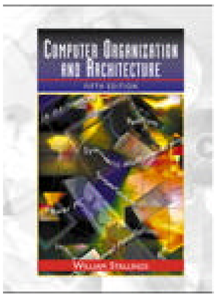
Campus de Gualtar
4710-057 Braga

UNIVERSIDADE DO MINHO
ESCOLA DE ENGENHARIA

Departamento de Informática



Computer Organization and Architecture

5th Edition, 2000

by William Stallings

Table of Contents

II. THE COMPUTER SYSTEM.

3. ...

4. Internal Memory. (29-Feb-00)

Characteristics of Computer Memory Systems (4.1)

- Location
  - CPU (registers and L1 cache)
  - Internal Memory (main)
  - External (secondary)
- Capacity
  - Word Size - typically equal to the number of bits used to represent a number and to the instruction length.
  - Number of Words - has to do with the number of addressable units (which are typically words, but are sometimes bytes, regardless of word size). For addresses of length A (in bits), the number of addressable units is 2A.
- Unit of Transfer
  - Word
  - Block
- Access Method
  - Sequential Access
    - information used to separate or identify records is stored with the records
    - access must be made in a specific linear sequence
    - the time to access an arbitrary record is highly variable
  - Direct Access
    - individual blocks or records have an address based on physical location
    - access is by direct access to general vicinity of desired information, then some search
    - access time is still variable, but not as much as sequential access
  - Random Access
    - each addressable location has a unique, physical location
    - access is by direct access to desired location
    - access time is constant and independent of prior accesses
  - Associative
    - desired units of information are retrieved by comparing a sub-part of the unit with a desired mask -- location is not needed
    - all matches to the mask are retrieved simultaneously
    - access time is constant and independent of prior accesses
    - most useful for searching - a search through N possible locations would take O(N) with Random Access Memory, but O(1) with Associative Memory
- Performance
  - Access Time
  - Memory Cycle Time - primarily for random-access memory = access time + additional time required before a second access can begin (refresh time, for example)
  - Transfer Rate
    - Generally measured in bits/second

        - Inversely proportional to memory cycle time for random access memory

- Physical Type
    - Most common - semiconductor and magnetic surface memories
    - Others - optical, bubble, mechanical (e.g. paper tape), core, esoteric/theoretical (e.g. biological)
- Physical Characteristics
    - volatile - information decays or is lost when power is lost
    - non-volatile - information remains without deterioration until changed -- no electrical power needed
    - non-erasable
        - information cannot be altered with a normal memory access cycle
        - As a practical matter, must be non-volatile
- Organization - the physical arrangement of bits to form words.
    - Obvious arrangement not always used
    - Ex. Characters vs. Integers vs. Floating Point Numbers

The Memory Hierarchy

- Design Constraints
    - How much? “If you build it, they will come.” Applications tend to be built to use any commonly available amount, so question is open-ended.
    - How fast? Must be able to keep up with the CPU -- don’t want to waste cycles waiting for instructions or operands.
    - How expensive? Cost of memory (also associated with “How much?”) must be reasonable vs. other component costs.
- There are trade-offs between the 3 key characteristics of memory (cost, capacity, and access time) which yield the following relationships:
    - Smaller access time -> greater cost per bit
    - Greater capacity -> smaller cost per bit
    - Greater capacity -> greater access time
- The designer’s dilemma
    - Would like to use cheaper, large capacity memory technologies
    - Good performance requires expensive, lower-capacity, quick-access memories
- Solution: Don’t rely on a single memory component or technology -- use a memory hierarchy
    - Organizes memories such that:
        - Cost/bit decreases
        - Capacity increases
        - Access time increases
    - Data and instructions are distributed across this memory according to:
        - Frequency of access of the memory by the CPU decreases (key to success)
    - This scheme will reduced overall costs while maintaining a given level of performance.
- Contemporary Memory Hierarchy

    Magnetic Tape
    Optical/Magnetic Disk
    Disk Cache
    Main Memory
    Cache
    Registers

- Success depends upon the locality of reference principle
  - memory references tend to cluster
    - temporal locality - if a location is referenced, it is likely to be referenced again in the near future
    - positional locality - when a location is referenced, it is probably close to the last location referenced
  - so a single (slower) transfer of memory from a lower level of the hierarchy to a higher level of the hierarchy will tend to service a disproportionate number of future requests, which can be satisfied by the higher (faster) level
  - This is the technique which is the basis for caching and virtual memory
  - Although we don't always refer to it as caching, this technique is used at all levels of the memory hierarchy, often supported at the operating system level

