William Stallings Computer Organization and Architecture

Chapter 4

Internal Memory

Characteristics

- Location
  - CPU, Internal, External
- Capacity
  - Word size, number of words
- Unit of transfer
  - Word on bus, block, cluster
- Access method
  - Direct, Random, Associative, Sequential
- Performance
  - Access, Cycle, Transfer time
- Physical type
  - Semiconductor, magnetic, optical
- Physical characteristics
  - Volatile , Erasable
- Organization
  - Physical arrangement of bits into words

Access Methods (1)

- Sequential
  - Start at the beginning and read through in order
  - Access time depends on location of data and previous location
  - e.g. tape
- Direct
  - Individual blocks have unique address
  - Access is by jumping to vicinity plus sequential search
  - Access time depends on location and previous location
  - e.g. disk

Access Methods (2)

- Random
  - Individual addresses identify locations exactly
  - Access time is independent of location or previous access
  - e.g. RAM
- Associative
  - Data is located by a comparison with contents of a portion of the store
  - Access time is independent of location or previous access
  - e.g. cache

Memory Hierarchy

- Registers
  - In CPU
- Internal or Main memory
  - May include one or more levels of cache
  - "RAM"
- External memory
  - Backing store

Performance

- Access time
  - Time between presenting the address and getting the valid data
- Memory Cycle time
  - Time may be required for the memory to "recover" before next access
  - Cycle time is access + recovery
- Transfer Rate
  - Rate at which data can be moved

Physical Characteristics

- Decay
- Volatility
- Erasable
- Power consumption

The Bottom Line

- How much?
  - Capacity
- How fast?
  - Time is money
- How expensive?

- Tradeoffs among all of these
  - E.g. Faster = More expensive, More = Less cost (per bit) but slower
  - Solution : Memory Hierarchy

Hierarchy List

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

- As one goes down the hierarchy
  - Decreasing cost per bit
  - Increasing capacity
  - Increasing access time
  - Decreasing frequency of access of the memory by the processor – locality of reference

So you want fast?

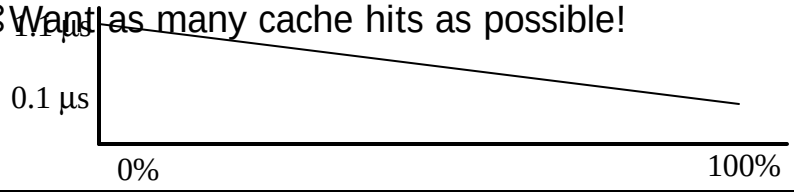
- It is possible to build a computer which uses only static RAM (see later)
- This would be very fast
- This would need no cache
  - How can you cache cache?
- This would cost a very large amount

Locality of Reference

- Temporal Locality
  - Programs tend to reference the same memory locations at a future point in time
  - Due to loops and iteration, programs spending a lot of time in one section of code
- Spatial Locality
  - Programs tend to reference memory locations that are near other recently-referenced memory locations
  - Due to the way contiguous memory is referenced, e.g. an array or the instructions that make up a program
- Locality of reference does not always hold, but it usually holds

Cache Example

- Consider a Level 1 cache capable of holding 1000 words with a 0.1 μs access time. Level 2 is memory with a 1 μs access time.
- If 95% of memory access is in the cache:
  - T=(0.95)*(0.1 μs) + (0.05)*(0.1+1 μs) = 0.15 μs
- If 5% of memory access is in the cache:
  - T=(0.05)*(0.1 μs) + (0.95)*(0.1+1 μs) = 1.05 μs
- Want as many cache hits as possible!



Semiconductor Memory

- RAM
  - Misnamed as all semiconductor memory is random access
  - Read/Write
  - Volatile
  - Temporary storage
  - Two main types: **Static** or **Dynamic**

Dynamic RAM

- Bits stored as charge in capacitors
- Charges leak
- Need refreshing even when powered
- Simpler construction
- Smaller per bit
- Less expensive
- Need refresh circuits (every few milliseconds)
- Slower
- Main memory

Static RAM

- ⌘ Bits stored as on/off switches via flip-flops
- ⌘ No charges to leak
- ⌘ No refreshing needed when powered
- ⌘ More complex construction
- ⌘ Larger per bit
- ⌘ More expensive
- ⌘ Does not need refresh circuits
- ⌘ Faster
- ⌘ Cache

Read Only Memory (ROM)

