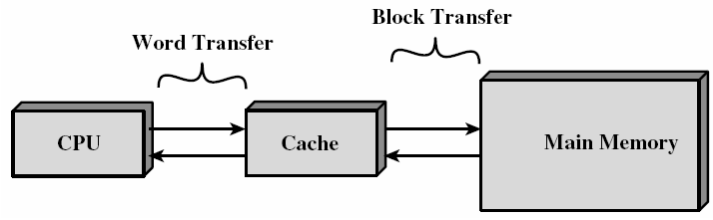
Locality of Reference principle

- Memory references by the processor, for both data and instructions, cluster
- Programs contain iterative loops and subroutines - once a loop or subroutine is entered, there are repeated references to a small set of instructions
- Operations on tables and arrays involve access to a clustered set of data word

Cache Memory Principles

Cache memory

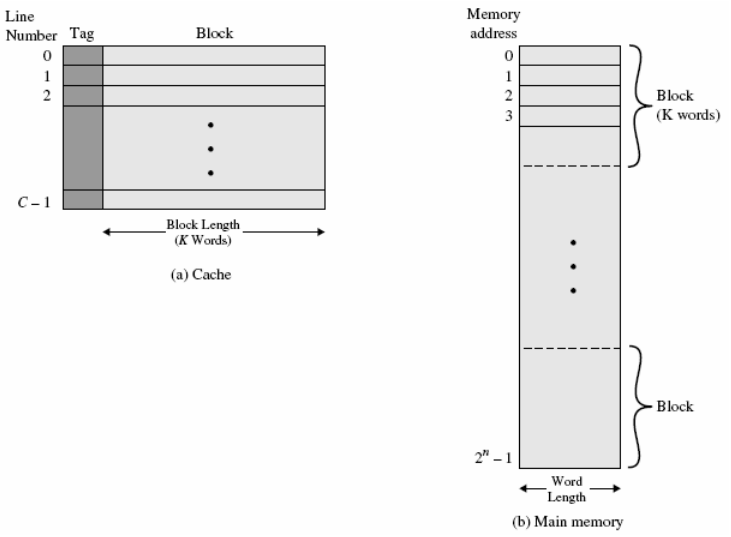
- Small amount of fast memory
- Placed between the processor and main memory
- Located either on the processor chip or on a separate module



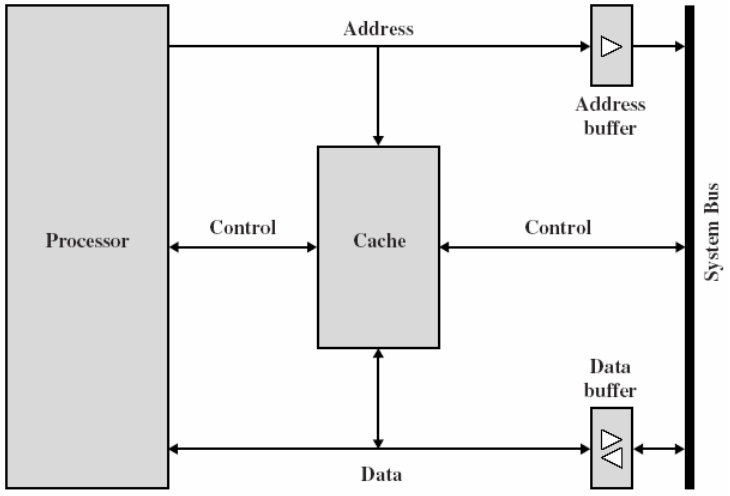
Cache Operation Overview

- Processor requests the contents of some memory location
- The cache is checked for the requested data
    - If found, the requested word is delivered to the processor
    - If not found, a block of main memory is first read into the cache, then the requested word is delivered to the processor

When a block of data is fetched into the cache to satisfy a single memory reference, it is likely that there will be future references to that same memory location or to other words in the block – locality or reference principle. Each block has a tag added to identify it.