Semiconductor Main Memory (4.2)

- Types of Random-Access Semiconductor Memory
  - RAM - Random Access Memory
    - misused term (all these are random access)
    - possible both to read data from the memory and to easily and rapidly write new data into the memory
    - volatile - can only be used for temporary storage (all the other types of random-access memory are non-volatile)
    - types:
      - dynamic - stores data as charge on capacitors
        - tend to discharge over time
        - require periodic charge (like a memory reference) to refresh
        - more dense and less expensive than comparable static RAMs
      - static - stores data in traditional flip-flop logic gates
        - no refresh needed
        - generally faster than dynamic RAMs
  - ROM - Read Only Memory
    - contains a permanent pattern of data which cannot be changed
    - data is actually wired-in to the chip as part of the fabrication process
      - data insertion step has a large fixed cost
      - no room for error
    - cheaper for high-volume production
  - PROM - Programmable Read Only Memory
    - writing process is performed electrically
    - may be written after chip fabrication
      - writing uses different electronics than normal memory writes
      - no room for error
    - attractive for smaller production runs
  - EPROM - Erasable Programmable Read Only Memory
    - read and written electrically, as with PROM
    - before a write, all cells must be erased by exposure to UV radiation (erasure takes about 20 minutes)
      - writing uses different electronics than normal memory writes
      - errors can be corrected by erasing and starting over
    - more expensive than PROM
  - EEPROM - Electrically Erasable Programmable Read Only Memory
    - byte-level writing - any part(s) of the memory can be written at any time
    - updateable in place - writing uses ordinary bus control, address, and data lines

    - writing takes much longer than reading
    - more expensive (per bit) and less dense than EPROM
  - Flash Memory
    - uses electrical erasing technology
    - allows individual blocks to be erased, but not byte-level erasure, and modern flash memory is updateable in place (some may function more like I/O modules)
    - much faster erasure than EPROM
    - same density as EPROM
    - sometimes refers to other devices, such as battery-backed RAM and tiny hard-disk drives which behave like flash memory for all intents and purposes

- Organization
  - All semiconductor memory cells:
    - exhibit 2 stable (or semi-stable states) which can represent binary 1 or 0
    - are capable of being written into (at least once), to set the state
    - are capable of being read to sense the state
  - Commonly, a cell has 3 functional terminals capable of carrying an electrical signal
- Chip Logic
  - Number of bits that can be read/written at a time
    - Physical organization is same as logical organization is one extreme (W words of B bits each)
    - Other extreme is 1-bit-per-chip -- data is read/written one bit at a time
  - Typical organization
    - 4 bits read/written at a time
    - Logically 4 square arrays of 2048x2048 cells
    - Horizontal lines connect to Select terminals
    - Vertical lines connect to Data-In/Sense terminals
    - Multiple DRAMs must connect to memory controller to read/write an 8 bit word
    - Illustrates why successive generations grow by a factor of 4 -- each extra pin devoted to addressing doubles the number of rows and columns