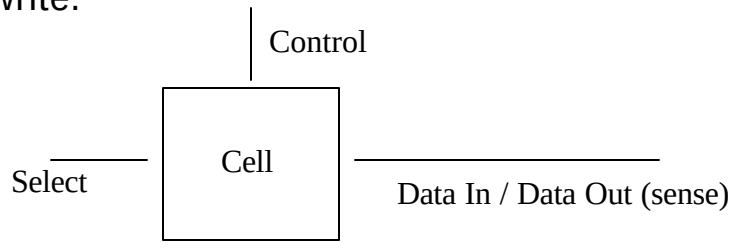
- ⌘ Permanent storage
- ⌘ Microprogramming
- ⌘ Library subroutines
- ⌘ Systems programs (BIOS)
- ⌘ Function tables

Types of ROM

- Written during manufacture
  - Very expensive for small runs
- Programmable (once)
  - PROM
  - Needs special equipment to program
- Read “mostly”
  - Erasable Programmable (EPROM)
    - Erased by UV
  - Electrically Erasable (EEPROM)
    - Takes much longer to write than read
  - Flash memory
    - Erase whole memory electrically

Chip Organization

- Consider an individual memory cell. Select line indicates if active, Control line indicates read or write.

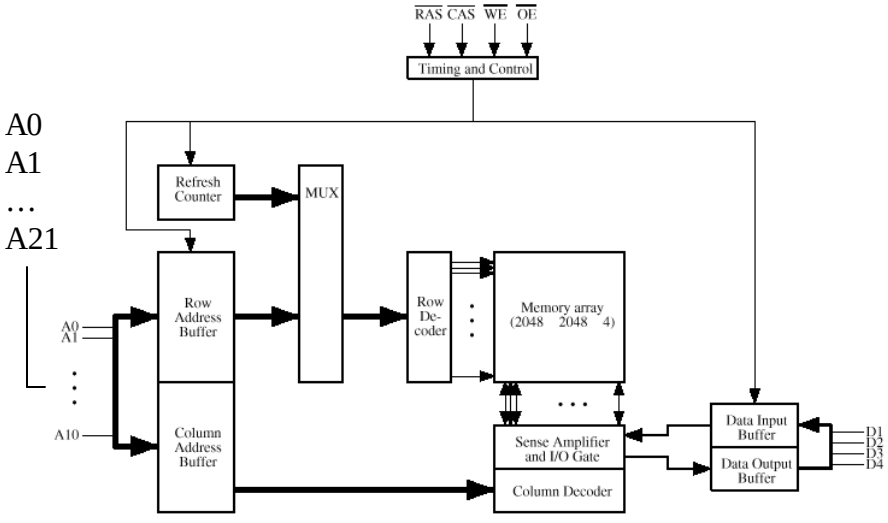


Memory Cell Operations

Organization in detail

- Some possible ways to create a 16Mbit chip
  - 1M of 16 bit words
  - 16 1Mbit chips, one chip for each bit of the desired 16 bit word
  - A 2048 x 2048 x 4bit array. Consider a 4 bit word size, so 4,194,304 addressable locations
    - Reduces number of address pins
    - Multiplex row address and column address
    - This example: 11 pins to address (2^{11}=2048), multiplex over the pins twice to get 22 bits (2^{22} = 4M) for each 4 bit word
    - To access memory, first send an address for the row (RAS), then send the address for the column (CAS). Together this activates the SELECT line. Need four lines for the Data In/Sense lines.
    - Adding one more pin doubles range of values so 4 times the capacity as we increase the dimensions

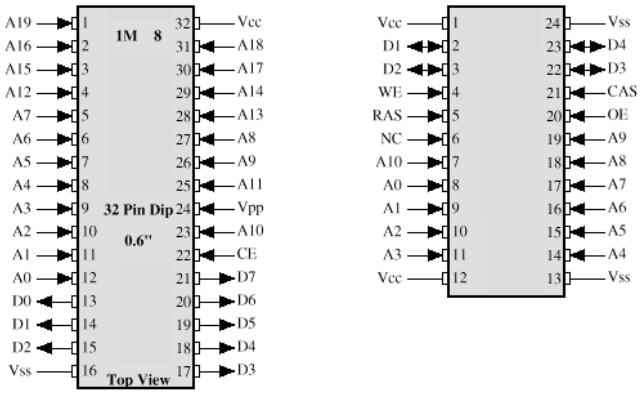
Typical 16 Mb DRAM (4M x 4)



Refreshing

- Refresh circuit included on chip
- Disable chip
- Count through rows
- Read & Write back
- Takes time
- Slows down apparent performance