An example of a typical cache organization is shown below:



Elements of Cache Design

Cache Size

- Small enough so overall cost/bit is close to that of main memory
- Large enough so overall average access time is close to that of the cache alone
    - Access time = main memory access time plus cache access time
- Large caches tend to be slightly slower than small caches

Mapping Function

An algorithm is needed to map main memory blocks into cache lines. A method is needed to determine which main memory block occupies a cache line. Three techniques used: direct, associative, and set associative. Assume the following:

- Cache of 64 Kbytes
- Transfers between main memory and cache are in blocks of 4 bytes each – cache organized as 16K = 2^14 lines of 4 bytes each
- Main memory of 16 Mbytes, directly addressable by a 24-bit address (where 2^24 = 16M) – main memory consists of 4M blocks of 4 bytes each

Direct Mapping

- Each block of main memory maps to only one cache line
    - "cache line #" = "main memory block #" % "number of lines in cache"
- Main memory addresses are viewed as three fields
    - Least significant w bits identify a unique word or byte within a block
    - Most significant s bits specify one of the 2^s blocks of main memory
        - Tag field of s-r bits (most significant)
        - Line field of r bits – identifies one of the m = 2^r lines of the cache

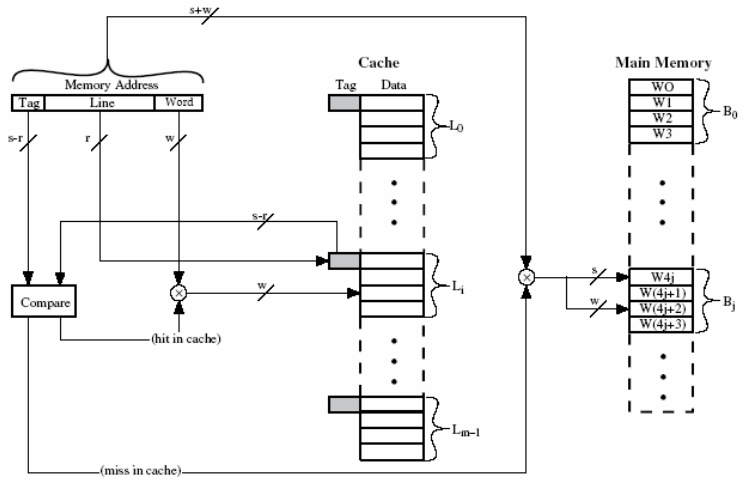
Tag (s-r)	Line or Slot (r)	Word (w)
8 bits	14 bits	2 bits

- 24 bit address
- 2 bit word identifier
- 22 bit block identifier
- 8 bit tag (22-14)
- 14 bit slot or line
- No two blocks in the same line have the same Tag field
- Check contents of cache by finding and checking Tag

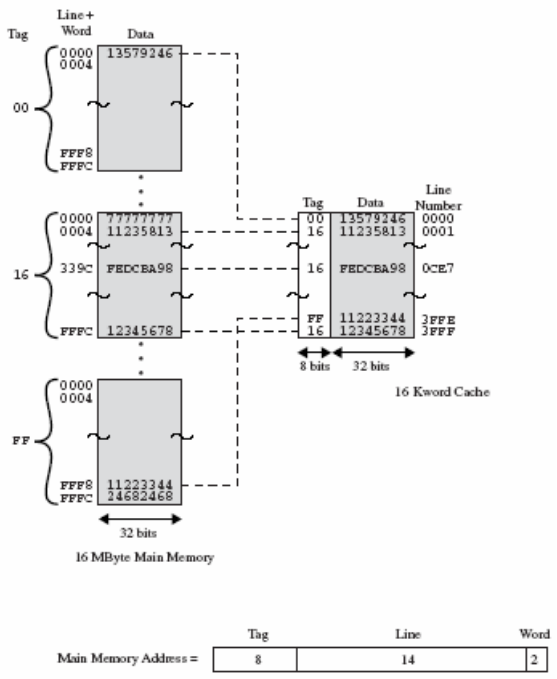
Direct Mapping Cache Table

Cache Line	Main memory blocks assigned
0	0, m, 2m, …, 2^s-m
1	1, m+1, 2m+1, …, 2^s-m+1
…	…
m-1	m-1, 2m-1, 3m-1, …, 2^s-1

Direct Mapping Cache Organization



Example of Direct Mapping



Direct Mapping Summary

- Address length = (s+w) bits
- Number of addressable units = 2^(s+w) words or bytes
- Block size = line size = 2^w words or bytes
- Number of blocks in main memory = 2^(s+w)/2^w = 2^s
- Number of lines in cache = m = 2^r
- Size of tag = (s-r) bits

