CSCI 4717/5717
Computer Architecture

Topic: Cache Memory
Reading: Stallings, Chapter 4

Characteristics of Memory “Location wrt Processor”

- Inside CPU – temporary memory or registers
- Inside processor – L1 cache
- Motherboard – main memory and L2 cache
- Main memory – DRAM and L3 cache
- External – peripherals such as disk, tape, and networked memory devices

Characteristics of Memory “Capacity – Word Size”

- The natural data size for a processor.
- A 32-bit processor has a 32-bit word.
- Typically based on processor's data bus width (i.e., the width of an integer or an instruction)
- Varying widths can be obtained by putting memory chips in parallel with same address lines

Characteristics of Memory “Capacity – Addressable Units”

- Varies based on the system's ability to allow addressing at byte level etc.
- Typically smallest location which can be uniquely addressed
- At mother board level, this is the word
- It is a cluster on disks
- Addressable units (N) equals 2 raised to the power of the number of bits in the address bus

Characteristics of Memory “Unit of transfer”

- The number of bits read out of or written into memory at a time.
- Internal – Usually governed by data bus width, i.e., a word
- External – Usually a block which is much larger than a word

Characteristics of Memory “Access method”

- Based on the hardware implementation of the storage device
- Four types
  - Sequential
  - Direct
  - Random
  - Associative

Sequential Access Method

- Start at the beginning and read through in order
- Access time depends on location of data and previous location
- Example: tape

Direct Access Method

- Individual blocks have unique address
- Access is by jumping to vicinity then performing a sequential search
- Access time depends on location of data within "block" and previous location
- Example: hard disk

Random Access Method

- Individual addresses identify locations exactly
- Access time is consistent across all locations and is independent previous access
- Example: RAM

Associative Access Method

- Addressing information must be stored with data in a general data location
- A specific data element is located by a comparing desired address with address portion of stored elements
- Access time is independent of location or previous access
- Example: cache

Performance – Access Time

- Time between "requesting" data and getting it
- RAM
  - Time between putting address on bus and getting data.
  - It's predictable.
- Other types, Sequential, Direct, Associative
  - Time it takes to position the read-write mechanism at the desired location.
  - Not predictable.

Performance – Memory Cycle time

- Primarily a RAM phenomenon
- Adds "recovery" time to cycle allowing for transients to dissipate so that next access is reliable.
- Cycle time is access + recovery

Performance – Transfer Rate

- Rate at which data can be moved
- RAM – Predictable; equals 1/(cycle time)
- Non-RAM – Not predictable; equals

$T_N = T_A + (N/R)$

where
- – T_N = Average time to read or write N bits
- – T_A = Average access time
- – N = Number of bits
- – R = Transfer rate in bits per second

Physical Types

- Semiconductor – RAM
- Magnetic – Disk & Tape
- Optical – CD & DVD
- Others
  - – Bubble (old) – memory that made a "bubble" of charge in an opposite direction to that of the thin magnetic material that on which it was mounted
  - – Hologram (new) – much like the hologram on your credit card, laser beams are used to store computer-generated data in three dimensions. (10 times faster with 12 times the density)

Physical Characteristics

- Decay
  - – Power loss
  - – Degradation over time
- Volatility – RAM vs. Flash
- Erasable – RAM vs. ROM
- Power consumption – More specific to laptops, PDAs, and embedded systems

Organization

- Physical arrangement of bits into words
- Not always obvious
- Non-sequential arrangements may be due to speed or reliability benefits, e.g. interleaved