Packaging

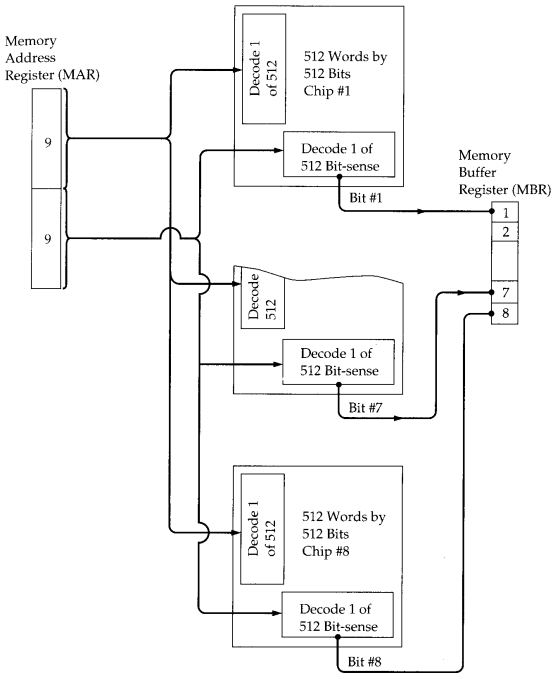


(a) 8 Mbit EPROM (b) 16 Mbit DRAM

CE = Chip Enable, Vss = Ground, Vcc=+V, OE = Output Enable, WE = Write Enable

Module Organization

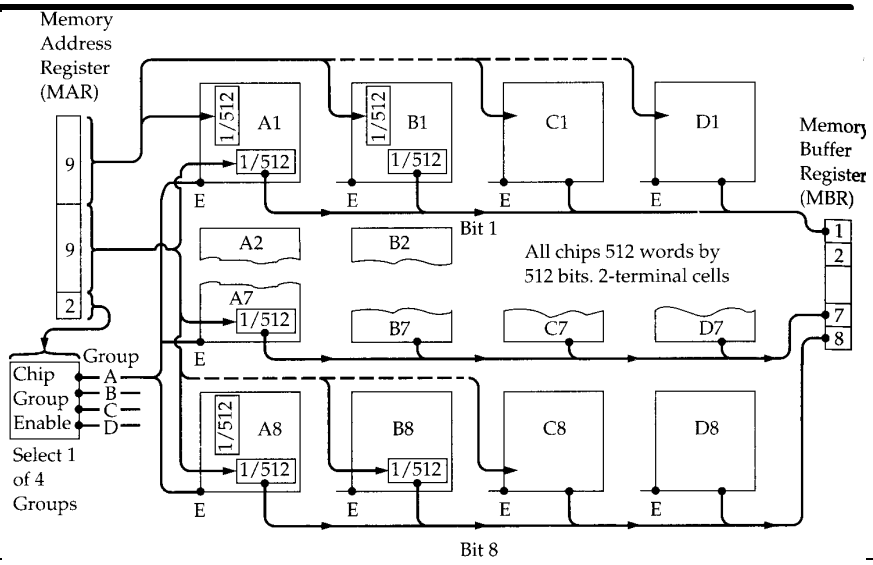
- Alternate Organization Using Modules to reference 256K 8 bit words
- 8 256K chip for each bit of the desired 8 bit word
- Full 18 bit address presented to each module, a single bit output. Data distributed across all chips for a single word



Module Organization – Larger Memories

- Can piece together existing modules to make even larger memories
- Consider previous 256K x 8bit system
  - If we want 1M of memory, can tie together four of the 256K x 8bit modules
  - How to tell which of the four modules contains the data we want?
  - Need 20 address lines to reference 1M
    - Use lower 18 bits to reference address as before
    - Use higher 2 bits into the Chip Select to enable only one of the four memory modules

Module Organization (2)

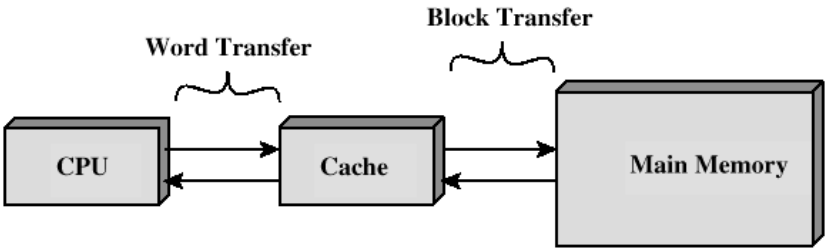


Error Correction

- Hard Failure
  - Permanent defect
- Soft Error
  - Random, non-destructive
  - No permanent damage to memory
- Hamming error correcting code one technique for detecting errors
  - Similar to parity bit, but contains enough information to correct data with single bit errors

