Cache Memory

Internal Memory
- **Processor Memory – Registers**
- **Cache Memory – L1, L2, L3**
- **Main Memory – Dynamic Random Access Memory (DRAM)**

Cost/Bit Decreases
Access Time Increases
Capacity Increases
Decreasing Frequency of Access

External Memory (Peripheral)
- **Fixed Hard Disk**
- **Removable Hard Disk**
- **Optical Disks – CD, DVD**
- **Solid State Memory -- Flash Drives, Memory Cards**
- **Magnetic Tape**

Principle of Locality of Reference
There is a high probability that future memory accesses are in the near vicinity of recent memory accesses

Unit of Transfer: number of electrical lines ←→ memory module

Word: (alternative definitions)
- **Unit of memory organization, i.e., the "natural" amount of memory that is retrievable in a single cycle**
- **Number of bits used to represent an integer or some multiple thereof**
- **Equal to the instruction length**
  - **Intel x86 word size 32 bits – instruction lengths vary**
  - **Cray C90 64 bit word, instruction length 46 bits**

Addressable Units
- **Word**
- **Byte**
  - **If (address length == A bits)**
    the number, N, of addressable units, i.e., bytes, is 2^A

Unit of Transfer
- **Main Memory -- number of memory bits read/written during a single event,**
  i.e., number of lines into and out of the memory module,
- **External Memory – blocks**

Access Methods
- **Sequential Access – magnetic tapes**
  - **sequential searching, counting, waiting**
  - **access time is highly variable**
- **Direct Access – disk drives**
  - **(track, sector) ➔ general vicinity**
  - **sequential searching, counting, waiting**
  - **access time is variable within strict limits**
- **Random Access – main memory**
  - **memory address ➔ data item**
  - **access time is**
    - **independent of previous searches**
    - **close to being constant**
- **Associative Access – cache memory**
  - **comparison of selected bit locations within a word, for a selected match, simultaneously**
  - **word is retrieved based on a portion of its contents rather than its address**
  - **access time is constant**

Performance

- **Access Time (Latency)**
  - **Random Access Memory**
    - **time differential: ((time that data is stored or made available) – (time address is presented))**
  - **Non-Random Access Memory**
    - **time differential: ((time that read-write mechanism is positioned at desired location) – (time that the (track, sector) is presented))**

- **Memory Cycle Time – System Bus**
  - **Access Time + time required for transient signals to die +**
    time required to regenerate data after destructive reads

- **Transfer Time -- rate at which data is transferred ⇐⇒ a memory unit**

  - **Random Access Memory**
    - **transfer time = 1/(cycle time)**

  - **Non-Random Access Memory**
    - **transfer time**

      $$T_N = T_A + \frac{n}{R}$$

      T_N : average time to read/write N bits
      T_A : average access time
      n : number of bits
      R : transfer rate measured in bits/second (bps)

- **Two Level Memory Access**

  Level 1 hold 1000 words with an access time 0.01 μs
  Level 2 holds 100,000 words with an access time 0.1 μs

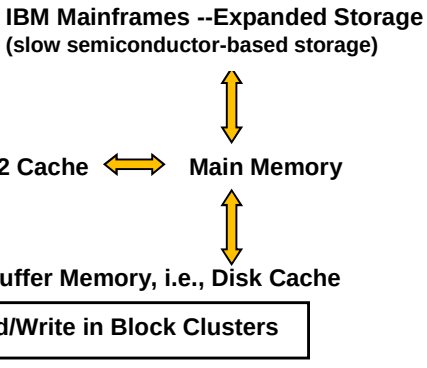
  $$hit\ ratio = \frac{\text{Level 1 memory hits}}{\text{total memory hits}}$$

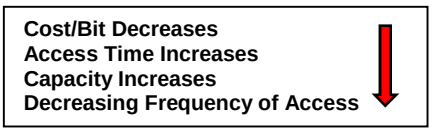
  if
  95% of the memory accesses are from Level 1
  5% of the memory accesses are from Level 2

  then the hit ratio is
  (0.95)(0.01 μs) + (0.05)(0.01 μs + 0.1 μs) ➔ 0.0095 μs + 0.0055 μs ➔ 0.015 μs

- **Locality of Reference**
  - **Instructions**
    - **Loops, Subroutines**
  - **Data**
    - **Tables, Arrays, Clustered Data Sets**