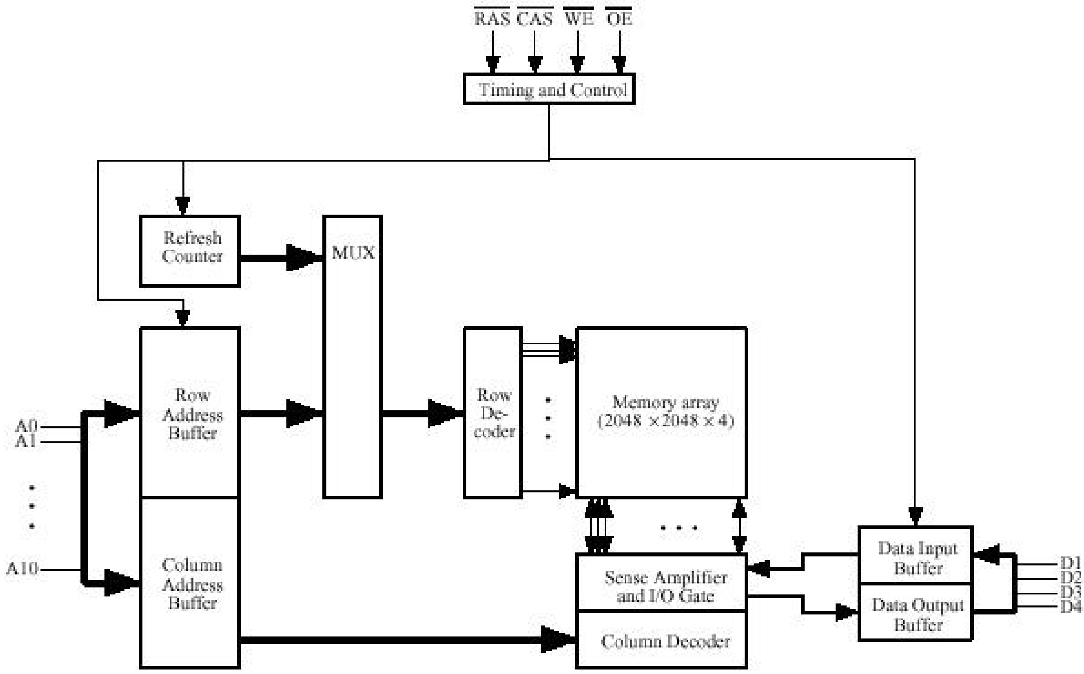


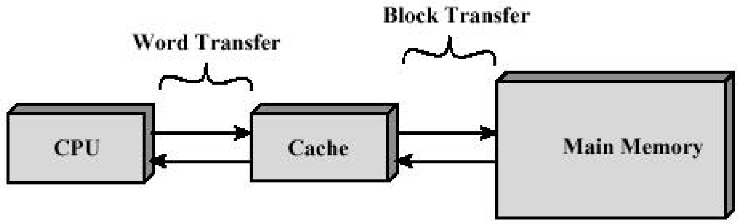
Figure 4.4 Typical 16 Megabit DRAM (4M × 4)

- Chip Packaging
  - Typical Pin outs
    - A0-An: Address of word being accessed (may be multiplexed row/column) for an n bit (n*2 bit) address
    - D0-Dn: Data in/out for n bits
    - Vcc: Power supply
    - Vss: Ground
    - CE: Chip enable - allows several chips to use same circuits for everything else, but only have one chip use them
    - Vpp: Program Voltage - used for writes to (programming) an EPROM
    - RAS: Row Address Select
    - CAS: Column Address Select
    - W or WE: Write enable
    - OE: Output enable
- Error Correction Principles
  - Hard Failure
    - a permanent defect
    - causes same result all the time, or randomly fluctuating results
  - Soft Error - a random, nondestructive event that alters the contents of one or more memory cells, without damaging the memory. Caused by:
    - power supply problems
    - alpha particles
  - Detection and Correction
- Hamming codes
  - An error-correcting code are characterized by the number of bit errors in a word that it can correct and detect
  - The Hamming Code is the simplest error-correcting code. For example, a hamming code for 4 data bits (1110) requires 3 parity bits (100), as shown (to make number of 1's in a circle even): Note that the parity bits (10) are now incorrect, and their intersection identifies the data bit in error, which can be corrected back to (1) by negation.
- SEC-DED (single-error-correcting, double-error-detecting) codes
  - Note that an error of more than a single bit cannot be corrected with the previous method (called a single-error-correcting code).
  - Instead, we can add an additional bit to make the total number of 1's (in both data and parity bits) even. If this bit compares differently, we know that a double error has been detected (although we cannot correct it)

Cache Memory (4.3)

- Principles
  - Intended to give memory speed approaching that of fastest memories available but with large size, at close to price of slower memories
  - Cache is checked first for all memory references.
  - If not found, the entire block in which that reference resides in main memory is stored in a cache slot, called a line
  - Each line includes a tag (usually a portion of the main memory address) which identifies which particular block is being stored
  - Locality of reference implies that future references will likely come from this block of memory, so that cache line will probably be utilized repeatedly.

- o The proportion of memory references, which are found already stored in cache, is called the hit ratio.



- Elements of Cache Design
  - o Cache Size
    - small enough that overall average cost/bit is close to that of main memory alone
    - large enough so that overall average access time is close to that of cache alone
    - large caches tend to be slightly slower than small ones
    - available chip and board area is a limitation
    - studies indicate that 1K-512K words is optimum cache size
- Mapping Function - determining which cache line to use for a particular block of main memory (since number_of_cache_lines << number_of_blocks)
  - o Direct mapping: line# = block# modulo number_of_cache_lines
    - each block of main memory gets a unique mapping
    - if a program happens to repeatedly reference words from two different blocks that map into the same cache line, then the blocks will be continually swapped in the cache and the hit ratio will be low.