Cache

- Small amount of fast memory
- Sits between normal main memory and CPU
- May be located on CPU chip or module

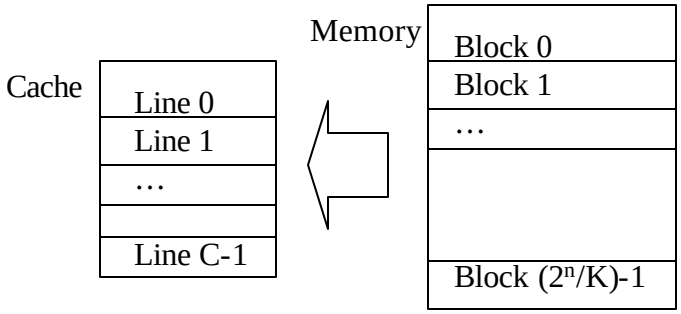


Cache operation - overview

- 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

Cache Design

- If memory contains 2^n addressable words
  - Memory can be broken up into blocks with K words per block. Number of blocks = 2^n / K
  - Cache consists of C **lines** or **slots**, each consisting of K words
  - C << M
  - How to map blocks of memory to lines in the cache?



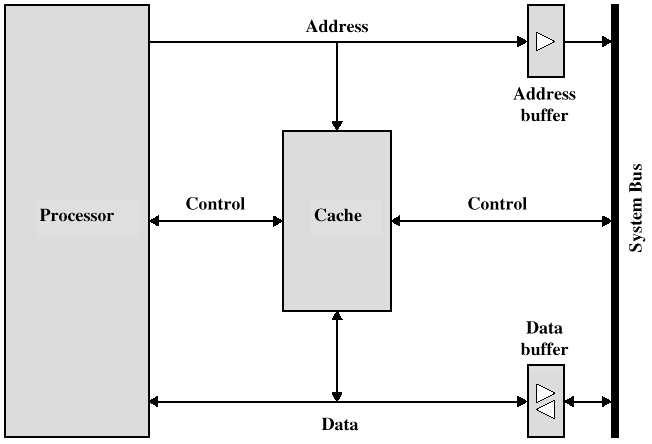
Cache Design

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

Size does matter

- Cost
  - More cache is expensive
- Speed
  - More cache is faster (up to a point)
  - Checking cache for data takes time
    - Adding more cache would slow down the process of looking for something in the cache

Typical Cache Organization



Mapping Function

- We'll use the following configuration example
  - Cache of 64KByte
  - Cache line / Block size is 4 bytes
    - i.e. cache is 16,385 (2^{14}) lines of 4 bytes
  - Main memory of 16MBytes
    - 24 bit address
    - (2^{24}=16M)
    - 16Mbytes / 4bytes-per-block → 4 MB of Memory Blocks
  - Somehow we have to map the 4Mb of blocks in memory onto the 16K of lines in the cache. Multiple blocks will have to map to the same line in the cache!

Direct Mapping

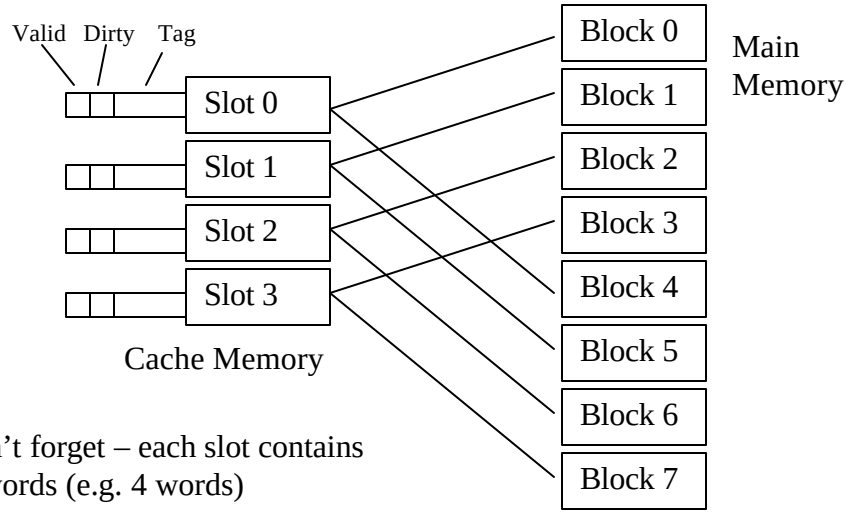
- Simplest mapping technique - each block of main memory maps to only one cache line
  - i.e. if a block is in cache, it must be in one specific place
- Formula to map a memory block to a cache line:
  - i = j mod c
    - i=Cache Line Number
    - j=Main Memory Block Number
    - c=Number of Lines in Cache