- **CPU ⟺ Memory Register Set (Cache) ⟺ L1 Cache ⟺ L2 Cache ⟺ Main Memory**





Performance Characteristics of Two-Level Memories

Spatial Locality – clustered memory locations

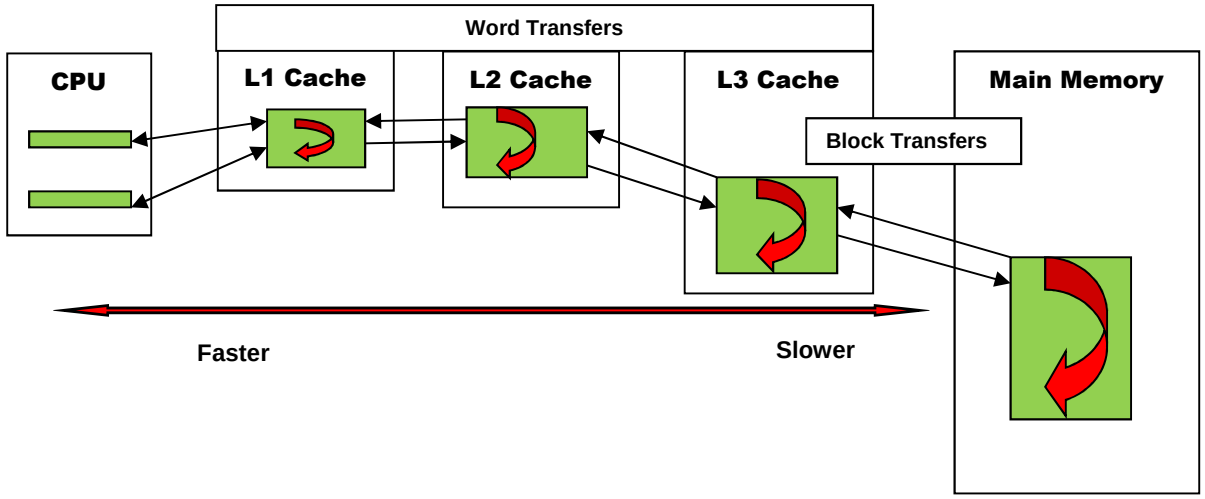
- **Sequential Instructions**
- **Sequential Data Locations**
- **Using larger cache blocks**
- **Incorporating prefetching mechanisms into the cache memory logic**

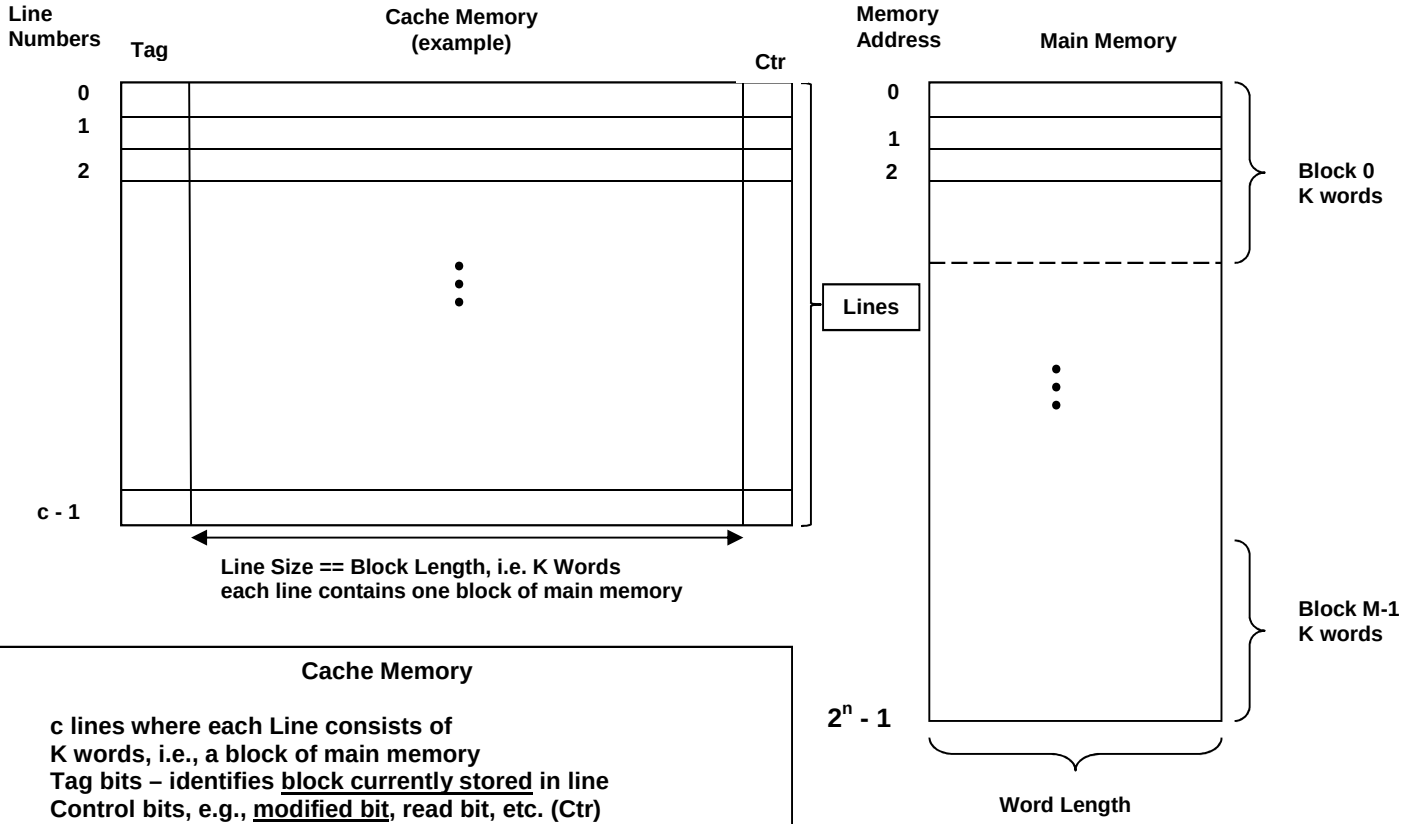
Temporal Locality – repeatedly access a set of memory locations

- **Loops**
- **Maintaining recently used data and instructions in cache memory**

Cache Memory

Locality of Reference – clustered sets of data/instructions





Cache Memory

c lines where each Line consists of
K words, i.e., a block of main memory
Tag bits – identifies block currently stored in line
Control bits, e.g., modified bit, read bit, etc. (Ctr)

