**Cache Memory**

It is possible to organize data across a memory hierarchy such that the percentage of accesses to each successively lower level is substantially less than that of the level above. Because memory references tend to cluster, the data in the higher level memory need not change very often to satisfy memory access requests.

Memory Hierarchy List

* Registers
* L1 Cache
* L2 Cache
* Main memory
* Disk cache
* Disk
* Optical
* Tape

**Locality of Reference** - During the course of the execution of a program, memory references tend to cluster. e.g. loops

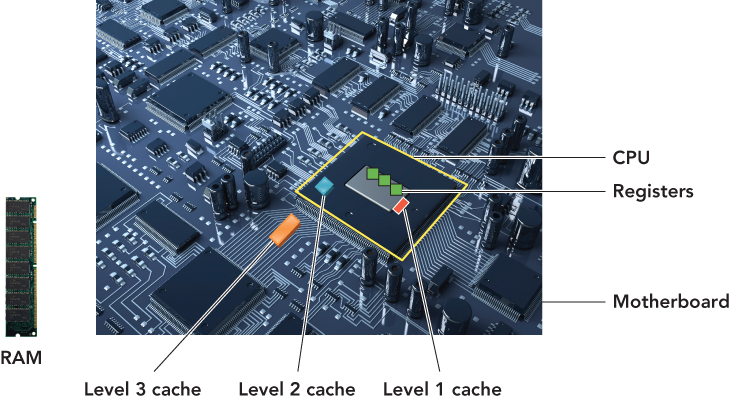
**Spatial locality** refers to the tendency of execution to involve a number of memory locations that are clustered. **Temporal locality** refers to the tendency for a processor to access memory locations that have been used recently.

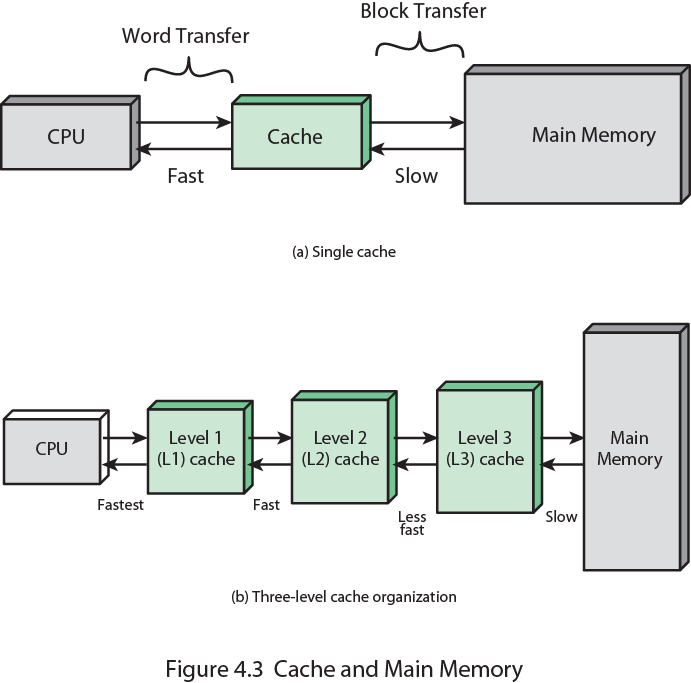
**Spatial locality** is generally exploited by using larger cache blocks and by incorporating prefetching mechanisms (fetching items of anticipated use) into the cache control logic.

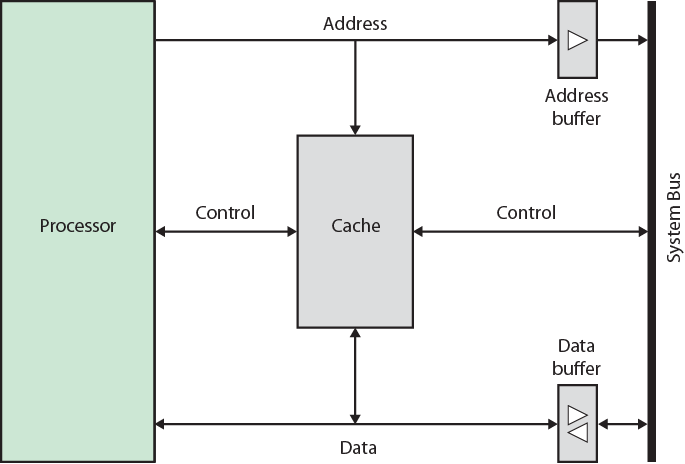
**Temporal locality** is exploited by keeping recently used instruction and data values in cache memory and by exploiting a cache hierarchy.