Direct Mapping with C=4

- Shrinking our example to a cache line size of 4 slots (each slot/line/block still contains 4 words):

Cache Line	Memory Block Held
0	0, 4, 8, …
1	1, 5, 9, …
2	2, 6, 10, …
3	3, 7, 11, …
In general:	
0	0, C, 2C, 3C, …
1	1, C+1, 2C+1, 3C+1, …
2	2, C+2, 2C+2, 3C+2, …
3	3, C+3, 2C+3, 3C+3, …

Direct Mapping with C=4

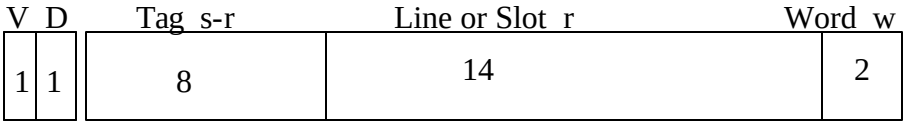


Don't forget – each slot contains K words (e.g. 4 words)

Direct Mapping Address Structure

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

Direct Mapping Address Structure



- Given a 24 bit address (to access 16Mb)
- 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
- Also need a Valid bit and a Dirty bit
  - Valid – Indicates if the slot holds a block belonging to the program being executed
  - Dirty – Indicates if a block has been modified while in the cache. Will need to be written back to memory before slot is reused for another block

Direct Mapping Example, 64K Cache

Cache Memory

Addr	Tag	W0	W1	W2	W3
0	00	F1	F2	F3	F4
1	1B	11	12	13	14
2					
3					
4					
5					
..					
..					
2^{14}-1					

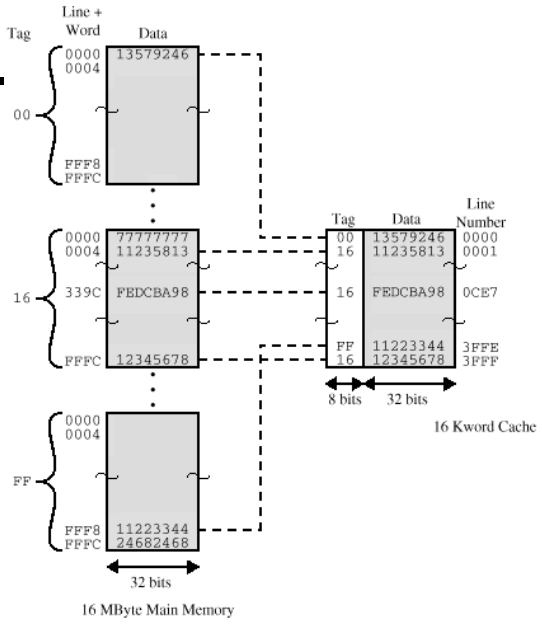
Main Memory

	Addr (hex)	Data
Line 0	000000	F1
	000001	F2
	000002	F3
	000003	F4
Line 1	000004	AB
	…	
Line 1	1B0004	11
	1B0005	12
	1B0006	13
	1B0007	14

1B0007 = 0001 1011 0000 0000 0000 0111
Word = 11, Line = 0000 0000 0000 01, Tag= 0001 1011