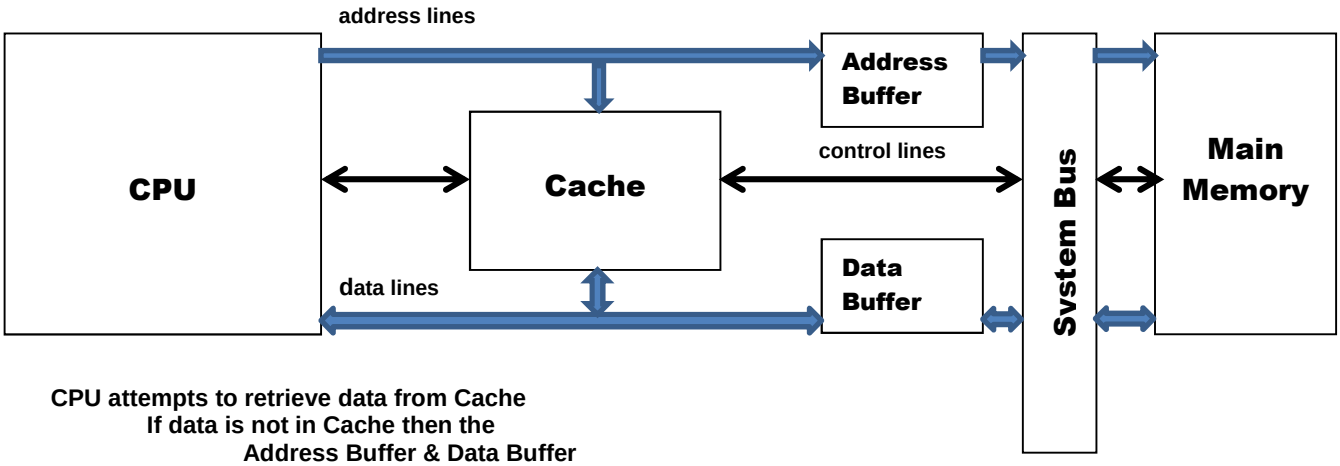
Byte addressable machines can have Lines as small as 32 bits, with 4 bytes/block

c << M

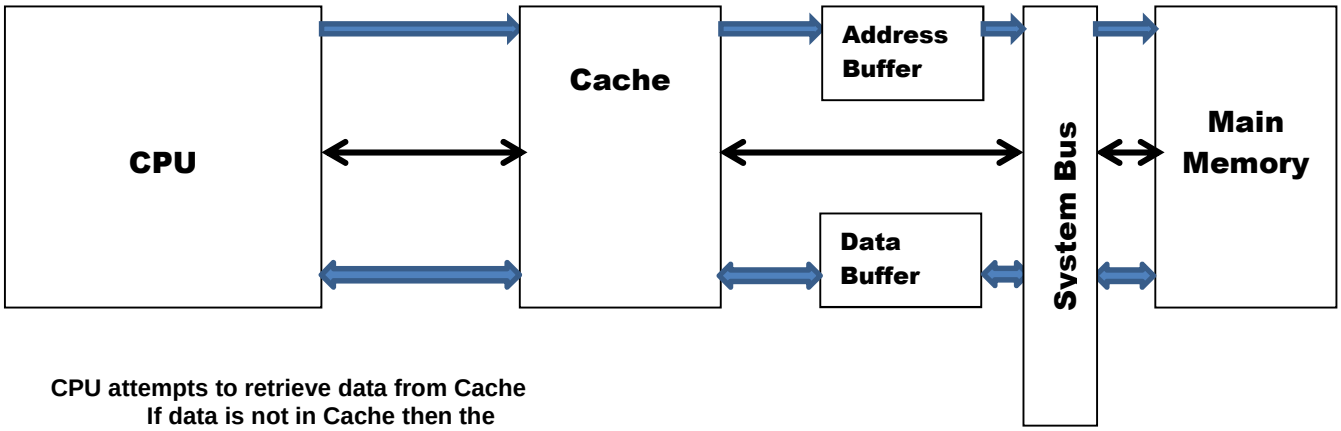
Main Memory

- 2^n addressable words
- each word has a unique n-bit address
- M blocks each consisting of K words
- M = $2^n / K$ i.e., M*K = 2^n

Alternative Organizations

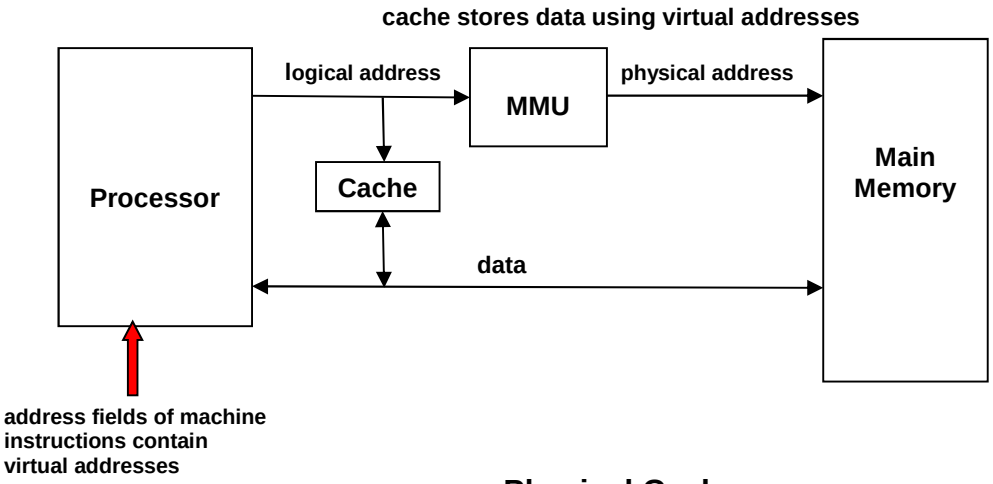


CPU attempts to retrieve data from Cache
If data is not in Cache then the
Address Buffer & Data Buffer
are used to access main memory via the system bus
the requested data item is returned to the CPU while the system
simultaneously returns the memory block to the Cache