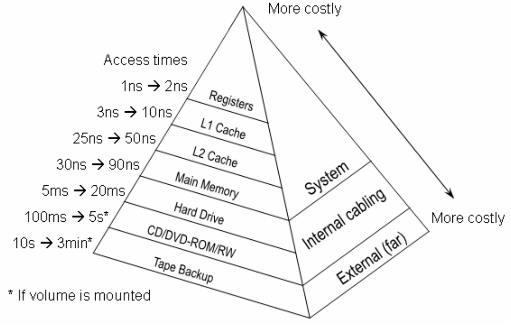
Memory Hierarchy

- Trade-offs among three key characteristics
  - – Amount – Software will ALWAYS fill available memory
  - – Speed – Memory should be able to keep up with the processor
  - – Cost – Whatever the market will bear
- Balance these three characteristics with a memory hierarchy
- Analogy –
  Refrigerator & cupboard (fast access – lowest variety)
  freezer & pantry (slower access – better variety)
  grocery store (slowest access – greatest variety)

Memory Hierarch (continued)

Implementation – Going down the hierarchy has the following results:
- – Decreasing cost per bit (cheaper)
- – Increasing capacity (larger)
- – Increasing access time (slower)
- – **KEY** – Decreasing frequency of access of the memory by the processor

Memory Hierarch (continued)



Source: Null, Linda and Lobur, Julia (2003). *Computer Organization and Architecture* (p. 236). Sudbury, MA: Jones and Bartlett Publishers.

Mechanics of Technology

- The basic mechanics of creating memory directly affect the first three characteristics of the hierarchy:
  - Decreasing cost per bit
  - Increasing capacity
  - Increasing access time
- The fourth characteristic is met because of a principle known as **locality of reference**

In-Class Exercise

- In groups, examine the following code. Identify how many times the processor "touches" each piece of data and each line of code:

```
int values[8] =
   {9, 34, 23, 67, 23, 7, 3, 65};
int count;
int sum = 0;
for (count = 0; count < 8; count++)
   sum += values[count];
```

- For better results, try the same exercise using the assembly language version found at: http://faculty.etsu.edu/tarnoff/ntes4717/week_03/assy.pdf

Locality of Reference

Due to the nature of programming, instructions and data tend to cluster together (loops, subroutines, and data structures)

- Over a long period of time, clusters will change
- Over a short period, clusters will tend to be the same

Breaking Memory into Levels

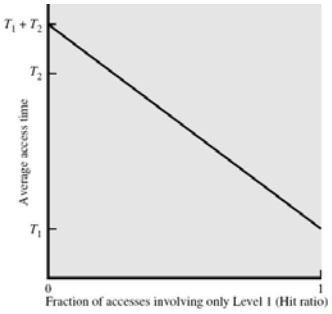
- Assume a hypothetical system has two levels of memory
  - Level 2 should contain all instructions and data
  - Level 1 doesn't have room for everything, so when a new cluster is required, the cluster it replaces must be sent back to the level 2
- These principles can be applied to much more than just two levels
- If performance is based on amount of memory rather than speed, lower levels can be used to simulate larger sizes for higher levels, e.g., virtual memory

Memory Hierarchy Examples

Example: If 95% of the memory accesses are found in the faster level, then the average access time might be:

(0.95)(0.01 uS) + (0.05)(0.1 uS) = 0.0095 + 0.0055
= 0.015 uS

Performance of a Simple Two-Level Memory (Figure 4.2)



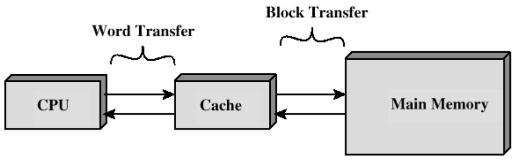
Hierarchy List

- Registers – volatile
- L1 Cache – volatile
- L2 Cache – volatile
- CDRAM (main memory) cache – volatile
- Main memory – volatile
- Disk cache – volatile
- Disk – non-volatile
- Optical – non-volatile
- Tape – non-volatile

Cache

- What is it? A cache is a small amount of fast memory
- What makes small fast?
  - Simpler decoding logic
  - More expensive SRAM technology
  - Close proximity to processor – Cache sits between normal main memory and CPU or it may be located on CPU chip or module

Cache (continued)