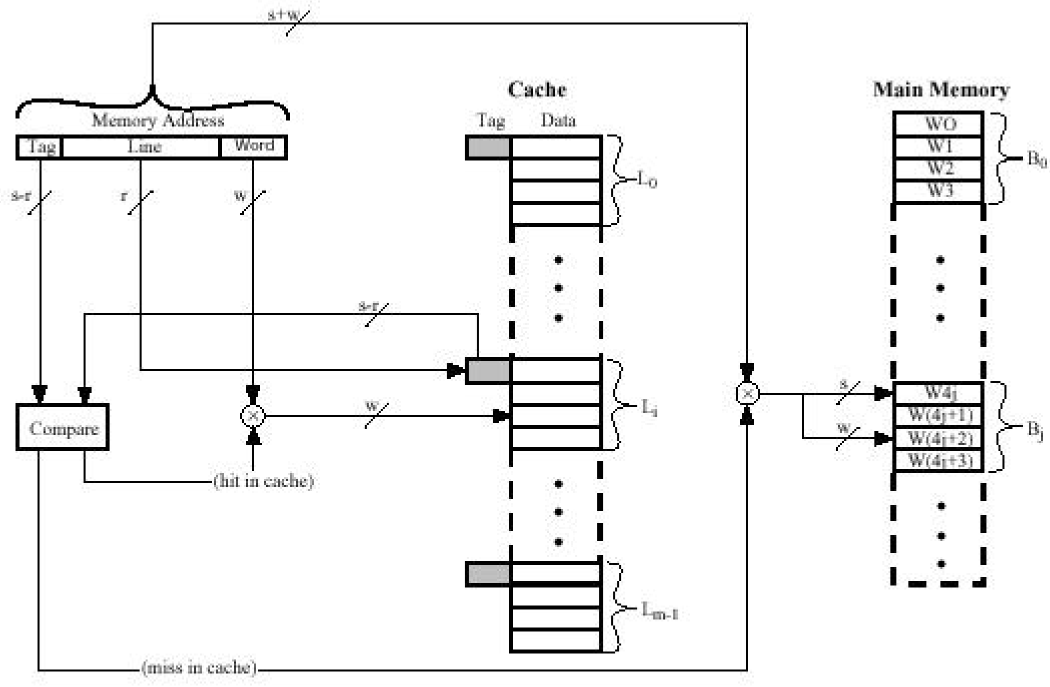


Figure 4.17 Direct-Mapping Cache Organization [HWAN93]

Note that

- all locations in a single block of memory have the same higher order bits (call them the block number), so the lower order bits can be used to find a particular word in the block.
- within those higher-order bits, their lower-order bits obey the modulo mapping given above (assuming that the number of cache lines is a power of 2), so they can be used to get the cache line for that block
- the remaining bits of the block number become a tag, stored with each cache line, and used to distinguish one block from another that could fit into that same cache line.

o Associative Mapping
  - Allows each memory block to be loaded into any line of the cache
  - Tag uniquely identifies a block of main memory
  - Cache control logic must simultaneously examine every line's tag for a match
  - Requires fully associative memory
    - very complex circuitry
    - complexity increases exponentially with size
    - very expensive