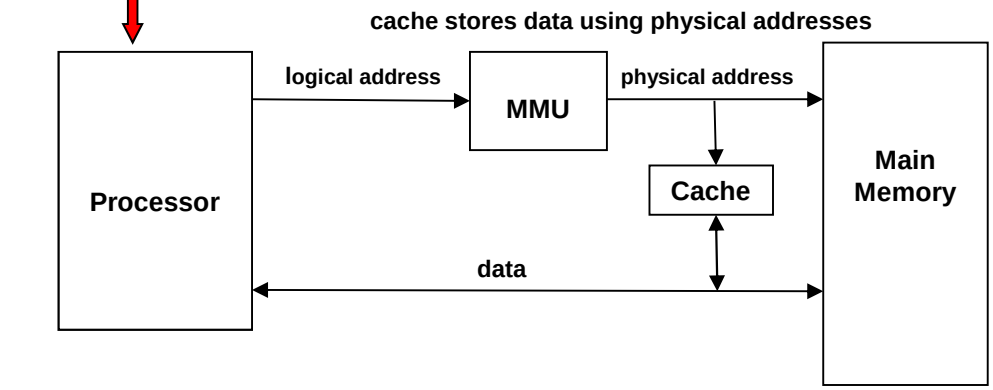


CPU attempts to retrieve data from Cache
If data is not in Cache then the
Address Buffer & Data Buffer are activated
to access main memory via the system bus;
1st the memory block is returned to the Cache and then
2nd the requested data item is returned to the CPU

Logical (Virtual) Cache

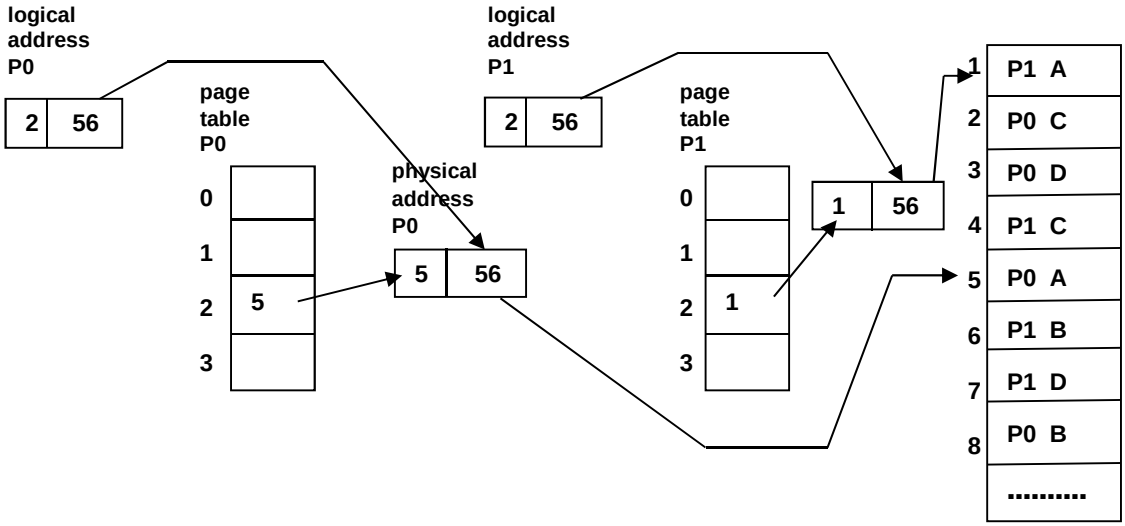


Physical Cache



Virtual Cache

- **delivers memory items to CPU faster, i.e., does not need to wait for Memory Management Unit**
- **most virtual memory systems start the pages at address zero; the same virtual address for two different processes refer to two different physical addresses; hence either the cache must be flushed with each process context switch or each line of the cache must contain additional bits to identify the correct virtual address space**



- **Logical Cache, i.e., Virtual Cache**
  Faster response time within each process
  Context switch ➔ flush cache memory
  OR
  add additional bits to each line which identifies the process that is using that line

- **Physical Cache**
  Slower response time since each cache access has to also invoke the MMU
  Context switch does not require that the cache memory be flushed

- **Cache Size Parameters**
  Small ➔ cost per bit of cache ≅ cost per bit of main memory
  Large ➔ overall average access time of (main + cache) memory ≅ access time of cache memory
  Δ↑ cache size ➔ Δ↑ logic gates ➔ Δ↓ cache access time
  chip & board size limits cache memory size

- **Mapping Function**
  Main Memory Blocks ➔ Cache Lines (number of cache lines << number of main memory blocks)
  Determining which memory block occupies a specific cache line

Assumptions for Direct, Associative & Set-Associative Mapping Functions

Page 125 Example 4.2

cache holds 64 KB
data transfer block size == 4 bytes
cache line size == 4 bytes
number of cache lines == 16K, i.e., 2^{14} lines of 4 bytes each

main memory == 16 MB; byte addressable; 24-bit address (2^{24} = 16 MB)
main memory == 4 MB blocks (4 bytes/block)