Direct Mapping Example

Original Example, 64K Cache with 4 words per Block



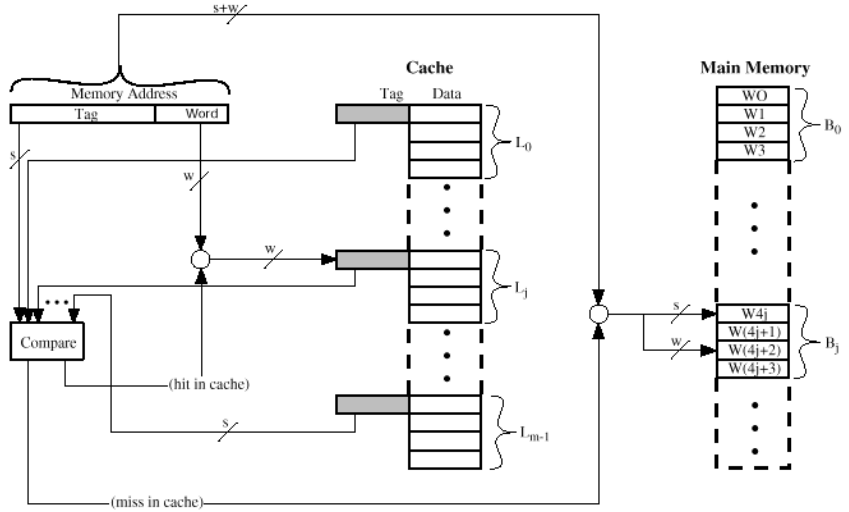
Direct Mapping pros & cons

- 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 – condition called **thrashing**

Fully Associative Mapping

- A fully associative mapping scheme can overcome the problems of the direct mapping scheme
  - 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
  - Also need a Dirty and Valid bit (not shown in examples)
- But Cache searching gets expensive!
  - Ideally need circuitry that can simultaneously examine all tags for a match
  - Lots of circuitry needed, high cost
- Need replacement policies now that anything can get thrown out of the cache (will look at last)

Fully Associative Cache Organization

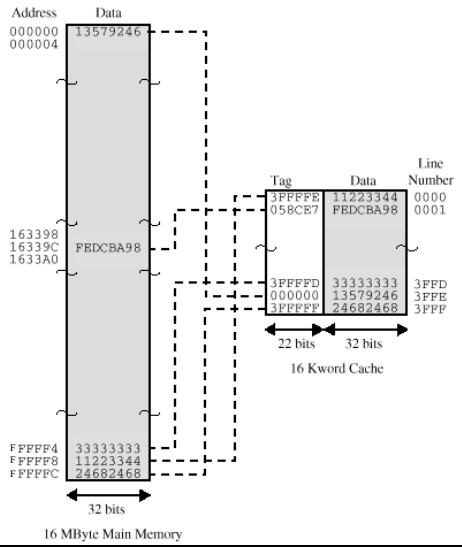


Associative Mapping Address Structure

Tag 22 bit	Word 2 bit

- 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 8 bit word is required from 32 bit data block
- e.g.
  - Address: FFFFFC = 1111 1111 1111 1111 1111 1100
    - Tag: Left 22 bits, truncate on left:
      - 11 1111 1111 1111 1111 1111
      - 3FFFFF
  - Address: 16339C = 0001 0110 0011 0011 1001 1100
    - Tag: Left 22 bits, truncate on left:
      - 00 0101 1000 1100 1110 0111
      - 058CE7