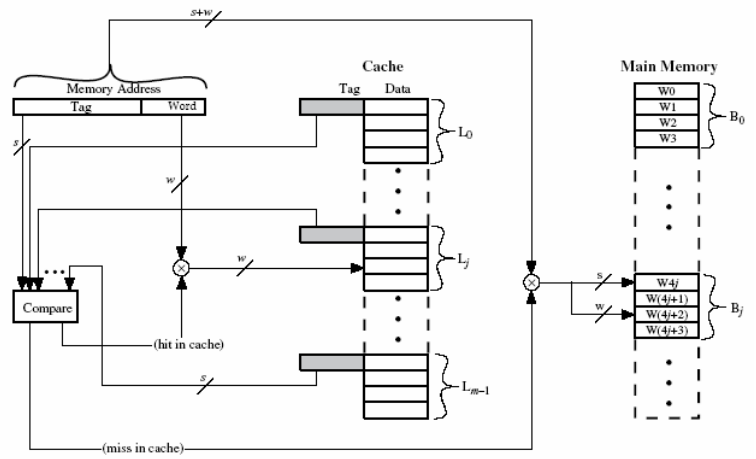
Direct Mapping Pros and Cons

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

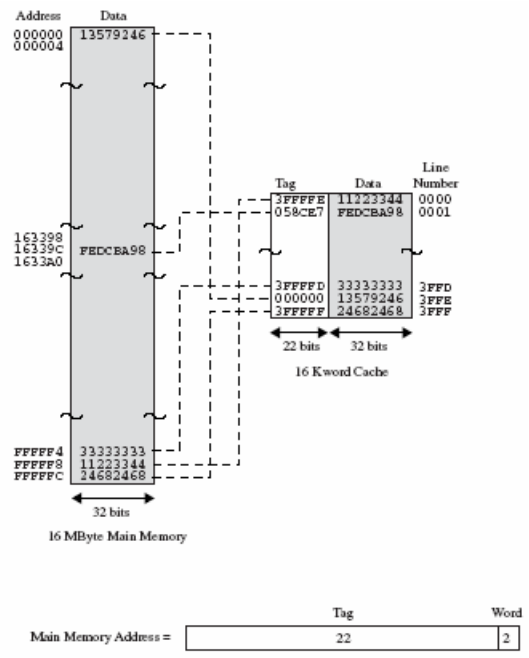
Associative Mapping

- A main memory block can be loaded into any line of the cache
- A memory address is interpreted as a tag and a word field
- The tag field uniquely identifies a block of main memory
- Each cache line's tag is examined simultaneously to determine if a block is in cache

Associative Mapping Cache Organization:



Example of Associative Mapping



- Main memory addresses are viewed as two fields

Tag (s)	**Word (w)**
22 bits	2 bits

- 22 bit tag stored with the 32 bit block of data
- Tag field compared with tag entry to check for cache hit
- 2 bit byte number

Example:
Hex Address: 16339C Binary Address: 0001 0110 0011 0011 1001 1100
Hex Tag: 058CE7 Binary Tag: 0000 0101 1000 1100 1110 0111

Associative Mapping Summary
- Address length = (s+w) bits
- Number of addressable units = 2^(s+w) words or bytes
- Block Size = line size = 2^w words or bytes
- Number of blocks in main memory = 2^(s+w)/2^w = 2^s
- Number of lines in cache = undetermined
- Size of tag = s bits

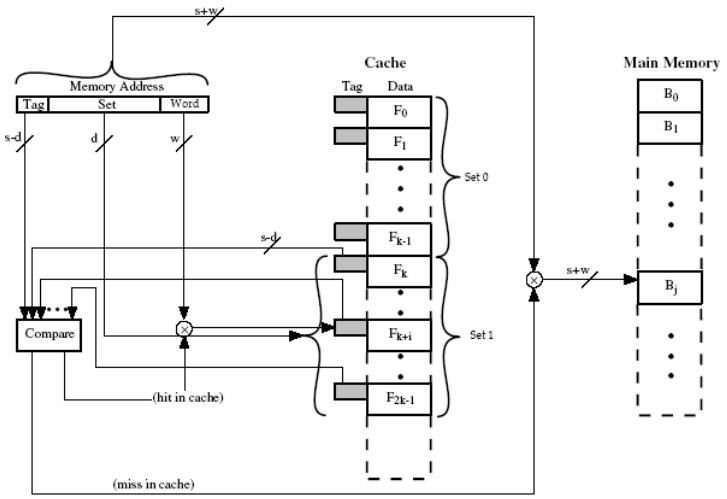
Associative Mapping Pros and Cons

- Flexibility as to which block to replace when a new block is read into cache
    - Replacement algorithms designed to maximize cache hit ratio
- Complex circuitry required to examine the tags of all cache lines in parallel