- **Direct Mapping**
  $i = j \bmod m$ where i : cache line number, j : main memory block number, m : number of cache lines

Cache Line	Main Memory Blocks Assigned
0	0, m, 2m, …, 2^s - m
1	1, m + 1, 2m + 1, … , 2^s – m + 1
2	2, m + 2, 2m + 2, … , 2^s – m + 2
…	…
…	…
…	…
m - 1	m – 1, 2m – 1, 3m – 1, … , 2^s - 1

Cache Access

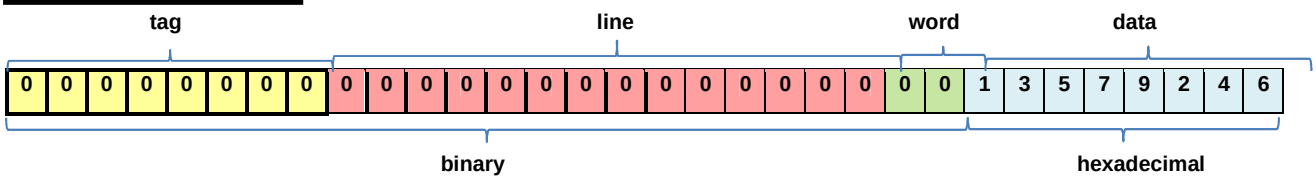
Page 128 Example 4.2a

m = 16K = 2^{14}

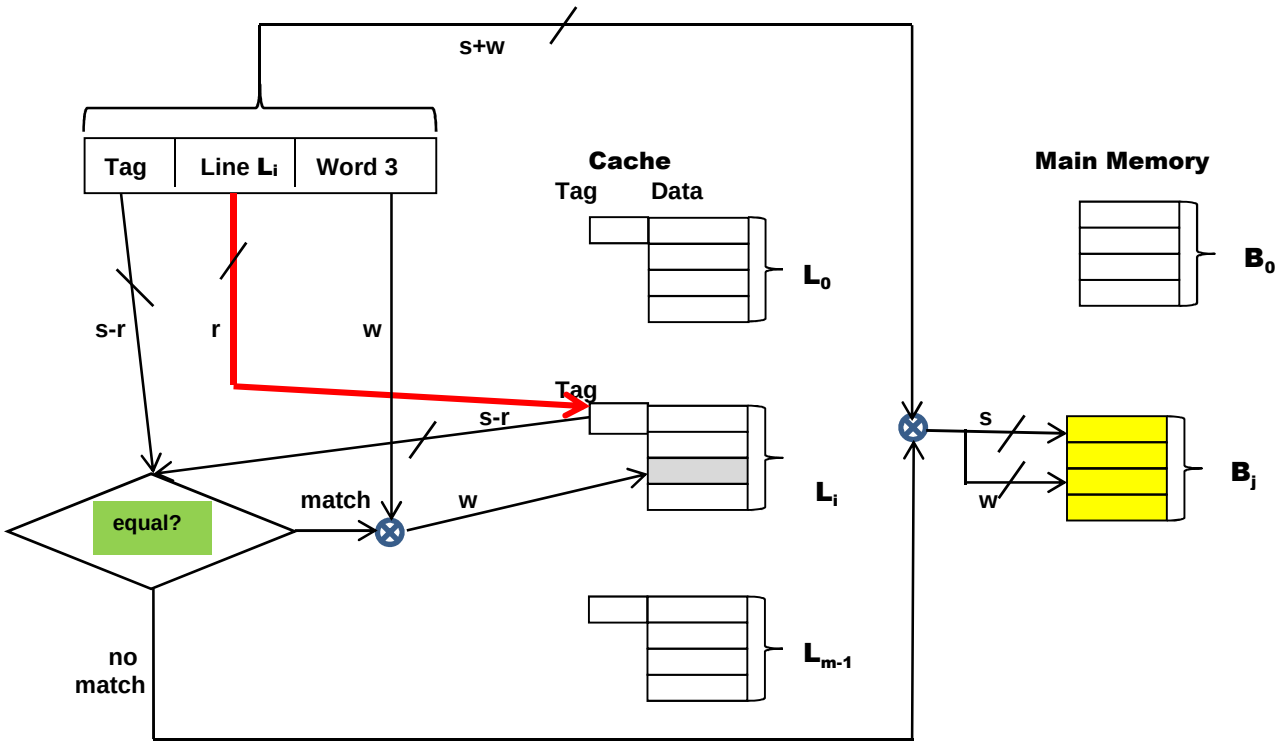
$i = j \bmod 2^{14}$

Cache Line	Block Address
0	000000, 010000, … , FF0000
1	000004, 010004, … , FF0004
…	…
2^{14} – 1	00FFFC, 01FFFC, … , FFFFFC

Page 129 Figure 4.10



If two different blocks have the same tag number they map onto different lines, e.g.,
Block 000101100011001110011100 containing FEDCBA98 maps to line 0CE7 with tag 16
Block 000101101111111111111100 containing 12345678 maps to line 3FFF with tag 16
Memory Address (24bits) ➔ Tag (8bits) + "Cache" Line (14) + "Byte" Cache Line Offset (2 bits)



w bits identifies unique byte/word within block
s bits specify a particular block in main memory; it is also used as the tag
r bits identifies a specified line of the cache

1. **Line number indexes into cache**
2. **Desired tag is compared with the tag value returned from the cache line**
3. **If equal, use the word value to index into the line to locate the desired word/byte**
4. **If not equal, use the tag value conjoined with the word value to locate the desired block in main memory**