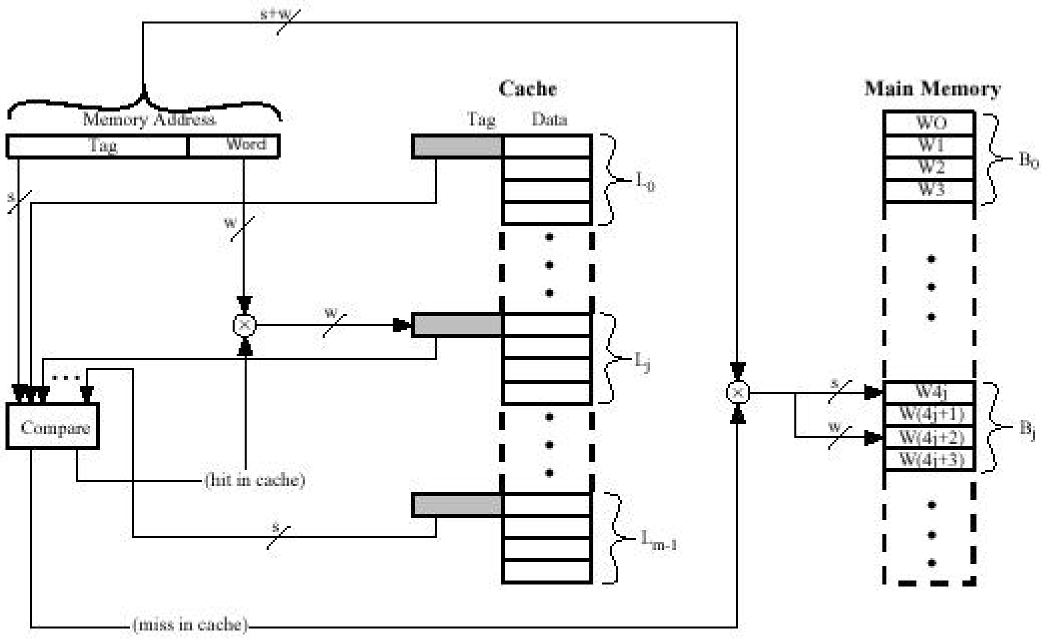


Figure 4.19 Fully Associative Cache Organization [HWAN93]

o Set Associative Mapping
  - Compromise between direct and associative mappings
  - Cache is divided into v sets, each of which has k lines
    - number_of_cache_lines = vk
    - set# = block# modulo v
    - so a given block will map directly to a particular set, but can occupy any line in that set (associative mapping is used within the set)
    - the most common set associative mapping is 2 lines per set, and is called two-way set associative. It significantly improves hit ratio over direct mapping, and the associative hardware is not too expensive.

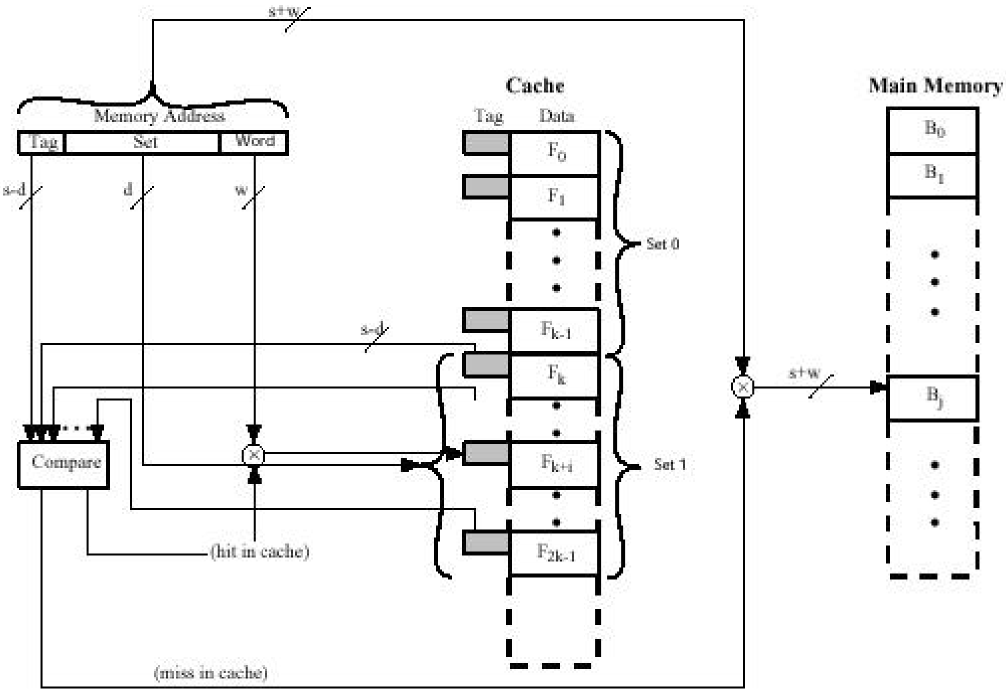


Figure 4.21 Two-Way Set Associative Cache Organization [HWAN93]

- Replacement Algorithms
  - When all lines are occupied, bringing in a new block requires that an existing line be overwritten
    - No choice possible with direct mapping
    - Algorithms must be implemented in hardware for speed
  - Least-recently-used (LRU)
    - Idea: replace that block in the set which has been in cache longest with no reference to it
    - Implementation: with 2-way set associative, have a USE bit for each line in a set. When a block is read into cache, use the line whose USE bit is set to 0, then set its USE bit to one and the other line's USE bit to 0.
    - Probably the most effective method
  - First-in-first-out (FIFO)
    - Idea: replace that block in the set which has been in the cache longest
    - Implementation: use a round-robin or circular buffer technique (keep up with which slot's "turn" is next)
  - Least-frequently-used (LFU)
    - Idea: replace that block in the set which has experienced the fewest references
    - Implementation: associate a counter with each slot and increment when used
  - Random
    - Idea: replace a random block in the set
    - Interesting because it is only slightly inferior to algorithms based on usage.