Associative Mapping Example

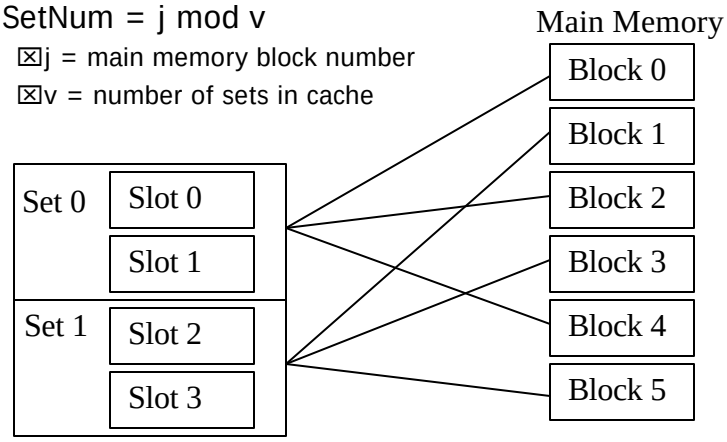


Set Associative Mapping

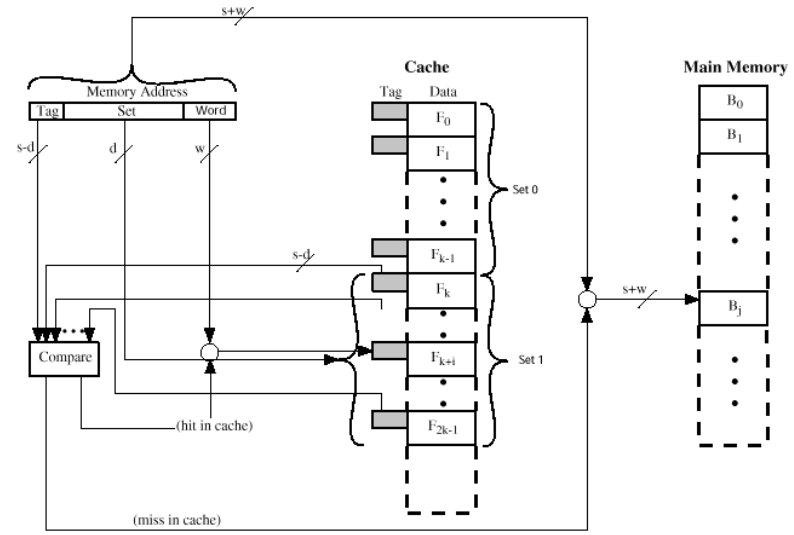
- Compromise between fully-associative and direct-mapped cache
  - Cache is divided into a number of sets
  - Each set contains a number of lines
  - A given block maps to any line in a specific set
    - Use direct-mapping to determine which set in the cache corresponds to a set in memory
    - Memory block could then be in any line of that set
  - e.g. 2 lines per set
    - 2 way associative mapping
    - A given block can be in one of 2 lines in a specific set
  - e.g. K lines per set
    - K way associative mapping
    - A given block can be in one of K lines in a specific set
    - Much easier to simultaneously search one set than all lines

Set Associative Mapping

- To compute cache set number:
  - SetNum = j mod v
    - j = main memory block number
    - v = number of sets in cache



Two Way Set Associative Cache Organization

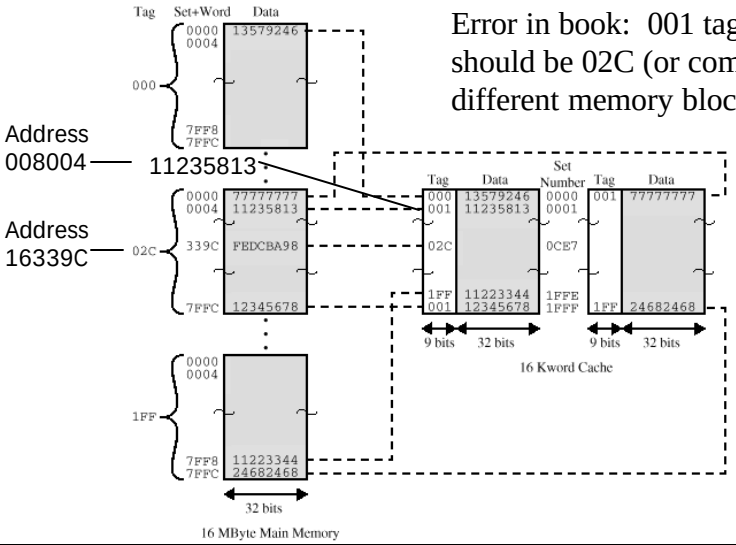


Set Associative Mapping Address Structure

Tag 9 bit	Set 13 bit	Word 2 bit

- E.g. given a 13 bit set number for 24 bit address
- Use set field to determine cache set to look in
- Compare tag field of all slots in the set to see if we have a hit, e.g.:
  - Address = 16339C = 0001 0110 0011 0011 1001 1100
    - Tag = 0 0010 1100 = 02C
    - Set = 0 1100 1110 0111 = 0CE7
    - Word = 00 = 0
  - Address = 008004 = 0000 0000 1000 0000 0000 0100
    - Tag = 0 0000 0001 = 001
    - Set = 0 0000 0000 0001 = 0001
    - Word = 00 = 0

Two Way Set Associative Mapping Example



Error in book: 001 tag in cache should be 02C (or come from a different memory block!)

K-Way Set Associative

- Two-way set associative gives much better performance than direct mapping
  - Just one extra slot avoids the thrashing problem
- Four-way set associative gives only slightly better performance over two-way
- Further increases in the size of the set has little effect other than increased cost of the hardware!

Replacement Algorithms (1) Direct mapping

- No choice
- Each block only maps to one line
- Replace that line

Replacement Algorithms (2) Associative & Set Associative

- Algorithm must be implemented in hardware (speed)
- Least Recently used (LRU)
  - e.g. in 2 way set associative, which of the 2 block is LRU?
    - For each slot, have an extra bit, USE. Set to 1 when accessed, set all others to 0.
  - For more than 2-way set associative, need a time stamp for each slot - expensive
- First in first out (FIFO)
  - Replace block that has been in cache longest
  - Easy to implement as a circular buffer
- Least frequently used
  - Replace block which has had fewest hits
  - Need a counter to sum number of hits
- Random
  - Almost as good as LFU and simple to implement