Memory Address Length = s+w bits
Number of addressable units of memory (words or bytes) = 2^{s+w}
Block size == Line size == 2^w words or bytes

Number of blocks in Main Memory = $\frac{2^{s+w}}{2^w} = 2^s$
Number of lines in cache = m = 2^r
Size of cache = 2^{r+w} words or bytes
Size of tag = (s-r) bits

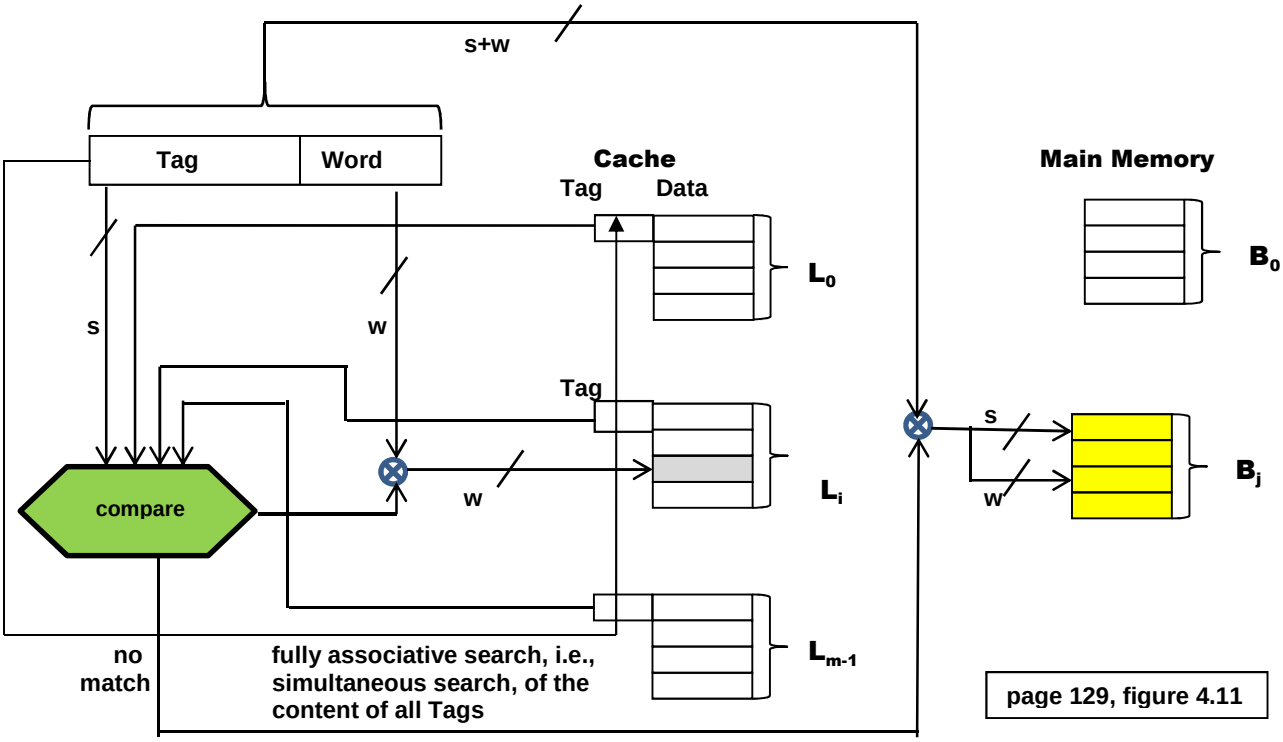
Implementation is inexpensive
If two different blocks map to the same line, repeated references to these two blocks will result in continual swapping of the blocks, i.e., thrashing!
Save swapped out line in a victim cache
Victim Cache – fully associative cache (4-16 cache lines) residing between direct mapped L1 cache & next level of memory

Associative Mapping

- each main memory block may be loaded into any line of the cache
- memory address ⟷ Tag + Word
  - Tag is used to identify a specific memory block
  - Word == offset within the line ; used to identify a specific word
- memory address does not identify the line
- number of lines is not determined by the address format
- control logic simultaneously examines every line's Tag for a match to the memory address's Tag

Memory Address Length = s+w bits
Number of addressable units of memory (words or bytes) = 2^{s+w}
Block size == Line size == 2^w words or bytes

Number of blocks in Main Memory = $\frac{2^{s+w}}{2^w} = 2^s$
Number of lines in cache = undetermined
Size of tag = s bits



page 129, figure 4.11

Tag field uniquely identifies a block of main memory
To determine whether the block is in the cache, the cache control logic must simultaneously examine every line's tag for a match

page 130
- Example 4.2b
- Figure 4.12
- Handouts

slides 38, 40, 41, 42, 43

Disadvantage – complex circuitry required to implement associative memory retrieval
Advantage – flexibility as to where a new block may be stored in the cache

Set Associative Mapping

- Set == Group of Lines
- Cache == Group of Sets

- k-way set-associative mapping function

$$m = v * k$$

$$i = j \bmod v$$

where
- i : cache set number
- j : main memory block number
- m : number of lines in the cache
- v : number of sets
- k : number of lines in each set