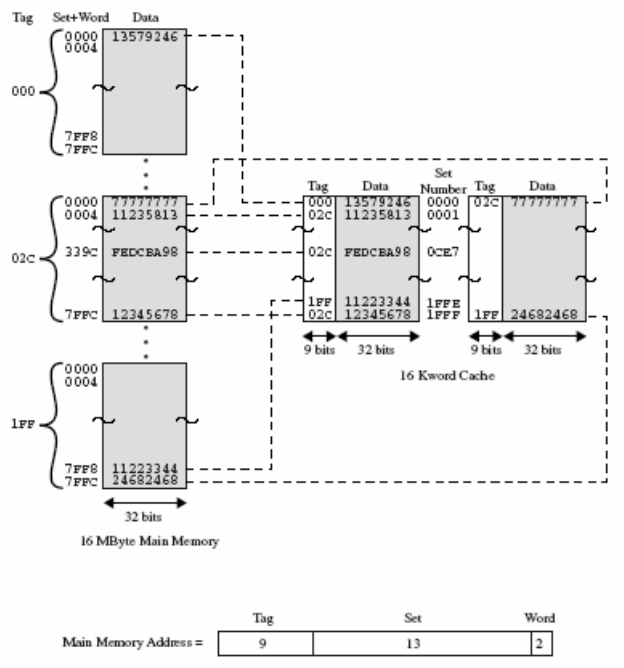
Set Associative Mapping

- Compromise between direct and associative mapping
- Cache divided into v sets
- Each set contains k lines
- A given block maps into any line in a given set
    - Ex: block B can be mapped into any line in set i
- Ex: Assume that k = 2 – meaning there are 2 lines per set
    - 2-way associative mapping – k-way mapping
    - A given block can map into either of the 2 lines in exactly 1 set

k-Way Set Associative Cache Organization



Example of Set Associative Mapping



- Main memory addresses are viewed as two fields

Tag (s-d)	Set (d)	Word (w)
9 bits	13 bits	2 bits

- 9 bit tag field – only compared with k tags in the given set
- 13 bit set field – specifies one of the v = 2^d sets
- 2 bit byte number

Example:

Hex Address:	16339C	Binary Address:	0001 0110 0011 0011 1001 1100
Hex Tag:	02C	Binary Tag:	0 0010 1100
Hex Address:	16339C	Binary Address:	0001 0110 0011 0011 1001 1100
Hex Set:	0CE7	Binary Set:	0 1100 1110 0111

“cache set #” = “main memory block #” % “number of sets”

Set Associative Mapping Summary

- Address length = (s+w) bits
- Number of addressable units = 2^(s+w) words or bytes
- Block Size = line size = 2^w words or bytes

- Number of blocks in main memory = 2^(s+w)/2^w = 2^s
- Number of lines in set = k
- Number of sets v = 2^d
- Number of lines in cache = kv = k*2^d
- Size of tag = (s-d) bits
- v = m, k = 1 reduces to direct mapping
- v = 1, k = m reduces to associative mapping
- 2-way mapping is most commonly used – significantly improves cache hit ratio over direct mapping
- 4-way mapping – makes a modest additional improvement for a relatively small additional cost

Replacement Algorithms

Direct Mapping
- No choice
- Each block only maps to one line
- Must replace that line

Associative and Set Associative
- Must be implemented in hardware for speed
- Most effective – Least Recently Used (LRU)
    - Replace the block in the set that has been in cache the longest with no references to it
    - 2-way set associative – each line includes a USE bit
- First-in-first-out (FIFO)
    - Replace the block in the set that has been in the cache the longest
    - Uses a round-robin or circular buffer technique
- Least Frequently Used (LFU)
    - Replace the block in the set that has experienced the fewest references
    - Associate a counter with each line
- Pick a line at random – not based usage
    - Only slightly inferior in performance to algorithms based on usage