Cache

* + Small unit of ultrafast memory built into or near the processor
  + Used to store frequently or recently access program instructions or data
  + Faster than RAM, more expensive than RAM
* Sits between normal main memory and CPU
* May be located on CPU chip or module
  + Three levels of cache on a system:
    - Level 1 (L1) cache (primary cache)
    - Level 2 (L2) cache (secondary cache)
    - Level 3 (L3) cache

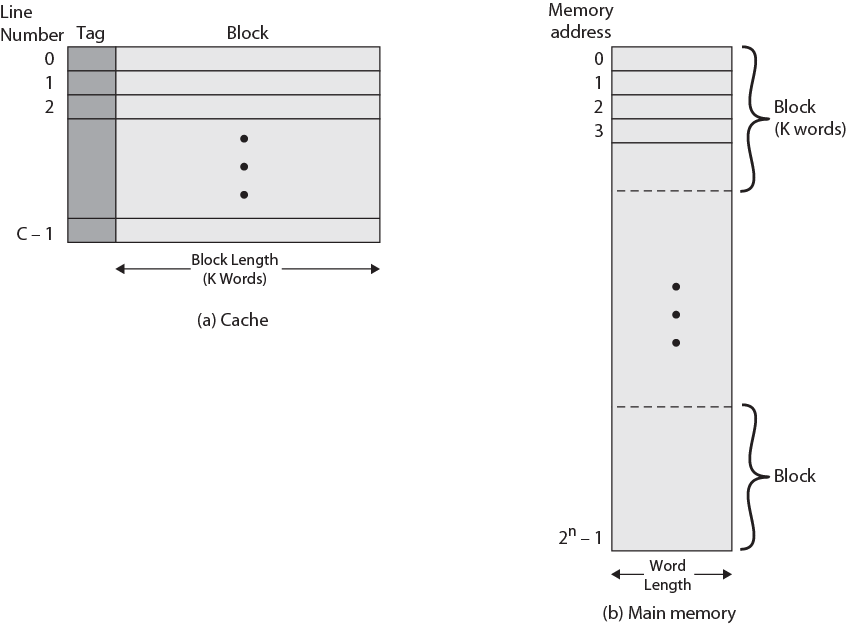


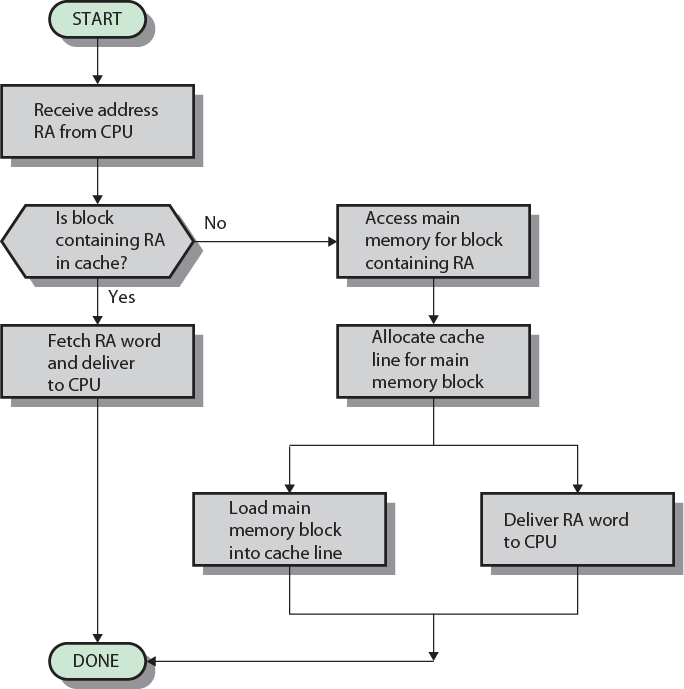




Cache operation

* 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
* Cache includes tags to identify which block of main memory is in each cache slot