(v == m & k == 1) ➔ direct mapping
(v == 1 & k == m) ➔ associative mapping

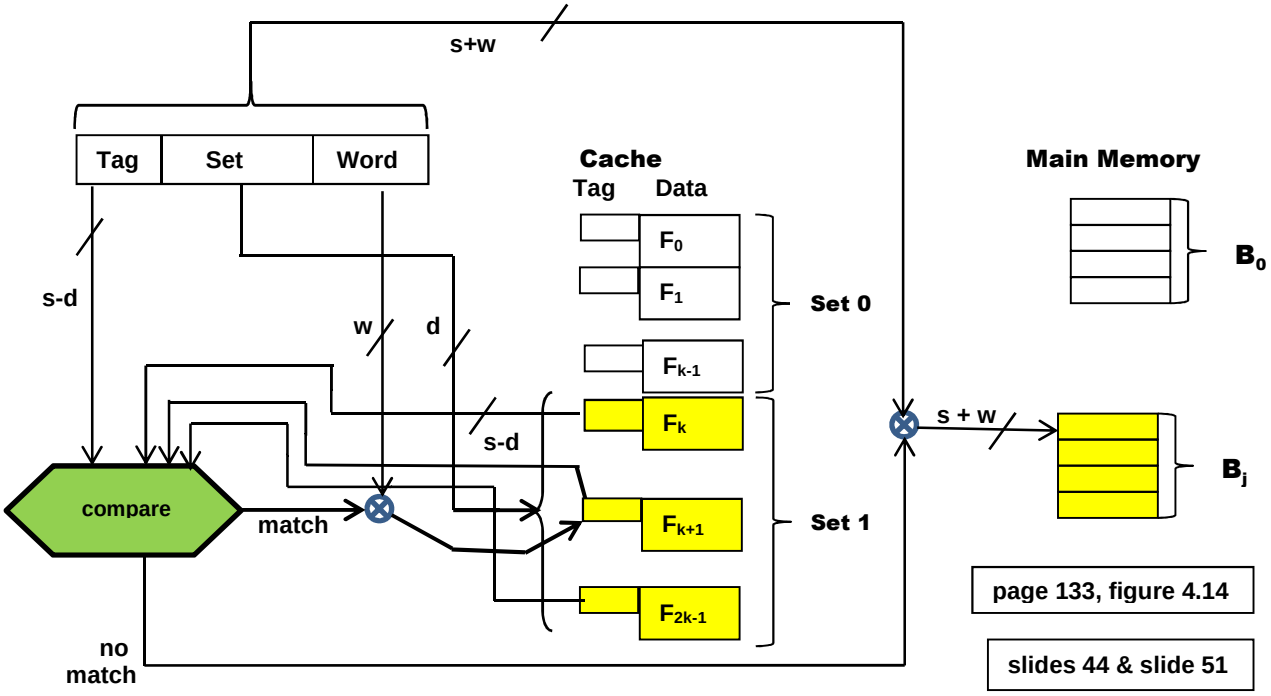
(v == m/2 & k == 2) ➔ 2-way set-associative cache memory
Significantly improves hit ratio over direct mapping

(v == m/4 & k == 4) ➔ 4-way set-associative cache memory
Modest improvement over 2-way for small additional cost

Block B_i can be mapped into any of the lines of the set i

- 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 = 2^d = v
- number of lines in cache = m = kv = $k*2^d$
- size of cache = $k*2^{d+w}$ words or bytes
- size of tag = s-d bits

- Physical implementation of a set-associative cache
  - v number of individual associative caches (sets)
  - k number of individual direct associative caches (lines in a set)
    - ✓ each direct associative cache ←➔ <u>way</u> which consists of v lines
    - ✓ first v blocks of main memory are directly mapped to the v lines of each <u>way</u>



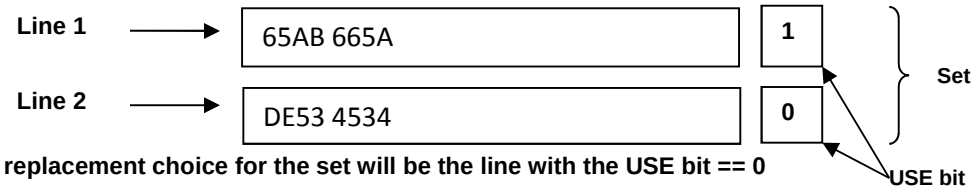
page 133, figure 4.14

slides 44 & slide 51

Replacement Algorithms

- **Direct Mapping --∀ block ∃! line, i.e.,**
  for every block there exists only one unique line, i.e., there is no choice

- **Associative & Set Associative**
  - **LRU, i.e., Least Recently Used – select block with longest time in cache with no memory references**
    - **2-way set-associative : each line has a USE bit – when a line is referenced its USE bit is set to 1 and the USE bit of the other line is set to 0**

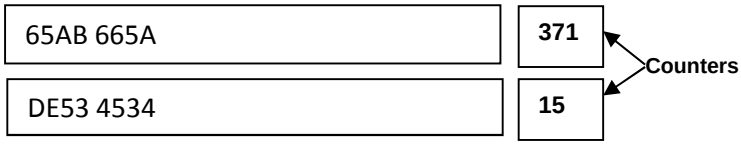


replacement choice for the set will be the line with the USE bit == 0

    - **n-way associative & associative : maintain a separate list of line numbers**
      when a line is referenced, its line number is placed at the front of the list
      replacement candidates are taken from the back of the list

  - **FIFO, i.e., first-in, first-out – select block with longest time in cache**
    round-robin -- circular buffer

  - **LRU, i.e., least recently used – select block with fewest references**
    each line has a counter to record the number of references



  - **Random choice – select block using a random number generator**

Write Policy

- **old block in cache has not been altered ➔ overwrite old block with new block**
- **old block in cache has been altered ➔**
  write old block to main memory before overwriting with the new block

- **Write-Through – all write operations change both the cache and the main memory**
  any other processor-cache module must monitor main memory traffic to maintain consistency
  <u>traffic bottleneck</u>