- Write Policy
  - If a block has been altered in cache, it is necessary to write it back out to main memory before replacing it with another block (writes are about 15% of memory references)

- Problems
  - I/O modules may be able to read/write directly to memory
  - Multiple CPU's may be attached to the same bus, each with their own cache
- write through
  - all write operations are made to main memory as well as to cache, so main memory is always valid
  - other CPU's monitor traffic to main memory to update their caches when needed
  - this generates substantial memory traffic and may create a bottleneck
- write back
  - when an update occurs, an UPDATE bit associated with that slot is set, so when the block is replaced it is written back first
  - accesses by I/O modules must occur through the cache
  - multiple caches still can become invalidated, unless some cache coherency system is used. Such systems include:
    - Bus Watching with Write Through - other caches monitor memory writes by other caches (using write through) and invalidates their own cache line if a match
    - Hardware Transparency - additional hardware links multiple caches so that writes to one cache are made to the others
    - Non-cacheable Memory - only a portion of main memory is shared by more than one processor, and it is non-cacheable

- Block Size
  - As the block size increases, more useful data is brought into the cache, increasing hit ratio, BUT
  - Larger blocks reduce the number of blocks that fit into a cache, and a small number of blocks results in data being overwritten shortly after it is fetched
  - As a block becomes larger, each additional word is farther from the requested word, therefore less likely to be needed in the near future
  - A size from 4 to 8 addressable units seems close to optimum
- Number of Caches
  - On-chip cache (L1 cache)
    - on same chip as CPU
    - requires no bus operation for cache hits
    - short data paths and same speed as other CPU transactions
    - reduces overall bus activity and increases not only CPU operations but overall system performance
  - Off-chip cache (L2 cache) may still be desirable
    - It can be much larger
    - It can be used with a local bus to buffer the CPU cache-misses from the system bus
- Unified vs. Split Cache
  - Unified cache
    - a single cache stores both data and instructions
    - has a higher hit rate that split cache, because it automatically balances load between data and instructions (if an execution pattern involves more instruction fetches than data fetches, the cache will fill up with more instructions than data)
    - only one cache need be designed and implemented

- o In a split cache, one cache is dedicated to instructions, and one cache is dedicated to data
  - trend is toward split cache because of superscalar CPU's
  - better for pipelining, prefetching, and other parallel instruction execution designs
  - eliminates cache contention between instruction processor and the execution unit (which uses data)

Pentium Cache Organization (4.4 + …)

- Evolution
  - o 80386 - No on-chip cache
  - o 80486 - unified 8Kbyte on-chip cache (16 byte line, 4-way set associative)
  - o Pentium - two 8Kbyte on-chip caches split between data and instructions (32 byte line, two-way set associative)
  - o Pentium Pro/II – 8K, 32 byte line, 4-way set associative instruction cache and 8K, 32 byte line, 2-way set associative data cache, plus a L2 cache on a dedicated local bus feeding both.