Cache operation – overview

- CPU requests contents of memory location
- Check cache for this data
- If present, get from cache (fast)
- If not present, one of two things happens:
  - read required block from main memory to cache then deliver from cache to CPU (cache physically between CPU and bus)
  - read required block from main memory to cache and simultaneously deliver to CPU (CPU and cache both receive data from the same data bus buffer)

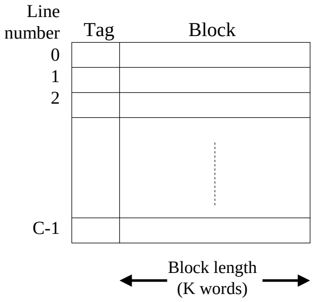
Going Deeper with Principle of Locality

- Cache "misses" are unavoidable, i.e., every piece of data and code thing must be loaded at least once
- What does a processor do during a miss? It waits for the data to be loaded.
- Power consumption varies linearly with clock speed and the square of the voltage.
- Adjusting clock speed and voltage of processor has the potential to produce cubic (cubed root) power reductions (http://www.visc.vt.edu/~mhsiao/papers/pacs00ch.pdf)
- Identify places in in-class exercise where this might happen.

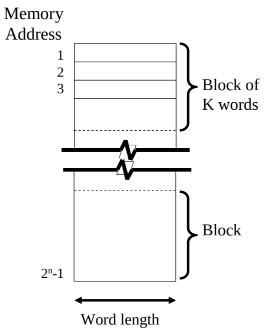
Cache Structure

- Cache includes tags to identify the address of the block of main memory contained in a line of the cache
- Each word in main memory has a unique n-bit address
- There are $M=2^n/K$ block of K words in main memory
- Cache contains C lines of K words each plus a tag uniquely identifying the block of K words

Cache Structure (continued)



Memory Divided into Blocks



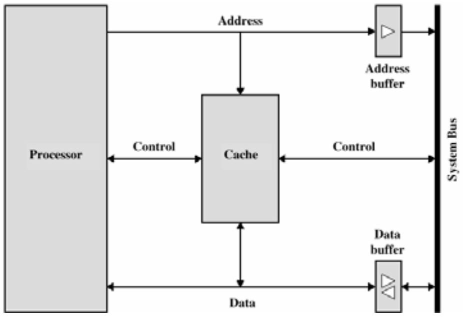
Cache Design

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

Cache size

- Cost – More cache is expensive
- Speed
  - More cache is faster (up to a point)
  - Larger decoding circuits slow up a cache
  - Algorithm is needed for mapping main memory addresses to lines in the cache. This takes more time than just a direct RAM

Typical Cache Organization



Mapping Functions

- A mapping function is the method used to locate a memory address within a cache
- It is used when copying a block from main memory to the cache and it is used again when trying to retrieve data from the cache
- There are three kinds of mapping functions
  - Direct
  - Associative
  - Set Associative

Cache Example

These notes use an example of a cache to illustrate each of the mapping functions. The characteristics of the cache used are:

- Size: 64 kByte
- Block size: 4 bytes – i.e. the cache has 16k (2^{14}) lines of 4 bytes
- Address bus: 24-bit– i.e., 16M bytes main memory divided into 4M 4 byte blocks

Direct Mapping Traits

- Each block of main memory maps to only one cache line – i.e. if a block is in cache, it will always be found in the same place
- Line number is calculated using the following function

$$i = j \text{ modulo } m$$

where

i = cache line number

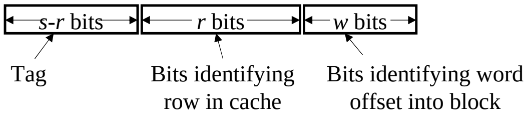
j = main memory block number

m = number of lines in the cache

Direct Mapping Address Structure

Each main memory address can by divided into three fields

- Least Significant w bits identify unique word within a block
- Remaining bits (s) specify which block in memory. These are divided into two fields
  - Least significant r bits of these s bits identifies which line in the cache
  - Most significant s-r bits uniquely identifies the block within a line of the cache


s-r bits
r bits
w bits
Tag
Bits identifying row in cache
Bits identifying word offset into block