Write Policy

- Must not overwrite a cache block unless main memory is up to date. I.e. if the "dirty" bit is set, then we need to save that cache slot to memory before overwriting it
- This can cause a BIG problem
  - Multiple CPUs may have individual caches
    - What if a CPU tries to read data from memory? It might be invalid if another processor changed its cache for that location!
    - Called the **cache coherency** problem
  - I/O may address main memory directly too

Write through

- Simplest technique to handle the cache coherency problem - All writes go to main memory as well as cache.
- Multiple CPUs must monitor main memory traffic (snooping) to keep local cache local to its CPU up to date in case another CPU also has a copy of a shared memory location in its cache
- Simple but Lots of traffic
- Slows down writes

- Other solutions: noncachable memory, hardware to maintain coherency

Write Back

- Updates initially made in cache only
- Dirty bit for cache slot is cleared when update occurs
- If block is to be replaced, write to main memory only if dirty bit is set
- Other caches can get out of sync
- If I/O must access invalidated main memory, one solution is for I/O to go through cache
  - Complex circuitry
- Only ~15% of memory references are writes

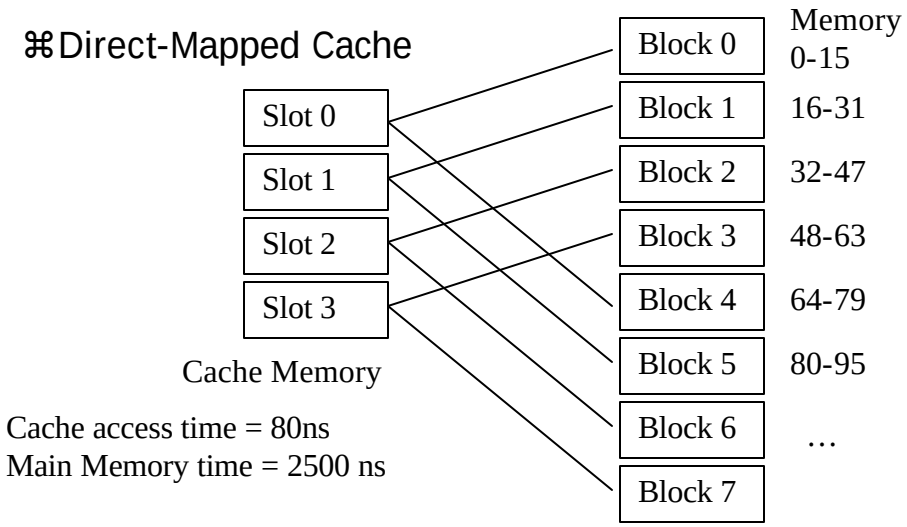
Cache Performance

- Two measures that characterize the performance of a cache are the **hit ratio** and the **effective access time**
  - Hit Ratio = (Num times referenced words are in cache) / (Total number of memory accesses)

$$\text{Eff. Access Time} = \frac{(\#\text{ hits})(\text{TimePerHit}) + (\#\text{ misses})(\text{TimePerMiss})}{(\text{Total number of memory accesses})}$$

Cache Performance Example

- Direct-Mapped Cache



Cache access time = 80ns
Main Memory time = 2500 ns

Cache Performance Example

⌘ Sample program executes from memory location 48-95 once. Then it executes from 15-31 in a loop ten times before exiting.

Event	Location	Time	Comment
1 miss	48	2500ns	Memory block 3 to cache slot 3
15 hits	49-63	80ns×15=1200ns	
1 miss	64	2500ns	Memory block 4 to cache slot 0
15 hits	65-79	80ns×15=1200ns	
1 miss	80	2500ns	Memory block 5 to cache slot 1
15 hits	81-95	80ns×15=1200ns	
1 miss	15	2500ns	Memory block 0 to cache slot 0
1 miss	16	2500ns	Memory block 1 to cache slot 1
15 hits	17-31	80ns×15=1200ns	
9 hits	15	80ns×9=720ns	Last nine iterations of loop
144 hits	16-31	80ns×144=12,240ns	Last nine iterations of loop
Total hits = 213		Total misses = 5	

Cache Performance Example

⌘ Hit Ratio: 213 / 218 = 97.7%

⌘ Effective Access Time: ((213)*(80ns)+(5)(2500ns)) / 218 = 136 ns

⌘ Although the hit ratio is high, the effective access time in this example is 75% longer than the cache access time due to the large amount of time spent during a cache miss

⌘ What sequence of main memory block accesses would result in much worse performance?