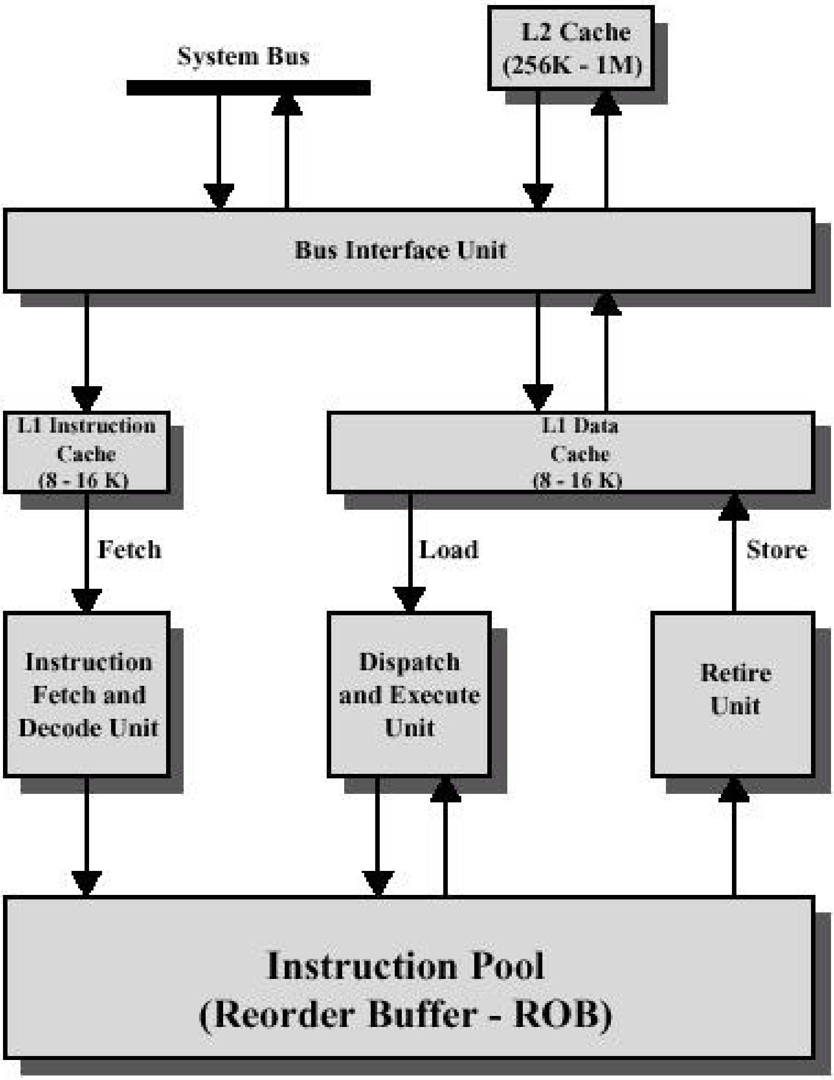


Figure 4.23 Pentium II Block Diagram

- Data Cache Internal Organization
    - Basics
        - Ways
            - 128 sets of two lines each
            - Logically organized as two 4Kbyte "ways" (each way contains one line from each set, for 128 lines per way)
        - Directories
            - Each line has a tag taken from the 20 most significant bits of the memory address of the data stored in the corresponding line
            - Each line has two state bits, one of which is used to support a write-back policy (write-through can be dynamically configured)
            - Logically organized as 2 directories, corresponding to the ways (one directory entry for each line)
        - LRU support
            - Cache controller uses a least-recently-used replacement policy
            - A single array of 128 LRU bits supports both ways (one bit for each set of two lines)
        - Level-2 cache is supported
            - May be 256 or 512 Kbytes
            - May use a 32-, 64-, or 128-byte line
            - Two-way set associative
- Data Cache Consistency
    - Supports MESI protocol
        - Supported by the two state bits mentioned earlier
        - Each line can be in one of 4 states:
            - Modified - The line in the cache has been modified and is available only in this cache
            - Exclusive - The line in the cache is the same as that in main memory and is not present in any other cache
            - Shared - The line in the cache is the same as that in main memory and may be present in another cache
            - Invalid - The line in the cache dopes not contain valid data
        - Designed to support multiprocessor organizations, but also useful for managing consistency between L1 and L2 caches in a single processor organization.
            - In such an organization, the L2 cache acts as the "memory" that is cached by the L1 cache.
            - So when MESI refers to a line being "the same as memory" (or not), it may be referring to the contents of another cache.

PowerPC Cache Organization (… 4.4)

- Evolution
    - PowerPC 601 - Unified 32Kbyte on-chip cache (32 byte line, 8-way set associative)
    - PowerPC 603 - two 8Kbyte on-chip caches split between data and instructions (32 byte line, two-way set associative)
    - PowerPC 604 - two 16Kbyte on-chip caches split between data and instructions (32 byte line, 4-way set associative)
    - PowerPC 620 - two 32Kbyte on-chip caches split between data and instructions (64 byte line, 8-way set associative)