Direct Mapping Address Structure (continued)

- Why are the r-bits used to identify which line in cache?
- More likely to have unique r bits than s-r bits based on principle of **locality of reference**

Direct Mapping Address Structure Example

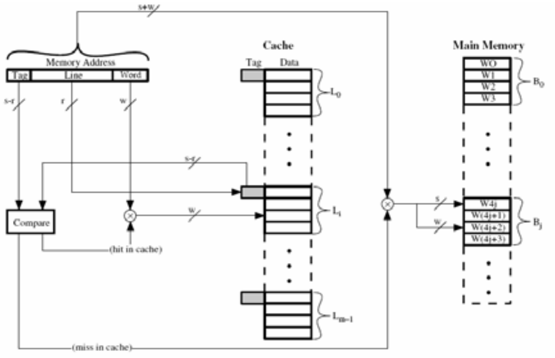
Tag s-r	Line or slot r	Word w
8	14	2

- 24 bit address
- 2 bit word identifier (4 byte block)
- 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
- Check contents of cache by finding line and comparing tag

Direct Mapping Cache Line Table

Cache line	Main Memory blocks held
0	0, m, 2m, 3m…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



Direct Mapping Examples

What cache line number will the following addresses be stored to, and what will the minimum address and the maximum address of each block they are in be if we have a cache with 4K lines of 16 words to a block in a 256 Meg memory space (28-bit address)?

Tag s-r	Line or slot r	Word w
12	12	4

a.) $9ABCDEF_{16}$
b.) 1234567_{16}

More Direct Mapping Examples

Assume that a portion of the tags in the cache in our example looks like the table below. Which of the following addresses are contained in the cache?

a.) $438EE8_{16}$ b.) $F18EFF_{16}$ c.) $6B8EF3_{16}$ d.) $AD8EF3_{16}$

Tag (binary)	Line number (binary)	Addresses wi/ block			
		00	01	10	11
0101 0011	1000 1110 1110 10				
1110 1101	1000 1110 1110 11				
1010 1101	1000 1110 1111 00				
0110 1011	1000 1110 1111 01				
1011 0101	1000 1110 1111 10				
1111 0001	1000 1110 1111 11				

Direct Mapping Summary

- Address length = (s + w) bits
- Number of addressable units = 2^{s+w} words or bytes
- Block size = line width = 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 & cons

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

Associative Mapping Traits

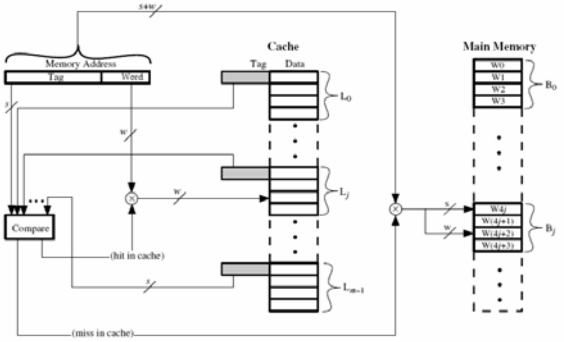
- A main memory block can load into any line of cache
- Memory address is interpreted as:
  - Least significant w bits = word position within block
  - Most significant s bits = tag used to identify which block is stored in a particular line of cache
- Every line's tag must be examined for a match
- Cache searching gets expensive and slower

Associative Mapping Address Structure Example

Tag – s bits (22 in example)	Word – w bits (2 in ex.)