- **Write-Back – updates are written only to the cache**
  - **when line is modified it's dirty bit is set**
  - **when the line is selected for replacement,**
    the line needs to be written to main memory only if it's dirty bit is set

<u>portions of main memory are invalid</u>
<u>access of I/O modules must be through the cache</u>
<u>complex circuitry & potential bottleneck</u>

Percentage of Memory Reference Writes, i.e., $\frac{\text{write memory references}}{\text{total memory references}}$

- **Normal computation 15%**
- **vector-vector multiplication 33%**
- **matrix transposition 50%**

Bus Organization

multiple devices may each have a cache memory module
even with write-through, multiple caches may hold invalid data

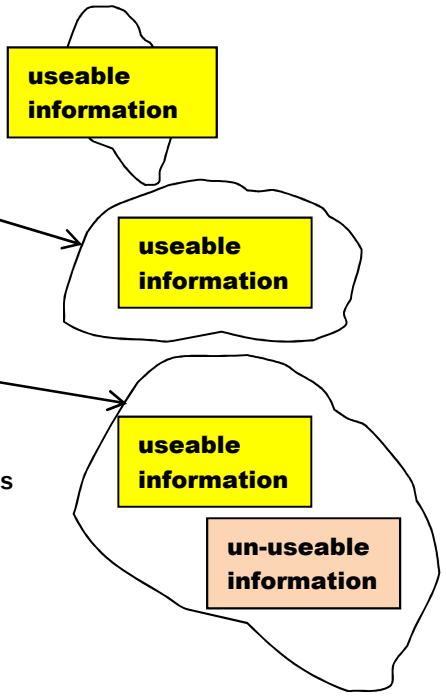
- Bus Watching with Write-Through
  - each cache controller monitors write operations to main memory by other bus masters
  - if a write is to a location in main memory that also resides in the local cache, the local cache data items are invalidated

- Hardware Transparency
  - All cache updates are written to main memory; any matching words in other caches are also updated

- Non-cacheable Memory
  - Shared memory
    - Is that portion of memory shared by more than one processor
    - declared to be non-cacheable, i.e., it is never copied into cache memory
    - all references to shared memory are cache misses

Principle of Locality

Block Size too small →



Up to a certain point: Block Size Δ ↑ ➔ Hit Ratio ↑

After that point: Block Size Δ ↑ ➔ Hit Ratio ↓

Optimal Line Size

- Normal Computation — 8 Bytes ≤ Line Size ≤ 64 Bytes
- High Performance — 64 Bytes ≤ Line Size ≤ 128 bytes

Multilevel Caches

- L1 : on-chip cache
  - bus access is eliminated → faster data retrieval
  - shorter data paths → faster data retrieval
- L2 : external cache, i.e., off-chip cache
  - processor does not need to engage in bus access to DRAM nor ROM
  - If L2 is implemented as a SRAM, i.e., static RAM chip, which matches the bus speed then data can be accessed using a zero-wait state transaction
  - separate data path may be used to reduce traffic on the data bus
- L2 may be placed on-chip yielding faster response times
  - L3 off-chip
- L3 on-chip

Page 139 figure 4.17 Simulation Study Two-level Cache Performance vs. Cache Size
L2 must be twice L1 size for Performance Increase

Split Caches – parallel instruction execution, prefetching predicted future instructions

- **Data Caches**
- **Instruction Caches**

Unified Cache – higher hit ratio –

Execution pattern

More instruction fetches than data fetches ➔ cache fills with instructions
More data fetches than instruction fetches ➔ cache fills with data

Pentium 4 Cache Organization

- **80386 cache exists but not on chip**
- **80486 single on-chip cache 8 KB, line size 16 bytes, 4-way set associative**
- **Pentium data on-chip L1 cache**
  instruction on-chip L1 cache

- **Pentium II L1 data cache 16 KB, line size 64 bytes, 4-way set associative, on-chip**
  L1 instruction cache, on-chip … see below
  L2 cache 512 KB, line size 128 bytes, 8-way associative, on-chip
  feeds both L1 cache memories

- **Pentium III L1 data cache 16 KB, line size 64 bytes, 4-way set associative, on-chip**
  L1 instruction cache, on-chip … see below
  L2 cache 512 KB, line size 128 bytes, 8-way associative, on-chip
  feeds both L1 cache memories
  L3 cache

- **Pentium 4 L1 data cache 16 KB, line size 64 bytes, 4-way set associative, on-chip**
  L1 instruction cache, on-chip, line size 128 bytes, 8-way associative, on-chip
  L2 cache 512 KB, line size 128 bytes, 8-way associative, on-chip
  feeds both L1 cache memories
  L3 cache, on-chip

  Fetch/Decode Unit
  - ✓ **Fetch instructions in order**
  - ✓ **Decode into micro-instructions**
  - ✓ **Store in L1 instruction cache**

  Out-Of-Order Execution Unit
  - ✓ **Schedules micro-instruction executions subject to**
    - ➢ **Data dependencies**
    - ➢ **Resource availability**
  - ✓ **Schedules speculative execution of micro-instructions**

  Execution Units
  - ✓ **Fetch required data from L1 data cache**
  - ✓ **Execute micro-instructions**
  - ✓ **Store results in registers**

  Memory
  - ✓ **L2 cache, L3 cache, System Bus, Main Memory**

  Data Cache
  - ✓ **Write-Back Policy – data written to main memory only when**
    - ➢ **removed from cache**
    - ➢ **data is modified**