- External Organizational Features

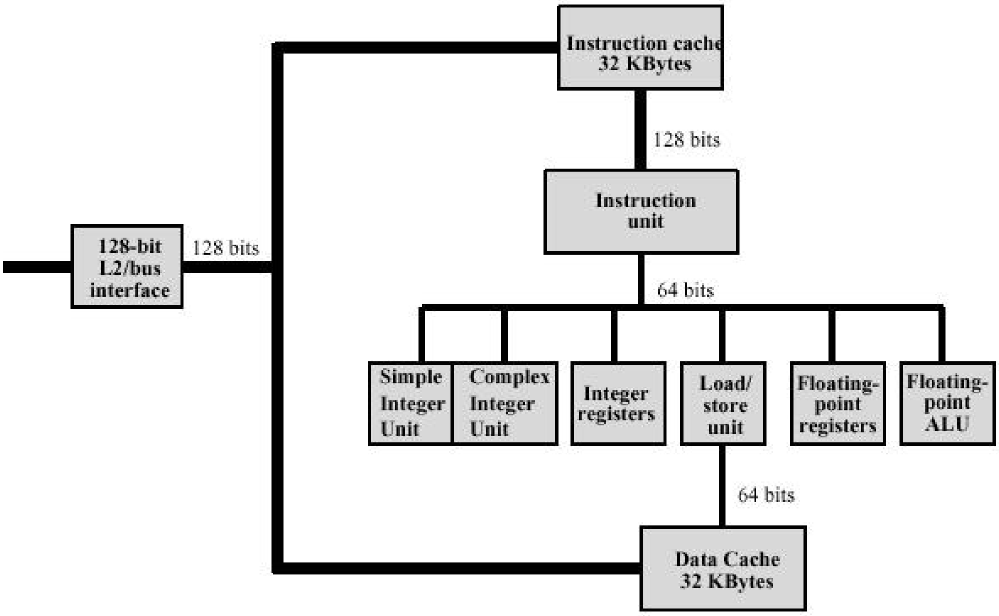


Figure 4.25 PowerPC G3 Block Diagram

- Code cache
  - Mostly ignored here -- see chap. 12 for detail
  - Read-only
- Data cache
  - uses a load/store unit to feed both floating point unit and any of the 3 parallel integer ALU's
  - Uses MESI, but adds Allocated (A) state - used when a block of data in a line is swapped out and replaced.

Advanced DRAM Organization (4.5)

- Fast Page Mode (FPM DRAM)
  - A row of memory cells (all selected by the same row address) is called a page
  - Only the first access in a page needs to have the row address lines precharged
  - Successive accesses in the same page require only precharging the column address lines
  - Supports bus speeds up to about 28.5Mhz (w/ 60ns DRAM's)
- Extended Data Out (EDO RAM)
  - Just like FPM DRAM, except that the output is latched into D flip-flops (instead of just being line transitions)
  - This allows row and/or column addresses for the next memory operation to be loaded in parallel with reading the output (because the flip-flops will not change until they receive a change signal)
  - Supports bus speeds up to about 40Mhz (w/ 60ns DRAM's)
- Burst EDO (BEDO RAM)
  - Allows bursting of sequential data, and independent generation of next addresses, so that only the first access needs row/column addresses from bus

- Supports bus speeds up to 66Mhz

- Enhanced DRAM
  - Developed by Ramtron
  - Integrates a small SRAM cache which stores contents of last 512-nibble row read
  - Refresh is in parallel to cache reads
  - dual ported - reads can be done in parallel with writes
- Cache DRAM
  - Developed by Mitsubishi
  - Similar to EDRAM, but:
    - uses a larger cache - 16K vs. 2K
    - uses a true cache, consisting of 64-bit lines
    - cache can also be used as a buffer to support the serial access of a block of data
- Synchronous DRAM
  - Developed jointly by several manufacturers
  - Standard DRAM is asynchronous
    - Memory controller watches for read request and address lines
    - After request is made, bus master must wait while DRAM responds
    - Bus master watches acknowledgment lines for operation to complete (and must wait in the meantime)
  - Synchronous DRAM moves data in an out in a set number of clock cycles, synchronized with the system clock, just like the processor
  - Other speedups
    - burst mode - after first access, no address setup or row/column line precharge time is needed
    - dual-bank internal architecture improves opportunities for on-chip parallelism
    - mode register allows burst length, burst type, and latency (between receipt of a read request and beginning of data transfer) to be customized to suit specific system needs
  - Current standard works with bus speeds up to 100Mhz (while bursting), or 75Mhz for so-called SDRAM Lite.
- Rambus DRAM
  - Developed by Rambus
  - Vertical package, all pins on one side, designed to plug into the RDRAM bus (a special high speed bus just for memory)
  - After initial 480 ns access time, provides burst speeds of 500 Mbps (compared w/ about 33 Mbps for asynchronous DRAM's)
- RamLink
  - Developed as part of the IEEE working group effort called Scalable Coherent Interface (SCI)
  - DRAM chips act as nodes in a ring network
  - Data is exchanged in packets
    - Controller sends a request packet to initiate mem transaction, containing cmd header, address, checksum, and data to be written (if a write). Extra data in cmd header allows more efficient access.
  - Supports a small or large number of DRAM's
  - Does not dictate internal DRAM structure