- 22 bit tag stored with each 32 bit block of data
- Compare tag field with tag entry in cache to check for hit
- Least significant 2 bits of address identify which of the four 8 bit words is required from 32 bit data block

Fully Associative Cache Organization



Fully Associative Mapping Example

Assume that a portion of the tags in the cache in our example looks like the table below. Which of the following addresses are contained in the cache?

a.) $438EE8_{16}$ b.) $F18EFF_{16}$ c.) $6B8EF3_{16}$ d.) $AD8EF3_{16}$

Tag (binary)	Addresses wi/ block			
	00	01	10	11
0101 0011 1000 1110 1110 10				
1110 1101 1100 1001 1011 01				
1010 1101 1000 1110 1111 00				
0110 1011 1000 1110 1111 11				
1011 0101 0101 1001 0010 00				
1111 0001 1000 1110 1111 11				

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

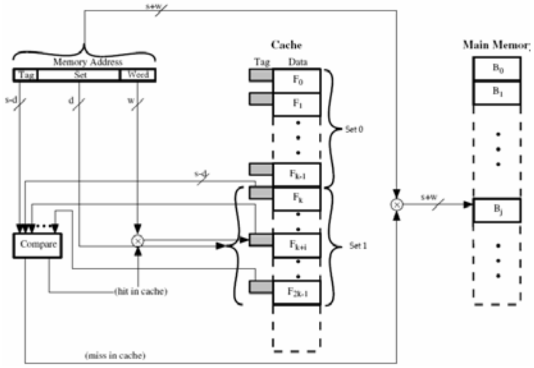
Set Associative Mapping Traits

- Address length is $s + w$ bits
- Cache is divided into a number of sets, $v = 2^d$
- k blocks/lines can be contained within each set
- k lines in a cache is called a k-way set associative mapping
- Number of lines in a cache = $v \cdot k = k \cdot 2^d$
- Size of tag = $(s-d)$ bits

Set Associative Mapping Traits (continued)

- Hybrid of Direct and Associative
  k = 1, this is basically direct mapping
  v = 1, this is associative mapping
- Each set contains a number of lines, basically the number of lines divided by the number of sets
- A given block maps to any line within its specified set – e.g. Block B can be in any line of set i.
- 2 lines per set is the most common organization.
  - Called 2 way associative mapping
  - A given block can be in one of 2 lines in only one specific set
  - Significant improvement over direct mapping

K-Way Set Associative Cache Organization



How does this affect our example?

- Let’s go to two-way set associative mapping
- Divides the 16K lines into 8K sets
- This requires a 13 bit set number
- With 2 word bits, this leaves 9 bits for the tag
- Blocks beginning with the addresses 000000_{16}, 008000_{16}, 010000_{16}, 018000_{16}, 020000_{16}, 028000_{16}, etc. map to the same set, Set 0.
- Blocks beginning with the addresses 00000416, 008004_{16}, 010004_{16}, 018004_{16}, 020004_{16}, 028004_{16}, etc. map to the same set, Set 1.

Set Associative Mapping Address Structure

Tag 9 bits	Set 13 bits	Word 2 bits

- Note that there is one more bit in the tag than for this same example using direct mapping.
- Therefore, it is 2-way set associative
- Use set field to determine cache set to look in
- Compare tag field to see if we have a hit

Set Associative Mapping Example

For each of the following addresses, answer the following questions based on a 2-way set associative cache with 4K lines, each line containing 16 words, with the main memory of size 256 Meg memory space (28-bit address):

- What cache set number will the block be stored to?
- What will their tag be?
- What will the minimum address and the maximum address of each block they are in be?

1. $9ABCDEF_{16}$
2. 1234567_{16}

Tag s-r	Set s	Word w
13	11	4

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

Replacement Algorithms

- There must be a method for selecting which line in the cache is going to be replaced when there’s no room for a new line
- Hardware implemented algorithm (speed)
- Direct mapping
  - There is no need for a replacement algorithm with direct mapping
  - Each block only maps to one line
  - Replace that line

Associative & Set Associative Replacement Algorithms

- Least Recently used (LRU)
  - Replace the block that hasn't been touched in the longest period of time
  - Two way set associative simply uses a USE bit. When one block is referenced, its USE bit is set while its partner in the set is cleared
- First in first out (FIFO) – replace block that has been in cache longest