**Design Issues**

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

Size

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

**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
* Address is in two parts
* Least Significant w bits identify unique word
* Most Significant s bits specify one memory block
* The MSBs are split into a cache line field r and a tag of s-r

Example:

cache size = 64KB (216 ); line size = 4 bytes (22 )

no. of lines in cache = 64Kb/4 = 16KB (214 )

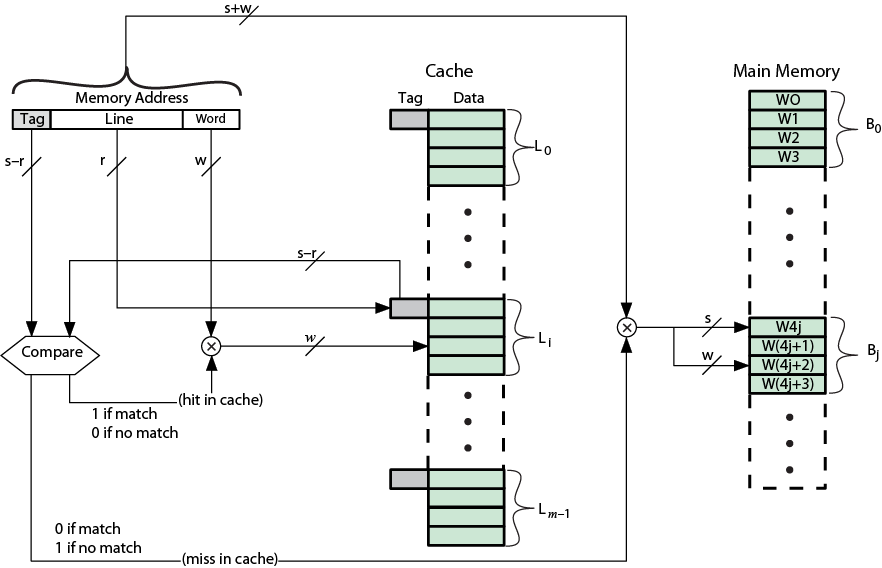
main memory size = 16MB (224 )

Address Structure

* 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 field
* Check contents of cache by finding line and checking Tag

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



**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
* Cache searching gets expensive

Address Structure (using previous example)

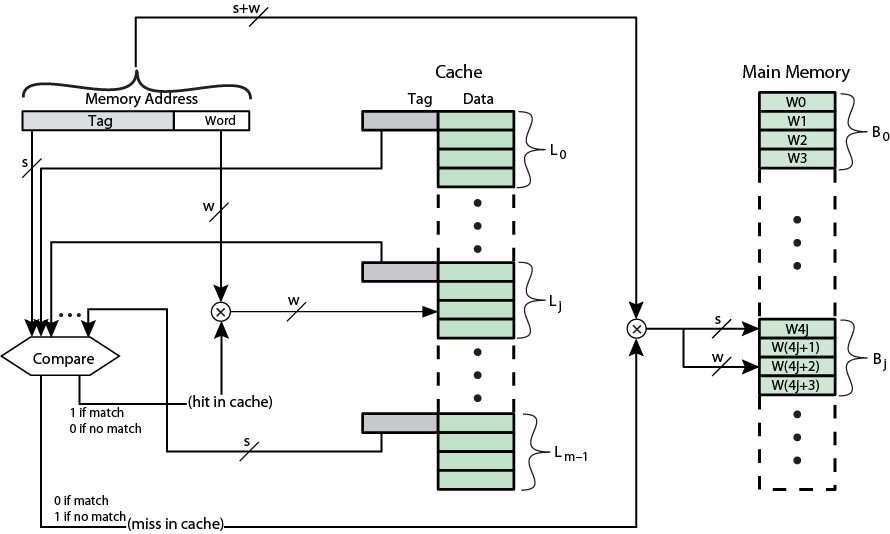
* 22 bit tag stored with each 32 bit block of data
* Compare tag field with tag entry in cache to check for hit
* Least significant 2 bits of address identify which 16 bit word is required from 32 bit data block
* e.g.
  + Address Tag Data Cache line
  + FFFFFC FFFFFC 24682468 3FFF