Write Policy
- What if cache has been altered and main memory doesn't match
    - must write to memory before replacing the word in cache
- What if multiple processors are present and each one has its own cache
- What if an I/O device addresses main memory directly

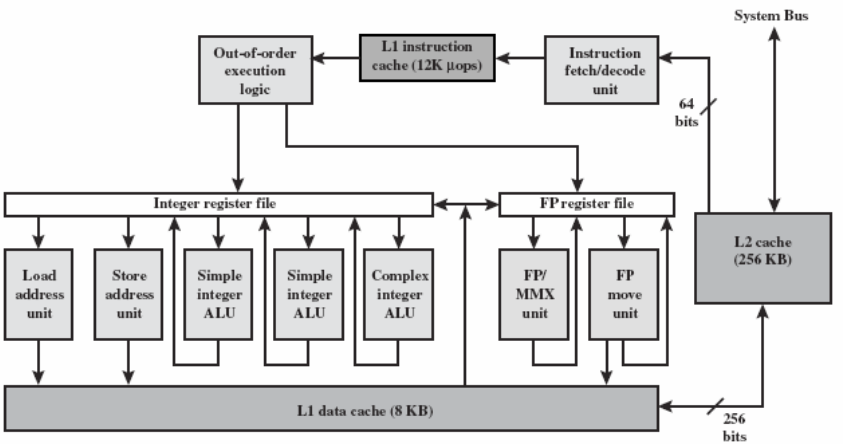
- Write Through
  - All writes go to both cache and main memory
  - Each processor can monitor main memory activity to keep local cache current
  - Generates a lot of memory traffic
- Write Back
  - Minimizes memory writes
  - Updates are made only in cache
  - UPDATE bit associated with the line is set
  - When the block is replaced it is written back to main memory only if the UPDATE bit is set
  - This may cause other caches to be out of sync
  - I/O must access main memory through the cache
  - 15% of memory references are writes

Pentium 4 and PowerPC Cache Organizations

Pentium 4 Cache Organization

- 80386 – no on chip cache
- 80486 – 8k bytes using 16 bytes/lines and 4-way set associative organization
- Pentium (all versions) two on chip L1 caches
  - Data and instructions
- Pentium 4
  - L1 caches
    - 8k bytes
    - 64 bytes/line
    - 4-way set associative
  - L2 cache
    - Feeds both L1 caches
    - 256k bytes
    - 128 bytes/line
    - 8-way set associative

Pentium 4 Block Diagram



Pentium 4 Core Processor

- Fetch/Decode unit
  - Fetch instructions from L2 cache
  - Decode into micro-operations
  - Store micro-operations in L1 cache
- Out-of-order execution logic
  - Schedules micro-operations
  - Based on data dependencies and resource availability
  - May schedule speculative execution as time permits
- Execution units
  - Executes micro-operations
  - Fetches required data from L1 data cache
  - Temporarily stores results in registers
- Memory subsystem
  - Includes L2 cache and system bus

Pentium 4 Design reasoning

- Decodes instructions into RISC like instructions called micro-operations
- Simple, fixed length micro-operations enable superscalar pipelining and schedule techniques that enhance performance
- Instructions are cumbersome to decode – variable length with many options
- Performance is enhanced by separating decoding form scheduling and pipelining
- Data cache is write back – can be configured for write through
- L1 cache controlled by 2 bits in control registers
  - CD – cache disable

    - NW – not write-through
    - INVD – instruction that invalidates (flushes) the cache
    - WNINVD – instruction that writes back the cache and then invalidates the cache

PowerPC Cache Organization

- 601
    - 32k bytes
    - 32 bytes/line
    - 8-way set associative
- 603
    - 2 – 8k bytes
    - 32 bytes/line
    - 2-way set associative
- 604
    - 2 – 16k bytes
    - 32 bytes/line
    - 4-way set associative
- 620
    - 2 – 32k bytes
    - 64 bytes/line
    - 8-way set associative
- G3
    - 2 – 32k bytes
    - 64 bytes/line
    - 8-way set associative
- G4
    - L1 caches
        - 2 – 32k bytes
        - 32 bytes/line
        - 8-way set associative
    - L2 cache
        - 126k, 512k, or 1M bytes
        - 2-way set associative

PowerPC G4 Block Diagram