Associative & Set Associative Replacement Algorithms (continued)

- Least frequently used (LFU) – replace block which has had fewest hits
- Random – only slightly lower performance than use-based algorithms LRU, FIFO, and LFU

Writing to Cache

- Must not overwrite a cache block unless main memory is up to date
- Two main problems:
  - If cache is written to, main memory is invalid or if main memory is written to, cache is invalid – Can occur if I/O can address main memory directly
  - Multiple CPUs may have individual caches; once one cache is written to, all caches are invalid

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
- Update bit for cache slot is set when update occurs
- If block is to be replaced, write to main memory only if update bit is set
- Other caches get out of sync
- I/O must access main memory through cache
- Research shows that 15% of memory references are writes

Multiple Processors/Multiple Caches

- Even if a write through policy is used, other processors may have invalid data in their caches
- In other words, if a processor updates its cache and updates main memory, a second processor may have been using the same data in its own cache which is now invalid.

Solutions to Prevent Problems with Multiprocessor/cache systems

- **Bus watching with write through** – each cache watches the bus to see if data they contain is being written to the main memory by another processor. All processors must be using the write through policy
- **Hardware transparency** – a "big brother" watches all caches, and upon seeing an update to any processor's cache, it updates main memory AND all of the caches
- **Noncacheable memory** – Any shared memory (identified with a chip select) may not be cached.

Line Size

- There is a relationship between line size (i.e., the number of words in a line in the cache) and hit ratios
- As the line size (block size) goes up, the hit ratio could go up due to more words available to the principle of **locality of reference**
- As block size increases, however, the number of blocks goes down, and the hit ratio will begin to go back down after a while
- Lastly, as the block size increases, the chances of a hit to a word farther from the initially referenced word goes down

Multi-Level Caches

- Increases in transistor densities have allowed for caches to be placed inside processor chip
- Internal caches have very short wires (within the chip itself) and are therefore quite fast, even faster then any zero wait-state memory accesses outside of the chip
- This means that a super fast internal cache (level 1) can be inside of the chip while an external cache (level 2) can provide access faster then to main memory

Unified versus Split Caches

- Split into two caches – one for instructions, one for data
- Disadvantages
  - Questionable as unified cache balances data and instructions merely with hit rate.
  - Hardware is simpler with unified cache
- Advantage
  - What a split cache is really doing is providing one cache for the instruction decoder and one for the execution unit.
  - This supports pipelined architectures.

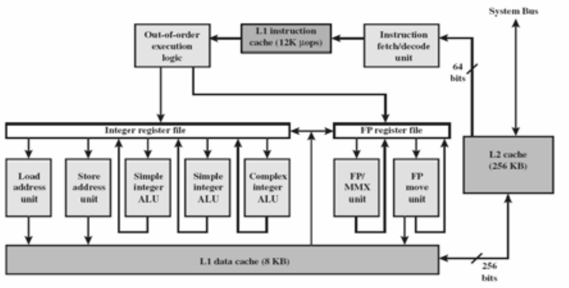
Intel x86 caches

- 80386 – no on chip cache
- 80486 – 8k using 16 byte lines and four-way set associative organization (main memory had 32 address lines – 4 Gig)
- Pentium (all versions)
  - Two on chip L1 caches
  - Data & instructions

Pentium 4 L1 and L2 Caches

- L1 cache
  - 8k bytes
  - 64 byte lines
  - Four way set associative
- L2 cache
  - Feeding both L1 caches
  - 256k
  - 128 byte lines
  - 8 way set associative

Pentium 4 (Figure 4.13)



Pentium 4 Operation – Core Processor

- Fetch/Decode Unit
  - Fetches instructions from L2 cache
  - Decode into micro-ops
  - Store micro-ops in L1 cache
- Out of order execution logic
  - Schedules micro-ops
  - Based on data dependence and resources
  - May speculatively execute

Pentium 4 Operation – Core Processor (continued)