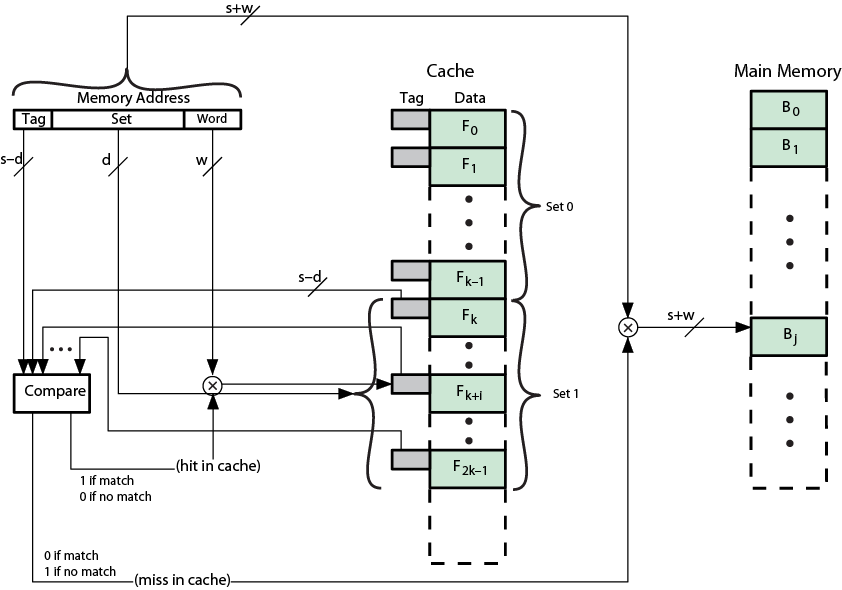
**Set Associative Mapping**

* Cache is divided into a number of sets
* Each set contains a number of lines
* A given block maps to any line in a given set
  + e.g. Block B can be in any line of set i
* e.g. 2 lines per set
  + 2 way associative mapping
  + A given block can be in one of 2 lines in only one set

Addressing

* Use set field to determine cache set to look in
* Compare tag field to see if we have a hit
* e.g
  + Address Tag Data Set number
  + 1FF 7FFC 1FF 12345678 1FFF
  + 001 7FFC 001 11223344 1FFF





**Replacement Algorithms**Direct mapping - No choice, Each block only maps to one line. Replace that line

Associative & Set Associative

* Hardware implemented algorithm (speed)
* Least Recently used (LRU) e.g. in 2-way set associative - Which of the 2 block is lru?
* First in first out (FIFO) - replace block that has been in cache longest
* Least frequently used - replace block which has had fewest hits
* Random

**Write Policy**

* 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

**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

**Line Size**

* Retrieve not only desired word but a number of adjacent words as well
* Increased block size will increase hit ratio at first
  + the principle of locality
* Hit ratio will decreases as block becomes even bigger
  + Probability of using newly fetched information becomes less than probability of reusing replaced
* Larger blocks
  + Reduce number of blocks that fit in cache
  + Data overwritten shortly after being fetched
  + Each additional word is less local so less likely to be needed
* No definitive optimum value has been found
* 8 to 64 bytes seems reasonable

**Multilevel Caches**

* High logic density enables caches on chip
  + Faster than bus access
  + Frees bus for other transfers
* Common to use both on and off chip cache
  + L1 on chip, L2 off chip in static RAM
  + L2 access much faster than DRAM or ROM
  + L2 often uses separate data path
  + L2 may now be on chip
  + Resulting in L3 cache
    - Bus access or now on chip…

**Unified v Split Caches**

* One cache for data and instructions or two, one for data and one for instructions
* Advantages of unified cache
  + Higher hit rate
  + Balances load of instruction and data fetch
  + Only one cache to design & implement

Advantages of split cache

* + Eliminates cache contention between instruction fetch/decode unit and execution unit
  + Important in pipelining