- Execution units
  - Execute micro-ops
  - Data from L1 cache
  - Results in registers
- Memory subsystem – L2 cache and systems bus

Pentium 4 Design Reasoning

- Decodes instructions into RISC like micro-ops before L1 cache
- Micro-ops fixed length – Superscalar pipelining and scheduling
- Pentium instructions long & complex
- Performance improved by separating decoding from scheduling & pipelining – (More later – ch14)

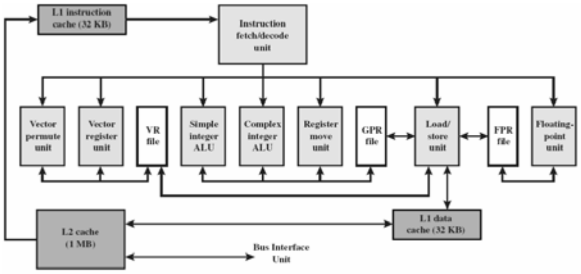
Pentium 4 Design Reasoning (continued)

- Data cache is write back – Can be configured to write through
- L1 cache controlled by 2 bits in register
  - CD = cache disable
  - NW = not write through
  - 2 instructions to invalidate (flush) cache and write back then invalidate

Power PC Cache Organization

- 601 – single 32kb 8 way set associative
- 603 – 16kb (2 x 8kb) two way set associative
- 604 – 32kb
- 610 – 64kb
- G3 & G4
  - – 64kb L1 cache – 8 way set associative
  - – 256k, 512k or 1M L2 cache – two way set associative

PowerPC G4 (Figure 4.14)



Comparison of Cache Sizes (Table 4.3)

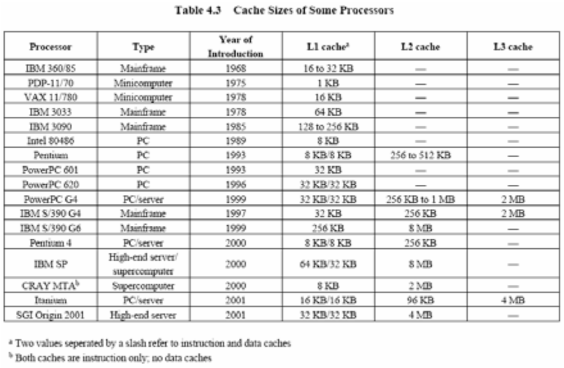


Table 4.3 Cache Sizes of Some Processors

Processor	Type	Year of Introduction	L1 cache[a]	L2 cache	L3 cache
IBM 360/85	Mainframe	1968	16 to 32 KB	—	—
PDP-11/70	Minicomputer	1975	1 KB	—	—
VAX 11/780	Minicomputer	1978	16 KB	—	—
IBM 3033	Mainframe	1978	64 KB	—	—
IBM 3090	Mainframe	1985	128 to 256 KB	—	—
Intel 80486	PC	1989	8 KB	—	—
Pentium	PC	1993	8 KB/8 KB	256 to 512 KB	—
PowerPC 601	PC	1993	32 KB	—	—
PowerPC 620	PC	1996	32 KB/32 KB	—	—
PowerPC G4	PC/server	1999	32 KB/32 KB	256 KB to 1 MB	2 MB
IBM S/390 G4	Mainframe	1997	32 KB	256 KB	2 MB
IBM S/390 G6	Mainframe	1999	256 KB	8 MB	—
Pentium 4	PC/server	2000	8 KB/8 KB	256 KB	—
IBM SP	High-end server/ supercomputer	2000	64 KB/32 KB	8 MB	—
CRAY MTA[b]	Supercomputer	2000	8 KB	2 MB	—
Itanium	PC/server	2001	16 KB/16 KB	96 KB	4 MB
SGI Origin 2001	High-end server	2001	32 KB/32 KB	4 MB	—

[a] Two values seperated by a slash refer to instruction and data caches

[b] Both caches are instruction only; no data caches